Inhalt

[Abbildungsverzeichnis III](#_Toc79314573)

[Tabellenverzeichnis III](#_Toc79314574)

[Danksagung III](#_Toc79314575)

[2 Einführung 1](#_Toc79314576)

[2.1 Projekteschreibung 1](#_Toc79314577)

[2.2 Projektziel 1](#_Toc79314578)

[3 Schema 2](#_Toc79314579)

[4 Komponenten 3](#_Toc79314580)

[4.1 nRF5340 Development Kit 3](#_Toc79314581)

[4.2 Geschwindigkeits- und Trittfrequenzsensor 3](#_Toc79314582)

[4.3 Herzfrequenzsensor 4](#_Toc79314583)

[4.4 Android Applikation 4](#_Toc79314584)

[5 Software 4](#_Toc79314585)

[6 Installation von Zephyr auf Windows 10 4](#_Toc79314586)

[7 Nordic SDK in Visual Studio Code 7](#_Toc79314587)

[8 Klassendiagramm nRF5340 Teil 7](#_Toc79314588)

[9 BLE 8](#_Toc79314589)

[9.1 GAP Schicht 8](#_Toc79314590)

[9.2 GATT Schicht 9](#_Toc79314591)

[9.3 BLE: Peripherie 10](#_Toc79314592)

[9.4 BLE: Zentral 13](#_Toc79314593)

[9.5 Batterie Service Klient 22](#_Toc79314594)

[9.6 Data Service (Server) 23](#_Toc79314595)

[9.7 Verwendung des Data Service 27](#_Toc79314596)

[9.8 Konfigurationsmöglichkeiten 28](#_Toc79314597)

[10 Android Applikation 29](#_Toc79314598)

[10.1 Voreinstellungen 30](#_Toc79314599)

[10.2 Scan – und Verbindungsvorgang 31](#_Toc79314600)

[10.3 Data Service (Klient) 32](#_Toc79314601)

[10.4 Senden / Empfangen von Daten 33](#_Toc79314602)

[10.5 Nachrichtenanzeige 34](#_Toc79314603)

[10.6 Referenzen Applikation 35](#_Toc79314604)

[11 Datenfluss 36](#_Toc79314605)

[12 Tests 37](#_Toc79314606)

[13 Schlussfolgerung 38](#_Toc79314607)

[14 Selbständigkeitserklärung 38](#_Toc79314608)

[15 Anhang 39](#_Toc79314609)

[16 Quellenverzeichnis 48](#_Toc79314610)

[17 Literaturverzeichnis 49](#_Toc79314611)

# Abbildungsverzeichnis

[Abbildung 1: Schema 2](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314555)

[Abbildung 2: nRF5340 DK 3](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314556)

[Abbildung 3: Geschwindigkeits- und Trittfrequenzsensor 3](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314557)

[Abbildung 4: Herzfrequenzmesser Polar 4](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314558)

[Abbildung 5: BLE Schichtenmodell 8](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314559)

[Abbildung 6: Erklärung GAP 9](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314560)

[Abbildung 7: Erklärung GATT 9](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314561)

[Abbildung 8: Nordic Thingy:52 13](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314562)

[Abbildung 9: Berechnung RPM 20](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314563)

[Abbildung 10: Berechnung Geschwindigkeit 21](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314564)

[Abbildung 11: Sequenzdiagramm Battery Manager 23](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314565)

[Abbildung 12: CSC Activity 29](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314566)

[Abbildung 13: Scanner Activity 29](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314567)

[Abbildung 14: Splash Screen Activity 29](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314568)

[Abbildung 15: Sequenzdiagramm Scan- und Verbindungsvorgang 31](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314569)

[Abbildung 16: Sequenzdiagramm Senden / Empfangen von Daten 34](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314570)

[Abbildung 17: Sequenzdiagramm Datenfluss 36](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314571)

[Abbildung 18: Test mit Thingy und Nordic Applikation 37](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79314572)

# Tabellenverzeichnis

[Tabelle 1: Verwendete Software 4](#_Toc79156661)

[Tabelle 2: Nachrichtencodes 34](#_Toc79156662)

# Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während meiner Bachelorarbeit unterstützt haben.   
Zuerst gebührt mein Dank Herr Rieder Medard, der meine Bachelorarbeit betreut hat. Ebenfalls möchte ich mich bei Rudaz Patrice, Rossier Yoan und Clausen Michael, welche mir bei Fragen hilfreich zur Seite standen, bedanken.

# Einführung

## Projekteschreibung

Nordic Semiconductor[[1]](#footnote-1) (1) hat vor kurzem erklärt, dass die Weiterentwicklung des traditionellen BluetoothLow Energy[[2]](#footnote-2) Stacks, dem sogenannten «*Softdevice*», eingestellt wird und die Versionen eingefroren werden. Alle Neuentwicklungen werden auf Zephyr, einem aktuellen und sehr leistungsfähigen RTOS[[3]](#footnote-3), durchgeführt. Gleichzeitig hat Nordic eine neue Chip-Familie herausgebracht, die nRF53xxx. Dabei handelt es sich um Dual-Core-Bausteine, wobei ein Kern mit dem Radio- und einer mit der Anwendersoftware belegt ist.

## Projektziel

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines einfachen BT5-Demonstrators[[4]](#footnote-4) mit einem nRF5340 Entwicklungs-Kit und Zephyr. Der Demonstrator ist ein leichter elektrischer Fahrzeug Bordcomputer. Der nRF5340 muss von einem Smartphone erkannt werden können und aber auch in der Lage sein, eine Reihe von Sensoren/Aktoren zu erkennen, die im Fahrzeug installiert sind, wie z.B. Trittfrequenzsensoren und Geschwindigkeitssensoren. Das bedeutet, dass er gleichzeitig die Rolle eines zentralen und eines periphereren Bluetooth-Geräts spielen muss und so als Gateway[[5]](#footnote-5) fungiert.

# Schema

Im folgenden Schema ist das zuvor beschriebene Projektziel sehr gut ersichtlich. Die Hauptkomponente ist das nRF5340 Entwicklungs-Kit. Die Daten der Sensoren werden über das Kit an die Android Applikation weitergesendet.

Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK\_SPL.jpg

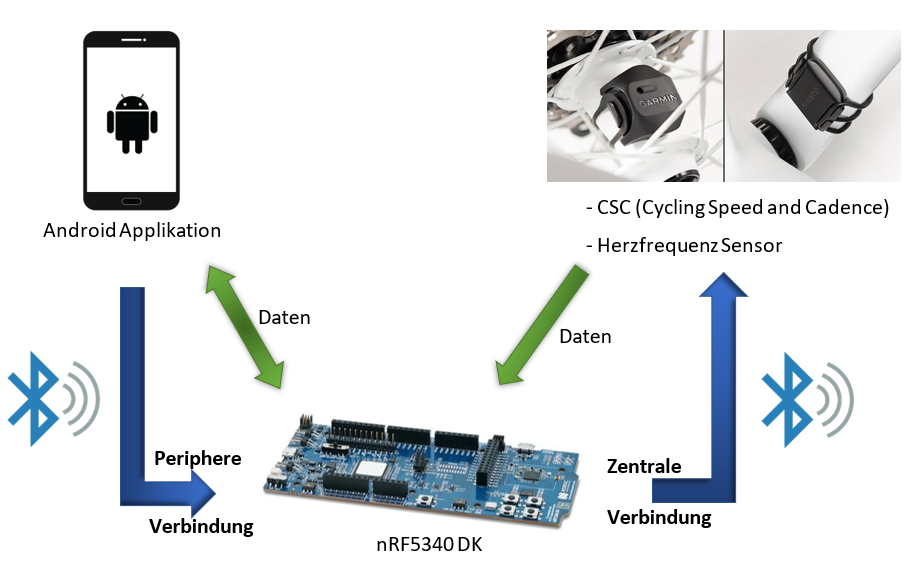


Abbildung : Schema

Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK\_SPL.jpg

Quelle: https://play-lh.googleusercontent.com/WshzYj8MVD1cDz55UP3z0Hl3eqUiBIkez5GSYr0yxwGSbVTkdLILlb0m0SXkqX-tWC

Die folgenden Schritte werden näher erläutert:

* Komponenten des Schema (Seite 2)
* Installation von Zephyr auf Windows 10 (Seite 4)
* Vorbereitungen für die Verwendung von Visual Studio Code (Seite 7)
* Ausführliche Erklärung, um eine periphere und zentrale Bluetooth Verbindung aufzubauen (Seite 8)
* Programmierung der Applikation in Android Studio (Seite 29)

# Komponenten

## nRF5340 Development Kit

Das nRF5340 DK (Abbildung 2) ist das Entwicklungs-Kit für den nRF5340 System-on-Chip (SoC) und enthält alles, was für den Einstieg in die Entwicklung benötigt wird, auf einer einzigen Platine.

Im Folgenden werden nur die Punkte aufgelistet, welche für diese Bachelorarbeit gebraucht wurden. Der nRF5340 SoC ist der Kern des Entwicklungskits. Er kombiniert einen Hochleistungs-Anwendungsprozessor mit einem voll programmierbaren Ultra-Low-Power-Netzwerkprozessor. Der nRF5340 unterstützt Bluetooth Low Energy. Das Entwicklungs-Kit wird via USB programmiert und mit Strom versorgt. Es gibt auch eine Batterie, welche bei Bedarf für die Stromversorgung verwendet werden kann.  
Auf dem Board sind vier programmierbare Taster und LEDs verbaut. (2)

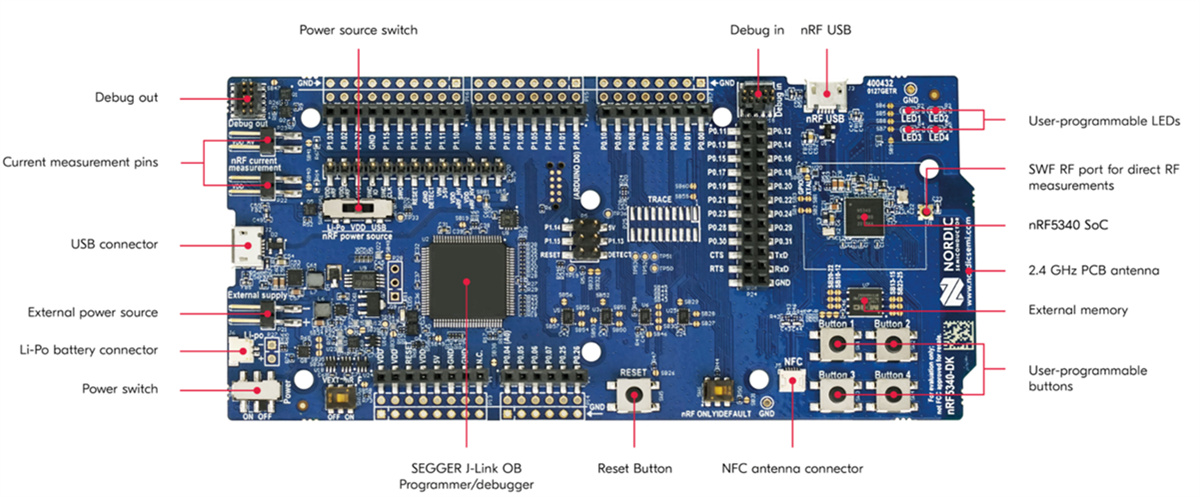


Abbildung : nRF5340 DK

Quelle: https://www.mouser.de/images/marketingid/2019/microsites/153282850/2020-12-21\_15-07-43.png

## Geschwindigkeits- und Trittfrequenzsensor

Die verwendeten CSC[[6]](#footnote-6) Sensoren (Abbildung 3) kommen von Garmin (3). Garmin gehört zu den weltweit führenden Unternehmen im Bereich Navigation für die Automobilbranche, sowie für die Bereiche Luftfahrt, Marine und Outdoor / Sport.  
Der Geschwindigkeitssensor wird auf eine der Radnaben[[7]](#footnote-7) montiert. Der Durchmesser kann direkt in der Applikation eingegeben werden. Der Trittfrequenzsensor wird an einer Seite an der Tretkurbel befestigt. Die gemessenen Daten werden per Bluetooth an den nRF5340 gesendet.



Abbildung : Geschwindigkeits- und Trittfrequenzsensor

Quelle: https://static.garmincdn.com/en/products/010-12103-00/g/cadence\_v03-eb1598f4-1e01-4e01-8691-8ddf6e108677.jpg

## Herzfrequenzsensor

Der verwendete Herzfrequenzsensor kommt von Polar (4). Polar ist seit 1977 führend im Bereich der Entwicklung von Herzfrequenz-Messgeräten und technischen Innovationen (5).



Abbildung : Herzfrequenzmesser Polar

Quelle: https://media.alltricks.com/hd/15743025eaaa35aa3e5d9.44386646.jpg

## Android Applikation

Um sich mit dem zuvor vorgestellten Entwicklungs-Kit verbinden zu können, wird eine Applikation benötigt. Diese wurde mit Android Studio (6) entwickelt.

# Software

|  |  |
| --- | --- |
| **Name** | **Version** |
| Windows | 10 pro |
| Microsoft Office | 365 |
| Visual Studio Code | 1.56.2 |
| Android Studio | 4.2.1 |

Tabelle : Verwendete Software

# Installation von Zephyr auf Windows 10

Im Folgenden ist eine Anleitung wie Zephyr auf Windows 10 installiert werden kann. Am Ende wird zum Testen ein Beispiel Code auf das Entwicklungs-Kit geladen. Die Anleitung kann auf der Webseite (7) von Nordic gefunden werden.

Benötigte Anwendungen:

* CMD (Windows-Eingabeaufforderung): Bereits vorhanden
* Windows PowerShell: Bereits vorhanden
* Chocolatey: Nicht vorhanden (Chocolatey ist eine Software-Verwaltungslösung)

*Installation von Chocolatey*

1. Windows PowerShell als Administrator öffnen
2. *«Get-ExecutionPolicy»* eingeben, wenn die Konsole *«restricted»* zurückgibt, muss Punkt 2.1 ausgeführt werden, ansonsten nicht.
   1. *«Set-ExecutionPolicy Bypass -Scope Process»* eingeben und die Anfrage mit *«J»* bestätigen
3. Nun kann Chocolately effektiv installiert werden. Dies geschieht mit folgendem Befehl:

«*Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))»*

Wenn keine Fehler erscheinen, ist Chocolatey installiert und kann verwendet werden.

*Installation von Zephyr und anderen Tools*

1. Windows PowerShell als Administrator öffnen
2. Folgende Befehle im Windows PowerShell eingeben:
   1. *«choco feature enable -n allowGlobalConfirmation»*
   2. *«choco install cmake --installargs 'ADD\_CMAKE\_TO\_PATH=System'»*
   3. *«choco install ninja gperf python git»*

Dies wird einen Moment dauern.

1. Windows PowerShell schliessen
2. CMD als Administrator öffnen und folgende Befehle ausführen
   1. *«pip3 install west»*
   2. *«cd %HOMEPATH%»*
   3. *«west init <zephyrFolder>»* (Hier wurde "zephyr\_workspace" gewählt)
   4. *«cd <zephyrfolder>»*
   5. *«west update»*
   6. *«west zephyr-export»*
   7. *«pip3 install -r %HOMEPATH%\<zephyrfolder>\zephyr\scripts\requirements.txt»*

Nun ist Zephyr und West installiert, fehlt noch die GNU Toolchain.

*GNU Tools*

1. Den folgenden Installer herunterladen: *«https://developer.arm.com/-/media/Files/downloads/gnu-rm/10-2020q4/gcc-arm-none-eabi-10-2020-q4-major-win32.exe?revision=9a4bce5a-7577-4b4f-910d-4585f55d35e8&la=en&hash=068C813EEFFB68060B5FB40E6541BDE7159AFAA0»*
2. Einen neuen Ordner im C: Verzeichnis: *«c:\gnu\_arm\_embedded»* erstellen
3. «Downloads» Ordner öffnen
4. Die vorhin heruntergeladen Datei *«gcc-arm-none-eabi-10-2020-q4-major-win32.exe»* ausführen.
5. Als Speicherort den zuvor erstellten Ordner auswählen *«C:\gnu\_arm\_embedded»*, und das automatisch hinzugefügte Unterverzeichnis entfernen.
6. Am Ende das Feld zum Hinzufügen des Pfades hinzufügen
7. Warten, bis die Installation beendet ist.

*JLink*

1. Den folgenden Installer herunterladen: *«https://www.nordicsemi.com/-/media/Software-and-other-downloads/Desktop-software/nRF-command-line-tools/sw/Versions-10-x-x/10-12-1/nRF-Command-Line-Tools\_10\_12\_1\_Installer\_64.exe»*
2. *«Downloads»* Ordner öffnen
3. Die vorhin heruntergeladene Datei *«nRF-Command-Line-Tools\_10\_12\_1\_Installer\_64.exe»* ausführen.
4. Alle Voreinstellungen beibehalten.

*Umgebungsvariablen*

1. Folgende Umgebungsvariablen mittels dieser Befehle erstellen
   1. *«GNUARMEMB\_TOOLCHAIN\_PATH=C:\gnu\_arm\_embedded»*
   2. *«ZEPHYR\_TOOLCHAIN\_VARIANT=gnuarmemb»*

*Nordic SDK initialisieren*

1. Neuen Ordner im C: Verzeichnis erstellen: «*c:\%HOMEPATH%\ncs*»
2. CMD öffnen, dieses Mal sind keine Administrator Rechte notwendig
3. *«cd %HOMEPATH%\<nordicPath>\zephyr»* (nordicPath ist hier "ncs")
4. *«west init -m https://github.com/nrfconnect/sdk-nrf --mr v1.5.1»*
5. *«west update»*

Dies wird einen Moment dauern.

*Test*

Als Beispielprojekt wurde das «*blinky*» Projekt von Zephyr verwendet. Dieses befindet sich im Ordner *«%HOMEPATH%\<nordicPath>\zephyr\samples\basic\blinky»* oder auf Github (8). Um das Projekt zu kompilieren[[8]](#footnote-8) und auszuführen müssen folgende Befehle in der Konsole eingegeben werden.

1. *«cd %HOMEPATH%\<nordicPath \zephyr»*
2. «*zephyr-env.cmd»*
3. *«west build -p auto -b <boardname> samples\basic\blinky»* (Hier wird das Board "nrf5340dk\_nrf5340\_cpuappns" verwendet)
4. *«west flash»*

Nun sollte die LED1 auf dem Development Board im Sekundentakt blinken.

# Nordic SDK in Visual Studio Code

Im Folgenden wird erklärt, wie ein Projekt in Visual Studio Code kompiliert, auf das Board geladen und gedebuggt[[9]](#footnote-9) wird. Das Projekt muss sich im zuvor erstellten Verzeichnis befinden. *«%HOMEPATH%\<nordicPath>\».*

Für dieses Beispielprojekt wurde ein Workspace, namens *«PerCen.code-workspace»* eingerichtet. Eine Vorlage von Rieder Medard wurde bereitgestellt. Diese Vorlage wurde angepasst und befindet sich im Anhang Ordner *«Anhänge»*. Wird diese Datei im Projektordner eingefügt, kann durch ein Doppelklick Visual Studio Code geöffnet werden.   
Je nach dem, in welchem Unterordner das Projekt erstellt wurde, müssen die Pfade für den *«zephyr»* Ordner und den des *«nrf»* Ordners angepasst werden. Ausserdem müssen die Pfade für die *«commands»* angepasst werden.  
Nun muss noch eine Zeile Code in der *«CMakeList.txt»* Datei hinzugefügt werden: *«set (BOARD nrf5340dk\_nrf5340\_cpuappns)»*. Diese gibt an welches Board verwendet wird. Wenn nicht das nRF5340 Development Kit verwenden wird, muss der Board Name angepasst werden. Eine Liste aller Namen ist auf der Webseite (9) von Nordic ersichtlich. Ausserdem müssen alle Klassen, welche kompiliert werden sollen, hier aufgelistet werden. Die *«CMakeList.txt»* sieht für dieses Projekt wie folgt aus.

  
  
Jetzt sind alle Voreinstellungen abgeschlossen. Über den Menu Punkt *«Terminal -> Run Task»* kann die gewünschte Operation ausgewählt werden. Die Befehle hinter diesen Auswahlmöglichkeiten befinden sich in der *«PerCen.code-workspace»* Datei. Zuerst muss der Code kompiliert werden. Soll der Debug Modus gestartet werden, muss in der Menu Leiste *«Run -> Start Debugging»* gewählt werden.

# Klassendiagramm nRF5340 Teil

Die Kernklasse ist die *«DeviceManager»* Klasse. Hier werden die Bluetooth Verbindungen aufgebaut und die notwendigen Charakteristiken eingeschrieben. Die Daten der Sensoren werden auch hier empfangen und anschliessend in der *«DataCSC»* Klasse gespeichert. Hier befinden sich auch die Berechnungen für die Geschwindigkeit und die Trittfrequenz. Um die Daten weiter an die Android Applikation zu senden, wurde ein Service implementiert. Dieser befindet sich in der *«DataService»* Klasse. Um den aktuellen Batteriestand der Sensoren zu kennen, wurde der *«GATT Battery Service Client»* (BAS) (10) von Nordic verwendet. Hierfür wurde eine Klasse *«BatteryManager»* erstellt. Ausserdem gibt es eine Klasse *«main»*, hier wird nur entschieden, ob der nRF5340 als zentrales und/oder als peripheres Gerät initialisiert wird. Im Anhang 1 befindet sich das Klassendiagramm. Im Folge dieses Berichts wird auf die einzelnen Klassen eingegangen.

# BLE

Folgende Aufgaben werden im BLE Teil behandelt:

* Initialisierung des nRF5340 DK, sodass es als peripheres und zentrales Gerät arbeiten kann
* Werben (Advertising) des Boards, sodass sich eine Applikation (zentrales Gerät) mit ihm verbinden kann (Peripherer Teil)
* Scan nach den zu verbindenden Sensoren (Zentral Teil)
* Verbindungsaufbau mit den Sensoren (Zentral Teil)
* Datenaustausch: Sensoren 🡺 nRF5340 DK 🡺 Applikation

Ein Sequenzdiagramm, welches den Ablauf dieser Schritte zeigt, befindet sich im Anhang 2.

Bluetooth Low Energy (BLE) ist eine Funktechnologie. Sie ist für geringen Stromverbrauch optimiert. Sie ermöglicht die Übertragung von Daten von einem Gerät zu einem anderen, z. B. die Übertragung der Temperatur von einem Sensor an ein anderes Gerät, das diese Daten sammelt. Die BLE-Architektur besteht aus mehreren Schichten (Abbildung 5). (11)

In den folgenden Abschnitten wird auf einige dieser Schichten eingegangen.

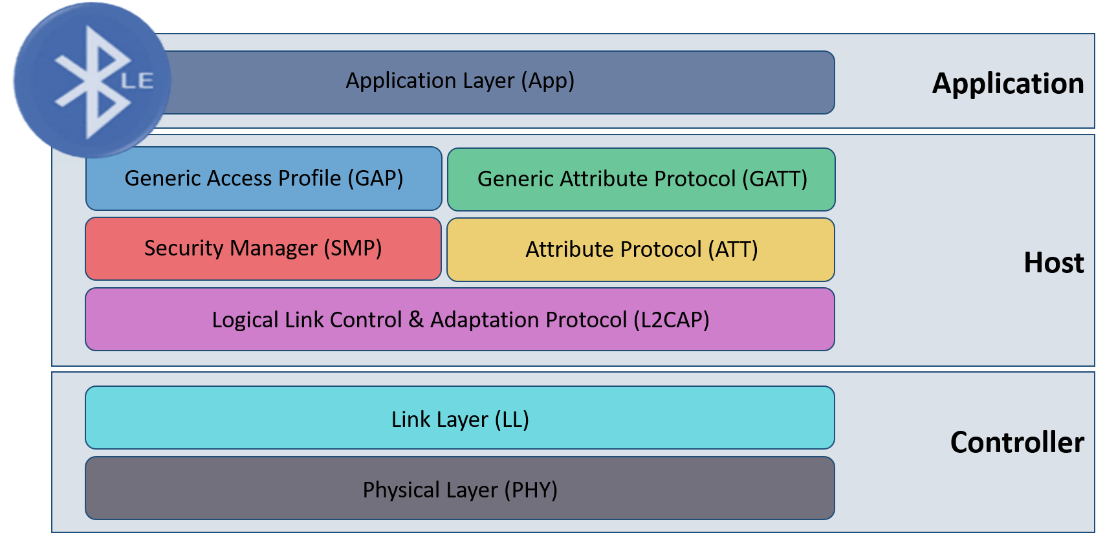


Abbildung : BLE Schichtenmodell

Quelle: https://miro.medium.com/max/1838/1\*7R\_hkwuk8v7gClsVXtoqPw.png

## GAP Schicht

Die Bezeichnung GAP bedeutet folgendes:

*«Generic Access Profil»*, was übersetzt *«Generisches Zugriffsprofil»* bedeutet. Es steuert die Verbindungen. GAP macht das Gerät für die Aussenwelt sichtbar und bestimmt wie zwei Geräte miteinander interagieren können. (12)   
Die meisten BLE Geräte können zwei Rollen haben, entweder eine zentrale oder periphere Rolle. Der nRF5340 kann beides zur selben Zeit. Das bedeutet, dass der Chip zuerst um sich wirbt (advertising), sodass sich die Applikation mit dem Board verbinden kann. Anschliessend scannt der nRF5340 nach den gewünschten Sensoren und baut eine Verbindung zu diesen auf.

Nach dem Verbindungsaufbau mit einem Bluetooth Low Energy Gerät, stellt dieses eine Auswahl von *«Generic Attribute Profile Services»* zur Verfügung. Jeder Service ist durch eine eineindeutige Nummer (UUID*[[10]](#footnote-10)*) gekennzeichnet und umfasst mehrere *«GATT Characteristics»*. Jede Charakteristik*[[11]](#footnote-11)* ist wiederum durch eine UUID, mehreren Flags wie z.B. *«read»* oder *«write»* und durch ein Datenwert (Byte Array[[12]](#footnote-12)) definiert. Dies ist in Abbildung 6 ersichtlich.

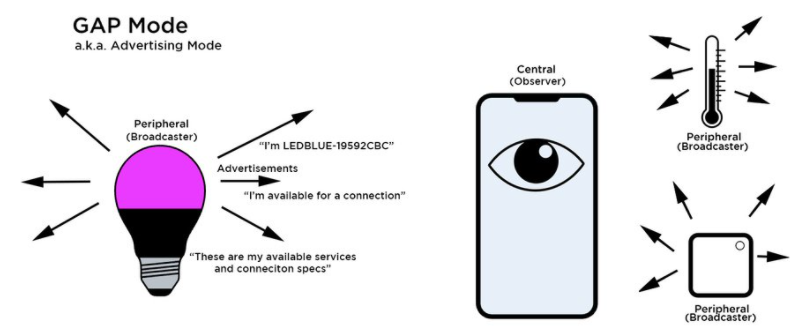


Abbildung : Erklärung GAP

Quelle: https://learn.adafruit.com/assets/86833

## GATT Schicht

Die Bezeichnung GATT bedeutet folgendes:

*«Generic Attribute Profile»*, was übersetzt *«Generisches Attributprofil»* bedeutet. Es bezeichnet die Art und Weise, wie zwei Bluetooth Low Energy Geräte Daten hin und her übertragen(Abbildung 7). (13)

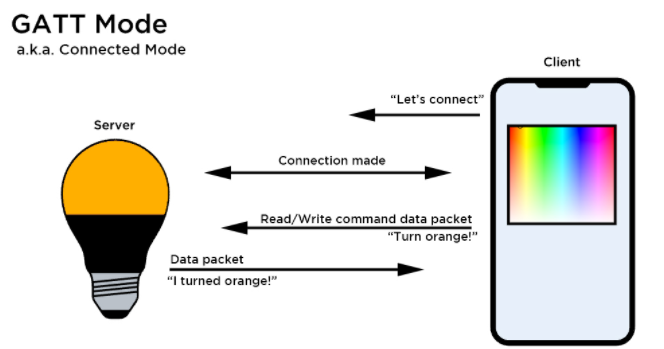


Abbildung : Erklärung GATT

Quelle: https://learn.adafruit.com/assets/86833

## BLE: Peripherie

In diesem Abschnitt wird alles rund um den Verbindungsaufbau zwischen dem nRF5340 und einem zentralen Gerät (Hier ein Smartphone) genauer erläutert. Die notwendigen Informationen wurden vom *«peripheral\_lbs»* (14) Beispiel von Nordic hinzugezogen. Dieser Code befindet sich in der *«DeviceManager»* Klasse. Bevor auf die einzelnen Schritte eingegangen werden kann, müssen einige Voreinstellungen getroffen werden. Im Projektordner muss eine *«prj.conf»* Datei erstellt werden, wenn diese noch nicht vorhanden ist. Anschliessend müssen folgende Konfigurationen eingestellt werden:



Diese Konfigurationen sind dringend notwendig.

Des Weiteren müssen einige Klassen eingebunden werden, ansonsten gibt der Kompilierer an, dass er gewisse Funktionen und Definitionen nicht kennt. Diese *«Includes»* befinden sich alle in der *«DeviceManager.h»* Datei.



Nun, da alle Konfigurationen einstellt und die nötigen Klassen eingebunden sind, kann in einem ersten Schritt der nRF5340 als peripheres Gerät konfiguriert werden. Mit einer bereits existierenden Applikation von nRF, *«nRF Connect»* (15), kann dieser Teil getestet werden. Es werden nur die elementaren Teile zum Verbindungsaufbau erklärt.

1. Bluetooth aktivieren

  
Diese Funktion gibt den Wert ‘0’ zurück, wenn die Aktivierung erfolgreich war. Als Parameter kann eine Funktion angegeben werden, welche aufgerufen wird, sobald das die Aktivierung abgeschlossen ist.

1. Nun kann die Rückrufstruktur definiert werden (callback). Danach müssen noch die Funktionen «*connected»* und «*disconnected»* deklariert und implementiert werden. Dies geschieht in einem späteren Schritt.



Jetzt müssen die Rückruffunktionen noch eingeschrieben werden. Dies geschieht über die Funktion:



Als Parameter wird die Adresse der Struktur mit den Rückruffunktionen übergeben.

1. Danach muss der nRF5340 beginnen um sich zu werben, dem sogenannten *«advertising»*.



Diese Methode besitzt einige Parameter. Als erstes muss der Methode eine Struktur mit den notwendigen Informationen übergeben werden. Hier wurde ein bereits existierendes *«define»* verwendet, nämlich *«BT\_LE\_ADV\_CONN»*.

Die Strukturen *«ad»* und *«sd»* sind Daten Arrays, die in Werbepaketen (advertising), bzw. Scan-Antwortpaketen (scanning) verwendet werden. Diese werden wie folgt deklariert.



Die zwei verbleibenden Parameter beinhalten die Informationen, über die Grösse dieser Strukturen. Der gesamte Methodenaufruf sieht nun folgendermassen aus.



Diese Funktion gibt den Wert ‘0’ zurück, wenn der *«advertising»* Vorgang erfolgreich gestartet wurde, ansonsten einen negativen Fehlercode.

1. Sobald sich nun ein zentrales Gerät, hier ein Smartphone, mit dem nRF5340 verbindet, wird die Rückruffunktion *«connected»* aufgerufen. Der Prototyp dieser Funktion sieht wie folgt aus:



Der erste Parameter ist die Adresse der Verbindung, die aufgebaut wurde. Der zweite Parameter ist ein Fehlercode, wenn dieser Wert ‘0’ beträgt, wurde die Verbindung erfolgreich aufgebaut.

Nun muss der Unterschied zwischen einer zentralen und periphereren Verbindung gemacht werden. Dies wird wie folgt erreicht:



Mittels der Funktion *«bt\_conn\_get\_info»*, welche als Parameter die Adressen der bestehenden Verbindung und die einer *«bt\_conn\_info»* Struktur nimmt, können diverse Informationen der Verbindung gewonnen werden. Die Funktion gibt den Wert ‘0’ zurück, wenn der Vorgang erfolgreich war.

Mittels folgender Abfrage kann getestet werden, ob es sich beim verbundenen Gerät um ein zentrales oder ein peripheres Gerät handelt.



In dieser Abfrage können nun diverse Anzeigemöglichkeiten hinzugefügt werden, welche dem Nutzer zeigen, dass ein peripheres Gerät verbunden ist.

Im nächsten Schritt wird die Konfiguration als *«zentral»* hinzugefügt.

## BLE: Zentral

In diesem Abschnitt wird auf den Verbindungsaufbau zwischen einem peripheren Gerät und dem nRF5340 eingegangen. Es werden zwei verschiedene Varianten vorgestellt. Bei der ersten Variante wurde als Testgerät ein Nordic Thingy:52 (Abbildung 8) verwendet. Die zweite Variante ist diejenige, welche mit den verwendeten Sensoren benutzt wird.

Variante mit Thingy:

*Das Nordic Thingy:52 ist ein kompaktes, energieoptimiertes Multi-Sensor-Entwicklung-Kit. Es ist eine einfach zu bedienende Entwicklungsplattform, die Entwicklern hilft, IoT-Prototypen und -Demos zu erstellen, ohne die Notwendigkeit, eigene Hardware zu bauen.* (16)



Abbildung : Nordic Thingy:52

Quelle: https://www.mouser.ch/images/marketingid/2017/img/116326113\_Nordic\_Thiny52IoTSensorDevelopmentKit.png?v=052820.0204

Es wird nur auf die elementaren Teile zum Verbindungsaufbau eingegangen. Die nötigen Informationen wurden vom *«central»* Beispiel von Zephyr hinzugezogen. Dieses befindet sich auf Github. (17)

Da bei der Initialierung des peripherie Teils, alle allgemeinen Initialisierungen ausgeführt wurden, sind diese hier nicht mehr notwendig und es kann direkt mit der Initialisierung des Scan-Vorgangs begonnen werden.

1. Um den Scan Vorgang starten, braucht es eine Struktur mit den nötigen Informationen.



Anstatt einer Struktur können auch die default Einstellungen gewählt werden.

Es wird eine Rückruffunktion benötigt, sobald das ein Gerät gefunden wird, wird diese Funktion aufgerufen.



Die Parameter enthalten Informationen wie, die Adresse, den RSSI Wert, dies ist der Leistungspegel und gibt an, wie nah oder fern ein Gerät ist, den Typ, und einen Zeiger auf einen Puffer.

Nun kann ein Scan Vorgang gestartet werden:



Die Funktion gibt den Wert ‘0’ zurück, wenn der Scan Vorgang erfolgreich gestartet wurde.

1. Sobald ein Gerät in der Nähe gefunden wird, wird die Funktion *«deviceFound»* aufgerufen. Hier wird zuerst kontrolliert, ob zum gefundenen Gerät eine Verbindung aufgebaut werden kann.

Ausserdem soll nur eine Verbindung aufgebaut werden, wenn sich das Gerät nahe beim Entwicklungs-Kit befindet.



Je kleiner der RSSI Wert, desto näher befindet sich das Gerät. Die Einheit ist Dezibel. Ein Wert von -70db entspricht einer noch knapp guten Verbindung.

Nun, da ein verbindbares, in der Nähe liegendes Gerät gefunden wurde, kann der Scan Vorgang abgebrochen werden.



Die Funktion gibt den Wert ‘0’ zurück, wenn der Scan Vorgang erfolgreich abgebrochen wurde.

Anschliessend wird die Adresse des gefundenen Geräts in einen String[[13]](#footnote-13) umgewandelt. So kann diese verglichen werden. Dafür gibt es eine Funktion.



Die Adresse im String Format befindet sich nun in der Variablen *«addr\_str»*, welche zuvor deklariert wurde. Nun kann die Adresse mit derjenigen des zu verbindenden Geräts verglichen werden. Hier ist es von Vorteil, wenn getestet wird, ob sich die gewünschte Adresse in *«addr\_str»* befindet. Denn am Ende von *«addr\_str»* befinden sich noch weitere Charaktere.



1. Jetzt, da sichergestellt wurde, dass es sich um das korrekte Gerät handelt, kann eine Verbindung aufgebaut werden.



Der erste Parameter ist die Adresse des zu verbindenden Geräts. Der zweite Parameter beinhaltet Informationen zum Verbindungsaufbau, hier wurden die Standardwerte verwendet. Der dritte Parameter ist der Verbindungsparameter, auch hier wurden die Standardwerte verwendet. Der vierte und letzte Parameter ist die Adresse eines Verbindungsobjektes. Dieses zeigt momentan auf nichts. Wenn die Verbindung aufgebaut ist, wird in dieser Variablen die Adresse der erzeugten Verbindung gespeichert sein. Die Funktion gibt den Wert ‘0’ zurück, wenn die Verbindung erfolgreich aufgebaut wurde.

In der Rückrufmethode *«connected»* muss wie zuvor im Peripherie Teil, getestet werden, ob es sich um eine zentrale oder periphere Verbindung handelt. Dafür muss zuerst wieder die *«bt\_conn\_get\_info»* Funktion aufgerufen werden. Eine genauere Erklärung befindet sich unter Punkt 4 im Kapitel *«BLE: Peripheral»*. Im Fall, dass ein Fehler bei der Verbindung aufgetreten ist, sprich ein negativer Error Code im Parameter, wird der Scan Vorgang erneut gestartet und die Attribute wieder auf *«NULL»* initialisiert.

Jetzt besteht eine Verbindung zwischen dem nRF5340 und der Applikation und einem peripheren Gerät.

Variante mit den CSC (Cycling Speed and Cadence) / Herzfrequenz Sensoren:

Mit Hilfe des Cycling Speed and Cadence Service (CSCS), ein Service für die Geschwindigkeit und die Trittfrequenz beim Radfahren, ist es möglich, sich mit den Sensoren von Garmin via Bluetooth zu verbinden. Anschliessend werden von den Sensoren regelmässig Daten übermittelt. Hierfür muss die notwendige Charakteristik benutzt werden, nämlich die *«CSC Measurement»*, die Messcharakteristik. Es werden zwei Sensoren benötigt, einer für die Geschwindigkeit und einer für die Trittfrequenz.

Sobald, dass alle Verbindungen zu den CSC Sensoren aufgebaut wurden, wird nach dem ausgewählten Herzfrequenzsensor gesucht und eine Verbindung zu diesem aufgebaut. Anschliessend wird auch hier eine Charakteristik verwendet, um die Daten zu empfangen, nämlich die *«HRS Measurement»* Charakteristik. In der folgenden Erklärungen wird nur auf den Verbindungsaufbau und das Abonnieren der Services der CSC Sensoren eingegangen. Für den Herzfrequenzsensor können die Erklärungen adaptiert werden.

Im vorderen Schritt wurde erklärt, wie eine zentrale Verbindung des nRF5340 mit einem Thingy:52 aufgebaut wird. Nun wird das Thingy mit den Sensoren von Garmin ersetzt. Da die Adressen dieser Sensoren nicht statisch sind, sie ändern sich, wenn die Batterie ausgetauscht wird, ist es nicht möglich gleich vorzugehen wie bei der Verbindung mit einem Thingy. In diesem Fall wurde das Scanning Modul (18) von Nordic verwendet. Hier kann bereits vor dem Beginn des Scanvorgangs, ein Filter gesetzt werden, sodass nur Sensoren mit einer *«CSC UUID»* erkannt werden. Im Folgenden wird erklärt, wie eine Verbindung mittels des Scanning Moduls zu den CSC Sensoren aufgebaut wird.   
Die nötigen Informationen wurden vom *«lte\_ble\_gateway»* Beispiel von Nordic hinzugezogen. Dieses ist auf Github(19) verfügbar.

1. Es werden ein Scan Parameter und ein Scan Initialisierungs-Parameter benötigt. Diese werden mittels Strukturen wie folgt definiert. Der Scan Parameter ist derselbe wie zuvor bei der Verbindung mit einem Thingy.



Mit der *«scanInit»* Strukur wird zuerst der *«scanParam»* als Referenz übergeben. Mit *«connect\_if\_match = 0»* wird erreicht, dass sich der nRF5340 nicht automatisch mit einem Gerät verbindet, wenn eines gefunden wird. Dies kann manuell erreicht werden. Als Verbindungsparameter wurden die Standardeinstellungen gewählt.

Nun wird das Initialisierungsmakro für das Scanning Modul verwendet. Der Prototyp sieht wie folgt aus:



*«\_name»* ist der Name der Rückruffunktion und kann frei gewählt werden. Nun folgen die Rückruffunktionen für den Fall, dass ein Gerät mit dem eingestellten Filter gefunden wurde (match\_fun), dass ein Gerät gefunden wurde, welches aber nicht auf die Filter zutrifft (no\_match\_fun), dass ein Fehler aufgetreten ist (error\_fun) und der Fall, dass eine Verbindung aufgebaut wird (connecting\_func). Es müssen nicht alle Funktion implementiert werden. In diesem Fall wurde die *«match\_fun»* und *«error\_fun»* implementiert. Das Initialisierungsmakro sieht wie folgt aus:



Nun kann das Scanning Modul initialisiert werden:



Als Parameter wird eine Referenz zur Initialisierungsstruktur übergeben. Als nächstes müssen die Rückruffunktionen noch eingeschrieben werden.



Als nächstes wird der korrekte Filter eingestellt. Die verwendeten Parameter sind bereits in der *«scan.h»* Bibliothek definiert.



Mit diesem Filter werden nun nur Geräte mit einer *«CSC UUID»* erkannt. Die Funktion gibt den Wert ‘0’ zurück, wenn die Filter erfolgreich gesetzt wurden.

Als letzter Schritt müssen die Filter noch aktiviert werden. Dies erfolgt durch den Aufruf folgender Funktion:



Mit dem zweiten Parameter kann bestimmt werden, dass ein Gerät auf alle Filter passen muss (wenn mehrere definiert wurden), oder ob es ausreicht, wenn ein Filter zutrifft. In diesem Fall wird nur ein Filter verwendet, darum wird diese Einstellung nicht benötigt. Die Funktion gibt den Wert ‘0’ zurück, wenn die Filter erfolgreich aktiviert wurden.

Mit dem Aufruf



wird der Scan Vorgang gestartet. Sobald ein Gerät, auf welches der Filter zutrifft, gefunden wird, befindet man sich in der *«scanFilterMatch»* Funktion. Die Funktion gibt den Wert ‘0’ zurück, wenn der Scan Vorgang erfolgreich gestartet wurde.

1. Wurde ein passender Sensor gefunden, wird in der *«scanFilterMatch»* Funktion zunächst der Scan Vorgang abgebrochen. Anschliessend wird eine Verbindung zu diesem Sensor aufgebaut. Die Funktion gibt den Wert ‘0’ zurück, wenn die Verbindung erfolgreich aufgebaut wurde.



1. Anschliessend, wenn die Verbindung aufgebaut wurde, wird die *«connected»* Funktion aufgerufen. Hier wird zwischen einem zentralen oder periphereren Verbindungsaufbau unterschieden.



Zuerst wird die Adresse der Verbindung gespeichert. Anschliessend folgt die Einschreibung des *«CSC Service»*. Hierfür werden Rückruffunktionen benötigt. Diese werden in einer Struktur gespeichert und später als Parameter übergeben.



Nun kann die Suche nach dem Service gestartet werden. Als Parameter werden die Adresse der erstellten Verbindung, der zu suchende Service, die Adresse der Rückruffunktionsstruktur, sowie ein Kontext übergeben. Als Kontext kann *«NULL»* übergeben werden. Der Funktionsaufruf sieht wie folgt aus:



Die Funktion gibt den Wert ‘0’ zurück, wenn der Suchvorgang erfolgreich gestartet wurde. Wird der Service im verbundenen Gerät gefunden, wird die *«discoveryCompleted»* Funktion aufgerufen. Hier muss die *«CSC Measurement»* Charakteristik eingeschrieben werden. Der Prototyp sieht wie folgt aus.



1. Als erstes wird eine *«bt\_gatt\_subscribe\_params»* Struktur definiert. Diese gibt an, welche Funktion aufgerufen wird, wenn Daten empfangen werden. Des Weiteren wird hier eingestellt, dass diese Rückruffunktion aufgerufen werden soll, wenn sich die Werte der Charakteristik ändern. Diese Struktur sieht wie folgt aus. Die Rückruffunktion wird in einem späteren Schritt implementiert.



Nun werden zwei *«bt\_gatt\_dm\_attr»* Attribute benötigt.



Das erste Attribut, *«chrc»*, wird benötigt, um die Charakteristik mittels der UUID zu bekommen.



Der Parameter *«dm»* ist bereits als Parameter in der Rückruffunktion enthalten und wird direkt weitergegeben. Die UUID befindet sich bereits in der *«scan.h»* Bibliothek.

Mit dem zweiten Attribut, *«desc»*, wird der Deskriptor mittels der UUID erhalten.



Nun wird eine weitere Einstellung im zuvor definierten *«param»* vollzogen. Der *«value handler»* wird zugewiesen.



Jetzt wird noch ein Deskriptor benötigt. Der sogenannte *«Client Characteristic Configuration Desciptor»* (CCCD). Dies erfolgt ähnlich wie zuvor. Es muss nur die UUID geändert werden.



Nun muss dieser auch wieder im *«bt\_gatt\_subscribe\_params param»* gespeichert werden. Dieses Mal wird er dem *«ccc handler»* zugewiesen.



Mit der folgenden Funktion kann die Charakteristik abonniert werden. Als Parameter werden eine Referenz zur Verbindung, sowie der konfigurierte *«param»* übergeben. Die Funktion gibt den Wert ‘0’ zurück, wenn die Einschreibung erfolgreich war.



Da nur jeweils eine Erkennungs-Prozedur zur selben Zeit gestartet werden kann, muss nun diese freigegeben werden. Dies geschieht durch folgenden Funktionsaufruf.



Nun ist es möglich Daten der CSC Sensoren zu erhalten.  
Mit derselben Vorgehensweise kann beispielsweise auch die Taster Charakteristik eines Thingy’s eingeschrieben werden. Es müssen lediglich die UUIDs geändert werden.

Im folgenden Schritt wird gezeigt, wie die Daten analysiert werden.

1. Die Datenanalyse befindet sich in der Rückruffunktion *«onReceived»*, diese sieht wie folgt aus.



Als Parameter werden Referenzen auf die Verbindung, die Abonnenten-Parameter, die Daten und die Länge der Daten erhalten. Der erste Wert des Datenzeigers beinhaltet den Typ des Sensors. Dieser gibt an, ob die erhaltenen Daten vom Geschwindigkeitssensor (Wert ‘0’) oder vom Trittfrequenzsensor (Wert ‘1’) stammen. Mit der folgenden Zeile kann der Typ aus den Daten gelesen werden.



Der Wert an der Stelle 2, also *«((uint8\_t\*) data) [1]»* gibt die gesamten Umdrehungen, welche zum gesendeten Zeitpunkt gemessen wurden an. An der Stelle 6, also *«(uint8\_t\*) data) [5]»* befindet sich der letzte Ereigniszeitwert für den Geschwindigkeitssensor. An der Stelle 4, der letzte Ereigniszeitwert für den Trittfrequenzsensor. Mit diesen Daten können die aktuelle Fahrtgeschwindigkeit und die Trittfrequenz berechnet werden.

Information  
Der Zähler im Sensor, welcher die Umdrehungen zählt, ist vom Typ *«uint32»* und kann somit 4'294'967'296 Umdrehungen zählen. Bei einer Annahme eines Radumfanges von 2,1 Meter (66,8cm Durchmesser), so kann eine Distanz von 9'019'431 Kilometer gefahren werden, bis es einen Überlauf gäbe. Da der Sensor eine durchschnittliche Lebensdauer von 5 Jahren hat und ein Spitzenradfahrer bis zu 15'000 Kilometer im Jahr zurücklegt (75'000km in 5 Jahren), übertrifft dieser Wert die Erwartung deutlich. (20)

1. Nun folgen Erklärungen zu den Berechnungen. Wichtig ist, dass die alten Werte gespeichert werden müssen. Um die Berechnungen durchführen zu können, werden die Differenzen benötigt. Die Formeln wurden im *«CSC Spezifikation»* (20) Dokument auf Seite 14 gefunden. Ausserdem wurde ein vorjähriger Bericht (21) zugezogen.

Trittfrequenz:

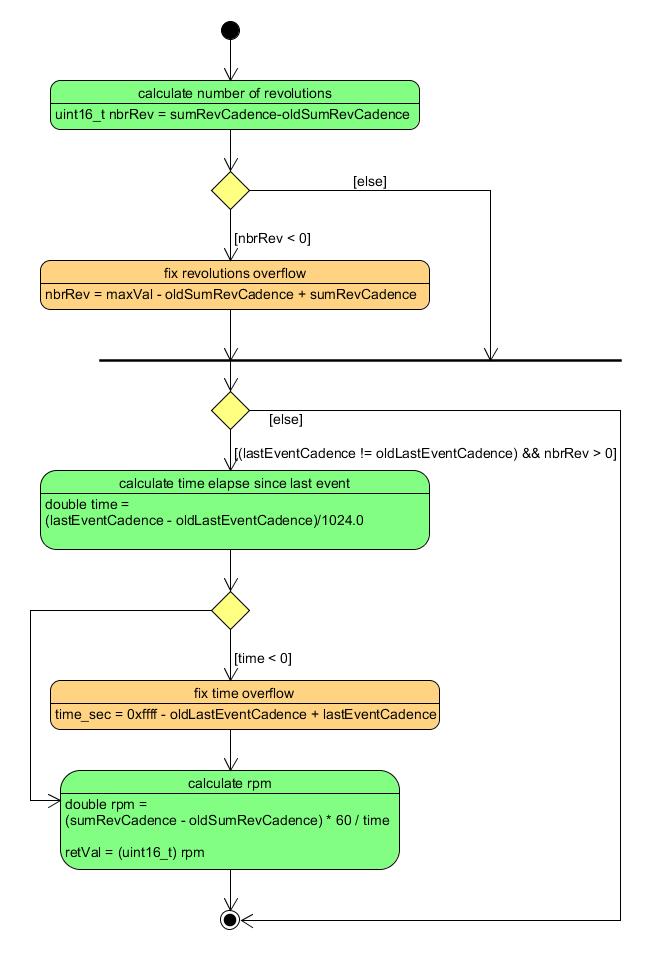


Abbildung : Berechnung RPM

Quelle: Eigene Darstellung

Geschwindigkeit:

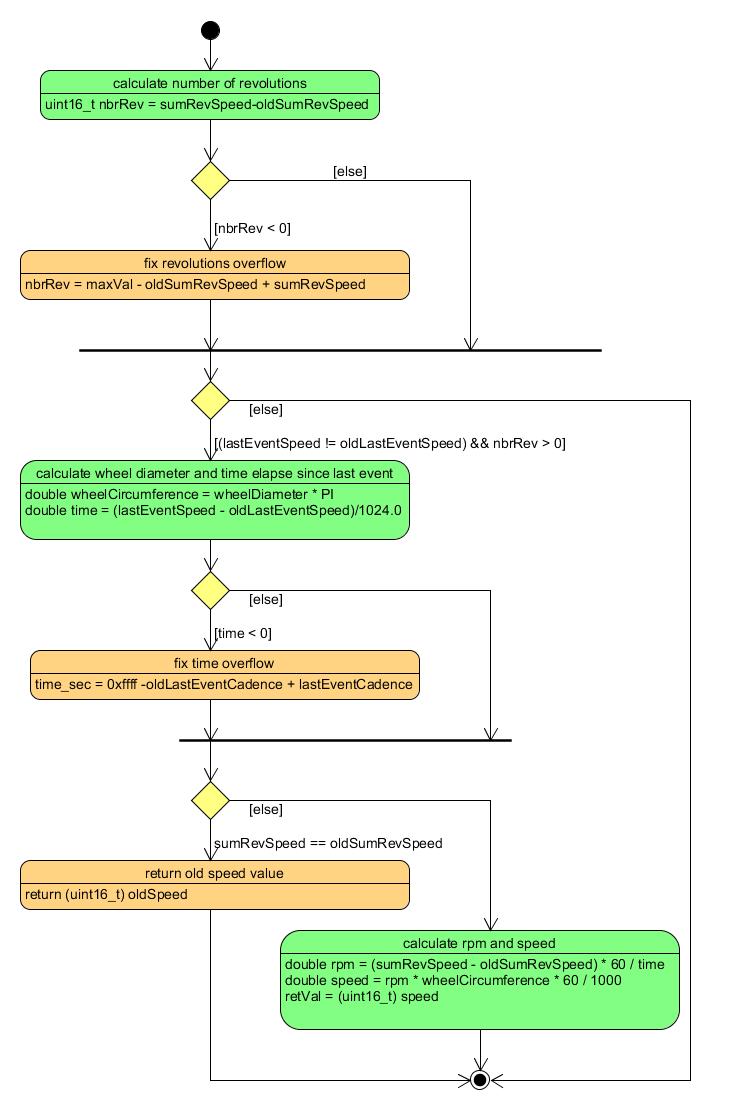


Abbildung : Berechnung Geschwindigkeit

Quelle: Eigene Darstellung

Die Geschwindigkeit wird in der Einheit *«km/h**x 100»* an den *«DeviceManager»* übergeben. So werden anschliessend die Geschwindigkeit auf den Hundertstel genau in der Applikation angezeigt.

## Batterie Service Klient

In diesem Kapitel wird auf die Implementierung des Batterie Service Klient eingegangen. Wichtig zu erwähnen ist, dass dies eine .c Klasse ist. Dies ist notwendig, da es ansonsten Probleme bei der Kompilierung gibt. Es wird nur auf die Nutzung des Services für einen Sensor eingegangen. Es muss nur folgende Datei eingeschlossen werden.

  
Um den Batterie Stand eines Gerätes zu erhalten, muss als erstes die Funktion *«initBatteryManager»* aufgerufen werden. Hier wird die Initialisierungsfunktion vom *«BAS Client»* aufgerufen. Dafür wird eine *«bt\_bas\_client»* Struktur benötigt.



Anschliessend kann nach dem Batterie Service gesucht werden. Nicht alle Geräte unterstützen diese Funktion. Dies wird mittels der Funktion *«gatt\_discover\_battery\_service(struct bt\_conn \*conn)»* erreicht. In dieser Funktion werden zuerst die Rückruffunktionen definiert.



Danach kann die Suche gestartet werden. Es wird dieselbe Funktion verwendet wie im Abschnitt *«BLE: Zentral»*, wo nach dem CSC Service gesucht wurde. Hier wird die UUID des Batterie Services verwendet. Diese Funktion gibt den Wert ‘0’ zurück, wenn der *«Discover»* Vorgang erfolgreich gestartet wurde, ansonsten einen negativen Fehlercode.



Sobald der Service gefunden wurde, wird die Funktion *«discovery\_completed\_cb(struct bt\_gatt\_dm \*dm, void \*context)»* aufgerufen. Im Fall, dass der Service nicht gefunden wurde, oder ein Fehler aufgetreten ist, werden die anderen Rückruffunktionen aufgerufen.  
Um das Verbindungsattribut, welches in der vorderen Funktion übergeben wurde, mit dem der Instanz des Modules zu verbinden muss folgender Aufruf getätigt werden. Dies macht es möglich, mehrere Geräte mit dem Batterie Service zu verknüpfen. Die Funktion gibt den Wert ‘0’ zurück, wenn die Vorgang erfolgreich war, ansonsten einen negativen Fehlercode.

  
Dies war bereits alles. Es ist möglich, immer nach einem gewissen Zeitintervall, eine Benachrichtigung zu erhalten. Dies wird hier nicht verwendet. Es wird von immer wieder der aktuelle Batterie Stand abgefragt.   
Da nur jeweils eine Erkennungs-Prozedur zur selben Zeit gestartet werden kann, muss nun diese freigegeben werden. Dies geschieht durch folgenden Funktionsaufruf.



Mit dem folgenden Aufruf kann der aktuelle Batterie Stand abgefragt werden.



In der Rückruffunktion *«read\_battery\_level\_cb\_speed»* wird der Batterie Stand des Geschwindigkeit Sensors als Parameter erhalten.

Im folgenden Sequenzdiagramm (Abbildung 11) ist dieser Ablauf noch genauer erläutert.

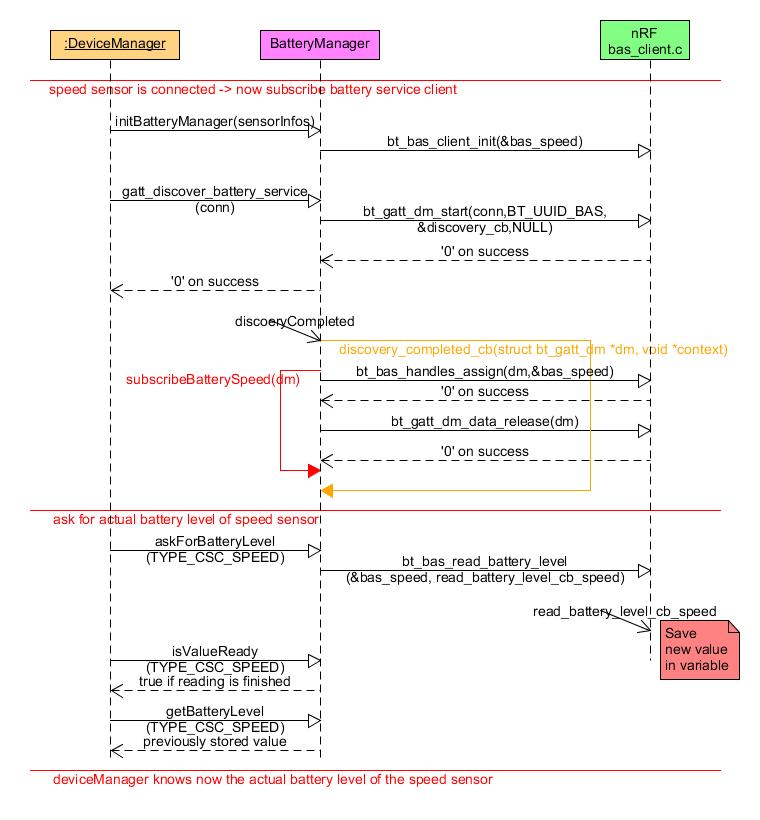


Abbildung : Sequenzdiagramm Battery Manager

Quelle: Eigene Darstellung

## Data Service (Server)

In diesem Schritt werden die Daten an die Android Applikation gesendet. Um dies zu erreichen, musste ein neuer Service implementiert werden, ein RX-TX Service (Receive-Transmitt). *«Transmitt»,* um die berechneten Werte zu schicken, und *«receive»,* um Daten von der Applikation zu erhalten. Der Nutzer kann in der Applikation den Durchmesser des Fahrrades eingeben. Dieser wird benötigt, um die Geschwindigkeit zu berechnen.  
Um den Server Teil (nRF5340) des Services zu implementieren wurde ein Tutorial (22) von Nordic befolgt. Dies wird nun genauer erläutert. Der Klient Teil (Android Applikation) wird im Kapitel 9 Android Applikation genauer erläutert.

Als erstes muss eine neue Klasse erstellt werden, in diesem Fall wurde der Name *«DataService»* vergeben. In der *«DataService.h»* Datei muss folgendes *«Include»* gemacht werden.



Nun wird eine Initialisierungsfunktion benötigt. Diese muss von der *«DeviceManager»* Klasse aufgerufen werden, bevor Daten gesendet oder empfangen werden können.



Es werden mindestens zwei Puffer benötigt, Einen zum Senden und Einen zum Empfangen der Daten.



Für den Service wird eine 128-Bit UUID benötigt. Diese kann auf *«uuidonline»* (23) generiert werden. Die Definition sieht wie folgt aus.



Ausserdem werden zwei UUIDs für die TX und RX Charakteristik benötigt. Die Definitionen befinden sich in der *«DataService.h»* Datei.



Nun müssen die UUIDs deklariert werden, die Deklaration befindet sich in der *«DataService.cpp»* Datei.



Mit den folgenden Hilfsmakros werden der Service und die Charakteristiken im BLE-Host Stack[[14]](#footnote-14) registriert.



Als nächstes werden die Funktionen für das Senden und Empfangen von Daten geschrieben. Zuerst die für das Empfangen. Im Parameter *«buf»* befinden sich die Daten.



Nun folgt die Funktion für das Senden von Daten. Es kann ein 8-Bit Array gesendet werden. Ausserdem wird kontrolliert, ob die Benachrichtigungen beim Klienten eingeschaltet sind.



In der Struktur *«params»* wird als letztes eine Funktion angegeben. Diese wird ausgeführt, sobald die Daten versandt sind. Sie sieht wie folgt aus.



Als letztes gibt es noch die Funktion *«on\_cccd\_changed»*. Diese wird aufgerufen, wenn das CCCD[[15]](#footnote-15) Register vom Klienten (Hier Android Applikation) geändert wurde.



Nun ist der RX-TX Service auf der Server Seite fertig implementiert.

## Verwendung des Data Service

Der Data Service wird für vier verschieden Datensätze verwendet.

1. Allgemeine Informationen, welche in der Applikation angezeigt werden sollen. Hierfür wird im zusendenden Array nur eine Stelle verwendet. Es wird eine Nachrichtencode geschickt, mit dem anschliessend in der Applikation weiter gearbeitet wird. Eine Liste mit allen Nachrichtencodes befindet sich im Abschnitt 9.5, *«Nachrichtenanzeige»*.
2. Um die Herzfrequenz zu übermitteln, werden zwei Stellen im zusendenden Array benötigt. Die erste Stelle gibt an, dass es sich bei den Daten um die Herzfrequenz handelt und die zweite Stelle hält den Wert der Herzfrequenz.
3. Die Geschwindigkeit und Trittfrequenz werden mit drei Stellen im zusendenden Array übermittelt. Die erste Stelle gibt an, um welchen Typ, Geschwindigkeit oder Trittfrequenz, es sich handelt. Bei der Geschwindigkeit werden an zweiter Stelle der 8 Bit Wert der Geschwindigkeit, welche vor vom Komma steht gespeichert. An dritter Stelle werden die 8 Bit nach dem Komma gespeichert.  
   Handelt es sich um die Trittfrequenz, werden an zweiter Stelle die 8 niederwertigsten Bits der Trittfrequenz gespeichert. An dritter Stelle befinden sich die 8 höchstwertigsten Bits.
4. Der Batteriestand wird auch mit drei Stellen im zusendenden Array übermittelt. Die erste Stelle gibt an, dass es sich um einen Batteriewert handelt. An zweiter Stelle befindet sich die Information, um welchen Sensor es sich handelt. An dritter Stelle befindet sich der Batteriestand in %.

## Konfigurationsmöglichkeiten

Der Benutzer hat verschiedene Möglichkeiten beim Verbindungsaufbau zu den Sensoren. Es ist möglich sich nur mit einem Sensor, zwei oder drei Sensoren zu verbinden. Dies müssen aber zwingend CSC oder Herzfrequenz Sensoren sein, da ansonsten der Service für den Datenaustausch von den Sensoren zum nRF5340 nicht gefunden wird.

Wird die Verbindung zwischen einem Sensor und dem nRF5340 unterbrochen, beginnt der nRF5340 direkt wieder mit dem Scan Vorgang und versucht erneut eine Verbindung zu diesem Sensor aufzubauen.  
Wird die Verbindung zwischen dem nRF5340 und der Applikation unterbrochen, beginnt der nRF5340 direkt wieder mit dem Werben und wird so in der Applikation wieder erkannt. Es kann wieder eine Verbindung aufgebaut werden.

# Android Applikation

Die Android Applikation wurde in Android Studio (6) in der Programmiersprache Java entwickelt. Als Vorlage wurde eine bereits existierende Open Source Applikation von Nordic verwendet. Die *«Android-nrf-Blinky»* Applikation. Sie ist auf Github (24) verfügbar. Es wurde diese gewählt, da hier der ganze Bluetooth Teil, mit Scan Vorgang und Verbindungsvorgang bereits zur Verfügung stand. Es wurden die unnötigen Dienste für die Leds und Taster entfernt, und ein neuer Service implementiert. Bei dieser Implementation wurden die bereits existierenden Services von Nordic adaptiert.

Die Applikation hat verschiedene Seiten, die sogenannten *«activitys»*. Es gibt deren sieben, sie befinden sich im Ordner *«res -> layout»*. Hier wird das Aussehen der verschiedenen Seiten der Applikation gestaltet. Beim Start der Applikation wird die *«activity\_splash\_screen»* (Abbildung 14) Seite aktiv. Dies wurde in der *«AndroidManifest.xml»* Datei festgelegt. Nach einer Sekunde wird die nächste Seite, *«activity\_scanner» (*Abbildung 13*)*, aktiv. Hier werden alle gefundenen Geräte aufgelistet. Oben rechts in der Ecke können zwei Filter eingestellt werden. Der erste ist der UUID Filter. Es werden nur Geräte mit der verwendeten UUID angezeigt, das wäre in diesem Fall das nRF5340 Board. Mit dem zweiten Filter werden nur Geräte, welche sich in der Nähe befinden angezeigt. Nun können das Board und die zu verbindenden Sensoren ausgewählt werden. Mit einem Klick auf den Taster *«Connect»* wird eine Verbindung zum nRF5340 aufgebaut. Dies ist über eine Ladeseite ersichtlich. Auf dem Bildschirm steht zuerst *«Connecting»* und anschliessend *«Initializing»*. Danach wird die *«activity\_csc» (*Abbildung 12*)* Seite aktiv. Hier sind die Geschwindigkeit, Trittfrequenz, die gefahrene Distanz und die Herzfrequenz ersichtlich. Bevor jedoch die Echtzeitdaten des Geschwindigkeit Sensors angezeigt werden können, muss ein Raddurchmesser eingegeben werden. Dieser kann bei Bedarf zurückgesetzt werden. Die Echtzeitdaten des Trittfrequenzsensors, sowie die des Herzfrequenzsensors werden direkt in der Applikation angezeigt.

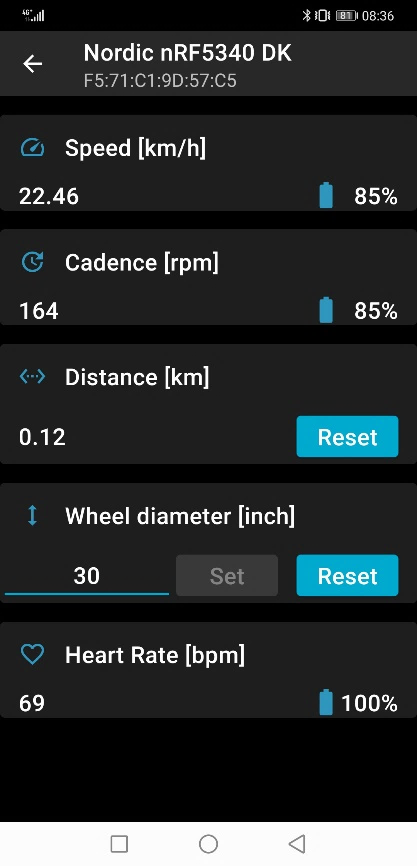


Abbildung : CSC Activity

Quelle: Eigene Darstellung

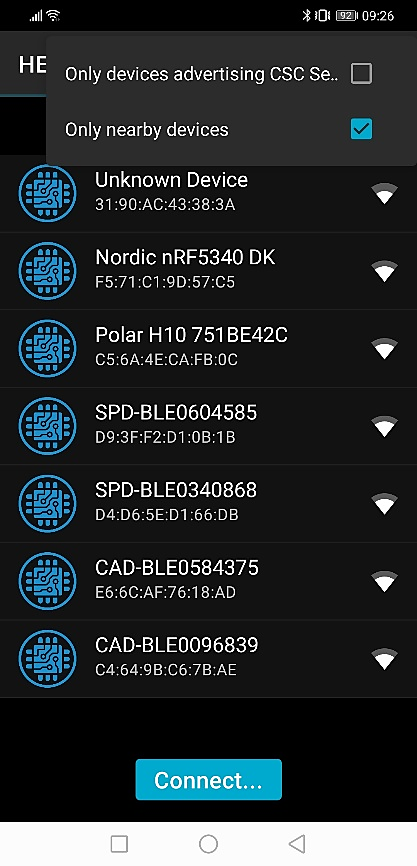


Abbildung 13: Scanner Activity

Quelle: Eigene Darstellung



Abbildung 14: Splash Screen Activity

Quelle: Eigene Darstellung

Die restlichen vier Activitys befinden sich im Anhang 4 *«Activitys Applikation»*.

## Voreinstellungen

Es gibt einige Einstellungen, welche zu Beginn vorgenommen werden müssen. Alle diese Einstellungen befinden sich in der *«AndroidManifest.xml»* Datei. In der folgenden Grafik ist diese Datei zu sehen. Am rechten Rand befinden sich Nummern, diese dienen zu Erklärungszwecken.  
Zunächst müssen einige Genehmigungen festgelegt werden. Die Applikation verwendet Bluetooth und ab dem Android Betriebssystem Marshmallow (Android 6.0) muss auch der Standort freigegeben werden. Dies wird über die Einstellungen im Punkt 1 erreicht. Im Punkt 2 wird das Logo der Applikation definiert. Mittels den Punkten 3, 4 und 5 werden die drei Hauptseiten (Activity) festgelegt. Ein weiteres Logo wird im Punkt 4 eingefügt. Dieses wird in der *«activity\_scanner»* Seiteverwendet. Im Punkt 6 wird die Datei festgelegt, in welcher der Service später implementiert wird.



## Scan – und Verbindungsvorgang

In diesem Kapitel wird auf den Scanvorgang mit dem anschliessenden Verbindungsaufbau genauer eingegangen. Es werden nur die notwendigen Schritte erklärt.  
Sobald die *«ScannerActivity»* aktiv ist, wird der Scan Vorgang gestartet. Im folgenden Sequenzdiagramm (Abbildung 15) ist der Ablauf gut ersichtlich.

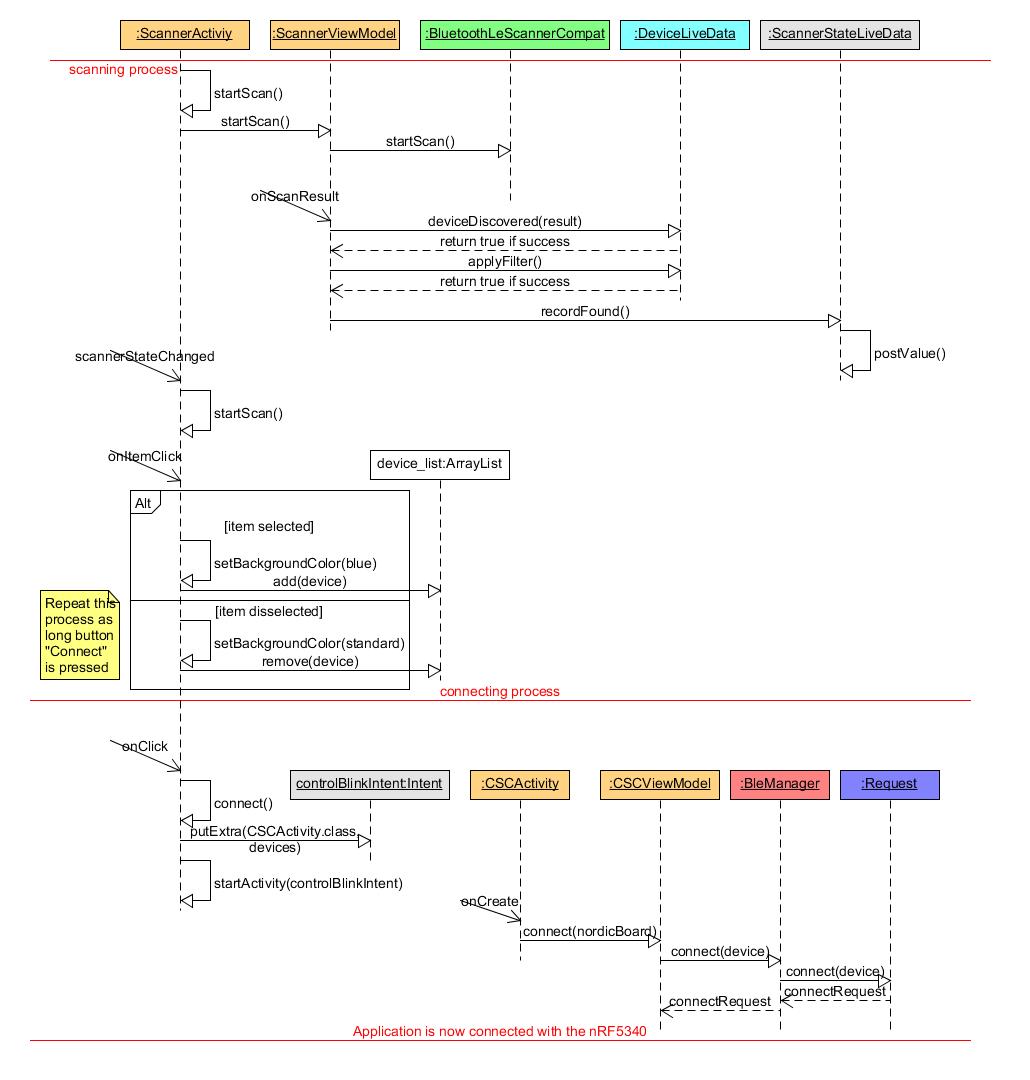


Abbildung : Sequenzdiagramm Scan- und Verbindungsvorgang

Quelle: Eigene Darstellung

## Data Service (Klient)

In der *«CSCManager»* Klasse wurden zunächst dieselben Charakteristik UUIDs definiert wie zuvor im Kapitel *«Data Service (Server)»*. Dies sieht wie folgt aus.



Im Unterordner *«Profil -> callback»* wurden die Klassen «*RXCallback»*, *«RXDataCallback»*, *«TXCallback»* und *«TXDataCallback»* hinzugefügt. Die Klassen «*RXCallback»* und *«TXCallback»* sind Interfaces, welche die Methode *«onCSCDataChanged»* zur Verfügung stellen. Diese wurden in der Klasse *«CSCManager»* implementiert. Werden neue Daten erhalten, wird zuerst die *«onDataReceived»* Methode der dementsprechenden Klasse aufgerufen. Hier werden anschliessend die Rückrufmethoden aufgerufen.

Im «*CSCManager*» wurden alle notwendigen Einstellungen hinzugefügt, sodass der Service einwandfrei funktionieren kann. Diese befinden sich in den Funktionen *«initialize»* und *«isRequiredServiceSupported»* und sehen wie folgt aus.



Hier wird eingestellt, dass eine Benachrichtigung erhalten wird, wenn sich ein Wert einer Charakteristik ändert.



Hier wird der Service mit seinen Charakteristiken kontrolliert, ob diese unterstützt werden.

Wenn nun Daten empfangen werden, müssen diese ausgelesen werden. Dies geschieht in der *«onCSCDataChanged»* Funktion des TX-Callbacks.



Die Weiterverarbeitung der Daten werden im nächsten Kapitel, Senden / Empfangen von Daten, behandelt. Ein Klassendiagramm mit den wichtigsten Teilen befindet sich im Anhang 3 *«Klassendiagramm Applikation»*.

## Senden / Empfangen von Daten

Es gibt drei Klassen die für das Senden, Empfangen und Anzeigen von Daten zuständig sind. Die Klasse *«CSCActivity»* ist für die Darstellung der Daten zuständig. Der *«CSCManager»* ist die Klasse, welche die Daten erhält, bzw. abschickt. Die *«CSCModelView»* Klasse dient als Zwischenklasse. Ein einfaches Szenario soll dies erklären. Sobald, dass alle Verbindungen aufgebaut sind, muss der Raddurchmesser angegeben werden. Sobald die Taste zum Bestätigen betätigt wurde, wird die Methode *«sendDiameter»* des *«CSCViewModels»* aufgerufen. Diese wiederum ruft die Methode *«sendDiameter»* des *«CSCManagers»* auf. Hier werden schlussendlich die Daten mit dem Aufruf *«writeCharacteristic»* gesendet. In diese Richtung funktioniert dies gut, also CSCActivity -> CSCViewModel -> CSCManager. Beim Empfangen von Daten kann dieser Vorgang nicht einfach andersrum angewendet werden. In der *«CSCActivity»* wird ein *«<LiveData> Double»* vom *«CSCManager»* observiert. Sobald neue Daten in dieser Variablen gespeichert werden, wird die *«CSCActivity»* benachrichtigt und kann so die neuen Daten in der Applikation anzeigen. Dieses Szenario wird im nachfolgenden Sequenzdiagramm (Abbildung 16) nochmals genauer aufgezeigt.

## Nachrichtenanzeige



Abbildung : Sequenzdiagramm Senden / Empfangen von Daten

Quelle: Eigene Darstellung

Während der Ausführung der Applikation werden immer wieder Kurzmitteilungen am unteren Rand angezeigt. Sogenannte *«Toasts»*. Die Information welche Nachricht angezeigt werden soll, wird vom nRF5340 zur Applikation mittels des Data Services gesendet. In der folgenden Tabelle sind alle Nachrichtencodes mit ihrer Bedeutung aufgelistet.

|  |  |
| --- | --- |
| **N°** | **Bedeutung** |
| 10 | Der Service wurde nicht gefunden |
| 11 | Verbindungsabbruch zu einem CSC Sensor |
| 12 | Verbindungsabbruch zum Herzfrequenzsensor |
| 13 | Verbindungsabbruch zu allen verbundenen Sensoren |
| 14 | Ein Geschwindigkeitssensor wurde ausgewählt und die Verbindung zu diesem wurde aufgebaut |
| 15 | Ein Trittfrequenzsensor wurde ausgewählt und die Verbindung zu diesem wurde aufgebaut |
| 16 | Ein Herzfrequenzsensor wurde ausgewählt und die Verbindung zu diesem wurde aufgebaut |
| 17 | Zum ersten Sensor (Geschwindigkeitssensor) von mehreren wurde eine Verbindung aufgebaut |
| 18 | Es wurden ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt und zum Ersten wurde eine Verbindung aufgebaut |
| 19 | Es wurden ein Trittfrequenz- und ein Geschwindigkeitssensor ausgewählt und beide Verbindungen wurden aufgebaut (Aufforderung zur Eingabe eines Raddurchmessers) |
| 20 | Es wurden ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt und beide Verbindungen wurden aufgebaut (Keine Aufforderung zur Eingabe eines Raddurchmessers) |
| 21 | Es wurden drei Sensoren ausgewählt und zum Zweiten (Trittfrequenzsensor) wurde eine Verbindung aufgebaut |
| 22 | Es wurden ein Geschwindigkeits- und ein Herzfrequenzsensor ausgewählt und beide Verbindungen wurden aufgebaut (Aufforderung zur Eingabe eines Raddurchmessers) |
| 23 | Es wurden drei Sensoren ausgewählt und alle Verbindungen wurden aufgebaut |
| 24 | Die Verbindung zu einem Herzfrequenzsensor wurde wieder neu aufgebaut |
| 25 | Der Durchmesser Wert muss eine Zahl sein |
| 26 | Der Durchmesser Wert ist zu gross |
| 27 | Der Durchmesser Wert ist zu klein |

Tabelle : Nachrichtencodes

## Referenzen Applikation

Applikationslogo:  
https://www.flaticon.com/download/icon/landing/92649?format=png&size=512

activity\_splash\_screen:  
HES-SO Logo:   
https://media-exp3.licdn.com/dms/image/C560BAQGeCvxg5z6RVQ/company-logo\_200\_200/0/1548143775894?e=2159024400&v=beta&t=dONOI4\_oqB9vbGZZoLo8tRLMrxLhPil7GcczDsFmSK0  
Valais Excellence Logo:   
https://www.valais-excellence.ch/media/image/0/label\_valaisexcellence\_cmjn.jpg

activity\_scanner:  
Item Schaltung:  
https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQonFqstVJNhFO\_Auor99-8YZ-fsB-1BLCGlg&usqp=CAU  
Fahrrad von der Seite:  
https://cdn5.vectorstock.com/i/1000x1000/97/59/bicycle-rider-icon-vector-21679759.jpg

activity\_csc:  
Symbol Geschwindigkeit, Trittfrequenz und Distanz:  
Android Studio Bibliothek

# Datenfluss

In diesem Kapitel wird der gesamte Datenfluss vom Start der Applikation bis zu den Daten von den Sensoren aufgezeigt. In der Abbildung 17 ist ein generelles Sequenzdiagramm zu sehen, welches den Datenfluss darstellt.

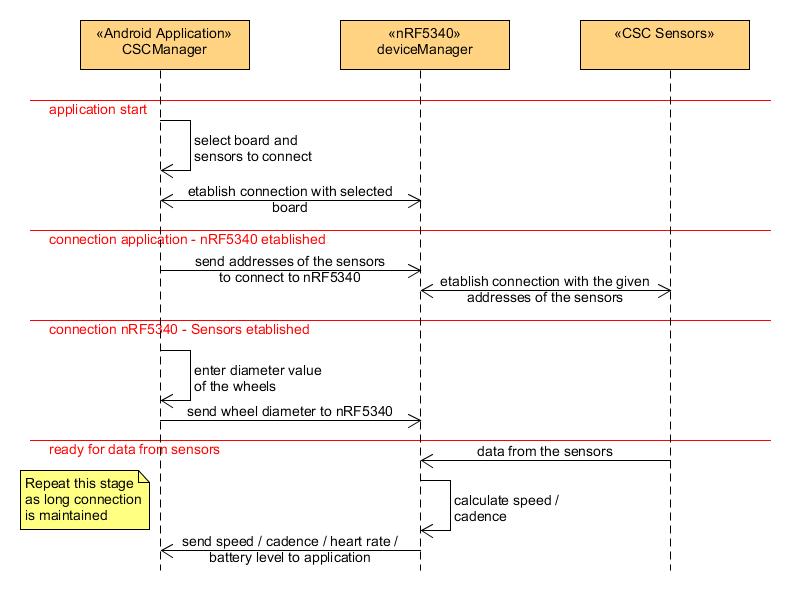


Abbildung : Sequenzdiagramm Datenfluss

Quelle: Eigene Darstellung

# Tests

Um die zentrale und periphere Verbindung zu testen wurde zu Beginn als peripheres Gerät ein Nordic Thingy:52 (16) verwendet. Da noch keine Applikation programmiert war, wurde die bereits bestehende Applikation von Nordic Semiconductors, *«nRF Connect for Mobile»* verwendet. Die Applikation ist kostenlos im Google Play Store (15) oder im App Store (25) erhältlich.  
Der Datenfluss wird mittels eines Tastendrucks auf dem Thingy:52 symbolisiert.   
In der Abbildung 18 ist die nRF Connect Applikation zu sehen, nachdem beide Verbindungen aufgebaut und die Taster Charakteristik eingeschrieben wurde. Wenn der Taster auf dem Thingy:52 betätigt wird, ist dies im roten Kasten durch einen Wechsel von *«Button released»* zu *«Button pressed»* ersichtlich.

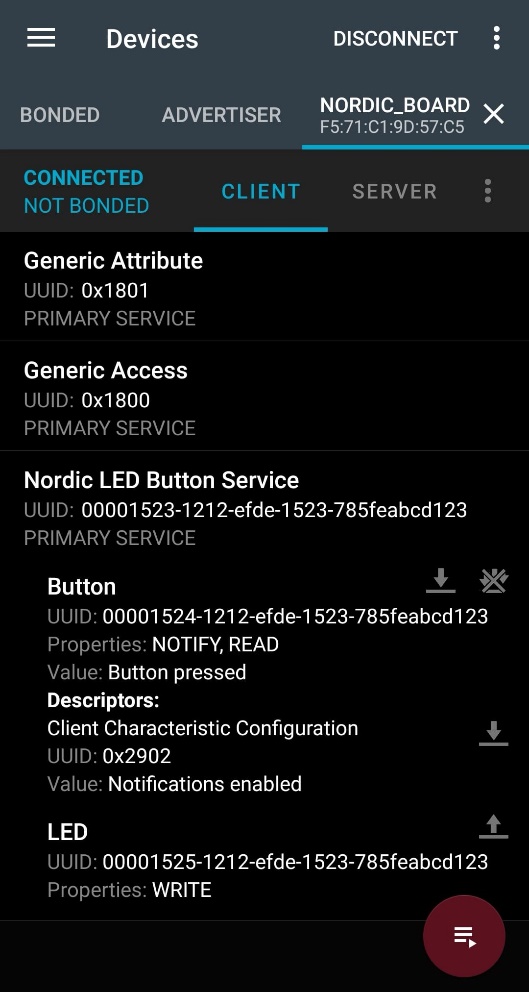


Abbildung 18: Test mit Thingy und Nordic Applikation

Eine komplette Liste mit allen Tests für die Applikation mit dem nRF5340 befindet sich im Anhang 5.

# Schlussfolgerung

Während dem Projekt wurde ein BT5-Demonstrator mit einem nRF5340 Entwicklungs-Kit und Zephyr entwickelt.

Dabei wurden folgende Ziele erreicht:

* Applikation *«HES-SO BIKE»* wurde erstellt
* Verbindungsaufbau: Applikation ⇨ nRF5340
* Verbindungsaufbau: nRF5340 ⇨ CSC Sensoren (max. 2)
* Verbindungsaufbau: nRF5340 ⇨ Herzfrequenz Sensor (max. 1)
* Verbindungsaufbau: nRF5340 ⇨ CSC Sensoren (max. 2) und einem Herzfrequenz Sensor
* Datenaustausch Applikation -> nRF5340
* Eingabe / Rücksetzung des Raddurchmessers in der Applikation
* Datenaustausch Sensoren -> nRF5340
* Berechnung und Weiterleitung der Daten an die Applikation
* Anzeige aller Echtzeitdaten in der Applikation
* Anzeige des Batteriestandes aller verbundenen Sensoren
* Anzeige von Kurznachrichten in der Applikation
* Verbindungsabbruch und Wiederherstellung der Verbindung zu Sensoren
* Verbindungsabbruch und Wiederherstellung der Verbindung zur Applikation

Aussicht

Die Gateway Funktion des nRF5340 mit Zephyr wird in Zukunft in weiteren Projekten wiederverwendet werden.

# Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.



# Anhang

[Anhang 1 - Klassendiagramm nRF5340 39](#_Toc79053690)

[Anhang 2 - Sequenzdiagramm nRF5340 40](#_Toc79053691)

[Anhang 3 - Klassendiagramm Applikation 41](#_Toc79053692)

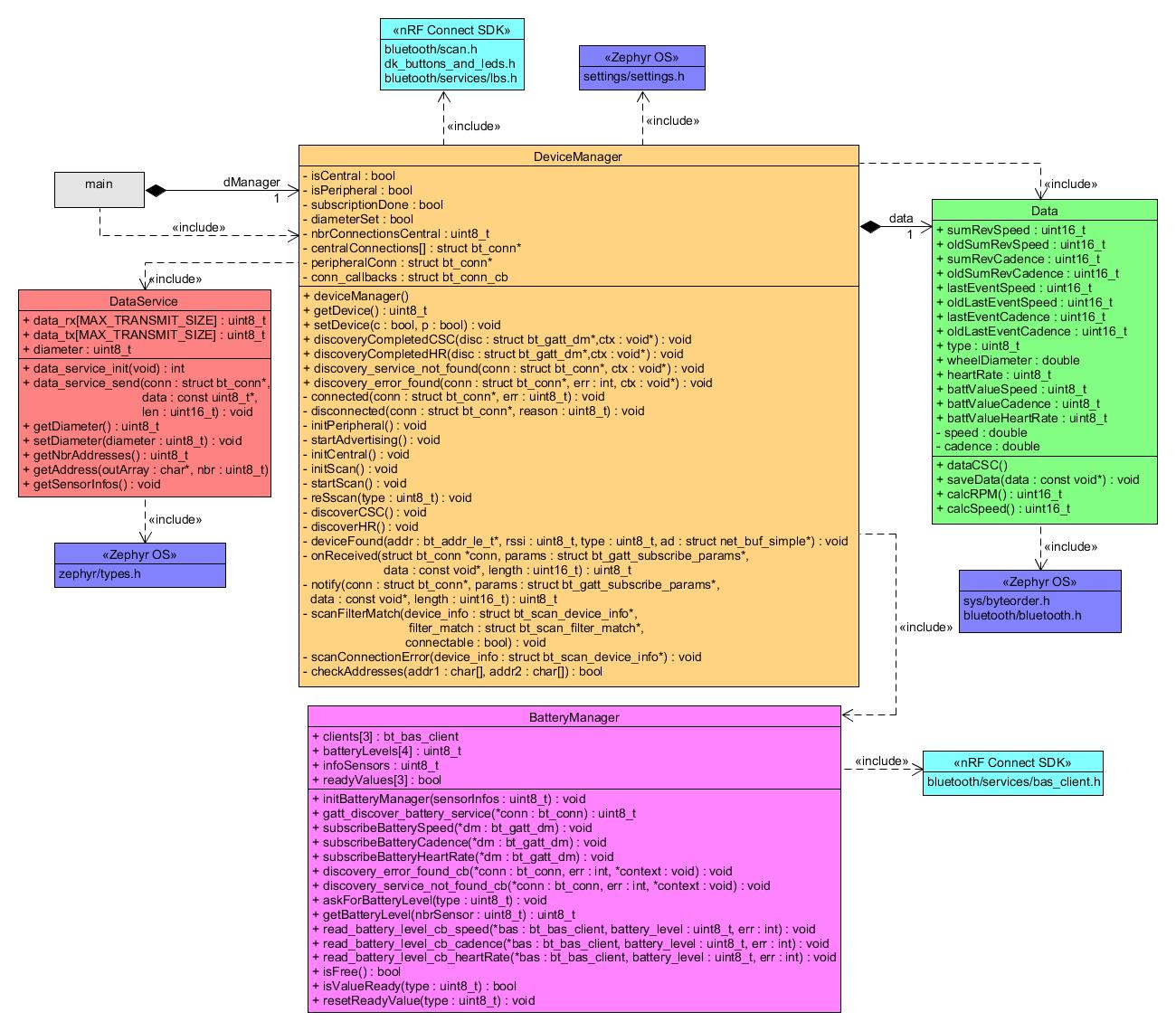
[Anhang 4 - Activitys Applikation 42](#_Toc79053693)

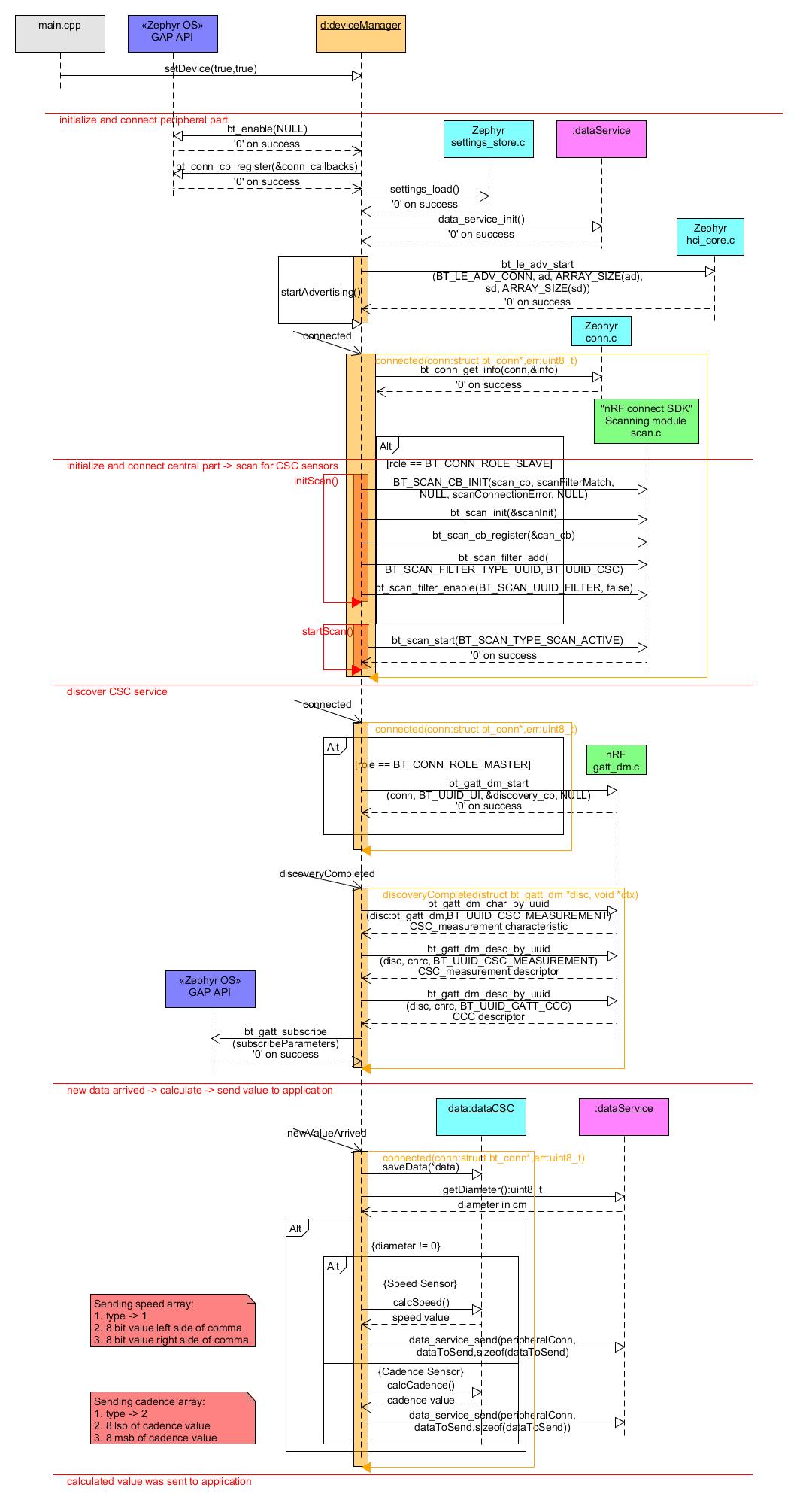
[Anhang 5 - Tests Applikation, nRF5340 43](#_Toc79053694)

Im Abgabeordner befinden sich folgende Unterordner:

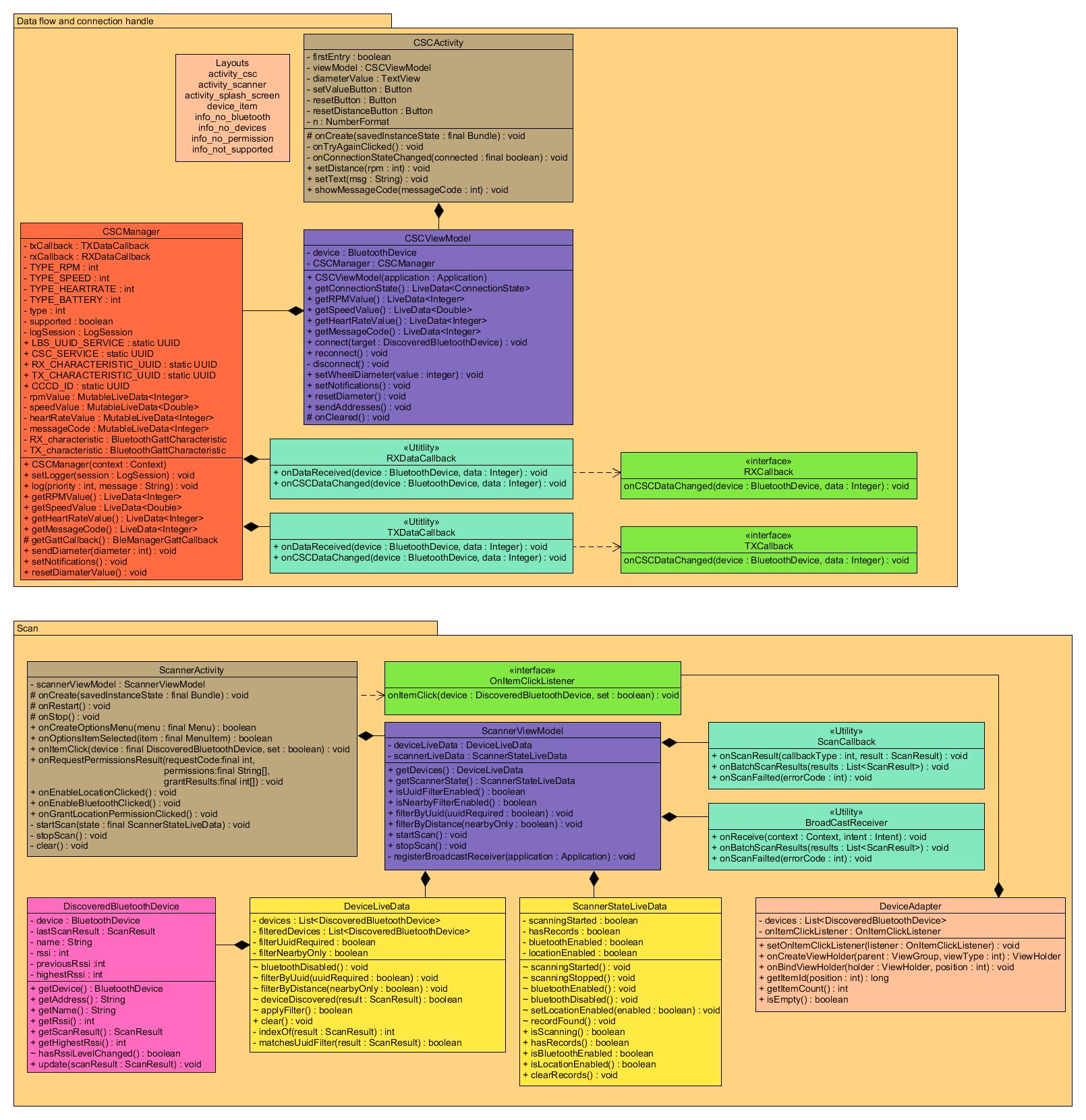
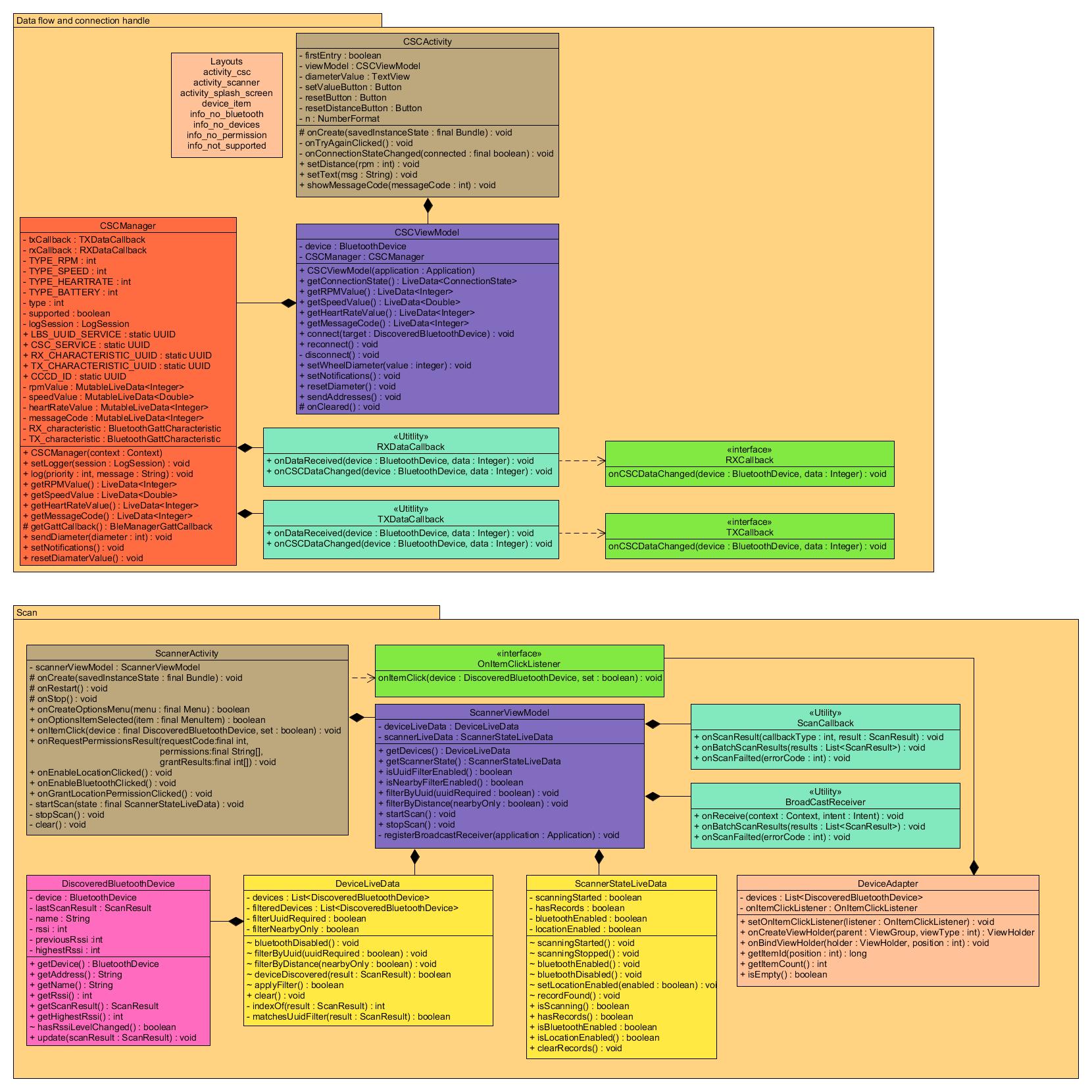
1. Datenblätter
2. Code Applikation
3. Code nRF5340
4. Bericht

Der gesamte Code und der Bericht ist ausserdem auf dem Github Ablage (https://github.com/bastianschwery/bachelor-thesis) des Verfassers auffindbar.

Anhang 1 - Klassendiagramm nRF5340

Anhang 2 - Sequenzdiagramm nRF5340

Anhang 3 - Klassendiagramm Applikation

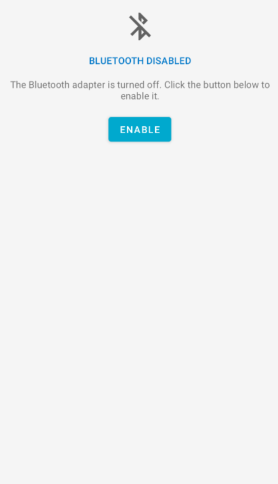


Anhang 4 - Activitys Applikation

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung

Anhang 5 - Tests Applikation, nRF5340

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test N°** | **Beschreibung des Tests** | **Was geschieht?** | **OK/NOT OK** | **Beschreibung Fehler** |
| 1 | Applikation wird gestartet | Die erste Seite wird angezeigt. Zu sehen ist das HES-SO Logo und das Valais Excellence Logo. | OK |  |
| 2a | Start des Scan Vorganges | Nach einer Sekunde wird die nächste Seite, die Scan Seite angezeigt. Es werden alle erkannten Geräte angezeigt. | OK |  |
| 2b | Bluetooth ist ausgeschaltet | Eine neue Seite öffnet sich mit der Aufforderung Bluetooth einzuschalten. Durch Betätigung der Taste *«Enable»* wird dies erreicht. | OK |  |
| 2c | Standort ist ausgeschaltet | Eine neue Seite öffnet sich mit der Aufforderung den Standortzugriff zuzulassen. Durch Betätigung der Taste *«Grant Permission»* wird dies erreicht.  Durch Betätigung der Taste *«Settings»* werden die Standorteinstellung des Smartphones aufgerufen. | OK |  |
| 2d | Filter *«Only devices advertising CSC Service UUID»* wird aktiviert | Es werden nur noch die Geräte angezeigt, welche eine CSC Service UUID besitzen. | OK |  |
| 2e | Filter *«Only nearby devices»* wird aktiviert | Es werden nur noch die Geräte angezeigt, welche sich in der Nähe des Smartphones befinden. | OK |  |
| 2f | Mit dem Filter *«advertising CSC Service UUID»* werden keine Geräte gefunden. | Es wird eine Information angezeigt, ob das korrekte Programm auf das Board geladen wurde. | OK |  |
| 3 | Es werden einige erkannte Geräte mittels eines Klicks ausgewählt. | Die Farbe der gesamten Zeile des ausgewählten Gerätes ist hellblau. Durch erneute Betätigung des Feldes, erscheint das Feld in der vorderen Farbe. | OK |  |
| 4a | Betätigung der Taste *«Connect»* ohne eine Auswahl von Geräten | Es erscheint eine Mitteilung, dass mindestens ein Gerät ausgewählt werden muss. | OK |  |
| 4b | Betätigung der Taste *«Connect»* ohne die Auswahl eines Nordic Boards | Es erscheint eine Mitteilung, dass eine Nordic Board ausgewählt werden muss. | OK |  |
| 4c | Betätigung der Taste *«Connect»* mit einer Auswahl von 5 oder mehr Geräten | Es erscheint eine Mitteilung, dass maximal vier Geräte ausgewählt werden können.  (Nordic Board, Geschwindigkeit Sensor, Trittfrequenzsensor, Herzfrequenz Sensor) | OK |  |
| 4d | Betätigung der Taste *«Connect»* mit einer Auswahl von Geräten die kleiner als 5 ist und ein Nordic Board sich darunter befindet | Es öffnet sich eine neue Seite. Es ist ein Balken mit der Schrift *«Connecting…»* zu sehen. Es wird eine Verbindung zum Nordic Board aufgebaut. Anschliessend ist die Schrift *«Initializing»* ersichtlich. | OK |  |
| 5a | Der Initialisierungsvorgang war nicht erfolgreich | Der Service für den Datenaustausch wurde nicht gefunden. Es wird ein passender Text mit Informationen angezeigt. | OK |  |
| 5b | Der Initialisierungsvorgang war erfolgreich | Es öffnet sich eine neue Seite auf der Geschwindigkeit, Trittfrequenz und die gefahrene Strecke ersichtlich ist. Zu Beginn sind alle Werte auf 0.  Es gibt ein Eingabefeld, hier muss der Durchmesser eingegeben werden. | OK |  |
| 6 | Warten auf Verbindungen zu den Sensoren. | Es erscheinen Nachrichten, sobald die Verbindungen aufgebaut wurden. Anschliessend folgt eine Nachricht mit der Aufforderung zur Eingabe des Durchmessers der Räder. | OK |  |
| 7a | Es wird kein Durchmesser der Räder im Eingabefeld eingegeben. | Die aktuelle Seite bleibt, wie sie ist. Es werden keine neuen Daten empfangen, welche angezeigt werden können. | OK |  |
| 7b | Es wird ein Durchmesser der kleiner als 1 Inch ist eingegeben. | Es erscheint eine Mitteilung, dass der Wert grösser als 1 Inch sein muss. | OK |  |
| 7c | Es wird ein Durchmesser der grösser als 255 Inch ist eingegeben. | Es erscheint eine Mitteilung, dass der Wert kleiner als 255 Inch sein muss. | OK |  |
| 7d | Es wird ein Durchmesser zwischen 1 und 255 Inch im Eingabefeld eingegeben. | Nach der Betätigung der Taste *«Set»* werden direkt die aktuellen Daten aller verbundenen Sensoren angezeigt. Die Taste *«Set»* ist blockiert und kann nicht mehr betätigt werden. | OK |  |
| 7e | Es wird die Taste *«Reset»* beim «Wheel Diameter» betätigt. | Die Geschwindigkeit- und die Trittfrequenzwerte zeigen ‘0’ an. Der Wert für den Durchmesser wird auch auf ‘0’ gesetzt. Die Taste *«set»* ist wieder verfügbar. | OK |  |
| 8 | Es wird die Taste *«Reset»* beim Feld *«Distance»* betätigt. | Der Wert, welcher bei *«Distance»* angezeigt wird, wird auf ‘0’ zurückgesetzt. | OK |  |
| 9 | Auf der Seite, auf der die Daten zu sehen sind, wird die «Seite zurück» Taste oben links betätigt. | Es öffnet sich wieder die Scan Seite. Es werden alle erkannten Geräte angezeigt. Es erscheint eine Mitteilung, dass die Verbindung zum Nordic Board unterbrochen wurde. | OK | Die Felder die zuvor ausgewählt wurden sind noch hellblau. Aber es sind keine Geräte gespeichert. |
| 10 | Es wird erneut ein Nordic Board für den Verbindungsaufbau ausgewählt. | Die öffnet sich die Seite mit den *«Connecting…»* und *«Initializing»* Balken. Anschliessend öffnet sich das Fenster den Daten. Nach dem der Durchmesser der Räder eingegeben wurde, werden die Daten wieder angezeigt. | OK |  |
| 11 | Im Scan Fenster wird nur ein Geschwindigkeitssensor ausgewählt. | Es wird eine Verbindung zu diesem Sensor aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Nach dem der Durchmesser eingeben wurde, wird die aktuelle Geschwindigkeit und die gefahrene Distanz angezeigt. | OK |  |
| 12 | Im Scan Fenster wird nur ein Trittfrequenzsensor ausgewählt. | Es wird eine Verbindung zu diesem Sensor aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt, wird die aktuelle Trittfrequenz angezeigt. | OK |  |
| 13 | Im Scan Fenster wird nur ein Herzfrequenzsensor ausgewählt. | Es wird eine Verbindung zu diesem Sensor aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt. | OK |  |
| 14 | Im Scan Fenster wird ein Geschwindigkeits- und ein Trittfrequenzsensor ausgewählt. | Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt wird die aktuelle Trittfrequenz angezeigt. Nach der Eingabe des Raddurchmesser wird auch die aktuelle Geschwindigkeit angezeigt. | OK |  |
| 15 | Im Scan Fenster wird ein Geschwindigkeits- und ein Herzfrequenzsensor ausgewählt. | Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt. Nach der Eingabe des Raddurchmesser wird auch die aktuelle Geschwindigkeit angezeigt. | OK |  |
| 16 | Im Scan Fenster wird ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt. | Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt wird die aktuelle Trittfrequenz angezeigt. Nachdem der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt. | OK |  |
| 17 | Im Scan Fenster wird ein Geschwindigkeits-, ein Trittfrequenz- und ein Herzfrequenzsensor ausgewählt. | Es wird eine Verbindung zu den Sensoren aufgebaut. Im Falle eines Fehlers, wird der Verbindungsaufbau erneut versucht. Sobald der Nutzer losfährt wird die aktuelle Trittfrequenz angezeigt. Nachdem der Nutzer den Herzfrequenzsensor korrekt anzieht, wird die aktuelle Herzfrequenz angezeigt.  Nach der Eingabe des Raddurchmesser wird auch die aktuelle Geschwindigkeit angezeigt. | OK |  |
|  |  |  |  |  |

# Quellenverzeichnis

[Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK\_SPL.jpg 3 2](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315064)

[Quelle: https://www.mouser.ch/images/nordicsemiconductor/lrg/NRF5340-DK\_SPL.jpg 3 2](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315065)

[Quelle: https://play-lh.googleusercontent.com/WshzYj8MVD1cDz55UP3z0Hl3eqUiBIkez5GSYr0yxwGSbVTkdLILlb0m0SXkqX-tWC 3 2](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315066)

[Quelle: https://www.mouser.de/images/marketingid/2019/microsites/153282850/2020-12-21\_15-07-43.png 4 3](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315067)

[Quelle: https://static.garmincdn.com/en/products/010-12103-00/g/cadence\_v03-eb1598f4-1e01-4e01-8691-8ddf6e108677.jpg 5 3](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315068)

[Quelle: https://media.alltricks.com/hd/15743025eaaa35aa3e5d9.44386646.jpg 6 4](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315069)

[Quelle: https://miro.medium.com/max/1838/1\*7R\_hkwuk8v7gClsVXtoqPw.png 7 8](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315070)

[Quelle: https://learn.adafruit.com/assets/86833 8 9](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315071)

[Quelle: https://learn.adafruit.com/assets/86833 9 9](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315072)

[Quelle: https://www.mouser.ch/images/marketingid/2017/img/116326113\_Nordic\_Thiny52IoTSensorDevelopmentKit.png?v=052820.0204 10 13](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315073)

[Quelle: Eigene Darstellung 11 20](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315074)

[Quelle: Eigene Darstellung 12 21](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315075)

[Quelle: Eigene Darstellung 13 23](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315076)

[Quelle: Eigene Darstellung 14 29](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315077)

[Quelle: Eigene Darstellung 15 29](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315078)

[Quelle: Eigene Darstellung 16 29](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315079)

[Quelle: Eigene Darstellung 17 31](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315080)

[Quelle: Eigene Darstellung 18 34](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315081)

[Quelle: Eigene Darstellung 19 36](file:///C:\Users\basti\OneDrive\Dokumente\3_Jahr\Bachelorarbeit\Dokumente\Bericht.docx#_Toc79315082)

# Literaturverzeichnis

1. **Nordic Semiconductor. [Online] https://www.nordicsemi.com/.**

**2. Nordic Semiconductor. [Online] https://www.nordicsemi.com/Products/Development-hardware/nrf5340-dk/getstarted.**

**3. Garmin. [Online] https://buy.garmin.com/de-CH/CH/p/641221.**

**4. Polar. [Online] https://www.polar.com/ch-de.**

**5. Polar. [Online] https://www.polar.com/ch-de/about\_polar/who\_we\_are.**

**6. Android Studio. [Online] https://developer.android.com/studio?hl=de&gclid=CjwKCAjwiLGGBhAqEiwAgq3q\_o36qDS-GY-Np7uyRZC4-uOIHB8YfH2V\_JkCmjwhXgom77PBtf\_UPhoCiOoQAvD\_BwE&gclsrc=aw.ds.**

**7. Nordic Semiconductor. [Online] https://developer.nordicsemi.com/nRF\_Connect\_SDK/doc/latest/nrf/gs\_installing.html.**

**8. Github. [Online] https://github.com/zephyrproject-rtos/zephyr/tree/main/samples/basic/blinky.**

**9. Nordic Semiconductor. [Online] https://developer.nordicsemi.com/nRF\_Connect\_SDK/doc/latest/nrf/app\_boards.html.**

**10. Github. [Online] https://developer.nordicsemi.com/nRF\_Connect\_SDK/doc/latest/nrf/include/bluetooth/services/bas\_client.html.**

**11. Centare. [Online] https://www.centare.com/insights/what-is-bluetooth-low-energy.**

**12. Adafruit. [Online] https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap.**

**13. Adafruit. [Online] https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.**

**14. Github. [Online] https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap.**

**15. Google Play Store. [Online] https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=de\_CH&gl=US.**

**16. Nordic Semiconductor. [Online] https://www.nordicsemi.com/Software-and-tools/Prototyping-platforms/Nordic-Thingy-52.**

**17. Github. [Online] https://github.com/zephyrproject-rtos/zephyr/tree/main/samples/bluetooth/central.**

**18. Nordic Semiconductor. [Online] https://developer.nordicsemi.com/nRF\_Connect\_SDK/doc/latest/nrf/include/bluetooth/scan.html.**

**19. Github. [Online] https://github.com/nrfconnect/sdk-nrf/tree/master/samples/nrf9160/lte\_ble\_gateway.**

**20. Bluetooth. [Online] https://www.bluetooth.com/de/specifications/specs/cycling-speed-and-cadence-profile-1-0/.**

**21. Rossier, Yoan. *IoT bike gateway for Swiss Cycling.* Sion : s.n., 2020.**

**22. Nordic Semiconductor. [Online] https://devzone.nordicsemi.com/nordic/nrf-connect-sdk-guides/b/getting-started/posts/ncs-ble-tutorial-part-1-custom-service-in-peripheral-role.**

**23. UUID online. [Online] https://uuidonline.net/.**

**24. Github. [Online] https://github.com/NordicSemiconductor/Android-nRF-Blinky.**

**25. Apple Store. [Online] https://apps.apple.com/ch/app/nrf-connect-for-mobile/id1054362403.**

**26. Nordic Semiconductor. [Online] https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug\_gsg\_ses%2FUG%2Fgsg%2Fsoftdevices.html.**

**27. Github. [Online] https://github.com/nrfconnect/sdk-nrf/tree/master/samples/bluetooth/peripheral\_lbs.**

1. Norwegischer Halbleiterhersteller [↑](#footnote-ref-1)
2. Stromsparende Version von Bluetooth [↑](#footnote-ref-2)
3. Real Time Operating System (Echtzeit Betriebsystem) [↑](#footnote-ref-3)
4. Bluetooth 5 [↑](#footnote-ref-4)
5. Stellt eine Verbindung zwischen zwei unterschiedlichen Systemen her [↑](#footnote-ref-5)
6. Cycling Speed and Cadence (Geschwindigkeit und Trittfrequenz beim Radfahren) [↑](#footnote-ref-6)
7. Zentrum eines Rades [↑](#footnote-ref-7)
8. Quellcode in eine für den Computer verständliche Programmiersprache übersetzen [↑](#footnote-ref-8)
9. Methode um einfach und effizient Fehler im Code zu lokalisieren [↑](#footnote-ref-9)
10. Universally Unique Identifier (Universell eindeutiger Bezeichner) [↑](#footnote-ref-10)
11. Eigenschaft [↑](#footnote-ref-11)
12. Zusammenfassung mehrerer Objekte eines bestimmten Datentyps [↑](#footnote-ref-12)
13. Zeichenkette [↑](#footnote-ref-13)
14. Schicht im BLE Modell [↑](#footnote-ref-14)
15. Client Characteristic Configuration Descriptor (Client-Merkmal Konfigurationsdeskriptor) [↑](#footnote-ref-15)