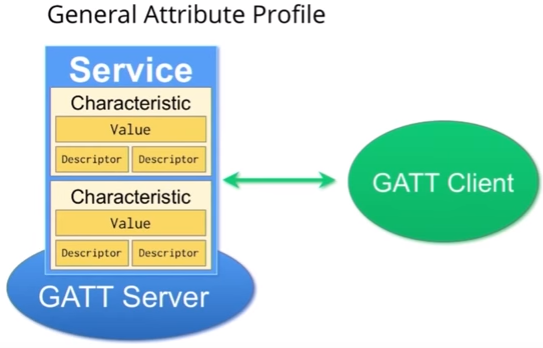
A Bluetooth Low Energy (BLE) bluetooth modulok és eszközök kizárólag **Android 4.3**-tól működnek, azaz **API Level 18**-tól. Az ilyen eszközök **nem képesek a klasszikus bluetooth megoldások Androidos kódjaival futn**i, tehát, ha ilyen eszközre is szeretnénk felcsatlakozni, akkor azt **külön le kell kezelnünk** a kódunkban. Továbbá ennek megfelelően nem is minden Androidos eszköz képes a BLE használatára.



A Gatt-ot támogató Androidos alkalmazásunk betölthet **szerver** szerepet, ekkor ő küldi az adatokat a másik eszköznek, vagy **kliens** szerepet, ekkor ő fogadja az adatokat a másik eszköztől. Továbbá az eszközök betölthetnek **központi** szerepet, ekkor figyelik, hogy ki akar rájuk csatlakozni, vagy betölthetnek **periféria** szerepet, ekkor csak hirdetik magukat, hogy ők tudnak csatlakozni a központra.

Minden eszköz megvalósít egy vagy akár több Generic Attribute (**GAtt) Profile**-t, mely leírja az adott eszköz működését. Minden Attribute-ot (Att) egy Unuversally Unique Identifier (**UUId**) azonosít, ami egy 128-bites sztring. Ezek szolgáltatásokat és karakterisztikákat azonosítanak. A **Characteristic** egy osztályzáshoz hasonló jellemző, amely egyetlen értéket és descriptorokat tartalmaz. Az úgynevezett **Descriptor**-ok pedig leírják a különböző ember által értelmezhető tulajdonságait a rendszernek. Ez lehet jellemző értéke, elfogadható tartomány, mértékegység, bármi. A **Service** szolgáltatás pedig egy karakterisztika gyűjtemény, amely leírja a szolgáltatás működését. Számtalan ilyen Gatt profil van, erről több információ megtalálható a: <https://www.bluetooth.com/specifications/adopted-specifications>

Ugyanazok az engedélyek szükségesek rá, mint a sima bluetooth-ra.

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

**De itt sem szabad elfelejteni, hogy Marshmallow (Android 6.0) alatt nem elég a Manifest.xml-ben felvenni az engedélyeket, hanem programkódból is le kell kezelni, hogy a felhasználó engedélyezi-e!!!**

**Ha az alkalmazásunk csak a BLE eszközöket kezeli le és a klasszikus bluetooth eszközöket nem**, akkor ezt fel kell tüntetni a Manifestben. A mienknél nem kell, mert mindkettőt szeretnénk támogatni.

**<uses-feature android:name="android.hardware.bluetooth\_le" android:required="true"/>**

**Ha mindkét eszköz típust kezelik**, akkor is kell ez a sor, csak más paraméterrel.

<uses-feature android:name="android.hardware.bluetooth\_le" android:required="**false**"/>

Hogy a telefonunk is támogatja-e a BLE-t, azt futásidőben kideríthetjük. Így a kódot elágaztathatjuk egy olyan esetre, amikor tudunk BLE-t kezelni, és amikor nem.

**if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE\_BLUETOOTH\_LE)) {**  
 //Toast.makeText(this, R.string.ble\_not\_supported, Toast.LENGTH\_SHORT).show();  
 //finish();  
**}**

**Android 4.3-nál már máshogy kapjuk meg a Bluetooth Adapter-t, hogy lekérhessük az elérhető eszközöket**, mint 4.0-nál. Itt egy BluetoothManager-t kell példányosítanunk, ami visszaadja nekünk az adaptert. Egyidejűleg nem tudjuk a klasszikus és ble eszközöket is lekérni, ezeket egymás után kell!

**private BluetoothAdapter mBluetoothAdapter;**

**final BluetoothManager bluetoothManager =  
 (BluetoothManager) getSystemService(Context.BLUETOOTH\_SERVICE);  
mBluetoothAdapter = bluetoothManager.getAdapter();**

Ha az adapter nem elérhető, vagy nem engedélyezett, akkor **megpróbáljuk bekapcsoltatni**:

**if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {  
 Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);  
 startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);  
}**

Az adapteren keresztül kapjuk meg az **elérhető eszközök listáját**. A **startLeScan** metódussal indítjuk el a keresést. De ez szívja az akkumulátor teljesítményét, így amint megtaláltuk amit keresünk állítjuk le a szkennelést, illetve sosem szkenneljünk folyamatosan. Adjunk meg idő limitet. A startLeScan metódus paraméterében egy **CallBack** függvényt kell megadjunk, ez hívódik meg a scannelés befejeztével. Ha pedig **csak bizonyos szolgáltatással rendelkezőekre** vagyunk kíváncsiak, akkor használhatjuk a startLeScan két paraméteres változatát, ahol az első egy **UUID tömb**, a második pedig a callback függvény.

**private static final long SCAN\_PERIOD = 10000;**

**private boolean szkenneljunkMeg;  
private Handler mHandler;**

**private void Szkenneles(final boolean enable) {  
 if (enable) {   
 //10mp után a szkennelés leállítjuk  
 mHandler.postDelayed(new Runnable() {  
 @Override  
 public void run() {  
 szkenneljunkMeg = false;  
 mBluetoothAdapter.stopLeScan(mLeScanCallback);  
 }  
 }, SCAN\_PERIOD);**

**szkenneljunkMeg = true;  
 mBluetoothAdapter.startLeScan(mLeScanCallback);  
 } else {  
 szkenneljunkMeg = false;  
 mBluetoothAdapter.stopLeScan(mLeScanCallback);  
 }  
}**

A felhasznált mLeScanCallback **callback függvénynek** itt egy konkrét megvalósítása.

**private LeDeviceListAdapter eszkozListaAdapter;**

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

**@Override  
 public void onLeScan(final BluetoothDevice device, int rssi,  
 byte[] scanRecord) {  
 runOnUiThread(new Runnable() {**

**@Override  
 public void run() {  
 eszkozListaAdapter.addDevice(device);  
 eszkozListaAdapter.notifyDataSetChanged();  
 }**

**});  
 }  
};**

A GAtt szerverre való **csatlakozás után egy BluetoothGatt típussal tér vissza** a metódus, melyen keresztül majd igénybe tudjuk venni a szerver szolgáltatásait. A metódusnak 3 paramétere van, egy **kontextus**, egy bool, hogy **automatikusan csatlakozzon-e** a szerverre az Androidos kliens alkalmazásunk, illetve egy **Callback** függvény. A callback-ban tudja feldolgozni a szerver válaszait a kliens alkalmazásunk.

**mBluetoothGatt = device.connectGatt(this, false, mGattCallback);**

A callback egy Sevice-ben van. Ez lép interakcióba a BLE eszközzel.

public class BluetoothLeService extends Service {

//ez csak a log-hoz kell, hogy tudjuk, hogy hol írja ki a log a sort  
 private final static String TAG = BluetoothLeService.class.getSimpleName();

private BluetoothManager mBluetoothManager;  
 private BluetoothAdapter mBluetoothAdapter;  
 private String mBluetoothDeviceAddress;  
 private BluetoothGatt mBluetoothGatt;

private static final int STATE\_DISCONNECTED = 0;  
 private static final int STATE\_CONNECTING = 1;  
 private static final int STATE\_CONNECTED = 2;

private int mConnectionState = STATE\_DISCONNECTED;

public final static String ACTION\_GATT\_CONNECTED =  
 "com.example.bluetooth.le.ACTION\_GATT\_CONNECTED"; //csatlakozott a GAtt szerverre

public final static String ACTION\_GATT\_DISCONNECTED =  
 "com.example.bluetooth.le.ACTION\_GATT\_DISCONNECTED"; //lecsatlakozott

public final static String ACTION\_GATT\_SERVICES\_DISCOVERED =  
 "com.example.bluetooth.le.ACTION\_GATT\_SERVICES\_DISCOVERED";  
//szolgáltatásokat fedezett fel

public final static String ACTION\_DATA\_AVAILABLE =  
 "com.example.bluetooth.le.ACTION\_DATA\_AVAILABLE";  
//adatokat fogadott az eszköztől, ami lehet egy olvasás eredménye, vagy egy értesítés

public final static String EXTRA\_DATA =  
 "com.example.bluetooth.le.EXTRA\_DATA";

public final static UUID UUID\_HEART\_RATE\_MEASUREMENT =  
 UUID.fromString(SampleGattAttributes.HEART\_RATE\_MEASUREMENT);

private final BluetoothGattCallback mGattCallback =  
 new BluetoothGattCallback() {  
 @Override  
 public void onConnectionStateChange(BluetoothGatt gatt, int status,  
 int newState) {  
 String intentAction;  
 if (newState == BluetoothProfile.STATE\_CONNECTED) {  
 intentAction = ACTION\_GATT\_CONNECTED;  
 mConnectionState = STATE\_CONNECTED;  
 broadcastUpdate(intentAction);  
 Log.i(TAG, "Connected to GATT server.");  
 Log.i(TAG, "Attempting to start service discovery:" +  
 mBluetoothGatt.discoverServices());  
 } else if (newState == BluetoothProfile.STATE\_DISCONNECTED) {  
 intentAction = ACTION\_GATT\_DISCONNECTED;  
 mConnectionState = STATE\_DISCONNECTED;  
 Log.i(TAG, "Disconnected from GATT server.");  
 broadcastUpdate(intentAction);  
 }  
 }

@Override  
 //új szolgáltatást fedeztünk fel  
 public void onServicesDiscovered(BluetoothGatt gatt, int status) {  
 if (status == BluetoothGatt.GATT\_SUCCESS) {  
 broadcastUpdate(ACTION\_GATT\_SERVICES\_DISCOVERED);  
 } else {  
 Log.w(TAG, "onServicesDiscovered received: " + status);  
 }  
 }

@Override  
 //a karakterisztika olvasás művelet eredménye  
 public void onCharacteristicRead(BluetoothGatt gatt,  
 BluetoothGattCharacteristic characteristic,  
 int status) {  
 if (status == BluetoothGatt.GATT\_SUCCESS) {  
 broadcastUpdate(ACTION\_DATA\_AVAILABLE, characteristic);  
 }  
 }

};

}

A broadcastUpdate metódus egy segéd metódus egy broadcast-ot küld. Az itten feldolgozás a Bluetoot Heart Rate Messaurement profil-nak megfelelő eszközzel működik.

private void broadcastUpdate(final String action) {  
 //általános működéshez  
 final Intent intent = new Intent(action);  
 sendBroadcast(intent);  
}

private void broadcastUpdate(final String action,  
 final BluetoothGattCharacteristic characteristic) {  
 final Intent intent = new Intent(action);

//a Heart Rate Measurement profil-hoz. Ahol az adatfeldolgolás az ott leírtaknak felel meg.  
 if (UUID\_HEART\_RATE\_MEASUREMENT.equals(characteristic.getUuid())) {  
 int flag = characteristic.getProperties();  
 int format = -1;  
 if ((flag & 0x01) != 0) {

format = BluetoothGattCharacteristic.FORMAT\_UINT16;  
 Log.d(TAG, "Heart rate format UINT16.");

} else {

format = BluetoothGattCharacteristic.FORMAT\_UINT8;  
 Log.d(TAG, "Heart rate format UINT8.");

}

final int heartRate = characteristic.getIntValue(format, 1);  
 Log.d(TAG, String.format("Received heart rate: %d", heartRate));

intent.putExtra(EXTRA\_DATA, String.valueOf(heartRate));

} else {

//az összes másik profilhoz az adatokat HEX-ben írjuk

final byte[] data = characteristic.getValue();  
 if (data != null && data.length > 0) {  
 final StringBuilder stringBuilder = new StringBuilder(data.length);  
 for(byte byteChar : data)  
 stringBuilder.append(String.format("%02X ", byteChar));

intent.putExtra(EXTRA\_DATA, new String(data) + "\n" + stringBuilder.toString());

}  
 }

sendBroadcast(intent);  
}

Az Aktivityben, ahol vezéreljük ezt az egészet, **el kell kapnunk a BroadcatReceiver üzeneteit**.

private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {

@Override  
 public void onReceive(Context context, Intent intent) {

final String action = intent.getAction();  
 if (BluetoothLeService.ACTION\_GATT\_CONNECTED.equals(action)) {

mConnected = true;  
 updateConnectionState(R.string.connected);  
 invalidateOptionsMenu();

} else if (BluetoothLeService.ACTION\_GATT\_DISCONNECTED.equals(action)) {

mConnected = false;  
 updateConnectionState(R.string.disconnected);  
 invalidateOptionsMenu();  
 clearUI();

} else if (BluetoothLeService.

ACTION\_GATT\_SERVICES\_DISCOVERED.equals(action)) {  
 displayGattServices(mBluetoothLeService.getSupportedGattServices());

} else if (BluetoothLeService.ACTION\_DATA\_AVAILABLE.equals(action)) {

displayData(intent.getStringExtra(BluetoothLeService.EXTRA\_DATA));

}  
 }  
};

A **displayGattServices** metódus az Activity-ben, mutatja meg az összes elérhető szolgáltatást és karakterisztikát a szerverhez. Ezek közül némelyik Attributumait lehet olvasni, másokat írni is. A mi eljárásunk csak bejárja ezeket és kiírja a GUI-ra.

private void displayGattServices(List<BluetoothGattService> gattServices) {

if (gattServices == null) return;

String uuid = null;  
 String unknownServiceString = getResources().getString(R.string.unknown\_service);  
 String unknownCharaString = getResources().getString(R.string.unknown\_characteristic);

ArrayList<HashMap<String, String>> gattServiceData =  
 new ArrayList<HashMap<String, String>>();

ArrayList<ArrayList<HashMap<String, String>>> gattCharacteristicData  
 = new ArrayList<ArrayList<HashMap<String, String>>>();

mGattCharacteristics = new ArrayList<ArrayList<BluetoothGattCharacteristic>>();

//a szolgáltatások bejárása  
 for (BluetoothGattService gattService : gattServices) {  
 HashMap<String, String> currentServiceData = new HashMap<String, String>();  
 uuid = gattService.getUuid().toString();

currentServiceData.put(LIST\_NAME, SampleGattAttributes.  
lookup(uuid, unknownServiceString));

currentServiceData.put(LIST\_UUID, uuid);  
gattServiceData.add(currentServiceData);

ArrayList<HashMap<String, String>> gattCharacteristicGroupData =  
 new ArrayList<HashMap<String, String>>();

List<BluetoothGattCharacteristic> gattCharacteristics =  
 gattService.getCharacteristics();

ArrayList<BluetoothGattCharacteristic> charas =  
 new ArrayList<BluetoothGattCharacteristic>();

// a szolgáltatáson belüli karakterisztikák bejárása  
 for (BluetoothGattCharacteristic gattCharacteristic : gattCharacteristics) {

charas.add(gattCharacteristic);  
 HashMap<String, String> currentCharaData = new HashMap<String, String>();  
 uuid = gattCharacteristic.getUuid().toString();

currentCharaData.put(LIST\_NAME, SampleGattAttributes.  
lookup(uuid,unknownCharaString));

currentCharaData.put(LIST\_UUID, uuid);

gattCharacteristicGroupData.add(currentCharaData);

}

mGattCharacteristics.add(charas);  
 gattCharacteristicData.add(gattCharacteristicGroupData);  
 }  
}

A legtöbb alkalmazás feliratkozik az eszközön történő **bizonyos karakterisztikák változásainak** eseményeire, hogy ezekről **értesítést kapjanak**. Ebben segít a **setCharacteristicNotification** metódus.

private BluetoothGatt mBluetoothGatt;  
BluetoothGattCharacteristic characteristic;  
boolean enabled;

mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);

BluetoothGattDescriptor descriptor = characteristic.getDescriptor(   
UUID.fromString(SampleGattAttributes.CLIENT\_CHARACTERISTIC\_CONFIG));  
descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);  
mBluetoothGatt.writeDescriptor(descriptor);

Ha feliratkoztunk és a karakterisztika megváltozik, akkor az **onCharacteristicChanged** callback fog meghívódni.

**@Override  
//a karakterisztika értesítés  
public void onCharacteristicChanged(BluetoothGatt gatt,  
 BluetoothGattCharacteristic characteristic) {  
 broadcastUpdate(ACTION\_DATA\_AVAILABLE, characteristic);  
}**

A kliens alkalmazás kapcsolatát a szerverrel az alábbi módon **zárhatjuk le**.

**public void close() {  
 if (mBluetoothGatt == null) {  
 return;  
 }  
 mBluetoothGatt.close();  
 mBluetoothGatt = null;  
}**