Bluetooth Low Energy in Android Java

1st Edition

Tony Gaitatzis

BackupBrain Publishing, 2017

ISBN: 978-1-7751280-4-5

backupbrain.co

Bluetooth Low Energy in Android Java

by Tony Gaitatzis

backupbrain@gmail.com

Services and Characteristics

Before data can be transmitted back and forth between a Central and Peripheral, the Peripheral must host a GATT Profile. That is, the Peripheral must have Services and Characteristics.

Identifying Services and Characteristics

Each Service and Characteristic is identified by a Universally Unique Identifier (UUID). The UUID follows the pattern 0000XXXX-0000-1000-8000-00805f9b34fb, so that a 32-bit UUID 00002a56-0000-1000-8000-00805f9b34fb can be represented as 0x2a56.

Some UUIDs are reserved for specific use. For instance any Characteristic with the 16-bit UUID 0x2a35 (or the 32-bit UUID 00002a35-0000-1000-8000-00805f9b34fb) is implied to be a blood pressure reading.

For a list of reserved Service UUIDs, see ***Appendix IV: Reserved GATT Services***.

For a list of reserved Characteristic UUIDs, see ***Appendix V: Reserved GATT Characteristics***.

Generic Attribute Profile

Services and Characteristics describe a tree of data access points on the peripheral. The tree of Services and Characteristics is known as the Generic Attribute (GATT) Profile. It may be useful to think of the GATT as being similar to a folder and file tree (Figure 6-1).

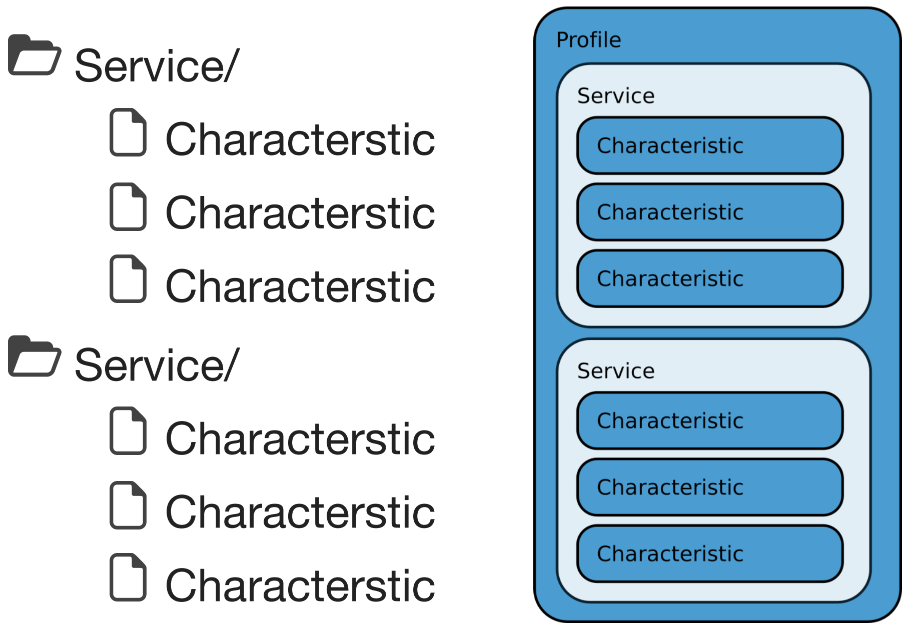


Figure 6-1. GATT Profile filesystem metaphor

Characteristics act as channels that can be communicated on, and Services act as containers for Characteristics. A top level Service is called a Primary service, and a Service that is within another Service is called a Secondary Service.

Permissions

Characteristics can be configured with the following attributes, which define what the Characteristic is capable of doing (Table 6-1):

Table 6-1. Characteristic Permissions

Descriptor	Description
Read	Central can read this Characteristic, Peripheral can set the value.
Write	Central can write to this Characteristic, Peripheral will be notified when the Characteristic value changes and Central will be notified when the write operation has occurred.
Notify	Central will be notified when Peripheral changes the value.

Because the GATT Profile is hosted on the Peripheral, the terms used to describe a Characteristic's permissions are relative to how the Peripheral accesses that Characteristic. Therefore, when a Central uploads data to the Peripheral, the Peripheral can "read" from the Characteristic. The Peripheral "writes" new data to the Characteristic, and can "notify" the Central that the data is altered.

Data Length and Speed

It is worth noting that Bluetooth Low Energy has a maximum data packet size of 20 bytes, with a 1 Mbit/s speed.

Programming the Central

The Central can be programmed to read the GATT Profile of the Peripheral after connection.

This is done by attaching a callback function to the connection gets called when the Central successfully connects to the Peripheral. When that happens, the Central requests the Service listing from the connected Peripheral, using the following command:

```
mBluetoothGatt.discoverServices();
```

This will prompt the Central to begin scanning the Peripheral for Services. The same BluetoothGattCallback used to handle the connection state change offers a callback specifically for discovered Services. This callback is executed when the Services are discovered.

Table 6-2. BluetoothGattCallback

Event	Description
onConnectionStateChange	Triggered when a BLE Peripheral connects or disconnects
onServicesDiscovered	Triggered when GATT Services are discovered
onCharacteristicRead	Triggered when data has been downloaded form a GATT Characteristic
onCharacteristicWrite	Triggered when data has been uploaded to a GATT Characteristic
onCharacteristicChanged	Triggered when a GATT Characteristic's data has changed

There are Primary Services and Secondary services. Secondary Services are contained within other Services; Primary Services are not. The type of Service can be discovered by inspecting the SERVICE_TYPE_PRIMARY flag.

```
int type = item.getType();
boolean isPrimary = (type == BluetoothGattService.SERVICE_TYPE_PRIMARY);
```

Each Service may contain one or more characteristics, which the Central can use to communicate with the Peripheral. Once the Peripheral's Services have been discovered, a map of each Service's Characteristics can be listed:

```
 public void onServicesDiscovered(BluetoothGatt gatt, int status) {
    // iterate through discovered services
    List<BluetoothGattService> gattServices = mBluetoothGatt.getServices();
    for (BluetoothGattService gattService : gattServices) {
        // iterate through service’s characteristics
        List<BluetoothGattCharacteristic> characteristics =
            gattService.getCharacteristics();
        for (BluetoothGattCharacteristic characteristic : characteristics) {
            // Do something with each Charateristic
        }
    }
}
```

Each Characteristic has certain permission properties that allow the Central to read, write, or receive notifications from it.

Table 6-3. BluetoothGattCharacteristic Permissions

Attribute	Permission	Description
PROPERTY_READ	Read	Central can read data altered by the Peripheral
PROPERTY_WRITE_NO_RESPONSE	Write	Central can send data. No response from Peripheral
PROPERTY_WRITE	Write	Central can send data, Peripheral reads
PROPERTY_NOTIFY	Notify	Central is notified as a result of a change

In Android, these properties are expressed as a binary integer. Permissions can be extracted like this:

```
int permissions = characteristic.getProperties(); // fetch permissions
boolean isWritable = permissions & BluetoothGattCharacteristic.PROPERTY_WRITE;
boolean isWritableNoResponse = permissions &
    BluetoothGattCharacteristic.PROPERTY_WRITE_NO_RESPONSE;
boolean isReadable = permissions & BluetoothGattCharacteristic.PROPERTY_READ;
boolean isNotifiable = permissions & \
    BluetoothGattCharacteristic.PROPERTY_NOTIFY;
```

A Note on Caching

Because Bluetooth was designed to be a low-power protocol, measures are taken to limit redundancy and power consumption through radio and CPU usage. As a result, a Peripheral's GATT Profile is cached on Android. This is not a problem for normal use, but when you are developing, it can be confusing to change Characteristic permissions and not see the updates reflected on Android.

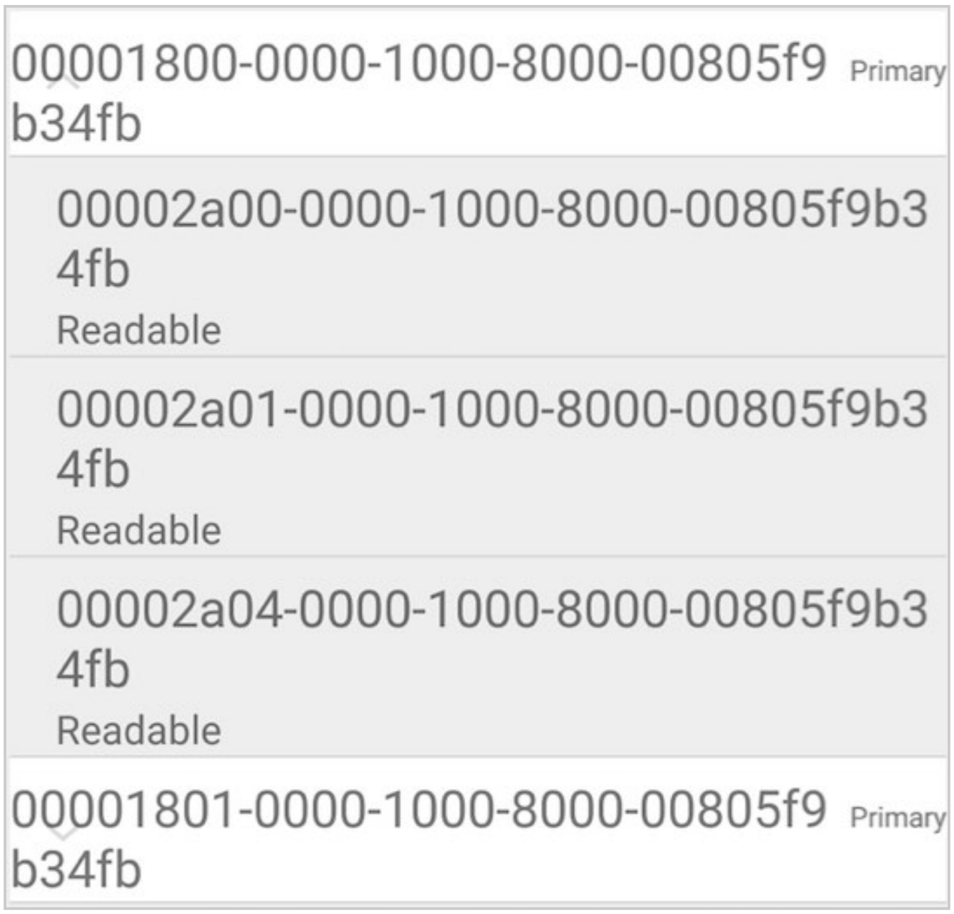
To get around this, the device cache can be refreshed:

```
 // use introspection to access hidden refresh method
// in the Bluetooth connection
Method localMethod = bluetoothGatt.getClass().getMethod(
    "refresh",
    new Class[0]
);
// if the refresh method exists, run it.
if (localMethod != null) localMethod.invoke(bluetoothGatt, new Object[0]));
```

Putting It All Together

Create a new project called ExampleBleServices and copy everything from the previous example.

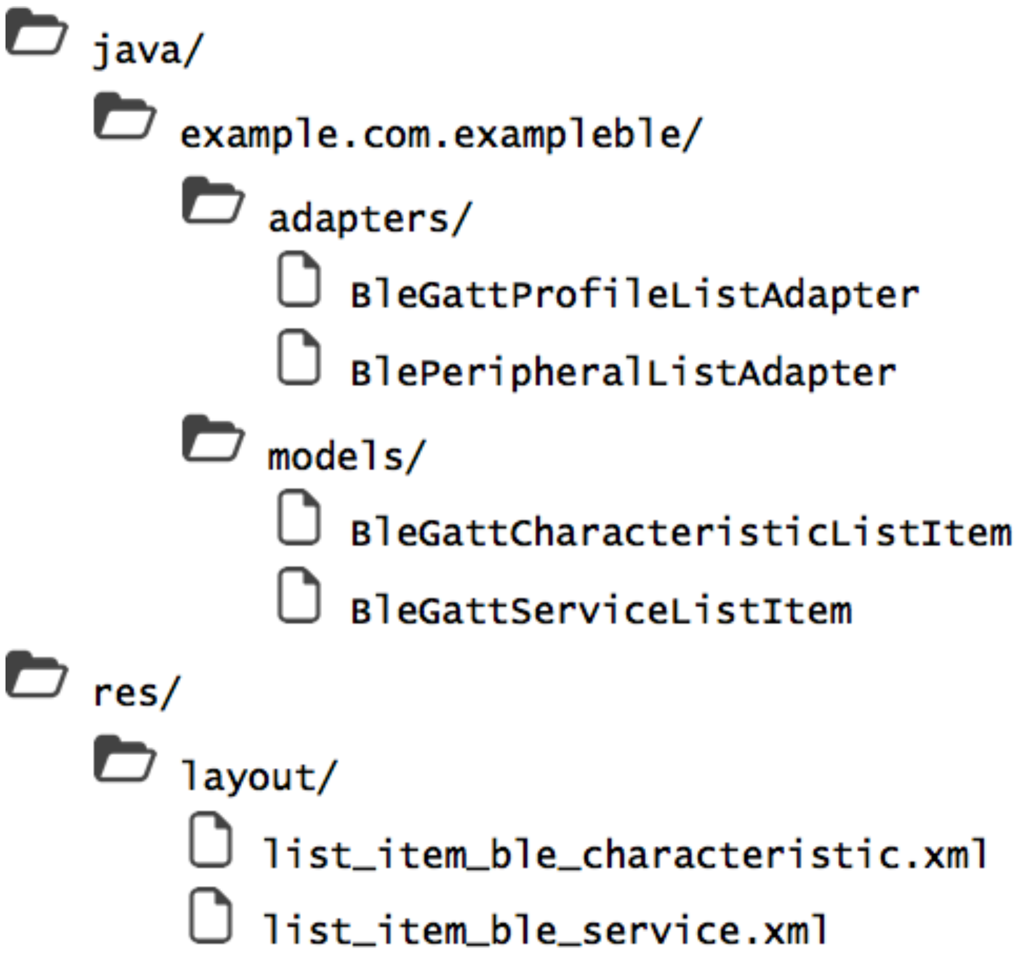
This app will work like the one from the previous chapter, except that once the it connects to the Peripheral, it will also list the GATT Profile for that peripheral. The GATT Profile will be displayed in an expandable list (Figure 6-3).



```
00001800-0000-1000-8000-00805f9b34fb    Primary
    00002a00-0000-1000-8000-00805f9b34fb
    Readable
    00002a01-0000-1000-8000-00805f9b34fb
    Readable
    00002a04-0000-1000-8000-00805f9b34fb
    Readable
00001801-0000-1000-8000-00805f9b34fb    Primary
```

Figure 6-3. GATT Profile downloaded from Peripheral

Create new classes and new layout resource and menu files in res/with the following names (Figure 6-4).



```
java/
    example.com.exampleble/
        adapters/
            BleGattProfileListAdapter
            BlePeripheralListAdapter
        models/
            BleGattCharacteristicListItem
            BleGattServiceListItem
res/
    layout/
        list_item_ble_characteristic.xml
        list_item_ble_service.xml
```

Figure 6-4. Project Structure

Resources

The Connect activity will now list the permissions of each characteristic. Add some definitions to res/values/strings.xml.

Example 6-1. res/values/strings.xml

```
...
    <string name="service_type_primary">Primary</string>
    <string name="service_type_secondary">Secondary</string>
    <string name="gatt_profile_list_empty">No GATT Profile Discovered</string>
    <string name="property_read">Readable</string>
    <string name="property_write">Writable</string>
    <string name="property_notify">Notifiable</string>
    <string name="property_none">No Access</string>
...
```

Objects

Modify BlePeripheral.java to inspect the permissions of a Characteristic and to disable device caching.

Example 6-2. java/example.com.exampleble/ble/BlePeripheral

```
...
    public boolean refreshDeviceCache(BluetoothGatt gatt) {
        try {
            BluetoothGatt localBluetoothGatt = gatt;
            Method localMethod =
            localBluetoothGatt.getClass().getMethod("refresh", new Class[0]);
            if (localMethod != null) {
                boolean bool = ((Boolean) localMethod.invoke(
                    localBluetoothGatt, new Object[0])).booleanValue();
                return bool;
            }
        }
        catch (Exception localException) {
            Log.e(TAG, "An exception occurred while refreshing device");
        }
        return false;
    }
    // true if characteristic is writable
    public static boolean isCharacteristicWritable(
        BluetoothGattCharacteristic pChar) {
        return (pChar.getProperties() &
            (BluetoothGattCharacteristic.PROPERTY_WRITE |
             BluetoothGattCharacteristic.PROPERTY_WRITE_NO_RESPONSE)) != 0;
    }
    // true if characteristic is readable
    public static boolean isCharacteristicReadable(
        BluetoothGattCharacteristic pChar) {
        return ((pChar.getProperties() &
        BluetoothGattCharacteristic.PROPERTY_READ) != 0);
    }
    // true if characteristic can send notifications
    public static boolean isCharacteristicNotifiable(
        BluetoothGattCharacteristic pChar) {
        return (pChar.getProperties() &
        BluetoothGattCharacteristic.PROPERTY_NOTIFY) != 0;
    }
...
```

Views

The GATT Profile will be represented as an Expandable List View, with BleGattServ-iceListItem as the parent nodes and BleGattCharacteristicListItems as the child nodes.

Each BleGattServiceListItem will display the UUID and type of a Service.

Example 6-3. java/example.com.exampleble/models/BleGattServiceListItem

```
package example.com.exampleble.models;
public class BleGattServiceListItem {
    private final int mItemId;
    private final BluetoothGattService mService;
    public BleGattServiceListItem(BluetoothGattService gattService,
            int serviceItemID) {
        mItemId = serviceItemID;
        mService = gattService;
    }
    public int getItemId() { return mItemId; }
    public UUID getUuid() { return mService.getUuid(); }
    public int getType() { return mService.getType(); }
    public BluetoothGattService getService() { return mService; }
}
```

The corresponding view will display details of a Service. It has two text fields: one for the UUID and one for the type (Primary or Secondary).

Example 6-4. res/layout/list_item_ble_service.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:background="@color/white">
    <TextView
        android:id="@+id/uuid"
        android:layout_width="276dp"
        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:paddingTop="@dimen/text_padding"/>
    <TextView
        android:id="@+id/service_type"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="10sp"
        android:paddingTop="@dimen/text_padding"/>
```

```
</LinearLayout>
```

BleGattCharacteristicListItem is similar. It displays the UUID and the permissions of a Characteristic.

Example 6-5. java/example.com.exampleble/models/BleGattCharacteristicListItem

```
package example.com.exampleble.models;
public class BleGattCharacteristicListItem {
    private int mItemId;
    private BluetoothGattCharacteristic mCharacteristic;
    public BleGattCharacteristicListItem(
            BluetoothGattCharacteristic characteristic, int itemId) {
        mCharacteristic = characteristic;
        mItemId = itemId;
    }
    public int getItemId() { return mItemId; }
    public BluetoothGattCharacteristic getCharacteristic() {
        return mCharacteristic;
    }
}
```

To connect these into a collapsable list, extend a BaseExpandableListAdapter, which allows list items to be grouped and child list items to be created. The “parent” list items will represent Services and the “child” list items will represent Characteristics.

Example 6-6. java/example.com.exampleble/adapters/BLEServiceListAdapter

```
package example.com.exampleble.adapters;
public class BleGattProfileListAdapter extends BaseExpandableListAdapter {
    private final static String TAG = \
        BleGattProfileListAdapter.class.getSimpleName();

    private ArrayList<BleGattServiceListItem> mBleGattServiceListItems = \
        new ArrayList<>(); // list of Services
    // list of Characteristics
    private Map<Integer, ArrayList<BleGattCharacteristicListItem>> \
        mBleCharacteristicListItems = \
        new HashMap<Integer, ArrayList<BleGattCharacteristicListItem>>();

    /**
     * Instantiate the class
     */
    public BleGattProfileListAdapter() {
    }

    /** PARENT (GATT SERVICE) METHODS FOR COLLAPSIBLE TREE STRUCTURE **/

    /**
     * Get the Service ListItem listed in a groupPosition
     * @param groupPosition the position of the ListItem
     * @return the ServiceListItem
     */
    @Override
    public BleGattServiceListItem getGroup(int groupPosition) {
        return mBleGattServiceListItems.get(groupPosition);
    }

    /**
     * How many Services are listed
     * @return number of Services
     */
    @Override
    public int getGroupCount() {
        return mBleGattServiceListItems.size();
    }

    /**
     * Add a new Service to be listed in the ListView
     * @param service the GATT Service to add
     */
```

```
public void addService(BluetoothGattService service) {
    int serviceItemID = mBleGattServiceListItems.size();
    BleGattServiceListItem serviceListItem = \
        new BleGattServiceListItem(service, serviceItemID);
    mBleGattServiceListItems.add(serviceListItem);
    mBleCharacteristicListItems.put(
        serviceItemID,
        new ArrayList<BleGattCharacteristicListItem>()
    );
}

/**
 * Add a new Characteristic to be listed in the ListView
 *
 * @param service the Service that this Characteristic belongs to
 * @param characteristic the Gatt Characteristic to add
 * @throws Exception if such a service does not exist
 */
public void addCharacteristic(
    BluetoothGattService service,
    BluetoothGattCharacteristic characteristic) throws Exception
{
    // find the Service in this listView with
    // matching UUID from the input service
    int serviceItemId = -1;
    for (BleGattServiceListItem bleGattServiceListItem :
        mBleGattServiceListItems)
    {
        //serviceItemId++;
        if (bleGattServiceListItem.getService().getUuid().equals(
            service.getUuid()))
        {
            Log.v(
                TAG,
                "Service found with UUID: "+service.getUuid().toString()
            );
```

```
                serviceItemId = bleGattServiceListItem.getItemId();
            }
        }
        // Throw an exception if no such service exists
        if (serviceItemId < 0) throw new Exception(
            "Service not found with UUID: "+service.getUuid().toString()
        );
        // add characterstic to the end of the sub-list for the parent service
        if (!mBleCharacteristicListItems.containsKey(serviceItemId)) {
            mBleCharacteristicListItems.put(
                serviceItemId,
                new ArrayList<BleGattCharacteristicListItem>()
            );
        }
        int characteristicItemId = mBleCharacteristicListItems.size();
        BleGattCharacteristicListItem characteristicListItem = \
            new BleGattCharacteristicListItem(
                characteristic,
                characteristicItemId
            );
        mBleCharacteristicListItems.get(
            serviceItemId
        ).add(characteristicListItem);
    }

    /**
     * Clear all ListItems from ListView
     */
    public void clear() {
        mBleGattServiceListItems.clear();
        mBleCharacteristicListItems.clear();
    }

    /**
     * Get the Service ListItem at some position
     *
     * @param position the position of the ListItem
     * @return BleGattServiceListItem at position
     */
    @Override
    public long getGroupId(int position) {
        return mBleGattServiceListItems.get(position).getItemId();
    }

    /**
     * This GroupViewHolder represents what UI components
     * are in each Service ListItem in the ListView
     */
    public static class GroupViewHolder{
        public TextView mUuidTV;
        public TextView mServiceTypeTV;
    }

    /**
     * Generate a new Service ListItem for some known position in the ListView
     *
     * @param position the position of the Service ListItem
     * @param convertView An existing List Item
     * @param parent The Parent ViewGroup
     * @return The Service List Item
     */
    @Override
    public View getGroupView(
        int position,
        boolean isExpanded,
        View convertView,
        ViewGroup parent)
    {
        View v = convertView;
        GroupViewHolder serviceListItemView;
        // if this ListItem does not exist yet, generate it
        // otherwise, use it
        if(convertView == null) {
            // convert list_item_peripheral.xml.xml to a View
            LayoutInflater inflater = LayoutInflater.from(parent.getContext());
            v = inflater.inflate(
                R.layout.list_item_ble_service,
                parent,
                false
            );
            // match the UI stuff in the list Item to what's in the xml file
            serviceListItemView = new GroupViewHolder();
            serviceListItemView.mUuidTV = (TextView) v.findViewById(R.id.uuid);
            serviceListItemView.mServiceTypeTV = (TextView) v.findViewById(
                R.id.service_type
            );
            v.setTag( serviceListItemView );
        } else {
            serviceListItemView = (GroupViewHolder) v.getTag();
        }
        // if there are known Services, create a ListItem that says so
        // otherwise, display a ListItem with Bluetooth Service information
        if (getGroupCount() <= 0) {
            serviceListItemView.mUuidTV.setText(
                R.string.peripheral_list_empty
            );
        } else {
            BleGattServiceListItem item = getGroup(position);
            serviceListItemView.mUuidTV.setText(item.getUuid().toString());
            int type = item.getType();
            // Is this a primary or secondary service
            if (type == BluetoothGattService.SERVICE_TYPE_PRIMARY) {
                serviceListItemView.mServiceTypeTV.setText(
                    R.string.service_type_primary
                );
            } else {
                serviceListItemView.mServiceTypeTV.setText(
                    R.string.service_type_secondary
                );
            }
        }
        return v;
    }

    /**
     * The IDs for this ListView do not change
     * @return <b>true</b>
     */
    @Override
    public boolean hasStableIds() {
        return true;
    }

    /** CHILD (GATT CHARACTERISTIC) METHODS FOR COLLAPSIBLE TREE STRUCTURE **/

    /**
     * Get the Characteristic ListItem at some position in the ListView
     * @param groupPosition the position of the Service ListItem in the ListView
     * @param childPosition the sub-position of the Characteristic ListItem under
    the Service
     * @return BleGattCharactersiticListItem at some groupPosition, childPosition
     */
    @Override
    public BleGattCharacteristicListItem getChild(
        int groupPosition,
        int childPosition
    ) {
        return mBleCharacteristicListItems.get(
            groupPosition
        ).get(childPosition);
    }

    /**
     * Get the ID of a Charactersitic ListItem
     *
```

```
     * @param groupPosition the position of a Service ListItem
     * @param childPosition The sub-position of a Characteristic ListItem
     * @return the ID of the Characteristic ListItem
     */
    @Override
    public long getChildId(int groupPosition, int childPosition) {
        return mBleCharacteristicListItems.get(
            groupPosition
        ).get(childPosition).getItemId();
    }

    /**
     * How many Characteristics exist under a Service
     *
     * @param groupPosition The position of the Service ListItem in the ListView
     * @return the number of Characteristics in this Service
     */
    @Override
    public int getChildrenCount(int groupPosition) {
        return mBleCharacteristicListItems.get(groupPosition).size();
    }

    /**
     * Charactersitics are selectable
     * because the user can click on them to open the TalkActivity
     *
     * @param groupPosition The Service ListItem position
     * @param childPosition The Charecteristic ListItem sub-position
     * @return <b>true</b>
     */
    @Override
    public boolean isChildSelectable(int groupPosition, int childPosition) {
        return true;
    }

    /**
```

```
     * This ChildViewHolder represents what
     * UI components are in each Characteristic List Item in the ListView
     */
    public static class ChildViewHolder {
        // displays UUID
        public TextView mUuidTV;
        // displays when Characteristic is readable
        public TextView mPropertyReadableTV;
        // displays when Characteristic is writeable
        public TextView mPropertyWritableTV;
        // displays when Characteristic is Notifiable
        public TextView mPropertyNotifiableTV;
        // displays when no access is given
        public TextView mPropertyNoneTV;
    }

    /**
     * Generate a new Characteristic ListItem
     * for some known position in the ListView
     *
     * @param groupPosition the position of the Service ListItem
     * @param childPosition the position of the Characterstic ListItem
     * @param convertView An existing List Item
     * @param parent The Parent ViewGroup
     * @return The Characteristic ListItem
     */
    @Override
    public View getChildView(
        final int groupPosition,
        final int childPosition,
        boolean isLastChild,
        View convertView,
        ViewGroup parent)
    {
        View v = convertView;
        ChildViewHolder characteristicListItemView;
        BleGattCharacteristicListItem item = getChild(
            groupPosition,
            childPosition
        );
        BluetoothGattCharacteristic characteristic = item.getCharacteristic();
        // if this ListItem does not exist yet, generate it
        // otherwise, use it
        if (convertView == null) {
            LayoutInflater inflater = LayoutInflater.from(
                parent.getContext()
            );
            v = inflater.inflate(R.layout.list_item_ble_characteristic, null);
            // match the UI stuff in the list Item to what's in the xml file
            characteristicListItemView = new ChildViewHolder();
            characteristicListItemView.mUuidTV = \
                (TextView) v.findViewById(R.id.uuid);
            characteristicListItemView.mPropertyReadableTV = \
                (TextView)v.findViewById(R.id.property_read);
            characteristicListItemView.mPropertyWritableTV = \
                (TextView)v.findViewById(R.id.property_write);
            characteristicListItemView.mPropertyNotifiableTV = \
                (TextView)v.findViewById(R.id.property_notify);
            characteristicListItemView.mPropertyNoneTV = \
                (TextView)v.findViewById(R.id.property_none);
        } else {
            characteristicListItemView = (ChildViewHolder) v.getTag();
        }
        if (characteristicListItemView != null) {
            // display the UUID of the characteristic
            characteristicListItemView.mUuidTV.setText(
                characteristic.getUuid().toString()
            );
            // Display the read/write/notify attributes of the Characteristic
            if (BlePeripheral.isCharacteristicReadable(characteristic)) {
                characteristicListItemView.mPropertyReadableTV.setVisibility(
                    View.VISIBLE
                );
            } else {
                characteristicListItemView.mPropertyReadableTV.setVisibility(
                    View.GONE
                );
            }
            if (BlePeripheral.isCharacteristicWritable(characteristic)) {
                characteristicListItemView.mPropertyWritableTV.setVisibility(
                    View.VISIBLE
                );
            } else {
                characteristicListItemView.mPropertyWritableTV.setVisibility(
                    View.GONE
                );
            }
            if (BlePeripheral.isCharacteristicNotifiable(characteristic)) {
                characteristicListItemView.mPropertyNotifiableTV.setVisibility(
                    View.VISIBLE
                );
            } else {
                characteristicListItemView.mPropertyNotifiableTV.setVisibility(
                    View.GONE
                );
            }
            if (!BlePeripheral.isCharacteristicNotifiable(characteristic) && \
                !BlePeripheral.isCharacteristicWritable(characteristic) && \
                !BlePeripheral.isCharacteristicReadable(characteristic))
            {
                characteristicListItemView.mPropertyNoneTV.setVisibility(
                    View.VISIBLE
                );
            } else {
                characteristicListItemView.mPropertyNoneTV.setVisibility(
                    View.GONE
                );
            }
            }
            return v;
        }
}
```

Activities

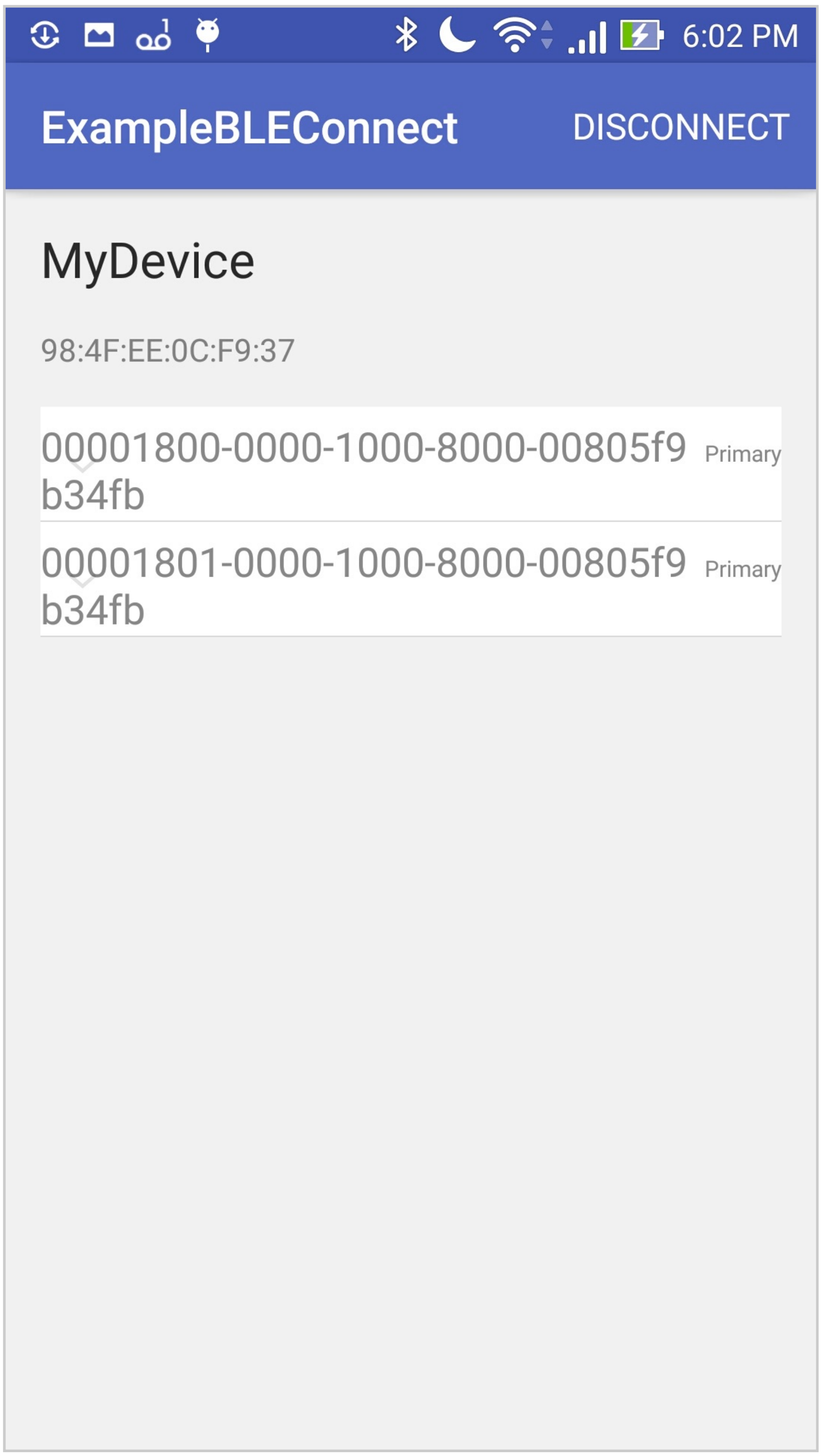
Add functionality in the Connect Activity to scan for Services when a Bluetooth Peripheral is connected, and to list those Services when they are discovered.

Example 6-7. java/example.com.exampleble/ConnectActivity.java

```
...
    private ExpandableListView mGattProfileListView;
    private BleGattProfileListAdapter mGattProfileListAdapter;
    private TextView mGattProfileListEmptyTV;
... // onCreate, onResume, onPause
    public void loadUI() {
        mPeripheralAdvertiseNameTV = \
            (TextView) findViewById(R.id.advertise_name);
        mPeripheralAddressTV = (TextView)findViewById(R.id.mac_address);
        mGattProfileListEmptyTV = \
            (TextView)findViewById(R.id.gatt_profile_list_empty);
        mGattProfileListView = \
            (ExpandableListView) findViewById(R.id.peripherals_list);
        mGattProfileListAdapter = new BleGattProfileListAdapter();
        mGattProfileListView.setAdapter(mGattProfileListAdapter);
        mGattProfileListView.setEmptyView(mGattProfileListEmptyTV);
    }
...
    // connect, disconnect, onBleConnected, etc
    /**
     * Bluetooth Peripheral GATT Profile discovered.  Update UI
     *
     * New in this chapter
     */
    public void onBleServiceDiscoveryStopped() {
        // update UI to reflect the GATT profile of the connected Perihperal
        mProgressSpinner.setVisible(false);
        mConnectItem.setVisible(false);
        mDisconnectItem.setVisible(true);
    }

    private BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
        /**
         * Characteristic value changed
         */
        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt,
                final BluetoothGattCharacteristic characteristic) {
            // We don't care about this here as we aren't communicating yet
        }
        /**
         * Peripheral connected or disconnected
         */
        @Override
        public void onConnectionStateChange(BluetoothGatt gatt,
                int status, int newState) {
            // There has been a connection or a disconnection
            // with a Peripheral.
            // If this is a connection, update the UI to reflect the change
            // and discover the GATT profile of the connected Peripheral
            if (newState == BluetoothProfile.STATE_CONNECTED) {
                Log.v(TAG, "Connected to peripheral");
                mBleConnected = true;
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        onBleConnected();
                        onBleServiceDiscoveryStarted();
                    }
                });
                gatt.discoverServices();
            } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
                mBlePeripheral.close();
                mBleConnected = false;
                if (mLeaveActivity) quitActivity();
            }
        }
        /**
         * Gatt Profile discovered
         */
        @Override
        public void onServicesDiscovered(
            BluetoothGatt bluetoothGatt,
            int status)
        {
            // if services were discovered, iterate through and display them
            if (status == BluetoothGatt.GATT_SUCCESS) {
                List<BluetoothGattService> services =
                    bluetoothGatt.getServices();
                for (BluetoothGattService service : services) {
                    if (service != null) {
                        Log.v(TAG, "Service uuid: " + service.getUuid());
                        // add the gatt service to our list
                        mGattProfileListAdapter.addService(service);
                        // update the UI to reflect the new Service
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mGattProfileListAdapter.notifyDataSetChanged();
                            }
                        });
                        // ask for this service's characteristics:
                        List<BluetoothGattCharacteristic> characteristics =
                            service.getCharacteristics();
                        for (BluetoothGattCharacteristic characteristic :
                                characteristics) {
```

```
                            if (characteristic != null) {
                                // add Characteristics to the Service's list
                                try {
                                    mGattProfileListAdapter.addCharacteristic(
                                        service, characteristic);
                                    // update the ListView UI
                                    runOnUiThread(new Runnable() {
                                        @Override
                                        public void run() {
                                            mGattProfileListAdapter.
                                                notifyDataSetChanged();
                                        }
                                    });
                                } catch (Exception e) {
                                    Log.e(TAG, e.getMessage());
                                }
                            }
                        }
                    }
                }
                disconnect(); // disconnect from the Peripheral
            } else {
                Log.e(TAG, "Problem discovering GATT services");
            }
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    onBleServiceDiscoveryStopped();
                }
            });
        }
    };
...
```

The resulting app will be able to connect to a Peripheral (Figure 6-5) and list the Services and Characteristics (Figure 6-6).



6:02 PM

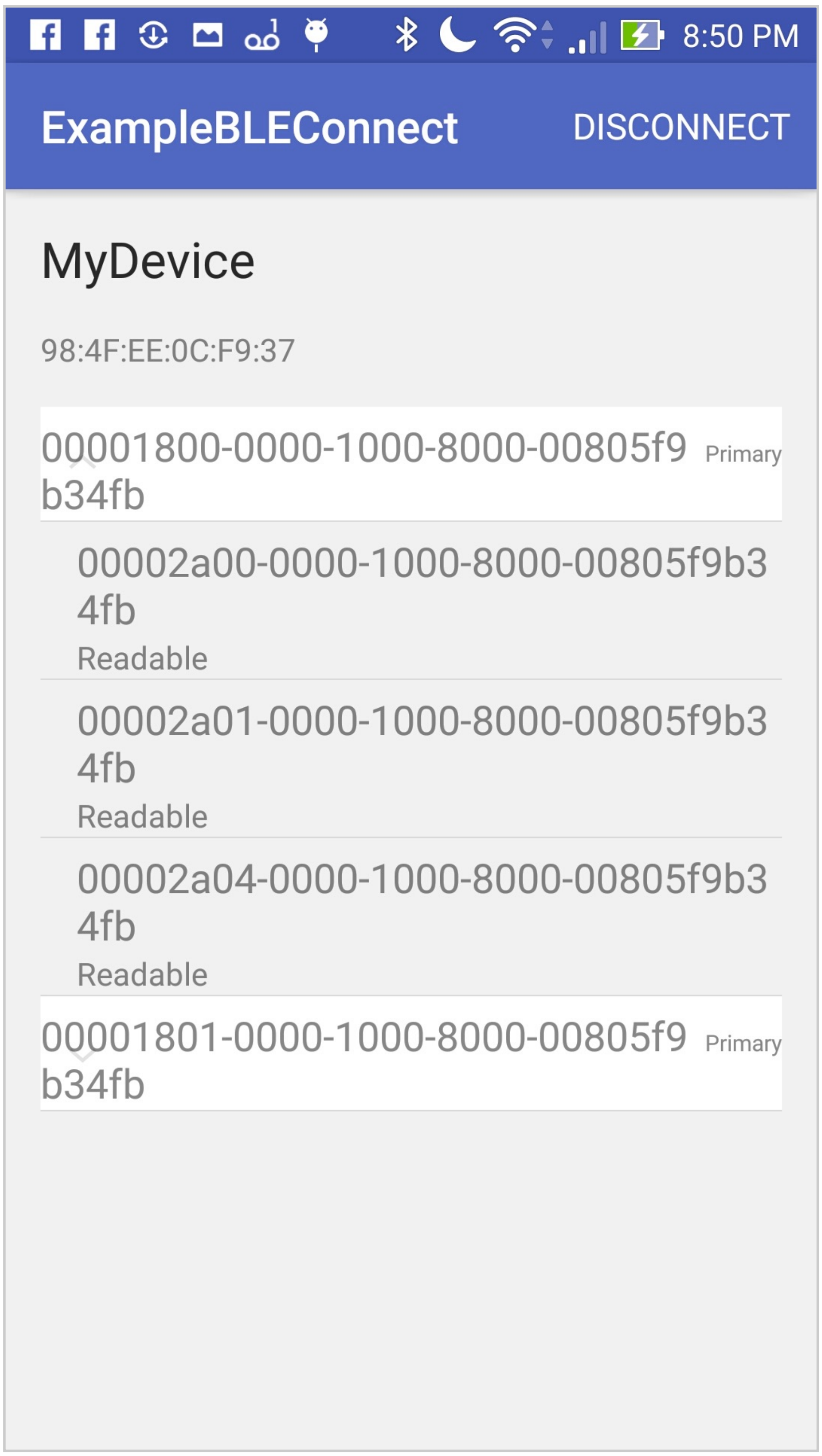
ExampleBLEConnect DISCONNECT

MyDevice

98:4F:EE:0C:F9:37

00001800-0000-1000-8000-00805f9b34fb Primary

00001801-0000-1000-8000-00805f9b34fb Primary



8:50 PM

ExampleBLEConnect DISCONNECT

MyDevice

98:4F:EE:0C:F9:37

00001800-0000-1000-8000-00805f9b34fb Primary

00002a00-0000-1000-8000-00805f9b34fb
Readable

00002a01-0000-1000-8000-00805f9b34fb
Readable

00002a04-0000-1000-8000-00805f9b34fb
Readable

00001801-0000-1000-8000-00805f9b34fb Primary

Figure 6-5. App screen showing primary Services hosted by a connected Peripheral

Figure 6-6. App screen showing an expanded primary Service, revealing several Characteristics

Programming the Peripheral

The Peripheral can be programmed to host a GATT Profile - the tree structure of Services and Characteristics that a connected Central will use to communicate with the Peripheral.

Services are created and added to the Peripheral's GATT Server via the BluetoothGattServer class. Instantiation requires the definition of a BluetoothGattServerCallback, which triggers callbacks when Characteristics are read from or written to.

New services are created and added to the Peripheral like this:

```
// Service UUID
UUID mServiceUuid = UUID.fromString("0000180c-0000-1000-8000-00805f9b34fb");

// Create a new Service
mService = new BluetoothGattService(
        mServiceUuid,
        BluetoothGattService.SERVICE_TYPE_PRIMARY);

// Add Service to the GATT Server
mGattServer.addService(mService);

// add the Service UUID to the Advertising Packet
advertiseBuilder.addServiceUuid(new ParcelUuid(serviceUuid));
```

Secondary (nested) Services are created like this:

```
// Service UUID
UUID mSecondaryServicUuid = \
    UUID.fromString("0000180d-0000-1000-8000-00805f9b34fb");

// Create a new Secondary Service
mSecondaryService = new BluetoothGattService(
        mSecondaryServicUuid,
        BluetoothGattService.SERVICE_TYPE_SECONDARY);
```

```
// Add Secondary Service to an existing Service
mService.addService(mSecondaryService);
```

New Characteristics are created and added to a Service like this:

```
// Create a new Characteristic
mCharacteristic = new BluetoothGattCharacteristic(
        UUID.fromString("00002a00-0000-1000-8000-00805f9b34fb"),
        BluetoothGattCharacteristic.PROPERTY_READ,
        BluetoothGattCharacteristic.PERMISSION_READ);

// Add the Characteristic to the Service
mService.addCharacteristic(mCharacteristic);
Characteristics must have both Properties and Permissions.
```

Properties are the access level that Centrals have to data, such as PROPERTY_READ, PROPERTY_WRITE, or PROPERTY_NOTIFY.

Some common Properties are given below:

Table 6-4. Common BluetoothGattCharacteristic Properties

Property	Description
PROPERTY_READ	Characteristic may be read from
PROPERTY_WRITE_NO_RESPONSE	Characteristic supports notifications
PROPERTY_WRITE	Characteristic may be written to
PROPERTY_NOTIFY	Characteristic may be written to but will not respond with a confirmation

Permissions are the ability for a Central to change a Descriptor within the Characteristic. For example, if the Central needs to be able to subscribe to a Characteristic, it will need to have PERMISSION_WRITE permission to that Characteristic.

Table 6-5. Common BluetoothGattCharacteristic Permissions

Permission	Description
PERMISSION_READ	Characteristic Descriptors may be read
PERMISSION_WRITE	Characteristic Descriptors may be modified

Properties and Permissions can be chained using the binary OR (|) operator.

For example, a read-only Characteristic that for which notifications can be subscribed to:

```
mCharacteristic = new BluetoothGattCharacteristic(
        UUID.fromString("00002a00-0000-1000-8000-00805f9b34fb"),
        BluetoothGattCharacteristic.PROPERTY_READ | \
            BluetoothGattCharacteristic.PROPERTY_NOTIFY,
        BluetoothGattCharacteristic.PERMISSION_READ | \
            BluetoothGattCharacteristic.PERMISSION_WRITE);
```

Or a write-only Characteristic without notifications:

```
mCharacteristic = new BluetoothGattCharacteristic(
        UUID.fromString("00002a00-0000-1000-8000-00805f9b34fb"),
        BluetoothGattCharacteristic.PROPERTY_WRITE,
        BluetoothGattCharacteristic.PERMISSION_READ);
```

A Note on GATT Profile Best Practices

All Peripherals should contain Device information and a Battery Service, resulting in a minimal GATT profile for any Peripheral that resembles this (Figure 6-7):

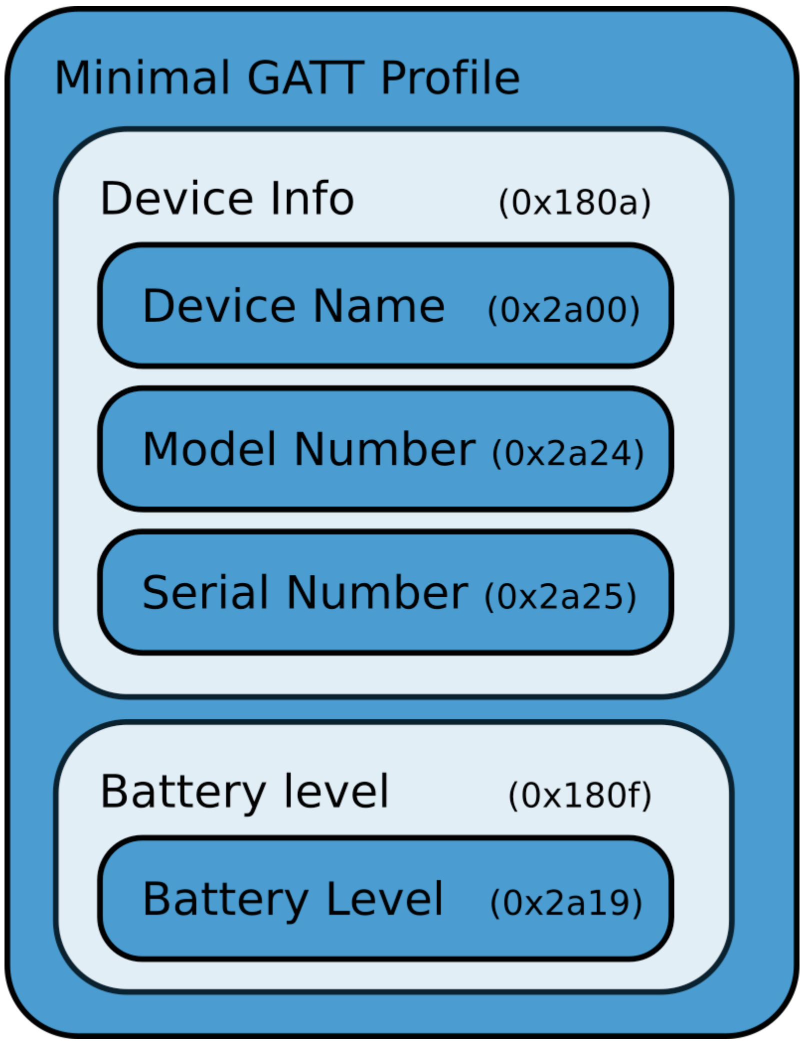


Figure 6-7. Minimal GATT

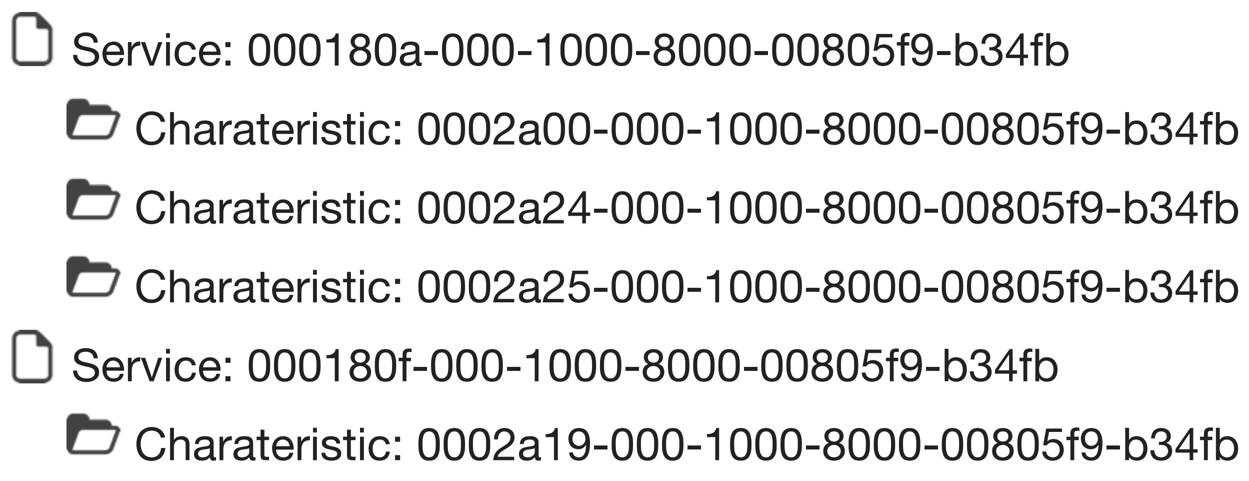
This provides Central software, surveying tools, and future developers to better understand what each Peripheral is, how to interact with it, and what the battery capabilities are.

For pedagogical reasons, many of the examples will not include this portion of the GATT Profile.

Putting It All Together

Create a new project called ExampleBlePeripheral and copy everything from the previous example.

This app will work like the one from the previous chapter, except that it will host a minimal GATT profile (Figure 6-8).



Service: 000180a-000-1000-8000-00805f9-b34fb

Charateristic: 0002a00-000-1000-8000-00805f9-b34fb

Charateristic: 0002a24-000-1000-8000-00805f9-b34fb

Charateristic: 0002a25-000-1000-8000-00805f9-b34fb

Service: 000180f-000-1000-8000-00805f9-b34fb

Charateristic: 0002a19-000-1000-8000-00805f9-b34fb

Figure 6-8. update image for minimal Gatt Profile

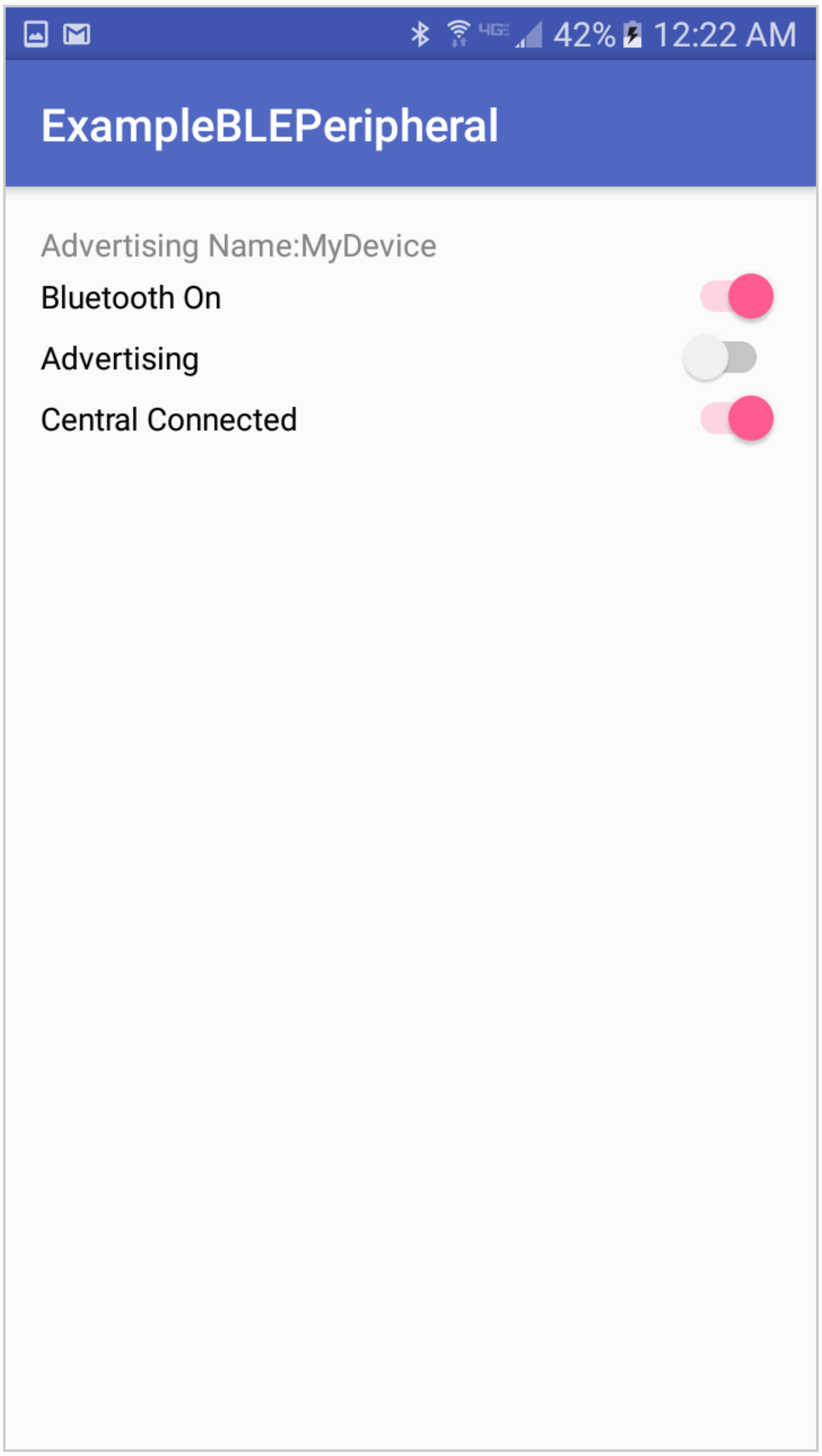
Objects

Modify BlePeripheral.java to build a minimal Gatt Services profile, including mock data to populate the Characteristic values.

Example 6-8. java/example.com.exampleble/ble/BlePeripheral

```
...
    public static final String CHARSET = "ASCII";

    // fake data for GATT Profile
    private static final String MODEL_NUMBER = "1AB2";
    private static final String SERIAL_NUMBER = "1234";

    /** Peripheral and GATT Profile **/
    public static final String ADVERTISING_NAME =  "MyDevice";

    public static final UUID DEVICE_INFORMATION_SERVICE_UUID = \
        UUID.fromString("0000180a-0000-1000-8000-00805f9b34fb");
    public static final UUID BATTERY_LEVEL_SERVICE = \
        UUID.fromString("0000180f-0000-1000-8000-00805f9b34fb");

    public static final UUID DEVICE_NAME_CHARACTERISTIC_UUID = \
        UUID.fromString("00002a00-0000-1000-8000-00805f9b34fb");
    public static final UUID MODEL_NUMBER_CHARACTERISTIC_UUID = \
        UUID.fromString("00002a24-0000-1000-8000-00805f9b34fb");
    public static final UUID SERIAL_NUMBER_CHARACTERISTIC_UUID = \
        UUID.fromString("00002a04-0000-1000-8000-00805f9b34fb");

    public static final UUID BATTERY_LEVEL_CHARACTERISTIC_UUID = \
        UUID.fromString("00002a19-0000-1000-8000-00805f9b34fb");
...
    private BluetoothGattServer mGattServer;
    private BluetoothGattService mDeviceInformationService;
    private BluetoothGattService mBatteryLevelService;
    private BluetoothGattCharacteristic mDeviceNameCharacteristic,
        mModelNumberCharacteristic,
        mSerialNumberCharacteristic,
        mBatteryLevelCharactersitic;

    /**
     * Construct a new Peripheral
     *
     * @param context The Application Context
     * @param blePeripheralCallback The callback handler
     * that interfaces with this Peripheral
     * @throws Exception Exception thrown if Bluetooth is not supported
     */
    public MyBlePeripheral(
        final Context context,
        BlePeripheralCallback blePeripheralCallback) throws Exception
    {
        mBlePeripheralCallback = blePeripheralCallback;
        // make sure Android device supports Bluetooth Low Energy
        if (!context.getPackageManager().hasSystemFeature(
            PackageManager.FEATURE_BLUETOOTH_LE))
        {
            throw new Exception("Bluetooth Not Supported");
        }
        // get a reference to the Bluetooth Manager class,
        // which allows us to talk to talk to the BLE radio
        final BluetoothManager bluetoothManager = (BluetoothManager) \
            context.getSystemService(Context.BLUETOOTH_SERVICE);
        mGattServer = bluetoothManager.openGattServer(
            context,
            mGattServerCallback
        );
        mBluetoothAdapter = bluetoothManager.getAdapter();
        if(!mBluetoothAdapter.isMultipleAdvertisementSupported()) {
            throw new Exception ("Peripheral mode not supported");
        }
        mBluetoothAdvertiser = mBluetoothAdapter.getBluetoothLeAdvertiser();
        // Use this method instead for better support
        if (mBluetoothAdvertiser == null) {
            throw new Exception ("Peripheral mode not supported");
        }
        setupDevice();
    }


    /**
     * Get the battery level
     */
    public int getBatteryLevel() {
        BatteryManager batteryManager = \
            (BatteryManager)mContext.getSystemService(BATTERY_SERVICE);
        return batteryManager.getIntProperty(
            BatteryManager.BATTERY_PROPERTY_CAPACITY
        );
    }

    /**
     * Set up the GATT profile
     */
    private void setupDevice() throws Exception {
        // set the device name
        mBluetoothAdapter.setName(ADVERTISING_NAME);
        // build Characteristics
        mDeviceInformationService = new BluetoothGattService(
            DEVICE_INFORMATION_SERVICE_UUID,
            BluetoothGattService.SERVICE_TYPE_PRIMARY
        );
        mDeviceNameCharacteristic = new BluetoothGattCharacteristic(
            DEVICE_NAME_CHARACTERISTIC_UUID,
            BluetoothGattCharacteristic.PROPERTY_READ,
            BluetoothGattCharacteristic.PERMISSION_READ);
        mModelNumberCharacteristic = new BluetoothGattCharacteristic(
            MODEL_NUMBER_CHARACTERISTIC_UUID,
            BluetoothGattCharacteristic.PROPERTY_READ,
            BluetoothGattCharacteristic.PERMISSION_READ);
        mSerialNumberCharacteristic = new BluetoothGattCharacteristic(
            SERIAL_NUMBER_CHARACTERISTIC_UUID,
            BluetoothGattCharacteristic.PROPERTY_READ,
            BluetoothGattCharacteristic.PERMISSION_READ);
        mBatteryLevelService = new BluetoothGattService(
            BATTERY_LEVEL_SERVICE,
            BluetoothGattService.SERVICE_TYPE_PRIMARY
        );
        mBatteryLevelCharactersitic = new BluetoothGattCharacteristic(
            BATTERY_LEVEL_CHARACTERISTIC_UUID,
            BluetoothGattCharacteristic.PROPERTY_READ,
            BluetoothGattCharacteristic.PERMISSION_READ);

        // Add Characteristics to Services
        mDeviceInformationService.addCharacteristic(
            mDeviceNameCharacteristic
        );
        mDeviceInformationService.addCharacteristic(
            mModelNumberCharacteristic
        );
        mDeviceInformationService.addCharacteristic(
            mSerialNumberCharacteristic
        );
        mBatteryLevelService.addCharacteristic(mBatteryLevelCharactersitic);
        // put in fake values for the Characteristic
        mDeviceNameCharacteristic.setValue(ADVERTISING_NAME.getBytes(CHARSET));
        mModelNumberCharacteristic.setValue(MODEL_NUMBER.getBytes(CHARSET));
        mSerialNumberCharacteristic.setValue(SERIAL_NUMBER.getBytes(CHARSET));
        // add Services to Peripheral
        mGattServer.addService(mDeviceInformationService);
        mGattServer.addService(mBatteryLevelService);
        // update the battery level
        // every BATTERY_STATUS_CHECK_TIME_MS milliseconds
        TimerTask updateBatteryTask = new TimerTask() {
            @Override
            public void run() {
                mBatteryLevelCharactersitic.setValue(
                    getBatteryLevel(),
                    BluetoothGattCharacteristic.FORMAT_UINT8,
                    0
                );
            }
        };
        Timer randomStringTimer = new Timer();
        // schedule the battery update and run it once immediately
        randomStringTimer.schedule(
            updateBatteryTask,
            0,
            BATTERY_STATUS_CHECK_TIME_MS
        );
    }
...
    /**
     * Start Advertising
     *
     * @throws Exception Exception thrown if Bluetooth Peripheral
     *                              mode is not supported
     */
    public void startAdvertising() throws Exception {
        // set the device name
        mBluetoothAdapter.setName(ADVERTISING_NAME);

        // Build Advertise settings with transmission power and advertise speed
        AdvertiseSettings advertiseSettings = new AdvertiseSettings.Builder()
            .setAdvertiseMode(mAdvertisingMode)
            .setTxPowerLevel(mTransmissionPower)
            .setConnectable(true)
            .build();
        AdvertiseData.Builder advertiseBuilder = new AdvertiseData.Builder();
        // set advertising name
        advertiseBuilder.setIncludeDeviceName(true);
        // add Services to Advertising Data
        advertiseBuilder.addServiceUuid(new ParcelUuid(
            DEVICE_INFORMATION_SERVICE_UUID
        ));
        advertiseBuilder.addServiceUuid(new ParcelUuid(BATTERY_LEVEL_SERVICE));
        AdvertiseData advertiseData = advertiseBuilder.build();
        // begin advertising
        mBluetoothAdvertiser.startAdvertising(
            advertiseSettings,
            advertiseData,
            mAdvertiseCallback
        );
    }
...
```

The resulting app will be able to host a minimal GATT Profile (Figure 6-9).



ExampleBLEPeripheral

Advertising Name:MyDevice

Bluetooth On

Advertising

Central Connected

Figure 6-9. App screen showing Services in GATT Profile

Example code

The code for this chapter is available online at: https://github.com/BluetoothLowEnergyInAndroidJava/Chapter06