Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorthesis

Emrah Demir

Applikation zur Verbesserung der Laderegelung in Smartphones

*Fakultät Technik und Informatik*
*Department Informations- und Elektrotechnik*

*Faculty of Engineering and Computer Science*
*Department of Information and Electrical Engineering*

Emrah Demir

# Applikation zur Verbesserung der Laderegelung in Smartphones

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung

im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Robert Heß
Zweitgutachter: Prof. Dr. Heike Neumann

Abgegeben am 14. Januar 2020

**Emrah Demir**

## Thema der Bachelorthesis

Applikation zur Verbesserung der Laderegelung in Smartphones

## Stichworte

Applikation Entwicklung in Java, Zweipunktregelung, Android Studio

## Kurzzusammenfassung

Diese Arbeit umfasst eine Applikation, welche die Akkuladung eines Smartphones mithilfe eines externen Schalters nach Vorgabe des Benutzers zwischen einem maximalen und einem unteren Schaltpunkt hält.

**Emrah Demir**

## Title of the paper

Application to improve the charge control in smartphones

## Keywords

Application development in Java, Two-point control, Android Studio

## Abstract

This work includes an application that keeps the battery charge of a smartphone between a maximum and a lower switching value according to the user's specifications using an external switch.

# Inhaltsverzeichnis

# 1 Einleitung

Die Smartphone Akkus werden typischerweise vollgeladen, um daraus die maximal mögliche Leistung erzielen zu können. Jedoch verkürzt es die Lebensdauer von Lithium-Ionen-Akkus, weil man sie ständig über 90% auflädt [1] [2]. Es gibt zurzeit noch keine Smartphone auf dem Markt, die dem Verbraucher ermöglicht, deren Ladevorgang zu beeinflussen. Man kann zum Beispiel nicht einstellen, dass die Akkuladung bei 90% aufhört. Es finden sich nur sogenannte „Charge Limit‘‘ Anwendungen, welche ohne Zugriff auf das Root des Systems nicht auskommen. Das hat den Nachteil, dass das „Battery Management“ des Systems manipuliert wird. Deshalb lehnen viele Hersteller bei sogenannten gerooteten Smartphones die Garantie ab.

In diesem Projekt wird eine Anwendung entwickelt, die dem Verbraucher ermöglicht, die Akkuladung zwischen einem maximalen und einem unteren Schaltwert einzuhalten. Dazu wird ein Ladegerät gebaut, welches sich per Bluetooth mit der Anwendung verbindet. Das Ladegerät schaltet damit die Stromversorgung von außen ohne Eingriff auf das Root des Systems [1] [2].

Stellt der Benutzer beispielsweise die maximale Akkuladung auf 90% und den unteren Schaltwert auf 3%, hält die Anwendung die Akkuladung mithilfe des Ladegerätes zwischen 90% und 87%.

Das Ladegerät besteht hauptsächlich aus einem Microcontroller und einem Relais. Der Microcontroller bearbeitet Ein-und Ausschaltsignale zur Stromversorgung von der Anwendung. Es ist eine relativ einfache Schaltung, um zu testen, ob die Anwendung funktioniert. In Weiterentwicklung dieses Projektes kann das Ladegerät zum Beispiel modulierte Signale von der Anwendung empfangen und somit den unteren Schaltwert weglassen.

Die Anwendung hat drei Benutzeroberflächen, welche in folgenden Kapiteln bildlich dargestellt sind. An der Hauptoberfläche kann der Verbraucher die gewünschte maximale Ladung auf dem Schieberegler eingeben, Verbindung zum Ladegerät herstellen und damit die Regelung starten. Auf der zweiten Oberfläche sind einige signifikante Eigenschaften des Akkus für den Verbraucher aufgelistet. Die letzte Oberfläche beinhaltet Anwendungseinstellungen. Der Benutzer wählt das Ladegerät aus, kann es umbenennen, den unteren Schaltwert eingeben und die Sprache der Anwendung auf Englisch oder Deutsch umstellen.

Die Hauptinformationsquelle für diese Bachelorarbeit ist die Android Dokumentation auf der Webseite https://developer.android.com/. In den folgenden Kapiteln sind zur Erklärung des Quellcodes nur die relevanten Stellen anstatt der gesamten Anweisung genommen.

# 2 Grundlagen

In diesem Kapitel werden die grundlegenden Eigenschaften einer Applikation erläutert.

## 2.1 API - Level

Die erste Entscheidung, die der Entwickler treffen sollte, ist der minimale API-Level der Anwendung. Was ist ein API-Level? Der API-Level ist ein ganzzahliger Wert, der auf einem Gerät mit Android System die Version eindeutig identifiziert [3]. Der minimale API-Level für diese Anwendung ist 19, der als KitKat bezeichnet wird. Der Grund dafür ist, dass die Anwendung auf einem alten Tablet mit der KitKat-Version getestet wurde. Es ist möglich diese Anwendung für vorherige Android Versionen zu benutzen, jedoch muss sie dementsprechend konfiguriert werden. Die Abbildung 1 zeigt die Liste der Android Versionen ab API Level 19. Diese Anwendung sollte für alle Versionen in der Liste funktionieren, es gibt jedoch Einschränkungen ab API Level 26. Die Anwendung kann im Hintergrund vom System beendet. oder neu gestartet werden, wenn sie viel Akku verbraucht.

| Codename | Version | API level/NDK release |
|---|---|---|
| Android10 | 10 | API level 29 |
| Pie | 9 | API level 28 |
| Oreo | 8.1.0 | API level 27 |
| Oreo | 8.0.0 | API level 26 |
| Nougat | 7.1 | API level 25 |
| Nougat | 7.0 | API level 24 |
| Marshmallow | 6.0 | API level 23 |
| Lollipop | 5.1 | API level 22 |
| Lollipop | 5.0 | API level 21 |
| KitKat | 4.4 - 4.4.4 | API level 19 |

Abbildung 1: Android API – Level [1]

## 2.2 Aktivität Lebenszyklus

Die wichtigste Anforderung an der Anwendung ist, dass die Bluetooth Verbindung immer aktiv sein muss, um die Regelung zu realisieren. Der Benutzer wird durch einen Hinweistext über den Verbindungsstatus informiert, der ständig aktualisiert wird. Ein weiterer Aspekt ist, dass das Layout seine Werte beibehalten muss, wenn der Benutzer zwischen Benutzeroberflächen

[1] *Quelle: https://source.android.com/setup/start/build-numbers*

wechselt oder den Bildschirm des Smartphones umdreht. Es ist daher sehr wichtig, den Lebenszyklus der Aktivität besser kennenzulernen. Die Aktivität ist eine Hauptkomponente des Android Systems. Diese Anwendung beinhaltet drei Aktivitäten jeweils mit einem eigenen Lebenszyklus. Der vereinfachte Lebenszyklus in der Abbildung 2 sieht für diese Anwendung wie folgt aus.

Wenn der Benutzer die Anwendung zum ersten Mal startet, befindet sich die Anwendung in der Hauptaktivität „MainActivity.“ Die „MainActivity“ führt nacheinander die drei Zustände (Methoden) onCreate, onStart und onResume aus. Die Benutzeroberfläche kommt in den Vordergrund, der Benutzer kann mit der Applikation interagieren.

Beispielsweise erbt (extends) MainActivity von der AppCompatActivity-Klasse und überschreibt (Override) die onCreate -Methode, um das Layout zu verwenden.

Der Code wird in der onCreate-Methode initialisiert. Die Variablen sollten in dieser Methode gespeichert werden, damit sie ihre Werte behalten, wenn sich die Zustände ändern. In dieser Anwendung wurden an vielen Stellen statische Variablen verwendet, um dieses Problem zu vermeiden. Dies wird im Kapitel MainActivity ausführlich behandelt.

Wenn der Benutzer zu einer anderen Aktivität wechselt oder den Bildschirm dreht, befindet sich die MainActivity im Zustand „onPause“. In diesem Zustand sind noch alle Speicherinformationen verfügbar. Sie müssen nicht erneut initialisiert werden. Tritt ein Fehler im Zustand „onPause“ auf, wird dieser dann in „onResume“ aktualisiert. Dies heißt, wenn die Bluetooth-Verbindung im Zustand „onPause“ verloren geht, wird der Hinweistext in „onResume“ dementsprechend umgesetzt.

Wenn die Anwendung längere Zeit im Hintergrund bleibt, kann das System die Anwendung in den Zustand „onStop“ versetzen. Dies liegt daran, dass der Benutzer die Anwendung wahrscheinlich nicht mehr verwendet und abbricht. Wenn der Verbraucher zur Hauptbenutzeroberfläche zurückkehrt, wird die MainActivity zuerst in den Zustand „onResume“ versetzt und dann im Vordergrund angezeigt.

In diesem Zustand „onResume“ verlieren die Variablen ihre Zuweisung, wenn sie nicht als statisch definiert wurden. Schwierig ist die Situation, wenn der Benutzer die Applikation in den Hintergrund verschiebt. Ist die Anwendung im Hintergrund und verbraucht dabei viel Strom, wird sie möglicherweise vom System zerstört, es ist dann gleichbedeutend wie ein Neustart der Anwendung.

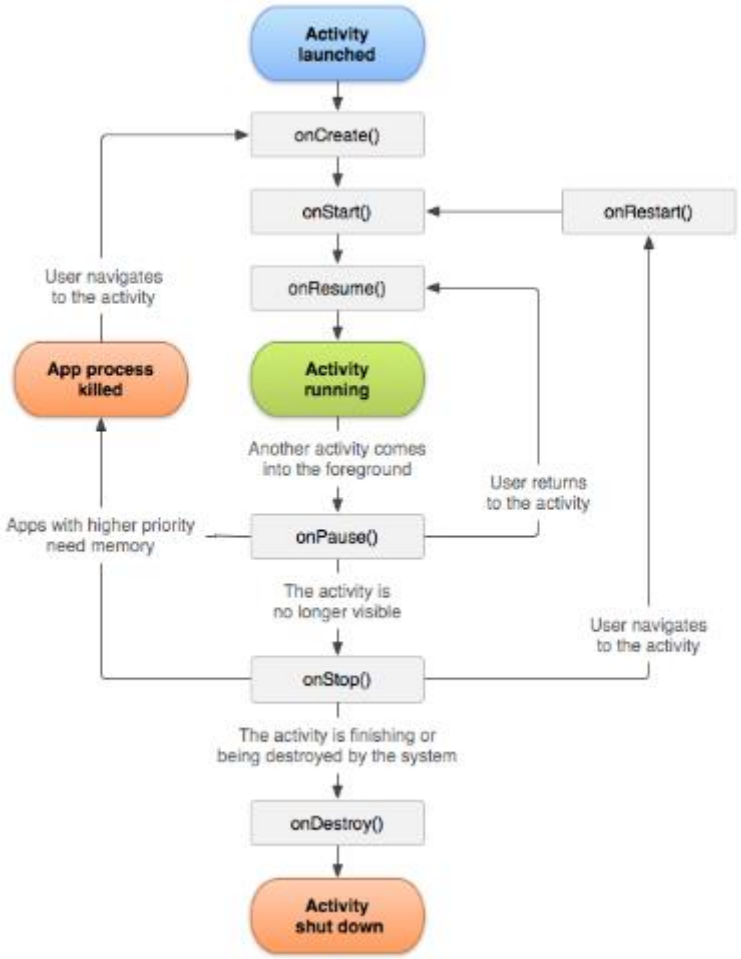



Abbildung 2: Aktivität Lebenszyklus [2]

## 2.3 Manifest der Anwendung

Jede Android Anwendung besitzt die sogenannte Manifest XML Datei [4]. Das Manifest beschreibt grundlegende Eigenschaften der Anwendung. Eins der wichtigen Attribute ist das Package, das zur eindeutigen Identifizierung der Anwendung in Google Play dient [4]. Das Package für diese Anwendung heißt „com.example.smartcharge“ In der Abbildung 3 ist das Manifest der Anwendung zu sehen.

[2] *Quelle: https://developer.android.com/guide/components/activities/activity-lifecycle*

MainActivity.java | AndroidManifest.xml | strings.xml | menu_main.xml | Constants.java | Pro

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smartcharge">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Smart Charge"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".SettingsActivity"
            android:label="Settings"
            android:parentActivityName=".MainActivity">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.smartcharge.MainActivity" />
        </activity>
        <activity android:name=".PropertiesActivity"
            android:parentActivityName=".MainActivity"
            android:label="@string/action_properties"
            />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Abbildung 3: Manifest der Anwendung

- <package>: Das ist der Name des Paketbezeichners, der zur eindeutigen Identifizierung der Anwendung dient [4].
- <uses-permission>: In diesem Block werden Erlaubnisse definiert, welche der Benutzer akzeptieren muss, um die Anwendung benutzen zu können. Diese Anwendung verlangt Zustimmung von BLUETOOH und BLUETOOTH_ADMIN [4].
- <application>: In diesem Block sind allgemeine Informationen über die Anwendung definiert. Die Anwendung erscheint in Google Play unter dem Namen „Smart Charge“. Das Attribut „allowBackup“ sorgt dafür, dass eine Sicherung bzw. Wiederherstellung der Anwendung möglich ist, wenn es auf „true“ gesetzt ist [5].
- <activity>: In diesem Block werden die in Anwendung enthaltene Aktivitäten beschrieben. In der Anwendung sind neben MainActivity noch zwei weitere Aktivitäten für Akkueigenschaften und Einstellungen vorhanden. Das Attribut „parentActivityName“ bedeutet, dass die MainActivity Hauptaktivität ist. Es sorgt auch dafür, wenn der Benutzer in Menü auf Zurückbutton (Pfeil) klickt, wieder auf Hauptoberfläche „Smart Charge“ zurückkehrt.

- <intent-filter>: Es dient in diesem Manifest dazu, dass die „MainActivity“ als Hauptaktivität zu definieren. Die Anwendung startet die MainActivity als erstens. Das Android-Smartphone sendet die Systemnachrichten als Broadcast. Um zu selektieren, welche Nachricht die Anwendung empfangen soll, benutzt man ebenfalls den Intent Filter (siehe Kapitel Implementierung). Es ist ein mächtiges Element in der Applikation Entwicklung, kann auch neben <action>, die Anweisungsblöcke <service> oder <receiver> beinhalten [6].

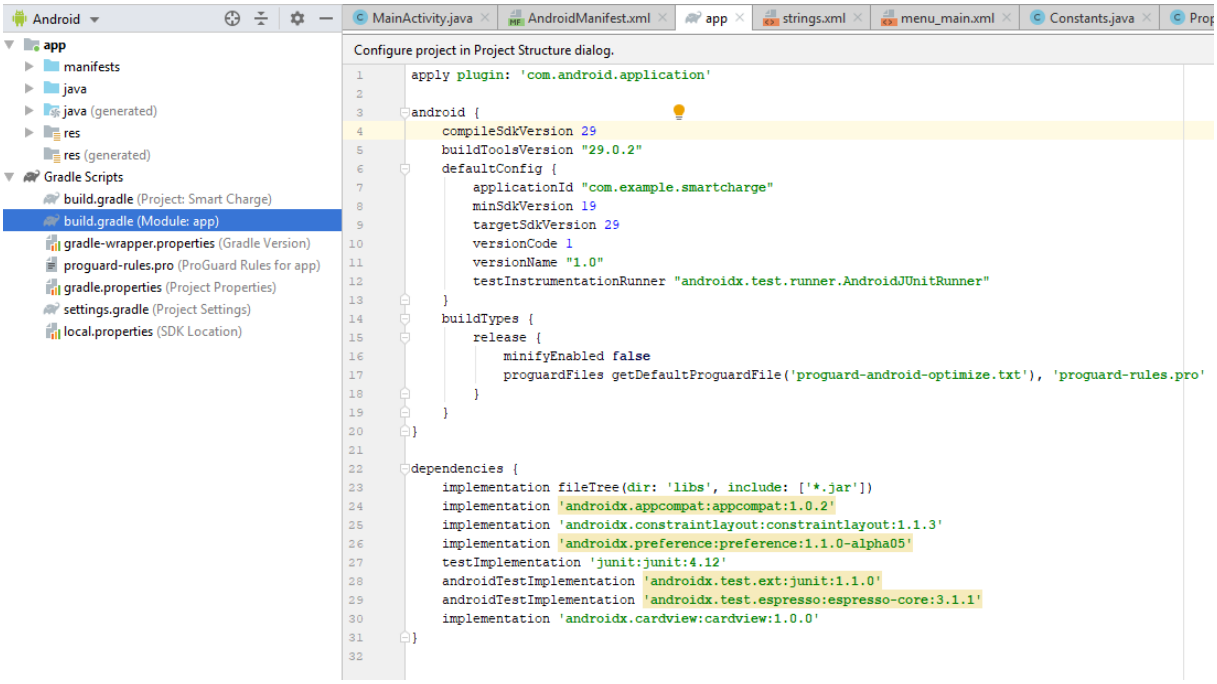



Abbildung 4: build gradle der Anwendung

Eine weitere wichtige Datei ist die „build.gradle“ Im Anweisungsblock „defaultconfig“ stehen grundlegende Eigenschaften der Anwendung. Die folgenden Attribute sind für diese Anwendung sehr wichtig:

- applicationId: Der Wert ist identisch mit dem Package vom Manifest. Das ist die eindeutige Identifizierung dieser Applikation [7].
- VersionCode: Dieses Attribut ist wichtig, wenn man die Applikation auf Google Play veröffentlichen möchte. Eine neuere Version dieser Anwendung kann nur hochgeladen werden, wenn sie einen höheren VersionCode als die alte hat. In diesem Fall, die nächste Variante dieser Anwendung muss einen „VersionCode“ mehr als 1 besitzen, um unter demselben Package auf Google Play veröffentlichen zu können [7].
- minSdkVersion: Hier wird der minimale API-Level (siehe Abbildung 1) der Anwendung definiert [8].
- targetSdkVersion: Das ist der gezielte API-Level, der Level 29 ist momentan die höchste Android-Plattform auf dem Markt.
- implementation: Zwei Implementierungen sind für diese Anwendung zu dem Anweisungsblock „dependencies“ hinzugefügt worden. Die erste lautet

„androidx.preference:preference:1.1.0-alpha05“. Dabei handelt sich um einen von Android Studio zur Verfügung gestellte Implementierung, welche nützliche Objekte für die Erstellung der Benutzeroberfläche Einstellungen beinhaltet. Die zweite ist „androidx.cardview:cardview:1.0.0“. Das ist ein zusätzliches UI-Widget, welches bei der Erstellung einer übersichtlichen Seite für die Akkueigenschaften dient.

## 2.4 BroadcastReceiver

Innerhalb der „MainActivity“ erfüllt die Komponente „BroadcastReceiver“ folgende Aufgaben.

1. Die momentane Akkuladung bestimmen
2. Den momentanen Ladestatus bestimmen
3. Laderegelung für Android Versionen unter dem API-Level 26 (siehe Abbildung 1)

In der „PropertiesActivity“ empfängt die Anwendung folgende Events:

1. Der Akku Typ
2. Akkuqualität
3. Ladestatus
4. Stromquelle
5. Akku Temperatur
6. Akkukapazität
7. Akkuspannung
8. Akkustrom

Deshalb ist es wichtig den Ablauf des Broadcastreceivers zu untersuchen. Der „BroadcastReceiver“ ist eine Hauptkomponente des Android OS [9]. Der Broadcast Receiver (Empfänger) kann sowohl im Quellcode als auch im Manifest der Applikation definiert werden, wobei die Definition im Manifest für höhere Android Versionen eingeschränkt wird [9], weil es die Performance der Applikation im Hintergrund beeinträchtigen könnte. Sie baut die Kommunikation über Intents zwischen der Aktivität und dem Android System.

Alle in dieser Anwendung verwendeten Events(Akkueigenschaften) sind über den Intent „Intent.ACTION_BATTERY_CHANGED“ aufrufbar. Dieser Intent lässt sich ausnahmsweise nicht im Manifest der Anwendung definieren, was einen Vorteil und einen Nachteil hat. Der Vorteil ist, dass der Benutzer keine zusätzlichen Erlaubnisse zustimmen muss. Der Nachteil ist, die Anwendung empfängt im Hintergrund keine Events. Das hat dann große Einwirkung auf die Akkuregelung im Hintergrund, weil es die beiden Informationen, nämlich die aktuelle Akkuladung und der aktuelle Ladezustand, fehlen.

# 3 Anforderungen

Die Anwendung soll folgende Anforderungen erfüllen:

1. **Ein- und Ausschalten der Stromversorgung mithilfe eines externen Schalters.**
   Die Anwendung soll sich per Bluetooth mit dem Ladegerät verbinden können. Das Ladegerät schaltet den Ladestrom für Smartphone entsprechend dem Einschaltsignal oder dem Ausschaltsignal von der Anwendung.
2. **Den Verbindungsstatus anzeigen.**
   Es soll angezeigt werden, ob die Anwendung mit dem Ladegerät verbunden ist oder nicht.
3. **Den Ladezustand anzeigen.**
   Es soll angezeigt werden, ob der Akku geladen oder entladen wird.
4. **Eingabe der maximalen Akkuladung über einen Schieberegler in Prozent.**
5. **Eingabe eines Bezugspunktes bzw. eines unteren Schaltwertes über einen Schieberegler in Prozent.**
6. **Umbenennen des Ladegerätes.**
   Der Benutzer soll den Namen des Ladegerätes innerhalb der Anwendung umbenennen können.
7. **Die aktuelle Akkuladung soll in Form einer Batterie dargestellt werden.**
8. **Die Anwendungssprache kann zwischen Englisch und Deutsch umgestellt werden. Die Standardsprache der Anwendung ist Englisch.**
9. **Grundlegende Informationen über den Akkuzustand für den Benutzer anzeigen.**
   Dies ist eine optionale Anforderung. Der Benutzer kann Informationen wie den Akku Typ (Li-Ion), die Akkuqualität (gut) usw. vom Smartphone auslesen.

Das Hauptziel dieser Anwendung ist die Regelung der Akkuladung nach Vorgabe des Benutzers einzustellen. Es handelt sich dabei um eine Zweipunktregelung. Die Akkuladung ist die Regelgröße(Istwert). Der Benutzer gibt eine maximale Ladung(Sollwert) und einen unteren Schaltpunkt in Prozent ein. Die Differenz zwischen der maximalen Ladung und dem unteren Schaltpunkt bildet den Regelbereich.

Der Akku wird aufgeladen, wenn das Ladegerät mit der Anwendung nicht verbunden ist. Wenn sich die tatsächliche Akkuladung unterhalb des Regelbereiches befindet, schaltet das Ladegerät die Stromversorgung ein, damit sie den Regelbereich erreicht. Innerhalb des Regelbereiches wird die Stromversorgung weder eingeschaltet noch ausgeschaltet. Steigt die tatsächliche Akkuladung über die Maximale, wird die Stromversorgung vom Ladegerät ausgeschaltet. Das Ziel der Regelung ist, die Akkuladung innerhalb des Regelbereiches zu halten.

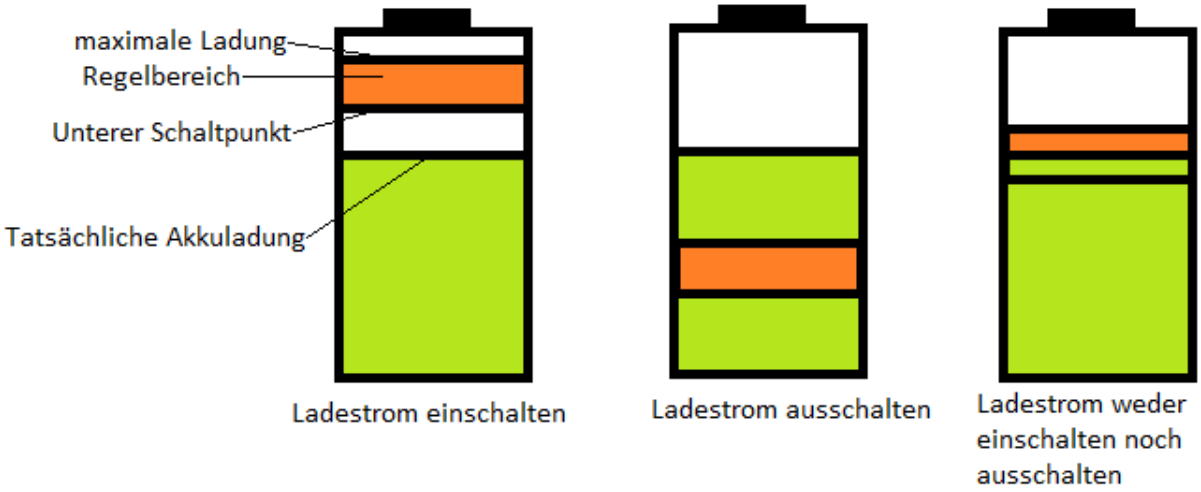



Abbildung 5: Skizze Regelung

# 4 Design und Layout

Das Programm Android Studio bietet die Möglichkeit an, die Benutzeroberflächen im Ordner res/layout in XML (Extensible Markup Language) zu definieren. Es werden eine Art Platzhalter (Views) benutzt, um zum Beispiel ein Text oder ein Bild auf dem Layout zu positionieren. Dafür verwendet man View Objekte. Diese View Objekte können in Java über den Identifikationsnamen instanziiert werden, um den Inhalt zu ändern.

Der erste Schritt ist die Vorbereitung der Ressourcen für die Erstellung von Layouts. Es handelt sich dabei um Texte, Bilder, Farben und Dimensionsgrößen, die man bei der Definition des Layouts in XML benutzt. Um einen Text hinzufügen, öffnet man die strings.xml Datei, welche sich im Ordner res/values befindet. Innerhalb der Anweisung <ressources> werden Texte als Konstanten des Objektes String deklariert. Das hat den Vorteil gegenüber einer direkten Zuweisung, dass man einen Textfehler in dem Quellcode schnell korrigiert. So sieht beispielsweise die Deklaration des Textes „Power Source“ aus.

<string name="strPlug">Power Source</string>.

Bei der Erstellung der Benutzeroberflächen wird auf diesen Namen „strPlug“ zugegriffen. Dieser Zugriff erfolgt über das Schlüsselwort „string“ mit dem Präfix @, dementsprechend ist die Deklaration in XML:

android:text="@string/strPlug".

In Java erfolgt der Zugriff durch den Aufruf der Methode „getString(R.string.strPlug)“ aus der Klasse android.content.res.Resources [10].

Für die Visualisierung des Akkus wurden zehn Bilder in svg (Scalabe Vector Graphics) Format gezeichnet. Sie bieten zwei wichtige Vorteile im Vergleich zum Rasterformat an. Zum einen benötigen sie wenig Speicherplatz, zum anderen kann das Bild ohne Qualitätsverlust skaliert werden. Über das Konfigurationsfenster von Vector Asset werden Bilder in den Ordner „res/drawable“ hinzugefügt. Das ist der Standardordner für die Bilder.

Es gibt weitere Ordner in Resources, weil nicht jedes Smartphone die gleiche Bildschirmgröße hat. Für die Visualisierung der Bilder in unterschiedlichen Bildschirmgrößen ist nicht nur die absolute Pixelauflösung wichtig, sondern auch die Pixeldichte dpi(Dots per Inch) [10]. Deshalb werden die Ordner für die Smartphones kategorisiert. Die Postfixe bedeuten dementsprechend hdpi(High dpi), mdpi(Medium dpi), xhdpi(Extra High dpi) usw. So kann zum Beispiel das Launcher Icon für Smartphones mit höherer Pixeldichte in besserer Bildqualität dargestellt werden.

Der Zugriff auf die Bilder in Java erfolgt durch den Aufruf der Methode „getDrawable“. Zum Beispiel:

getResources().getDrawable(R.drawable.ic_bt_00_00)

Bei der Konfiguration wird die Breite und Höhe eines Bildes in dp (denstiy-independent pixel) eingegeben, welche bei diesen Bildern 160dp x 184dp ist. Die Maßeinheit dp (density-independent pixel) wird normalerweise für alle Größeneinträge wie Abstand, Breite und Höhe einer View in XML verwendet, mit Ausnahme von String [10]. Die Textgröße wird meistens in der Maßeinheit sp(scale independent pixel) definiert [10]. Die Maßeinheit dp entspricht einem Pixel bei einem mdpi Smartphone mit 160dpi. Das Ziel ist eine Größenangabe wie der Abstand eines Bildes zu einem Text unabhängig von der Pixeldichte zu definieren. Damit ist der Abstand für Smartphones mit gleicher Bildschirmdiagonale und unterschiedlicher Pixeldichte gleich.

Auf gleiche Weise wird eine Farbe in der Datei colors.xml in RGB Format definiert, dabei stehen die Buchstabe für die Farben R Rot, G Gün und B Blau. Die Deklaration in XML sieht wie folgt aus.

<color name = "colorIcon">#19aeff</color>

Die Farbe wird hexadezimal über dem Präfix # codiert. Jede Farbkomponente kann einen dezimalen Wert zwischen 0 und 255 oder äquivalent einen hexadezimalen Wert zwischen 00 und FF haben. Der Zugriff in XML erfolgt wieder auf den Namen „colorIcon“. In Java kann der Zugriff wieder durch den Aufruf der Methode „getResources()“ erfolgen. Im Quellcode wurde alternativ die Color-Klasse verwendet, um die Farbe des Buttons durch Zuweisen von Color.YELLOW in Gelb zu ändern.

Die XML Sprache erlaubt eine Dimensionsgröße auf folgender Weise zu deklarieren.

<dimen name="title_size">220dp</dimen>

Der Aufruf in XML erfolgt auf gleiche Weise wie bei „string“ Konstanten. Die Maßeinheit wird hinter dem ganzzahligen Wert geschrieben.

## 4.1 Benutzeroberfläche Smart Charge

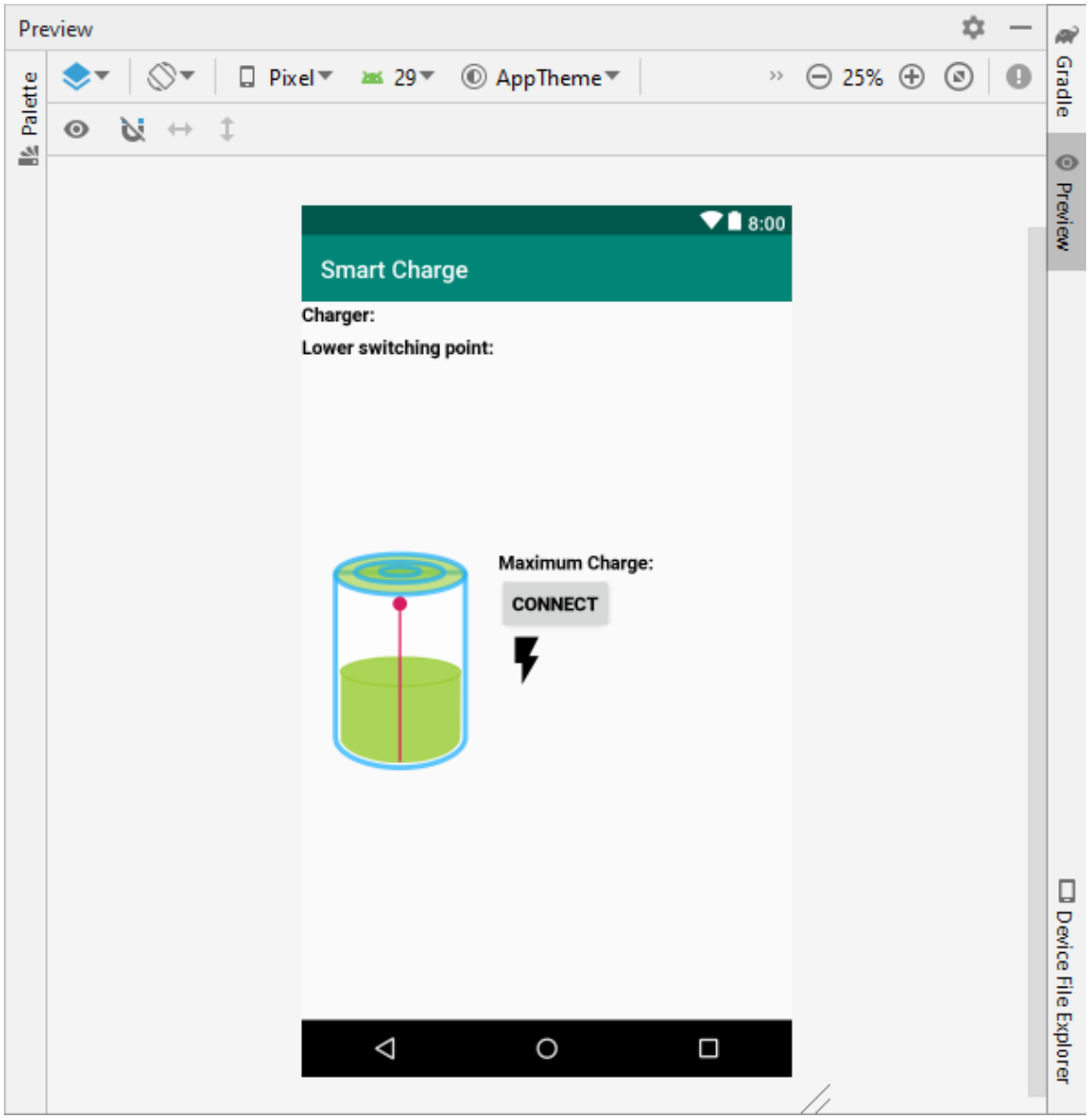



Abbildung 6: Benutzeroberfläche Smart Charge designen

Die Hauptoberfläche der Anwendung soll wie in Abbildung 6 dargestellt, erscheinen. Auf diesem Fenster kann der Benutzer die maximale Akkuladung über den Schieberegler einstellen.

Der Button „CONNECT“ soll bewirken, dass sich die Anwendung per Bluetooth mit dem Ladegerät verbindet und die Regelung startet, wenn die Verbindung erfolgreich ist. Der Benutzer wird anhand des Hinweistextes „VERBUNDEN MIT DEM LADEGERÄT“ oder „VERBINDUNG MIT DEM LADEGERÄT FEHLGESCHLAGEN“ über den Verbindungsstatus informiert. Der Benutzer soll die Anwendung auch ohne Regelung benutzen können, in diesem Fall erscheint kein Hinweistext.

Die Akkuladung wird in Form eines Batteriesymbols dargestellt. Rechts neben dem Akkusymbol soll die eingestellte maximale Ladung in Prozent stehen. Es soll ein dynamischer Text sein, der Wert ändert sich, sobald der Benutzer auf den Schieberegler neuen Zielwert eingibt. Anhand des Blitz Icons soll der Benutzer erkennen, ob sich der Akku auflädt.

Der Benutzer wird über den Regelungsstatus wieder anhand eines Hinweistextes informiert. Im Normalfall soll entweder „eingeschaltet“, wenn das Ladegerät die Stromversorgung einschaltet oder „ausgeschaltet“, wenn die Stromversorgung von Ladegerät abgebrochen wird, erscheinen. Tritt ein Fehler während der Regelung auf, wie zum Beispiel Strom fließt, während der Regelbereich sich unterhalb der aktuellen Akkuladung befindet, erscheint der Hinweis „Überprüfen Sie, ob das Ladegerät richtig angeschlossen ist.“.

Der Name des Ladegeräts und der untere Schaltpunkt in Prozent soll oben links stehen. Der Regelbereich befindet sich zwischen der maximalen Ladung und dem unteren Schaltwert. Über dem Optionsmenü (Dreipunkte) wechselt der Benutzer auf die Benutzeroberfläche Eigenschaften oder Einstellungen. Das Informationsicon im Menü soll eine Anleitung über die Bedienung der Anwendung öffnen.

Die Benutzeroberfläche „Smart Charge“ befindet sich in der activity_main.xml Datei im Anhang. Sie ist im „RelativeLayout“ deklariert worden. Das „RelativeLayout“ positioniert seine Platzhalter (child views) relativ zueinander. Das hat den Vorteil gegenüber einer exakten Definition, dass die Platzhalter automatisch an die Bildschirmgröße angepasst werden. Somit sieht die Benutzeroberfläche bei unterschiedlichen Bildschirmgrößen gleich aus.

Bei der Zuweisung der Breite und Höhe einer View wird meistens die Befehle „wrap_content" und „match_parent“ verwendet. Das Befehl „wrap_content“ beschreibt, dass die View nur so viel Platz auf dem Layout annimmt, wie er benötigt“. Mit dem Befehl "match_parent" nimmt die View die gesamte Länge des Layouts ein, je nachdem, ob es sich um die Breite oder die Höhe handelt. Man kann sich dabei auf die Gesamtoberfläche (Parent) beziehen oder auf eine andere View. Das Akkubild ist über die Schlüsselwörter „centerInParent“ und „alignParentLeft“ in der Mitte links der Parent in einem „ImageView“ positioniert worden. Die anderen Views beziehen sich über die Schlüsselwörter „above“, „below“, „toLeftOf“ und „toRightOf“. auf den Akku. Benutzerdefinierte Abstände werden durch den Verweis „margin“ festgelegt.

## 4.2 Benutzeroberfläche Akkueigenschaften

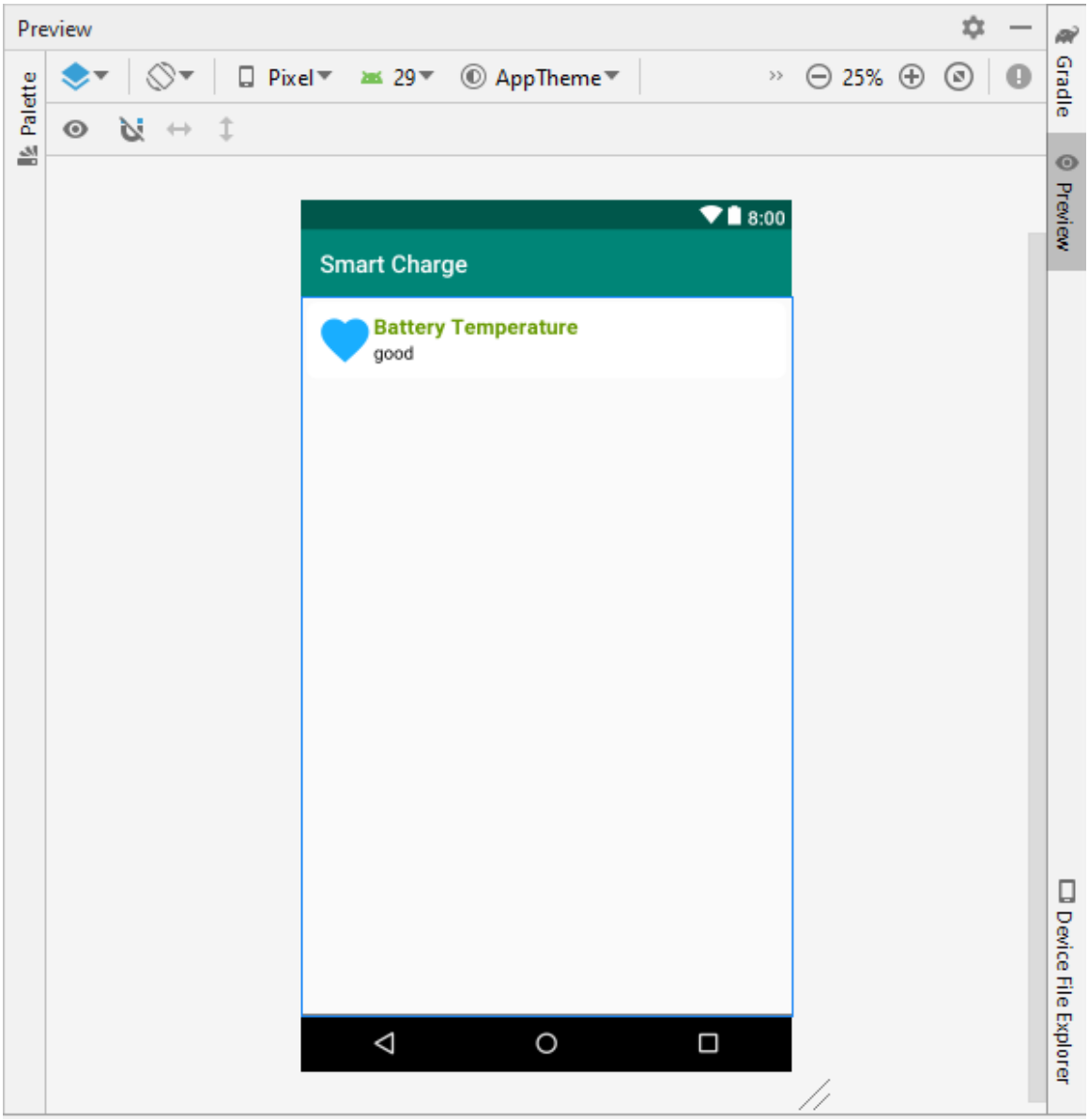



Abbildung 7: Benutzeroberfläche Akkueigenschaften designen

Zweites Fenster soll wichtige Informationen über den Akku für den Verbraucher beinhalten. Es handelt sich dabei um eine benutzerdefinierte Liste in Kombination mit dem UI (User Interface) Widget „CardView“. Jedes Listenelement besitzt einen Titel, eine Beschreibung des Titels und ein entsprechendes Icon. Die Informationen werden aus der Klasse „BatteryManager“ des Smartphones gelesen [11]. Die benutzerdefinierte Liste soll die folgenden Akkueigenschaften anzeigen.

1. Akku Typ
2. Akkuqualität
3. Stromquelle
4. Ladestatus
5. Akku Temperatur

6. Akku Spannung
7. Kapazität
8. Akku Strom

Erster Eintrag soll der Akku Typ sein, welches in meisten Fällen Li-Ion ist. Dann kommt die Akkuqualität. Die Beschreibung „gut“ würde den normalen Zustand bezeichnen. Die Akkuqualität kann neben „gut“ noch die Zustände kalt, überhitzt, Überspannung, tot, unbekannt und „nicht spezifizierter Ausfall“ [11] annehmen.

Die Stromquelle kann vier Werte annehmen, diese sind „AC“, „USB“, „WIRELESS“ und „nicht angeschlossen“ [11].

Das nächste Listenelement informiert über den Ladestatus. Die Akkutemperatur ist ein ganzzahliger Wert in Grad Celsius. Die Akkuspannung liegt in der Regel bei Li-Ion um die 3,7V Bereich, in der Liste soll sie in mV angezeigt werden. Aus dem Smartphone wird sie in µV abgelesen und in mV umgewandelt.

Das letzte Listenelement ist der Akkustrom. Der Akkustrom ist positiv, wenn der Akku aufgeladen wird, ansonsten ist es negativ.

Für die Erstellung der Benutzeroberfläche Eigenschaften sind zwei Layouts deklariert worden. Das erste Layout „activity_properties.xml“ beinhaltet nur den Container ListView. Es handelt sich dabei um eine scrollbare Liste. Die Listenelemente sind nach dem gleichen Schema untereinander aufgebaut. Die Attribute in der Datei „activity_properties.xml" definieren, dass die Liste am oberen Rand des Layouts beginnt und sich über die gesamte Benutzeroberfläche erstreckt.

Der Standard (Default) „ListView“ kann nur ein String platzieren. In diesem Fall besitzt jedes Element zwei Strings und ein Icon. Wie sie auf dem Listenelement platziert werden sollen, wird in dem anderen Layout „custom_listview.xml“ beschrieben. Der UI-Widget „TextView“ ist Platzhalter für Strings und der „ImageView“ für das Icon.

## 4.3 Benutzeroberfläche Einstellungen

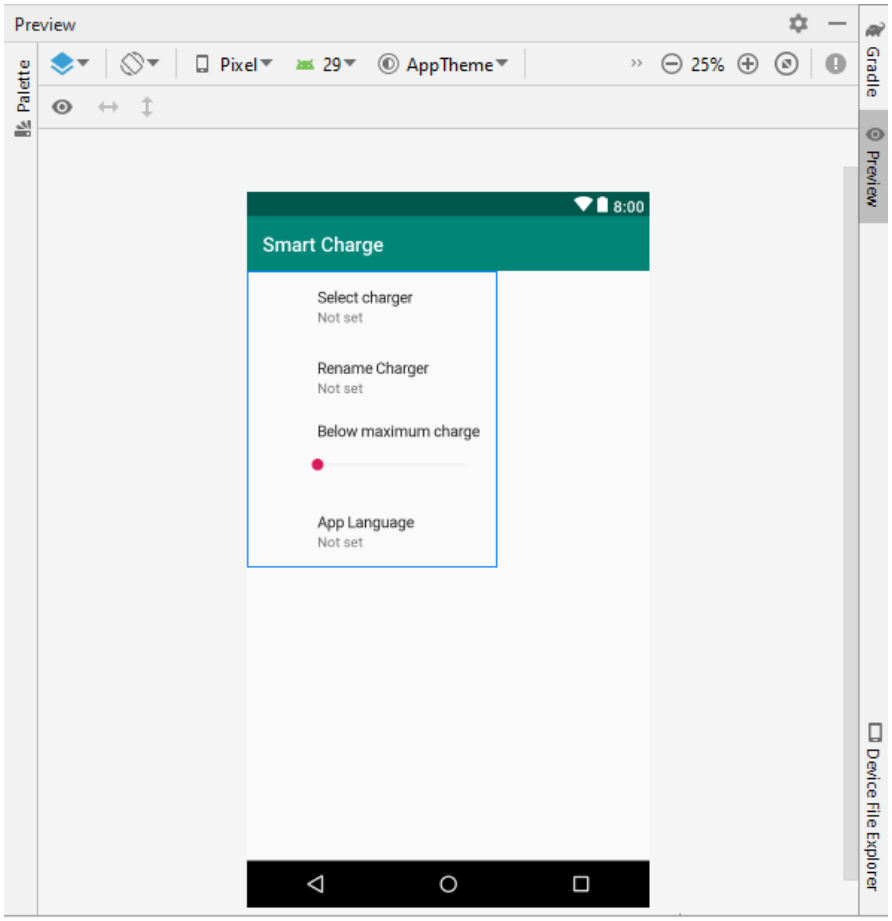



Abbildung 8: Benutzeroberfläche Einstellungen designen

Der Benutzer soll das Ladegerät auswählen und umbenennen können. Der untere Schaltpunkt ist zwischen 2% und 8% einstellbar. Die Sprache der Anwendung ist zwischen Englisch und Deutsch auswählbar. Wenn der Benutzer auf „Ladegerät auswählen" klickt, sollte sich automatisch eine dynamische Liste der bereits gekoppelten Bluetooth-Geräte in einem Dialogfenster öffnen. Dieses Fenster würde sich automatisch an die Bildschirmgröße anpassen. In der dynamischen Liste würden sich alle gepaarten Bluetooth-Geräte befinden. Der Benutzer wählt mit einem Klick das Ladegerät aus. Das ausgewählte Ladegerät soll für die nächste Benutzung dieser Anwendung gespeichert werden. Der Name des Ladegerätes soll gleichzeitig unterhalb des Titels „Ladegerät auswählen“ erscheinen. Dieses Textfeld wird auch Summary genannt. Das Summary soll für alle Optionen auf der Seite Einstellungen angezeigt werden. Falls das Ladegerät nicht in der Liste ist, bedeutet, dass das Ladegerät in Bluetooth Einstellungen nicht gepaart ist.

Das Ladegerät soll innerhalb der Anwendung umbenannt werden, dazu klickt man auf die Option „Ladegerät umbenennen“. Es soll sich wieder ein Dialogfeld mit dem gespeicherten Namen des Ladegerätes öffnen. In das Textfeld kann der Benutzer einen beliebigen Namen eingeben und mit „OK“ bestätigen. Der geänderte Name erscheint dann sofort auf Summary bzw. unterhalb der Option „Ladegerät umbenennen“. Der Benutzer soll jederzeit den Namen

wieder ändern können. Der Name bleibt für die Wiederbenutzung dieser Applikation gespeichert. Der geänderte Name würde solange gespeichert bleiben, bis der Benutzer sich für ein anderes Ladegerät entschieden hat.

Der Benutzer soll die Sprache der Anwendung auf Englisch oder Deutsch umschalten können. Die ausgewählte Sprache wird auch gespeichert. Die Anwendung soll beim nächsten Start in der ausgewählten Sprache ausgeführt werden. Wenn der Benutzer die Anwendung zum ersten Mal öffnet, wird die Anwendung in Englisch erscheinen.

Die Einstellung der Benutzeroberfläche ist nicht in einem Standardlayout wie das „RelativeLayout“ deklariert worden, sondern in einem „PreferenceScreen“. Das „PreferenceScreen“ wird automatisch von Android Studio beim Auswählen einer „Settings Activity“ durch Hinzufügen einer Implementation erstellt(siehe Abbildung 4).

## 4.4 Menü

Die Benutzeroberfläche der Seite „Smart Charge“ soll ein sogenanntes Optionsmenü beinhalten. Dazu wurde zunächst ein Ordner in Resources erstellt. Das Menü wird ebenfalls wie die Layouts in XML geschrieben. Die folgende Abbildung 9 zeigt die XML Datei von Menü Main.

Jedes Menü Element muss zumindest eine Identifikation(id) besitzen. Anhand der Identifikation wird auf das Element in der Aktivität zugegriffen. Das Attribut Titel beschreibt den Namen des Menüeintrages, der Benutzer sieht diesen Namen, wenn er das Menü öffnet.

Das Attribut „showAsAction“ definiert, wie der Menüeintrag im „AppBar“ bzw. oben rechts neben dem Aktivitätsnamen stehen soll. Das Schlüsselwort „never“ heißt, dass der Eintrag für den Benutzer nur sichtbar ist, wenn er auf die drei Punkte klickt. Das andere Schlüsselwort „ifRoom“, macht den Eintrag Info sichtbar, wenn der Benutzer die Seite „Smart Charge“ öffnet. Für jede Aktivität ist ein Menü erstellt worden.

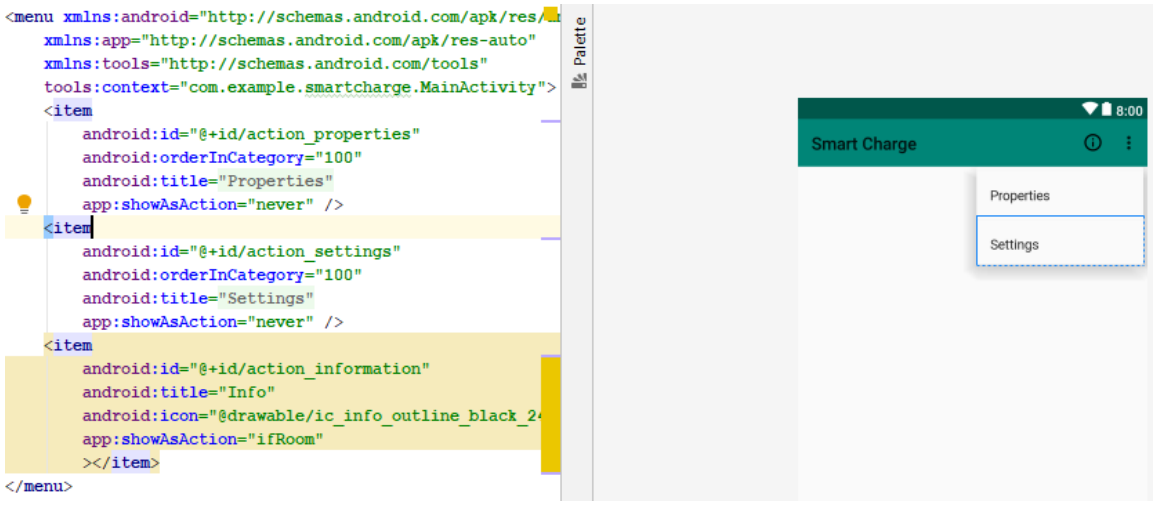



Abbildung 9: Menu Main

# 5 Entwurf des Ladegerätes

In diesem Kapitel wird der Entwurf des Ladegerätes erklärt. Da das Ladegerät nur zu Testzwecken erstellt wurde und daher nicht im Mittelpunkt dieser Arbeit steht, wird nicht tiefgehend behandelt. Es geht bei der Schaltung des Ladegerätes darum, die stromführenden 5V Pin, über das Relais zu steuern. Das Ladegerät kann man sich wie ein Verlängerungskabel vorstellen. Die Eingangsseite ist ein Mini-USB Adapter, welches der Benutzer an seinem gewöhnlichen Ladegerät anschließen kann. Die Ausgangsseite ist dann wieder eine Mini-USB Eingangsbuchse. (siehe Abbildung 12) . In der Schaltung sind folgende Bauteile verwendet worden:

1. Arduino Nano
2. Bluetooth Modul HC-05
3. Printrelais 5V/DC 36.11.9.005.4011
4. Widerstand 1kOhm
5. Transistor NPN BC547B
6. Diode 1N4148
7. MINI-USB-A Adapter
8. MINI-USB-A Einbaubuchse
9. Lochrasterplatine

Alle Adern der Eingangsseite außer den VBUS wird wieder am Ausgang verwendet. Nur VBUS und GND Pins sind in der Schaltung verwendbar. Der VBUS sorgt für die 5V Betriebsspannung für Arduino und Printrelais.

Das Bluetooth-Modul wird über 3,3V Ausgang des Micro Controllers mit Strom versorgt. Das Printrelais ist ein Wechsler. Der VBUS ist gleichzeitig über den COM Eingang des Relais über den NO Kontakt mit dem Ausgang verbunden, sodass im Ruhezustand Strom fließen darf.

Die Ankerspannung des Relais wird in Treiberschaltung von einem Ausgangspin des Microcontrollers steuert. Die Treiberschaltung setzt sich aus dem Widerstand, Transistor, Diode und Relais zusammen. Der Bemessungsstrom des Relais ist 72mA [12], welcher der Arduino Ausgangspin auch zur Verfügung stellen kann. Der Arbeitsbereich des Relais ist laut Datenblatt zwischen 3,7 V und 6,5 V [12].

Das Relais würde auch ohne die Treiberschaltung funktionieren, jedoch erledigt die Treiberschaltung zwei wichtige Aufgaben. Die erste Aufgabe ist, das Relais über den Transistor mit genügend Strom zu versorgen, damit es nicht permanent im Grenzbereich arbeitet. Die zweite Aufgabe ist, durch die Selbstinduktion verursachte Spannung zum Schutz des Relais über die Diode abzubauen.

Beim Hochladen des Quellcodes auf den Arduino Nano sollte die Versorgungsspannung des Bluetooth Moduls nicht angeschlossen sein, weil es sonst ein Fehler auftritt. Die Abbildung 10 zeigt die Farben der Adern eines Standard USB Adapter.

**Standardstecker A / B**

| Pin | Signalname | Adernfarbe | Beschreibung |
|---|---|---|---|
| Gehäuse | Schirm | n.a. | Schirmgeflecht |
| 1 | VBUS | Rot | +5 V |
| 2 | D− | Weiß | Daten USB 2.0, differentielles Paar −/+ |
| 3 | D+ | Grün | |
| 4 | GND | Schwarz | Masse |

Abbildung 10 USB Adern Farbe[3]

Die Abbildung 11 ist zeigt die Schaltung des Ladegerätes im Prinzip an. Der Microcontroller wird über den VIN mit Strom versorgt. Das Bluetooth Modul HC-05 braucht 3,3V Gleichspannung, deshalb wird es an 3V3 Pin des Microcontrollers angeschlossen. Der RX-Pin des Moduls ist mit TX und der TX des Moduls mit RX des Microcontrollers verbunden. Der Ausgangspin D2 steuert das Relais.

Das Relais hat 5 Kontaktanschlüsse. Sie sind gekennzeichnet mit A1 und A2 für Ankerspannung, 11 für den gemeinsamen Anschluss COM ,14 für den NC (Normally closed) Kontakt und 12 für NO (Normally Open) Kontakt. Der Schließerkontakt 14 ist in dieser Schaltung nicht erforderlich, weil im Ruhezustand Strom fließen soll.

Der Kollektorausgang des Transistors ist mit A1, der A2 und COM mit dem VBUS des USB-Anschlusses verbunden. Die USB-Buchse ist mit dem Öffnerkontakt 12 des Relais verbunden

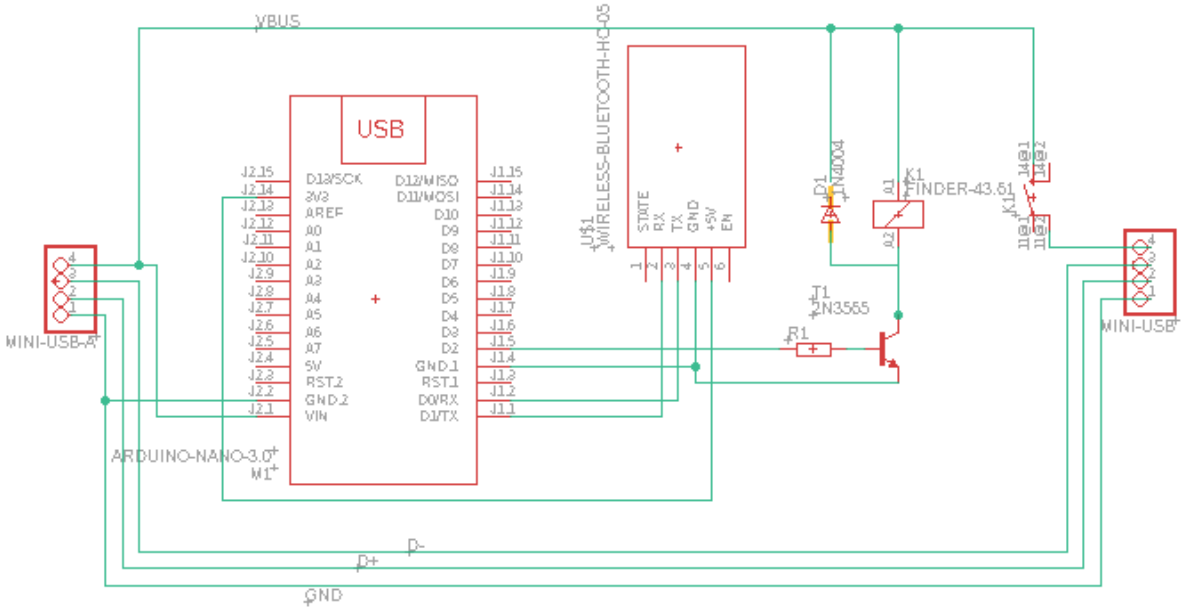



Abbildung 11: Ladegerät Schaltung

[3] *Quelle: https://de.wikipedia.org/wiki/Universal_Serial_Bus#Pinouts_und_Adernfarben*

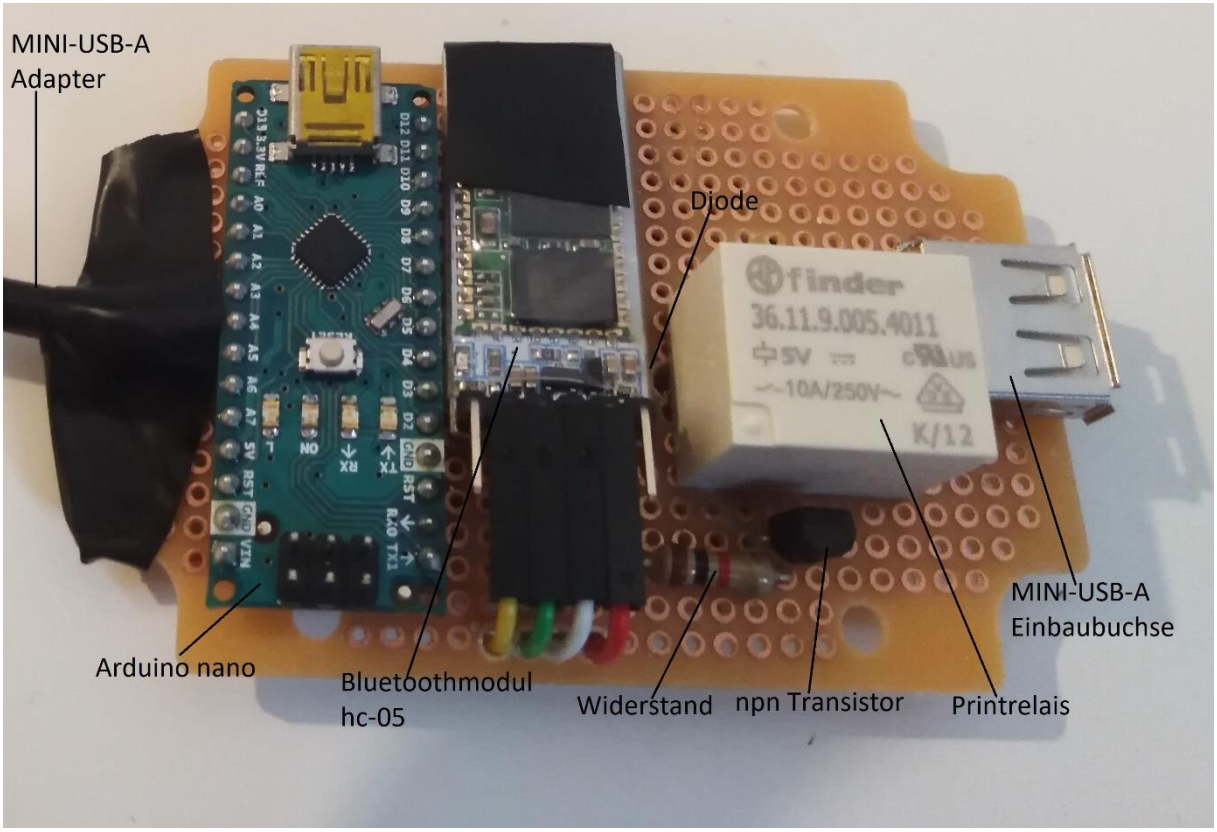



Abbildung 12: Ladegerät gelötet auf Lochrasterplatine

# 6 Implementierung

In diesem Kapitel wird zunächst schrittweise das Starten des Projektes anhand von Bildern(Screenshots) erklärt. Anschließend werden die Aktivitäten und deren Aufgaben in der Anwendung näher beschrieben. Das Projekt wurde in Android Studio Version 3.5 entwickelt. Es ist folgendermaßen entwickelt worden.

Zuerst wird die MainActivity hinzugefügt und entsprechend konfiguriert. Bei der Konfiguration wird der Anwendungsname, Paket, die Programmiersprache und der minimale API-Level definiert. Anschließend wird die „PropertiesActivity“ hinzugefügt.

Die „PropertiesActivity“ bekommt die MainActivity als Hauptfenster zugewiesen, diese Zuweisung wird im Manifest der Anwendung übernommen. Der Benutzer kann somit über den Zurück-Pfeil auf das Hauptfenster „Smart Charge“ zurückkehren.

Als letztes wird für Anwendungseinstellungen die „SettingsActivity“ hinzugefügt. Die „SettingsActivity“ bekommt ebenfalls die MainActivity als Hauptfenster zugewiesen. Beim Hinzufügen von „SettingsActivity“ wird automatisch ein Fragment für das Managen der Anwendungseinstellungen erstellt. Danach wird für die in Anwendung verwendeten Konstanten eine Klasse mit dem Namen „Constants“ initialisiert. Zum Schluss bekommt das Projekt ein „DialogFragment“ zur Erstellung des Anleitungsfensters.

In der zweiten Phase des Projektes geht es um die Vorbereitung der Ressourcen. Zunächst werden die Zeichenkonstanten „Strings“ in Englisch definiert. Danach öffnet man den Translation Editor. Unter dem Menüpunkt „Show All Keys“, wählt man aus der Liste „German(de) in Germany(DE)“. Es öffnet sich automatisch ein Fenster mit leeren Kästchen, welche die entsprechende deutsche Übersetzung des Strings erwartet. Dann werden alle Zeichenketten bis auf Anwendungsname ins Deutsche übersetzt. Im Ressourcenordner erscheint dann automatisch eine weitere „strings.xml“ mit deutscher Übersetzung.

Im Unterordner Values wird eine dimen.xml Datei erstellt, indem man Konstanten für Pixelgrößen definieren kann. Anschließend werden Bilder und Icons dem Projekt hinzugefügt. Alle in diesem Projekt verwendeten Icons sind bereits in Android Studio verfügbare Material Icons. Es sind zehn Bilder für die Darstellung des Akkus hinzugefügt worden. Die Bilder wurden im SVG-Format erstellt. Es ist eine skalierbare Pixelgröße, die ihre Helligkeit bei unterschiedlichen Bildschirmgrößen nicht verliert.

Um ein Icon zum Projekt hinzuzufügen, geht man auf den Ordner „drawable“ in Ressource und klickt auf die rechte Maustaste und wählt aus der Liste die Option „Vector Asset“ aus. Das Fenster „Vector Asset“ wird automatisch geöffnet. Der Button „Clip Art“ öffnet eine Liste von Icons, welche in Android Studio dem Entwickler frei zur Verfügung stehen. Das ausgewählte Icon wird bei der Originalgröße 24dp x 24dp belassen. Über den „next“ Button wird das Icon in den Ordner „drawable“ kopiert. Das Hinzufügen von Akkubilder erfolgt in ähnlicher Weise. Es wird die „local svg“ Kästchen anstelle „Clip Art“ markiert.

Die Farbe der Akkubilder und Icons konnten nicht mithilfe einer Farbe Konstante „<color>“ realisieren werden, weil im API-Level 19 diese Funktion nicht unterstützt wird.

Der nächste Schritt ist die Erstellung eines Menüordners in Ressource, in dem sich die Dateien menu_main.xml, menu_properties.xml und menu_settings.xml befinden. Als letztens werden die Layout XML Dateien für die Benutzeroberflächen erstellt. Die Layouts sind im Anhang zu finden.

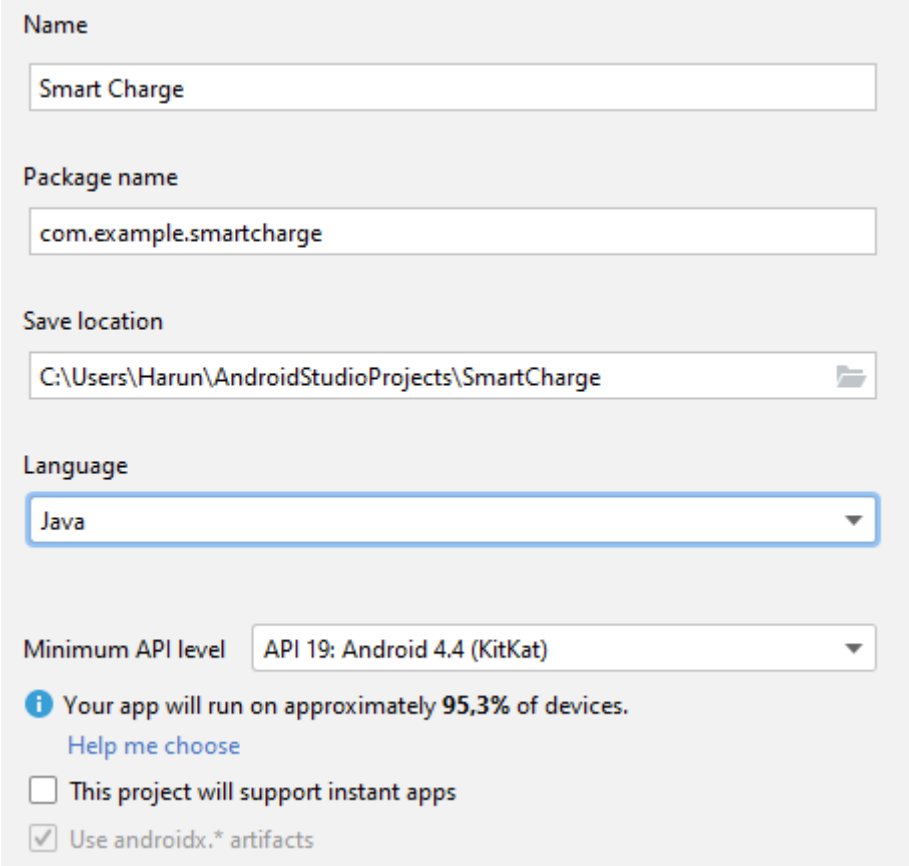



Abbildung 13: Konfiguration Main Aktivität

Der Name des Projektes ist „Smart Charge“, diese Anwendung ist unter diesem Namen auf dem Smartphone bekannt. Das nächste Attribut der Anwendung ist „Package name“, diese Anwendung wird unter dem Namen „com.example.smartcharge“ in Android System und in Google Play identifiziert. Diese beiden Attribute werden automatisch auf das Manifest der Anwendung übernommen. Dann kommt der Speicherort des Projektordners im Computer. Das Attribut „language“ setzt die Programmiersprache ein. Diese Anwendung ist in der Programmiersprache Java entwickelt worden. Das Minimum API Level der Applikation ist 19 Version KitKat. Der Entwickler wird auch während des Programmierens darauf hingewiesen, wenn er eine Methode verwendet, welche in diesem API-Level nicht enthalten ist.

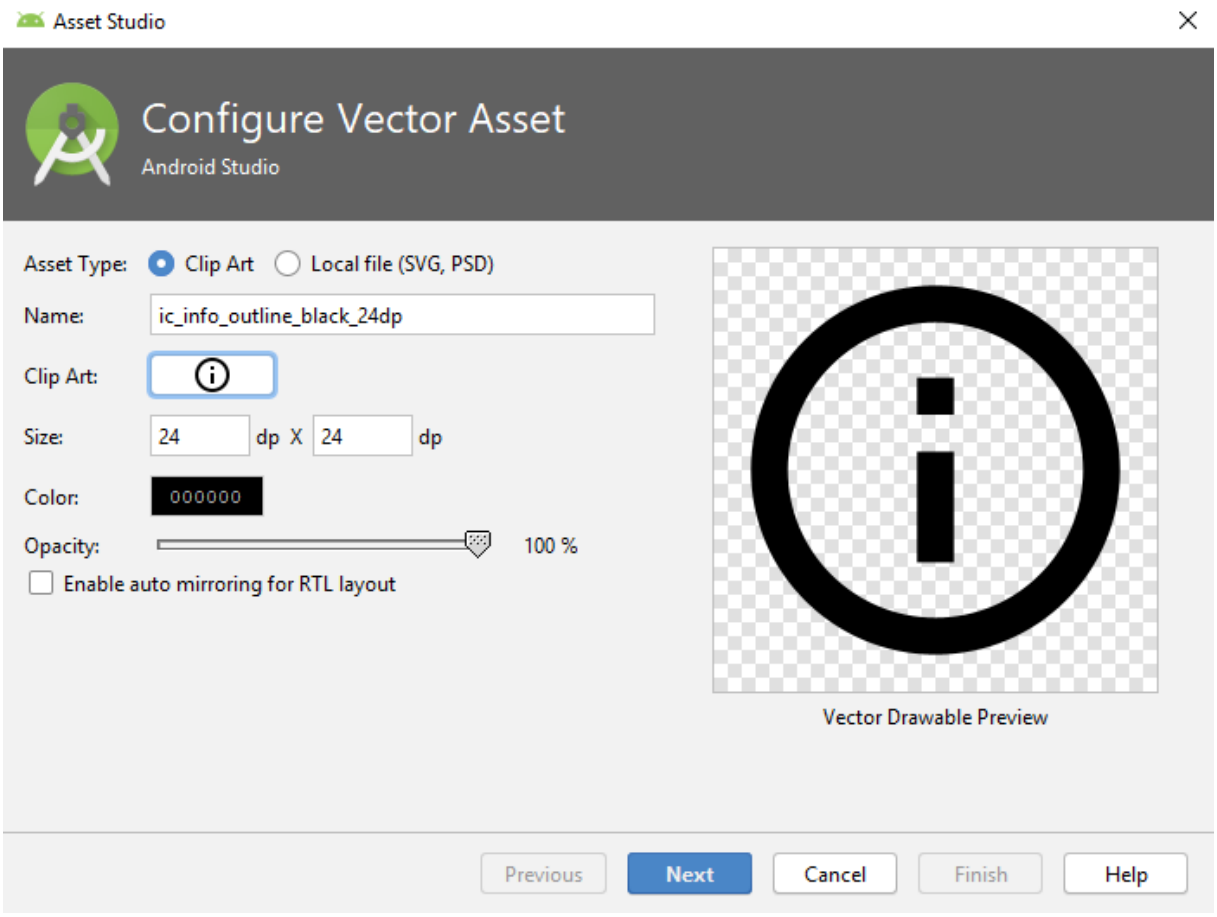



Abbildung 14: Konfiguration Vector Asset

Die Abbildung 14 zeigt das Fenster „Vector Asset“. In diesem Fenster kann man dem Projekt ein Icon oder ein Bild hinzufügen. Bei der Option „Clip Art“ handelt es sich um die in Android Studio verfügbaren Material Icons.

Die andere Option „Local file(SVG, PSD)“ im Attribut „Asset Type“ wird verwendet, um die zuvor in einem SVG-Editor erstellten Akkubilder zum Ordner „drawable“ hinzuzufügen. Der Name, die Farbe, die Pixelgröße und die Helligkeit des Icons und des Bildes können eingestellt werden.

## 6.1 MainActivity

Die Klasse beinhaltet drei Threads, zwei Handler, einen „BroadcastReceiver“ und UI-Widgets. Für die Bluetooth Verbindung werden zwei Threads verwendet den „ConnectThread“ und den „ConnectedThread“. Innerhalb von „ConnectThread“ wird es verbunden. Der „ConnectedThread“ wird zum Verwalten der Kommunikation verwendet.

Der erste Handler „handlerConnectionState“ liest den Verbindungsstatus aus diesen beiden Threads. Mit dem dritten Thread „SampleCapacityThread“ wird die tatsächliche Akkuladung ermittelt und alle drei Sekunden gemessen.

Der zweite Handler „handlerControlCapacity“ holt die aktuelle Akkuladung aus dem „SampleCapacityThread“ und regelt sie. Der „BroadcastReceiver“ dient zur Ermittlung des

aktuellen Ladezustands und der Akkuladung. Die in XML erstellte Benutzeroberfläche „Smart Charge“ wird durch Aufrufen der Methode „setContentView(R.layout.activity_main)“ eingesetzt. Der Zugriff auf ein View Objekte in der layout.activity_main.xml Datei erflogt folgender Weise:

1. private static TextView tvConnectionState
2. tvConnectionState = findViewById(R.id.tvConnected)
3. tvConnectionState.setText(R.string.strConnectedTo)

Es wird eine statische Variable deklariert, damit sie ihre Speicherinformation in den unterschiedlichen Zuständen der Aktivität (siehe Abbildung 2) beibehält, beispielsweise beim Umdrehen des Bildschirmes. Sie wird instanziiert, indem die Methode „findViewById“ mit dem Identifikationsnamen „tvConnected“ aufgerufen wird, welcher in der Klasse R mit einem ganzzahligen Wert gespeichert ist.

Die Methode „setText“ setzt die String-Konstante „strConnectedTo“ aus der strings.xml Datei auf der Benutzeroberfläche ein.

### 6.1.1 Visualisierung der Akkuladung

Die volle Akkuladung wird auf zehn Bereiche skaliert, indem man sich zehn Bilder für den entsprechenden zehnprozentigen Bereich erstellt. Empfangen der aktuellen Akkuladung aus dem Android System passiert über die abstrakte Klasse „BroadcastReceiver“. Die Registrierung des Broadcastreceivers in dieser Anwendung sieht folgendermaßen aus.

1. private BroadcastReceiver receiverBatteryState =new BroadcastReceiver()
2. public void onReceive(Context context, Intent intent)
3. registerReceiver(receiverBatteryState,new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
4. unregisterReceiver(receiverBatteryState);

Innerhalb der MainActivity wird eine Instanz der Klasse „BroadcastReceiver“ erstellt. In der „onReceive“-Methode gelangt man an die Informationen über den Parameter „Intent intent“. Die „onReceive“- Methode ist an den Lebenszyklus der Aktivität gebunden [9].

Dies bedeutet, dass der Receiver beim Start der Anwendung mit einem geeigneten „IntentFilter“ registriert werden muss. Diese Registrierung muss wieder entfernt werden, wenn die Anwendung zerstört wird.

Die Registrierung geschieht durch den Aufruf der Methode „registerReceiver“ im Zustand onCreate im Lebenszyklus der Aktivität. Die Registrierung des Broadcastreceivers wird in der Methode „onDestroy“ durch Aufrufen der Methode „unregisterReceiver“ wieder abgebrochen. Das ist ein wichtiges Merkmal von BroadcastReceiver, da die Anwendung ohne Registrierung nicht über den aktuellen Status informiert wird, selbst im Vordergrund kann das System den aktuellen Ladestatus nicht lesen.

Diese Akkubilder werden in einem Array gespeichert. Mithilfe einer Methode, welcher man als Parameter die tatsächliche Akkuladung übergibt, wird das Bild für den entsprechenden zehnprozentigen Ladungsbereich ermittelt. Die Initialisierung findet im „BroadcastReceiver“ wie folgt statt.

1. public static final int[] intSymbolArray = {R.drawable.ic_bt_00_00,.,.R.drawable.ic_bt_80_80
2. ivBattery.setImageDrawable(getResources().getDrawable(intSymbolArray[Constants.levelArea(currentBatteryLevel)]))

Die Bilder werden im „drawable“-Ordner mit ganzzahligen (Integer) Werten identifiziert. Zugriff auf die Bilder in diesem Ordner erfolgt durch den Aufruf der Methode „getResources().getDrawable“.

Beispiel: Die momentane Akkuladung beträgt 80%. Die Methode „LevelArea“ empfängt als Parameter den ganzzahligen Wert 80 und gibt 8 als Rückgabewert zurück, welcher dem Arrayindex 9 entspricht. An der neunten Stelle ist das Bild für diesen Bereich im Integer Array „intSymbolArray“ gespeichert. Das Bild wird zu der Instanz „ivBattery“ zugewiesen.

### 6.1.2 Eingabe der maximalen Ladung

Hierfür wurde das Widget-SeekBar verwendet, welches nicht weiter als ein Schieberegler ist. Der Schieberegler ist genau in der Mitte des Akkusymbols positioniert worden, damit der Benutzer sich bildlich darstellen kann, wo ungefähr der Regelbereich ist. Der „SeekBar“ wird angesprochen, in dem man die implementierten Methoden überschreibt. In der Methode wird der aktuelle Wert gespeichert und zeitgleich als Information in „TextView“ über „tvSetPoint“ ausgegeben. Die Werte werden in der Klasse „SharedPreferences“ gespeichert. Diese Klasse benutzt Schlüsselpaare, ein Schlüssel für den Oberbegriff und weitere für die gespeicherten Variablen. Die Methode „showNameRelative“ sorgt dafür, dass der Wert des Schiebereglers beim Neustart der Anwendung an der richtigen Position ist. Diese Methode setzt den Schieberegler und die maximale Ladung auf die zuvor eingestellten Benutzereingaben. Die Implementierung sieht wie folgt aus.

1. prefSetPoint = getSharedPreferences(strSetPointExtra, Context.MODE_PRIVATE);
   editorSetPoint = prefSetPoint.edit();
2. public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)
3. editorSetPoint.putInt(strSetPoint,progress);
   tvSetPointValue.setText(Integer.toString(progress)+"%");
   int intLowerSwitchingPoint = progress - intRelativePoint;
   tvRelPoint.setText(getString(R.string.strRelPoint)+"
   "+intLowerSwitchingPoint+"%");
   editorSetPoint.apply();

Der Oberschlüssel für die maximale Ladung ist die String Konstante „strSetPointExtra“. In der implementierten Methode „onProgressChanged“ kann man auf Änderung des Schieberegler bzw. der maximalen Ladung in Echtzeit reagieren.

Das Parameter „int Progress“ ist der Rückgabewert des Schiebereglers. Durch Aufrufen der „putInt“ Methode, wird das Integer „progress“ mit dem String „strSetPoint“ gespeichert. Dann wird der untere Schaltpunkt berechnet, welches der Differenz „progress - intRelativePoint“ entspricht. Der untere Schaltpunkt wird dann zu der Variable „tvRelPoint“ zugewiesen.

Durch den Aufruf der Methode „apply()“ bleibt die Eingabe, die maximale Akkuladung, auch dann erhalten, wenn die Anwendung auf dem Smartphone unterbrochen wird. Dies hat den Vorteil, dass die Einstellungen des Benutzers beim Öffnen der Anwendung wieder verfügbar sind.

### 6.1.3 Feststellung des Ladezustands

Den aktuellen Ladezustand erfährt man über den „BroadcastReceiver“ unter der „onRecevie“-Methode. In dieser Anwendung wurde die Beispielimplementierung aus der Android-Dokumentation verwendet [13]. Der aktuelle Ladezustand wird wie folgt empfangen:

1. int intStatus = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
2. isCharging = intStatus == BatteryManager.BATTERY_STATUS_CHARGING || intStatus == BatteryManager.BATTERY_STATUS_FULL;

Die Integer „intStatus“ nimmt einen ganzzahligen Wert an, der dem aktuellen Ladestatus entspricht. Es wird durch den Vergleichsoperator festgestellt, zu welchem Ladezustand es gehört. Die boolesche Variable „isCharging“ bekommt den aktuellen Ladezustand zugewiesen.

### 6.1.4 Anwendung per Bluetooth mit dem Ladegerät verbinden

Damit das Smartphone und das Ladegerät sich per Bluetooth verbinden können, muss sich eine Komponente als Server und die andere als Client implementieren [14]. Der Client kontaktiert den Server, indem er eine Anforderung zum Herstellen einer Verbindung sendet. Der Server akzeptiert die kommende Verbindungsanforderung. Beide haben das Ziel, ein Bluetooth Socket zu erstellen, um gegenseitig Daten zu übertragen. Das Bluetooth Modul HC-05 im Ladegerät hat eine installierte Server-Implementierung. Diese Anwendung wurde dann als Client implementiert.

Die beiden Threads ConnectThread und ConnectedThread sind Beispielimplementierungen aus der Android-Dokumentation [14].

Die Verbindung zum Ladegerät über Bluetooth ist wie folgt:

1. bluetoothDevice=bluetoothAdapter.getRemoteDevice(mac_address);
2. connectThread= new ConnectThread(bluetoothDevice);
   connectThread.start();

3. private final BluetoothSocket mmSocket;
   BluetoothSocket tmp = null;
4. tmp = device.createRfcommSocketToServiceRecord(uuidIdentifier);
5. mmSocket.connect();

Wenn das Smartphone Bluetooth nicht unterstützt, kann es keine Verbindung zum Ladegerät herstellen. Dann muss das Bluetooth aktiviert sein.

Zum Herstellen der Verbindung ist zunächst eine Instanz des BluetoothDevice-Objekts erforderlich. Um ein „BluetoothDevice“ Objekt zu instanziieren, muss die MAC-Adresse des Ladegerätes bekannt sein. Wählt der Verbraucher in Anwendungseinstellungen sein Ladegerät, wird die Mac-Adresse des Ladegerätes in Preference gespeichert und er muss nicht bei jedem Start der Anwendung das Ladegerät nochmal auswählen.

Der „ConnectThread“ ist eine Client-Implementierung. Dem Konstruktor übergibt man als Parameter die Instanz „bluetoohDevice“ des Objektes BluetoothDevice. Durch den Aufruf der Methode „createRfcommSocketToServiceRecord(uuidIdentifier)“ wird zunächst ein temporäres BluetoothSocket-Objekt „tmp“ innerhalb der Klasse „ConnectThread“ erstellt. Die verwendete UUID muss auf beiden Komponenten dieselbe sein. Die eingegebene UUID ist universell für Arduino Microcontrollers.

Im letzten Schritt schließt sich der Socket durch den Aufruf der Methode „connect()“ in der Methode „run()“ an Bluetooth Modul des Ladegerätes. Wenn es dabei keinen Fehler auftritt, können beide Komponenten gegenseitig auf diesem Socket Daten austauschen.

Aus diesem Thread empfängt man mithilfe des Handler-Objektes „handlerConnectionState“ den Verbindungsstatus „VERBUNDEN ZUM LADEGERÄT‘‘ oder „VERBINDUNG ZUM LADEGERÄT FEHLGESCHLAGEN‘‘.

Wozu wird der Handler benötigt? Das Android System erlaubt keine direkte Zuweisung eines TextView-Objektes in einem im Hintergrund laufenden Thread [15]. Das TextView-Objekt wird verwendet, um Hinweistexte wie der Verbindungsstatus für den Benutzer auszugeben. Die MainActivity stellt den Hauptthread (UI - Thread) dar. Die anderen Threads in MainActivity laufen im Hintergrund asynchron zueinander [15]. Die Kommunikation zwischen dem UI-Thread und den anderen erfolgt über Nachrichten(Messages) mit Handler. Das folgende Beispiel zeigt wie der Verbindungsstatus „VERBUNDEN“ im „ConnectThread“ abgelesen wird.

1. Message messageConnected = Message.obtain();
2. messageConnected.what = intStateConnected;
3. handlerConnectionState.sendMessage(messageConnected);

Zunächst wird ein Message Objekt erstellt.

Die Nachricht wird am UI-Thread wie folgt behandelt:

```
1. Handler handlerConnectionState=new Handler(new Handler.Callback() {
2. public boolean handleMessage(@NonNull Message msg) {
3. switch (msg.what){
   case intStateConnected:
   intState = intStateConnected;
   strTvState = strConnected;
   tvConnectionState.setText(R.string.strConnectedTo);
   btStart.setVisibility(View.INVISIBLE);
```

Innerhalb der Methode „handleMessage“ wird die Nachricht verarbeitet. Die Nachricht über den Verbindungsstatus ist im Parameter „Message msg“ enthalten, welche man durch Vergleich in einer Switch Anweisung abliest. Der Verbindungsstatus wird aktualisiert, indem der Variablen „intState“ der aktuelle Status zugewiesen wird.

Dann wird der Zustand der Variable „strTvState“ aktualisiert. Das ist eine zusätzliche Variable, welche nur drei Zustande haben kann, verbunden, Verbindung fehlgeschlagen oder Initialisierungsphase. Diese Variable wird in den Methoden „onResume“, „sendMessage“ und im Handler der Regelschleife zur eindeutigen Identifizierung des Verbindungszustandes benutzt.

In der onResume-Methode wird der Verbindungszustand mithilfe der strTvState-Variablen aktualisiert, bevor die Anwendung im Vordergrund angezeigt wird. Sie wird im Handler des Regelkreises zum Starten oder zum Beenden der Regelung verwendet. In der Methode „sendMessage“ verhindert sie, einen Schreibfehler zu begehen.

Der Button wird für den Benutzer unsichtbar, wenn die Verbindung zum Ladegerät erfolgreich ist. Wenn die Verbindung fehlgeschlagen ist, wird der Button wieder sichtbar und hat eine gelbe Hintergrundfarbe.

Der „ConnectedThread“ verwaltet den Datenaustausch zwischen der Anwendung und dem Ladegerät per Bluetooth. Dem Konstruktor übergibt man als Parameter die Instanz vom Objekt „BluetoothSocket“, welche innerhalb des „ConnectThread“ initialisiert worden ist.

Anhand des Sockets werden für den Datenaustauch „InputStream“ und „OutputStream“ erstellt. Der „InputStream“ wird durch den Aufruf der Methode „getInputStream“ und der „OutputStream“ durch „getOutputStream“ initialisiert.

```
1. InputStream tmpIn = null
   OutputStream tmpOut = null;
2. tmpIn = socket.getInputStream();
   tmpOut = socket.getOutputStream();
```

Dieser Thread bewältigt zwei wichtige Aufgaben Das Auslesen der Dateien aus dem „InputStream“ und das Einschreiben der Dateien in den „OutputStream“ durch den Aufruf der Mehtode „write(byte[] bytes)“.

Die vom Ladegerät gesendete Nachricht wird von der Anwendung wie folgt gelesen und verarbeitet.

1. mmBuffer = new byte[1024];
   int numBytes;
2. numBytes = mmInStream.read(mmBuffer);
3. Message messageRead =
   handlerConnectionState.obtainMessage(intStateMessageRead,numBytes, -
   1,mmBuffer);
   messageRead.sendToTarget();

```
final Message          obtainMessage(int what, int arg1, int arg2, Object obj)
```

Abbildung 15: obtainMessage[4]

Die Nachricht wird in Bytes gelesen. Dazu wird das Byte Array „mmBuffer“ in der Klasse „ConnectedThread“ initialisiert. Der Integer Variable „numBytes“ wird Anzahl der Bytes zugewiesen, sie dient dazu ein String Array der Größe „numBytes“ zu definieren. Der Handler versendet diese Nachricht ebenfalls in Bytes an die Main UI-Thread. Der erste Parameter „what“ bezeichnet den aktuellen Verbindungsstatus in Integer, der zweite Parameter „arg1“ beinhaltet den Wert der Variable „numBytes“ bzw. die Anzahl der eingelesenen Bytes, dem dritten Parameter „arg2“ wird keine Variable im „ConnectThread“ zugewiesen, deshalb bekommt es einfach den Wert -1 und der letzte Parameter ist das Byte Array.

Die durch Aufrufen der Methode „obtainMessage“ gesendete Nachricht, wird im Hauptthread (UI-Thread) folgendermaßen verarbeitet.

1. case intStateMessageRead:
2. byte[] readBuff= (byte[]) msg.obj;
3. String strTemp= new String(readBuff,0,msg.arg1);
4. strReadMessage = strTemp;

Es geht um die Konvertierung der gesendeten Nachricht von einem Byte Array in eine Zeichenkette (String). Zunächst wird wieder ein Byte Array „readBuff“ durch den Parameter „obj“ erstellt. Danach wird das erstellte Byte Array in eine Zeichenkette über den Konstruktor „String(readBuff,0,msg.arg1)“ in die temporäre Zeichenkette „strTemp“ konvertiert.

Der erste Parameter ist das Byte-Array „readBuff“, der zweite der Offset und der letzte der Wert der Variablen „numBytes“, die über den Parameter „arg1“ gesendet wird. Anschließend wird der temporäre String zu einem globalen zugewiesen, um auf die einkommende Nachricht überall in der MainActivity zuzugreifen.

[4] *Quelle:https://developer.android.com/reference/android/os/Handler*

Aus dem „ConnectedThread“ wird durch den Handler folgende Zustände abgelesen „READ“, „WRITE“, „READ_ERROR“ und „WRITE_ERROR“.

### 6.1.5 Akkuladung regeln

Um mit der Regelung zu beginnen, muss die aktuelle Akkuladung(Istwert), die gewünschte maximale Ladung(Sollwert) und der untere Schaltpunkt bekannt sein, welche von Preferencen wie folgt abgelesen wird:

1. prefSetPoint = getSharedPreferences(strSetPointExtra, Context.MODE_PRIVATE);
   int intSetPoint=prefSetPoint.getInt(strSetPoint,90);
2. prefRelativePoint= getSharedPreferences(strRelativeExtra,
   Context.MODE_PRIVATE);
   int intRelativePoint=prefRelativePoint.getInt(strRelativePoint,3);

Der Regelkreis entspricht einer einfachen Zweipunktregelung. Die Regelgröße ist die tatsächliche Akkuladung. Der Sollwert ist die maximale Ladung, die der Verbraucher eingibt. Die Stellgröße ist das Signal per Bluetooth an das Ladegerät. Das Relais ist das Stellglied. Die Anwendung misst die Akkuladung vergleicht sie mit der Benutzereingabe und berechnet daraus die Regelabweichung. Die Implementierung des Regelkreises sieht in MainActivity folgendermaßen aus:

1. if (isCharging && currentBatteryLevel >= intSetPoint)
   sendMessage(strStopCharging);
2. if (!isCharging && currentBatteryLevel <= intSetPoint - intRelativePoint)
   sendMessage(strStartCharging);

Die Anwendung sendet das Signal zum Ausschalten der Stromversorgung an das Ladegerät, wenn sich die tatsächliche Akkuladung oberhalb des Regelbereiches befindet und der Akku geladen wird. Die Absicht dahinter ist, dass der Akku sich entlädt bis der Sollwert erreicht wird. Das kann viel Zeit in Anspruch nehmen, wenn die Abstände zu groß gewählt sind.

Das Signal zum Einschalten der Stromversorgung wird gesendet, wenn sich die tatsächliche Akkuladung unterhalb des Regelbereiches befindet und der Akku entladen wird bzw. nicht an eine Stromquelle angeschlossen ist. Befindet sich die Akkuladung im Regelbereich wird der Ladestrom weder eingeschaltet noch ausgeschaltet.

Der Ladestatus „isCharging“ verhindert auch, dass ständig Ausschaltsignal an das Ladegerät gesendet wird, welches die Verbindung belastet.

Empfängt das Ladegerät das Signal zum Einschalten „strStartCharging“, schaltet den Ladestrom mithilfe des Relais aus und sendet ein Signal bzw. die Zeichenkette „X“ an die Anwendung zurück. Die Anwendung überprüft, ob das Signal „X“ angekommen ist.

Wenn es danach der Akku nicht aufgeladen wird, deutet es auf einen Fehler am Ladegerät. Der Benutzer wird darüber anhand eines Hinweistextes informiert. Das gleiche Verfahren passiert

auch beim Ausschaltsignal „strStopCharging“, in diesem Fall sendet das Ladegerät die Zeichenkette „Y“ zurück.

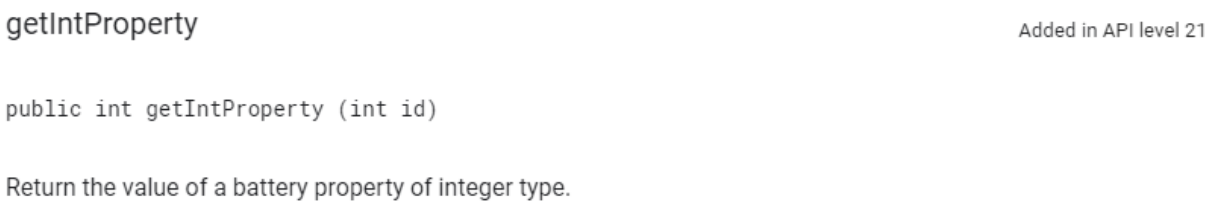
Die entsprechende Codierung für das Einschaltsignal sieht folgendermaßen aus:

```
1. if(strReadMessage.equals("X")){
   if(!isCharging){
   tvCheckAvailable.setText(R.string.strAvailableError);}else{
   tvCheckAvailable.setText(R.string.strTurnedOn);
```

Diese Regelschleife wurde einmal im Handler „handlerControlCapacity“ und in der Methode „onReceive“ des Broadcastreceivers implementiert.

Man kann zwar die aktuelle Akkuladung in der Methode „onReceive“ abfragen, jedoch nur, wenn sich die Anwendung im Vordergrund befindet. Dies bedeutet, dass die Informationen verloren gehen, sobald die Anwendung in den Hintergrund geht. Ein weiterer Aspekt bei der Verwendung des Handlers ist, dass die Aktualisierung der tatsächlichen Akkuladung dem „BroadcastReceiver“ überlassen bleibt. Daher musste eine Lösung gefunden werden, um die Akkuladung unabhängig vom Aktivitätslebenszyklus zu erfassen.

Der Thread „RegulateBatteryCapacity“ tastet unabhängig vom Lebenszyklus der Aktivität alle drei Sekunden die Akkuladung ab und sendet den Wert an die Handler „handlerControlCapacity“. Die boolesche Variable „blStopThread“ startet den Thread, wenn das Ladegerät per Bluetooth mit der Anwendung verbunden ist, ansonsten stoppt er. Die Methode „getIntProperty“ zum Lesen der aktuellen Akkuladung funktioniert jedoch nur ab API-Level 21.


getIntProperty

Added in API level 21

```
public int getIntProperty (int id)
```

Return the value of a battery property of integer type.

Abbildung 16: Methode getIntProperty [5]

Deshalb vergleicht man zuerst die Versionen, bevor es mit der Abtastung der Ladung beginnt.

```
1. if (Build.VERSION.SDK_INT < Build.VERSION_CODES.LOLLIPOP)
```

Den aktuellen Wert liest man sich mithilfe des Message Objektes. Die Information beinhaltet das letzte Parameter der Methode „obtainMessage“, dem übergibt man den Wert der aktuellen Akkuladung.

### 6.1.6 Starten der Verbindung und Regelung

Der Button „VERBINDE“ bewirkt, dass sich die Anwendung per Bluetooth mit dem Ladegerät verbindet und startet die Regelung. Die Implementierung findet in der Methode

[5] *Quelle https://developer.android.com/reference/android/os/BatteryManager.html#getIntProperty(int)*

„startConnection“ statt. Die Abbildung 18 zeigt das entsprechende Flussdiagramm der Methode „startConnection“ an. Die Anfangsphase ist, wenn der Benutzer die Anwendung neu startet. Mit der Betätigung des Buttons „VERBINDE“ wird die boolesche Variable „blStopThread“ auf „false“ gesetzt. Das heißt, die Regelung startet, wenn die Anwendung mit dem Ladegerät verbunden ist. Sie dient zum Pausieren des „sampleCapacityThread“, wenn es bei der Bluetooth Verbindung ein Fehler auftritt.

Sie ist als volatile Boolean deklariert worden. Durch das Schlüsselwort „volatile“ wird sichergestellt, dass der aktuelle Wert aus dem Hauptspeicher und nicht aus dem Cache gelesen wird [16].

Wenn das Smartphone kein Bluetooth unterstützt, wird eine Fehlermeldung „strSupportBlt“ ausgegeben. Die Anwendung kann somit die Akkuladung nicht regulieren. Das Smartphone unterstützt jedoch Bluetooth, es wird geprüft, ob es aktiviert ist. Wenn es nicht aktiviert ist, bittet die Anwendung den Benutzer um die Berechtigung, es zu aktivieren.


Abbildung 17: Erlaubnis zum Aktivieren

Wenn es aktiviert ist, wird überprüft, ob es eine Mac-Adresse gespeichert ist. Das ist der Fall, wenn der Benutzer die Anwendung zum ersten Mal startet. Wenn das Lagegerät nicht bereits verbunden ist, wird versucht eine Verbindung mit dem Ladegerät per Bluetooth aufzubauen und startet die Regelung über den „sampleCapacityThread“.

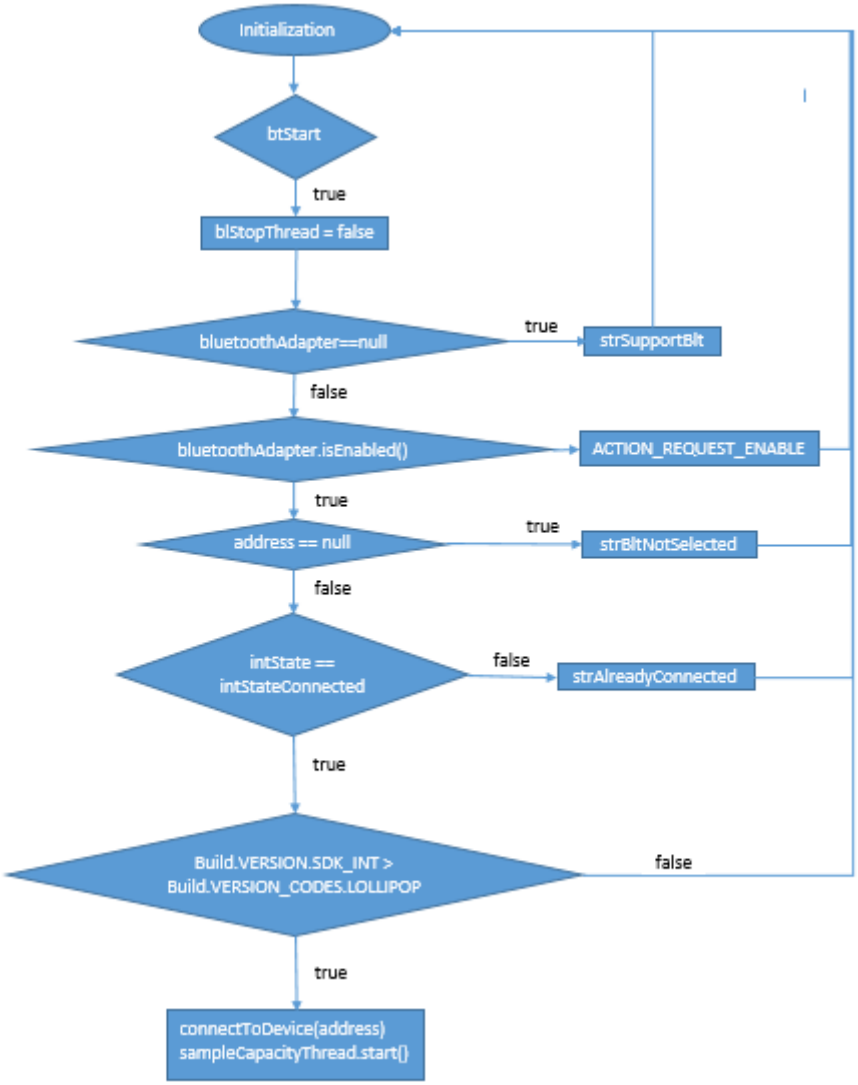



Abbildung 18: Flussdiagramm zum Starten der Bluetooth Verbindung

## 6.2 PropertiesActivity

Innerhalb dieser Aktivität wird eine benutzerdefinierte Liste erstellt. Die Liste beinhaltet einige signifikante Eigenschaften über den Akku. Um die Daten zu empfangen, registriert man wieder einen Broadcastreceiver(Empfänger) über den Intent.ACTION_BATTERY_CHANGED. Die Eigenschaften werden in einem Container Widget „ListView“ (Komponente für Benutzeroberfläche) aufgelistet. Um die Eigenschaften bzw. Daten zu diesem Container hinzufügen braucht man einen sogenannten „Adapter“. Der „normale“ ListView kann nur einen String jeweils für ein Listenelement ausgeben. In der „ProspertiesActivity“ wird der „ListView“ erweitert, in dem man für jedes Element zusätzlich ein Icon und eine Beschreibung hinzufügt. Es muss auch ein passendes Layout erstellt werden, um zu definieren wie der Titel, die Beschreibung und das Icon in dem Listenelement positioniert werden. Das benutzerdefiniertes Layout „custom_listview“ steht als xml Datei im Anhang.

### 6.2.1 Erstellung der benutzerdefinierten Liste

Jedes Listenelement besitzt einen Titel, eine Beschreibung des Titels und ein entsprechendes Icon. Zunächst wird eine Klasse Namens „ListInformation“ erstellt, daraus man durch „getter“ Methoden den Titel, die Beschreibung und das Icon bekommen kann. Als nächstes wird eine weitere Klasse „CustomAdapter“ erstellt, welche von der Klasse „BaseAdapter“ erbt. Die Hauptschnittstellen in der Aktivität PropertiesActivity lauten wie folgt:

1. public ListInformation(String strTitle, String strDescription, int intImages)
2. public CustomAdapter(Context context, ArrayList<ListInformation> listInfos)
3. ArrayList<ListInformation> listInfos;
4. public View getView(int i, View view, ViewGroup viewGroup) {
5. view=
   LayoutInflater.from(context).inflate(R.layout.custom_listview,viewGroup,false);
6. tvTitle.setText(listInfos.get(i).getStrTitle());
   tvDescription.setText(listInfos.get(i).getStrDescription());
   ivImage.setImageResource(listInfos.get(i).getIntImages());

Der Konstruktor der Klasse „Listinformation“ bekommt als Parameter den Titel, die Beschreibung und das Icon. Danach wird eine Array Liste dieser Klasse innerhalb der „CustomAdapter“ definiert. Dem Konstruktor wird als Parameter die Array Liste der Klasse „ListInformation“ übergebeben. Das benutzerdefinierte Layout wird durch den Aufruf der Klasse „LayoutInflater“ eingesetzt. Die Integer „i“ in der Methode „getView“ steht für die Position des aktuellen Wertes in der Liste. Der Titel, die Beschreibung und das Icon an der gleichen „i. ten“ Stelle bildet ein Listenelement im benutzerdefinierten Layout.

Im nächsten Schritt geht es um Empfangen der Akkueigenschaften innerhalb des Broadcastreceivers, welche in die Array Liste der Klasse „ListInformation“ hinzugefügt werden. Die Implementierung des Akku-Typs sieht beispielsweise folgendermaßen aus:

1. public void onReceive(Context context, Intent intent)
2. String strTech =
   intent.getExtras().getString(BatteryManager.EXTRA_TECHNOLOGY);
3. listInfos.add(newListInformation(getString(R.string.strTech),strTech,R.drawable.ic_battery_std_black_24dp));
4. customAdapter= new CustomAdapter(((PropertiesActivity)context),listInfos);
5. listView.setAdapter(customAdapter);

Die Information steckt im Parameter „Intent intent“ in der Methode „onReceive“. Durch den Aufruf der Methode „getExtras()“ gelangt man durch das Schlüsselwort EXTRA_TECHNOLOGY in der Klasse BatteryManager an die Technologie des Akkus, welche eine Zeichenkette ist, die dem String „strTech“ zugewiesen wird. Anschließend wird eine Instanz der Klasse „ListInformation“ erstellt, in dem man dem Konstruktor den Titel, die Beschreibung und das Icon übergibt. Der Titel und das Icon sind im Ressourcenordner

gespeichert. In diesem Fall ist die Beschreibung des Strings „strTech“ wichtig. Mit der Methode „add“ wird diese Instanz der Array-Liste hinzugefügt. Die Reihenfolge der Eigenschaften im Code ist identisch mit der Reihenfolge auf der Benutzeroberfläche der Anwendung. Als letztes wird dem „customAdapter“ die Array Liste zugewiesen. Der benutzerdefinierte Adapter wird für den Container „ListView“ eingesetzt.

## 6.3 SettingsActivity

Die Einstellungsaktivität beinhaltet den „SettingsFragment“. Fragmente kann man sich als ein Unterelement der Aktivität vorstellen, welches ein eigenes Lebenszyklus besitzt. Der „SettingsFragment“ implementiert das Interface „SharedPreferences.OnShared PreferenceChangeListener“ und überschreibt die Methode „onSharedPreferenceChanged“. Das ist sehr vorteilhaft, weil man sofort auf eine Benutzereingabe in Einstellungen reagieren kann. Diese Methode muss registriert werden. Sie wird beim Erscheinen des Settings Fragments in der Methode „onCreatePreferences“ wie folgt registriert.

„getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this)“

Die Registrierung wird wieder beim Zerstören des „SettingsFragments“ abgebrochen.

In Anwendungseinstellungen kann der Verbraucher folgende Änderungen vornehmen

i. Das Ladegerät auswählen
ii. Das Ladegerät umbenennen
iii. Den unteren Schaltpunkt eingeben
iv. Die Sprache der Anwendung ändern

### 6.3.1 Ladegerät auswählen

Dazu definiert man sich zuerst ein Set Objekt und zwei Arrays des Objektes CharSequence. Das Set Objekt ist eine Art Array Liste, in dem man die bereits gepaarten Bluetooth-Geräte speichern kann [14]. Daraus kann man auf die Informationen zugreifen insbesondere auf die MAC-Adresse des Ladegerätes, weil es für die Erstellung des Bluetooth Socket notwendig ist.

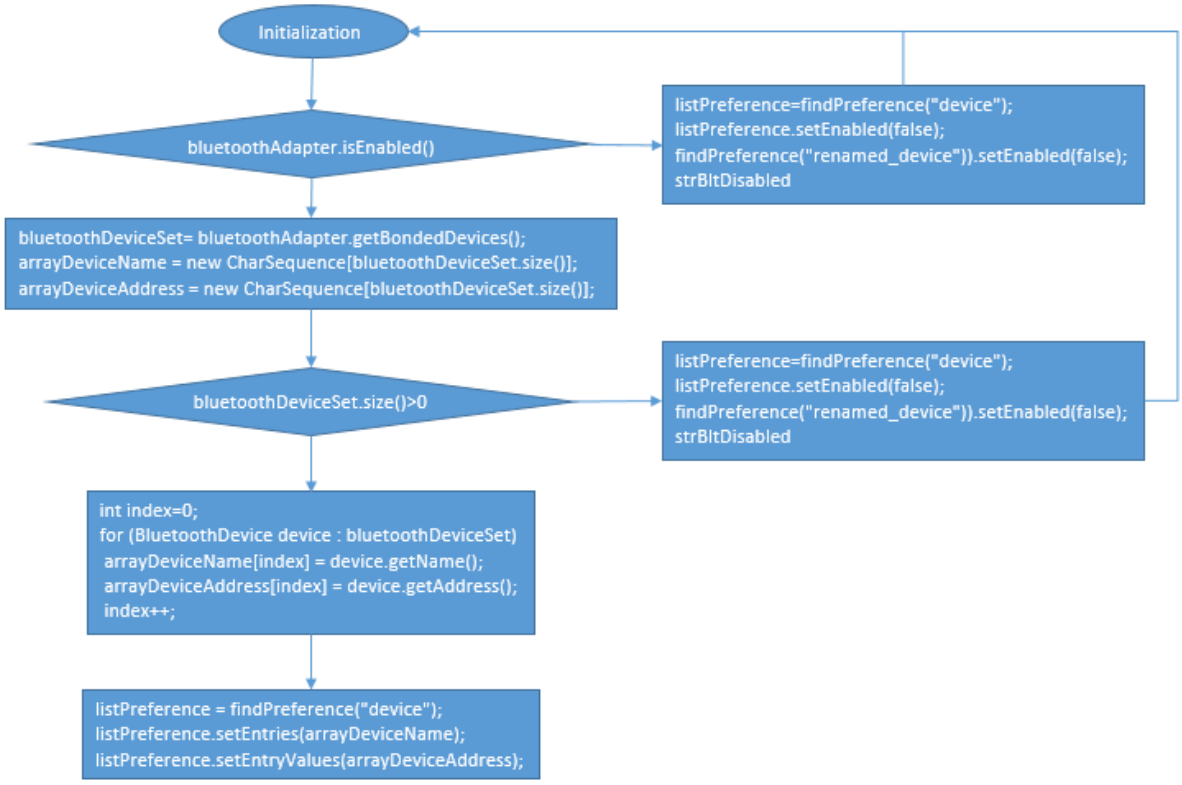



Abbildung 19: ListPreference

Die Abbildung 19 zeigt das Diagramm der Implementierung an, wie man die gepaarten Bluetooth-Geräte aus dem Android System in die Auswahlliste „ListPreference“ einträgt. Zunächst wird es überprüft, ob der Bluetooth aktiviert ist, weil man nur im aktivierten Fall auf die gepaarten Geräte zugreifen kann. Im deaktivierten Fall ist das Auswählen und Umbenennen des Ladegerätes nicht möglich. Der Rückgabewert der Methode „getBondedDevices()“ ist eine Array Liste des Set Objektes. Dann werden für die Speicherung der Namen und MAC-Adressen zwei „CharSequence“ Arrays in der Größe der Array Liste initialisiert. Wenn auf dem Android Smartphone kein Bluetooth-Gerät gepaart ist, empfängt die Anwendung einen „NullPointerException“ Fehler, der die Anwendung kollabiert. Deshalb muss überprüft werden, ob die Array Liste zumindest einen Eintrag besitzt. Innerhalb der „for-each“ Schleife werden den „CharSequence“ Arrays die Namen und MAC-Adressen des gepaarten Bluetooth-Gerätes aus dem Set Objekt zugewiesen. Die Eintragung der Namen und MAC-Adressen erfolgt durch den Aufruf der Methoden „setEntries“ für Namen und „setValues“ für die MAC-Adressen.

```
void setEntryValues(CharSequence[] entryValues)
```

Abbildung 20: setEntryValues[6]

Der Parameter dieser Methoden sind „CharSequence“ Arrays, das ist der Grund, warum die Arrays kein „String“ sind [17].

[6] *Quelle: https://developer.android.com/reference/android/preference/ListPreference*

Die Speicherung des ausgewählten Bluetooth-Gerätes aus der Liste vom Dialogfenster erfolgt in der implementierten Methode „onSharedPreferenceChanged“ folgenderweise:

```
1. public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String
   key)
2. if(key.equals("device")
3. Preference pre=findPreference(key);
   ListPreference listPreference = (ListPreference) pre;
4. editorMacAddress.putString(strBltDeviceAddress,listPreference.getValue().toString())
```

Welche Einstellungen der Benutzer gerade vornimmt, empfängt man gleichzeitig im Parameter „String key“ der implementierten Methode „onSharedPreferenceChanged“. Der Parameter „key“ ist ein in root_preferences .xml deklarierter Schlüssel als String von möglichen Einstellungen. Die Auswahlliste der gepaarten Bluetooth-Geräte ist in der „root_preferences .xml“ Datei unter dem Schlüssel „device“ deklariert worden. Danach wird eine Instanz vom Objekt „ListPreference“ durch den Aufruf der Methode „findPreference“ erstellt. Die Auswahl des Benutzers bekommt man über die Methode „getValue“, welche gleich in Preference gespeichert wird. Die Deklaration der Auswahlliste „ListPreference“ in „root_preferences .xml“ sieht folgendermaßen aus:

```
1. <ListPreference
   app:title="@string/strOtherDevice"
   app:key="device"
   app:dialogTitle="@string/strChooseDevice"
   app:defaultValue="0"
   app:useSimpleSummaryProvider="true"
   ></ListPreference>
```

Da es sich um eine dynamische Liste handelt und man nicht weiß, welches Listenelement dem Ladegerät gehört, gibt es keine voreingestellte Auswahl.

Eine sehr nützliche Deklaration ist „useSimpleSummaryProvider“, sie sorgt dafür, dass der ausgewählte Name sofort unterhalb des Titels (Summary) erscheint. Andernfalls müsste man den „Summary“ in der Methode „onResume“, bevor das Layout in den Vordergrund eintritt und in der Methode „onSharedPreferenceChanged“ aktualisieren, was sehr umständlich wäre.

### 6.3.2 Ladegerät umbenennen

Den Namen des Ladegerätes kann man innerhalb der Anwendung umbenennen, jedoch bleibt es in der Auswahlliste der gepaarten Bluetooth-Geräte unverändert. Der Grund liegt darin, dass die angezeigten Namen der Bluetooth-Geräte in der Auswahlliste vom Android System übernommen werden.

Wenn man den Namen trotzdem in der Auswahlliste ändern möchte, gebe es zwei Möglichkeiten. Die erste Möglichkeit besteht darin, den Namen über ein Bluetooth-Signal im

Ladegerät umzubenennen. Diese Umbenennung wäre jedoch erst nach einem Neustart der Anwendung möglich und man müsste auf die Threads in MainActivity zugreifen um das Signal aus „SettingsActivity“ zu senden. Die zweite Möglichkeit besteht darin, das Ladegerät umzubenennen, bevor es zur Auswahlliste hinzugefügt wird. Dazu müsste der geänderte Name in einer zusätzlichen Preference gespeichert werden und bei jedem Öffnen der Auswahlliste und der „SettingsActivity“ zusätzlich für Summary verglichen werden.

Diese beiden Optionen wurden getestet. Sie verursachten meist „NullPointerException“ und die Namen waren zu falschen MAC Adressen zugeordnet. Deshalb wird der Name des Ladegerätes in der Auswahlliste nicht geändert.

Das Ladegerät wird wie folgt umbenannt. Der Benutzer wählt das Ladegerät aus der Auswahlliste im Dialogfenster aus. Der Name des Ladegerätes erscheint gleichzeitig im Summary der Auswahlliste und im Textfeld „Ladegerät umbenennen“. Wenn der Benutzer auf den Text „Ladegerät umbenennen“ klickt, öffnet sich dann automatisch ein Dialogfenster mit dem Namen des ausgewählten Ladegerätes im Textfeld. Der Benutzer kann das Ladegerät umbenennen oder das Umbenennen abbrechen. Das umbenannte Ladegerät erscheint dann im Summary von „Ladegerät umbenennen“., wird in der MainActivity unter diesem Namen gespeichert. Die Implementierung sieht folgendermaßen aus:

1. public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key)
2. if(key.equals("device")
3. ((EditTextPreference)findPreference("renamed_device")).setText(listPreference.getEntry().toString());
4. if(key.equals("renamed_device")
5. String strRenamed = editTextPreference.getText();
6. editorMacAddress.putString(strBltDeviceName,strRenamed);

Ob der Benutzer ein Ladegerät ausgewählt hat, wird durch den Vergleich in der If-Anweisung festgestellt. Das Schlüsselwort (key) von der Auswahlliste ist die Zeichenkette „device“.

Wenn das Schlüsselwort „device“ ist, wird eine Instanz des Objektes „EditTextPreference“ durch den Aufruf der Methode „findPreference("renamed_device")“ erstellt, wobei die Zeichenkette „renamed_device“ das Schlüsselwort für „EditTextPreference“ ist, dann nimmt man den Namen des Ladegerätes über die getter Methode „getEntry“, und setzt in das Textfeld von „EditTextPreference“. Wenn der Benutzer das Dialogfenster von „EditTextPreference“ öffnet, beinhaltet der Parameter „String key“ der implementierten Methode „onShared PreferenceChanged“ das Schlüsselwort „renamed_device“. Beim Schließen des Dialogfensters wird die Eingabe des Benutzers zum Umbenennen des Ladegerätes in Preference gespeichert.

### 6.3.3 Unteren Schaltpunkt eingeben

Mit Eingabe des unteren Schaltpunktes definiert man die Breite des Regelbereiches. Es muss mindestens eine Regelbreite von einem Prozent vorhanden sein, weil es ansonsten das Ladegerät den Ladestrom ständig ein-und ausschaltet. Die minimale Regelbreite dieser Anwendung ist zwei Prozent und die maximale acht Prozent.

1) if (isCharging && currentBatteryLevel >= intSetPoint)
   sendMessage(strStopCharging);
2) !isCharging && currentBatteryLevel <= intSetPoint – intRelativePoint
   sendMessage(strStartCharging);

Die Feststellung der Regelbereiche passiert über den Vergleichsoperator „größer gleich“ und „kleiner gleich“.

Zum Beispiel wäre die aktuelle Akkuladung 90% und es gäbe keinen unteren Schaltpunkt, dann wäre nur die boolesche Variable „isCharging“ wichtig, welche durch die Änderung des Ladestatus ständig wechselnd auf „true“ und „false“ gesetzt wird. Diese Anwendung würde dann das Startsignal und das Stoppsignal entsprechend an das Ladegerät senden.

Die Beseitigung dieses Problems scheint im Sinne dieses Regelkreises durch einfaches Ein- und Ausschalten des Ladestroms im Applikationscode nicht möglich zu sein, so dass eine Definition eines unteren Schaltpunktes in der Applikation unumgänglich war. Eine Einstellung der maximalen Ladung von glatt 90% kann durch entsprechende Konfiguration des Ladegeräts realisiert werden, was das Ziel der weiteren Entwicklung dieses Projekts wäre.

Die Implementierung des unteren Schaltpunktes in „SettingsActivity“ geschieht über das Objekt „SeekBarPreference“ folgendermaßen:

1. public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key)
2. if(key.equals("relative")
3. int intSummary= (int)seekBarPreference.getValue() + 2;
4. editorRelPoint.putInt(strRelativePoint,intSummary);
5. seekBarPreference.setSummary(getString(R.string.strControlRange)+" "+intSummary+"%");

Der Schieberegler „seekBarPreference“ zum Speichern des unteren Schaltpunktes findet in der implementierten Methode „onSharedPreferenceChanged“ unter dem Schlüsselwort(key) „relative“. Durch den Aufruf der Methode „getValue()“ in der Klasse „SeekBarPreference“ liest man die Benutzereingabe zeitgleich. Der Mindestrückgabewert vom Schiebregler ist null. Deshalb wird noch zwei Prozent dazu addiert, welches die minimale Regelbreite ist.

Für das Objekt „SeekBarPreference“ ist keine Definition von „useSimpleSummaryProvider“ in der „root_preferences .xml“ Datei möglich. Deshalb muss das Summary im „SettingsFragment“ in der Methode „onResume“, bevor der Schieberegler im Vordergrund

erscheint und in der Methode „onSharedPreferenceChanged“, wenn der Benutzer mit dem Schieberegler interagiert, aktualisiert werden. Das Eintragen des Rückgabewertes in das Textfeld Summary erfolgt durch den Aufruf der Methode „setSummary“. Der Integer Wert „intSummary“ wird zum Verarbeiten in „MainActivity“ in Preference unter dem Schlüsselwort „strRelativePoint“ gespeichert.

### 6.3.4 Anwendungssprache ändern

Für das Erstellen der Auswahlliste im Dialogfeld wird es wieder das Objekt „ListPreference“ verwendet. Es handelt sich dabei um eine feste Liste, die nur zwei Eintrage hat, deshalb kann es im Ordner strings.xml als ein String Array definiert werden. So sieht die Array Definition in der strings.xml Datei.

- <string-array name="language">
  <item>ENGLISH</item>
  <item>DEUTSCH</item>
  </string-array>

In der „root_preferences .xml“ Datei legt man als Standardsprache der Anwendung Englisch fest. Es wird wieder der einfache Summary Provider „useSimpleSummaryProvider“ benutzt, die ausgewählte Sprache erscheint gleichzeitig im Textfeld Summary. Die ausgewählte Sprache wird im Preference unter dem Schlüsselwort „language“ gespeichert. Die Deklaration der Auswahlliste im Dialogfeld sieht in der „root_preferences .xml“ Datei folgendermaßen aus:

- <ListPreference
  app:title="@string/strLanguage"
  app:entries="@array/language"
  app:entryValues="@array/language"
  app:dialogTitle="@string/strLanguage"
  app:defaultValue="ENGLISH"
  app:key="language"
  app:useSimpleSummaryProvider="true"
  ></ListPreference>

Vorteilhaft ist bei dieser festen Liste die Einträge über „app:entries“ und „app:entryValues“ zu definieren. In diesem Fall haben beide dasselbe String Array „@array/language". Die Sprachänderung ist ein umständlicher Prozess. Das heißt, es muss bei jeder Aktivität in der Methode „onCreate“ bevor die Benutzeroberfläche im Vordergrund erscheint die Texte auf die ausgewählte Sprache umgesetzt werden.

Dennoch muss man beim Wechsel zwischen den Aktivitäten auf die ausgewählte Sprache achten und gegebenenfalls umstellen. Damit die ausgewählte Sprache sofort wirkt, muss die „SettingsActivity“ neu gestartet werden. Ein Sprachwechsel wird in „MainActivity“ wie folgt durchgeführt:

1. protected void onCreate(Bundle savedInstanceState)
2. SharedPreferences sharedPref =
   PreferenceManager.getDefaultSharedPreferences(this);
   strLanguageMain = sharedPref.getString("language", "ENGLISH");
3. switchLanguage(this.getResources(),strLanguageMain);

Die verwendete Sprache wird anhand des Schlüsselworts „language“ aus dem Preference gelesen und dem String „strLanguageMain“ zugewiesen. Der String wird als Parameter an die Methode „switchLanguage“ übergegeben, die die Texte auf der Benutzeroberfläche in die ausgewählte Sprache konvertiert. Die Methode „switchLanguage“ wird in der Klasse „Constants“ wie folgt implementiert.

```
public static void switchLanguage(Resources resources,String locale){

    DisplayMetrics metrics= resources.getDisplayMetrics(); //access class DisplayMetrics
    Configuration configuration= resources.getConfiguration(); //access class Configuration
    switch (locale){
        case "ENGLISH":
            configuration.setLocale(new Locale("en")); // ,,en'' key for english
            break;
        case "DEUTSCH":
            configuration.setLocale(new Locale("de")); // ,,de'' key for german
            break;
        default:
            configuration.setLocale(new Locale("en")); // default language english
            break;
    }
    resources.updateConfiguration(configuration,metrics); //refresh values
}
```

Abbildung 21: Methode switchLanguage

Das Schlüsselwort für die englische Sprache ist „en“ und für die deutsche Sprache ist „de“ [18]. Dies erfordert den Zugriff auf die Klassen Configuration und DisplayMetrics [19]. Es übersetzt nicht nur Strings, die in der Datei strings.xml enthalten sind, sondern auch die Hinweise wie ABBRECHEN oder CANCEL, wenn ein Dialogfenster geöffnet wird.

```
Log.d("main", "onPostResume: "+ strLanguageMain);
SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);

String strChosenLanguage = sharedPref.getString("language", "ENGLISH");

if(!strLanguageMain.equals(strChosenLanguage)){
    Log.d("main", "onPostResume: sprache aendern");
    strChosenLanguage = strLanguageMain;
    switchLanguage(this.getResources(),strChosenLanguage);
    recreate();
}
```

Abbildung 22: Sprachänderung onResume

Im Zustand „onResume“ wird verglichen, ob sich die Sprache umgewechselt hat. Gegebenenfalls wird sie mit der Methode „switchLanguage“ geändert. Zusätzlich wird der Ressourcenordner aktualisiert, sodass die ausgewählte Sprache sofort auf der Benutzeroberfläche angezeigt wird.

# 7 Anwendung testen

Es gibt zwei Möglichkeiten, eine Anwendung in Android Studio zu testen. Man kann ein virtuelles Gerät erstellen, das als „Emulator“ bezeichnet wird, oder die Anwendung auf einem realen Smartphone testen. Dieses Projekt wurde selten mit einem Emulator getestet, da die Kommunikation zwischen dem Ladegerät und der Anwendung nicht realisiert werden konnte. Ein weiterer Aspekt war, dass der Emulator viel Speicherplatz in Anspruch nahm und das Programm Android Studio verlangsamte. Bei vielen Smartphones und Tablets mit unterschiedlichen Pixelgrößen im Hoch- und Querformat konnte das Layout jedoch auch ohne Emulator überprüft werden.

Um eine Anwendung in Android Studio auf das Smartphone hochladen zu können, müssen die Entwickleroptionen des Smartphones aktiviert sein. Sie werden durch siebenmaliges Klicken auf die „Build-Nummer“ in Einstellungen des Smartphones aktiviert. In der Entwickleroption muss auch die Verwendung von USB-Debugging zulässig sein. Diese Anwendung wurde auf einem Smartphone mit einem API-Level 26 und einem Tablet-API-Level 19 getestet.

## 7.1 Testen der Anwendung auf der Benutzeroberfläche

Von der Anwendung wurde für jeden möglichen Zustand ein Bild (Screenshot durchnummeriert) erstellt. Wenn die Anwendung zum ersten Mal startet, gibt es kein voreingestelltes Ladegerät, klickt der Benutzer dennoch auf den Button, erscheint ein Dialogfeld zum Erlauben von Bluetooth, wenn es deaktiviert ist (Test 1 und Test 5).

Wenn Bluetooth aktiviert und kein Ladegerät ausgewählt ist, erscheint ein Hinweistext in Form einer Toast Nachricht. Die Toast Nachricht ist ein kleines Fenster mit Hinweistext und verschwindet nach einer Weile von selbst (Test 6).

Die maximale Ladung von 90% ist der Default Wert. Der Schieberegler ist an der richtigen Stelle positioniert. Der untere Schaltpunkt ist 87%, weil der Default Regelbreite drei Prozent ist. Man kann anhand des kleinen Batteriebild oben in der Statusleiste des Smartphones feststellen, dass das Icon für den Ladestatus richtig ist. Die Anzeige der aktuellen Ladung kann man auch durch den Vergleich feststellen. Es ist das richtige Akkubild, für diesen Bereich eingesetzt worden.

Wie man auf der Abbildung 25 sehen kann, besitzt die Benutzeroberfläche besitzt ein Optionsmenu, klickt man auf das Optionsmenu, sind die beiden Elementen Eigenschaften und Einstellungen aufgelistet.

Das Menü-Icon zum Öffnen der Anleitung funktioniert auch wie gewünscht in richtiger Sprache (Test 2).

Die Views sind an der richtigen Stelle positioniert, wie man sie auf dem „RelativeLayout“ in der activity_main.xml Datei deklariert hat. Auf der Benutzeroberfläche Eigenschaften sind die Attribute in richtiger Reihenfolge aufgelistet (Test 12).

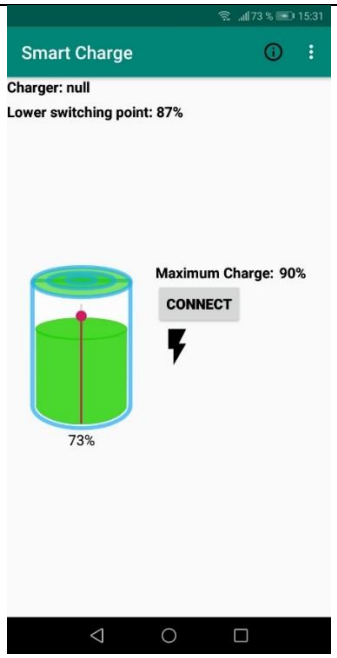



Abbildung 23: Test 1 Startseite

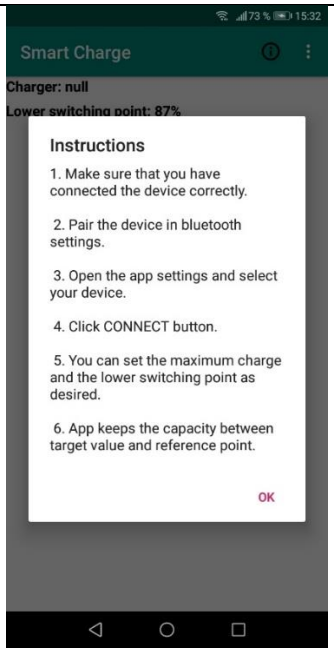



Abbildung 24: Test 2 Menu Anleitung

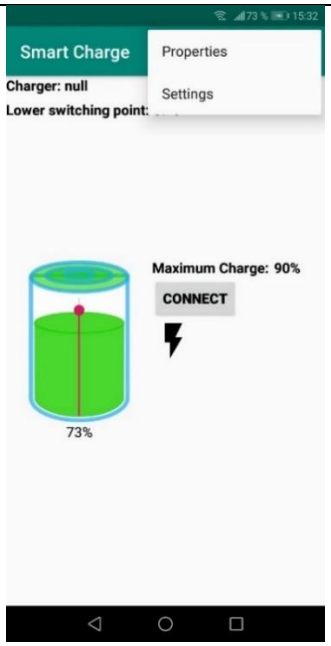



Abbildung 25: Test 3 Optionsmenu

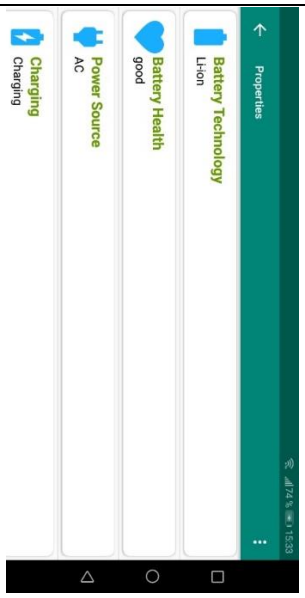



Abbildung 26: Test 4 Properties Bildschirm gedreht


Abbildung 27: Test 5 Erlaubnis zum Einschalten von Bluetooth

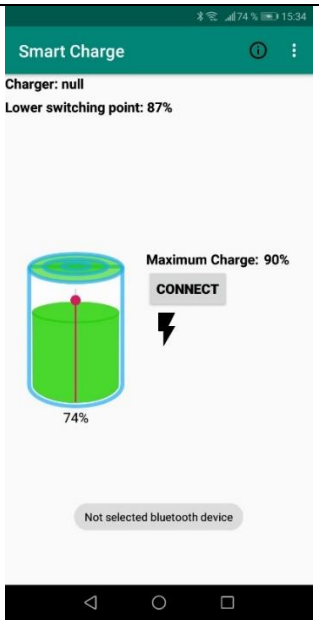



Abbildung 28: Test 6 Kein ausgewähltes Ladegerät

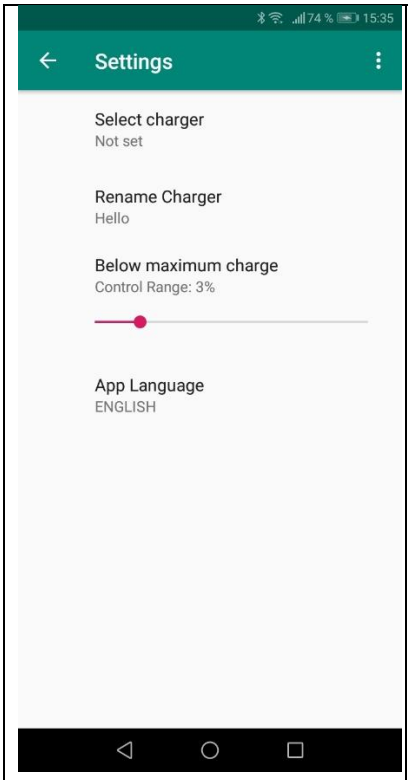



Abbildung 29: Test 7 Settings

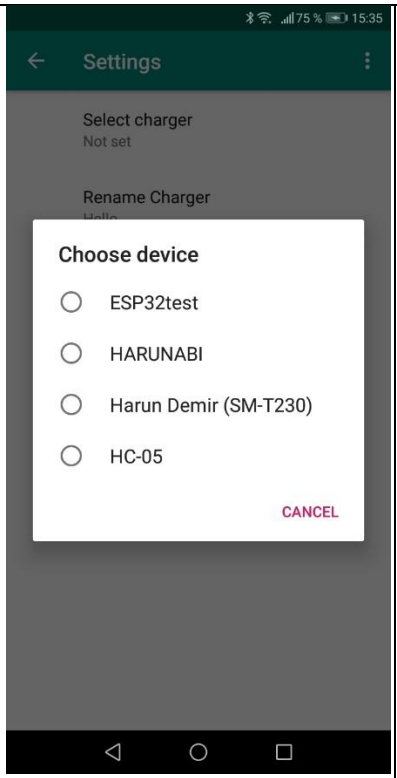



Abbildung 30: Test 8 Ladegerät auswählen

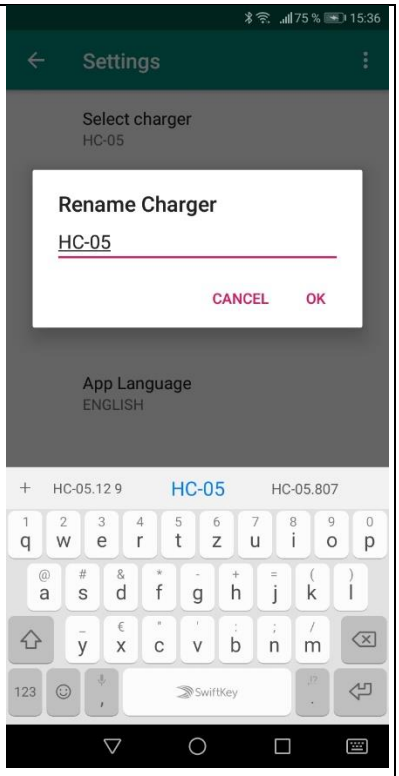



Abbildung 31: Test 9 Ladegerät umbenennen

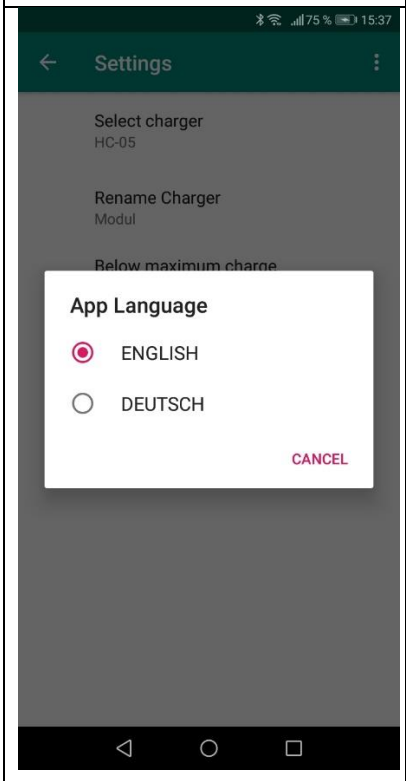



Abbildung 32: Test 10 Anwendungssprache ändern

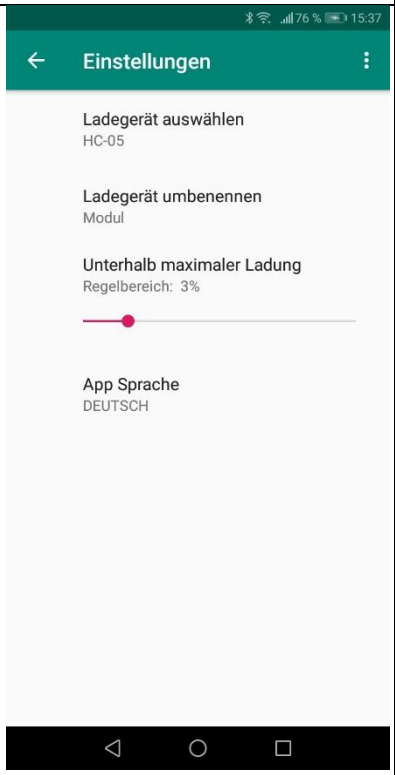



Abbildung 33: Test 11 Einstellungen

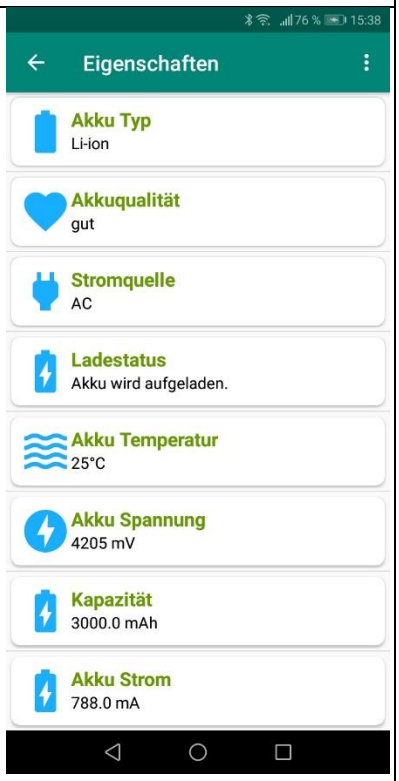



Abbildung 34: Test 12 Eigenschaften

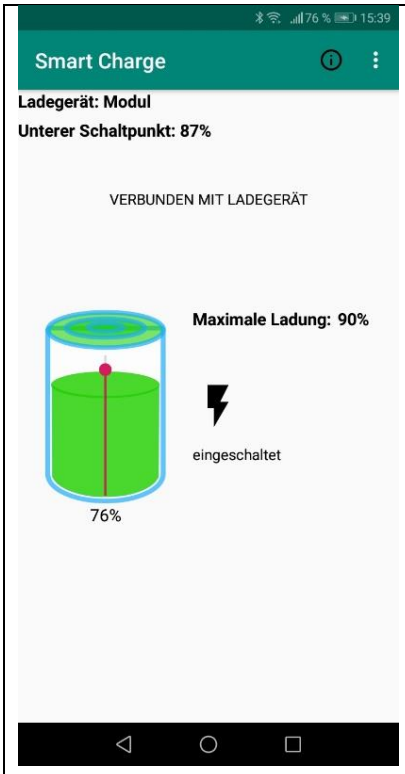



Abbildung 35: Test 13 Regelbereich oberhalb der aktuellen Ladung

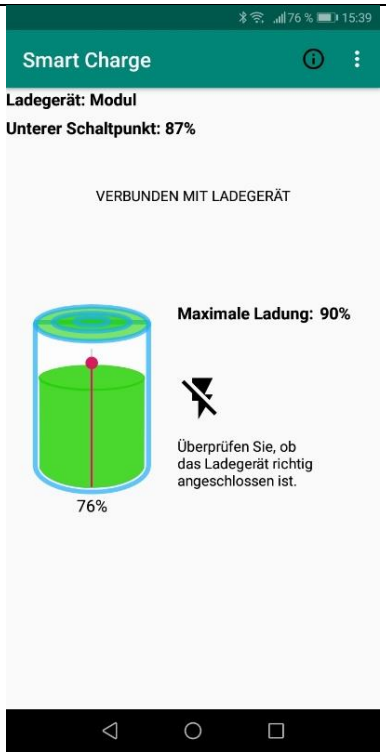



Abbildung 36: Test 14 Fehlermeldung kein Ladestrom

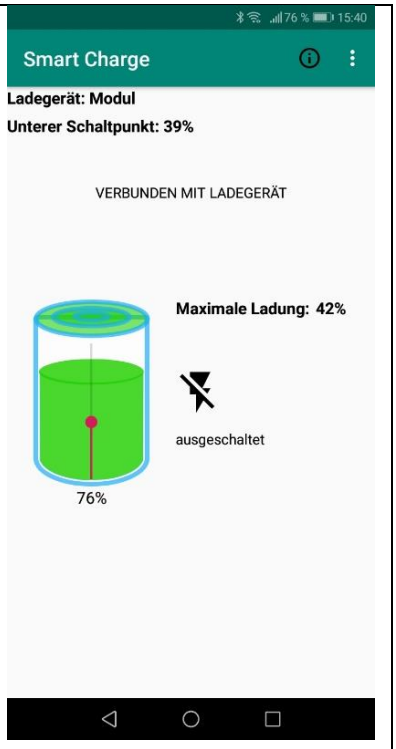



Abbildung 37: Test 15 Regelbereich unterhalb der aktuellen Ladung

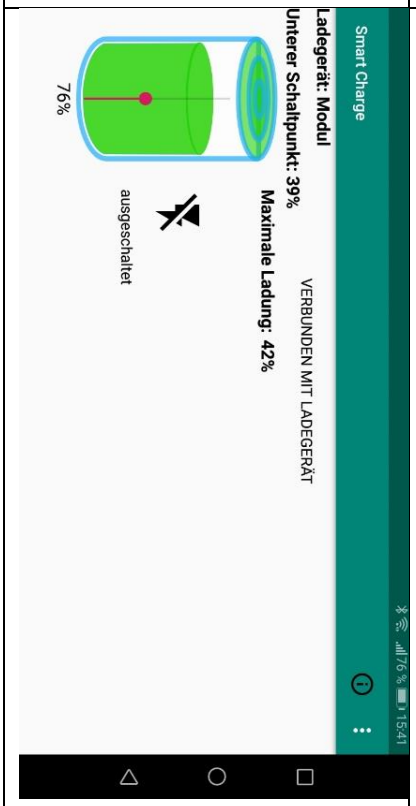



Abbildung 38: Test 16 Bildschirm während der Regelung umgedreht

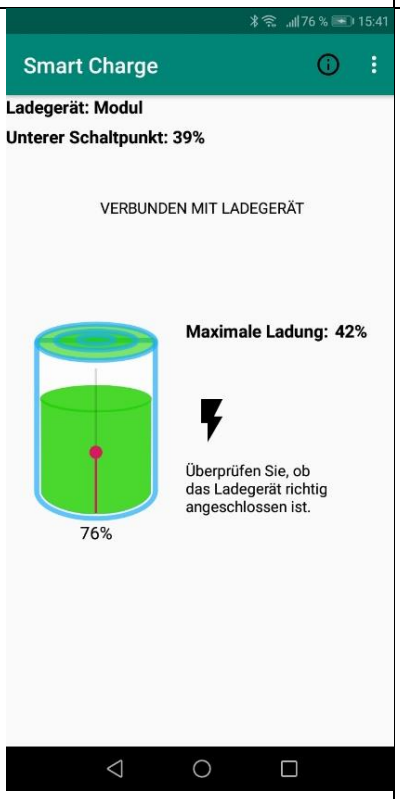



Abbildung 39: Test 17 Fehlermeldung Ladestrom

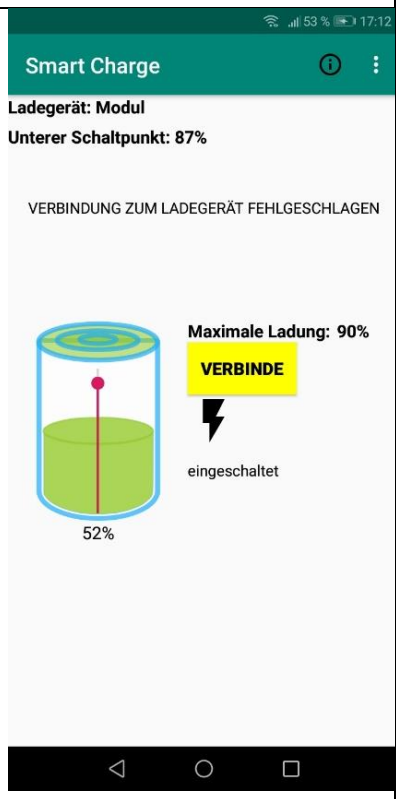



Abbildung 40: Test 18 Bluetooth wurde ausgeschaltet

Die Schriftfarbe, Schriftgröße und das Icon entsprechen auch der Deklaration in der activity_properties.xml Datei Beim Umdrehen des Bildschirms in das Querformat sind vier Attribute auf der Oberfläche sichtbar, Man kann aber durch Scrollen die anderen vier Attribute auch ansehen (Test 4).

Die Benutzeroberfläche Einstellungen wird wie erwartet über das Optionsmenu mit richtiger Beschriftung angezeigt. Die Reihenfolge der Optionen und Beschriftungen sind wie in der „root_preferences.xml“ Datei deklariert ist. Weil es noch kein Ladegerät ausgewählt ist, erscheint im Summary der Hinweis „Not Set“. Der Text „Hello“ ist der Default Wert, es ist ohne Bedeutung solange kein Ladegerät ausgewählt ist (Test 7).

Das Summary drei Prozent von dem unteren Schaltpunkt ist auch der Standardwert. Der Schieberegler für die Eingabe des Regelbereiches ist an der richtigen Stelle positioniert.

Es zeigt die Standardsprache Englisch im Summary an. Ein Klick auf den Text „Ladegerät auswählen“ öffnet die Auswahlliste im Dialogfenster mit vier Bluetooth-Geräten, die zuvor auf diesem Smartphone gekoppelt wurden. Dann wird das Ladegerät „HC-05“ ausgewählt. Es ist der Name des Bluetooth Moduls auf dem Ladegerät. Das Dialogfenster erlischt nach der Auswahl sofort. Der Name stand zeitgleich auch im Summary von „Rename Charger“. Es öffnet sich auch ein Dialogfenster mit einem Klick auf den Text „Rename Charger“. Im Textfeld erscheint wie erwartet der Name des Ladegerätes „HC - 05“. Der Name wird auf Modul geändert und mit „OK“ bestätigt. Das Dialogfenster erlischt und der neue Name steht im Summary von „Rename Charger“ jedoch nicht im Summary von „Select Charger“.

Um die Sprachänderung zu testen, wird auf den Text „App language“ geklickt, öffnet sich wieder eine Auswahlliste im Dialogfenster mit den Optionen „ENGLISH“ und „DEUTSCH“. Dann wird „DEUTSCH“ ausgewählt. Das Fenster verschwindet und die Benutzeroberfläche wird wie erwartet neu gestartet. Die Anwendungssprache wechselt von Englisch auf Deutsch. Dann wird auf den Zurückbutton geklickt, die Benutzeroberfläche „Smart Charge“ öffnet. Die Sprache hat sich auch auf Deutsch umgesetzt außer der Überschrift „Smart Charge“, die bleibt wie gewünscht unverändert. Somit sind alle Anforderungen an der Anwendung beim Testen auf der Benutzeroberfläche erfüllt worden.

## 7.2 Testen der Laderegelung und Bluetooth Verbindung

In diesem Abschnitt wird die Regelschleife unter die Lupe genommen. Dabei geht man auf die Hauptanforderung ein, den Ladestrom nach Vorgabe des Benutzers einzustellen. Zuerst wird der Zustand überprüft, indem sich der Regelbereich oberhalb der tatsächlichen Akkuladung befindet.

Es wird eine maximale Ladung von 90% und eine Regelbreite von drei Prozent eingegeben. Der untere Punkt beträgt demnach 87%. Mit einem Klick auf den Button VERBINDE hat sich die Anwendung per Bluetooth mit dem Ladegerät verbunden. Der Hinweistext „VERBUNDEN MIT LADEGERÄT“ über den Verbindungstatus erschien oberhalb des Akkubildes an der deklarierten Stelle (siehe Abbildung 35).

Der Button ist unsichtbar, weil er keine Funktion mehr hat, solange sie erfolgreich verbunden sind. Das Ladegerät schaltete den Ladestrom ein und machte dem Benutzer über den Hinweistext „eingeschaltet“ bekannt.

Dann wird die Stromversorgung unterbrochen, die Anwendung erkennt den Fehler und meldet über den Hinweistext „Überprüfen Sie, ob das Ladegerät richtig angeschlossen ist“ (siehe Abbildung 36).

Die Bluetooth-Verbindung besteht weiterhin, die Regelung kann jedoch nicht fortgesetzt werden. Wenn man das Smartphone wieder an die Stromversorgung anschließt, erlischt die Fehlermeldung. Das gleiche Überprüfungsverfahren wird auch für den Regelbereich unterhalb der tatsächlichen Akkuladung gemacht. In diesem Fall wird der Ladestrom ausgeschaltet. Der Hinweis über den Verbindungsstatus und über die Fehlermeldung funktioniert im Vordergrund einwandfrei.

Die Überprüfung der Laderegelung im Hintergrund wird anhand der Log Anweisungen gemacht, welche im Handler „handlerControlCapacity“ und in der Methode „onReceive“ des Broadcastreceivers initialisiert sind.

Dazu wird das Smartphone über USB-Anschluss mit Android Studio verbunden und das Ladegerät an die Stromversorgung. Das Ladegerät wartet auf eine Anforderung der Anwendung, um eine Verbindung herzustellen. Die Anwendung „Smart Charge“ wird nochmal hochgeladen. Nun kann man die Log Anweisungen im „Run“ Block beobachten. Die Benutzeroberfläche „Smart Charge“ wird gestartet.

```
D/.MainActivity: run: connected
D/.MainActivity: handleMessage: connected
D/.MainActivity: handlerControlCapacity: 50setpoint 79 relative 3
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: onReceive: 50
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 50setpoint 79 relative 3
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 50setpoint 79 relative 3
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 50setpoint 79 relative 3
D/.MainActivity: onReceive: 50
```

Abbildung 41: Abschnitt Run Fenster Verbunden

Die Abbildung 41 zeigt den Abschnitt des Run Fensters nach einer erfolgreichen Verbindung. Man sieht, dass die log Anweisung der „onReceive“ Methode zuerst im Run Fenster erscheint. Der Aufruf dieser Methode ist dem Android System überlassen bzw. erfolgt in ungleichen Zeitabständen. Bei der Überprüfung auf dem Tablet dauert es um einige Sekunden mehr. Mit einem Klick auf den Button „VERBINDE“ erscheint im Fenster die Log Anweisung „run: connected“. Es bedeutet, dass die Bluetooth-Verbindung zwischen der Anwendung und dem Ladegerät hergestellt wurde. Dieses Log ist im „ConnectThread“ definiert worden, das heißt, dass dieser Thread jetzt im Hintergrund ausgeführt wird. Dann sieht man die Nachricht, welche an den Handler „handlerConnectionState“ gesendet wurde. Diese Nachricht wird nur einziges Mal bei einem erfolgreichen Bluetooth Verbindung versendet. Danach startet der „SampleCapacityThread“ zur Abtastung der aktuellen Akkuladung alle drei Sekunden. Der abgetastete Wert wird mithilfe des Handlers „HandlerControlCapacity“ gelesen. Zur Kontrolle werden auch neben der aktuellen Akkuladung auch die maximale Ladung „setPoint“ und die Regelbreite „relative“ angezeigt.

Um den Regelbereich unterhalb der aktuellen Akkuladung zu testen, wird die maximale Ladung auf 36% eingestellt. Die folgende Abbildung 42 zeigt den Abschnitt des Run Fenster in diesem Regelbereich.

```
D/.MainActivity: handlerControlCapacity: 42setpoint 36 relative 3
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: write:
D/.MainActivity: handleMessage: write
D/.MainActivity: run: READ
D/.MainActivity: handleMessage: read Y
D/.MainActivity: handlerControlCapacity: 42setpoint 36 relative 3
D/.MainActivity: write:
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handleMessage: write
D/.MainActivity: onReceive: 42
D/.MainActivity: onReceive: 42
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 42setpoint 36 relative 3
```

Abbildung 42: Log Unterer Regelbereich im Vordergrund

Es läuft weiterhin die Abtastung der Akkuladung alle drei Sekunden. In diesem Fall beträgt die aktuelle Ladung 42%, die maximale Ladung 36% und der untere Schaltpunkt 33%.

Wie erwartet sendet die Anwendung das Ausschaltsignal an das Ladegerät, welches man an dem Log „write“ erkennt. Daraufhin schaltet das Ladegerät die Stromversorgung aus und sendet als Rückantwort ein einziges Mal den String „Y“ an die Anwendung zurück, um die Bluetooth Verbindung nicht zu belasten.

Die Rückantwort erkennt man an dem Log „read Y“. Das Smartphone wird weiterhin über den USB-Anschluss aufgeladen, deshalb kann an dieser Stelle die Regelung nicht fortgesetzt werden. Auf der Benutzeroberfläche „Smart Charge“ wird entsprechende Fehlermeldung als Text ausgegeben.

Es tritt jedoch das Problem auf, dass alle drei Sekunden im Rhythmus der Abtastung das Ausschaltsignal an das Ladegerät gesendet wird. Es hängt von der If-Anweisung im Handler „handlerControlCapacity“ ab, indem das Ausschaltsignal initialisiert ist.

- if (isCharging && intCurrentBatteryLevel >= intSetPoint) {
  sendMessage(strStopCharging); }

Die Variable „isCharging“ für den Ladestatus verhindert normalerweise, dass ständig Ausschaltsignal gesendet wird. In diesem Fehlerfall kann sie ihre Aufgabe nicht erfüllen, weil es dafür auf „false“ gesetzt werden müsste. Eine explizite Deklaration von „isCharging = false“ in der If Anweisung wird die Regelung zu einem Fehler führen, weil es nicht der Realität entspricht. In dem anderen Fall, wenn die Anwendung Einschaltsignal sendet und es fließt weiterhin kein Ladestrom, wird die Anwendung auch ständig Einschaltsignal an das Ladegerät senden. Dieser Fall kann anhand von Log Anweisungen nicht überprüft werden, weil es dafür das Smartphone von USB-Anschluss entfernt werden musste.

Nun wird der Bildschirm der Anwendung bei einer laufenden Regelung umgedreht, um zu beobachten, wie sich die Regelung im Lebenszyklus der Main Aktivität verhält. Die Abbildung 43 zeigt die Log Anweisungen beim Umdrehen des Bildschirms in das Querformat an.

```
D/.MainActivity: onReceive: 50
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 50setpoint 79 relative 3
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 50setpoint 79 relative 3
D/.MainActivity: onReceive: 50
```

Abbildung 43: Bildschirm umgedreht

Man stellt fest, dass der Broadcastreceiver auch die Akkuladung im Querformat aktualisiert. Das bedeutet, die Regelung besteht weiterhin beim Wechsel zwischen den Benutzeroberflächen. Zum Schluss wird die Anwendung in den Hintergrund verschoben, um die Regelung und den Verbindungsstatus zu beobachten.

```
D/.MainActivity: handleMessage: write
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 42setpoint 36 relative 3
D/.MainActivity: write:
D/.MainActivity: handleMessage: write
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 42setpoint 36 relative 3
D/.MainActivity: write:
D/.MainActivity: handleMessage: write
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 42setpoint 36 relative 3
D/.MainActivity: write:
D/.MainActivity: handleMessage: write
```

Abbildung 44:Anwendung im Hintergrund

Im Hintergrund läuft die Regelung nur im Handler „handlerControlCapacity“. In der Broadcastreceiver Methode „onReceive“ werden die aktuelle Akkuladung und Ladestatus nicht mehr aktualisiert. Aus diesem Grund können Smartphones mit einem API-Level von weniger 21 die Akkuladung nur im Vordergrund regeln. Das hat auch einen negativen Einfluss auf die Regelschleife im Handler, weil die Variable „isCharging“ zur Bestimmung des Ladestatus in der Methode „onReceive“ initialisiert ist.

Dies würde beispielsweise bedeuten, dass der Ladestrom beim Erreichen einer maximalen Ladung von 90% vom Ladegerät wie erwartet ausgeschaltet wird, weil die Variable „isCharging“ in diesem Fall der Realität entspricht. Im nächsten Schritt beim Einschalten des Ladestromes kann es jedoch zu einem Fehler kommen, wenn die Anwendung zwischendurch nicht einmal im Vordergrund erschien, weil „isCharging“ nicht mehr aktuell ist. Dieses Problem könnte man umgehen, wenn im Thread auch der Ladestatus abgetastet wird. Eine mögliche Definition könnte wie folgt aussehen:

```
int status = batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_STATUS);
boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING
        || status == BatteryManager.BATTERY_STATUS_FULL;
```

Abbildung 45: Eine mögliche Definition für den Ladestatus im Thread

BATTERY_PROPERTY_STATUS Added in API level 26

```
public static final int BATTERY_PROPERTY_STATUS
```

Battery charge status, from a BATTERY_STATUS_* value.

Constant Value: 6 (0x00000006)

Abbildung 46:Property Status[7]

Jedoch wird es nur funktionieren, wenn das Smartphone mindestens API- Level 26 hat.(siehe Abbildung 1). Als letztes wird der Ausfall der Bluetooth-Verbindung während der Laderegelung getestet. Der Ausfall wird im „ConnectedThread“ erkannt, der über die Nachricht des Handlers „READ ERROR“ erkannt wird. In fast allen Fällen tritt der Ausfall im „Inputstream“ auf.

```
D/.MainActivity: run: SampleCapacityThread
D/.MainActivity: handlerControlCapacity: 42setpoint 76 relative 3
D/.MainActivity: handleMessage: READ ERROR
D/.MainActivity: Input stream was disconnected
    java.io.IOException: bt socket closed, read return: -1
        at android.bluetooth.BluetoothSocket.read(BluetoothSocket.java:537)
        at android.bluetooth.BluetoothInputStream.read(BluetoothInputStream.java:96)
        at java.io.InputStream.read(InputStream.java:101)
        at com.example.smartcharge.MainActivity$ConnectedThread.run(MainActivity.java:763)
```

Abbildung 47: Log Bluetooth Ausfall

Trotzdem wird für diese Anwendung der Ladestatus in der Broadcastreceiver Methode „onReceive“ bestimmt. Erstens die Wahrscheinlichkeit für das Auftreten dieses Fehlers ist sehr gering, weil Aufladen und Entladen langfristige Prozesse sind. Zweitens muss man sich nicht auf die Smartphones mit einem API-Level von mehr als 26 beschränken. Drittens es gibt keine zweite „isCharging“ Variable im Handler. Das heißt, der Ladestatus wird nicht alle drei Sekunden abgetastet, sondern vom Android System bei einer Änderung aktualisiert.

[7] *Quelle https://developer.android.com/reference/android/os/BatteryManager.html#BATTERY_PROPERTY_STATUS*

# 8 Zusammenfassung

Das Ziel dieses Projektes war eine Anwendung zu entwickeln, welche die Akkuladung mithilfe eines äußeren Schalters nach Vorgabe des Benutzers regelt. Dieses Ziel ist zum größten Teil erreicht worden. Diese Anwendung ermöglicht dem Benutzer Eingabe einer beliebigen maximalen Ladung. Der Benutzer kann die Eingabe jederzeit nach seinen Wünschen ändern. Die Benutzereingaben bleiben selbst beim Beenden der Anwendung auf dem Smartphone gespeichert. Der Benutzer braucht nur auf einen Button drücken, um die Laderegelung zu starten. Die Anwendung reagiert sofort auf die Zustände der Bluetooth-Verbindung und informiert den Benutzer mithilfe des entsprechenden Hinweistextes auf der Benutzeroberfläche.

Jedoch gab es Schwächen wie es beim Testen festgestellt wurde im Hintergrund, weil der Anwendung die Information über den aktuellen Ladestatus und der Akkuladung gefehlt hatte. Ein weiterer Aspekt war, dass die Variablen im Zustand „onStop“ der Aktivität ihre Zuweisung verlieren, wenn sie davor nicht gespeichert werden. Durch die Verwendung von statischen Variablen und Preferencen konnte dieses Problem aufgehoben werden.

Aus diesem Grund wurde ein weiterer Thread in der Hauptaktivität initialisiert, um im Hintergrund fehlende Informationen durch Abtasten zu bekommen. Damit kann man für die Weiterentwicklung dieses Projektes die Anwendung komplett ohne Registrierung des Broadcastreceivers und unabhängig vom Lebenszyklus der Aktivität programmieren. Das wird für Android Version ab einem API-Level von 26 gelten. Bei dieser Anwendung war das Ziel, so viele Versionen wie möglich ansprechen, deshalb wurde in der „MainActivity“ für höhere Versionen der Thread gestartet und für niedrige Versionen der Broadcastreceiver registriert.

Das ist ein vielseitiges Projekt gewesen. Als Neueinsteiger in der Android Programmierung habe ich versucht eine gute Benutzeroberfläche zu gestalten, das ist mir bei Layouts Eigenschaften und Einstellungen gut gelungen. Die Hauptbenutzeroberfläche ist auch wie gewünscht einfach gehalten worden. Der Entwurf des Ladegerätes, die Kommunikation über Bluetooth Verbindung, Bestimmung der Verbindungszustände, Behandlung der Fehlerfälle und Gestaltung eines schönen Designs für die Benutzeroberflächen kosteten mich Zeit, um das Konzept der Laderegelung im Zeitraum der Bachelorarbeit zu erweitern.

# Abbildungsverzeichnis

# Literaturverzeichnis

[1] M. Baumeister, „Handyakku richtig laden,“ 16 11 2019. *[Online]. Available: https://handy.de/magazin/handyakku-richtig-laden-so-gehts/.*

[2] B. PREDAN-HALLABRIN, „Rooten Vorteile und Nachteile,“ *[Online]. Available: https://www.chip.de/artikel/android-rooten-vorteile-und-nachteile_118546. [Zugriff am 17 07 2019].*

[3] „What is API Level?: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/topics/manifest/uses-sdk-element#ApiLevels.*

[4] „App Manifest Overview: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.*

[5] „allowbackup: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/topics/manifest/application-element#allowbackup.*

[6] „Intent Filter: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/topics/manifest/intent-filter-element.*

[7] U. Post, Android-Apps entwickeln für Einsteiger, Bonn: Rheinwerk Verlag, 2015.

[8] „uses-sdk: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/topics/manifest/uses-sdk-element.*

[9] „Broadcasts overview: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/components/broadcasts.*

[10] A. Becker und M. Pant, Android 5 Programmieren für Smartphones und Tablets, Heidelberg: dpunkt.verlag, 2015.

[11] „BatteryManager: Dokumentation,“ *[Online]. Available: https://developer.android.com/reference/android/os/BatteryManager.*

[12] „Datenblatt,“ Finder, *[Online]. Available: https://asset.conrad.com/media10/add/160267/c1/-/de/000503243DS01/datenblatt-503243-finder-361190054011-printrelais-5-vdc-10-a-1-wechsler-1-st.pdf.*

[13] „Monitor the Battery Level and Charging State: Dokumentation,“ *[Online]. Available: https://developer.android.com/training/monitoring-device-state/battery-monitoring.*

[14] „Bluetooth overview: Dokumentation,“ *[Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth.*

[15] „Communicate with the UI thread: Dokumentation," *[Online]*. *Available: https://developer.android.com/training/multiple-threads/communicate-ui*.

[16] „Starting a background thread: coding in flow," *[Online]*. *Available: https://codinginflow.com/tutorials/android/starting-a-background-thread*.

[17] „ListPreference: Dokumentation," *[Online]*. *Available: https://developer.android.com/reference/android/preference/ListPreference.*

[18] „Translationseditor: Coding in Flow," *[Online]*. *Available: https://codinginflow.com/tutorials/android/translations-editor*.

[19] G. Agis, „chage app language," *[Online]*. *Available: https://blog.guillaumeagis.eu/change-language/.*

[20] W.-M. Lee, Android - Bausteine Sofort einsetzbare Code-Lösungen für anspruchvolle Apps, Weinheim: WILEY-VCH Verlag GmbH & Co.KGaA, 2013.

[21] „Universal Serial Bus," 17 Dezember 2019. *[Online]*. *Available: https://de.wikipedia.org/wiki/Universal_Serial_Bus#Pinouts_und_Adernfarben.*

# Anhang

## A: Erstellen von Ressourcen für die Layouts

### A1: Strings

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\values\strings.xml

```
 1 <resources>
 2     <string name="app_name" translatable="false">Smart
   Charge</string>
 3     <string name="strActionSettings">Settings</string>
 4     <string name="strActionProperties">Properties</string>
 5     <string name="strHealth">Battery Health</string>
 6     <string name="strTemperature">Battery Temperature</
   string>
 7     <string name="strTech">Battery Technology</string>
 8     <string name="strPlug">Power Source</string>
 9     <string name="strCapacity">Capacity</string>
10     <string name="level" translatable="false">Level</
   string>
11     <string name="strVoltage">Battery Voltage</string>
12     <string name="strCurrentNow">Current</string>
13     <string name="strGood">good</string>
14     <string name="strCold">cold</string>
15     <string name="strDead">dead</string>
16     <string name="strOverVoltage">over voltage</string>
17     <string name="strOverheat">over heat</string>
18     <string name="strUnknown">unknown</string>
19     <string name="strUnspecifiedFailure">UNSPECIFIED
   FAILURE</string>
20     <string name="AC" translatable="false">AC</string>
21     <string name="USB" translatable="false">USB</string>
22     <string name="WIRELESS" translatable="false">WIRELESS
   </string>
23     <string name="strCharging">Charging</string>
24     <string name="strIsCharging">Charging</string>
25     <string name="strNotCharging">Not Charging</string>
26     <string name="strDischarging">Discharging</string>
27     <string name="strChargingFull">Full</string>
28     <string name="start" translatable="false">START</
   string>
29     <string name="strNotPlugged">Not plugged</string>
30     <string name="strControlRange">Control Range:</string>
31     <string name="strSetPoint">Maximum Charge:</string>
32     <string name="strNoBltSelected">No bluetooth device
   selected</string>
33     <string name="strBltDisabled">Bluetooth is disabled.</
   string>
34     <string name="strSupportBlt">Device does not support
   Bluetooth</string>
35     <string name="strBltNotSelected">Not selected
   bluetooth device</string>
```

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\values\strings.xml

```
36      <string name="strAlreadyConnected">Device is already
    connected</string>
37      <string name="strConnectTo">Connecting to</string>
38      <string name="strButton">CONNECT</string>
39      <string name="strAvailableError">Check that the
    charger is connected correctly.</string>
40      <string-array name="language">
41          <item>ENGLISH</item>
42          <item>DEUTSCH</item>
43      </string-array>
44      <string name="strChooseDevice">Choose device</string>
45      <string name="strTvSeekBar">Below maximum charge</
    string>
46      <string name="strOtherDevice">Select charger </string>
47      <string name="strLanguage">App Language</string>
48      <string name="strRename">Rename Charger</string>
49      <string name="strRelPoint">Lower switching point:</
    string>
50      <string name="strDevice">Charger:</string>
51      <string name="strInstructions">
52                  1. Make sure that you have connected the
    device correctly.\n\n
53                  2. Pair the device in bluetooth settings.\
    n\n
54                  3. Open the app settings and select your
    device.\n\n
55                  4. Click CONNECT button.\n\n
56                  5. You can set the maximum charge and the
    lower switching point as desired.\n\n
57                  6. App keeps the capacity between target
    value and reference point.\n
58      </string>
59      <string name="strInstrTitle">Instructions</string>
60      <string name="strConnectedTo">CONNECTED TO CHARGER</
    string>
61      <string name="strFailedTo">CONNECTION FAILED TO
    CHARGER</string>
62      <string name="strTurnedOn">turned on</string>
63      <string name="strTurnedOff">turned off</string>
64      <string name="strNullEntryError">Pair the device in
    the Bluetooth settings beforehand</string>
65
66  </resources>
67
```

### A2: Menü für Smart Charge

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\menu\menu_main.xml

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/
   android"
2      xmlns:app="http://schemas.android.com/apk/res-auto"
3      xmlns:tools="http://schemas.android.com/tools"
4      tools:context="com.example.smartcharge.MainActivity">
5      <item
6          android:id="@+id/action_properties"
7          android:orderInCategory="100"
8          android:title="@string/strActionProperties"
9          app:showAsAction="never" />
10     <item
11         android:id="@+id/action_settings"
12         android:orderInCategory="100"
13         android:title="@string/strActionSettings"
14         app:showAsAction="never" />
15     <item
16         android:id="@+id/action_information"
17         android:title="Info"
18         android:icon="@drawable/ic_info_outline_black_24dp
   "
19         app:showAsAction="ifRoom"
20         ></item>
21 </menu>
22
```

### A3: Menü für Akkueigenschaften

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\menu\menu_properties.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/
   android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      tools:context="com.example.smartcharge.
   PropertiesActivity">
6      <item
7          android:id="@+id/action_home"
8          android:orderInCategory="100"
9          android:title="@string/app_name"
10         app:showAsAction="never" />
11
12     <item
13         android:id="@+id/action_settings"
14         android:orderInCategory="100"
15         android:title="@string/strActionSettings"
16         app:showAsAction="never" />
17
18 </menu>
```

### A4: Menü für Einstellungen

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\menu\menu_settings.xml

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/
   android"
2      xmlns:app="http://schemas.android.com/apk/res-auto"
3      xmlns:tools="http://schemas.android.com/tools"
4      tools:context="com.example.smartcharge.Settings">
5      <item
6          android:id="@+id/action_home"
7          android:orderInCategory="100"
8          android:title="@string/app_name"
9          app:showAsAction="never" />
10
11     <item
12         android:id="@+id/action_properties"
13         android:orderInCategory="100"
14         android:title="@string/strActionProperties"
15         app:showAsAction="never" />
16 </menu>
```

# B: Benutzeroberflächen XML

## B1: Layout Smart Charge

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/
   apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9
10     <ImageView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:src="@drawable/ic_bt_50_50"
14         android:id="@+id/ivBatterySymbol"
15         android:layout_centerInParent="true"
16         android:layout_alignParentLeft="true"
17         android:contentDescription="TODO"></ImageView>
18
19     <TextView
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:id="@+id/tvSetPoint"
23         android:layout_centerInParent="true"
24         android:layout_above="@id/btStart"
25         android:layout_toRightOf="@id/ivBatterySymbol"
26         android:text="@string/strSetPoint"
27         android:gravity="right"
28         android:textColor="@android:color/black"
29         android:textStyle="bold"
30         android:textSize="@dimen/mainTextSize"
31         ></TextView>
32
33     <TextView
34         android:layout_width="wrap_content"
35         android:layout_height="wrap_content"
36         android:id="@+id/tv4"
37         android:layout_centerInParent="true"
38         android:layout_toLeftOf="@+id/ivBatterySymbol"
39
40         ></TextView>
41
42     <TextView
43         android:layout_width="wrap_content"
44         android:layout_height="wrap_content"
```

```
45          android:id="@+id/tv3"
46          android:layout_centerInParent="true"
47          android:layout_toLeftOf="@+id/ivBatterySymbol"
48          android:layout_below="@id/tv4"
49          ></TextView>
50
51
52      <ImageView
53          android:layout_width="wrap_content"
54          android:layout_height="wrap_content"
55          android:id="@+id/ivIsCharging"
56          android:layout_toRightOf="@id/ivBatterySymbol"
57          android:layout_centerInParent="true"
58          android:src="@drawable/ic_flash_on_black_24dp"
59
60          >
61      </ImageView>
62      <TextView
63          android:layout_width="wrap_content"
64          android:layout_height="wrap_content"
65          android:id="@+id/tvRight1"
66          android:layout_centerInParent="true"
67          android:layout_toRightOf="@id/ivBatterySymbol"
68          android:layout_below="@id/ivIsCharging"
69          >
70      </TextView>
71
72
73
74      <TextView
75          android:id="@+id/tvConnected"
76          android:layout_width="match_parent"
77          android:layout_height="wrap_content"
78          android:layout_above="@id/ivBatterySymbol"
79          android:gravity="center"
80          android:textColor="@android:color/black"
81          android:layout_alignParentTop="true"
82
83          ></TextView>
84      <TextView
85          android:layout_width="wrap_content"
86          android:layout_height="wrap_content"
87          android:id="@+id/tvSetPointValue"
88          android:textColor="@android:color/black"
89          android:textSize="@dimen/mainTextSize"
```

```
90          android:textStyle="bold"
91          android:layout_marginLeft="6dp"
92          android:layout_centerInParent="true"
93          android:layout_above="@id/btStart"
94          android:layout_toRightOf="@id/tvSetPoint"
95          ></TextView>
96
97      <SeekBar
98          android:layout_width="wrap_content"
99          android:layout_height="wrap_content"
100         android:id="@+id/seekBar"
101         android:layout_toRightOf="@id/tv3"
102         android:layout_toLeftOf="@id/tvRight1"
103         android:layout_below="@id/tv4"
104         android:progress="100"
105         android:rotation="270"
106         ></SeekBar>
107
108     <TextView
109         android:layout_width="wrap_content"
110         android:layout_height="wrap_content"
111         android:id="@+id/tvPercentage"
112         android:layout_toRightOf="@id/tv3"
113         android:layout_toLeftOf="@id/tvRight1"
114         android:layout_below="@id/ivBatterySymbol"
115         android:gravity="center"
116         android:textColor="@android:color/black"
117         android:textSize="@dimen/mainTextSize"
118         ></TextView>
119
120     <TextView
121         android:layout_width="wrap_content"
122         android:layout_height="wrap_content"
123         android:id="@+id/tvDeviceName"
124         android:text="@string/strDevice"
125         android:gravity="left"
126         android:textColor="@android:color/black"
127         android:textStyle="bold"
128         android:textSize="@dimen/mainTextSize"
129         android:layout_toLeftOf="@id/btStart"
130         android:layout_alignParentStart="true"
131         ></TextView>
132     <TextView
133         android:layout_width="wrap_content"
134         android:layout_height="wrap_content"
```

```
135         android:layout_below="@id/tvDeviceName"
136         android:id="@+id/tvRelativePoint"
137         android:text="@string/strRelPoint"
138         android:textColor="@android:color/black"
139         android:textStyle="bold"
140         android:textSize="@dimen/mainTextSize"
141         android:gravity="left"
142         android:layout_marginTop="6dp"
143         ></TextView>
144
145     <Button
146         android:layout_width="wrap_content"
147         android:layout_height="wrap_content"
148         android:id="@+id/btStart"
149         android:layout_above="@id/ivIsCharging"
150         android:layout_toRightOf="@id/ivBatterySymbol"
151         android:layout_centerInParent="true"
152         android:text="@string/strButton"
153         android:textColor="@android:color/black"
154         android:textStyle="bold"
155         android:textSize="@dimen/mainTextSize"
156         android:onClick="startConnection"
157         ></Button>
158     <TextView
159         android:layout_width="wrap_content"
160         android:layout_height="wrap_content"
161         android:layout_centerInParent="true"
162         android:layout_below="@id/ivIsCharging"
163         android:layout_toRightOf="@id/ivBatterySymbol"
164         android:layout_marginTop="12dp"
165         android:textColor="@android:color/black"
166         android:id="@+id/tvCheckAvailable"
167         ></TextView>
168
169 </RelativeLayout>
```

## B2: Layout Akkueigenschaften

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\layout\activity_properties.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/
   apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".PropertiesActivity">
8      <ListView
9          android:id="@+id/listView"
10         android:layout_width="match_parent"
11         android:layout_height="match_parent"
12         android:layout_alignParentTop="true"
13         ></ListView>
14
15 </RelativeLayout>
```

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\layout\custom_listview.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="2dp">

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/cardview"
        app:cardCornerRadius="8dp"
        app:contentPadding="2dp"
        android:layout_margin="3dp"
        android:focusable="true"
        android:scrollbars="vertical"
        >
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="6dp"
            >
            <ImageView
                android:layout_width="@dimen/icon_width"
                android:layout_height="@dimen/icon_width"
                android:id="@+id/ivImage"
                android:layout_alignParentStart="true"
                android:layout_alignParentTop="true"
                android:src="@drawable/ic_favorite_black_24dp"

                ></ImageView>
            <TextView
                android:layout_width="@dimen/title_size"
                android:layout_height="wrap_content"
                android:id="@+id/tvTitle"
                android:textStyle="bold"
                android:textColor="@android:color/holo_green_dark"
                android:textSize="@dimen/tvTitle_textSize"
                android:text="@string/strTemperature"
                android:layout_alignParentTop="true"
                android:layout_toRightOf="@id/ivImage"
                ></TextView>
```

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\layout\custom_listview.xml

```
43              <TextView
44                  android:layout_width="@dimen/title_size"
45                  android:layout_height="wrap_content"
46                  android:id="@+id/tvDescription"
47                  android:text="@string/strGood"
48                  android:textSize="@dimen/
   tvDescription_textSize"
49                  android:textColor="@android:color/black"
50                  android:layout_toRightOf="@id/ivImage"
51                  android:layout_below="@id/tvTitle"
52                  ></TextView>
53          </RelativeLayout>
54
55      </androidx.cardview.widget.CardView>
56
57 </RelativeLayout>
58
59
```

## B3: Layout Einstellungen

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\layout\settings_activity.xml

```
1 <LinearLayout xmlns:android="http://schemas.android.com/
  apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent">
4
5     <FrameLayout
6         android:id="@+id/settings"
7         android:layout_width="match_parent"
8         android:layout_height="match_parent" />
9 </LinearLayout>
```

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\res\xml\root_preferences.xml

```
1
2  <PreferenceScreen xmlns:app="http://schemas.android.com/
   apk/res-auto"
3      xmlns:android="http://schemas.android.com/apk/res/
   android">
4
5      <ListPreference
6          app:title="@string/strOtherDevice"
7          app:key="device"
8          app:dialogTitle="@string/strChooseDevice"
9          app:defaultValue="0"
10         app:useSimpleSummaryProvider="true"
11         ></ListPreference>
12     <EditTextPreference
13         app:title="@string/strRename"
14         app:key="renamed_device"
15         app:dialogTitle="@string/strRename"
16         app:defaultValue="Hello"
17         app:useSimpleSummaryProvider="true"
18         ></EditTextPreference>
19     <SeekBarPreference
20         app:title="@string/strTvSeekBar"
21         android:max="6"
22         app:key="relative"
23         app:defaultValue="1"
24         app:useSimpleSummaryProvider="true"
25         ></SeekBarPreference>
26     <ListPreference
27         app:title="@string/strLanguage"
28         app:entries="@array/language"
29         app:entryValues="@array/language"
30         app:dialogTitle="@string/strLanguage"
31         app:defaultValue="ENGLISH"
32         app:key="language"
33         app:useSimpleSummaryProvider="true"
34         ></ListPreference>
35
36
37 </PreferenceScreen>
38
```

# C: Quellcode Java

## C1: Klasse Constants

```
1  package com.example.smartcharge;
2
3  import android.bluetooth.BluetoothAdapter;
4  import android.bluetooth.BluetoothDevice;
5  import android.content.res.Configuration;
6  import android.content.res.Resources;
7  import android.graphics.Color;
8  import android.util.DisplayMetrics;
9  import android.widget.TextView;
10
11 import java.util.Locale;
12 import java.util.UUID;
13
14 public class Constants {
15
16
   //-------------------------------------------------------
   -------------------------------------
17     //part to create a connection per bluetooth
18     public static BluetoothAdapter bluetoothAdapter; //
   BluetoothAdapter need for all bluetooth activities
19     public static BluetoothDevice bluetoothDevice; //
   BluetoothDevice need to create a connection
20     //UUID identifier for Bluetooth socket
21     public static final UUID uuidIdentifier = UUID.
   fromString("00001101-0000-1000-8000-00805F9B34FB");
22     public static final int intEnableBT=11; //Return value
    of bluetooth enable
23
24
   //-------------------------------------------------------
   -------------------------------------
25     //part to indicate connection states
26     public static final int intStateConnected=3; //
   Connection state connected
27     public static final int intStateConnectionFailed=4; //
   Connection state connection failed
28     public static final int intStateMessageRead=6; //
   Connection state reading incoming message
29     public static final int intStateMessageWrite=7; //
   Connection state writing message ton send
30     public static final int intStateWriteError=8; //
   Connection state ERROR writing message ton send
31     public static final int intStateReadError=9; //
   Connection state ERROR reading incoming message
```

```
32      public static final int intStateNone=10; //Connection
    state by initializing
33
34      //indicate connection state in code
35      public static final String strConnected="CONNECTED";
    //TextMessage Information CONNECTED
36      public static final String strFailed="CONNECT FAILED";
     //TextMessage Information CONNECT FAILED
37
38
    //-------------------------------------------------------
    -------------------------------------
39      //part Data Storage
40      //Data storage bluetooth attributes
41      public static final String strBltDeviceExtra="
    BLUETOOTH DEVICE"; //key contains device and mac
42      public static final String strBltDeviceName="BLUETOOTH
     DEVICE NAME"; //Data storage device name
43      public static final String strBltDeviceAddress="
    BLUETOOTH DEVICE ADDRESS"; //Data storage MAC_ADDRESS
44      //Data storage relative point attributes
45      public static final String strRelativeExtra="RELATIVE"
    ; //key contains value for relative point
46      public static final String strRelativePoint="RELATIVE
    POINT"; //Data storage RelativePoint
47      //Data storage SetPoint attributes
48      public static final String strSetPointExtra="SET POINT
    "; //key contains value for set point
49      public static final String strSetPoint="SetPoint"; //
    Data storage SetPoint
50
51
    //-------------------------------------------------------
    -------------------------------------
52      //part regulate capacity level
53      public static final int intStateCapacity=12; //State
    sample the capacity
54      public static final String strStartCharging = "e"; //
    Command to device start charging
55      public static final String strStopCharging = "a"; //
    Command to device stop charging
56
57
    //-------------------------------------------------------
    -------------------------------------
```

```
58      //part Language
59      public static String strLanguageMain; //Language of
    class main
60      public static String strLanguageProperties; //
    Language of class properties
61      public static String strLanguageSettings; //Language
    of class settings
62
63      //method sets corresponding language
64      //needs to access Resources files
65      //example switchLanguage(this.getResources(),
    language_PROPERTIES)
66      public static void switchLanguage(Resources resources
    ,String locale){
67
68          DisplayMetrics metrics= resources.
    getDisplayMetrics(); //access class DisplayMetrics
69          Configuration configuration= resources.
    getConfiguration(); //access class Configuration
70          switch (locale){
71              case "ENGLISH":
72                  configuration.setLocale(new Locale("en"))
    ; // ,,en'' key for english
73                  break;
74              case "DEUTSCH":
75                  configuration.setLocale(new Locale("de"))
    ; // ,,de'' key for german
76                  break;
77              default:
78                  configuration.setLocale(new Locale("en"))
    ; // default language english
79                  break;
80          }
81          resources.updateConfiguration(configuration,
    metrics); //refresh values
82      }
83
84
    //-----------------------------------------------------
    -------------------------------------
85      //part battery image
86      //Array contains battery symbols
87      public static final int[] intSymbolArray = {R.
    drawable.ic_bt_00_00, R.drawable.ic_bt_10_10,
88              R.drawable.ic_bt_20_20, R.drawable.
```

```
 88 ic_bt_30_30, R.drawable.ic_bt_40_40,
 89             R.drawable.ic_bt_50_50, R.drawable.
    ic_bt_60_60, R.drawable.ic_bt_70_70,
 90             R.drawable.ic_bt_80_80, R.drawable.
    ic_bt_90_90};
 91
 92     //this method divides whole capacity into ten areas
 93     //parameter level is the capacity level in percentage
 94     //return corresponding area of battery images
 95     public static int levelArea(int level) {
 96         int answer = -1;
 97
 98         if (level <= 10) {
 99             answer = 0;
100
101         } else if (level > 10 && level <= 20) {
102             answer = 1;
103
104         } else if (level > 20 && level <= 30) {
105             answer = 2;
106
107         } else if (level > 30 && level <= 40) {
108             answer = 3;
109
110         } else if (level > 40 && level <= 50) {
111             answer = 4;
112
113         } else if (level > 50 && level <= 60) {
114             answer = 5;
115
116         } else if (level > 60 && level <= 70) {
117             answer = 6;
118
119         } else if (level > 70 && level <= 80) {
120             answer = 7;
121
122         } else if (level > 80 && level <= 90) {
123             answer = 8;
124         } else if (level > 90) {
125             answer = 9;
126         }
127
128         return answer;
129
130     }
```

```
131
132     //this method divides whole capacity into seven areas
    for using TextViews
133     //parameter value is level in percentage
134     //returns the area of level on battery image
135     public static int check_area(int value){
136         int answer =-1;
137         if(value<12){
138             answer=0;
139         }else if(value>=12 && value<=24){
140             answer=1;
141         }else if(value>24 && value <=36){
142             answer=2;
143         }else if(value>36 && value <=48 ){
144             answer=3;
145         }else if(value >48 && value <=60){
146             answer=4;
147         }else if(value >60 && value <=72){
148             answer=5;
149         }else if(value >72 && value <=84){
150             answer=6;
151         }else if(value >84){
152             answer=7;
153         }
154         return answer;
155     }
156 }
157
```

## C2: Klasse InfoFragment

File - C:\Users\Harun\AndroidStudioProjects\SmartCharge\app\src\main\java\com\example\smartcharge\InfoFragment.java

```java
package com.example.smartcharge;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

public class InfoFragment extends DialogFragment {
    @NonNull
    @Override
    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {

        AlertDialog.Builder builder= new AlertDialog.Builder(getActivity());
        builder.setTitle(R.string.strInstrTitle);
        builder.setMessage(getString(R.string.strInstructions)
        );
        builder.setCancelable(false);
        builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {

            }
        });

        AlertDialog alertDialog=builder.create();

        return alertDialog;
    }
}

```

### C3: Aktivität MainActivity

```
1  package com.example.smartcharge;
2
3  //-----------------------------------------------------
4  //Author:   Emrah Demir
5  //Date:     09.01.2020
6  //Project:  Application to improve the charge control in
   smartphones
7  //This work includes an application that keeps the battery
    charge
8  // of a smartphone between a maximum and a lower switching
    value
9  // according to the user's specifications using an
   external switch
10 //-----------------------------------------------------
11
12 import androidx.annotation.NonNull;
13 import androidx.appcompat.app.AppCompatActivity;
14 import androidx.fragment.app.FragmentManager;
15 import androidx.preference.PreferenceManager;
16
17 import android.bluetooth.BluetoothAdapter;
18 import android.bluetooth.BluetoothDevice;
19 import android.bluetooth.BluetoothSocket;
20 import android.content.BroadcastReceiver;
21 import android.content.Context;
22 import android.content.Intent;
23 import android.content.IntentFilter;
24 import android.content.SharedPreferences;
25 import android.graphics.Color;
26 import android.os.BatteryManager;
27 import android.os.Build;
28 import android.os.Bundle;
29 import android.os.Handler;
30 import android.os.Message;
31 import android.util.Log;
32 import android.view.Menu;
33 import android.view.MenuItem;
34 import android.view.View;
35 import android.widget.Button;
36 import android.widget.ImageView;
37 import android.widget.SeekBar;
38 import android.widget.TextView;
39 import android.widget.Toast;
40
41 import java.io.IOException;
```

```
42 import java.io.InputStream;
43 import java.io.OutputStream;
44
45
46 import static com.example.smartcharge.Constants.
   intStateCapacity;
47 import static com.example.smartcharge.Constants.
   strBltDeviceAddress;
48 import static com.example.smartcharge.Constants.
   strBltDeviceExtra;
49 import static com.example.smartcharge.Constants.
   intEnableBT;
50 import static com.example.smartcharge.Constants.
   strLanguageMain;
51 import static com.example.smartcharge.Constants.
   strRelativeExtra;
52 import static com.example.smartcharge.Constants.
   strRelativePoint;
53 import static com.example.smartcharge.Constants.
   intStateConnected;
54 import static com.example.smartcharge.Constants.
   intStateConnectionFailed;
55 import static com.example.smartcharge.Constants.
   intStateMessageRead;
56 import static com.example.smartcharge.Constants.
   intStateMessageWrite;
57 import static com.example.smartcharge.Constants.
   intStateNone;
58 import static com.example.smartcharge.Constants.
   intStateReadError;
59 import static com.example.smartcharge.Constants.
   intStateWriteError;
60 import static com.example.smartcharge.Constants.
   strSetPoint;
61 import static com.example.smartcharge.Constants.
   strSetPointExtra;
62 import static com.example.smartcharge.Constants.
   bluetoothAdapter;
63 import static com.example.smartcharge.Constants.
   bluetoothDevice;
64 import static com.example.smartcharge.Constants.
   intSymbolArray;
65 import static com.example.smartcharge.Constants.
   strBltDeviceName;
66 import static com.example.smartcharge.Constants.
```

```
66 strConnected;
67 import static com.example.smartcharge.Constants.strFailed
   ;
68 import static com.example.smartcharge.Constants.
   strStartCharging;
69 import static com.example.smartcharge.Constants.
   strStopCharging;
70 import static com.example.smartcharge.Constants.
   switchLanguage;
71 import static com.example.smartcharge.Constants.
   uuidIdentifier;
72
73 public class MainActivity extends AppCompatActivity {
74     public static final String TAG=".MainActivity";
75
76
   //-----------------------------------------------------
   -------------------------------------
77     //part manage connection per bluetooth
78     public static ConnectThread connectThread; //instance
    of class ConnectThread
79     public static ConnectedThread connectedThread; //
   instance of class ConnectedThread
80
81
   //-----------------------------------------------------
   -------------------------------------
82     //part define global variables
83     static int intState = intStateNone; //defines
   connecting state
84     static String strTvState=""; //connecting state for
   user
85     private String strReadMessage="";//global variable
   for read message in inputStream
86     volatile boolean blStopThread=false; //stop Thread
   class SampleCapacityThread
87
88
   //-----------------------------------------------------
   -------------------------------------
89     //part initialize layout widgets
90     private static TextView tvConnectionState;
91     private static TextView tvDeviceName;
92     private static TextView tvRelPoint;
93     private static TextView tvCheckAvailable;
```

```
94      private static TextView tvSetPointValue;
95      private SeekBar seekBar;
96      private static Button btStart;
97
98
    //-------------------------------------------------------
    ---------------------------------------
99      //Use Preferences for Data Storage
100     private SharedPreferences prefSetPoint;
101     private SharedPreferences prefBltDevice;
102     private SharedPreferences prefRelativePoint;
103     private SharedPreferences.Editor editorSetPoint;
104     private static boolean isCharging=false;
105
106
    //-------------------------------------------------------
    ---------------------------------------
107     //BroadcastReceiver to update Battery Level, Power
    Source and Battery symbol
108     private BroadcastReceiver receiverBatteryState =new
    BroadcastReceiver() {
109
110
    //-------------------------------------------------------
    ---------------------------------
111         //define layout widgets
112         private ImageView ivBattery;
113         private TextView tvPercentage;
114         private ImageView ivIsCharging;
115
116
    //-------------------------------------------------------
    ---------------------------------
117         //receiving current values
118         @Override
119         public void onReceive(Context context, Intent
    intent) {
120
121
    //-------------------------------------------------------
    ------------------------------
122             //initialize layout widgets
123             tvPercentage = ((MainActivity) context).
    findViewById(R.id.tvPercentage);
124             ivBattery = ((MainActivity) context).
```

```
124 findViewById(R.id.ivBatterySymbol);
125             ivIsCharging = ((MainActivity) context).
    findViewById(R.id.ivIsCharging);
126
127
    //-------------------------------------------------------
    ------------------------------
128             String action = intent.getAction();
129             if (action != null && action.equals(Intent.
    ACTION_BATTERY_CHANGED)) {
130
131
    //-------------------------------------------------------
    --------------------------
132                 //determine the current battery level
133                 int level = intent.getIntExtra(
    BatteryManager.EXTRA_LEVEL, -1);
134                 int scale = intent.getIntExtra(
    BatteryManager.EXTRA_SCALE, -1);
135                 int intCurrentBatteryLevel = (int) ((
    level / (float) scale) * 100);
136                 Log.d(TAG, "onReceive: "+
    intCurrentBatteryLevel);
137                 //monitoring current battery level on
    layout
138                 tvPercentage.setText(String.valueOf(
    intCurrentBatteryLevel) +"%");
139
140
    //-------------------------------------------------------
    --------------------------
141                 int intStatus = intent.getIntExtra(
    BatteryManager.EXTRA_STATUS, -1);
142
143                 isCharging = intStatus == BatteryManager.
    BATTERY_STATUS_CHARGING ||
144                         intStatus == BatteryManager.
    BATTERY_STATUS_FULL;
145
146                 if(isCharging){
147
148                     ivIsCharging.setImageDrawable(
    getResources().getDrawable(R.drawable.
    ic_flash_on_black_24dp));
149
```

```
150                 }else{
151
152                     ivIsCharging.setImageDrawable(
    getResources().getDrawable(R.drawable.
    ic_flash_off_black_24dp));
153                 }
154
155                 switch (intStatus) {
156                     case BatteryManager.
    BATTERY_STATUS_CHARGING:
157                         ivBattery.setImageDrawable(
    getResources().getDrawable(intSymbolArray[Constants.
    levelArea(intCurrentBatteryLevel)]));
158                         break;
159                     case BatteryManager.
    BATTERY_STATUS_DISCHARGING:
160                         ivBattery.setImageDrawable(
    getResources().getDrawable(intSymbolArray[Constants.
    levelArea(intCurrentBatteryLevel)]));
161                         break;
162                     case BatteryManager.
    BATTERY_STATUS_FULL:
163                         ivBattery.setImageDrawable(
    getResources().getDrawable(intSymbolArray[Constants.
    levelArea(intCurrentBatteryLevel)]));
164                         break;
165                     case BatteryManager.
    BATTERY_STATUS_NOT_CHARGING:
166                         ivBattery.setImageDrawable(
    getResources().getDrawable(intSymbolArray[Constants.
    levelArea(intCurrentBatteryLevel)]));
167                         break;
168                     case BatteryManager.
    BATTERY_STATUS_UNKNOWN:
169                         ivBattery.setImageDrawable(
    getResources().getDrawable(intSymbolArray[Constants.
    levelArea(intCurrentBatteryLevel)]));
170                         break;
171                 }
172
173
    //-------------------------------------------------------
    --------------------------
174                 //creates a new regulate in
    BroadcastReceiver if build version below LOLLIPOP
```

```
175                 //can sample current battery level only
    in foreground
176                 if (Build.VERSION.SDK_INT < Build.
    VERSION_CODES.LOLLIPOP){
177
178
    //-------------------------------------------------------
    ----------------------
179                     //reads saved setPoint value
180                     prefSetPoint = getSharedPreferences(
    strSetPointExtra, Context.MODE_PRIVATE);
181                     int intSetPoint=prefSetPoint.getInt(
    strSetPoint,90);
182
183
    //-------------------------------------------------------
    ----------------------
184                     //reads saved relativePoint value
185                     prefRelativePoint=
    getSharedPreferences(strRelativeExtra, Context.
    MODE_PRIVATE);
186                     int intRelativePoint=
    prefRelativePoint.getInt(strRelativePoint,3);
187
188                     Log.d(TAG, "onReceive: "+
    intCurrentBatteryLevel+ "setPoint "+intSetPoint+"" +
189                             " relativePoint "+
    intRelativePoint);
190
191
    //-------------------------------------------------------
    ----------------------
192                     //begin regulate if connection
    successful
193                     if(strTvState == strConnected){
194
195
    //-------------------------------------------------------
    ------------------
196                         //sends message to µC to stop
    charging
197                         if (isCharging &&
    intCurrentBatteryLevel >= intSetPoint) {
198
199                             sendMessage(strStopCharging);
```

```
200                         }
201
202
    //-------------------------------------------------------
    ------------------
203                         //sends message to µC to start
    charging
204                         if (!isCharging &&
    intCurrentBatteryLevel <= intSetPoint - intRelativePoint)
     {
205
206                             sendMessage(strStartCharging)
    ;
207
208                         }
209
210
    //-------------------------------------------------------
    ------------------
211                         if(!strReadMessage.equals("")){
212
213                             //return value of µC if
    charging start
214                             if(strReadMessage.equals("X")
    ){
215
216                                 if(!isCharging){
217
218                                     tvCheckAvailable.
    setText(R.string.strAvailableError);
219
220                                 }else{
221
222                                     tvCheckAvailable.
    setText(R.string.strTurnedOn);
223                                 }
224
225                             }
226
227                             //return value of µC if
    charging stop
228                             if(strReadMessage.equals("Y")
    ){
229
230                                 if(isCharging){
```

```
231
232                                     tvCheckAvailable.
    setText(R.string.strAvailableError);
233
234                                 }else{
235
236                                     tvCheckAvailable.
    setText(R.string.strTurnedOff);
237                                 }
238
239
240                             }
241
242                         }
243
244                         //test signal
245                         if(strReadMessage.equals("hello")
    ){
246
247                             sendMessage("OK");
248                         }
249                     }
250
251                 }
252
253             }
254
255         }
256
257     };
258
259
    //-------------------------------------------------------
    ---------------------------------------
260     // Activity lifecycle method onCreate
261     @Override
262     protected void onCreate(Bundle savedInstanceState) {
263         super.onCreate(savedInstanceState);
264
265
    //-------------------------------------------------------
    ----------------------------------
266         //changes language before the activity is in
    foreground
267         SharedPreferences sharedPref = PreferenceManager.
```

```
267 getDefaultSharedPreferences(this);
268         strLanguageMain = sharedPref.getString("language"
    , "ENGLISH");
269         switchLanguage(this.getResources(),
    strLanguageMain);
270 
271         //sets layout for MainActivity
272         setContentView(R.layout.activity_main);
273 
274         //need for all bluetooth events
275         bluetoothAdapter=BluetoothAdapter.
    getDefaultAdapter();
276 
277         //initialize UI-Widgets
278         initialize();
279 
280         //method shows chargerName and lower switching
    point on layout
281         showNameRelative();
282 
283         Log.d(TAG, "onCreate: ");
284 
285 
    //-----------------------------------------------------
    ---------------------------------
286         //receive saved  relativePoint from preference
287         prefRelativePoint = getSharedPreferences(
    strRelativeExtra, Context.MODE_PRIVATE);
288         final int intRelativePoint = prefRelativePoint.
    getInt(strRelativePoint,3);
289 
290 
    //-----------------------------------------------------
    ---------------------------------
291         //Listener to receive seekBar Value and store in
    preference strSetPoint
292         seekBar.setOnSeekBarChangeListener(new SeekBar.
    OnSeekBarChangeListener() {
293             @Override
294             public void onProgressChanged(SeekBar seekBar
    , int progress, boolean fromUser) {
295                 //regulate only when current battery
    level above 12%
296                 if(progress<20){
297 
```

```
298                     editorSetPoint.putInt(strSetPoint,20)
    ;
299                     tvSetPointValue.setText(Integer.
    toString(20)+"%");
300                     int intLowerSwitchingPoint = 20 -
    intRelativePoint;
301                     tvRelPoint.setText(getString(R.string
    .strRelPoint)+" "+intLowerSwitchingPoint+"%");
302
303                 }else{
304
305                     editorSetPoint.putInt(strSetPoint,
    progress);
306                     tvSetPointValue.setText(Integer.
    toString(progress)+"%");
307                     int intLowerSwitchingPoint = progress
     - intRelativePoint;
308                     tvRelPoint.setText(getString(R.string
    .strRelPoint)+" "+intLowerSwitchingPoint+"%");
309                 }
310
311                 editorSetPoint.apply(); //setPoint value
    saved here
312
313             }
314
315             @Override
316             public void onStartTrackingTouch(SeekBar
    seekBar) {
317
318             }
319
320             @Override
321             public void onStopTrackingTouch(SeekBar
    seekBar) {
322
323             }
324         });
325
326
    //-----------------------------------------------------
    ----------------------------------
327         //register BroadCastReceiver
328         //receives data from Intent
    ACTION_BATTERY_CHANGED
```

```
329
330         registerReceiver(receiverBatteryState,new
    IntentFilter(Intent.ACTION_BATTERY_CHANGED));
331
332     }
333
334
    //-------------------------------------------------------
    ---------------------------------------
335     //method initializes layout widgets
336     public void initialize(){
337
338         tvConnectionState = findViewById(R.id.tvConnected
    );
339         tvDeviceName = findViewById(R.id.tvDeviceName);
340         tvRelPoint = findViewById(R.id.tvRelativePoint);
341         tvSetPointValue = findViewById(R.id.
    tvSetPointValue);
342         tvCheckAvailable = findViewById(R.id.
    tvCheckAvailable);
343         seekBar  =findViewById(R.id.seekBar);
344         btStart = findViewById(R.id.btStart);
345
346     }
347
348
    //-------------------------------------------------------
    ---------------------------------------
349     //method shows chargerName and lower switching point
    on layout
350     public void showNameRelative(){
351
    //-------------------------------------------------------
    ---------------------------------
352         //receive saved setPoint from preference
353         prefSetPoint = getSharedPreferences(
    strSetPointExtra, Context.MODE_PRIVATE);
354         editorSetPoint = prefSetPoint.edit();
355         int intSetPoint = prefSetPoint.getInt(strSetPoint
    ,90);
356
357
    //-------------------------------------------------------
    ---------------------------------
358         //receive saved  relativePoint from preference
```

```
359         prefRelativePoint = getSharedPreferences(
    strRelativeExtra, Context.MODE_PRIVATE);
360         int intRelativePoint = prefRelativePoint.getInt(
    strRelativePoint,3);
361
362
    //-------------------------------------------------------
    ---------------------------------
363         // receive saved device name
364         prefBltDevice = getSharedPreferences(
    strBltDeviceExtra, Context.MODE_PRIVATE);
365         String name = prefBltDevice.getString(
    strBltDeviceName,null);
366
367
368         int intLowerSwitchingPoint = intSetPoint -
    intRelativePoint;
369
370         seekBar.setProgress(intSetPoint);
371         tvSetPointValue.setText(Integer.toString(
    intSetPoint)+ "%");
372
373         tvDeviceName.setText(getString(R.string.strDevice
    )+" "+name);
374         tvRelPoint.setText(getString(R.string.strRelPoint
    )+" "+intLowerSwitchingPoint+"%");
375     }
376
377
    //-------------------------------------------------------
    ---------------------------------
378     // method sends message to the charging device
379     public void sendMessage(String message){
380
381         if(strTvState == strConnected){
382             connectedThread.write(message.getBytes());
383         }else{
384             Log.d(TAG, "sendMessage: connect failed");
385         }
386
387     }
388
389
    //-------------------------------------------------------
    ---------------------------------
```

```
390     //Activity lifecycle method onCreate
391     //sets chosen language before activity is in
    foreground
392     //restores TextViews values for connection states
393     @Override
394     protected void onPostResume() {
395         super.onPostResume();
396         Log.d(TAG, "onPostResume: ");
397 
398         if(strTvState == strConnected){
399 
400             tvConnectionState.setText(R.string.
    strConnectedTo);
401             //make button CONNECT invisible
402             btStart.setVisibility(View.INVISIBLE);
403 
404 
405         }else if(strTvState == strFailed){
406 
407             tvConnectionState.setText(R.string.
    strFailedTo);
408 
409             //enable CONNECT button if connection failed
410             btStart.setVisibility(View.VISIBLE);
411             btStart.setEnabled(true);
412             btStart.setBackgroundColor(Color.YELLOW);
413 
414         }else{
415 
416             tvConnectionState.setText("");
417         }
418 
419 
420         Log.d(TAG, "onPostResume: "+strLanguageMain);
421         SharedPreferences sharedPref = PreferenceManager.
    getDefaultSharedPreferences(this);
422 
423         String strChosenLanguage = sharedPref.getString("
    language", "ENGLISH");
424 
425         if(!strLanguageMain.equals(strChosenLanguage)){
426             Log.d(TAG, "onPostResume: language changed ")
    ;
427             strChosenLanguage = strLanguageMain;
428             switchLanguage(this.getResources(),
```

```
428 strChosenLanguage);
429             recreate();
430         }
431     }
432
433
    //-------------------------------------------------------
    ---------------------------------------
434     //Activity lifecycle method onCreate
435     //unregisterReceiver BroadcastReceiver when the app
    is destroyed.
436     @Override
437     protected void onDestroy() {
438         super.onDestroy();
439
440         unregisterReceiver(receiverBatteryState);
441     }
442
443
    //-------------------------------------------------------
    ---------------------------------------
444     //method starts a new connectedThread
445     //parameter socket is created in Thread connectThread
446     public void manageMyConnectedSocket(BluetoothSocket
    socket){
447
448         if(connectedThread != null){
449             connectedThread.cancel();
450             connectedThread=null;
451         }
452         connectedThread= new ConnectedThread(socket);
453         connectedThread.start();
454     }
455
456
    //-------------------------------------------------------
    ---------------------------------------
457     //method creates a new bluetoothDevice
458     //parameter mac_address is the MAC ADDRESS of
    charging device
459     public void connectToDevice(String mac_address){
460         if(connectThread !=null){
461             connectThread.cancel();
462             connectThread=null;
463         }
```

```
464         if(connectedThread != null){
465             connectedThread.cancel();
466             connectedThread=null;
467         }
468         bluetoothDevice=bluetoothAdapter.getRemoteDevice(
    mac_address);
469         connectThread= new ConnectThread(bluetoothDevice)
    ;
470         connectThread.start();
471     }
472
473
    //-------------------------------------------------------
    ---------------------------------------
474     //part menu
475     @Override
476     public boolean onCreateOptionsMenu(Menu menu) {
477         // Inflate the menu; this adds items to the
    action bar if it is present.
478         getMenuInflater().inflate(R.menu.menu_main, menu)
    ;
479         return true;
480     }
481
482     @Override
483     public boolean onOptionsItemSelected(MenuItem item) {
484         // Handle action bar item clicks here. The action
     bar will
485         // automatically handle clicks on the Home/Up
    button, so long
486         // as you specify a parent activity in
    AndroidManifest.xml.
487         int id = item.getItemId();
488
489         //noinspection SimplifiableIfStatement
490         if (id == R.id.action_settings) {
491             Intent intent = new Intent(this,
    SettingsActivity.class);
492             startActivity(intent);
493             return true;
494         }else if (id == R.id.action_properties) {
495             Intent intent = new Intent(this,
    PropertiesActivity.class);
496             startActivity(intent);
497             return true;
```

```
498         }else if (id == R.id.action_information) {
499             FragmentManager manager =
    getSupportFragmentManager();
500             InfoFragment infoFragment= new InfoFragment()
    ; //Information is a DialogFragment
501             infoFragment.show(manager,"Information");
502 
503             return true;
504         } else {
505 
506             return super.onOptionsItemSelected(item);
507         }
508     }
509 
510 
    //-----------------------------------------------------
    ---------------------------------------
511     //Thread measures current battery level every two
    seconds
512     //can sample current battery level both in the
    background and in the foreground
513     //build version must be higher than LOLLIPOP
514     private class SampleCapacityThread extends Thread{
515 
516         @Override
517         public void run() {
518 
519             while(!blStopThread){
520 
521                 BatteryManager batteryManager=(
    BatteryManager)getSystemService(Context.BATTERY_SERVICE);
522                 if (Build.VERSION.SDK_INT > Build.
    VERSION_CODES.LOLLIPOP) {
523 
524                     int currentBatteryLevel =
    batteryManager.getIntProperty(BatteryManager
525                             .BATTERY_PROPERTY_CAPACITY);
526 
527 
528                     Message writtenMsg =
    handlerControlCapacity.obtainMessage(
529                             intStateCapacity, -1, -1,
    currentBatteryLevel);
530                     writtenMsg.sendToTarget();
531 
```

```
532                 }
533                 Log.d(TAG, "run: SampleCapacityThread");
534                 try {
535
536                     Thread.sleep(3000);
537
538                 } catch (InterruptedException e) {
539
540                     e.printStackTrace();
541                 }
542             }
543         }
544     }
545
546
547
    //-------------------------------------------------------
    ------------------------------------
548     //Method starts the connection when the START button
    is clicked
549     //if connection was successful, regulates battery
    level
550     public void startConnection(View view) {
551
552
    //-------------------------------------------------------
    ---------------------------------
553         //receive saved chargerName and macAddress from
    preference
554         prefBltDevice = getSharedPreferences(
    strBltDeviceExtra, Context.MODE_PRIVATE);
555         String address = prefBltDevice.getString(
    strBltDeviceAddress, null);
556         String name=prefBltDevice.getString(
    strBltDeviceName,null);
557
558         //boolean for SampleCapacityThread
559         blStopThread = false;
560
561
    //-------------------------------------------------------
    ---------------------------------
562         if(bluetoothAdapter==null){ //check if the
    smartPhone support bluetooth
563
```

```
564             Toast.makeText(this,R.string.strSupportBlt,
    Toast.LENGTH_SHORT).show();
565
566         }else{ //bluetooth is enabled
567
568             if(bluetoothAdapter.isEnabled()){ //check if
    bluetooth is enable
569
570                 if (address == null || "".equals(address)
    ) {//not selected bluetooth device
571
572                     Toast.makeText(this,R.string.
    strBltNotSelected,Toast.LENGTH_SHORT).show();
573
574                 }else { // device is selected
575
576                     //if smartPhone connected to charger
577                     if (intState == intStateConnected ||
    intState == intStateMessageRead || intState ==
    intStateMessageWrite) {
578
579                         Toast.makeText(this, R.string.
    strAlreadyConnected, Toast.LENGTH_SHORT).show();
580
581                     } else { //smartPhone is not
    connected to charger or state is none
582
583                         //begins connection
584                         connectToDevice(address);
585
586                         String identifier1= (name !=null
    )? name: address;
587                         Toast.makeText(this,getString(R.
    string.strConnectTo)+" "+identifier1,Toast.LENGTH_SHORT).
    show();
588
589                         //Thread begins sampling current
    battery level every three seconds
590                         if (Build.VERSION.SDK_INT > Build
    .VERSION_CODES.LOLLIPOP) {
591                             SampleCapacityThread
    sampleCapacityThread = new SampleCapacityThread();
592                             sampleCapacityThread.start();
593                         }
594                     }
```

```
595                 }
596             }else{ //if start button clicked and bluetoot
     is disabled
597 
598                 //user enables bluetooth
599                 Intent itf=new Intent(BluetoothAdapter.
    ACTION_REQUEST_ENABLE);
600                 startActivityForResult(itf,intEnableBT);
601 
602             }
603         }
604     }
605 
606 
    //-----------------------------------------------------
    -------------------------------------
607     //Thread generates from BluetoothDevice per UUID
    Identifier a new BluetoothSocket
608     //Source
609     //https://developer.android.com/guide/topics/
    connectivity/bluetooth
610     private class ConnectThread extends Thread {
611         private final BluetoothSocket mmSocket;
612         private final BluetoothDevice mmDevice;
613 
614 
    //-----------------------------------------------------
    ---------------------------------
615         //Constructor
616         public ConnectThread(BluetoothDevice device) {
617             // Use a temporary object that is later
    assigned to mmSocket
618             // because mmSocket is final.
619             BluetoothSocket tmp = null;
620 
621             mmDevice = device;
622 
623             try {
624                 // Get a BluetoothSocket to connect with
    the given BluetoothDevice.
625                 // MY_UUID is the app's UUID string, also
     used in the server code.
626                 tmp = device.
    createRfcommSocketToServiceRecord(uuidIdentifier);
627 
```

```
628             } catch (IOException e) {
629                 //Socket's create() method failed
630                 Log.e(TAG, "Socket's create() method
    failed", e);
631             }
632
633             mmSocket = tmp;
634         }
635
636         public void run() {
637             // Cancel discovery because it otherwise
    slows down the connection.
638             bluetoothAdapter.cancelDiscovery();
639
640             try {
641                 // Connect to the remote device through
    the socket. This call blocks
642                 // until it succeeds or throws an
    exception.
643                 mmSocket.connect();
644
645
    //-----------------------------------------------------
    --------------------------
646                 //sends message state connected to
    handler
647                 Message messageConnected = Message.obtain
    ();
648                 messageConnected.what = intStateConnected
    ;
649                 handlerConnectionState.sendMessage(
    messageConnected);
650
651                 Log.d(TAG, "run: connected");
652             } catch (IOException connectException) {
653                 // Unable to connect; close the socket
    and return.
654
655
    //-----------------------------------------------------
    --------------------------
656                 //sends message state connection failed
    to handler
657                 Message messageConnectionFailed = Message
    .obtain();
```

```
658                 messageConnectionFailed.what =
    intStateConnectionFailed;
659                 handlerConnectionState.sendMessage(
    messageConnectionFailed);
660
661                 Log.d(TAG, "run: connect failed");
662                 try {
663                     //closes the socket if connection
    failed
664                     mmSocket.close();
665
666                 } catch (IOException closeException) {
667                     //Exception could not close the
    client socket
668                     Log.e(TAG, "Could not close the
    client socket", closeException);
669                 }
670                 return;
671             }
672             // The connection attempt succeeded. Perform
    work associated with
673             // the connection in a separate thread.
674             manageMyConnectedSocket(mmSocket);
675         }
676
677         // Closes the client socket and causes the thread
     to finish.
678         public void cancel() {
679             try {
680                 //closes the socket when this method
    called
681                 mmSocket.close();
682
683             } catch (IOException e) {
684                 //Exception could not close the client
    socket
685                 Log.e(TAG, "Could not close the client
    socket", e);
686             }
687         }
688     }
689
690
    //-----------------------------------------------------
    -------------------------------------
```

```
691     //Thread manages InputStream, OutputStream and
    Connection
692     //Source
693     //https://developer.android.com/guide/topics/
    connectivity/bluetooth
694     private class ConnectedThread extends Thread {
695         private final BluetoothSocket mmSocket;
696         private final InputStream mmInStream;
697         private final OutputStream mmOutStream;
698         private byte[] mmBuffer; // mmBuffer store for
    the stream
699
700
    //-----------------------------------------------------
    ---------------------------------
701         //Constructor
702         public ConnectedThread(BluetoothSocket socket) {
703
704             mmSocket = socket;
705             InputStream tmpIn = null;
706             OutputStream tmpOut = null;
707
708             // Get the input and output streams; using
    temp objects because
709             // member streams are final.
710             try {
711                 //reads InputStream and stores in
    temporary variable
712                 tmpIn = socket.getInputStream();
713
714             } catch (IOException e) {
715                 //Exception Error occurred when creating
    input stream
716                 Log.e(TAG, "Error occurred when creating
    input stream", e);
717             }
718             try {
719                 //reads OutputStream stores in temporary
    variable
720                 tmpOut = socket.getOutputStream();
721
722             } catch (IOException e) {
723                 //Exception Error occurred when creating
    output stream
724                 Log.e(TAG, "Error occurred when creating
```

```
724 output stream", e);
725             }
726
727
    //-------------------------------------------------------
    ------------------------------
728             // Value assignment from temporary variables
729             mmInStream = tmpIn;
730             mmOutStream = tmpOut;
731
732         }
733
734         public void run() {
735
736             mmBuffer = new byte[1024];
737             int numBytes; // bytes returned from read()
738
739
    //-------------------------------------------------------
    ------------------------------
740             // Keep listening to the InputStream until an
     exception occurs.
741             while (true) {
742                 try {
743                     // Read from the InputStream.
744                     numBytes = mmInStream.read(mmBuffer);
745
746
    //-------------------------------------------------------
    ----------------------
747                     // Send the obtained bytes to the UI
    activity.
748                     Message messageRead =
    handlerConnectionState.obtainMessage(
749                             intStateMessageRead, numBytes
    , -1,
750                             mmBuffer);
751                     messageRead.sendToTarget();
752
753                     Log.d(TAG, "run: READ");
754
755                 } catch (IOException e) {
756
    //-------------------------------------------------------
    ----------------------
```

```
757                     Message messageReadError = Message.
    obtain();
758                     messageReadError.what =
    intStateReadError;
759                     handlerConnectionState.sendMessage(
    messageReadError);
760
761
762                     Log.d(TAG, "Input stream was
    disconnected", e);
763                     break;
764                 }
765             }
766         }
767
768
    //-------------------------------------------------------
    ---------------------------------
769         // Call this from the main activity to send data
    to the remote device.
770         public void write(byte[] bytes) {
771             try {
772                 //writes outputStream
773                 mmOutStream.write(bytes);
774
775
    //-------------------------------------------------------
    --------------------------
776                 Message messageWrite =
    handlerConnectionState.obtainMessage(
777                         intStateMessageWrite, -1, -1,
    mmBuffer);
778                 messageWrite.sendToTarget();
779
780                 Log.d(TAG, "write: ");
781             } catch (IOException e) {
782
    //-------------------------------------------------------
    --------------------------
783                 Message messageWriteError = Message.
    obtain();
784                 messageWriteError.what =
    intStateWriteError;
785                 handlerConnectionState.sendMessage(
    messageWriteError);
```

```
786
787                 Log.e(TAG, "Error occurred when sending
    data", e);
788
789             }
790         }
791
792         // Call this method from the main activity to
    shut down the connection.
793         public void cancel() {
794             try {
795
796                 mmSocket.close();
797
798             } catch (IOException e) {
799
800                 Log.e(TAG, "Could not close the connect
    socket", e);
801             }
802         }
803     }
804
805
    //-----------------------------------------------------
    ---------------------------------------
806     //Handler reads capacity from SampleCapacityThread
    and regulate the level
807     Handler handlerControlCapacity= new Handler(new
    Handler.Callback() {
808         @Override
809         public boolean handleMessage(@NonNull Message msg
    ) {
810
811             if(msg.what == intStateCapacity){
812                 int intCurrentBatteryLevel= (int) msg.obj
    ; //msg.obj contains current capacity level
813
814                 /* BatteryManager batteryManager=(
    BatteryManager)getSystemService(Context.BATTERY_SERVICE);
815                  int status = batteryManager.
    getIntProperty(BatteryManager.BATTERY_PROPERTY_STATUS);
816                 boolean isVoltage= status ==
    BatteryManager.BATTERY_STATUS_CHARGING
817                         || status == BatteryManager.
    BATTERY_STATUS_FULL;
```

```
818
819                 */
820
821     //-------------------------------------------------------
    -------------------------
822                 //reads saved setPoint value
823                 prefSetPoint = getSharedPreferences(
    strSetPointExtra, Context.MODE_PRIVATE);
824                 int intSetPoint=prefSetPoint.getInt(
    strSetPoint,90);
825
826     //-------------------------------------------------------
    -------------------------
827                 //reads saved relativePoint value
828                 prefRelativePoint= getSharedPreferences(
    strRelativeExtra, Context.MODE_PRIVATE);
829                 int intRelativePoint=prefRelativePoint.
    getInt(strRelativePoint,3);
830                 Log.d(TAG, "handlerControlCapacity: "+
    intCurrentBatteryLevel+ "setpoint "+intSetPoint+"
    relative "+ intRelativePoint);
831
832     //-------------------------------------------------------
    -------------------------
833                 //begin regulate if connection successful
834                 if(strTvState == strConnected){
835
836     //-------------------------------------------------------
    -----------------
837                     //sends message to µC to stop
    charging
838                     if (isCharging &&
    intCurrentBatteryLevel >= intSetPoint) {
839
840                         sendMessage(strStopCharging);
841
842                        //isCharging = false;
843
844                     }
845
846
```

```
846
    //-------------------------------------------------------
    ------------------
847                     //sends message to µC to start
    charging
848                     if (!isCharging &&
    intCurrentBatteryLevel <= intSetPoint - intRelativePoint)
     {
849
850                         sendMessage(strStartCharging);
851
852                         //isCharging = true;
853
854                     }
855
856
    //-------------------------------------------------------
    ------------------
857                     if(!strReadMessage.equals("")){
858
859                         //return value of µC if charging
    start
860                         if(strReadMessage.equals("X")){
861
862                             if(!isCharging){
863
864                                 tvCheckAvailable.setText(
    R.string.strAvailableError);
865
866                             }else{
867
868                                 tvCheckAvailable.setText(
    R.string.strTurnedOn);
869                             }
870
871                         }
872
873                         //return value of µC if charging
    stop
874                         if(strReadMessage.equals("Y")){
875
876                             if(isCharging){
877
878                                 tvCheckAvailable.setText(
    R.string.strAvailableError);
```

```
879
880                             }else{
881
882                                 tvCheckAvailable.setText(
    R.string.strTurnedOff);
883                             }
884
885
886                         }
887
888                     }
889
890                     //test signal
891                     if(strReadMessage.equals("hello")){
892
893                         sendMessage("OK");
894                     }
895                 }
896             }
897             return true;
898         }
899     });
900
901
    //-----------------------------------------------------
    -------------------------------------
902     //Handler reads the current state from ConnectThread
    and ConnectedThread
903     Handler handlerConnectionState=new Handler(new
    Handler.Callback() {
904         @Override
905         public boolean handleMessage(@NonNull Message msg
    ) {
906             switch (msg.what){
907
908                 //App connected to charger
909                 case intStateConnected:
910                     //use for method startConnection to
    define connection state
911                     intState = intStateConnected;
912
913                     //use in Resume to restore connection
     state for user on layout
914                     //in Handler handlerControlCapacity
915                     //in Method sendMessage to except
```

```
915 write error
916                     strTvState = strConnected;
917 
918                     //sets CONNECTED message on layout
919                     tvConnectionState.setText(R.string.
    strConnectedTo);
920 
921                     //disable START button if connected
922                    // btStart.setEnabled(false);
923                    // btStart.setBackgroundColor(Color.
    GREEN);
924                     btStart.setVisibility(View.INVISIBLE)
    ;
925 
926                     Log.d(TAG, "handleMessage: connected"
    );
927                     break;
928 
929                 //App connected to charger and reads
    inputStream
930                 case intStateMessageRead:
931 
932                     //use for method startConnection to
    define connection state
933                     intState = intStateMessageRead;
934 
935                     //use in Resume to restore connection
     state for user on layout
936                     //in Handler handlerControlCapacity
937                     //in Method sendMessage to except
    write error
938                     strTvState = strConnected;
939 
940                     //sets CONNECTED message on layout
941                     tvConnectionState.setText(R.string.
    strConnectedTo);
942 
943                     //disable START button if connected
944                     //btStart.setEnabled(false);
945                    // btStart.setBackgroundColor(Color.
    GREEN);
946                     btStart.setVisibility(View.INVISIBLE)
    ;
947 
948 
```

```
948
    //-------------------------------------------------------
    ---------------------
949                     //read byte from inputStream and
    convert to string
950                     byte[] readBuff= (byte[]) msg.obj;
951                     String strTemp= new String(readBuff,0
    ,msg.arg1);
952                     strReadMessage = strTemp;
953
954                     Log.d(TAG, "handleMessage: read "+
    strTemp);
955                     break;
956
957                 //App connected to charger and writes
    outputStream
958                 case intStateMessageWrite:
959                     //use for method startConnection to
    define connection state
960                     intState = intStateMessageWrite;
961
962                     //use in Resume to restore connection
     state for user on layout
963                     //in Handler handlerControlCapacity
964                     //in Method sendMessage to except
    write error
965                     strTvState = strConnected;
966
967                     //sets CONNECTED message on layout
968                     tvConnectionState.setText(R.string.
    strConnectedTo);
969
970                     //disable START button if connected
971                    // btStart.setEnabled(false);
972                    // btStart.setBackgroundColor(Color.
    GREEN);
973                     btStart.setVisibility(View.INVISIBLE)
    ;
974
975                     Log.d(TAG, "handleMessage: write");
976                     break;
977
978                 //App could not connect to the charger
979                 case intStateConnectionFailed:
980                     //use for method startConnection to
```

```
980 define connection state
981                     intState = intStateConnectionFailed;
982
983                     //use in Resume to restore
    connection state for user on layout
984                     //in Handler handlerControlCapacity
985                     //in Method sendMessage to except
    write error
986                     strTvState = strFailed;
987
988                     //sets CONNECTION FAILED message on
    layout
989                     tvConnectionState.setText(R.string.
    strFailedTo);
990
991                     //enable START button if connection
    failed
992                     btStart.setVisibility(View.VISIBLE);
993                     btStart.setEnabled(true);
994                     btStart.setBackgroundColor(Color.
    YELLOW);
995
996                     //stop Thread to sample capacity
    because connection failed
997                     blStopThread = true;
998
999                     Log.d(TAG, "handleMessage:
    connection failed");
1000                    break;
1001
1002                // Error occurred when sending data and
    connection failed to charger
1003                case intStateWriteError:
1004                    //use for method startConnection to
    define connection state
1005                    intState = intStateWriteError;
1006
1007                    //use in Resume to restore
    connection state for user on layout
1008                    //in Handler handlerControlCapacity
1009                    //in Method sendMessage to except
    write error
1010                    strTvState = strFailed;
1011
1012                    //sets CONNECTION FAILED message on
```

```
1012 layout
1013                     tvConnectionState.setText(R.string.
     strFailedTo);
1014
1015                     //enable START button if connection
     failed
1016                     btStart.setVisibility(View.VISIBLE);
1017                     btStart.setEnabled(true);
1018                     btStart.setBackgroundColor(Color.
     YELLOW);
1019
1020                     //stop Thread to sample capacity
     because connection failed
1021                     blStopThread = true;
1022
1023                     Log.d(TAG, "handleMessage: WRITE
     ERROR");
1024                     break;
1025
1026                 // Error occurred when reading data and
     connection failed to charger
1027                 case intStateReadError:
1028                     //use for method startConnection to
     define connection state
1029                     intState = intStateReadError;
1030
1031                     //use in Resume to restore
     connection state for user on layout
1032                     //in Handler handlerControlCapacity
1033                     //in Method sendMessage to except
     write error
1034                     strTvState = strFailed;
1035
1036                     //sets CONNECTION FAILED message on
     layout
1037                     tvConnectionState.setText(R.string.
     strFailedTo);
1038
1039                     //enable START button if connection
     failed
1040                     btStart.setVisibility(View.VISIBLE);
1041                     btStart.setEnabled(true);
1042                     btStart.setBackgroundColor(Color.
     YELLOW);
1043
```

```
1044                    //stop Thread to sample capacity
    because connection failed
1045                    blStopThread = true;
1046
1047                    Log.d(TAG, "handleMessage: READ
    ERROR");
1048                    break;
1049                }
1050                return true;
1051            }
1052        });
1053 }
1054
```

### C4: Aktivität PropertiesActivity

```
1  package com.example.smartcharge;
2
3  import androidx.appcompat.app.ActionBar;
4  import androidx.appcompat.app.AppCompatActivity;
5  import androidx.preference.PreferenceManager;
6
7  import android.content.BroadcastReceiver;
8  import android.content.Context;
9  import android.content.Intent;
10 import android.content.IntentFilter;
11 import android.content.SharedPreferences;
12 import android.os.BatteryManager;
13 import android.os.Build;
14 import android.os.Bundle;
15 import android.util.Log;
16 import android.view.LayoutInflater;
17 import android.view.Menu;
18 import android.view.MenuItem;
19 import android.view.View;
20 import android.view.ViewGroup;
21 import android.widget.BaseAdapter;
22 import android.widget.ImageView;
23 import android.widget.ListView;
24 import android.widget.TextView;
25
26 import java.util.ArrayList;
27
28 import static com.example.smartcharge.Constants.
   strLanguageProperties;
29 import static com.example.smartcharge.Constants.
   switchLanguage;
30
31 public class PropertiesActivity extends AppCompatActivity
   {
32
33
   //-------------------------------------------------------
   ------------------------------------
34     //part define Layout-Widgets
35     ListView listView; //Widget Container
36     ArrayList<ListInformation> listInfos; //ArrayList of
   class ListInformation
37     CustomAdapter customAdapter; // Instance  of class
   CustomAdapter
38
```

```
39
40
   //-------------------------------------------------------
   -----------------------------------
41     //BroadcastReceiver update and refresh significant
   battery values
42     private BroadcastReceiver receiver= new
   BroadcastReceiver() {
43
44
45
   //-------------------------------------------------------
   -------------------------------
46         //method to read values
47         @Override
48         public void onReceive(Context context, Intent
   intent) {
49
50
   //-------------------------------------------------------
   ----------------------------
51             //define ListView, ArrayList and class
   BatteryManager
52             listView=((PropertiesActivity)context).
   findViewById(R.id.listView);
53             listInfos= new ArrayList<ListInformation>();
54             BatteryManager batteryManager
55                     = (BatteryManager)context.
   getSystemService(Context.BATTERY_SERVICE);
56
57
58
   //-------------------------------------------------------
   ----------------------------
59             //Technology of battery
60             String strTech =intent.getExtras().getString(
   BatteryManager.EXTRA_TECHNOLOGY);
61             listInfos.add(new ListInformation(getString(R.
   string.strTech),strTech,R.drawable.
   ic_battery_std_black_24dp));
62
63
64
   //-------------------------------------------------------
   ----------------------------
```

```
65              //health states
66              int intHealth = intent.getIntExtra(
    BatteryManager.EXTRA_HEALTH,-1);
67              switch(intHealth){
68                  case BatteryManager.BATTERY_HEALTH_COLD:
69                      listInfos.add(new ListInformation(
    getString(R.string.strHealth),getString(R.string.strCold)
    ,
70                              R.drawable.
    ic_favorite_black_24dp));  break;
71                  case BatteryManager.BATTERY_HEALTH_DEAD:
72                      listInfos.add(new ListInformation(
    getString(R.string.strHealth),getString(R.string.strDead)
    ,
73                              R.drawable.
    ic_favorite_black_24dp));   break;
74                  case BatteryManager.BATTERY_HEALTH_GOOD:
75                      listInfos.add(new ListInformation(
    getString(R.string.strHealth),getString(R.string.strGood)
    ,
76                              R.drawable.
    ic_favorite_black_24dp));   break;
77                  case BatteryManager.
    BATTERY_HEALTH_OVER_VOLTAGE:
78                      listInfos.add(new ListInformation(
    getString(R.string.strHealth),getString(R.string.
    strOverVoltage),
79                              R.drawable.
    ic_favorite_black_24dp));   break;
80                  case BatteryManager.
    BATTERY_HEALTH_OVERHEAT:
81                      listInfos.add(new ListInformation(
    getString(R.string.strHealth),getString(R.string.
    strOverheat),
82                              R.drawable.
    ic_favorite_black_24dp)); break;
83                  case BatteryManager.
    BATTERY_HEALTH_UNKNOWN:
84                      listInfos.add(new ListInformation(
    getString(R.string.strHealth),getString(R.string.
    strUnknown),
85                              R.drawable.
    ic_favorite_black_24dp)); break;
86                  case BatteryManager.
    BATTERY_HEALTH_UNSPECIFIED_FAILURE:
```

```
87                      listInfos.add(new ListInformation(
   getString(R.string.strHealth),getString(R.string.
   strUnspecifiedFailure),
88                              R.drawable.
   ic_favorite_black_24dp));  break;
89              }
90
91
92
   //-------------------------------------------------------
   ------------------------------
93              //plug states
94              int intPlug = intent.getIntExtra(
   BatteryManager.EXTRA_PLUGGED,-1);
95              switch (intPlug){
96                  case BatteryManager.BATTERY_PLUGGED_AC:
97                      listInfos.add(new ListInformation(
   getString(R.string.strPlug),getString(R.string.AC),
98                              R.drawable.
   ic_power_black_24dp));  break;
99                  case BatteryManager.BATTERY_PLUGGED_USB:
100                     listInfos.add(new ListInformation(
   getString(R.string.strPlug),getString(R.string.USB),
101                             R.drawable.ic_usb_black_24dp)
   );   break;
102                 case BatteryManager.
   BATTERY_PLUGGED_WIRELESS:
103                     listInfos.add(new ListInformation(
   getString(R.string.strPlug),getString(R.string.WIRELESS),
104                             R.drawable.ic_wifi_black_24dp
   ));  break;
105                 default:listInfos.add(new ListInformation
   (getString(R.string.strPlug),getString(R.string.
   strNotPlugged),
106                         R.drawable.ic_power_off_24px));
   break;
107             }
108
109
110
   //-------------------------------------------------------
   ------------------------------
111             //charging states
112             int intStatus =intent.getIntExtra(
   BatteryManager.EXTRA_STATUS,-1);
```

```
113             switch (intStatus){
114                 case BatteryManager.
    BATTERY_STATUS_CHARGING:
115                     listInfos.add(new ListInformation(
    getString(R.string.strCharging),getString(R.string.
    strIsCharging),
116                             R.drawable.
    ic_battery_charging_full_black_24dp));  break;
117                 case BatteryManager.
    BATTERY_STATUS_DISCHARGING:
118                     listInfos.add(new ListInformation(
    getString(R.string.strCharging),getString(R.string.
    strDischarging),
119                             R.drawable.
    ic_battery_charging_full_black_24dp));  break;
120                 case BatteryManager.BATTERY_STATUS_FULL:
121                     listInfos.add(new ListInformation(
    getString(R.string.strCharging),getString(R.string.
    strChargingFull),
122                             R.drawable.
    ic_battery_full_black_24dp)); break;
123                 case BatteryManager.
    BATTERY_STATUS_NOT_CHARGING:
124                     listInfos.add(new ListInformation(
    getString(R.string.strCharging),getString(R.string.
    strNotCharging),
125                             R.drawable.ic_power_off_24px)
    ); break;
126                 case BatteryManager.
    BATTERY_STATUS_UNKNOWN:
127                     listInfos.add(new ListInformation(
    getString(R.string.strCharging),getString(R.string.
    strUnknown),
128                             R.drawable.
    ic_battery_unknown_black_24dp)); break;
129             }
130
131
    //-----------------------------------------------------
    ------------------------------
132             //Temperature
133             int intTemperature = intent.getIntExtra(
134                     BatteryManager.EXTRA_TEMPERATURE,-1)
    / 10;
135             listInfos.add(new ListInformation(getString(R
```

```
135 .string.strTemperature),String.valueOf(intTemperature+ "°
    C"),
136                     R.drawable.ic_waves_24px));
137
138
139
    //-------------------------------------------------------
    ------------------------------
140             //Voltage
141             int intVoltage = intent.getIntExtra(
142                     BatteryManager.EXTRA_VOLTAGE,-1);
143             listInfos.add(new ListInformation(getString(R
    .string.strVoltage),String.valueOf(intVoltage+" mV"),
144                     R.drawable.ic_offline_bolt_24px));
145
146
    //-------------------------------------------------------
    ------------------------------
147             //Current and Battery Capacity
148             if (Build.VERSION.SDK_INT > Build.
    VERSION_CODES.LOLLIPOP) {
149
150
    //-------------------------------------------------------
    --------------------------
151                 //add capacity to list
152                 listInfos.add(new ListInformation(
    getString(R.string.strCapacity) ,
153                         String.valueOf(batteryManager.
    getIntProperty(BatteryManager
154                                 .
    BATTERY_PROPERTY_CHARGE_COUNTER)*0.001)+" mAh",
155                         R.drawable.
    ic_battery_charging_full_black_24dp));
156
157
    //-------------------------------------------------------
    --------------------------
158                 //add current to list
159                 listInfos.add(new ListInformation(
    getString(R.string.strCurrentNow) ,
160                         String.valueOf(batteryManager.
    getIntProperty(BatteryManager
161                                 .
    BATTERY_PROPERTY_CURRENT_NOW)*0.001)+" mA",
```

```
162                         R.drawable.
    ic_battery_charging_full_black_24dp));
163             }
164
165
166             //initialize ListView
167             customAdapter= new CustomAdapter(((
    PropertiesActivity)context),listInfos);
168             listView.setAdapter(customAdapter);
169
170         }
171     };
172
173
174
175
    //-----------------------------------------------------
    -------------------------------------
176     //Activity lifecycle method onCreate
177     @Override
178     protected void onCreate(Bundle savedInstanceState) {
179         super.onCreate(savedInstanceState);
180
181         //PreferenceManager.setDefaultValues(this, R.xml.
    root_preferences, false);
182         SharedPreferences sharedPref = PreferenceManager.
    getDefaultSharedPreferences(this);
183         strLanguageProperties = sharedPref.getString("
    language", "ENGLISH");
184         switchLanguage(this.getResources(),
    strLanguageProperties);
185
186         setContentView(R.layout.activity_properties);
187         //Toolbar toolbar = findViewById(R.id.toolbar);
188         //setSupportActionBar(toolbar);
189         ActionBar actionBar = getSupportActionBar();
190         if (actionBar != null) {
191             actionBar.setTitle(R.string.
    strActionProperties);
192             actionBar.setDisplayHomeAsUpEnabled(true);
193         }
194         //getSupportActionBar().setDisplayHomeAsUpEnabled
    (true);
195
196         registerReceiver(receiver,new IntentFilter(Intent
```

```
196 .ACTION_BATTERY_CHANGED));
197     }
198
199
200
    //-------------------------------------------------------
    ---------------------------------------
201     //Activity lifecycle method onCreate
202     //sets chosen language before activity is in
    foreground
203     @Override
204     protected void onPostResume() {
205         super.onPostResume();
206
207         Log.d("main", "onPostResume: "+
    strLanguageProperties);
208         SharedPreferences sharedPref = PreferenceManager.
    getDefaultSharedPreferences(this);
209         String strChosenLanguage = sharedPref.getString("
    language", "ENGLISH");
210         if(!strLanguageProperties.equals(
    strChosenLanguage)){
211             Log.d("main", "onPostResume: sprache aendern"
    );
212             strChosenLanguage = strLanguageProperties;
213             switchLanguage(this.getResources(),
    strChosenLanguage);
214             recreate();
215         }
216     }
217
218
219
    //-------------------------------------------------------
    ---------------------------------------
220     //Activity lifecycle method onDestroy
221     //unregisterReceiver BroadcastReceiver when the app
    is destroyed.
222     @Override
223     protected void onDestroy() {
224         super.onDestroy();
225         unregisterReceiver(receiver);
226     }
227
228
```

```
229
    //-----------------------------------------------------------
    ------------------------------------
230     //part menu
231     @Override
232     public boolean onCreateOptionsMenu(Menu menu) {
233         // Inflate the menu; this adds items to the
    action bar if it is present.
234         getMenuInflater().inflate(R.menu.menu_properties,
     menu);
235         return true;
236     }
237
238     @Override
239     public boolean onOptionsItemSelected(MenuItem item) {
240         // Handle action bar item clicks here. The action
     bar will
241         // automatically handle clicks on the Home/Up
    button, so long
242         // as you specify a parent activity in
    AndroidManifest.xml.
243         int id = item.getItemId();
244
245         //noinspection SimplifiableIfStatement
246         if (id == R.id.action_settings) {
247             Intent intent = new Intent(this,
    SettingsActivity.class);
248             startActivity(intent);
249             finish();
250             return true;
251         }else if (id == R.id.action_home) {
252             Intent intent = new Intent(this, MainActivity
    .class);
253             startActivity(intent);
254             finish();
255             return true;
256         }else {
257
258             return super.onOptionsItemSelected(item);
259         }
260     }
261
262
263
    //-----------------------------------------------------------
```

```
263 ---------------------------------------
264     //Class sets the list with title, description and
    image
265     public class ListInformation{
266         private String strTitle; //Title of battery
    attribute
267         private String strDescription; //Description of
    title
268         private int intImages; //Image of title
269 
270         //Constructor of class ListInformation
271         public ListInformation(String strTitle, String
    strDescription, int intImages) {
272             this.strTitle = strTitle;
273             this.strDescription = strDescription;
274             this.intImages = intImages;
275         }
276 
277         //Method returns Title
278         public String getStrTitle() {
279             return strTitle;
280         }
281 
282         ////Method returns Description of title
283         public String getStrDescription() {
284             return strDescription;
285         }
286 
287         //Method returns corresponding image of title
288         public int getIntImages() {
289             return intImages;
290         }
291     }
292 
293 
294 
    //-------------------------------------------------------
    ---------------------------------------
295     //Class to customize default BaseAdapter for ListView
296     public class CustomAdapter extends BaseAdapter {
297         Context context; //Context of the activity
298         ArrayList<ListInformation> listInfos; //Instance
    of class ListInformation
299 
300         //Constructor of CustomAdapter
```

```
301         public CustomAdapter(Context context, ArrayList<
    ListInformation> listInfos) {
302             this.context = context;
303             this.listInfos = listInfos;
304         }
305 
306         //Method returns size of elements in the list
307         @Override
308         public int getCount() {
309             return listInfos.size();
310         }
311 
312         @Override
313         public Object getItem(int i) {
314             return null;
315         }
316 
317         @Override
318         public long getItemId(int i) {
319             return 0;
320         }
321 
322 
    //-----------------------------------------------------
    ---------------------------------
323         //Method to create an adapter
324         @Override
325         public View getView(int i, View view, ViewGroup
    viewGroup) {
326 
327             view= LayoutInflater.from(context).inflate(
328                     R.layout.custom_listview,viewGroup,
329                     false);
330 
331 
    //-----------------------------------------------------
    ------------------------------
332             //define
333             TextView tvTitle= view.findViewById(R.id.
    tvTitle);
334             TextView tvDescription = view.findViewById(R.
    id.tvDescription);
335             ImageView ivImage= view.findViewById(R.id.
    ivImage);
336 
```

```
337
    //-------------------------------------------------------
    ------------------------------
338             //initialize
339             tvTitle.setText(listInfos.get(i).getStrTitle(
    ));
340             tvDescription.setText(listInfos.get(i).
    getStrDescription());
341             ivImage.setImageResource(listInfos.get(i).
    getIntImages());
342
343             return view;
344         }
345     }
346
347 }
348
```

### C5: Aktivität SettingsActivity

```
 1 package com.example.smartcharge;
 2
 3 import android.bluetooth.BluetoothDevice;
 4 import android.content.Intent;
 5 import android.content.SharedPreferences;
 6 import android.os.Bundle;
 7 import android.util.Log;
 8 import android.view.Menu;
 9 import android.view.MenuItem;
10 import android.widget.Toast;
11
12 import androidx.appcompat.app.ActionBar;
13 import androidx.appcompat.app.AppCompatActivity;
14 import androidx.preference.EditTextPreference;
15 import androidx.preference.ListPreference;
16 import androidx.preference.Preference;
17 import androidx.preference.PreferenceFragmentCompat;
18 import androidx.preference.PreferenceManager;
19 import androidx.preference.SeekBarPreference;
20
21 import java.util.Set;
22
23 import static com.example.smartcharge.Constants.
   strBltDeviceAddress;
24 import static com.example.smartcharge.Constants.
   strBltDeviceName;
25 import static com.example.smartcharge.Constants.
   strBltDeviceExtra;
26 import static com.example.smartcharge.Constants.
   strRelativeExtra;
27 import static com.example.smartcharge.Constants.
   strRelativePoint;
28 import static com.example.smartcharge.Constants.
   bluetoothAdapter;
29 import static com.example.smartcharge.Constants.
   strLanguageSettings;
30 import static com.example.smartcharge.Constants.
   switchLanguage;
31
32 public class SettingsActivity extends AppCompatActivity {
33     public static final String TAG="SettingsActivity";
34
35
   //-----------------------------------------------------
   -------------------------------------
```

```
36      //part list bonded bluetooth devices
37      //Set Array contains bonded bluetooth devices
38      public static Set<BluetoothDevice> bluetoothDeviceSet;
39      //CharSequence Array contains names of bonded devices
40      public static CharSequence[] arrayDeviceName;
41      //CharSequence Array contains mac_address of bonded
    devices
42      public static CharSequence[] arrayDeviceAddress;
43
44
    //-----------------------------------------------------
    -----------------------------------
45      //Activity lifecycle method onCreate
46      @Override
47      protected void onCreate(Bundle savedInstanceState) {
48          super.onCreate(savedInstanceState);
49          PreferenceManager.setDefaultValues(this, R.xml.
    root_preferences, false);
50          SharedPreferences sharedPref = PreferenceManager.
    getDefaultSharedPreferences(this);
51          strLanguageSettings = sharedPref.getString("
    language", "ENGLISH");
52          switchLanguage(this.getResources(),
    strLanguageSettings);
53
54          setContentView(R.layout.settings_activity);
55          getSupportFragmentManager()
56                  .beginTransaction()
57                  .replace(R.id.settings, new
    SettingsFragment())
58                  .commit();
59          ActionBar actionBar = getSupportActionBar();
60          if (actionBar != null) {
61              actionBar.setTitle(R.string.strActionSettings)
    ;
62              actionBar.setDisplayHomeAsUpEnabled(true);
63          }
64
65
66      }
67
68
69
    //-----------------------------------------------------
    -----------------------------------
```

```
70      //part menu
71      @Override
72      public boolean onCreateOptionsMenu(Menu menu) {
73          // Inflate the menu; this adds items to the
    action bar if it is present.
74          getMenuInflater().inflate(R.menu.menu_settings,
    menu);
75          return true;
76      }
77
78      @Override
79      public boolean onOptionsItemSelected(MenuItem item) {
80          // Handle action bar item clicks here. The action
     bar will
81          // automatically handle clicks on the Home/Up
    button, so long
82          // as you specify a parent activity in
    AndroidManifest.xml.
83          int id = item.getItemId();
84          //noinspection SimplifiableIfStatement
85          if (id == R.id.action_home) {
86              Intent intent = new Intent(this, MainActivity
    .class);
87              startActivity(intent);
88              finish();
89              return true;
90          }else if (id == R.id.action_properties) {
91              Intent intent = new Intent(this,
    PropertiesActivity.class);
92              startActivity(intent);
93              finish();
94              return true;
95          }else {
96
97              return super.onOptionsItemSelected(item);
98          }
99      }
100
101
102
    //-------------------------------------------------------
    ---------------------------------------
103     //Activity lifecycle method onCreate
104     //sets chosen language before activity is in
    foreground
```

```
105     @Override
106     protected void onPostResume() {
107         super.onPostResume();
108         Log.d(TAG, "onPostResume: ");
109         SharedPreferences sharedPref = PreferenceManager.
    getDefaultSharedPreferences(this);
110         String strChosenLanguage = sharedPref.getString("
    language", "ENGLISH");
111         if(!strLanguageSettings.equals(strChosenLanguage)
    ){
112             strChosenLanguage = strLanguageSettings;
113             switchLanguage(this.getResources(),
    strChosenLanguage);
114             recreate();
115         }
116     }
117
118
119
    //-------------------------------------------------------
    -------------------------------------
120     //Class creates a SettingsFragment
121     public static class SettingsFragment extends
    PreferenceFragmentCompat
122             implements SharedPreferences.
    OnSharedPreferenceChangeListener {
123
124
    //-------------------------------------------------------
    ---------------------------------
125         //data storage mac_address referencePoint
126         SharedPreferences pref_mac_address;
127         SharedPreferences pref_rel_point;
128         SharedPreferences.Editor editorMacAddress;
129         SharedPreferences.Editor editorRelPoint;
130
131         static ListPreference listPreference; //define a
    instance of class ListPreference
132
133
134
    //-------------------------------------------------------
    ---------------------------------
135         //
136         @Override
```

```
137         public void onCreatePreferences(Bundle
    savedInstanceState, String rootKey) {
138             setPreferencesFromResource(R.xml.
    root_preferences, rootKey);
139
140
141
    //-------------------------------------------------------
    ------------------------------
142             //SharedPreferences for MAC_ADDRESS,
    deviceName and relativePoint
143             pref_mac_address=getActivity().
    getSharedPreferences(strBltDeviceExtra,MODE_PRIVATE);
144             editorMacAddress=pref_mac_address.edit();
145             //SharedPreferences for relativePoint
146             pref_rel_point=getActivity().
    getSharedPreferences(strRelativeExtra,MODE_PRIVATE);
147             editorRelPoint=pref_rel_point.edit();
148
149
    //-------------------------------------------------------
    ------------------------------
150             //fill the list with bonded devices, when
    bluetooth is enabled
151             if(bluetoothAdapter.isEnabled()){
152                 bluetoothDeviceSet= bluetoothAdapter.
    getBondedDevices();
153                 arrayDeviceName = new CharSequence[
    bluetoothDeviceSet.size()];
154                 arrayDeviceAddress = new CharSequence[
    bluetoothDeviceSet.size()];
155
156                 int index=0;
157                 if(bluetoothDeviceSet.size()>0) {
158                     for (BluetoothDevice device :
    bluetoothDeviceSet) {
159                         // btArray[index] = device;
160                         String strIdentifier = ((device.
    getName()==null || device.getName()== "" ))?
161                                 device.getAddress():
    device.getName();
162
163                         arrayDeviceName[index] =
    strIdentifier;
164                         arrayDeviceAddress[index] =
```

```
164 device.getAddress();
165                         index++;
166                     }
167 
168                 }else{
169                     //String[] strNullPointer={"NULL"};
170                     listPreference=findPreference("device
    ");
171                     listPreference.setEnabled(false);
172                     ((EditTextPreference) findPreference(
    "renamed_device")).setEnabled(false);
173 
174                     Toast.makeText(getContext(),
175                             R.string.strNullEntryError,
176                             Toast.LENGTH_LONG).show();
177                 }
178 
179             }else{
180                 listPreference=findPreference("device");
181                 listPreference.setEnabled(false);
182                 ((EditTextPreference) findPreference("
    renamed_device")).setEnabled(false);
183                 Toast.makeText(getContext(),R.string.
    strBltDisabled,Toast.LENGTH_LONG).show();
184             }
185 
186 
    //-----------------------------------------------------
    ------------------------------
187             // initialize list and set values
188             listPreference = findPreference("device");
189             listPreference.setEntries(arrayDeviceName);
190             listPreference.setEntryValues(
    arrayDeviceAddress);
191 
192 
193 
    //-----------------------------------------------------
    ------------------------------
194             //register
195             getPreferenceScreen().getSharedPreferences().
    registerOnSharedPreferenceChangeListener(this);
196 
197 
198 
```

```
198
    //-------------------------------------------------------
    ------------------------------
199             //changes the language and restart the
    activity
200             findPreference("language").
    setOnPreferenceChangeListener(
201                     new Preference.
    OnPreferenceChangeListener() {
202                 @Override
203                 public boolean onPreferenceChange(
    Preference preference, Object newValue) {
204
205                     startActivity(new Intent(getContext()
    ,SettingsActivity.class));
206                     getActivity().finish();
207
208                     return true;
209                 }
210             });
211         }
212
213
    //-------------------------------------------------------
    ---------------------------------
214         //Activity lifecycle method onResume
215         //refresh seekBarPreference summary
216         @Override
217         public void onResume() {
218             super.onResume();
219
220             Preference pre = findPreference("relative");
221             SeekBarPreference seekBarPreference= (
    SeekBarPreference)pre;
222             int intSummary= (int)seekBarPreference.
    getValue() + 2;
223             seekBarPreference.setSummary(getString(R.
    string.strControlRange)+" "+intSummary+"%");
224
225         }
226
227
228
    //-------------------------------------------------------
    ---------------------------------
```

```
229         //read in this method, when a preference changes
    its value
230         @Override
231         public void onSharedPreferenceChanged(
    SharedPreferences sharedPreferences, String key) {
232
233
    //-------------------------------------------------------
    ------------------------------
234             //if the list changes, save MAC_ADDRESS and
    NAME
235             if(key.equals("device")){
236                 Preference pre=findPreference(key);
237                 ListPreference listPreference = (
    ListPreference) pre;
238                 editorMacAddress.putString(
    strBltDeviceAddress,listPreference.getValue().toString())
    ;
239
240                 if(listPreference.getEntry() !=null){
241                     editorMacAddress.putString(
    strBltDeviceName,listPreference.getEntry().toString());
242                     ((EditTextPreference) findPreference(
    "renamed_device")).setText(
243                             listPreference.getEntry().
    toString());
244
245                     Log.d(TAG, "onSharedPreferenceChanged
    : "+String.valueOf(
246                             listPreference.getEntry())+"
    Value "+listPreference.getValue());
247                 }
248                 editorMacAddress.apply();
249             }
250
251
252
    //-------------------------------------------------------
    ------------------------------
253             //if the relative point changes, save
    relativePoint
254             if(key.equals("relative")){
255                 Preference pre=findPreference(key);
256                 SeekBarPreference seekBarPreference= (
    SeekBarPreference)pre;
```

```
257                 int intSummary= (int)seekBarPreference.
    getValue() + 2;
258                 editorRelPoint.putInt(strRelativePoint,
    intSummary);
259                 editorRelPoint.apply();
260                 seekBarPreference.setSummary(getString(R.
    string.strControlRange)+" "+intSummary+"%");
261                 Log.d(TAG, "onSharedPreferenceChanged: "+
     seekBarPreference.getValue());
262             }
263
264
    //-----------------------------------------------------
    ------------------------------
265             //if the relative point changes, save
    relativePoint
266             if(key.equals("renamed_device")){
267                 Preference pre=findPreference(key);
268                 EditTextPreference editTextPreference = (
    EditTextPreference) pre;
269                 String strRenamed = editTextPreference.
    getText();
270                 if(!strRenamed.equals("") && strRenamed
    != null){
271                     editorMacAddress.putString(
    strBltDeviceName,strRenamed);
272                     editorMacAddress.apply();
273
274                 }
275
276                 Log.d(TAG, "onSharedPreferenceChanged: "+
     editTextPreference.getText());
277             }
278
279         }
280
281
    //-----------------------------------------------------
    ---------------------------------
282         //Activity lifecycle method onResume
283         //must to unregister
    OnSharedPreferenceChangeListener
284         @Override
285         public void onStop() {
286             super.onStop();
```



```
287             getPreferenceScreen().getSharedPreferences().
    unregisterOnSharedPreferenceChangeListener(this);
288         }
289     }
290 }
```

### C6: Code Microcontroller

```
#define CONTROL_PIN 2
char chrCommand;
static int intKey = 1;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(CONTROL_PIN, OUTPUT);
}

void loop() {
  chrCommand = 'x';
  // put your main code here, to run repeatedly:
 if(Serial.available()){ // Checks whether data is comming from the serial port
 chrCommand = Serial.read(); // Reads the data from the serial port
  if(intKey == 1){
      if(chrCommand == 'e'){
        intKey = 2;
      }else if(chrCommand == 'a'){
        intKey = 3;
      }
    }

    if(chrCommand == 'e' && intKey == 2){
      digitalWrite(CONTROL_PIN,LOW);
      Serial.print("X");
      intKey = 3;
    }
    delay(500);
    if(chrCommand == 'a' && intKey == 3){
      digitalWrite(CONTROL_PIN,HIGH);
      Serial.print("Y");
      intKey = 2;
    }
 }
}
```

# Erklärung zur selbstständigen Bearbeitung

Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

**Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit**

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

*Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI*

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

**Erklärung zur selbstständigen Bearbeitung der Arbeit**

Hiermit versichere ich,

Name: Demir

Vorname: Emrah

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Applikation zur Verbesserung der Laderegelung in Smartphones

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

*- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -*

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

| Hamburg | 14.01.2020 | |
|---|---|---|
| Ort | Datum | Unterschrift im Original |