Documentación Software Aplicacion para Android

Permisos

Para la impelementacion Bluetooth Low Energy “BLE” en dispositivos Android es necesario el uso de permisos dentro el archivo manifest.xml. Para que la aplicacion tengan un funcionamiento optimo los permisos que se deben implementar son:

```
<!-- Permission for Bluetoth Low Energy-->

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

BluetoothLeService.java

Esta clase es proporcionada por Android para el manejo de BLE, con mínimas modificaciones puede ser implementada y permite realizar todas las funciones de CALLBACK de BLE. Para poder leer es necesario llamar dentro de la función readCharacteristic(mreadcharacteristic) e ingresar la referencia en la función.

Estas funciones en Android funcionan de manera asíncrona, de tal manera que no siguen un orden secuencial, las funciones de nombre onXXXX son llamadas por defecto sin la necesidad de ser invocadas, en el caso anterior al llamar a ***readCharacteristic (),*** empieza automáticamente a correr la función ***onReadCharacteristic ().***

```
public void readCharacteristic(BluetoothGattCharacteristic characteristic) {
    if (mBluetoothAdapter == null || mBluetoothGatt == null) {
        Log.w(TAG,  msg: "BluetoothAdapter not initialized");
        return;
    }
    mBluetoothGatt.readCharacteristic(characteristic);
}
```

```
    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
                                     BluetoothGattCharacteristic characteristic,
                                     int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
            setCharacteristicNotification(characteristic,  enabled: true);
        }
    }
```

Esta funcion utilizan “broadcastUpdate()” para modificar el TextView , en esta zona se utiliza el ***setCharacteristicNotification(characteristic, true)*** , donde se envía la referencia de la característica y se activa, la opción de notificación en el descriptor de la característica del servicio, la opción de notificación debe ser previamente programada en el dispositivo central, y dentro de nuestro dispositivo periférico se podrá activar, de esta forma se logra cambiar los resultados, de la característica automáticamente sin necesidad de presionar el botón de lectura constantemente.

```
public void setCharacteristicNotification(BluetoothGattCharacteristic characteristic, boolean enabled) {
    if (mBluetoothAdapter == null || mBluetoothGatt == null) {
        Log.w(TAG,  msg: "BluetoothAdapter not initialized");
        return;
    }

    mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);
          // This is specific to Heart Rate Measurement.
    if (UUID_ACCELEROMETER_THESIS.equals(characteristic.getUuid())) {
        BluetoothGattDescriptor descriptor = characteristic.getDescriptor(
                UUID.fromString(SampleGattAttributes.CLIENT_CHARACTERISTIC_CONFIG));
        descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
        mBluetoothGatt.writeDescriptor(descriptor);
    }
}
```

Los datos recibidos por el bluetooth, vienen en una cadena de datos, los cuales tienen que ser descifrados directamente en el código, donde se podrán recibir hasta 20 bytes de comunicación, mientras que en el advertising se pueden utilizar hasta 31 bytes y con el uso de SCAN RESPONSE. Para poder descifrar los datos se utiliza un método genérico ofrecido por Android con el código.

```
        int flag = characteristic.getProperties();
        int format = -1;
        if ((flag & 0x01) != 0) {  /// Se realiza la modificacion de 0 a 1 , de esta forma pasamos  a formato UINT16
            format = BluetoothGattCharacteristic.FORMAT_UINT16;
            Log.d(TAG,  msg: "Accelerometer format UINT16.");
        } else {
            format = BluetoothGattCharacteristic.FORMAT_UINT8;
            Log.d(TAG,  msg: "Accelerometer format UINT8.");
        }
        final int heartRate = characteristic.getIntValue(format,  offset: 1);
        Log.d(TAG, String.format("Received heart rate: %d", heartRate));
        intent.putExtra(EXTRA_DATA, String.valueOf(heartRate));
    } else {
        // For all other profiles, writes the data formatted in HEX.
        final byte[] data = characteristic.getValue();
```

DeviceControlActivity.java

En esta clase se crean todas las funciones que se conectan directamente con los botones del layout, esta clase java hereda de activity, contiene las clases override onCreate (), onResume (), onPause (), onDestroy (), onCreateOptionsMenu (), onOptionsItemSelected (), las cuales se encargan de las funcionalidades básicas de BLE, a continuación se presenta una lista de sus funciones: OnCreate (): cada vez que se haga llamado a la actividad DeviceControlActivity.java, lo primero que realizara es entrar en el onCreate, siendo este su punto inicial.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.layout_activity_second);
    callId();
    mRealm =  Realm.getDefaultInstance();
    getIntentExtras();
    eventsFloatingBtn();

    Intent gattServiceIntent = new Intent( packageContext: DeviceControlActivity.this, BluetoothLeService.class);
    bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);

    ////GPS

    ActivityCompat.requestPermissions( activity: this,new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 1);
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED)
    {
        Toast.makeText( context: this,  text: "No se han definido los permisos necesarios", Toast.LENGTH_SHORT).show();
        return;
    }else
    {
```

Los Intent en este método son utilizados con el finn de obtener información enviada desde la clase DeviceScanActivity(), donde se recibe el nombre del dispositivo y su dirección. Una vez terminadas todas las funciones del onCreate, la actividad llama al método onResume ().

```
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
    if (mBluetoothLeService != null) {
        final boolean result = mBluetoothLeService.connect(mBluetoothDevice);
        Log.d(TAG,  msg: "Connect request result=" + result);
    }
}
```

En este punto se crea un registerReceiver() que es leído dentro del BroadcastUpdate en el método OnReceive(), al cual se le enviara el contexto y un intent que contiene los diferentes posibles estados de la conexión. Se emplean métodos de seguridad, preguntando si existe el servicio buscado.

```java
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mGattUpdateReceiver);
}
```

Cuando se cierra la aplicación por completo se hace llamado al método onDestroy() dondelo único que hace es resetear el servicio encontrado a nulo, y hacer la desconexión del servicio.

```java
@Override
protected void onDestroy() {
    super.onDestroy();
    unbindService(mServiceConnection);
    mBluetoothLeService = null;
}
```

Para hacer una conexión con el servicio previamente creado en la clase java BluetoothLe-Service.java es necesario implementar el siguiente código.

El cual se encarga de manejar las operaciones cuando se intenta realizar una conexión al servicio, o cuando se intenta desconectar del servicio. En caso de no poder iniciar la conexión al servicio, la actividad finaliza con el comando finish().

```java
// Code to manage Service lifecycle.
private final ServiceConnection mServiceConnection = new ServiceConnection() {

    @Override
    public void onServiceConnected(ComponentName componentName, IBinder service) {
        mBluetoothLeService = ((BluetoothLeService.LocalBinder) service).getService();
        if (!mBluetoothLeService.initialize()) {
            Log.e(TAG,  msg: "Unable to initialize Bluetooth");
            finish();
        }
        // Automatically connects to the device upon successful start-up initialization.
        mBluetoothLeService.connect(mBluetoothDevice);
    }

    @Override
    public void onServiceDisconnected(ComponentName componentName) {
        mBluetoothLeService = null;
    }
};
```

Por otro lado al encontrar el servicio se envía el comando connect(), si se establece la conexión. La información recibida, es el envío de paquetes de otras actividades que entran a la clase. Existen algunas condicionales que plantean medidas de seguridad para comprueban que haya llegado información. En caso de determinar que realmente hay información y no informacion basura, se procede a obtenerla y es enviada de la clase java BluetoothLEService.java.