iBKS SDK for Android Reference Manual

ABSTRACT

The iBKS SDK for Android is a library which allows communication, managing and interaction with **iBKS Beacons.**

This document contains listings of iBKS SDK function prototypes which cover the five packages integrated in the library:

- Connection
- Scan
- Eddystone Service
- iBeacon Service
- Global Service

AUDIENCE

This document is primarily focused for Android software developers with basic knowledge of beacon configuration

IMPORTANT

The service packages are related to the main services of the iBKS firmware version **EDSTEID V5.2016.06.29.1**

SO Requirements: **Android 5.0** or higher

Revision 0 | July 2016

Index

1. Before you start

This SDK will help you to manage iBKS Beacons with your own Android APP in a few easy steps.

All you need:

- Android Studio
- Android device with 4.3 version or above.
- At least one iBKS Beacon.

2. Let's play

2.1 Create a project

First of all, create a new Android Studio project and add the iBKS SDK to the build.gradle (Module:app) declaring the following dependency:

```
compile 'com.accent_systems.ibks-sdk:ibks-sdk:1.0.0'
```

2.1.1 App permissions

To manage Bluetooth in Android it's necessary to request some permissions at user:

Location permission

If the Android version is 6.0 or higher it's necessary to request location permission. To do this it's necessary to add permission in AndroidManifest.xml:

```
<uses-permission  android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Bluetooth permission

In order to use Bluetooth in Android device, the first thing to do is to check if the device that runs the app has Bluetooth Low Energy (beacon works with this type of Bluetooth) and if it is enabled. To enable Bluetooth it's necessary to add permission in AndroidManifest.xml:

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

Internet permission

In order to use some functions that request access to internet, it's necessary to add permissions on AndroidManifest.xml:

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

2.2 Get started with a sample project

From the iBKS SDK github documentation, you will find the "iBKS SDK Sample Project". This example shows how to do the main tasks on a Bluetooth APP such as:

- Scan devices
- Connect devices
- Read/Write characteristics
- Set/Get Eddystone Slots
- Register EID beacon
- Get EID in Clear
- Set/Get iBeacon Slots
- Set/Get Characteristics of Global service

3. iBKS SDK Functions

Function Listing Format

This is the general format for listing a function.

function_name

A short description of what function **function_name** does.

Prototype

Provides a prototype for the function **function_name**

Parameters

Parameter_1 --- Paramter_1 definition
Parameter_2 --- Paramter_2 definition
Parameter_n --- Paramter_n definition

Returns

Specifies any value or values returned by the function.

Callback

Callback function

Description

Describes the function **function_name**. This section also describes any special characteristics or restrictions that might apply

3.1 Connection Package

The connection package functions allow you to establish connection with Bluetooth devices (Beacons) and access to their services and characteristics.

3.1.1 ASConDevice

This function is the constructor of ASConDevice class, which is the main class of connection package.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java
public ASConDevice(
    Activity activity,
    BluetoothAdapter mBluetoothAdapter,
    ASConDeviceCallback conDeviceCallback)
```

Parameters

activity	--- Activity which calls the constructor
mBluetoothAdapter	--- Local device Bluetooth adapter
conDeviceCallback	--- Activity where are defined the callback functions defined on ASConDevice Interface (ASConDeviceCallback.java)

Returns

None

Description

This function initializes the ASConDevice object in order to use all the connection methods associated to that object.

3.1.2 connectDevice

This method establishes the connection with a Bluetooth device.

Prototype (2)

```
…\ibkslibrary\connections\ASConDevice.java

public static void connectDevice(
    String address)

public static void connectDevice(
    BluetoothDevice device)
```

Parameters

address	--- device MAC address
device	--- BluetoothDevice object

Returns

- **TASK_OK**
- **Errors**

Error	Description
TASK_ERROR	IllegalArgument exception
ERROR_BLE_NOT_SUPPORTED	BLE not supported on Android device
ERROR_BLUETOOTH_NOT_SUPPORTED	Bluetooth not supported on Android device
ERROR_BLUETOOTH_ADAPTER_NULL	Error getting bluetooth adapter
ERROR_BLUETOOTH_NOT_ENABLED	Bluetooth not enabled on Android device

Callback

```
void onChangeStatusConnection(int result, BluetoothGatt blgatt)

void onServicesCharDiscovered(
    int result,
    BluetoothGatt blgatt,
    ArrayList<BluetoothGattService> services,
    ArrayList<BluetoothGattCharacteristic> characteristics)
```

*Look at **ASConDevice Callbacks** for further information*

Description

This method checks Bluetooth Adapter of Android device and connect to the GATT Server hosted by the remote device. If connection successes, it tries to discover all services and characteristics.

3.1.3 disconnectDevice

This method allows disconnecting from Bluetooth device.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java
```

```
public static void disconnectDevice()
```

Parameters

None

Returns

None

Callback

```
void onChangeStatusConnection(int result, BluetoothGatt blgatt)
```

*Look at **ASConDevice Callbacks** for further information*

Description

This method is used to disconnect an established connection, or cancels a connection attempt currently in progress.

3.1.4 readCharacteristic

This method is used in order to read any characteristic of the connected device.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java

public static void readCharacteristic(
    final BluetoothGattCharacteristic characteristic)
```

Parameters

characteristic --- characteristic to read from the remote device

Returns

None

Callback

```
void onReadDeviceValues (
    int result,
    BluetoothGattCharacteristic characteristic,
    String value)
```

*Look at **ASConDevice Callbacks** for further information*

Description

This method is used to read a characteristic value.

Important: Until the callback is received, no write or read can be done.

3.1.5 writeCharacteristic

This method is used in order to read any characteristic of the connected device.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java

public static void writeCharacteristic(
    final BluetoothGattCharacteristic characteristic,
    final byte[] value)
```

Parameters

characteristic	--- characteristic to read from the remote device
value	--- value to be written

Returns

None

Callback

```
void onWriteDeviceChar (
    int result,
    BluetoothGattCharacteristic characteristic)
```

*Look at **ASConDevice Callbacks** for further information*

Description

This method is used to write a value in a characteristic.

Important: Until the callback is received, no write or read can be done.

3.1.6 readRSSI

This method is used to read the RSSI value of the connected device.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java
```

```
public static void readRSSI()
```

Parameters

None

Returns

None

Callback

```
void onReadDeviceValues (
    int result,
    BluetoothGattCharacteristic characteristic,
    String value)
```

*Look at **ASConDevice Callbacks** for further information*

Description

This method is used to read the RSSI value.

Important: Until the callback is received, no write or read can be done.

3.1.7 setCharNotification

This method is used to enable/disable notification for a specific characteristic.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java

public static void setCharNotification(
    final BluetoothGattCharacteristic characteristic,
    boolean enable)
```

Parameters

characteristic	--- characteristic to enable notification
enable	--- true: enable / false: disable

Returns

None

Callback

```
void onReadDeviceValues (
    int result,
    BluetoothGattCharacteristic characteristic,
    String value)
```

*Look at **ASConDevice Callbacks** for further information*

Description

This method is used to read the RSSI value.

Important: Until the callback is received, no write or read can be done.

3.1.8 getServicesArray

This method is used to get Services Array of the connected device.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java
```

```
public static ArrayList<BluetoothGattService>
getServicesArray()
```

Parameters

None

Returns

ArrayList<BluetoothGattService> services --- array list of discovered services

Callback

None

Description

This method is used to get Services Array of the connected device. If there's no device connected, it returns null value.

3.1.9 getCharacteristicsArray

This method is used to get Characteristics Array of the connected device.

Prototype

```
…\ibkslibrary\connections\ASConDevice.java
```

```
public static ArrayList<BluetoothGattCharacteristic>
getCharacteristicsArray()
```

Parameters

None

Returns

ArrayList<BluetoothGattCharacteristic> chars --- array list of discovered characteristics

Callback

None

Description

This method is used to get Characteristics Array of the connected device. If there's no device connected, it returns null value.

3.1.10 ASConDevice Callbacks

In order to get the callbacks, it is necessary to implement the ASConDevice Interface on the class that uses it.

```
public class MainApp implements ASscannerCallback {

    ...

    void onChangeStatusConnection(int result, BluetoothGatt 
blgatt)
    {
        //Do something
    }
    ...

}
```

3.1.10.1 onChangeStatusConnection

This callback is called when the connection status is changed.

Prototype

```
…\ibkslibrary\connections\ASConDeviceCallback.java

void onChangeStatusConnection(
    int result,
    BluetoothGatt blgatt)
```

Parameters

result --- result of the callback

- **GATT_DEV_CONNECTED:** Device is connected successfully
- **GATT_DEV_DISCONNECTED:** Device is disconnected

blgatt --- BluetoothGatt Object used in the connection

3.1.10.2 onServicesCharDiscovered

This callback is called when the services and characteristics are discovered.

Prototype

```
…\ibkslibrary\connections\ASConDeviceCallback.java

void onServicesCharDiscovered (
    int result,
    BluetoothGatt blgatt,
    ArrayList<BluetoothGattService> services,
    ArrayList<BluetoothGattCharacteristic> characteristics)
```

Parameters

result --- result of the callback

- **GATT_SERV_DISCOVERED_OK:** Services and chars discovered successfully
- **GATT_SERV_DISCOVERED_ERROR:** Failed to discover services and chars

blgatt --- BluetoothGatt Object used in the connection
services --- Array List of Services discovered
characteristics --- Array List of Characteristics discovered

3.1.10.3 onReadDeviceValues

This callback is called when a value is read (notification, read characteristic or read RSSI)

Prototype

```
…\ibkslibrary\connections\ASConDeviceCallback.java

void onReadDeviceValues (
    int result,
    BluetoothGattCharacteristic characteristic,
    String value)
```

Parameters

result	--- result of the callback

- **READ_OK:** Characteristic value read OK
- **READ_ERROR:** Error reading characteristic value
- **GATT_NOTIFICATION_RCV:** Notification value received
- **GATT_RSSI_OK: RSSI** value received OK
- **GATT_RSSI_ERROR:** Error reading RSSI value

characteristic	--- characteristic object read
value	--- value read in string format

3.1.10.4 onWriteDeviceChar

This callback is called when a value is written on a characteristic.

Prototype

```
…\ibkslibrary\connections\ASConDeviceCallback.java

void onWriteDeviceChar (
    int result,
    BluetoothGattCharacteristic characteristic)
```

Parameters

result	--- result of the callback

- **WRITE_OK: Characteristic value read OK**
- **WRITE_ERROR: Error reading characteristic value**

characteristic	--- characteristic object written

3.1.11 Example of use ASConDevice

This example shows how to connect to a device and how the Connection callbacks are managed. Also it shows how to read a characteristic.

```java
public class MainApp extends AppCompatActivity implements ASConDeviceCallback
{

BluetoothGattCharacteristic myCharRead;

...
//Start connection to device
BluetoothAdapter mBluetoothAdapter = ASBleScanner.getmBluetoothAdapter();
if(mBluetoothAdapter != null) {
    ASConDevice mcondevice;
    mcondevice =  new ASConDevice(this, mBluetoothAdapter, this);
 //connectDevice will call onChangeStatusConnection and
onServicesCharDiscovered callbacks
    ASConDevice.connectDevice(address);
} else{
    Log.i("MainApp","BLE not enabled/supported!");
}
...

//Callback from ASConDevice
public void onChangeStatusConnection(int result, BluetoothGatt blgatt){
  switch (result){
  case ASUtils.GATT_DEV_CONNECTED:
      Log.i("MainApp", "DEVICE CONNECTED: "+blgatt.getDevice().getName());
  break;
  case ASUtils.GATT_DEV_DISCONNECTED:
      Log.i("MainApp", "DEVICE DISCONNECTED: "+blgatt.getDevice().getName());
  break;
  default:
      Log.i("MainApp ", "ERROR PARSING");
  break;
  }
}

//Callback from ASConDevice
public void onServicesCharDiscovered(int result, BluetoothGatt blgatt,
ArrayList<BluetoothGattService> services,
ArrayList<BluetoothGattCharacteristic> characteristics)
{
  switch (result){
  case ASUtils.GATT_SERV_DISCOVERED_OK:
      int err;
      Log.i("MainApp ", "SERVICES DISCOVERED OK");

      for(int i=0; i<characteristics.size(); i++){
        if(characteristics.get(i).getUuid().toString().contains("00002a28")){
           myCharRead = characteristics.get(i);
        }
      }
      ASConDevice.readCharacteristic(myCharRead);
  break;
  case ASUtils.GATT_SERV_DISCOVERED_ERROR:
      Log.i("MainApp ", "SERVICES DISCOVERED ERROR");
  break;
  default:
      Log.i("MainApp ", "ERROR PARSING");
  break;
  }
}
```

```
//Callback from ASConDevice
public void onReadDeviceValues(int result,
BluetoothGattCharacteristic characteristic, String value){
  switch (result){
        case ASUtils.GATT_READ_SUCCESSFULL:

if(characteristic.getUuid().toString().contains("00002a28")) {
                 value = ASResultParser.StringHexToAscii(value);
              }
              Log.i("DEVICE CONNECTION", "READ VALUE: " + value);
            break;
        case ASUtils.GATT_READ_ERROR:
            Log.i("DEVICE CONNECTION", "READ ERROR");
            break;
        case ASUtils.GATT_NOTIFICATION_RCV:
            Log.i("DEVICE CONNECTION", "READ NOTIFICATION: " +
value);
            break;
        case ASUtils.GATT_RSSI_OK:
            Log.i("DEVICE CONNECTION", "READ RSSI: " + value);
            break;
        case ASUtils.GATT_RSSI_ERROR:
            Log.i("DEVICE CONNECTION", "READ RSSI ERROR");
            break;
        default:
            Log.i("DEVICE CONNECTION", "ERROR PARSING
onReadDeviceValues");
            break;
    }
}

//Callback from ASConDevice
public void onWriteDeviceChar(int result, BluetoothGattCharacteristic
characteristic) {
    switch (result) {
        case ASUtils.GATT_WRITE_SUCCESSFULL:
            Log.i("MainApp", "WRITE SUCCESSFULL
on:"+characteristic.getUuid().toString());

            break;
        case ASUtils.GATT_WRITE_ERROR:
            Log.i("MainApp","WRITE ERROR
on:"+characteristic.getUuid().toString());
            break;
        default:
            Log.i("MainApp","ERROR PARSING");
            break;
    }
}
```

3.2 Scanner Package

The functions of the Scanner package allow you to scan and read Bluetooth packets.

3.2.1 ASBleScanner

This function is the constructor of ASBleScanner class, which is the main class of Scanner package.

Prototype

```
…\ibkslibrary\scanner\ASBleScanner.java

public ASBleScanner(
    Activity activity,
    ASScannerCallback scannerCallback)
```

Parameters

activity	--- Activity which calls the constructor
scannerCallback	--- Activity where are defined the callback functions defined on ASBleScanner Interface (ASScannerCallback.java)

Returns

None

Description

This function initializes the ASBleScanner object in order to use all the scan methods associated to that object.

3.2.2 startScan

This method starts the scan of bluetooth packets.

Prototype

```
…\ibkslibrary\scanner\ASBleScanner.java
```

```
public static void startScan()
```

Parameters

None

Returns

- **TASK_OK**
- **Errors**

Error	Description
TASK_ERROR	IllegalArgument exception
ERROR_BLE_NOT_SUPPORTED	BLE not supported on Android device
ERROR_BLUETOOTH_NOT_SUPPORTED	Bluetooth not supported on Android device
ERROR_BLUETOOTH_ADAPTER_NULL	Error getting bluetooth adapter
ERROR_BLUETOOTH_NOT_ENABLED	Bluetooth not enabled on Android device
ERROR_LOCATION_PERMISSION_NOT_GRANTED	Location permission not granted

Callback

```
void scannedBleDevices(int result)
```

*Look at **ASBleScanner Callbacks** for further information*

Description

This method checks Bluetooth Adapter and Location permissions (only for Android versions higher than 6.0) of Android device and start Bluetooth LE scan.

3.2.3 stopScan

This method stops bluetooth packets scan.

Prototype

```
…\ibkslibrary\scanner\ASBleScanner.java
```

```
public static void stopScan()
```

Parameters

None

Returns

None

Callback

None

Description

This method stops the ongoing Bluetooth LE scan.

3.2.4 setScanMode

This method sets the scan mode.

Prototype

```
…\ibkslibrary\scanner\ASBleScanner.java

public static void setScanMode(
     int ScanMode)
```

Parameters

ScanMode --- Scan Mode:

- **SCAN_MODE_OPPORTUNISTIC**
- **SCAN_MODE_LOW_POWER**
- **SCAN_MODE_BALANCED**
- **SCAN_MODE_LOW_LATENCY** (recommended)

Returns

- **TASK_OK**
- **Errors**

Error	Description
ERROR_BLE_NOT_SUPPORTED	BLE not supported on Android device
ERROR_BLUETOOTH_NOT_SUPPORTED	Bluetooth not supported on Android device
ERROR_BLUETOOTH_ADAPTER_NULL	Error getting bluetooth adapter
ERROR_BLUETOOTH_NOT_ENABLED	Bluetooth not enabled on Android device

Callback

None

Description

This method sets the scan mode.

3.2.5 getmBluetoothAdapter

This method returns the Bluetooth Adapter.

Prototype

```
…\ibkslibrary\scanner\ASBleScanner.java
```

```
public static BluetoothAdapter getmBluetoothAdapter()
```

Parameters

None

Returns

BluetoothAdapter mBluetoothAdapter

Callback

None

Description

This method returns the Bluetooth Adapter commonly used for connect to the remot device.

3.2.6 ASResultParser

The class ASResultParser contains several methods to parse and convert data.

The functions names are intuitive but the method **getDataFromAdvertising** needs a mention:

Prototype

```
…\ibkslibrary\scanner\ASResultParser.java

public static JSONObject getDataFromAdvertising(ScanResult
result)
```

Parameters

result --- ScanResult received on scannedBleDevices callback.

Returns

JSONObject data

The returned value is a JSONObject that contains the parsed data of the advertising. Depending on the frame type, the JSON parameters are ones or the others:

Frame type	JSON parameters
TYPE_IBEACON	FrameType, AdvTxPower, UUID, Major, Minor
TYPE_EDDYSTONE_UID	FrameType, AdvTxPower, Namespace, Instance
TYPE_EDDYSTONE_URL	FrameType, AdvTxPower, Url
TYPE_EDDYSTONE_TLM	FrameType, Version (0), Vbatt, Temp, AdvCount, TimeUp
	FrameType, Version (1), EncryptedTLMData,Salt, IntegrityCheck
TYPE_EDDYSTONE_EID	FrameType, AdvTxPower, EID

Callback

None

Description

This method parses the advertising packet and returns a JSONObject with the parsed data. The data depends on the frame type.

3.2.7 ASBleScanner Callbacks

In order to get the callback it is necessary to implement the ASBleScaner Interface on the class that use it.

```
public class MainApp implements ASScannerCallback {

   ...

   void scannedBleDevices(ScanResult result)
   {
       //Do something
   }
   ...
}
```

3.2.7.1 scannedBleDevices

This callback is called when a bluetooth packet is received.

Prototype

`…\ibkslibrary\scanner\ASScannerCallback.java`

```
void scannedBleDevices(
    ScanResult result)
```

Parameters

result --- is a ScanResult object that contains all the information of the Bluetooth packet received.

3.2.8 Example of use ASBleScanner

This example shows how to start a Scan and how to manage the Scan callback.

```
public class MainApp extends AppCompatActivity implements ASBleScannerCallback
{

...
//Start scan
new ASBleScanner(this, this).setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY);
ASBleScanner.startScan();

...

//Callback from ASBleScanner
public void scannedBleDevices(ScanResult result){

 String advertisingString =
ASResultParser.byteArrayToHex(result.getScanRecord().getBytes());

 Log.i("MainApp", result.getDevice().getAddress()+" - RSSI:
"+result.getRssi()+"\t - "+advertisingString+" -
"+result.getDevice().getName());

    switch (ASResultParser.getAdvertisingType(result)){
        case ASUtils.TYPE_IBEACON:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" -
iBEACON");
            break;
        case ASUtils.TYPE_EDDYSTONE_UID:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" - UID");
            break;
        case ASUtils.TYPE_EDDYSTONE_URL:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" - URL");
            break;
        case ASUtils.TYPE_EDDYSTONE_TLM:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" - TLM");
            break;
        case ASUtils.TYPE_EDDYSTONE_EID:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" - EID");
            break;
        case ASUtils.TYPE_DEVICE_CONNECTABLE:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" -
CONNECTABLE");
            break;
        case ASUtils.TYPE_UNKNOWN:
            Log.i("ADVERTISING TYPE", result.getDevice().getName()+" -
UNKNOWN");
            break;
        default:
            Log.i("ADVERTISING TYPE", "ERROR PARSING");
            break;
    }
}
```

3.3 Eddystone Service

The functions of the Eddystone Service package allow you to access to all characteristics of the service and configure slots easily.

3.3.1 ASEDSTService

This function is the constructor of ASEDSTService class, which is the main class of Eddystone Service package.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java

public ASEDSTService(
    ASConDevice conDevice,
    ASEDSTCallback eidcallback,
    int timeoutcallbk)
```

Parameters

conDevice	---	ASConDevice Object
eidcallback	---	Activity where are defined the callback functions defined on ASEDSTCallback Interface (ASEDSTCallback.java)
timeoutcallbk	---	timeout for return a response on callback

Returns

None

Description

This function initializes the ASEDSTService object in order to manage all the functionalities of Eddystone Service.

3.3.2 ASEDSTSlot

This function is the constructor of ASEDSTSlot class.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTSlot.java

public ASEDSTSlot(
    int ft,
    int adv_int,
    int txpwr,
    int advtxpwr,
    String data)
```

Parameters

ft --- Frame type:

- **FT_EDDYSTONE_UID** (0x00)
- **FT_EDDYSTONE_URL** (0x10)
- **FT_EDDYSTONE_TLM** (0x20)
- **FT_EDDYSTONE_EID** (0x30)
- **FT_ERASE_SLOT**

adv_int	--- Advertising interval in milliseconds
txpwr	--- Tx power (-30,-20,-16,-12,-8,-4,0,4)
advtxpwr	--- Advertised Tx power @0m
data	--- Data to be written on Advertising slot

Frame type	Data	Num chars	Example
UID	Namespace(20)+Instance(12) [HEX]	32	“0102030405060708090a0b0c0d0e0f10”
URL	URL	< 34 *(it depends on URL scheme and encoding used)*	“http://www.google.com”
TLM	null	0	“”
EID	ID(32)+k exponent(2) [HEX]	34	“0102030405060708090a0b0c0d0e0f100a”

Returns

None

Description

This function initializes the ASEDSTSlot object in order to define the configuration of a slot. In order to use setEDSTSlots, first of all, it is necessary to create an array of ASEDSTSlot, and set the parameters for each one, being the first position of the array the slot 0, the second is the slot 1 and so on. If one slot it is not necessary to be configured, set the correspondent array position to *null*.

3.3.3 setClient_ProjectId

This method is used to set the *ProximityBeacon client* and the *ProjectId* registered on Google Cloud Platform. The *client* must have permissions to access to *Proximity Beacon API* and the proper *Credentials* created to access from your APP in order to register EID beacons.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java

public static void setClient_ProjectId(
      ProximityBeacon cli,
      String projId)
```

Parameters

cli	--- ProximityBeacon client registered on Google API Console
projId	--- Project Id get from client account

Returns

None

Callback

None

Description

This method is used to set the required credentials (ProximityBeacon client and ProjectId) to register beacon on Google Cloud Platform.

Important: This method must be called before setEDSTSlots if any of the slots is configured as an Eddystone-EID.

3.3.4 setEDSTSlots

This method is the easiest way to configure Eddystone Slots with any of the different 4 frame types

- Eddystone-UID
- Eddystone-URL
- Eddystone-TLM
- Eddystone-EID

, and configure all the related parameters.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java

public static void setEDSTSlots(
    final ASEDSTSlots slots[])
```

Parameters

slots[ ] --- ASEDSTSlot array that includes all the information to be configured on slots

Returns

None

Callback

```
void onEDSTSlotsWrite(int result)
```

*Look at **ASEDSTService Callbacks** for further information*

Description

This method configures the desired slots of Eddystone Service and registers EID slot if it's required.

Important: Until the callback is received, no write or read can be done. The method setClient ProjectId must be called before this one if any of the slots is configured as an Eddystone-EID.

3.3.5 getEDSTSlots

This method is the easiest way to get the configuration of all Eddystone Service slots.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java
```

```
public static void getEDSTSlots()
```

Parameters

None

Returns

None

Callback

```
void onGetEDSTSlots(int result, ASEDSTSlot[]slots)
```

*Look at **ASEDSTService Callbacks** for further information*

Description

This method get all the parameters configured on Eddystone slots and return them on the callback as ASEDSTSlot array.

Important: Until the callback is received, no write or read can be done.

3.3.6 getEIDInClear

This method is used to get the ID in clear from an EID slot.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java

public static void getEIDInClear(
    final int slot)
```

Parameters

slot --- Slot index where is configured the Eddystone-EID packet

Returns

None

Callback

```
void onGetEIDInClear(int result, String EID, String msg)
```

Look at ***ASEDSTService Callbacks*** *for further information*

Description

This method gets the ID in clear from an EID slot connecting with Google Cloud Platform.

Important: The method setClient_ProjectId must be called before this one in order to get credentials to get access on Google Cloud Platform.

3.3.7 getEIDInClearByTheAir

This method is used to get the ID in clear from the advertised packet by the air.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java

public static void getEIDInClearByTheAir(
    final ScanResult result)
```

Parameters

result --- ScanResult packet received on scannedBleDevices callback.

Returns

None

Callback

```
void onGetEIDInClear(int result, String EID, String msg)
```

*Look at **ASEDSTService Callbacks** for more info*

Description

This method gets the ID in clear from EID packet advertised by the air and connects with Google Cloud Platform in order to get ID in clear.

Important: The method setClient_ProjectId must be called before this one in order to get credentials to get access on Google Cloud Platform. It's recommended to call getEIDInClearByTheAir and wait for callback to call again this function.

3.3.8 get*EddystoneService_Characteristics*

There is a method to read each characteristic of Eddystone Service.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java
```

```
public static void getCharacteristicName()
```

Methods

Method	Characteristic read
getBroadcastCapabilities()	a3c87501-8ed3-4bdf-8a39-a01bebede295
getActiveSlot()	a3c87502-8ed3-4bdf-8a39-a01bebede295
getAdvertisingInterval()	a3c87503-8ed3-4bdf-8a39-a01bebede295
getRadioTxPower()	a3c87504-8ed3-4bdf-8a39-a01bebede295
getAdvTxPower()	a3c87505-8ed3-4bdf-8a39-a01bebede295
getLockState()	a3c87506-8ed3-4bdf-8a39-a01bebede295
getUnlockChallenge()	a3c87507-8ed3-4bdf-8a39-a01bebede295
getPublicECDHKey()	a3c87508-8ed3-4bdf-8a39-a01bebede295
getEIDIdentityKey()	a3c87509-8ed3-4bdf-8a39-a01bebede295
getRWAdvSlot()	a3c8750a-8ed3-4bdf-8a39-a01bebede295
getRemainConnectable()	a3c8750c-8ed3-4bdf-8a39-a01bebede295

Parameters

None

Returns

None

Callback

```
void onReadEDSTCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic,
    byte[] readval)
```

*Look at **ASEDSTService Callbacks** for further information*

Description

These methods are used to read the characteristics of Eddystone Service.

Important: Until the callback is received, no write or read can be done.

3.3.9 *setEddystoneService_Characteristics*

This is a method to set each characteristic of Eddystone Service.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTService.java

public static void setCharacteristicName(parameters)
```

Methods & parameters

Method	Characteristic read	Parameter description
setActiveSlot(int **slot**)	a3c87502	Slot to active
setAdvertisingInterval(int **ms**)	a3c87503	Advertising interval in milliseconds
setRadioTxPower(int **txpower**)	a3c87504	Tx power in dBm
setAdvTxPower(int **advtxpower**)	a3c87505	Advertised Tx power in dBm
setLockState(int **nbytes**, int **lockbyte**, String **encryptedkey**)	a3c87506	More info on **Parameters**
setUnlockKey(String **unlock_key**)	a3c87507	More info on **Parameters**
setRWAdvSlot(int **frametype**, byte[] **data**)	a3c8750a	More info on **Parameters**
setResetFactory()	a3c8750b	Makes reset factory if Lock State is '0x01'
getRemainConnectable(Boolean **state**)	a3c8750c	state=**true** → conectable state=**false** → nonconnectable

Parameters

<u>**setLockState**</u>

nbytes	--- number of bytes to be written (1 or 17 bytes)
lockbyte	--- Lock byte (0x00 or 0x02)
encryptedkey	--- encryptedkey = encryptAES128ECB(key=actual_lock_code[16], text=new_lock_code[16])). This string it's in hexadecimal format with a length of 32 characters (16 bytes).

Write 1 byte or 17 bytes to transition to a new lock state:

- **(nbytes=1, lockbyte=0x00):** A single byte of 0x00 written to this characteristic will transition the interface to the LOCKED state without changing the current security key value.

- **(nbytes=17, lockbyte=0x00, encryptedkey):** A 16 byte encrypted key value written to this characteristic will transition the interface to the LOCKED state and update the security key to the unencrypted value of encryptedkey[16]. The client shall AES128ECB encrypt the new code with the existing lock code:

encryptedkey[16] = encrypt(key=actual_lock_code[16], text=new_lock_code[16])

- **(nbytes=1, lockbyte=0x02):** A single byte of 0x02 written to this characteristic will disable the automatic relocking capability of the interface.

setUnlockKey

unlock_key --- This string it's in hexadecimal format with a length of 32 characters (16 bytes). To create the unlock_key, it first reads the randomly generated 16 byte challenge (**getUnlockChallenge**()) and generates it using AES128ECB.

unlock_key[16] = encrypt(key=beacon_lock_code[16], text=challenge[16])

If the introduced unlock_key is correct, Lock State sets to 0x01.

setRWAdvSlot

frametype --- Frame type to be set on the active slot.

- **FT_EDDYSTONE_UID** (0x00)
- **FT_EDDYSTONE_URL** (0x10)
- **FT_EDDYSTONE_TLM** (0x20)
- **FT_EDDYSTONE_EID** (0x30)
- **FT_ERASE_SLOT**

data[] --- Data to be written on characteristic **a3c8750a**, the format depends on the frame type:

- **FT_EDDYSTONE_UID**
  data = Namespace+Instance in hexadecimal format with a length of 32 characters. (i.e.: "0102030405060708090a0b0c0d0e0f10")

- **FT_EDDYSTONE_URL**
  data = url (i.e.: "http://www.google.com")

- **FT_EDDYSTONE_TLM**
  data = none

- **FT_EDDYSTONE_EID**
  data = serviceECDHPublicKey+rotationExponent in hexadecimal format with a length of 66 characters (32 bytes + 1 byte) (i.e.: “0a1b56788ce5466aff0157888def225789af0b12e163458700001f534eda41120a”, where K=10 (last 0x0A byte))

- **FT_ERASE_SLOT**
  data = none

Callback

```
void onWriteEDSTCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic)
```

*Look at **ASEDSTService Callbacks** for further information*

Description

These methods are used to set the characteristics of Eddystone Service.

Important: Until the callback is received, no write or read can be done.

3.3.10 ASEDSTService Callbacks

In order to get the callbacks it is necessary to implement the ASEDSTService Interface on the class that use it.

```
public class MainApp implements ASEDSTCallback {

    ...

    void onEDSTSlotsWrite(int result)
    {
        //Do something
    }
    ...
}
```

3.3.10.1 onEDSTSlotsWrite

This callback is called when method setEDSTSlots is finished.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTCallback.java

void onEDSTSlotsWrite(
     int result)
```

Parameters

result --- result of the callback

- **WRITE_OK:** Slots wrote successfully
- **WRITE_ERROR:** Error in write process
- **WRITE_TIMEOUT:** Timeout while writing
- **FRAMETYPE_NOT_VALID:** Frame type specified in slot is not valid
- **SLOT_ID_NOT_VALID:** Slot index is not valid
- **ADVINT_VAL_ERR:** Advertising Interval Value incorrect
- **TX_POWER_NOT_SUPP:** Tx power not supported
- **RWADVSLOT_DATA_ERR:** Error in slot data length
- **WRITE_ERR_EID:** Error while registering EID
- **READ_ERROR:** Error while reading broadcast capabilities or EID data.

3.3.10.2 onGetEDSTSlots

This callback is called when method getEDSTSlots is finished.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTCallback.java
```

```
void onGetEDSTSlots(
    int result,
    ASEDSTSlot[] slots)
```

Parameters

result --- result of the callback

- **READ_OK:** Slots read successfully
- **READ_ERROR:** Error reading characteristics
- **READ_TIMEOUT:** Timeout while reading
- **WRITE_ERROR:** Error while writing active slot
- **WRITE_TIMEOUT:** Timeout while writing active slot

slots --- Array of ASEDSTSlot with all the read data. If there's an EID slot configured, the data put on the array is the advertised data (1 byte exponent, 4byte clock value, 8byte EID) and the method getEIDInClear must be called to get ID in clear. If there's a eTLM/TLM slot configured, the data put on the array is the advertised data:

- **TLM:** 1byte version, 2 bytes battery voltage, 2 bytes temperature, 4 bytes PDU count, 4 bytes time since power-on
- **eTLM:** 1 byte version, 12 bytes Encrypted TLM data, 2 bytes Salt, 2 bytes Integrity Check

3.3.10.3 onReadEDSTCharacteristic

This callback is called when any of the [get*EddystoneService_Characteristics*] methods is finished.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTCallback.java
```

```
void onReadEDSTCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic,
    byte[] readval)
```

Parameters

result --- result of the callback

- **READ_OK:** Slots read successfully
- **READ_ERROR:** Error reading characteristics
- **READ_TIMEOUT:** Timeout while reading

characteristic --- characteristic read
readval --- data read in a byte array

Method	data read
getBroadcastCapabilities ()	byte array { version_byte (0x00), max_supported_total_slots (0x04), max_supported_eid_slots (0x01), capabilities_bit_field (0x03), supported_frame_types_bit_field[0] (0x00), supported_frame_types_bit_field[1] (0x0F), supported_radio_tx_power[0] … supported_radio_tx_power[N1] (E2:EC:F0:F4:F8:FC:00:04) } [14 bytes HEX]
getActiveSlot()	Reads the active slot number, 0-indexed
getAdvertisingInterval()	Reads the advertising interval in ms for the active slot [2bytes HEX]
getRadioTxPower()	Reads the Tx power in dBm for the active slot [1byte HEX]
getAdvTxPower()	Reads the advertised Tx power in dBm for the active slot [1 byte HEX]
getLockState()	Reads the Lock State:

	0x00: LOCKED 0x01: UNLOCKED 0x02: UNLOCKED AND AUTOMATIC RELOCK DISABLED
getUnlockChallenge()	Reads a 128bit challenge token [16 bytes HEX]
getPublicECDHKey()	Reads the beacon's 256bit public ECDH key. [32 bytes HEX]
getEIDIdentityKey()	Reads the identity key for the active slot. If the slot isn't configured as EID, returns an error. To prevent this data being broadcast in the clear, its AES128 encrypted with the lock code for the beacon. [16 bytes HEX]
getRWAdvSlot()	Reads the data set in the active slot to be broadcast. The interpretation of the read data is characterized by the frame_type byte, which maps to the Eddystone frame type bytes of 0x00 (UID), 0x10 (URL), 0x20 (TLM), 0x30 (EID). In the case of a UID, URL or TLM frame, the length and data are those of the broadcast data: - **UID**: FT(0x00)+AdvTXPwr(1b)+Namespace(10b)+ Instance(6b) [18 bytes HEX] - **URL**: FT(0x01)+AdvTxPwr(1b)+URL Scheme(1b)+Encoded URL(1-17b) [<=20 bytes HEX] - **TLM**: FT(0x02)+Version(0x00)+Batt Voltage(2b)+Temp(2b)+PDU count(4b)+Time power-up(4b) [14 bytes HEX] - **eTLM**: FT(0x02)+Version(0x01)+Encrypted TLM data(12b)+Salt(2b)+Integrity Check(2b) [18 bytes HEX] If the slot is configured to advertise EID frames, the length is 14: 1 byte frame type, 1 byte exponent, 4byte clock value, 8byte EID. These are the parameters required for registration, along with the beacon's public key, which is exposed through a separate characteristic, and the resolver's public key, which the provisioning / registering beacon knows.

getRemainConnectable()	Returning a nonzero value indicates that the beacon is capable of becoming nonconnectable. Returning a zero value indicates that the beacon is limited to running in an always connectable state.

3.3.10.4 onWriteEDSTCharacteristic

This callback is called when method *setEddystoneService_Characteristics* is finished.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTCallback.java

void onWriteEDSTCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic)
```

Parameters

result --- result of the callback (depends on the method used)

- **WRITE_OK:** Slots wrote successfully
- **WRITE_ERROR:** Error in write process
- **WRITE_TIMEOUT:** Timeout while writing
- **FRAMETYPE_NOT_VALID:** Frame type specified in slot is not valid
- **SLOT_ID_NOT_VALID:** Slot index is not valid
- **ADVINT_VAL_ERR:** Advertising Interval Value incorrect
- **TX_POWER_NOT_SUPP:** Tx power not supported
- **RWADVSLOT_DATA_ERR:** Error in slot data length
- **WRITE_ERR_EID:** Error while registering EID
- **READ_ERROR:** Error while reading broadcast capabilities or EID data.

3.3.10.5 onGetEIDInClear

This callback is called when method getEIDInClear is finished.

Prototype

```
…\ibkslibrary\EDSTService\ASEDSTCallback.java

void onGetEIDInClear(
    int result,
    String EID,
    String msg)
```

Parameters

result --- result of the callback (depends on the method used)

- **READ_OK:** EID read OK
- **READ_ERROR:** Error in read process
- **READ_TIMEOUT:** Timeout while reading a characteristic
- **CREDENTIALS_ERROR_EID:** Error in client or projectId info.

EID --- EID in clear

msg --- message for specifying the source of the error or give information of the registration status in case of READ_OK.

3.3.11 Example of use ASEDSTService

This example shows how to configure all the slots of Eddystone Service. One of them is configured as EID, so it's necessary to define the client and project Id where the beacon will be registered. The method "setEDSTSlots", do the registration of EID automatically if detects an EID slot.

```
public class MainApp extends AppCompatActivity implements
ASConDeviceCallback, ASEDSTCallback {

public static ProximityBeacon client;

...

//Start connection to device
BluetoothAdapter mBluetoothAdapter = ASBleScanner.getmBluetoothAdapter();
if(mBluetoothAdapter != null) {
    ASConDevice mcondevice;
    mcondevice =  new ASConDevice(this, mBluetoothAdapter, this);
    new ASEDSTService(mcondevice,this,10);

 //connectDevice will call onChangeStatusConnection and
onServicesCharDiscovered callbacks
    ASConDevice.connectDevice(address);
} else{
    Log.i("MainApp","BLE not enabled/supported!");
}
//get client and project Id if it's necessary

...

//Callback from ASConDevice
public void onServicesCharDiscovered(int result, BluetoothGatt blgatt,
ArrayList<BluetoothGattService> services,
ArrayList<BluetoothGattCharacteristic> characteristics)
{
  switch (result){
  case ASUtils.GATT_SERV_DISCOVERED_OK:
    int err;
    Log.i("MainApp ", "SERVICES DISCOVERED OK");

    ASEDSTSlot[] slots = new ASEDSTSlot[4];
    ASEDSTService.setClient_ProjectId(client,getPrefs.getString("projectId",
null));

    slots[0] = new ASEDSTSlot(ASEDSTDefs.FT_EDDYSTONE_UID,800,-4,-35,
"0102030405060708090a0b0c0d0e0f11");
    slots[1] = new ASEDSTSlot(ASEDSTDefs.FT_EDDYSTONE_EID,950,-4,-35,
"1112131415161718191a1b1c1d1e1f200a");
    slots[2] = new ASEDSTSlot(ASEDSTDefs.FT_EDDYSTONE_URL,650,0,-21,
"http://goo.gl/yb6Mgt");
    slots[3] = new ASEDSTSlot(ASEDSTDefs.FT_EDDYSTONE_TLM,9000,4,-17,null);

    ASEDSTService.setEDSTSlots(slots);


  break;
  case ASUtils.GATT_SERV_DISCOVERED_ERROR:
      Log.i("MainApp ", "SERVICES DISCOVERED ERROR");
  break;
  default:
      Log.i("MainApp ", "ERROR PARSING");
  break;
  }
}
```

```
//Callback from ASEDSTService
public void onEDSTSlotsWrite(int result)
{
    if(result == ASUtils.WRITE_OK) {
        Log.i("Main", "Slots EDST write OK!");

        //Read Eddystone slots
        ASEDSTService.getEDSTSlots();

     }
    else
        Log.i("Main","Error (" + Integer.toString(result) + ") writing EDST
slots!");
}


//Callback from ASEDSTService
public void onGetEDSTSlots(int result, ASEDSTSlot[] slots){
    if(result == ASUtils.READ_OK)
    {
        for(int i=0;i<slots.length;i++) {
            if(slots[i].frame_type == ASEDSTDefs.FT_EDDYSTONE_EID)
//if there is an Eddystone-EID configured, read the ID in clear
                ASEDSTService.getEIDInClear(i);
        }
    }
    else
        Log.i("Main","Error (" + Integer.toString(result) + ") reading EDST
slots!");
}

//Callback from ASEDSTService
public void onGetEIDInClear(int result, String EID, String msg){
    if(result == ASUtils.READ_OK) {
        Log.i("MainApp", "EID read OK = "+ EID);
    }
    else
        Log.i("MainApp","Error reading EID (" + Integer.toString(result) +
"): "+ msg);

}
```

3.4 iBeacon Service

The functions of the iBeacon Service package allow you to access to all characteristics of the service and configure slots easily.

3.4.1 ASiBeaconService

This function is the constructor of ASiBeaconService class, which is the main class of iBeacon Service package.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconService.java

public ASiBeaconService(
    ASConDevice conDevice,
    ASiBeaconCallback ibeaconcallback,
    int timeoutcallbk)
```

Parameters

conDevice	---	ASConDevice Object
ibeaconcallback	---	Activity where are defined the callback functions defined on ASiBeaconCallback Interface (ASiBeaconCallback.java)
timeoutcallbk	---	timeout for return a response on callback

Returns

None

Description

This function initializes the ASiBeaconService object in order to manage all the functionalities of iBeacon Service.

3.4.2 ASiBeaconSlot

This function is the constructor of ASiBeaconSlot class.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconSlot.java

public ASiBeaconSlot(
    boolean clearslot,
    int adv_int,
    int txpwr,
    int advtxpwr,
    String uuid,
    String major,
    String minor,
    Boolean eb)
```

Parameters

clearslot	--- true: clear slot / false: set slot
adv_int	--- Advertising interval in milliseconds
txpwr	--- Tx power (-30,-20,-16,-12,-8,-4,0,4)
advtxpwr	--- Advertised Tx power @1m
UUID	--- UUID [16 bytes HEX]
major	--- Major [2 bytes HEX]
minor	--- Minor [2 bytes HEX]
ExtraByte	--- true: active / false: not active. Extra byte adds a byte to the end of the frame with the value of remaining battery voltage in %.

Returns

None

Description

This function initializes the ASiBeaconSlot object in order to define the configuration of a slot. In order to use setiBeaconSlots, first it is necessary to create an array of ASiBeaconSlot, and set the parameters for each one, being the first position of the array the slot 0 and the second, the slot 1 (there's only 2 slots for ibeacon). If one slot it is not necessary to be configured, set the correspondent array position to *null*.

3.4.3 setiBeaconSlots

This method is the easiest way to configure iBeacon Slots

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconService.java

public static void setiBeaconSlots(
    final ASiBeaconSlots slots[])
```

Parameters

slots[] --- ASiBeaconSlot array that includes all the information to be configured on slots

Returns

None

Callback

```
void oniBeaconSlotsWrite(int result)
```

*Look at **ASiBeaconService Callbacks** for further information*

Description

This method configures the desired slots of iBeacon

Important: Until the callback is received, no write or read can be done.

3.4.4 getiBeaconSlots

This method is the easiest way to get the configuration of all iBeacon Service slots.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconService.java
```

```
public static void getiBeaconSlots()
```

Parameters

None

Returns

None

Callback

```
void onGetiBeaconSlots(int result, ASiBeaconSlot[]slots)
```

*Look at **ASiBeaconService Callbacks** for further information*

Description

This method get all the parameters configured on iBeacon slots and return them on the callback as ASiBeaconSlot array.

Important: Until the callback is received, no write or read can be done.

3.4.5 get*iBeaconService_Characteristics*

There is a method to read each characteristic of iBeacon Service.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconService.java
```

```
public static void getCharacteristicName()
```

Methods

Method	Characteristic read
getActiveSlot()	0000fa01-0000-1000-8000-00805f9b34fb
getAdvertisingInterval()	0000fa02-0000-1000-8000-00805f9b34fb
getRadioTxPower()	0000fa03-0000-1000-8000-00805f9b34fb
getAdvTxPower()	0000fa04-0000-1000-8000-00805f9b34fb
getUUIDMajorMinor()	0000fa05-0000-1000-8000-00805f9b34fb
getExtraByte()	0000fa06-0000-1000-8000-00805f9b34fb

Parameters

None

Returns

None

Callback

```
void onReadiBeaconCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic,
    byte[] readval)
```

*Look at **ASiBeaconService Callbacks** for further information*

Description

These methods are used to read the characteristics of iBeacon Service.

Important: Until the callback is received, no write or read can be done.

3.4.6 set*iBeaconService_Characteristics*

There is a method to set each characteristic of iBeacon Service.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconService.java
```

```
public static void setCharacteristicName(parameters)
```

Methods & parameters

Method	Characteristic read	Parameter description
setActiveSlot(int **slot)**	0000fa01	Slot to active
setAdvertisingInterval(int **ms)**	0000fa02	Advertising interval in milliseconds
setRadioTxPower(int **txpower)**	0000fa03	Tx power in dBm
setAdvTxPower(int **advtxpower)**	0000fa04	Advertised Tx power in dBm
setUUIDMajorMinor(boolean **clearslot**, String **UUID**, String **Major**, String **Minor**)	0000fa05	**clearslot** -- **true**: clear slot / **false**: set slot **UUID** -- UUID [16 bytes HEX] **Major** -- Major [2 bytes HEX] **Minor** -- Minor [2 bytes HEX]
setExtraByte(boolean **eb**)	0000fa06	**true**: active / **false**: not active

Callback

```
void onWriteiBeaconCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic)
```

*Look at **ASiBeaconService Callbacks** for further information*

Description

These methods are used to set the characteristics of iBeacon Service.

Important: Until the callback is received, no write or read can be done.

3.4.7 ASiBeaconService Callbacks

In order to get the callbacks it is necessary to implement the ASiBeaconService Interface on the class that uses it.

```
public class MainApp implements ASiBeaconCallback {

    ...

    void oniBeaconSlotsWrite(int result)
    {
        //Do something
    }
    ...
}
```

3.4.7.1 oniBeaconSlotsWrite

This callback is called when method setiBeaconSlots is finished.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconCallback.java

void oniBeaconSlotsWrite(
    int result)
```

Parameters

result --- result of the callback

- **WRITE_OK:** Slots wrote successfully
- **WRITE_ERROR:** Error in write process
- **WRITE_TIMEOUT:** Timeout while writing
- **FRAMETYPE_NOT_VALID:** Frame type specified in slot is not valid
- **SLOT_ID_NOT_VALID:** Slot index is not valid
- **ADVINT_VAL_ERR:** Advertising Interval Value incorrect
- **TX_POWER_NOT_SUPP:** Tx power not supported
- **UUIMAJORMINOR_LENGTH_ERR:** Error in uuid-major-minor length
- **READ_ERROR:** Error while reading broadcast capabilities

3.4.7.2 onGetiBeaconSlots

This callback is called when method getiBeaconSlots is finished.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconCallback.java
```

```
void onGetiBeaconSlots(
    int result,
    ASiBeaconSlot[] slots)
```

Parameters

result --- result of the callback

- **READ_OK:** Slots read successfully
- **READ_ERROR:** Error reading characteristics
- **READ_TIMEOUT:** Timeout while reading
- **WRITE_ERROR:** Error while writing active slot
- **WRITE_TIMEOUT:** Timeout while writing active slot

slots --- Array of ASiBeaconSlot with all the read data

3.4.7.3 onReadiBeaconCharacteristic

This callback is called when any of the getiBeaconService_Characteristics methods is finished.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconCallback.java


void onReadiBeaconCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic,
    byte[] readval)
```

Parameters

result --- result of the callback

- **READ_OK:** Slots read successfully
- **READ_ERROR:** Error reading characteristics
- **READ_TIMEOUT:** Timeout while reading

characteristic --- characteristic read
readval --- data read in a byte array

Method	data read
getActiveSlot()	Reads the active slot number, 0-indexed
getAdvertisingInterval()	Reads the advertising interval in ms for the active slot [2bytes HEX]
getRadioTxPower()	Reads the Tx power in dBm for the active slot [1byte HEX]
getAdvTxPower()	Reads the advertised Tx power in dBm for the active slot [1 bytes HEX]
getUUIDMajorMinor()	Reads the UUID(16b)+Major(2b)+Minor(2b) [20 bytes HEX]
getExtraByte()	Returning a ‘01’ value indicates that the extra byte is active. Returning a ‘00’ value indicates that is not active.

3.4.7.4 onWriteiBeaconCharacteristic

This callback is called when method setiBeaconService_Characteristics is finished.

Prototype

```
…\ibkslibrary\iBeaconService\ASiBeaconCallback.java

void onWriteiBeaconCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic)
```

Parameters

result --- result of the callback (depends on the method used)

- **WRITE_OK:** Slots wrote successfully
- **WRITE_ERROR:** Error in write process
- **WRITE_TIMEOUT:** Timeout while writing
- **FRAMETYPE_NOT_VALID:** Frame type specified in slot is not valid
- **SLOT_ID_NOT_VALID:** Slot index is not valid
- **ADVINT_VAL_ERR:** Advertising Interval Value incorrect
- **TX_POWER_NOT_SUPP:** Tx power not supported
- **UUIMAJORMINOR_LENGTH_ERR:** Error in uuid-major-minor length
- **READ_ERROR:** Error while reading broadcast capabilities

3.4.8 Example of use ASiBeaconService

This example shows how to configure the iBeacon slots in iBeacon Service.

```
public class MainApp extends AppCompatActivity implements
ASConDeviceCallback, ASiBeaconCallback {

...

//Start connection to device
BluetoothAdapter mBluetoothAdapter = ASBleScanner.getmBluetoothAdapter();
if(mBluetoothAdapter != null) {
    ASConDevice mcondevice;
    mcondevice =  new ASConDevice(this, mBluetoothAdapter, this);
    new ASEDSTService(mcondevice,this,10);

 //connectDevice will call onChangeStatusConnection and
onServicesCharDiscovered callbacks
    ASConDevice.connectDevice(address);
} else{
    Log.i("MainApp","BLE not enabled/supported!");
}

...

//Callback from ASConDevice
public void onServicesCharDiscovered(int result, BluetoothGatt blgatt,
ArrayList<BluetoothGattService> services,
ArrayList<BluetoothGattCharacteristic> characteristics)
{
  switch (result){
  case ASUtils.GATT_SERV_DISCOVERED_OK:
    int err;
    Log.i("MainApp ", "SERVICES DISCOVERED OK");

    ASiBeaconSlot[] slotsib=new
ASiBeaconSlot[ASiBeaconDefs.MAX_iBEACON_SLOTS];
    slotsib[1] = new ASiBeaconSlot(false,400,0,-58,
"01010101010101010101010101010101","0002","0002",false);
    slotsib[0] = new ASiBeaconSlot(true,0,0,0,"","","",true);
    ASiBeaconService.setiBeaconSlots(slotsib);

  break;
  case ASUtils.GATT_SERV_DISCOVERED_ERROR:
      Log.i("MainApp ", "SERVICES DISCOVERED ERROR");
  break;
  default:
      Log.i("MainApp ", "ERROR PARSING");
  break;
  }
}

//Callback from ASiBeaconService
public void oniBeaconSlotsWrite(int result)
{
    if(result == ASUtils.WRITE_OK) {
        Log.i("Main", "Slots iBeacon write OK!");

     }
    else
        Log.i("Main","Error (" + Integer.toString(result) + ") writing EDST
slots!");
}
```

3.5 Global Service

The functions of the Global Service package allow you to access to all characteristics of the service.

3.5.1 ASGlobalService

This function is the constructor of ASGlobalService class, which is the main class of Global Service package.

Prototype

```
…\ibkslibrary\GlobalService\ASGlobalService.java

public ASGlobalService(
    ASConDevice conDevice,
    ASGlobalCallback globalcallback,
    int timeoutcallbk)
```

Parameters

conDevice	--- ASConDevice Object
globalcallback	--- Activity where are defined the callback functions defined on ASGlobalCallback Interface (ASGlobalCallback.java)
timeoutcallbk	--- timeout for return a response on callback

Returns

None

Description

This function initializes the ASGlobalService object in order to manage all the functionalities of Global Service.

3.5.2 get*GlobalService_Characteristics*

There is a method to read each characteristic of Global Service.

Prototype

```
…\ibkslibrary\GlobalService\ASGlobalService.java
```

```
public static void getCharacteristicName()
```

Methods

Method	Characteristic read
getDeviceName()	0000ff01-0000-1000-8000-00805f9b34fb
getGATTVersion()	0000fa02-0000-1000-8000-00805f9b34fb
getConnPeriod()	0000fa03-0000-1000-8000-00805f9b34fb
getConnWindow()	0000fa04-0000-1000-8000-00805f9b34fb
getOnHour()	0000fa07-0000-1000-8000-00805f9b34fb
getOffHour()	0000fa08-0000-1000-8000-00805f9b34fb

Parameters

None

Returns

None

Callback

```
void onReadGlobalCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic,
    byte[] readval)
```

*Look at **ASGlobalService Callbacks** for further information*

Description

These methods are used to read the characteristics of Global Service.

Important: Until the callback is received, no write or read can be done.

3.5.3 set*GlobalService_Characteristics*

There is a method to set each characteristic of Global Service.

Prototype

```
…\ibkslibrary\GlobalService\ASGlobalService.java
```

```
public static void setCharacteristicName(parameters)
```

Methods & parameters

Method	Characteristic read	Parameter description
setDeviceName(String **devname)**	0000ff01	Device Name (<=20 characters)
setConnPeriod(int **connperiod)**	0000ff03	Connectable period in seconds
setWindowPeriod(int **winperiod)**	0000ff04	Connectable window in seconds
setFirmwareUpdate**()**	0000ff05	Calling this method a reset factory is done (only if Lock State is 0x01)
setONOFFAdvertising(int **hourON**, int **hourOFF**)	0000fa06-07-08	**hourON** -- hour (in resolution of hours) that the advertising should start **hourOFF** -- hour (in resolution of hours) that the advertising should stop The hours range is between 0-23h and the difference between ON and OFF must be lower than 17hours.

Callback

```
void onWriteGlobalCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic)
```

*Look at **ASiBeaconService Callbacks** for further information*

Description

These methods are used to set the characteristics of Global Service.
Important: Until the callback is received, no write or read can be done.

3.5.4 ASGlobalService Callbacks

In order to get the callbacks it is necessary to implement the ASGlobalService Interface on the class that use it.

```
public class MainApp implements ASGlobalCallback {

    ...

    void onReadGlobalCharacteristic(int result)
    {
        //Do something
    }
    ...
}
```

3.5.4.1 onReadGlobalCharacteristic

This callback is called when any of getGlobalService_Characteristics methods is finished.

Prototype

```
…\ibkslibrary\GlobalServices\ASGlobalCallback.java


void onReadGlobalCharacteristic(
     int result,
     BluetoothGattCharacteristic characteristic,
     byte[] readval)
```

Parameters

result --- result of the callback

- **READ_OK: Slots read successfully**
- **READ_ERROR:** Error reading characteristics
- **READ_TIMEOUT:** Timeout while reading

characteristic --- characteristic read
readval --- data read in a byte array

Method	data read
getDeviceName()	Reads the Device Name [HEX]
getGATTVersion()	Reads the GATT Version [HEX]
getConnPeriod()	Reads the Connectable Period [2 bytes HEX]
getConnWindow()	Reads the Connectable Window [2 bytes HEX]
getOnHour()	Reads the On Hour [1 byte HEX]. If both On and Off are set to zero value, the Advertising ON/OFF is deactivated.
getOffHour()	Reads the Off Hour [1 byte HEX]. If both On and Off are set to zero value, the Advertising ON/OFF is deactivated.

3.5.4.2 onWriteiBeaconCharacteristic

This callback is called when method setGlobalService_Characteristics is finished.

Prototype

```
…\ibkslibrary\GlobalService\ASGlobalCallback.java

void onWriteGlobalCharacteristic(
    int result,
    BluetoothGattCharacteristic characteristic)
```

Parameters

result --- result of the callback (depends on the method used)

- **WRITE_OK:** Slots wrote successfully
- **WRITE_ERROR:** Error in write process
- **WRITE_TIMEOUT:** Timeout while writing
- **CONN_PER_WIN_VAL_ERR:** Error in value of Connectable period or window
- **DEV_NAME_LENGTH_ERR:** Error in device name length

3.5.5 Example of use ASGlobalService

This example shows how to set and get some characteristics and how to manage Global Service Callbacks.

```
public class MainApp extends AppCompatActivity implements ASConDeviceCallback,
ASGlobalCallback {

...

//Start connection to device
BluetoothAdapter mBluetoothAdapter = ASBleScanner.getmBluetoothAdapter();
if(mBluetoothAdapter != null) {
    ASConDevice mcondevice;
    mcondevice =  new ASConDevice(this, mBluetoothAdapter, this);
    new ASEDSTService(mcondevice,this,10);

 //connectDevice will call onChangeStatusConnection and onServicesCharDiscovered
callbacks
    ASConDevice.connectDevice(address);
} else{
    Log.i("MainApp","BLE not enabled/supported!");
}

...

//Callback from ASConDevice
public void onServicesCharDiscovered(int result, BluetoothGatt blgatt,
ArrayList<BluetoothGattService> services, ArrayList<BluetoothGattCharacteristic>
characteristics)
{
  switch (result){
  case ASUtils.GATT_SERV_DISCOVERED_OK:
    int err;
    Log.i("MainApp ", "SERVICES DISCOVERED OK");

    ASGlobalService.setONOFFAdvertising(9,22);


  break;
  case ASUtils.GATT_SERV_DISCOVERED_ERROR:
      Log.i("MainApp ", "SERVICES DISCOVERED ERROR");
  break;
  default:
      Log.i("MainApp ", "ERROR PARSING");
  break;
  }
}

//Callback from ASGlobalService
public void onWriteGlobalCharacteristic(int result, BluetoothGattCharacteristic
characteristic)
{
    if(result == ASUtils.WRITE_OK) {
        //Read device name
        ASGlobalService.getDeviceName();
    }
    else
        Log.i("MainApp","Error (" + Integer.toString(result) + ") writing ");
}
//Callback from ASGlobalService
public void onReadGlobalCharacteristic(int result, BluetoothGattCharacteristic
characteristic, byte[] readval)
{
    if(result == ASUtils.READ_OK) {
        if(characteristic.getUuid().toString().contains(ASGlobalDefs.DEVICE_NAME)){
           Log.i("MainApp","Device Name is "+ASResultParser.StringHexToAscii
(ASResultParser.byteArrayToHex(readval)));
        }
    }
    else
        Log.i("MainApp","Error (" + Integer.toString(result) + ") reading ");
}
```

Revision History

The following revision history table summarizes changes contained in this document.

Revision Number	Revision Date	Description of Changes
Rev 0	07/2016	Initial Release

Bluetooth® SMART iBeacon Eddystone

accent systems

www.accent-systems.com

Accent Systems

Office
Avda. Francesc Macia 46-50, 7th Floor
08208 Sabadell
Barcelona - Spain

Factory
Terra Alta, 1-3 (Pol. Ind. Can Carner)
08211 Castellar del Vallès
Barcelona - Spain