Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

HENNA MÄKINEN

Testausympäristö Bluetooth Low Energy -pohjaisten laitteiden kehitykseen

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2023

TIIVISTELMÄ

Mäkinen, Henna: Testausympäristö Bluetooth Low Energy -pohjaisten laitteiden kehitykseen
Opinnäytetyö, AMK
Tietojenkäsittely
Toukokuu 2023
Sivumäärä: 37

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa testausympäristö Bluetooth Low Energy -pohjaisille laitteille. Työ perustuu ohjelmistokehitysprojektiin, joka on tuotettu Oras Oy:lle. Tarkoituksena oli saada heille korvaava testausympäristö vanhan tilalle, joka ei toimi halutulla tavalla ja ei täytä kaikkia toivottuja ominaisuuksia. Työhön haluttiin myös toimiva käyttöliittymä, jonka avulla pystytään hallitsemaan toiminnallisuuksia.

Työ toteutettiin käyttäen Python-ohjelmointikieltä ja kehitysympäristönä toimi Visual Studio Code. Käyttöliittymän luomiseen käytettiin Pythonin omaa Tkinter-moduulia. Lisäksi käytössä oli Nordic Semiconductorin Dongle sekä emolevy, jossa pyöri tarvittava sovellus testausympäristön testaamista varten.

Työssä perehdyttiin siihen, millainen testausympäristö saatiin aikaiseksi ja millaisia keinoja sen luomiseen on käytetty. Lisäksi opinnäytetyössä käytiin läpi, millaisia kehityskohteita ympäristölle oli ja millaisia haasteita eteen tuli.

Opinnäytetyön tuloksena syntyi testausympäristö käyttöliittymineen, joka hakee halutusti palvelut, ominaisuudet ja niiden tietosisällöt. Opinnäytetyö ei tällaisenaan täyttänyt kaikkia haluttuja ominaisuuksia, joten sen kehitys jatkuu.

Avainsanat: Bluetooth, tuotekehitys, ohjelmistokehitys, Python

Abstract

Mäkinen, Henna: Test environment for the development of Bluetooth Low Energy-based devices
Bachelor’s thesis
Business Information Systems
May 2023
Number of pages: 37

The purpose of this thesis was to design and implement a test environment for Bluetooth Low Energy based devices. The work is based on a software development project produced for Oras Oy. The purpose was to provide them with a replacement test environment to replace an old one that did not work as desired and did not fulfill all the desired features. They also wanted a functional user interface to manage the functionality.

The environment was implemented using the Python programming language and the development environment was Visual Studio Code. Python's own Tkinter module was used to create the user interface. In addition, a Nordic Semiconductor Dongle and a motherboard were used to run the required application for testing environment.

This thesis focused on what kind of testing environment was created and what methods were used to create it. In addition, the thesis reviered what kind of development targets there were for the environment and what kind of challenges there were.

As a result of the thesis, a testing environment was created with a user interface that retrieves the services, features, and their data content as desired. As it is, the thesis did not meet all the desired features, so its development continues.

Keywords: Bluetooth, product development, software development, Python

SISÄLLYS

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimii Oras Oy. Tavoitteena on saada testausympäristö Bluetooth Low Energy -pohjaisten laitteiden kehitykseen. Samankaltainen ympäristö on käytössä, mutta tämän opinnäytetyönä tehdystä ympäristöstä haluttaisiin saada sille korvaaja, joka palvelisi toiveita paremmin.

Opinnäytetyön tarkoituksena on siis saada toimiva testausympäristö, joka täyttäisi halutut vaatimukset. Testausympäristön tehtävänä on päästä testaamaan uusia laitteita sekä laitteiden uusia ominaisuuksia. Tavoitteena on pystyä yhdistämään haluttuun laitteeseen, joka erikseen halutaan suodattaa ja saada sen sisällä olevia tietoja käyttöliittymään näkyviin. Tällaisia tietoja ovat muun muassa palvelut, ominaisuudet ja mainosdata. Opinnäytetyössä käytetään ohjelmointikielenä Pythonia ja kehitysympäristönä toimii Visual Studio Coden.

Opinnäytetyön sisältö etenee siten, että alkuun käsittelen työhön kuuluvia keskeisiä aiheita. Näiden jälkeen kerron tarkemmin Bluetoothin yhteyden muodostamista. Teoriaosuuksien jälkeen käydään läpi testausympäristön halutut vaatimukset, niiden työvaiheet ja toteutustavat koodiesimerkkien avulla. Nostan esille myös haasteita, joita testausympäristön luomisessa tuli eteen. Lopuksi pohdin työn lopputulosta sekä kehitysideoita tulevaisuutta varten.

2 ORAS OY

Oras perustettiin Raumalle vuonna 1945 Erkki Paasikiven toimesta. Pieni paja pantiin aluille Paasikiven kellariin heti toisen maailmansodan jälkeen ja vuonna 1951 Oras alkoi valmistamaan vesihanoja. Kansainvälisyyden Oras saavutti 1980-luvulla. Myöhemmin vuonna 2004 perustettiin suomalainen perheyritys Oras Invest, joka omistaa Oraksen. Kun vuonna 2013 Oras osti saksalaisen kilpailijansa Hansan, muodostettiin Oras Group yhtiöryhmä, jonka alla ovat Hansa ja Oras. Nykypäivänä Oras Group on merkittävä eurooppalainen talotekniikan vesikalustetoimittaja. (Oras Invest, 2023.) Pääkonttori sijaitsee edelleen Raumalla, ja Suomen lisäksi tuotantolaitoksia sijaitsee Puolassa, Saksassa ja Tšekissä.

Liikevaihto vuonna 2021 oli Oras Groupilla 233,5 miljoonaa euroa ja Oras Investillä 4,2 miljardia euroa. Yhteensä Oras Group ja Oras Invest ovat työllistäneet vuonna 2021 noin 10 000 ihmistä useassa eri maassa. (STT Info, 2022.) Oras Invest omistaa Oraksen lisäksi 25,7 % Uponorista sekä 20,6 % Kemirasta. 2022 Oras Invest teki uusia investointeja Nelekseen ja Valmetiin. (Oras Invest, 2023.)

3 BLUETOOTH

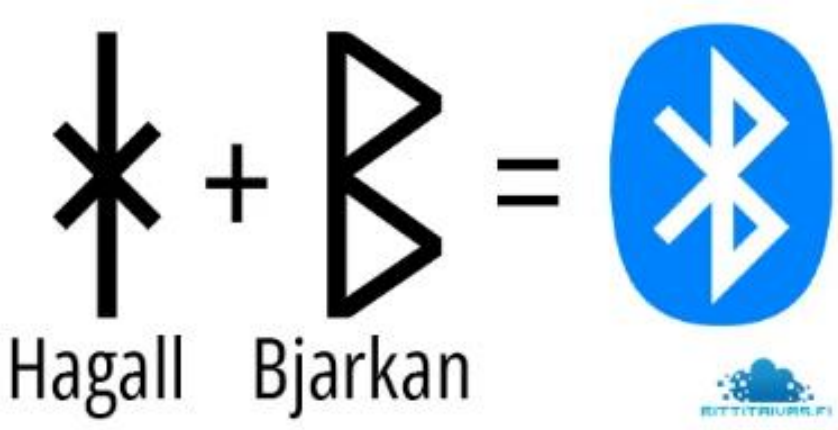
Bluetooth on nykypäivänä standardoitunut, ja se löytyy lähes jokaisesta laitteesta. Bluetooth on lyhyen kantaman langaton tiedonsiirtotekniikka, joka perustuu radiotekniikkaan. Bluetoothin tarkoituksena on ollut korvata kaapelit oheislaitteiden välillä.

Bluetooth sai alkunsa vuonna 1989, kun Ruotsin televiestintäyhtiö Ericssonilla töissä oleva Nils Rydbeck sai tehtäväkseen kehittää langattomat kuulokkeet. Toimiva prototyyppi saatiin kehitettyä vuoteen 1997 mennessä. Myöhemmin samana vuonna IBM halusi Ericssonin apua, jotta se voisi yhdistää puhelimen

toiminnallisuuden kannettavaan tietokoneeseen. Pian kuitenkin huomattiin, että virrankulutus on liian suuri, mikäli puhelin yhdistettäisiin kannettavaan tietokoneeseen. Uuden lyhyen kantaman radioteknologiaa haluttiin kuitenkin hyödyntää, jolloin yhteistyössä Intelin, Toshiban ja Nokian kanssa IBM ja Ericsson perustivat Bluetooth SIG:in (Special Interest Group). Vuonna 1999 ensimmäinen Bluetooth-laite julkaistiin. Se oli puhelimille tarkoitettu kuulokemikrofoni. Se voitti heti palkinnon parhaasta uudesta teknologian näytöstä. (Bittitaivas, 2022.)

Bluetooth-teknologia on lisensoitua sekä maksullista. Jotta tuotteessa saisi käyttää Bluetooth-logoa, täytyy läpikäydä Bluetooth SIG:n ylläpitämä sertifiointiprosessi. Bluetooth SIG ohjaa Bluetooth-standardien kehitystä ja yhteisössä on nykyisin mukana noin 36 000 yritystä. (Eurofins, 2023.)

Alun perin Bluetooth-nimen oli tarkoitus olla työnimi, joka ajateltiin korvata myöhemmin joko nimellä RadioWire tai Personal Area Network. Intelin työntekijä ehdotti Bluetooth-nimeä kuullessaan tarinan Harald Bluetooth:ista. Hän näki kuvan skandinaavisista riimuista Hagall ja Bjarkan. Koska Harald Bluetoothin nimikirjaimet ovat H ja B, päätti työntekijä yhdistää riimut, joiden avulla saatiin Bluetoothin logo. Nimeä ei enää myöhemmin muutettu, sillä Bluetooth oli nimenä jo hyvin vakiintunut. (Bittitaivas, 2022.)



Kuva 1: Bluetoothin logon muodostuminen. (Bittitaivas, 2022.)

Bluetoothin tarkoitus on käyttää mahdollisimman vähän virtaa ja samalla välttää ruuhkautumista muiden lähellä olevien laitteiden kanssa. Tämän takia Bluetoothin kantavuus ei ole niin suuri kuin esimerkiksi Wi-Fi:ssä. Matalan virrankulutuksen takia Bluetoothia voidaan käyttää pienissä laitteissa, joissa akkukapasiteetti voi olla hyvin rajallinen. (Bittitaivas, 2022.) Bluetooth luo yhteyden käyttämällä henkilökohtaista verkkoa (Personal Area Network). Henkilökohtainen verkko vaatii vähän fyysisiä komponentteja, jonka takia sen valmistus on sekä edullista että helppoa, eikä sen integrointi laitteisiin ole vaikeaa. (Bittitaivas, 2023.)

3.1 Bluetooth Low Energy

Bluetooth Low Energy (tästä eteenpäin BLE) on erittäin pienitehoinen teknologia, joka kuluttaa vähemmän energiaa kuin klassinen Bluetooth. Energian kulutus klassisella Bluetoothilla on 1 W, kun taas BLE:llä se on 0,01–0,50 W. BLE julkaistiin, kun matalaenergiset tiedonsiirtotekniikat yleistyivät. Alussa se tuki ainoastaan kahden laitteen välistä kommunikaatioita, mutta nykyisin BLE tukee solmuverkkoja. (Kranz, 2022.) BLE toimii 2,4 GHz taajuusalueella. Se käyttää yhteensä 40 kanavaa, jotka ovat kahden megahertsin välein eroteltuna. Tiedonsiirtoon käytetään 37 kanavaa ja mainostamiseen kolmea kanavaa. Tällä tavoin vältetään ruuhkautuminen ja kuormittuminen. Klassinen Bluetooth käyttää sen sijaan 79 kanavaa, yhden megahertsin välein eroteltuna. (Bluetooth, 2023)

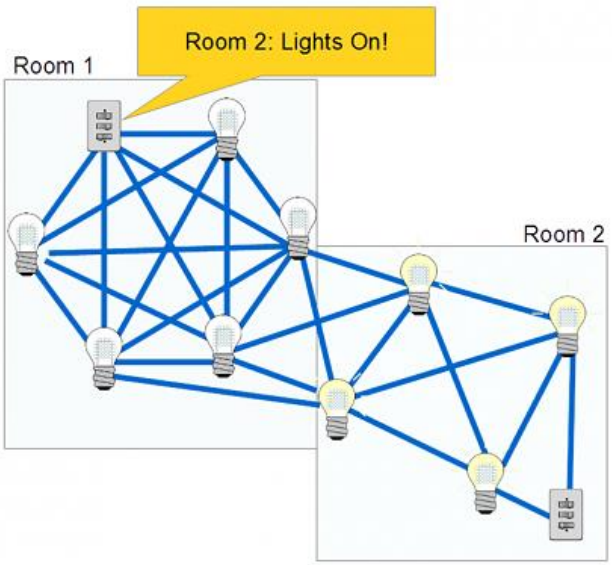
BLE:n etuja ovat siis muun muassa matala virrankulutus, pienemmät moduulien ja piirisarjojen kustannukset. BLE:ssä on kuitenkin matala tiedonsiirtokapasiteetti, sillä tiedonsiirtonopeus on 125 kbit/s – 2 mbit/s. Klassisen Bluetoothin tiedonsiirtonopeus on 1–3 mbit/s. BLE:n alue on alle 100 metriä, joten suurempiin aluetarpeisiin klassinen Bluetooth on parempi. (Krantz, 2022.)

3.2 Solmuverkko

Solmuverkko (Mesh Network) on Bluetooth SIG:in keksimä standardi, jonka avulla voidaan yhdistää kerralla iso joukko Bluetooth 4- ja Bluetooth 5 -standardien laitteita keskustelemaan keskenään. Tämä on tuonut uusia mahdollisuuksia esineiden internetin tietoliikenteelle. (Laakso, 2017.)

Solmuverkossa kaikki verkossa mukana olevat laitteet toimivat solmukohtina. Esimerkiksi WLAN- ja mobiiliverkoissa solmukohtina toimivat ainoastaan reitittimet ja tukiasemat. Nämä keskustelevat päälaitteiden kanssa, ja perinteinen verkko on tukiaseman ja reitittimen varassa. Solmuverkossa jokainen laite keskustelee keskenään. Data siirtyy lähettäjältä tukiasemien sijaan kaikkien tarpeeksi lähellä olevien laitteiden kautta. Yhden solmukohdan toimimattomuus ei johda siihen, että kaikki tiedonsiirto sulkeutuisi. Verkon kattavuutta voidaan myös kasvattaa lisäämällä laitteita. Data liikkuu salattuna, ja standardissa on määritelty Time to Live (TTL), joka tarkoittaa, että lähetetty viesti elää vain tietyn ajan verkossa. Näin saadaan estettyä viestin sinkoileminen laitteiden välillä, joka tukkisi kaistaa. (Laakso, 2017.)

Tässä opinnäytetyössä ei käytä solmuverkkoa, sillä tavoitteena on alkuun vain yksi yhteys. Solmuverkko on kuitenkin tärkeä osa Bluetoothia.

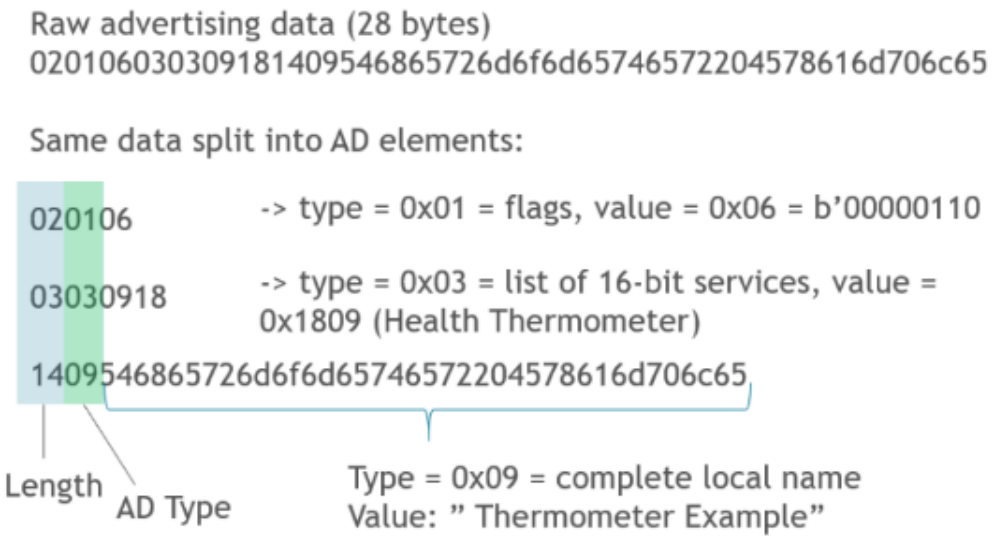


Kuva 2: Solmuverkon toimintaperiaate älyvalaisimissa. (Laakso, 2017.)

3.3 Bluetooth mainosviestit

BLE-laitteen mainostaessa, se lähettää ajoittain paketteja, jotka sisältävät tietoa esimerkiksi pääsyosoitteesta, mainonnan tietosisällöstä, CRC (Cyclic redundancy check) ja Bluetooth lähettäjän osoitteesta. Sovelluskehittäjiä yleensä kiinnostaa mainonnan tietosisältö, joka on 0–31 tavua pitkä. Mainosdata koostuu yhdestä tai useammasta elementistä. Jokainen elementti muodostuu seuraavasti:

- Ensimmäinen tavu on elementin pituus.
- Toinen tavu kertoo AD-tyypin eli mitä tietoja elementtiin sisältyy.
- Näiden lisäksi on AD-data, joka on yksi tai useampi tavu. (Silicon Labs, 2023a.)



Kuva 3: Mainosdata selitettynä auki. (Silicon Labs, 2023a.)

Raa'an mainosdatan purkamisesta on kuvassa kolme esimerkki. Esimerkissä raaka tietosisältö on jaettu kolmeen elementtiin. Koska ensimmäinen tavu tarkoittaa pituutta, on helppoa löytää elementin raja. Ensimmäinen elementti on liput. Tietosisältö on vain yhden tavun pitkä, mikä tarkoittaa, että kahdeksan lippua voidaan asettaa. Esimerkkikuvassa on annettu kaksi lippua, jotka ovat bittipaikoilla 1 ja 2:

- bitti 1: LE General Discoverable Mode

- bitti 2: BR/EDR (Bluetooth Basic Rate/Enhanced Data Rate) ei tueta.

LE General Discoverable Mode tarkoittaa, että keskuslaite voi nähdä oheislaitteen niin kauan kuin se mainostaa. Tämä on hyvin yleinen löydettävissä oleva lippu. BR/EDT ei tueta tarkoittaa, että laite on BLE-laite, eikä klassista Bluetoothia tueta. (Qualcomm, 2023.)

Toinen elementti sisältää luettelon hyväksytyistä palveluista, joka on 16-tavuinen tunnus. Näitä 16-tavuisia tunnuksia kutsutaan nimellä UUID. UUID:llä (Universal Unique Identifier) tarkoitetaan tietojärjestelmien yhteydessä käytetty yksilöintijärjestelmä. Tässä tapauksessa on vain yksi palvelu, jota mainostetaan. Se on Health Thermometer UUID:llä 0x1809. Kolmas elementti sisältää laitteen nimen, joka tässä esimerkissä on ”Thermometer”. Tämä on se nimi, joka näkyy yhdistäessä laitteeseen. (Silicon Labs, 2023a.)

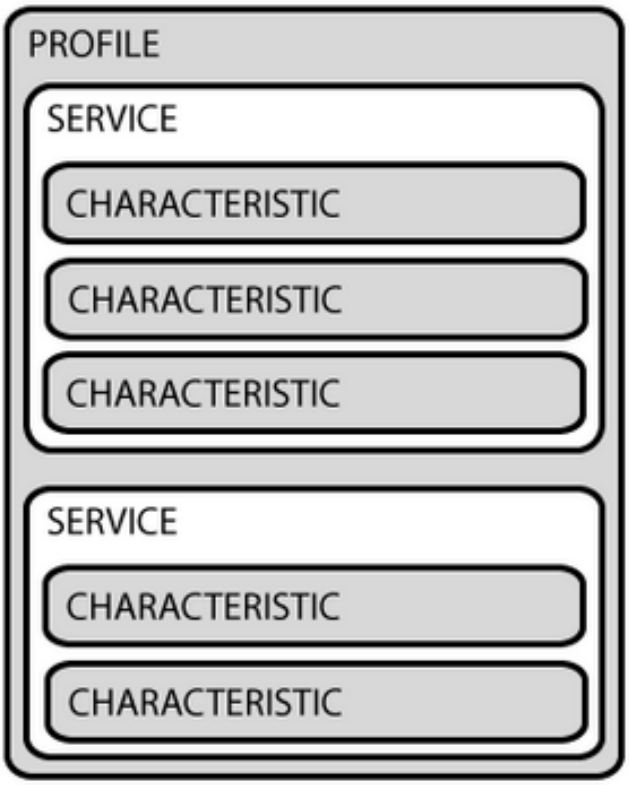
3.4 Yhdistys ja GATT-tapahtumat

Jotta voidaan ymmärtää paremmin testausympäristön toteutusta, on hyvä ymmärtää miten BLE:n yhdistäminen toimii tässä ympäristössä ja mitä kaikkea siihen kuuluu.

BLE-laitteiden skannaustapoja on olemassa aktiivisia ja passiivisia. Skannausta käytetään, jotta oikea laite yhdistämistä varten löytyisi. Tässä ympäristössä käytetään passiivista skannausta, joka tarkoittaa, että ympäristö ainoastaan vastaanottaa mainospaketteja. Aktiivisessa skannauksessa ympäristö lähettäisi skannauspyynnön, johon sensori vastaisi. (Silicon Labs, 2023b.) Tässä tapauksessa sensori muodostaa ensin mainosviestin, jonka jälkeen se aloittaa mainostamaan säännöllisesti. Testausympäristö puolestaan aloittaa mainosten skannauksen ja parseroi mainosten sisällön. Tästä mainosviestin sisällöstä saadaan muun muassa selville laitteen Bluetooth-osoite, jonka kautta yhteys pystytään muodostamaan. Mikäli kyseessä on haluttu mainostaja, mainosten skannaus pysäytetään ja yhteys avataan. Muussa tapauksessa mainosviesti hylätään. Kun tieto osoitteesta on saatu, voidaan avata

yhteys näiden kahden laitteen välille ja aloittaa tiedonsiirto. Yhteyttä varten testausympäristö lähettää pyynnön yhteyden muodostamiseen ja sensori vastaa tähän pyyntöön.

GATT (Generic ATTribute) käyttää yleistä dataprotokollaa nimeltä Attribute Protocol. Tätä käytetään palveluiden, ominaisuuksien ja niihin liittyvien tietojen tallentamiseen yksinkertaiseen hakutaulukkoon, jossa käytetiin 16-tavuisia tunnuksia kullekin taulukon merkinnälle. GATT-yhteydet ovat yksinoikeudellisia. Tämä tarkoittaa sitä, että heti kun oheislaite (sensori) muodostaa yhteyden keskuslaitteeseen (testausympäristö) se lakkaa mainostamasta itseään, eivätkä muut voi nähdä sitä tai yhdistää siihen. Yhteyden muodostaminen on ainoa tapa mahdollistaa kaksisuuntainen viestintä. BLE:n GATT-tapahtumat perustuvat korkeatasoisiin, sisäkkäisiin objekteihin. Näitä kutsutaan profiileiksi, palveluiksi ja ominaisuuksiksi. (Townsend, 2014.)

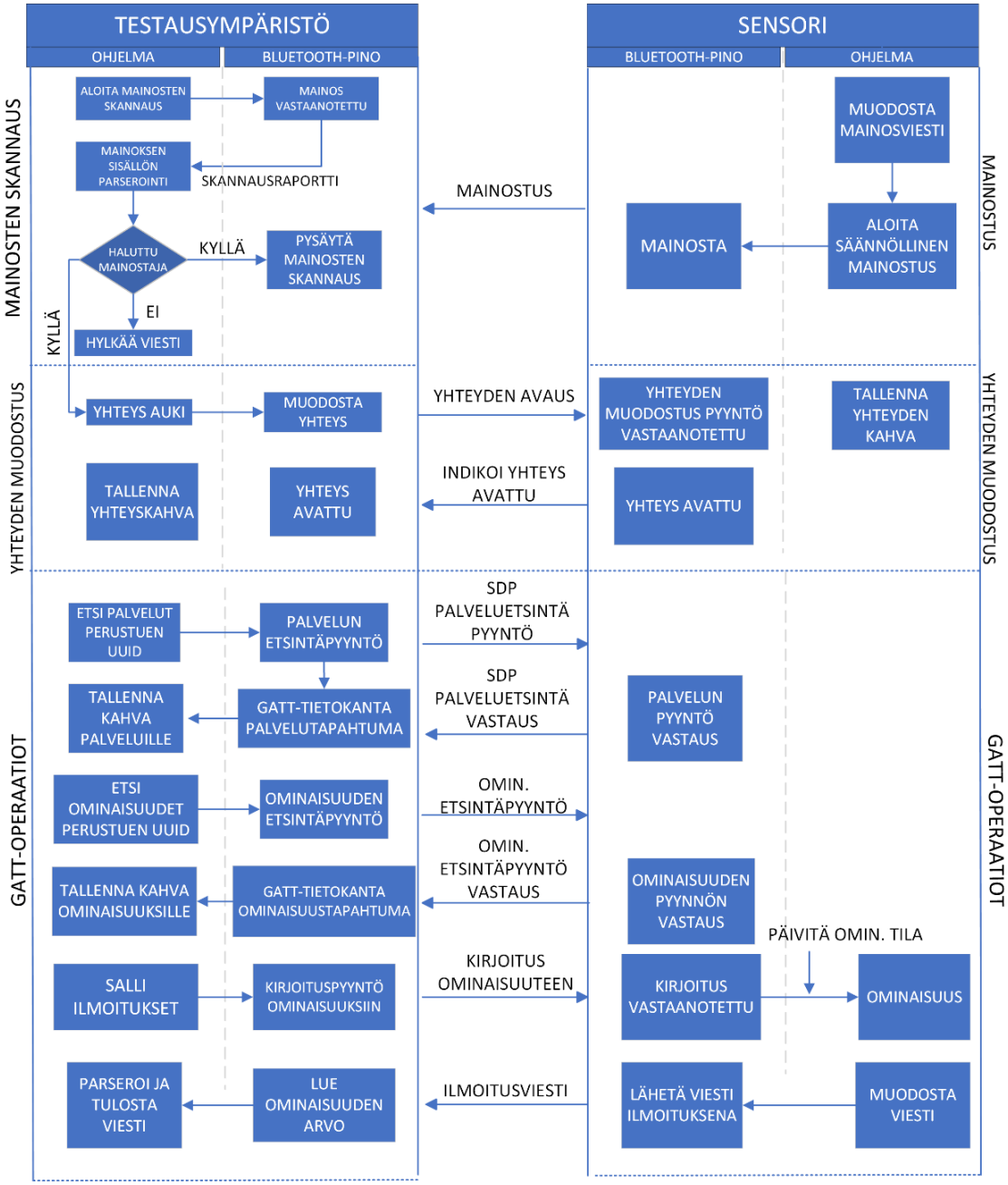


Kuva 3. GATT-tapahtuman objektit. (Townsend, 2014.)

Profiili ei itse ole olemassa BLE-oheislaitteessa. Se on joko Bluetooth SIG:n tai oheislaitteiden suunnittelijoiden laatima ennalta määritelty kokoelma palveluita. Palveluita käytetään tietojen jakamiseen loogisiin kokonaisuuksiin. Nämä kokonaisuudet sisältävät tietopaketteja, joita kutsutaan ominaisuuksiksi. Ominaisuuksia voi olla yksi tai useampi. Ominaisuuksilla on oma yksilöllinen numeroitu UUID-tunniste. (Townsend, 2014.)

GATT-tapahtumien alin käsite on siis ominaisuudet. Ominaisuuksilla on olemassa attribuutteja, joihin kuuluu lista oikeuksista. Oikeudet ovat nimeltään pääsy (access), jonka sisällä on lukeminen (read), kirjoittaminen (write), ilmoittaminen (notify) ja ilmaiseminen (indicate). Pääsyn lisäksi oikeuksiin kuuluu salaus (encryption) ja valtuuttaminen (authorization). (Microchip, n.d.) Näistä oikeuksista tässä opinnäytetyössä suuressa osassa on pääsyn sisällä oleva ilmoittaminen. Sen sisällä on niin sanottu data, joka halutaan saada tallennettua ja tarkasteltua. (Townsend, 2014.)

Sen jälkeen, kun yhteys on saatu auki, testausympäristö lähettää SDP (Service Discovery Protocol) palveluetsintäpyynnön etsiä palvelut tietystä UUID:stä. Tähän sensori vastaa. Testausympäristö lähettää myös ominaisuuksien etsintäpyynnön samanlaista kaavaa käyttäen. Sekä palvelut että ominaisuudet tallentuu. Ympäristö lähettää näiden vaiheiden jälkeen pyynnön sallia ilmoitukset halutusta UUID:stä ja kirjoittaa ominaisuuksiin. Sensori vastaanottaa kirjoituspyynnön, päivittää ominaisuuden tilaa, muodostaa viestin ja lähettää tämän viestin ilmoituksena. Ympäristö lukee ominaisuuden arvon, parseroi ja tulostaa viestin. Tämä viesti on data, jota halutaan lukea. Testausympäristöön on erikseen määritelty aika, kuinka kauan vastaanotetaan tätä dataa. Tämän ajan jälkeen yhteys katkeaa automaattisesti. Kuvassa neljä on vuokaavion avulla selkeytetty mainosten skannausta, yhdistystä sekä GATT-operaatiot.



Kuva 4: Bluetooth-yhdistämisen vuokaavio.

4 NORDIC SEMICONDUCTOR

Nordic Semiconductor (tästä eteenpäin Nordic) on vuonna 1983 perustettu norjalainen puolijohdealan pörssiyhtiö, joka on erikoistunut esineiden Internetin toimivaan langattomaan viestintätekniikkaan. Nordicin BLE-ratkaisut ovat

palkittuja ja se on maailmanlaajuinen markkinajohtaja BLE-alalla. Nordicin BLE-ratkaisuja käytetään tunnettujen tuotemerkkien laitteissa. Nordic tarjoaa laajan valikoiman langattomia ja moniprotokollaisia System-on-Chips (SoC) järjestelmiä. SoC on mobiililaitteissa ja sulatetuissa järjestelmissä käytetty mikropiiri. (Nordic Semiconductor, 2023.)

Yksi Nordicin tuotteista on nRF52840-Dongle, jota käytetään tässä opinnäytetyössä. Dongle on pieni tietokonelaitteisto, joka liitetään toisen laitteen porttiin. Tämän avulla saadaan lisätoimintoja, esimerkiksi kirjaston käyttö koodissa. NRF52840-Dongle kuuluu nRF52-sarjaan ja se koostuu seitsemästä moniprotokollaisesta Bluetooth 5.3 SoC:sta. Flash-muisti niissä vaihtelee 192–1024 kilotavuun ja keskusmuisti 24–256 kilotavuun. NRF52-sarjat ovat tarkoitettu todella alhaisen energian langattomiin ratkaisuihin, ja niiden SoC on täysin flash-pohjaisia. Flash-muistin avulla tuotteisiin saadaan joustavuus ja täydellinen päivitettävyys. (Nordic Semiconductor, 2023.)

The nRF52840 Dongle hardware consists of the board (PCA10059).

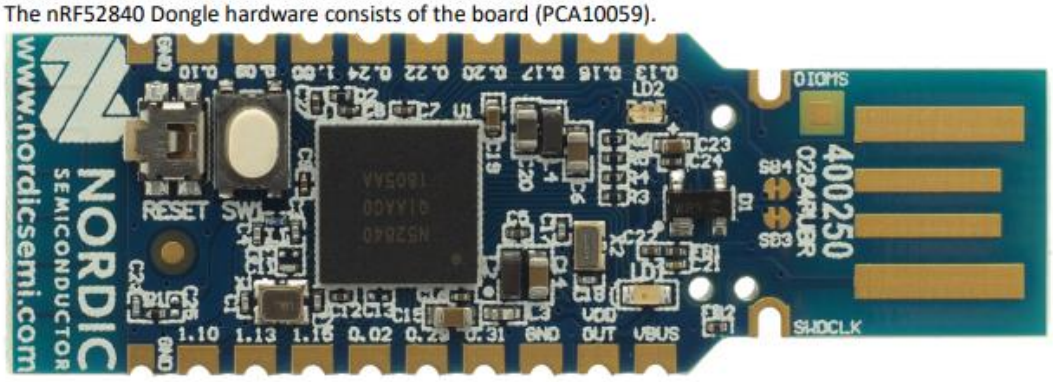


Figure 1: nRF52840 Dongle hardware (PCA10059) front

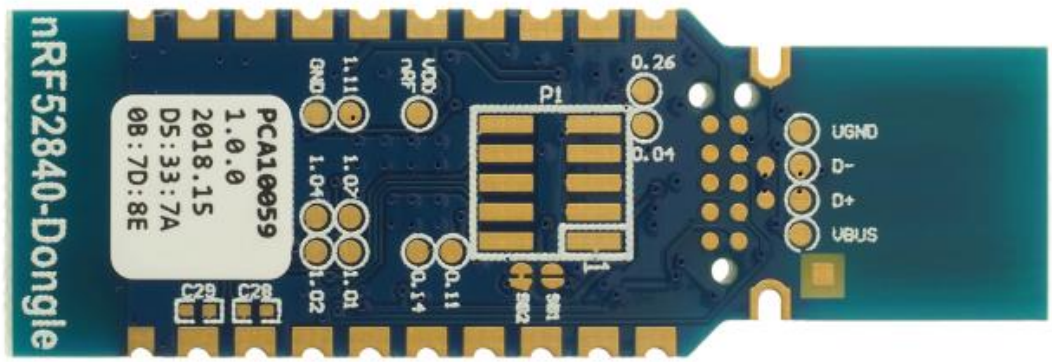


Figure 2: nRF52840 Dongle hardware (PCA10059) back

Kuva 6: nRF52840 Dongle. (Nordic Semiconductor, 2019, s. 6)

5 PYTHON

Python on suosittu avoimen lähdekoodin ohjelmistokieli, jolla toteutetaan tässä työssä esitettävä testausympäristö. Ensimmäinen versio, Python 0.9.0, julkaistiin vuonna 1991 (Ostrowska, 2022). Pythonin syntaksi on yksinkertainen, jonka takia sitä on helppo lukea sekä ymmärtää, sillä se mukailee tavallisia englannin kielen sanoja. Tästä syystä myös aloittelijoiden on yleensä helppo omaksua sitä. Pythonilla on valtava määrä erilaisia kirjastoja erilaisiin tarkoituksiin, jotka myös helpottavat sen käyttöä. Pythonin yksinkertaisen syntaksin hahmottamisen takia kuvassa seitsemän on Pythonin lisäksi verrattuna kuuden suositun ohjelmistokielen (C, C++, C#, PHP, Java, JavaScript) tapaa saada terminaaliin ”Hello World”. Tästä huomataan Pythonin yksinkertaisuus.

Kuva 7: Pythonin vertailu muihin ohjelmistokieliin.

Pythonin käyttötarkoitus on laaja. Yksi tunnetuimmista käyttötarkoituksista on erilaiset tekoälyn sekä koneoppimisen sovellukset. Tekoälyyn soveltuvien kirjastojen Pandas, Numpy ja Keras myötä Python on noussut yhdeksi data-analyytikkojen suosikkikieleksi. Muun muassa Netflixin suositusalgoritmeissa on

hyödynnetty Pythonia. Python taipuu myös backend-kehitykseen. Sille on luotu monia kirjastoja ja ohjelmistokehyksiä nimenomaan backend-kehitystä varten. Niin ikään tehtävien automatisoinnissa ja ohjelmien testauksessa Python on kätevä. Yksinkertaisen syntaksin avulla automatisoidut skriptit, esimerkiksi matemaattiset laskut sujuvat helposti. (upGrad, 2022.)

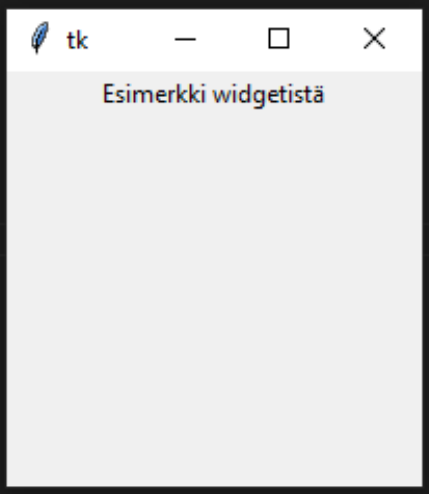
Vaikka Python on taipuva ja monipuolinen kieli, sillä on myös heikkoutensa. Se on hitaampi kuin esimerkiksi C, C++ tai Java. Sen tietorakenteet tarvitsevat myös enemmän muistitilaa, joten se ei sovellu projekteihin, jossa muistia on rajoitetusti. Se ei siis sovellu myöskään mobiili- tai pelikehittämiseen, vaan Pythonia käytetään pääasiassa työpöytä- ja web-palvelinpuolen kehitystöissä. (Singh, 2023.)

6 TKINTER

Tkinter on ilmainen graafinen työkalu Python-pohjaisten käyttöliittymien tekemiseen ja sitä käytetään tässä opinnäytetyössä. Se kuuluu Pythonin omiin moduuleihin ja usein tarvittavat kirjastot tulevat Pythonin asennuksen mukana. Se on Linux, Windows ja macOs tuettu. Käytöltään se on myös kevyttä ja helppoa. Kritiikkiä se saa kuitenkin vanhanaikaisen näköisestä käyttöliittymästä. Tämän muuttamiseksi on luotu tkinter.tkk-moduuli, joka mahdollistaa hieman modernimman käyttöliittymän. Tässä opinnäytetyössä on hyödynnetty tätä moduulia.

Peruselementti Tkinterin graafisessa käyttöliittymässä on ikkuna. Ikkuna on niin sanottu säilö, jossa kaikki käyttöliittymän elementit sijaitsevat. Näitä elementtejä kutsutaan widgeteiksi. Widgettejä ovat muun muassa tekstilaatikot ja painikkeet.

```python
#Tuodaan kirjasto käyttöön
import tkinter as tk

#Luodaan ikkuna
window = tk.Tk()

#Määritellään ikkunan koko
window.geometry("200x200")

#Luodaan yksi widget
greeting = tk.Label(text="Esimerkki widgetistä")

#Tällä komennolla lisätään widget ikkunaan
greeting.pack()

#Ajetaan Tkinterin event loop
window.mainloop()
```



Kuva 8: Esimerkki ikkunasta ja sen luomisesta.

7 TESTAUSYMPÄRISTÖN VAATIMUKSET

Työ halutaan ottaa tuotekehityskäyttöön, jonka takia sen täytyy täyttää tietyt vaatimukset korvatakseen nyt käytössä olevan ympäristön. Testausympäristöä käytetään, kun laitteisiin halutaan uusia ominaisuuksia tai käyttöön tulee kokonaan uusia laitteita. Ympäristön avulla laitteiseen pystytään ottamaan yhteys ja virhejäljittämään laitteita niiden kehitysvaiheessa. Näiden lisäsi ympäristöä voitaisiin käyttää uuden ominaisuuden tai laitteen testaamisessa.

Nykyisessä käytössä olevassa ympäristössä haasteita tuottaa oikean laitteen valitseminen. Laitteen valinta joudutaan tekemään ainoastaan tietämällä laitteen tyyppi. Tässä käytössä olevassa ympäristössä ei ole käytössä sellaista laitesuodatusta, jonka avulla pystyttäisiin helposti skannaamaan laitteet ja valita niistä oikea. Tämän työn ympäristössä halutaan näkyviin skannauksissa muun muassa laitteen sarjanumero. Sarjanumeron avulla saadaan heti valittua oikea laite, johon yhdistää. Käytössä oleva ympäristö on myös niin sanotusti kovakoodattu ominaisuuksiltaan. Tämä tarkoittaa sitä, että kun laitteisiin halutaan kehittää uusia ominaisuuksia, täytyy myös ympäristöön tehdä

muutoksia. Tärkeää on siis saada testausympäristö, jolla pystytään dynaamisesti lukemaan mitä tahansa dataa kehitettävästä ympäristöstä.

Työn yhtenä olennaisena vaatimuksena oli käyttöliittymän luominen, jotta saataisiin tiedot näkyviin siihen eikä kehitysympäristön terminaaliin. Käyttöliittymän visuaaliseen ilmeeseen sain täysin vapaat kädet. Tarkoituksena oli kuitenkin panostaa yksinkertaisuuteen ja keskittyä pääsääntöisesti toimivaan käyttöliittymään. Käyttöliittymässä olisi oltava näkyvillä sekä portti- että laitevalikko pudotusvalikkotyylillä. Tästä voitaisiin valita haluttu portti ja/tai laite. Laitevalikossa tulisi lukea laitteen nimi, osoite, mainoksen sisältö ja RSSI (Received Signal Strength Indicator). RSSI siis ilmaisee vastaanotetun signaalin voimakkuutta, jota ilmaistaan negatiivisena lukuna. Esimerkiksi RSSI ollessa -100, signaali on erittäin heikko. Mikäli arvo on alle -50, signaali on voimakas. Jos yhdistettävä laite on lähellä, RSSI avulla voidaan haluttu laite havaita helposti. Jotta laitelistaan ei tulisi näkyviin jokaista mahdollista lähellä olevaa BLE-laitetta, haluttiin laitesuodatus. Tärkeää olisi pystyä suodattamaan tulokset ainakin laitteen nimen ja mahdollisesti myös RSSI mukaan.

Tärkeänä ominaisuutena tulisi myös olemaan mahdollisuus yhdistää laitteeseen, jotta päästäisiin näkemään laitteessa olevia tietoja, esimerkiksi palvelut. Alkuun täytyy saada yhdistettyä yhteen laitteeseen, mutta mahdollisuuksien mukaan tulisi huomioida monitukiyhteys. Ennen kuin laitteeseen yhdistettäisiin, näkyviin haluttaisiin valitun laitteen mainostama mainosdata. Tämä helpottaisi myös hahmottamaan, mistä laitteesta on kysymys.

Kun haluttu laite on saatu yhdistettyä, päästään näkemään sen sisältämät palvelut. Nämä palvelut halutaan näyttöön näkyville ja näiden palveluiden ominaisuudet tulisi olla haettavissa ja luettavissa. Tarvittaessa nämä ominaisuudet voitaisiin tallentaa tiedostoon. Mainosdata on itsessään sellaisessa muodossa, että se saattaa näyttää vain satunnaisilta numeroilta ja kirjaimilta peräkkäin. Tätä dataa halutaan päästä purkamaan, jotta saadaan selkeästi näkyviin ne tiedot, jotka ovat tärkeitä ja halutaan tarkastella.

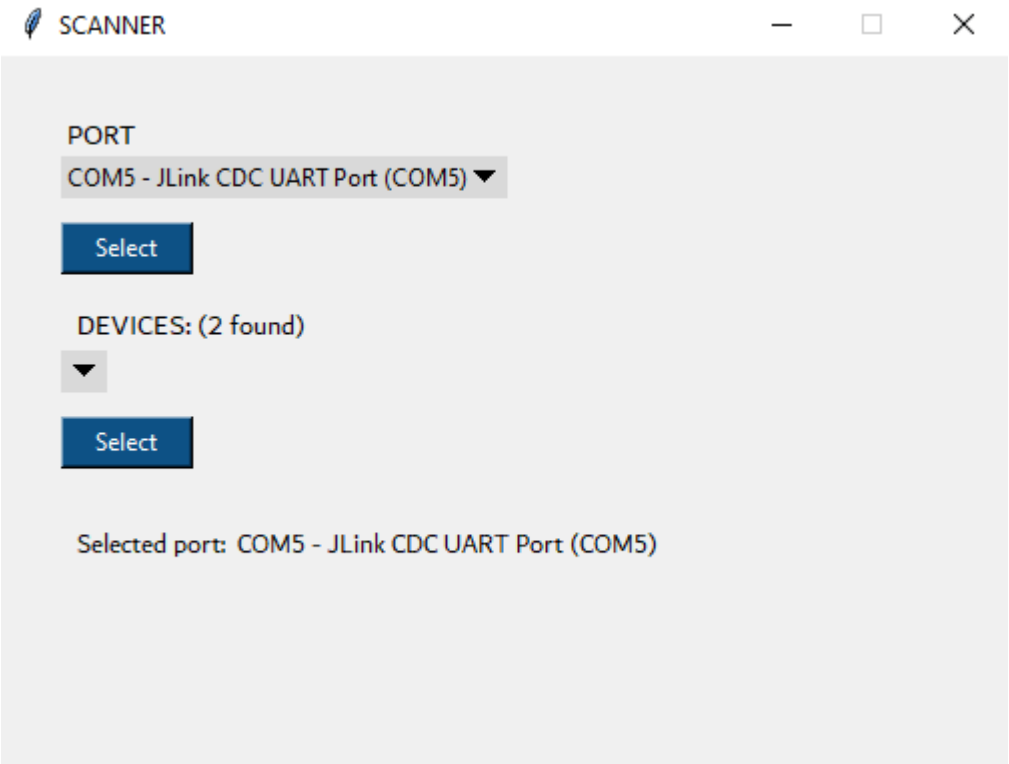
Käyttöliittymään halutaan saada toiminto laitteen yhteyden katkaisuun. Ilman monitukiyhteyttä, ainoastaan yksi laite voi olla yhdistettynä. Ilman mahdollisuutta yhteyden katkaisuun, laite voisi jäädä yhdistettyyn tilaan huomaamatta ja siihen uudelleen yhdistys aiheuttaisi virheen.

8 TYÖVAIHEET JA TOTEUTUS

Käyttöliittymä toteutettiin käyttäen Tkinteriä ja hyödyntäen ttk-moduulia, joka antaa laajemman mahdollisuuden muokata käyttöliittymää visuaalisesti. Käyttöliittymä sisältää yhteensä kolme erillistä ikkunaa. Tämä helpottaa ikkunoiden hallintaa sekä koodista saadaan selkeämpää, kun jokaisen ikkunan tieto on omassa kohdassaan. Yksittäisen ikkunan muokkaaminen ei myöskään tällä toteutuksella vaikuta muihin ikkunoihin.

Ikkunoiden tyylit on tehty yhteneväiseksi. Halusin valita neutraalit sävyt, joten ikkunoiden taustat ovat harmaat. Painikkeet ovat toimintojen mukaan joko siniset tai vihreät. Halusin käyttöliittymää suunnitellessa pitää kiinni niin sanotuista käyttöliittymän standardeista, jotta käyttäjäystävällisyys saataisiin toteutettua. Esimerkiksi vihreää väriä on käytetty painikkeeseen, jonka toimintona on yhdistää laitteeseen. Siniset painikkeet toimivat sen sijaan neutraalina painikkeina, joiden painaminen vahingossa ei aiheuta vahinkoa.

Ensimmäinen ikkuna sisältää portti- ja laitevalikot. Porttivalikon alla oleva Select-painike tallentaa portin omaan muuttujaansa sekä ilmoittaa valitun portin vielä alempana ikkunassa. Myös laitevalikosta valittu laite tallennetaan omaan muuttujaan, ja painettaessa laitevalikon alla olevaa Select painiketta päästään seuraavaan ikkunaan.



Kuva 9: Ensimmäisen ikkunan näkymä

Laitteiden hakemisessa käytettään Bleak:ia (Bluetooth Low Energy platform Agnostic Klient). Alun perin tarkoituksena oli käyttää Nordicin omaa blatann-kirjastoa, jonka takia työssä aluksi päätettiin käyttää nRFF52840-Donglea. Blatann-kirjaston avulla yhdistys Bluetooth-laitteeseen onnistuu myös. Bleak:in päädyin puoliksi vahingossa, sillä sain halutun yhdistystoiminnon toimimaan vaivattomasti sen kautta. En kokenut enää myöhemmin tarvetta vaihtaa käytettävää tapaa, mutta jatkokehityksen kannalta myös Blatann-kirjasto on edelleen hyvä pitää vaihtoehtona.

Bleak on GATT-asiakasohjelmisto, joka pystyy muodostamaan yhteyden GATT-palvelimena toimiviin BLE-laitteisiin. Sen avulla voidaan muun muassa kommunikoida antureiden kanssa. (Bleak Documentation, 2020.) BleakScanner.discover-luokan avulla saadaan esille laitteen nimi, osoite, RSSI, metadatan ja lisätiedot. Metadata sisältää sarakkeet UUID sekä manufacturer_data. Manufacturer_data sisältää osan mainosdatasta. Aluksi oli tarkoituksena saada koko mainonnan tietosisältö näkyviin, josta sitten olisi lähdetty

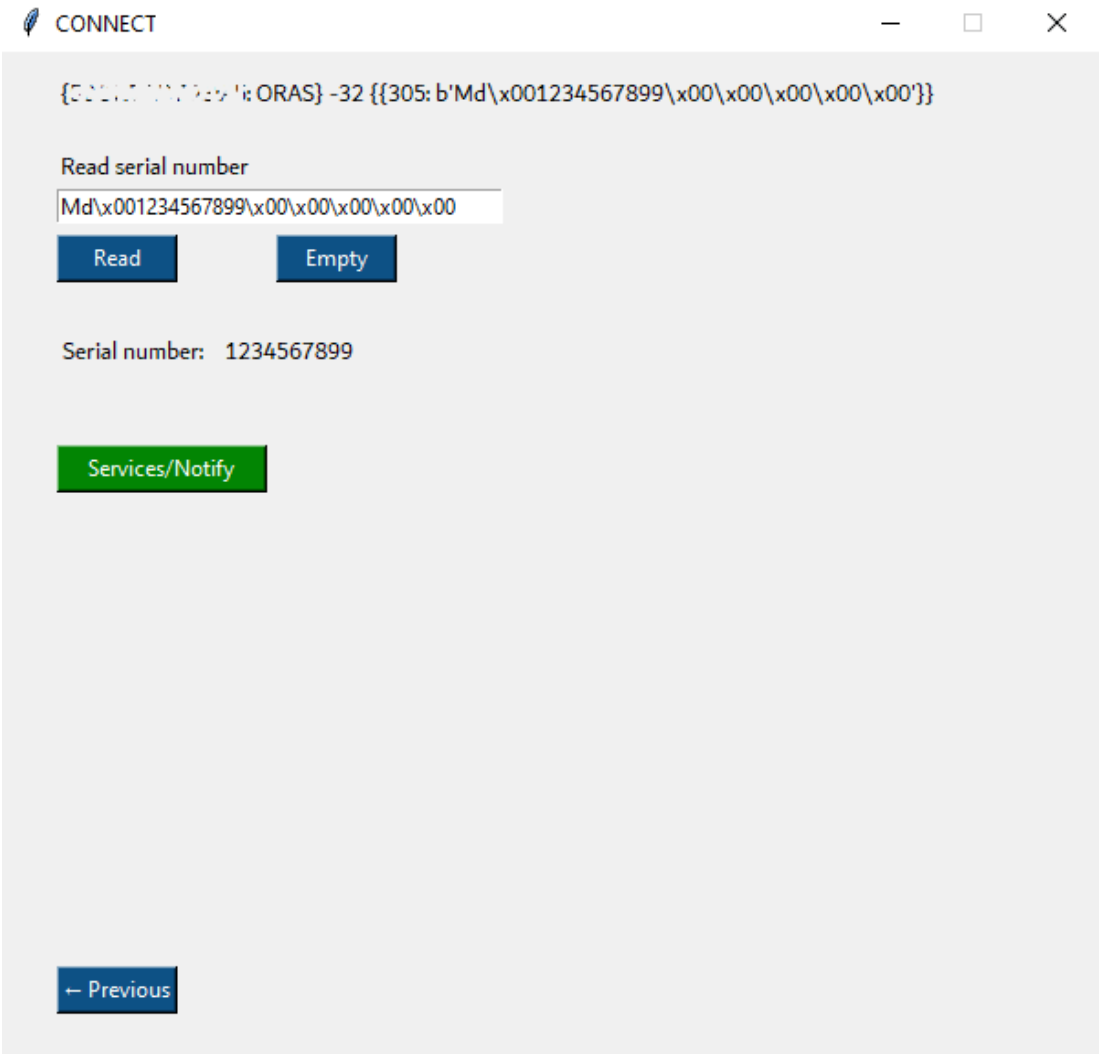
purkamaan sitä osiin. BleakScanner.discover-luokka kuitenkin purkaa jo valmiiksi mainosdataa, jonka takia mainonnan tietosisältö on huomattavasti lyhyempi.

Jotta halutut laitteet saataisiin näkyviin, on tätä varten luotu tyhjä lista oras_devices, johon lisätään laitteen osoite, rssi ja metadatasta löytyvä manufacturer_data. Kuvassa kymmenen näkyy, että laitteiden rajaamiseen on käytetty ainoastaan laitteen nimeä, jonka halutaan olevan ORAS. Tämä täytyy myöhemmin muuttaa myös toimivaan, jos laitteen nimi on kirjoitettu pienillä kirjaimilla. Mikäli halutaan rajata laitteita myös RSSI mukaan, lisätään määritelmä d.rssi < haluttu etäisyys.

```
41  #Scanning Devices
42  async def scan():
43      start = time.time()
44      time.sleep(1.5)
45      print(current_time, "Scanning devices...")
46      devices = await BleakScanner.discover(timeout=10)
47      for d in devices:
48              if d.name == 'ORAS':
49                  oras_devices.append([d] + [d.rssi] + [d.metadata['manufacturer_data']])
50              continue
51      end = time.time()
52      print(current_time, 'Scanning complete', f"\tTime taken: {end-start} s")
53
```

Kuva 10: Laitteiden haku

Toinen ikkuna sisältää ensimmäisessä ikkunassa valitun laitteen tiedot: osoite, nimi, RSSI sekä osan mainosdatasta, joka on saatu BleakScanner.discover manufacturer_data:sta. Tämä mainosdata on juuri se sisältö, joka vaatimuksissa haluttiin saada selville. Se sisältää sarjanumeron, joka saadaan toiminnolla parsittua selkeämpään muotoon. Ilman parsimista mainosdata näyttää: b'Md\x001234567899\x00\x00\x00\x00\x00'. Tässä tapauksessa mainosdatan osuus on selkeä ja ilman parsimista sarjanumero on pääteltävissä. Näin ei kuitenkaan aina ole.



Kuva 11: Toisen ikkunan näkymä

Toisen ikkunan Service/Notify-painike yhdistää valittuun laitteeseen. Tämän jälkeen aukeaa kolmas ikkuna, joka tulostaa näkyviin palvelut, ominaisuudet ja alkaa näyttämään dataa, mikäli koodiin määritelty service_uuid sitä sisältää.

Datan käsittelyä varten koodiin on luotu funktio nimeltä notification_handler. Tämän funktion tarkoitus on hallita tulevaa dataa ja tehdä tulevasta datasta selkeän muotoista. Ilman datan puhdistamista se näyttää: bytearray(b'\x14\x01Radio Entropy: 153'), kun taas puhdistettu data on huomattavasti selkämpi ja nopeammin luottevissa: Radio Entropy: 153.

```
33  #Notification handler for upcoming data
34  def notification_handler(handle: int, data: bytearray):
35      data = data.decode()
36      data = clean(data, no_emoji=True, lower=False)
37      label.config(text=data)
38
```

Kuva 12: Datan käsittelyn funktio.

Yhdistämiseen ja palveluiden sekä ilmoituksien hakuun on käytetty Bleak:in BleakClient:ia. BleakClient:ia voidaan käyttää asynkronisena kontekstinhallintaohjelmana, jolloin se muodostaa ja katkaisee yhteyden automaattisesti. Koodiin on lisätty muuttuja service_uuid, jossa tiedetään olevan näytettävää dataa (notify-toiminto) sekä muuttuja address, joka sisältää yhdistettävän laitteen osoitteen. Koodi yhdistää annettuun osoitteeseen ja hakee tämän jälkeen kyseisen laitteen palvelut sekä service_uuid sisällä olevan datan. Viimeiseen ikkunaan tulostuu siis lista palveluista ja valitun palvelun sisällä olevat ominaisuudet. Ominaisuuden sisältä myös haetaan ilmoitukset eli näkyviin ikkunaan tulee myös data, joka on ilmoituksen takana. Tämän ikkunan tietojen näyttäminen on jätetty pois, sillä se sisältää UUID-tunnisteita.

```
55  #Search service charasteristics and start notify
56  async def discover_service_characteristic_uuid(address, service_uuid):
57      async with BleakClient(address) as client:
58          label3.config(text=f"\nConnected to {bluetooth_address}")
59          for service in client.services:
60              await asyncio.sleep(1)
61              label3.config(text=label3.cget("text") + f"\n {service}")
62              if str(service.uuid) == service_uuid:
63                  characteristics = service.characteristics
64                  for char in characteristics:
65                      label4.config(text=label4.cget("text") + f"\n{service.uuid} characteristic uuid is {char.uuid}")
66                      await client.start_notify(char.uuid, notification_handler)
67                      continue
68          await asyncio.sleep(5)
69          await client.stop_notify(char.uuid)
70          await client.disconnect()
```

Kuva 13: Yhdistäminen (rivi 57), palveluiden ja ominaisuuksien näyttämisestä sekä datan näyttämisestä (rivit 66–70)

Koska dataa halutaan tallentaa tarkastelua varten, tehdään kansio, jonka sisälle luodaan tiedosto nimeltä ble_advertisement_data. Data tallentuu .CSV-tiedostoon. Mikäli tiedosto löytyy jo, lisää se aina samaan tiedostoon uuden datan vanhan alapuolelle. Excelissä on omat sarakkeet aikaleimalle, mainoksen tyypille ja sen arvolle.

```
#Creates directory and save the data
    if not os.path.exists(path):
        os.mkdir(path)
        print('Directory created!')

    data_parts = data.split(':')
    timestamp = datetime.datetime.now().strftime('%H:%M:%S')

    with open(os.path.join(path, 'ble_advertisements_data.csv'), 'a', newline='') as file:
        writer = csv.writer(file)
        if os.stat(os.path.join(path, 'ble_advertisements_data.csv')).st_size == 0:
            writer.writerow(['Timestamp', 'Advertisement Type', 'Value'])
        writer.writerow([timestamp, data_parts[0].strip(), data_parts[1].strip()])
```

Kuva 14: Kansion luonti ja datan tallennus tiedostoon.

	A	B	C	D
1	Timestamp	Advertisement Type	Value	
2	8:44:57	Radio Entropy	104	
3	8:44:58	Radio Entropy	217	
4	8:44:59	Radio Entropy	573	
5	8:45:00	Radio Entropy	422	
6	8:45:01	Radio Entropy	315	
7	8:45:02	Radio Entropy	133	
8				

Kuva 15: CSV-tiedosto datan tallennuksen jälkeen.

Yhtenä haasteena Tkinterissä oli se, että teksti ei automaattisesti ole kopioitavissa. On monia keinoja saada teksti ikkunaan, ja näiden tekstien saaminen kopioitavaksi on hieman erilainen riippuen siitä, mitä metodia käyttää tekstin lisäämiseen. Hankaluuksia tuotti se, että kopioitavissa oleva arvo tulee toisesta muuttujasta, joka saattaa vaihtua tai päivittyä. Alkuun käytin Tkinterin label-widgettiä. Tällä tavoin teksti on kuitenkin ainoastaan vain luettavissa. Helpoin tapa oli yhdistää entry ja insert-widget. Ensin luodaan muuttuja entry-widgetin avulla. Tämän jälkeen samaan muuttujaan lisätään haluttu kopioitavissa oleva arvo insertin avulla. Näiden jälkeen muuttuja on mahdollista asettaa sellaiseksi, että teksti on kopioitavissa. Tämä on kuitenkin hieman kömpelö tapa, jonka riskinä on, että koodin luettavuus kärsii.

```python
#Show serialnumber and makes it copyable
def read_serialnumber(self):
    global selected_serialnumber, serial_text2_entry
    selected_serialnumber = var2.get()
    selected_serialnumber = ast.literal_eval(f'b"{var2.get()}"')
    serial_text.config(text = "Serial number:  ")
    serial_text.place(x=30, y=150)
    serial_text2_entry = StringVar()
    serial_text2_entry = Entry(self, font=('Dubai', 10), width=20, textvariable=serial_text2_entry)
    serial_text2_entry.insert(0, selected_serialnumber[2:])
    serial_text2_entry.config(state='readonly', relief='flat')
    serial_text2_entry.place(x=120, y=151)
```

Kuva 16: Tekstin saaminen kopioitavaksi, kun muuttuja tulee muualta kuin käyttäjältä tai valmiiksi koodiin kirjoitettuna.

Tämä entryn ja insertin yhdistäminen ei kuitenkaan toiminut, kun kolmannen ikkunan tekstistä haluttiin saada kopioitava. Tämä saattaa johtua siitä, että muuttuja päivittyy jatkuvasti ja tulostaa reaaliajassa aina uuden palvelun, ominaisuuden ja datan. Päädyin ratkaisuun, jonka avulla saadaan kopioitua aina haluttu rivi. Tämän huonona puolena on se, että se kopioi kaiken samalla rivillä olevan tekstin, eikä ainoastaan haluttua kohtaa.

```python
#allows rows to be copyable
menu = tk.Menu(root, tearoff=0)
def copy_text(line_num):
    root.clipboard_clear()
    text = label3.cget("text").splitlines()[line_num]
    if line_num == 0:
        text += label4.cget("text")
    root.clipboard_append(text)

def show_menu(event):
    font_desc = label3.configure()["font"]
    line_height = tkfont.Font(font=(font_desc, 10, "normal", "roman")).metrics("linespace")
    line_num = int(event.y // line_height)
    menu.delete(0, tk.END)
    menu.add_command(label=f"Copy line {line_num}", command=lambda: copy_text(line_num))
    if line_num == 0:
        menu.add_command(label="Copy label4", command=lambda: root.clipboard_append(label4.cget("text")))
    menu.post(event.x_root, event.y_root)

label3.bind("<Button-3>", show_menu)
label4.bind("<Button-3>", show_menu)
```

Kuva 17: Rivien saaminen kopioitavaksi.

9 LOPPUPOHDINTA

Työn tekeminen on ollut erittäin opettavaista. Haasteita tuli eteen paljon, sillä aiempaa kokemusta Bluetoothin käyttö tässä tarkoituksessa ei ole ollut tuttua. Alkuun testausympäristön tekeminen oli siis todella hidasta, sillä uusien asioiden sisäistäminen vei oman aikansa. Loppua kohden alkoi sujumaan paremmin ja pieni ymmärrys esimerkiksi Bluetooth Low Energy:stä on syntynyt.

Tavoitteena oli tehdä testausympäristö, joka palvelisi paremmin kuin nyt käytössä oleva ympäristö. Tuloksena saatiin testausympäristö, jonka etuna edelliseen on sarjanumeron näkyminen, joka helpottaa oikean laitteen valitsemista. Työ täyttää isoimmat vaatimukset ja toimii siten, miten on haluttu.

Jatkoa ajatellen yhtenä suurena parannuksena halutaan tyylikkäämpi käyttöliittymä, selkeämpi ja virheystävällisempi koodi. Tällä hetkellä koodissa ei ole juurikaan virheenkäsittelyä ja virheen tapahtuessa on koko koodi suoritettava alusta. Lisäksi halutaan uusia toiminnallisuuksia muun muassa mahdollisuus datan näytön pysäyttämisestä, siten, että taustalla dataa edelleen tallentuu .CSV-tiedostoon, mutta käyttöliittymään ei uutta dataa päivity. Olisi myös hyvä, jos tulevaisuudessa sarjanumeron parsiminen tapahtuu käyttäjälle näkymättömästi.

Enemmän aikaa olisin halunnut käyttää käyttöliittymän visuaaliseen ilmeeseen. Tällaisenaan se on todella pelkistetty. Olisin myös halunnut muutaman toteutuksen tehdä eri tavalla. Jatkan kuitenkin tämän ympäristön kehitystä, joten se tulee vielä saamaan parannuksia. Testausympäristö oli kuitenkin lyötävä lukkoon tällaisenaan, muuten riskinä olisi ollut jatkuvien uusien vaatimusten lisääntyminen, enkä olisi ikinä saanut työtä dokumentoitua tähän opinnäytetyöhön. Kaiken kaikkiaan työn teko oli mieluista ja olen tyytyväinen siitä, millaisen ympäristön sain aikaiseksi sillä tietotaidolla, joka minulla oli.

LÄHTEET

Bittitaivas. (23.9.2022). Mikä Bluetooth on? Eri versiot selitetty ja näin tarkistat omasi (2023). Haettu 2.2.2023 osoitteesta https://bittitaivas.fi/bluetooth-versiot/

Bleak Documentation. (2020). Bleak Documentation. Haettu 10.4.2023 osoitteesta https://bleak.readthedocs.io/en/latest/index.html

Bluetooth. (2023). Bluetooth Wireless Technology. Haettu 3.3.2023 osoitteesta https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

Eurofins. (2023). Haettu 20.3.2023 osoitteesta https://www.eurofins.fi/electrical-and-electronics-fi/palvelumme/sertifiointi-ja-tuotehyvaeksyntaepalvelut/bluetooth-sig-hyvaeksyntae/

Kranz, A. (24.5.2022). Kaiken kattava Bluetooth Low Energy -opas. Haettu 24.3.2023 osoitteesta https://www.mokoblue.com/fi/guide-on-bluetooth-low-energy/

Laakso, J. (20.7.2017). Bluetoothin uusi solmuverkko avaa aivan uusia mahdollisuuksia tiedonsiirrolle. Haettu 20.3.2023 osoitteesta https://teknavi.fi/ilmio/bluetoothille-uusi-standardi-solmuverkko-avaa-uusia-mahdollisuuksia-tiedonsiirrolle/

Microchip. (n.d). BLE Roles, Devices, Characteristics, and Services. Haettu 28.4.2023 osoitteesta https://skills.microchip.com/android-development-process-for-bm70-rn4870/695181

Nordic Semiconductor. (2019). nRF52840 Dongle User Guide. https://infocenter.nordicsemi.com/pdf/nRF52840_Dongle_User_Guide_v1.1.pdf

Nordic Semiconductor. (2023). Bluetooth Low Energy. Haettu 8.3.2023

osoitteesta https://www.nordicsemi.com/Products/Bluetooth-Low-Energy/What-is-Bluetooth-Low-Energy?lang=en#infotabs

Oras Invest. (2023). Haettu 14.3.2023 osoitteesta https://orasinvest.fi/

Ostrowska, K. (20.6.2022). A Brief History of Python. Haettu 5.5.2023 osoitteesta https://learnpython.com/blog/history-of-python/

Qualcomm. (2023). Understanding the Generic Access Profile. Haettu 9.3.2023 osoitteesta https://developer.qualcomm.com/hardware/qca4020-qca4024/learning-resources/understanding-generic-access-profile

Silicon Labs. (2023a). Bluetooth Advertising Data Basics. Haettu 10.3.2023 osoitteesta https://docs.silabs.com/bluetooth/4.0/general/adv-and-scanning/bluetooth-adv-data-basics

Silicon Labs. (2023b). Bluetooth Connection Flowchart. Haettu 28.4.2023 osoitteesta https://docs.silabs.com/bluetooth/4.0/general/connections/bluetooth-connection-flowcharts

Singh, V. (12.5.2023). Advantages and Disadvantages of Python – Make a Favorable Decision. Haettu 24.5.2023 osoitteesta https://squareboat.com/blog/advantages-and-disadvantages-of-python

STT Info. (13.4.2022). Oras Investin vuosi 2021: positiivinen vuosi haastavista olosuhteista huolimatta. Haettu 15.3.2023 osoitteesta https://www.sttinfo.fi/tiedote/oras-investin-vuosi-2021-positiivinen-vuosi-haastavista-olosuhteista-huolimatta?publisherId=69819329&releaseId=69938253

Townsend, K. (2014). Introduction to Bluetooth Low Energy. Haettu 28.4.2023 osoitteesta https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

UpGrad. (29.9.2022.) Top 10 Reasons Why Python is So Popular With Developers in 2023. Haettu 24.5.2023 osoitteesta https://www.upgrad.com/blog/reasons-why-python-popular-with-developers/

LIITE 1: LÄHDEKOODI

```
# Tiedosto run.prog.py
from blatann import BleDevice
from tkinter.ttk import Label
from tkinter import *
from tkinter import ttk
import serial.tools.list_ports
import asyncio
from bleak import BleakScanner
import asyncio
import time
import time
from threading import Thread
import ast

window = Tk()
style = ttk.Style()
window.title('SCANNER')

window.resizable(False, False)

current_time = (time.strftime('%T'))

#Instantiate BLE Central Device
port = "COM4"
ports = list(serial.tools.list_ports.comports())
com_list = []
for i in ports:
    com_list.append(i)

#Opening ble device
ble_device = BleDevice(port)

try:
    ble_device.open()
    ble_device.close()
    print(current_time,'Device ok')
except:
    print(current_time,"Failed to open device")

oras_devices = []

#Scanning Devices
async def scan():
    start = time.time()
    time.sleep(1.5)
    print(current_time, "Scanning devices...")
```

```
    devices = await BleakScanner.discover(timeout=10)
    for d in devices:
            if d.name == 'ORAS':
                oras_devices.append([d] + [d.rssi] +
[d.metadata['manufacturer_data']])
            continue
    end = time.time()
    print(current_time, 'Scanning complete', f"\tTime
taken: {end-start} s")

    def show_ports():
        selected_port.config(text = "Selected port:  " +
clicked_port.get())
    clicked_port = StringVar()

    def show_devices():
        selected_device.config(text = '' + clicked_de-
vice.get())
    clicked_device = StringVar()

#Theme customing
    canvas = Canvas(window, width = 500,
                    height = 350)
    canvas.pack(fill = "both", expand = True)
    canvas.create_text(50, 40, text = "PORT", font='Du-
bai', fill='black')
    canvas.create_text(95, 135, text =  "DEVICES: (" +
str(len(oras_devices)) + " found)", font='Dubai',
fill='black')

    Button(window, height=1 , width=8, bg='#0D5185', text
= "Select", fg="white", command=show_ports).place(x = 30,
y = 83)

# Creating a Custom Theme
    style.theme_create('custom_style',
                       parent='clam',
                      )
    style.theme_use('custom_style')

    port_dropbox=ttk.OptionMenu(window, clicked_port,
ports[0], *ports)
    port_dropbox.pack()
    port_dropbox.place(x=30, y=50)

    device_dropbox=ttk.OptionMenu(window, clicked_device,
None, *oras_devices)
    device_dropbox.pack()
    device_dropbox.place(x=30, y=147)

    selected_port = Label(window, font=('Dubai', 10))
```

```
    selected_port.place(x=35, y=230)
    selected_device = Label(window, font=('Dubai', 10))
    selected_device.place(x=35, y=260)

    def notify():
        import run_serv_notify

    #Second window configure
    class ConnectWindow(Toplevel):

        def on_cancel(self):
            self.destroy()

        #Show serialnumber and makes it copyable
        def read_serialnumber(self):
            global selected_serialnumber, se-
rial_text2_entry
            selected_serialnumber = var2.get()
            selected_serialnumber = ast.lit-
eral_eval(f'b"{var2.get()}"')
            serial_text.config(text = "Serial number:  ")
            serial_text.place(x=30, y=150)
            serial_text2_entry = StringVar()
            serial_text2_entry = Entry(self, font=('Du-
bai', 10), width=20, textvariable=serial_text2_entry)
            serial_text2_entry.insert(0, selected_serial-
number[2:])
            serial_text2_entry.config(state='readonly',
relief='flat')
            serial_text2_entry.place(x=120, y=151)

        def delete_serial(self):
            serial_text2_entry.destroy()


        def __init__(self, master = None):
            super().__init__(master = master)

            global var2, serial_text

            self.title("CONNECT")
            self.geometry("600x550")
            self.resizable(False, False)
            Button(self, height=1 , width=8,
bg='#0D5185', text = "Read", fg="white", com-
mand=self.read_serialnumber).place(x = 30, y = 100)
            Button(self, height=1 , width=8,
bg='#0D5185', text = "Empty", fg="white", com-
mand=self.delete_serial).place(x = 150, y = 100)
```

```
            Button(self, height=1 , width=8,
bg='#0D5185', text = "← Previous", fg="white", com-
mand=self.on_cancel).place(x = 30, y = 500)
           #Button(self, height=1 , width=8,
bg='#0D5185', text = "Select", fg="white", com-
mand=self.read_address).place(x = 30, y = 207)
            Button(self, height=1 , width=15,
bg='#028503', text = "Services/Notify", fg="white", com-
mand=notify).place(x = 30, y = 215)

            #Displays the selected device in another win-
dow
            label = Entry(self,
state="readonly",font=('Dubai', 10))
            label.config(textvariable=clicked_device, re-
lief='flat',state="readonly", width=600)
            label.pack()
            label.place(x = 30, y = 10)

            #Add "Read serial number" text
            label2 = Entry(self, state="readonly",
font=('Dubai', 10))
            var = StringVar()
            label2.config(textvariable=var, re-
lief='flat',state="readonly", width=600)
            var.set('Read serial number')
            label2.pack()
            label2.place(x = 30, y = 50)

            #Creates serialnumber entrybox
            var2 = StringVar()
            entry = Entry(self, width=40, textvaria-
ble=var2)
            entry.pack()
            entry.place(x = 30, y = 75)
            serial_text = Label(self, font=('Dubai', 10))

           # address_text2 = Label(self, font=('Dubai',
10))

#Connect to device
    def connecting_to_device():
        Button(window, height=1 , width=8, bg='#0D5185',
text = "Select", fg="white", command=ConnectWin-
dow).place(x = 30, y = 180)

    thread1 = Thread(target=connecting_to_device)
    thread1.start()

asyncio.run(scan())
window.mainloop()
```

```
# tiedosto run.serv.notify.py
import tkinter as tk
from cleantext import clean
from tkinter.ttk import Label
from tkinter import *
import tkinter as tk
import asyncio
from bleak import BleakClient
import asyncio
import os
from cleantext import clean
import csv
import datetime
import tkinter.font as tkfont

# Create a Tkinter window with a label widget
root = tk.Tk()
root.geometry("650x400+900+450")
root.title('SERVICES/NOTIFY')

label3 = Label(root, text="\nServices:")
label3.pack()

label4 = Label(root, text="")
label4.pack()

label = Label(root, text="\nData from notification:")
label.pack()

# set the directory path where you want to save the CSV
file
path = os.path.join(os.path.dirname(__file__), "adver-
tisements")


#Notification handler for upcoming data
def notification_handler(handle: int, data: bytearray):
    data = data.decode()
    data = clean(data, no_emoji=True, lower=False)
    label.config(text=data)

#Creates directory and save the data
    if not os.path.exists(path):
        os.mkdir(path)
        print('Directory created!')

    data_parts = data.split(':')
```

```
    timestamp =
datetime.datetime.now().strftime('%H:%M:%S')

    with open(os.path.join(path, 'ble_advertise-
ments_data.csv'), 'a', newline='') as file:
        writer = csv.writer(file)
        if os.stat(os.path.join(path, 'ble_advertise-
ments_data.csv')).st_size == 0:
            writer.writerow(['Timestamp', 'Advertisement
Type', 'Value'])
        writer.writerow([timestamp,
data_parts[0].strip(), data_parts[1].strip()])

bluetooth_address = 'XX:XX:XX:XX:XX:XX'

#Search service charasteristics and start notify
async def discover_service_characteristic_uuid(address,
service_uuid):
    async with BleakClient(address) as client:
        label3.config(text=f"\nConnected to {blue-
tooth_address}")
        for service in client.services:
            await asyncio.sleep(1)
            print(service)
            label3.config(text=label3.cget("text") + f"\n
{service}")
            if str(service.uuid) == service_uuid:
                characteristics = service.characteristics
                for char in characteristics:
                    label4.config(text=la-
bel4.cget("text") + f"\n{service.uuid} characteristic
uuid is {char.uuid}")
                    await client.start_notify(char.uuid,
notification_handler)
                    continue
        await asyncio.sleep(5)
        await client.stop_notify(char.uuid)
        await client.disconnect()

#allows rows to be copyable
menu = tk.Menu(root, tearoff=0)
def copy_text(line_num):
    root.clipboard_clear()
    text = label3.cget("text").splitlines()[line_num]
    if line_num == 0:
        text += label4.cget("text")
    root.clipboard_append(text)

def show_menu(event):
    font_desc = label3.configure()["font"]
```

```
    line_height = tkfont.Font(font=(font_desc, 10, "nor-
mal", "roman")).metrics("linespace")
    line_num = int(event.y // line_height)
    menu.delete(0, tk.END)
    menu.add_command(label=f"Copy line {line_num}", com-
mand=lambda: copy_text(line_num))
    if line_num == 0:
        menu.add_command(label="Copy label4", com-
mand=lambda: root.clipboard_append(label4.cget("text")))
    menu.post(event.x_root, event.y_root)

label3.bind("<Button-3>", show_menu)
label4.bind("<Button-3>", show_menu)

async def main():

    # Start the BLE discovery and notification process in
a separate task.
    asyncio.create_task(discover_service_characteris-
tic_uuid(bluetooth_address, "service_uuid"))

#service_uuid

    # Run the main event loop and update the Tkinter win-
dow in real-time
    while True:
        root.update()
        await asyncio.sleep(0.01)
try:
    asyncio.run(main())
except:
    print()
```