Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

ALEKSI VIANTO

RFID OHJELMOINTI C# YMPÄRISTÖSSÄ

SÄHKÖ- JA AUTOMAATIOTEKNIIKAN
KOULUTUSOHJELMA
2020

Tekijä
Vianto, Aleksi

Julkaisun laji
Opinnäytetyö AMK

Päivämäärä
11 / 2020

Sivumäärä
29

Julkaisun kieli
Suomi

Julkaisun nimi
RFID OHJELMOINTI C# YMPÄRISTÖSSÄ

Tutkinto-ohjelma
Sähkö- ja automaatiotekniikan koulutusohjelma

Tämän opinnäytetyön tarkoituksena on antaa selkeä ohjelmointi ohjeistus lukijalle, jota hän voi itse soveltaa omissa RFID projekteissaan. Opinnäytetyössä käydään syvällisesti läpi, miten RFID-tunnisteiden lukemiseen tarvittavaan lukijaan yhdistäminen tapahtuu C#-kielen avulla käyttäen Visual Studio 2017 versiota.

Tässä opinnäytetyössä käsitellään myös yleisesti RFID-tekniikkaa, ja tietoa on otettu useammasta lähteestä, jotta on saatu monipuolinen kuva RFID-tekniikasta. Alussa perehdytään mitä tarvitaan, jotta tekniikkaa saa käytettyä. Tekniikasta käydään läpi myös historiaa, miten tekniikkaa nykyään käytetään ja mitä mahdollisuuksia tekniikalla on esimerkkien kautta.

Ohjelmoinnin osalta ei perehdytä C#-kieleen. Ohjelmoinnin termit käydään kuitenkin läpi, jotta lukija ymmärtää valmiiden koodipohjien ja esimerkkiohjelman koodien selitykset.

Opinnäytetyössä havaittiin sekä todettiin se, että ohjelman tekeminen C#-kielen avulla ei ole vaikeaa, mutta vaatii silti perusteet ohjelmoinnista, jotta esimerkki koodeja ymmärtää ja niistä saa kaiken tiedon irti. Todettiin myös, että koodin jatkokehitys on helppoa, ja sillä saa tehtyä monipuolisia sovelluksia RFID projekteihin.

Asiasanat
RFID, C#, Ohjelmointi

Author
Vianto, Aleksi

Type of Publication
Bachelor's thesis

Date
11 / 2020

Number of pages
29

Language of publication:
Finnish

Title of publication
RFID PROGRAMMING IN C# ENVIROMENT

Degree program
Electrical and Automation Engineering

The purpose of this thesis is to provide clear programming instructions to the reader, how to make software for reading RFID tags. The thesis provides an in-depth overview of how to connect to the RFID reader using C# code language with Visual Studio Version 2017.

This thesis also deals with RFID technology in general, and the information has been taken from several sources in order to get a comprehensive picture from the technology. "The history of RFID technology" is also discussed, "how RFID technology is used today" and "what possibilities technology has" with examples.

In terms of programming, we do not get into C# language. Programming terms are explained so that the reader understands explanations of code examples and example program code.

In the thesis, it was found and stated that making a program using C# code language is not difficult, but it still requires basic knowledge from programming in order to understand example codes and get everything out of them. It was also noted that further development of the code is easy, and you can make versatile RFID applications with it.

Key words
RFID, C#, Programming

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä keskitytään RFID-järjestelmän ohjelmointiin. Tarkoituksena on tehdä ohjeistus, jota soveltamalla voidaan lähteä tekemään sovelluksia RFID-tekniikkaa käyttäen. Opinnäytetyön alussa tutustutaan RFID-tekniikkaan yleisellä tasolla. Alussa käydään läpi, miten RFID-tekniikka toimii ja mihin sitä käytetään. Myös laitteistoon perehdytään yleisellä tasolla.

Opinnäytetyössä keskitytään RFID-tekniikkaan käyttäen JADAK:in valmistamia laitteita ja niiden ominaisuuksia. Ohjelmointiin siirryttäessä käydään läpi ohjelmointikielten vaihtoehdot, mutta syvemmin käydään läpi C# ympäristöä Visual Studiolla. Kun C# ohjelmointia käydään läpi, tukeudutaan JADAK:in "Mercury Api Programmers Guideen" ja käydään läpi mitä ohjelmakoodi yksinkertaisimmillaan sisältää. Opinnäytetyössä tehdään myös esimerkki ohjelma, joka on testattu toimivaksi. Opinnäytetyön koodit on kehitetty Mercury Apin esimerkki koodeista.

Osiossa "Ohjelmoinnin sanakirja" käydään läpi ohjelmoinnissa käytettävää sanastoa. Esimerkkikoodien selityksessä ja ohjelman tekemisen aikana ei syvennytä C#-kieleen. Esimerkki koodien selityksessä paneudutaan ThingMagic kirjaston "Reader" olion metodeihin ja miten niitä pitää käyttää, jotta ohjelman kanssa saadaan luotua yhteys ThingMagic sarjan lukijaan.

2 MITÄ ON RFID-TEKNIIKKA?

RFID (Radio Frequency Identification) on yleinen nimitys tunnistusmenetelmille, jotka toimivat radiosignaalien avulla tietyllä radiotaajuudella. Se mahdollistaa datan siirtymisen RFID-tunnisteen ja lukijan välillä.

Tekniikassa käytetään hyväksi tunnistetta, joka pystyy vastaanottamaan ja lähettämään takaisin radiosignaaleja. Tunnisteessa on myös vähän muistia, jotta tunnisteelle voidaan asettaa arvo ja tietoa. Tunnisteisiin lähetetään radiosignaaleja antennilla, joka on yhteydessä lukijaan. Kun antenni lähettää signaalin tunnisteelle, tunniste vastaanottaa radiosignaalin. Tämän jälkeen tunniste lähettää antennille takaisin radiosignaalin, jossa on muistissa oleva tieto. Tämän jälkeen antenni vastaanottaa radiosignaalin ja lähettää sen lukijalle. Lukijalta tieto voidaan ohjata taustajärjestelmään, josta voidaan lukea muistissa oleva arvo. Taustajärjestelmä on ohjelmoitu ohjelma, joka tekee datalle mitä on ohjelmoitu. [1]

RFID-tekniikka voittaa viivakoodi tunnisteen (barcode) siinä, että RFID-tekniikka ei tarvitse suoraa näköyhteyttä luettavaan kohteeseen. Lukeminen voi tapahtua esineiden läpi, kunhan materiaali on sellaista, joka päästävää radioaalto signaalin läpi. Tällaisia materiaaleja ovat esimerkiksi muovi ja puu. Radioaalto signaalia ei päästä läpi esimerkiksi metallit.

2.1 RFID-tekniikan yleisimmät käyttökohteet

RFID-tekniikka on saanut alkunsa, kun radiotekniikkaa on aloitettu käyttämään. Tunnisteita on radiosignaalien avulla aloitettu lukemaan toisen maailman sodan aikana. Lentokoneihin asennettavilla RFID-tunnisteilla nähtiin omat lentokoneet ja näin voitiin varmistaa, että omia lentokoneita ei tuliteta.[3]

Tietotekniikan osaamisen kasvaessa, tekniikkaa alettiin soveltaa esimerkiksi tuotantoeläinten merkitsemiseen, autojen käynnistyksenestoon [4] sekä teollisuudessa logistiikassa tavaroiden tunnistamiseen. [2]

2.2 Nykyiset RFID:n käyttökohteet

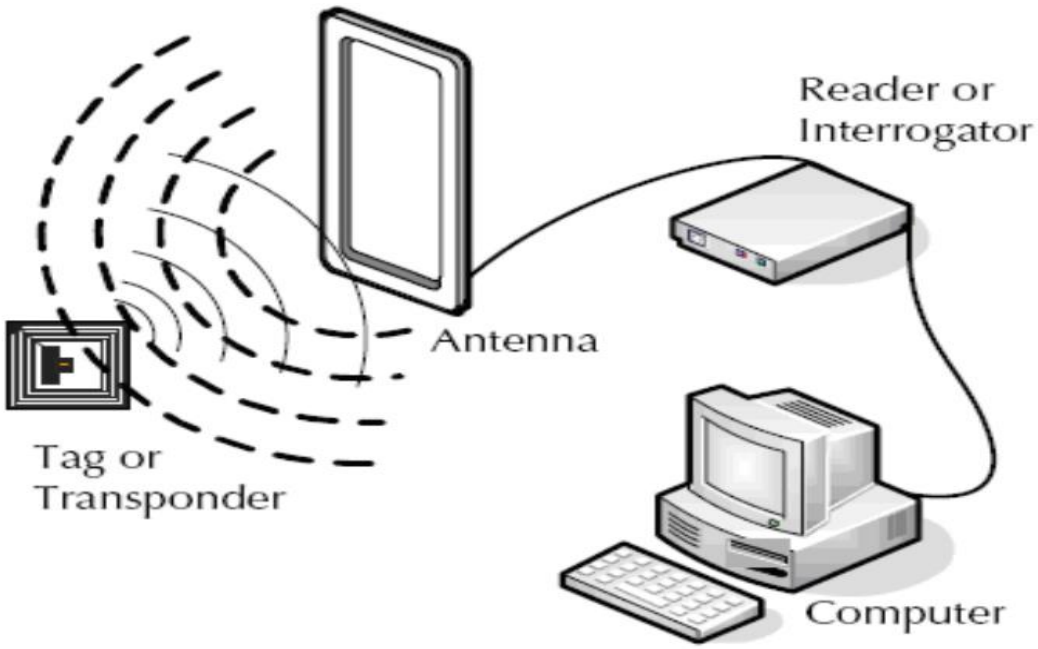
Nykyään tunnisteet on saatu kooltaan pieniksi ja lukuetäisyydet ovat kasvaneet, joten RFID-tekniikan käyttökohteet ovat laajentuneet huomattavasti. Teknologiaa käytetään

edelleen esimerkiksi teollisuudessa tuotannon tehostamisessa ja logistiikassa tavaravirtojen seuraamisessa, mutta myös esimerkiksi henkilötunnistukseen ja kulunvalvontaan, sekä varashälyttimissä. [5]

Esimerkiksi varastoinnin tilaa voidaan seurata asettamalla RFID-tunniste jokaiseen varastoitavaan tuotteeseen.[13] Tällä menetelmällä saadaan seurattua reaaliaikaisesti varaston tuotteiden määrää. RFID-tekniikkaa on myös käytetty esimerkiksi koirien opetusradalla, jossa tunnisteet asetetaan haisteltaviin näytepurkkeihin. Koiran on tarkoitus haistaa oikea näyte ja ilmoittaa siitä omistajalleen. Näin näytepurkkeja on helppo tunnistaa ja koirien opetus on tehokkaampaa.[12].

3 RFID-TEKNIIKAN LAITTEISTO

3.1 Laitteisto



Kuva 3.1 RFID-tekniikassa tarvittava laitteisto.

RFID-tekniikka tarvitsee toimiakseen RFID-lukijan, joka lähettää tunnisteiden tietoja tietokoneohjelmalle, josta niitä voi analysoida. RFID-lukija tarvitsee tunnisteiden lukemiseen antennin, joka lähettää ja vastaanottaa signaalin tunnisteelta. RFID-tekniikassa käytettävät tunnisteet vastaanottavat radiosignaalia RFID-lukijan antennilta, ja saavat tästä tarpeeksi energiaa, jotta tunnisteet voivat lähettää radiosignaalin takaisin. Tunnisteelta lähtevässä radiosignaalissa on mukana tunnisteen tiedot. [7]

3.1.1 Tunnisteet (tagit)

RFID-tekniikassa käytettään tunnisteita, joissa on kaksi osaa: radioaaltoja vastaanottava antenni sekä mikropiiri (Integrated Circuit). Tunnisteiden antennia kasvattamalla, saadaan lisättyä radiosignaalien vastaanotto- ja lähetyskykyä. Näin saadaan suurennettua etäisyyttä ja radiosignaalin energian voimakkuutta. Tunnisteet kommunikoivat antennille niin, että radiosignaali lähetetään lukijan antennilta tunnisteen antennille, josta tunniste saa energiaa. Tällä energialla tunniste menee päälle, ja muokkaa tunnisteen tiedot lähetettävään muotoon. Tämän jälkeen tunniste lähettää tiedot oman antenninsa avulla takaisin lukijan antennille. Tunnisteen lähettämää viestiä kutsutaan englanniksi termillä “backscatter”. Tunnisteet voidaan luokitella aktiivisiin, semipassiivisiin ja passiivisiin tunnisteisiin.[6]

Aktiivisella tunnistimella on oma virtalähteensä, joka antaa virtaa koko tunnisteelle. Näiden etuina on suurempi lukuetäisyys, mutta haittapuolena suurempi koko.

Semipassiivisella tunnisteella on myös oma virtalähteensä, mutta se antaa virtaa vain tunnisteen mikropiirille ja vaatii toimiakseen virtaa RFID-antennilta.

Passiiviset tunnisteet saavat kaiken virran RFID-antennilta. Passiivisten tunnisteiden haittapuolena on se, että ne tarvitsevat vahvaa radiosignaalia lukijan antennilta. [6]

Varmimman lukutuloksen saamiseksi tunnisteiden antenni kannattaa sijoittaa niin, että mahdollisimman paljon siitä näkyy lukijan antennille. Tällä menetelmällä lukijan an-

tennin lähettämät radiosignaalit osuvat parhaiten tunnisteen antenniin. Suurella etäisyydellä pienetkin kallistumat tunnisteen antennissa voivat vaikuttaa suuresti tunnisteiden luettavuuteen.

3.1.2 Antenni ja lukija

Antenni ja lukija toimivat yhteistyössä radiosignaalien lähettämisessä ja vastaanottamisessa. Antenni voi olla lukijassa kiinteästi, tai se voi olla ulkoinen antenni lukijan mallista riippuen. Antennin ja lukijan yhdistelmät saadaan tehtyä pieniksi, joten tunnisteita voidaan tunnistaa käsikäyttöisellä lukijalla. Antennien koko määräytyy lukemisen käyttötarkoituksella. Jos on käytössä esimerkiksi kulkukortti, niin antenni voi olla todella pieni, koska tunniste asetetaan lukijan eteen. Lukija toimii antennin aivoina ja se lähettää ja vastaanottaa tietoa antennilta ja taustajärjestelmältä. [7]

4 C# OHJELMOINNIN SANAKIRJA

Tässä opinnäytetyössä ei syvennytä C# ohjelmointiin. Opinnäytetyössä selitetään kuitenkin käytetyt termit, jotta lukija ymmärtää koodien selitykset paremmin. C# on olio-ohjelmointikieli (Object-Oriented Programming), joka on monimutkainen, mutta monipuolinen ohjelmointi tekniikka.[14] Olio-ohjelmointi perustuu muodostettaviin luokkiin, joille voidaan määritellä esimerkiksi muuttujia ja metodeja. Jos luokkaa haluaa käyttää, tehdään luokkaan perustuva olio, joka saa kaikki asetusarvot ja metodit, jotka luokassa on määritelty. Olio-ohjelmoinnin hyödyt saadaan laajoissa ohjelma kokonaisuuksissa, jolloin oliot selkeyttävät koodin lukemista ja kirjoittamista. [15]

4.1 Muuttuja

Muuttujalla tarkoitetaan asiaa, joka saa tietynlaisia arvoja. Arvot voivat erilaisia numeroita (int, double), tekstiä (string) tai päälle/pois muuttujia (bool). Arvoja voidaan

asettaa tai hakea muuttujista. Jos muuttujat määritellään staattiseksi (Static), niitä ei voi muuttaa. [17]

4.2 Metodi

Metodi tarkoittaa toimintoa, joka suoritetaan tekstipohjaisessa koodikielessä rivi kerrallaan. Ohjelmallisesti metodit koodataan ensin luokkaan, ja sitten niitä voi käyttää oliolla kutsumalla niitä. Metodeille voidaan asettaa parametrejä, joita metodin suorituksessa tarvitaan, mutta metodi voi myös olla ilman parametrejä, jolloin toiminto on tarkkaan määritelty oliolle. C# ohjelmoinnissa kirjastojen luokissa olevien metodien koodin toteutusta ei ole pakko sisäistää, kunhan tietää mitä metodi tekee oliolle. [14]

4.3 Luokka

Luokalla tarkoitetaan muuttujien ja metodien kokonaisuutta. Kun luokkaa halutaan käyttää, siitä on luotava olio. Olion alkuarvot määrää konstruktori. Jos konstuktoria ei itse määrittele, se luodaan automaattisesti. Tällöin kaikki arvot asetetaan tietotyypin mukaan joko false (bool), 0 (int, double) tai null (string). Konstruktori suositellaan määrittämään itse, jolloin voit asettaa aloitusarvot. Konstruktoreihin voi myös määritellä parametrit, jotka olion luontivaiheessa määritellään. Tätä kutsutaan konstuktorin ylikuormittamiseksi. [14]

4.4 Kirjasto

Kirjastolla tarkoitetaan ohjelmoinnissa käytettävää kokoelmaa, johon kuuluu valmis luokka tai monta luokkaa. Näitä luokkia voidaan käyttää omassa koodissa ilman, että niitä tarvitsee itse kirjoittaa. Jotta kirjastoa voi käyttää, se pitää lisätä omaan ohjelmaan ja se pitää kutsua luokan määrittelyssä. [16]

5 JADAK THINGMAGIC RFID OHJELMOINTI

Tässä opinnäytetyössä keskitytään Jadakin tuotteisiin ja kyseisen valmistajan laittei den ohjelmointiin. Jadak tuottaa räätälöityjä haivainnointi- ja analysointityökaluja muun muassa viivakoodin lukijoita, valo ja väri mittausjärjestelmiä sekä konenäkö järjestelmiä. [8] Myös RFID-tekniikan laitteistoja tuotetaan ja heillä on siihen oma tuoteryhmä, ThingMagic RFID.

ThingMagic RFID tuoteperheestä löytyy esimerkiksi RFID aloituspakkaus (Thing-Magic RFID Development Kit), johon kuuluu lukija ja antenni, RFID-tunnisteita, sekä ”Universal Reader Assistant”.

Universal Reader Assistant ohjelmalla saadaan havaittua RFID-tunnisteita ja asetettua niille arvoja. Ohjelman käyttöliittymässä näkyy myös esimerkiksi, kuinka monta kertaa samaan tunnisteeseen on saatu yhteys, sekä kuinka vahva yhteys on ollut. Ohjelmalla päästään hyvin RFID-tunnisteiden lukemisen maailmaan, ja sillä saa luettua ja asetettua tunnisteiden arvoja.

Aloitus-pakkauksen mukana tulee myös ”ThingMagic Mercury Api”, joka on Thing-Magic tuotteiden ohjelmointiin tarkoitettu ohjelmointi kirjasto. Mercury Api kirjaston avulla saadaan nopeasti luotua ohjelmia, joilla luetaan RFID-tunnisteita [9]

Tässä opinnäytetyössä käytetään hyväksi “Mercury Api Programmers Guide” opasta, sekä valmiita koodipohjia, jotka ovat C#-kielellä kirjoitettu. Koodipohjat löytyvät ThingMagic Mercury Apin tuotesivulta.[11] Tältä sivulta löytyy myös “Mercury Api Programmers Guide”.[10]

5.1 Eri ohjelmointi vaihtoehtoja

ThingMagic Mercury Api kirjastoa voi käyttää “Java”, “C” tai “C#” ohjelmointi kielillä. Nämä kaikki sopivat koodaukseen, kun käytetään Windows, Linux tai MacOS käyttöjärjestelmille luotuja ohjelmia. .NET ympäristössä käytetään C#-kieltä, android ympäristöön Java-kieltä ja iOS sekä mikrokontrollereihin C-kieltä. [10]

5.2 Ohjelmointi C# ympäristössä

Seuraavien esimerkkien avulla havainnollistetaan mitä metodeja käytetään C#-kielellä ohjelmoitaessa. Esimerkkien avulla saadaan koodattua ohjelman yhdistäminen lukijaan, sekä muodostettua metodi, jossa havaitun tunnisteen tietoihin päästään käsiksi. Seuraavien metodien avulla tehdään myös esimerkkiohjelma, jossa lukijaan yhdistetään, ja näytetään havaitun tunnisteen tiedot.

Ensimmäiseksi lisätään ThingMagic kirjasto luotuun Visual Studio ohjelmaan. Lisäyksen jälkeen se lisätään luokkaan komennolla “using ThingMagic”. Tämän jälkeen ohjelmassa luodaan “Reader” olio. Tämä olio on ThingMagic kirjaston luoma olio, jossa on kaikki tarvittavat metodit tietokoneelta lukijaan yhdistämiseksi. Reader oliolla on myös kaksi alemman luokan oliota, jotka ovat “SerialReader” ja “RqlReader”.

Reader olio luodaan komennolla “Reader reader = Reader.Create(readerIp)”. Create metodi palauttaa Reader olion, jonka IP osoitteeksi on määritelty readerIp. Reader olio laitetaan “using blockin” sisään, jotta saadaan automaattinen lukijan sammutus kun “using block” on luettu.

```
public static void EsimerkkiMetodi()
{
    try
    {
        //Reader olio luodaan ja sille asetetaan
        //parametriksi lukijan IP osoite
        using (Reader r = Reader.Create("192.168.0.1"))
        {

        }
    }
    catch{}
}
```

Kuva 5.1 Reader olion luominen.

Kun Reader olio on tehty, oliolla kutsutaan “connect” metodia, joka yrittää luoda yhteyden lukijaan. Metodi odottaa “transportTimeout” parametrin verran vastausta. Jos vastausta ei saada, tulee poikkeustiedote, jossa todetaan, että vastausta ei tullut. Connect metodin jälkeen pitää etsiä kaikki toiminta-alueet millä lukija voi operoida. Tämä

tapahtuu luomalla “Reader.Region” olioista lista. Ensimmäinen toiminta-alue syötetään Reader oliolle “ParamSet” metodilla. Jos toiminta-alueita ei ole, keskeytetään yhdistäminen lukijaan. Tämän jälkeen varmistetaan vielä, että antenniin yhdistys on mahdollista.

```
using (Reader r = Reader.Create("192.168.0.1"))
{
    //"Connect" metodi yrittää yhdistää lukijaan
    r.Connect();

    //Tämä IF lause etsii ja asettaa toiminta alueen lukijalle
    if (Reader.Region.UNSPEC == (Reader.Region)r.ParamGet("/reader/region/id"))
    {
        Reader.Region[] supportedRegions = (Reader.Region[])r.ParamGet("/reader/region/supportedRegions");
        if (supportedRegions.Length < 1)
        {
            throw new FAULT_INVALID_REGION_Exception();
        }
        r.ParamSet("/reader/region/id", supportedRegions[0]);
    }
}
```

Kuva 5.2 Toiminta-alueiden etsiminen.

Kun edellä olevat metodit on suoritettu onnistuneesti, lukijaan on saatu yhteys. Tämän jälkeen määritellään antennien lukeminen, protokollat mitä käytetään sekä tunnisteiden lukemiseen käytettävät suodattimet. Tähän voidaan käyttää “SimpleReadPlan” tai “MultiReadPlan” objektia. SimpleReadPlan objektilla on yksi “ReadPlan” objekti, kun taas MultiReadPlanilla niitä voi olla useampia.

Käytämme tulevassa esimerkkiohjelmassa SimpleReadPlan objektia ja se on yleisempi yksinkertaisissa tunnistus sovelluksissa. Luomalla kyseinen objekti komennolla “SimpleReadPlan plan = SimpleReadPlan()” voidaan luotavaan objektiin määritellä suodattimia, joita tunnisteiden lukemisessa käytetään. Yleisimmät suodattimet ovat käytettävien antennien määrä ja tunnisteiden lukemisessa käytettävä protokolla. Esimerkkiohjelmassa käytämme protokollaa “GEN2”, joka vakio protokolla.

```
//Tämä IF lause etsii ja asettaa toiminta alueen lukijalle
if (Reader.Region.UNSPEC == (Reader.Region)r.ParamGet("/reader/region/id"))
{
    Reader.Region[] supportedRegions = (Reader.Region[])r.ParamGet("/reader/
    if (supportedRegions.Length < 1)
    {
        throw new FAULT_INVALID_REGION_Exception();
    }
    r.ParamSet("/reader/region/id", supportedRegions[0]);
}
//Nämä rivit luovat "SimplePlan" olion,
//joka asetetaan lukijan "plan" parametriksi.
SimpleReadPlan plan = new SimpleReadPlan();
r.ParamSet("/reader/read/plan", plan);
```

Kuva 5.3 Tunnisteiden lukemisessa käytettävä suodatin olio.

Tunnisteiden lukemiseen tarvitaan metodi, joka suoritetaan kun tunniste havaitaan. Tätä kutsutaan “TagRead” metodiksi. Metodin sisälle ohjelmoidaan, mitä löydetyille tunnisteille tehdään. Tunnisteet tulevat “TagReadData” objekteina, jotka sisältävät niin sanotun “metadatan”, eli kaiken luetun datan. TagReadData sisältää myös “TagData” objektin, jotka ovat yksittäisten tunnisteiden tietoja.

```
//Nämä rivit luovat "SimplePlan" olion,
//joka asetetaan lukijan "plan" parametriksi.
SimpleReadPlan plan = new SimpleReadPlan();
r.ParamSet("/reader/read/plan", plan);

//Metodi "TagRead" käy toteen,
//kun lukija havaitsee tunnisteen.
r.TagRead += delegate (Object sender, TagReadDataEventArgs e)
{
    //Tähän koodataan, mitä tehdään kun tunniste havaitaan.
};
```

Kuva 5.4 Tunnisteiden lukemiseen tarvittava metodi.

Jos tunnisteiden lukiessa tulee jokin virhe, se ilmoitetaan “r_ReadException” metodilla. Tässä metodissa luodaan “ReaderException” olio, joka lisätään“Reader” olion “ReaderException” parametriin.

```
r.TagRead += delegate (Object sender, TagReadDataEventArgs e)
{
    //Tähän koodataan, mitä tehdään kun tunniste havaitaan.
};

//Tämä metodi käy toteen
//kun lukija havaitsee virheen
r.ReadException += new EventHandler<ReaderExceptionEventArgs>(r_ReadException);


private static void r_ReadException(object sender, ReaderExceptionEventArgs e)
{
    Console.WriteLine("Error: " + e.ReaderException.Message);
}
```

Kuva 5.5 Metodi virheiden ilmoitukseen.

Tämän jälkeen voidaan alkaa lukea tunnisteita. Tunnisteiden lukemiseen voidaan käyttää kahta metodia tarpeen mukaan, joko “Read” tai “StartReading”.

“Read” metodilla lukeminen on synkronista. Metodille voi myös asettaa ajan, jonka aikana antenni lukee tunnisteita ja tekee niistä “TagReadData” objekteja lukijan muistiin. Ajan päätyttyä palautuu lukijalta lista TagReadData objekteja, joista päästään käsiksi luettuihin tunnisteisiin ja niiden arvoihin.

"StartReading" metodilla lukeminen on asynkronista. Tällä metodilla syötetään jatkuvasti tunnisteen tietoja metodia kutsuvalle "Threadille" tai lukijasta riippuen, omalle Threadille. Lukemiseen voidaan vaikuttaa esimerkiksi lukemiseen kuluvan ajan kesto tai luettavien tunnisteiden välinen viive. StartReading metodia käytettäessä yksittäisiä "TagReadData" objekti korvaa aina tulevan, ja objektien säilytys pitää tehdä ohjelman puolella. Tunnisteiden lukeminen päättyy "StopReading" komennolla.

```
//Tämä metodi käy toteen
//kun lukija havaitsee virheen
r.ReadException += new EventHandler<ReaderExceptionEventArgs>(r_ReadException);

//"StartReading" metodi aloittaa lukemisen
r.StartReading();
//Lukijan annetaan lukea 10 000 millisekunttia
Thread.Sleep(10000);
//StopReading metodi pysäyttää lukemisen
r.StopReading();
```

Kuva 5.6 Lukemisen aloitus ja odotus metodit.

Jos tunnisteiden lukiessa tapahtuu jokin virhe, se ilmoitetaan joko:

"catch (ReaderException)" lauseella, jolla saadaan lukemisessa tapahtuneet virheet näkyviin.

"catch (Exception)" lauseella, jolla saadaan "try" lauseen sisällä tapahtunut muu virhe.

```
        Thread.Sleep(10000);
        //StopReading metodi pysäyttää lukemisen
        r.StopReading();
    }
}
//CATCH jos lukemisessa sattuu virhe
catch (ReaderException re) { Console.WriteLine(re.ToString()); }
//CATCH jos TRY lauseessa sattuu virhe
catch (Exception ex) { Console.WriteLine(ex.ToString()); }
```

Kuva 5.7 Virheet saadaan näkyviin "catch" lauseilla.

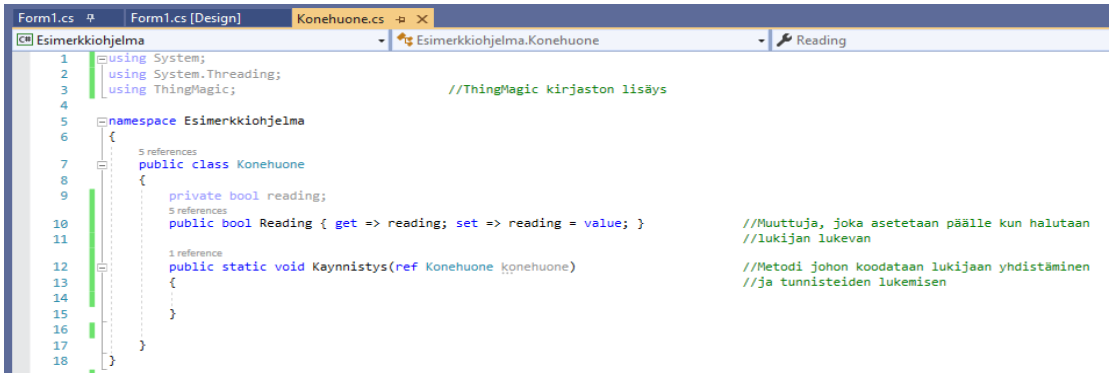
6 ESIMERKKIOHJELMA C# YMPÄRISTÖSSÄ

Seuraava esimerkkiohjelma on yksinkertainen tagien lukija. Ohjelmassa painetaan "Käynnistä" painiketta, jonka jälkeen lukijaan yritetään muodostaa yhteyttä. Jos yhteyksen luonti epäonnistuu, ilmoitetaan virhe ohjelman tekstikentässä. Jos lukijaan yhdistäminen onnistuu, ilmoitetaan siitä tekstikentässä. Tämän jälkeen lukija yrittää lukea tunnisteita. Kun se saa tunnisteen luettua, tulostetaan tekstikenttään "TagReadData" olion tiedot.

Esimerkkiohjelman tarkoitus on havainnollistaa, mitä näin yksinkertaiseen ohjelman tekemiseen vaaditaan ohjelmakoodin osalta. Esimerkki ohjelmassa käytetään hyväksi ThingMagic Mercury Apin valmiita koodipohjia, jotka löytyvät kyseisen tuotteen tuotesivulta.[11] Koodi, jota käytämme pohjana, on "ReadAsync".

6.1 Aloitus

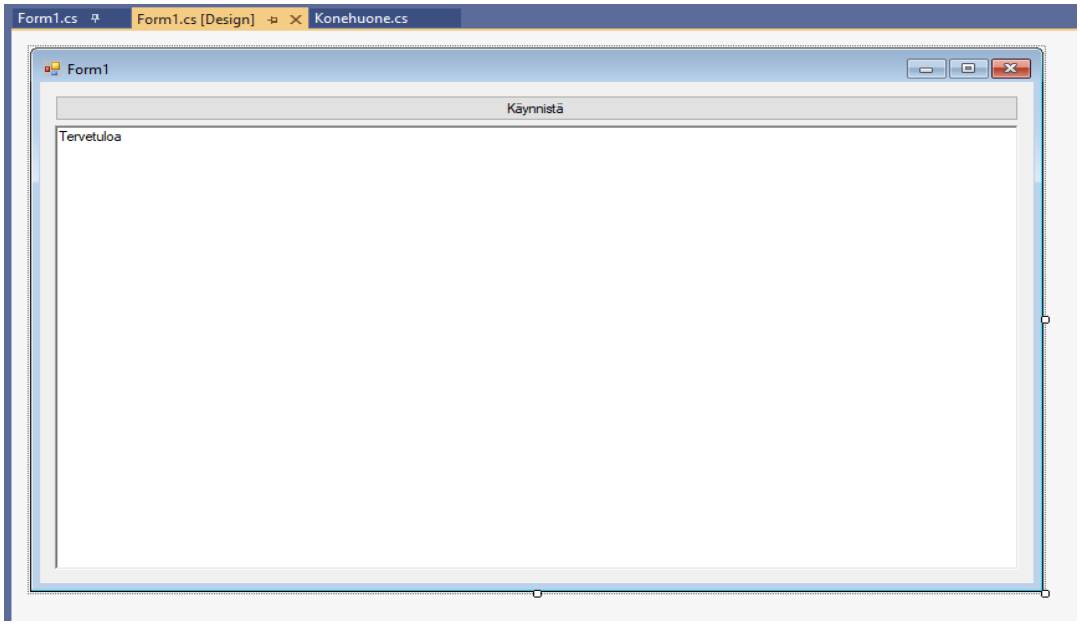
Luominen aloitetaan tekemällä uusi WinForms ohjelma. Ohjelmaan lisätään “ThingMagic” kirjasto. Tämän jälkeen on hyvä luoda julkinen luokka "Konehuone", jossa lukijaan yhdistäminen ja lukeminen tapahtuu. Konehuone luokkaan lisätään “ThingMagic” kirjasto ja “bool” tyyppinen muuttuja "Reading", jota voidaan ohjata käyttöliittymä puolelta. Luokkaan lisätään myös julkinen staattinen metodi "Kaynnistys", jolle annetaan "Konehuone" olio referenssi muuttujaksi.



Kuva 6.1”Konehuone” luokan sisältö.

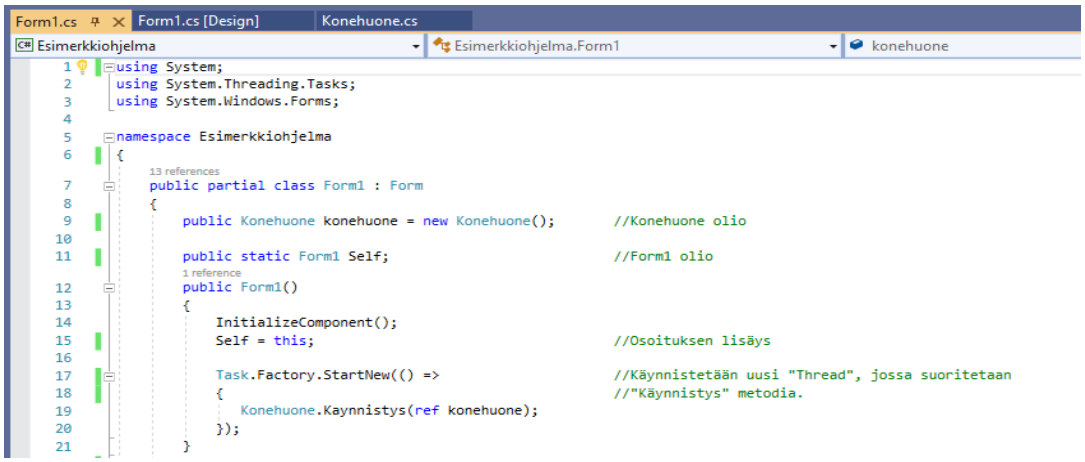
6.2 Form1 luokka

Ohjelman käyttöliittymään tarvitaan painike (Button) ja tekstikenttä (RichTextBox). Nimetään nappula "btnReading" nimiseksi, ja annetaan sille teksti "Käynnistä". Annetaan tekstikentälle nimeksi "richTextBox" ja annetaan sille teksti "Tervetuloa".



Kuva 6.2 Form1 käyttöliittymän näkymä.

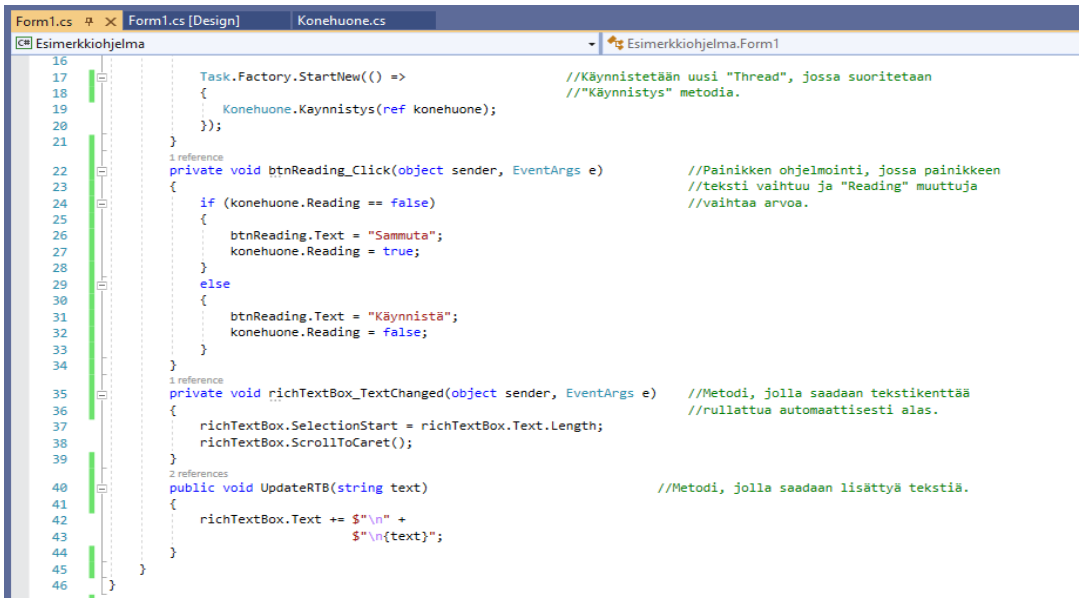
Tämän jälkeen voidaan siirtyä käyttöliittymän ohjelmointiin. Aloitetaan lisäämällä ohjelmaan "Konehuone" luokan olio ja staattinen Form1 olio "Self". Metodiin, jolla alustetaan käyttöliittymä, lisätään osoitus "Self = this". Tämän jälkeen luodaan uusi säie (jatkossa “Thread”), jossa konehuone olion metodi "kaynnistys" suorittuu. Metodiin asetetaan luotu konehuone olio referenssiksi.



Kuva 6.3”Form1” käyttöliittymän ohjelman alku.

Tämän jälkeen voidaan lisätä metodeja käyttöliittymään. "btnReading" painikkeeseen lisätään "Click" metodi. Metodissa katsotaan, onko konehuoneen "Reading" muuttuja tosi. Jos ei ole, niin painikkeen tekstiksi vaihdetaan "Sammuta", ja konehuone olion "Reading" muuttuja muutetaan todeksi. Jos taas "Reading" ei ole tosi, niin painikkeen tekstiksi muuttuu "Käynnistä" ja muuttujan arvoksi epätosi.

Tekstikenttään lisäämme metodin "TextChanged", jonka tarkoituksena on rullata tekstikenttää automaattisesti alemmas, kun uutta tekstiä tulee tekstikenttään. Luodaan myös metodi "UpdateRTB", johon voi asettaa tekstiä. Metodin tarkoitus on luoda väli jokaiselle tekstikentälle.

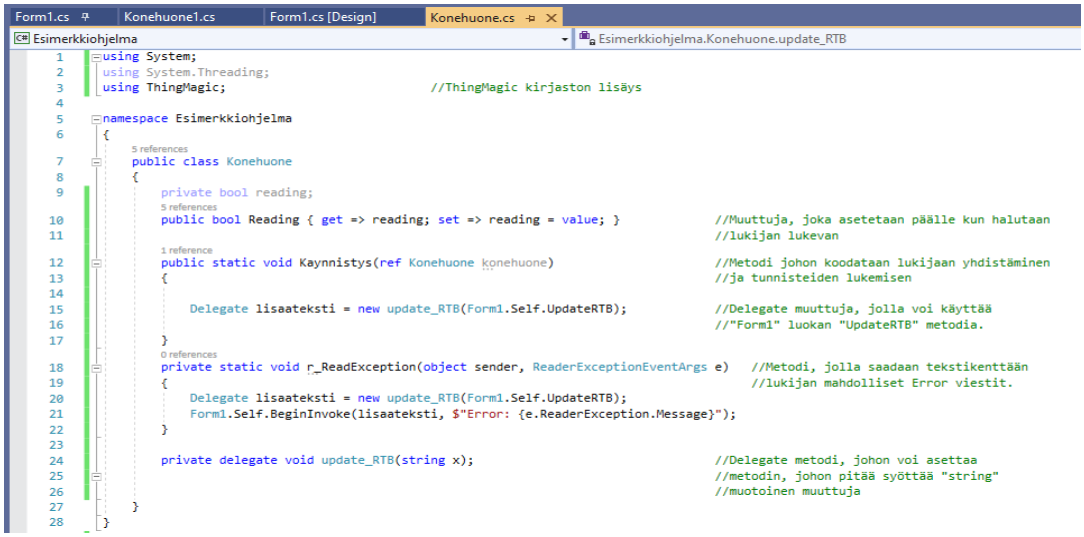


Kuva 6.4 ”Form1” käyttöliittymän ohjelman loppu.

6.3 Konehuone luokka

Konehuone luokkaan on tähän asti tehty muuttuja "OnOff" ja staattinen metodi "Kaynnistys". Kommunikointi "Kaynnistys" metodin ja “Form1” luokan tekstikentän välillä on haastavaa, koska ne ovat eri “Threadeissa”. Kommunikointi pitää suorittaa Delegate muuttuijen avulla. Lisäämällä “Kaynnistys” metodin sisälle delegate muuttuja "lisaateksti", johon asetetaan delegate metodi "update_RTB", johon on asetettu Form1 luokassa luotu "UpdateRTB" metodi. Tällä menetelmällä voidaan käyttää tekstin lisäystä tekstikenttään.

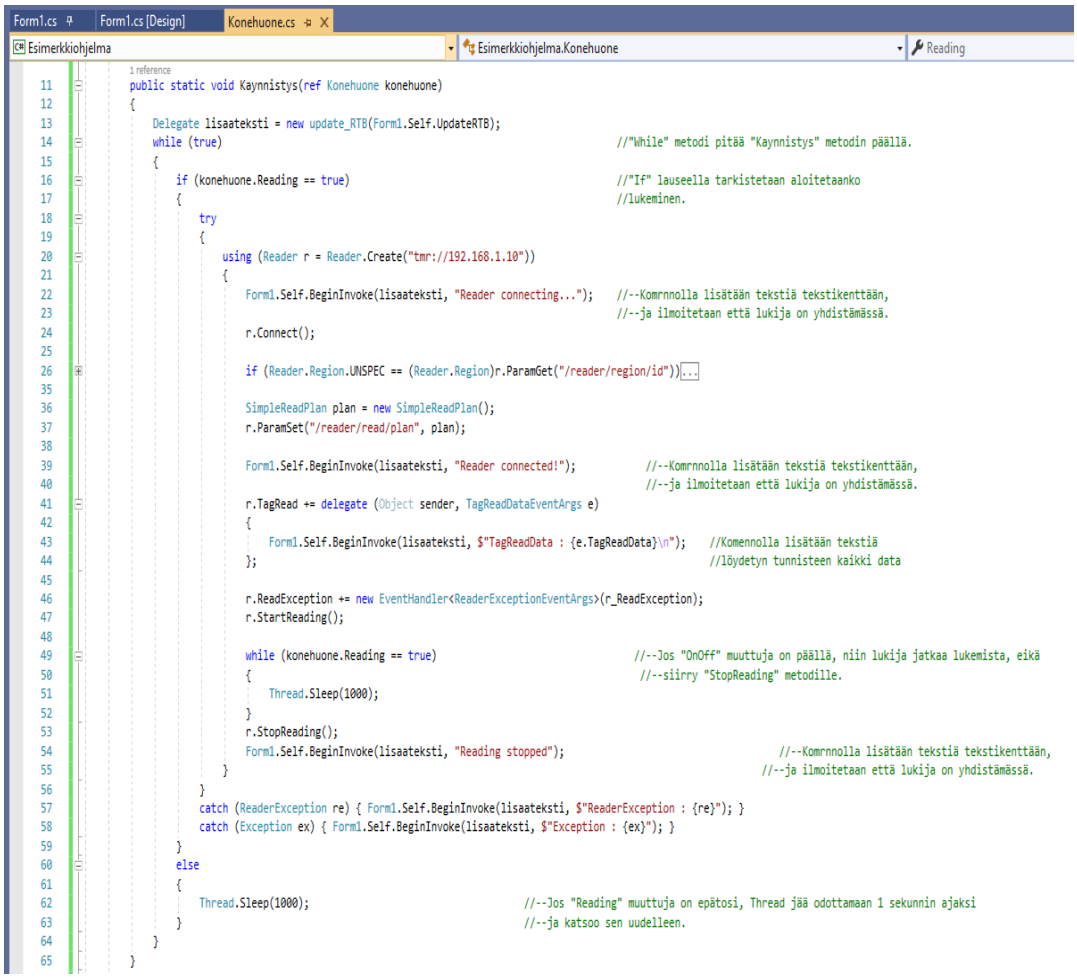
Tehdään myös staattinen metodi "r_ReadException", jolla saa esiin mahdolliset lukijan virhe ilmoitukset. “r_readException” metodiin lisätään myös samanlainen “lisaateksti” muuttuja, kuin “Kaynnistys” metodissa.



Kuva 6.5 ”Konehuone” luokkaan metodien lisäystä.

Seuraavaksi keskitytään "Kaynnistys" metodin sisältöön. Metodin tarkoitus on olla jatkuvasti päällä ja odottaa, että käynnistyspainiketta painetaan. Ohjelman käynnistyessä ehtolause katsoo, onko "Reading" asetettu päälle. Jos ei ole, “Käynnisty” metodi odottaa yhden sekunnin ja aloittaa alusta. Kun "Reading" asetetaan päälle, aloitetaan yhdistäminen.

Lukijaan yhdistäminen ja tunnisteiden lukeminen on esitelty opinnäytetyön osiossa "Ohjelmointi C# ympäristössä", ja käytämme siinä tehtyä metodia “Käynnistys” metodin pohjana. Kuvassa 6.6 lisätyt kohdat on kommentoitu, ja kommentissa on selitetty mitä kyseinen lisäys tekee.



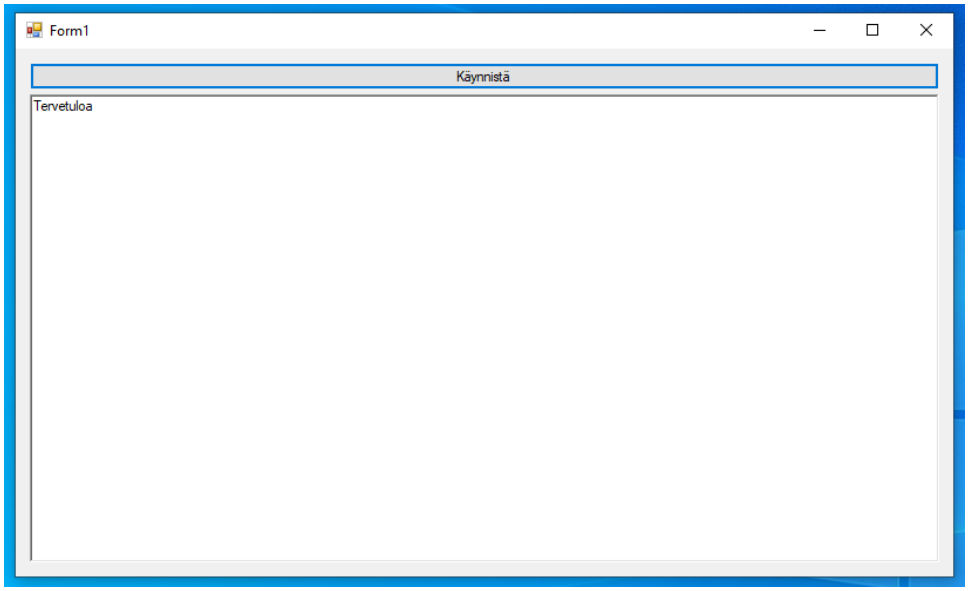
Kuva 6.6 ”Konehuone” luokka kokonaisuudessaan.

6.4 Käytetyt laitteet ja kommunikointi

Ohjelmaa testatessa tarvitaan RFID-tunnisteiden lukemiseen tarvittavat laitteet. Sain lainattua laitteet testausta varten. Lainaamani laitteet ovat “ThingMagic UHF Fixed/Finished Reader Development Kit” paketissa tulevat tuotteet, joihin kuuluvat lukija ja antenni. Lukijan ja antennin välissä on antenni kaapeli, josta tieto siirtyy antennilta lukijaan. Lukijan ja tietokoneen kommunikointia varten sain myös reitittimen, jonka kautta kommunikoidaan. Yhdistin lukijan tietokoneeseen ja asetin lukijan IP osoitteeksi “192.168.1.10”. Asetin reitittimen IP osoitteeksi “192.168.1.100”. Yhdistämällä tietokone reitittimeen voidaan kommunikoida lukijan kanssa.

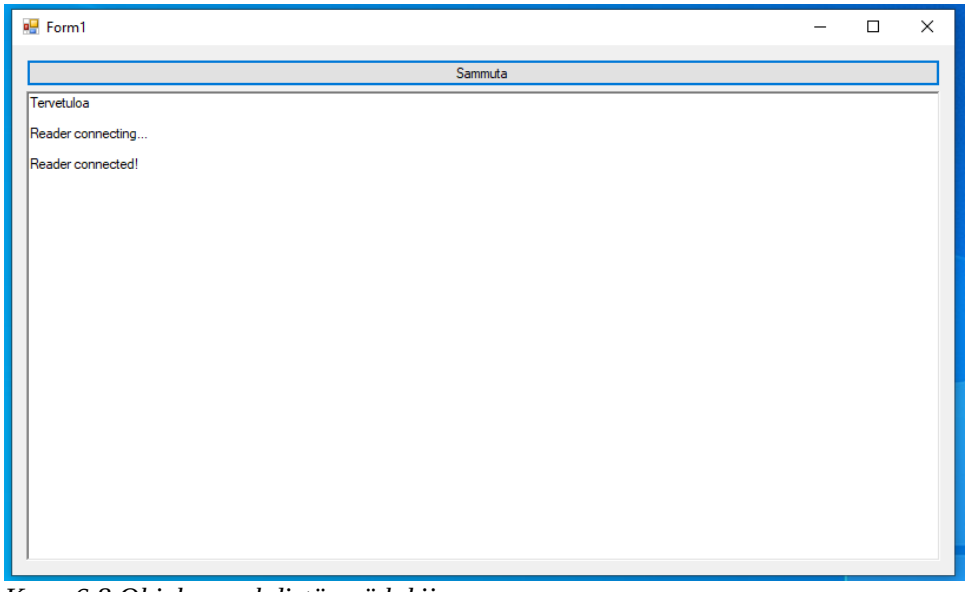
6.5 Ohjelman testaus

Kun ohjelma on käynnistynyt, näkyy “Tervetuloa” teksti.



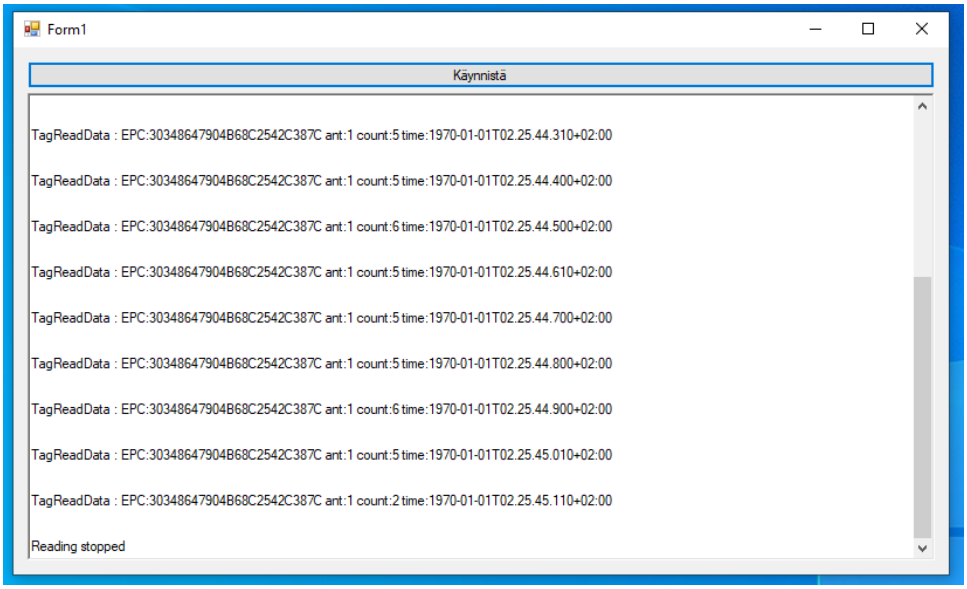
Kuva 6.7 Ohjelman käynnistyessä.

Kun “Käynnistä” painiketta painaa, painikkeen teksti vaihtuu “Sammuta” tekstiksi. Ohjelma myös tulostaa tekstin “Connecting to reader...” ja tämän jälkeen yritetään yhdistää lukijaan. Lukijaan yhdistäminen on onnistunut, kun teksti “Reader connected!” tulee näkyviin. Tämän jälkeen lukija aloittaa lukemisen.



Kuva 6.8 Ohjelman yhdistäessä lukijaan.

Ja kun antennin eteen asetetaan tunniste, niin ohjelma tulostaa “TagReadData” olion tiedot. Kun painetaan “Sammuta” näppäintä, niin lukija lopettaa lukemisen ja tulee teksti “Reading stopped”.



Kuva 6.9 Ohjelma lukee tunnisteita, ja on lopettanut lukemisen.

7 POHDINTA

C# ohjelmointikokemusta itselläni on muutaman nettikurssin, sekä muutaman koulussa käydyn kurssin verran. Näillä tiedoilla pärjäsin hyvin, ja ymmärsin kaiken esimerkkikoodeista, jotka olivat ThingMagic Mercury Apin mukana tulleet.

Ohjelmointi käyttäen JADAK:in MagicThing tuotteita on mielestäni tehty todella helpoksi aloittaa. Heidän sivuiltaan löytyvä “Programmers Guide” ja esimerkki koodeilla pääsee todella helposti kiinni RFID ohjelmointiin, ja miten eri tavoilla RFID-tekniikkaa voi hyödyntää heidän tuotteillaan. Valmiita koodipohjia on myös helppo soveltaa ja tehdä niiden pohjalta omia sovelluksia. Tehdessäni esimerkkiohjelmaa käytinkin heidän valmista “ReadAsync” koodipohjaa, joka löytyy heidän esimerkki koodeista. Sovelluksen tekeminen tällä pohjalla ja C#-kielen osaamisella oli helppoa, eikä tuottanut suurempia haasteita. Ohjelma on yritetty tehdä mahdollisimman yksinkertaisesti, ja sen avulla on helppo jatkokehittää ja lisätä ominaisuuksia lukemiseen.

Ennen opinnäytetyötä olin työskennellyt RFID-tekniikan parissa ohjelmoijana, mutten suuremmin ollut keskittynyt siihen mitä tapahtuu yhdistäessä lukijaan. Tämä johtui siitä, että koodi oli jo saatu siihen vaiheeseen, että lukijaan yhdistäminen onnistui. Opinnäytetyötä tehdessä opin enemmän juuri tästä tärkeimmästä vaiheesta eli miten saadaan lukijaan yhteys ja miten lukeminen oikeasti tapahtuu. Opin myös mitä eri mahdollisuuksia on ohjelmoinnissa ja miksi eri ominaisuuksia kannattaa käyttää eri tilanteissa. Tulevaisuuden RFID ohjelmointiprojekteissa aion soveltaa oppimaani saadakseni luotua parempaa koodia, jotta lukemisesta tulisi mahdollisimman tehokasta käytettävän projektin tarpeisiin.

RFID-tekniikan käyttökohteista ja toiminnasta opin myös enemmän, koska ennen olin vain keskittynyt käyttöliittymä koodin tekemiseen. RFID-tekniikka on kehittynyt selvästi teknologian kehittyessä, ja sille tullaan luultavasti keksimään yhä enemmän käyttökohteita erilaisiin tunnistus tarpeisiin. Työskennellessäni RFID-tekniikan parissa, olemmekin projektissa mukana olleiden kanssa suunnitelleet, mitä kaikkea tekniikkaa hyödyntämällä saa aikaiseksi. Mielestäni uusien innovaatioiden kehittäminen on tärkeää ja haluan olla mukana sitä RFID ohjelmointi osaamisellani.

LÄHTEET¨

[1] https://toptunniste.fi/rfid-nfc-teknologia/ Viitattu 07.10.2020.

[2] http://www.rfidlab.fi/rfid-teknologia/mita-on-rfid/ Viitattu 07.10.2020.

[3] https://www.electronics-notes.com/articles/connectivity/rfid-radio-frequency-identification/development-history.php Viitattu 07.10.2020.

[4] http://www.autonavain.fi/ajonesto-tekniikka.php Viitattu 07.10.2020.

[5] http://www.rfidlab.fi/rfid-teknologia/soveltamisalueet/ Viitattu 07.10.2020.

[6] https://www.atlasrfidstore.com/what-are-uhf-rfid-tags/?utm_source=RFID-Beginners-Guide&utm_medium=eBook&utm_campaign=Content&utm_content=tag-guide Viitattu 07.10.2020.

[7] https://www.atlasrfidstore.com/rfid-beginners-guide Viitattu 10.10.2020.

[8] https://www.jadaktech.com/about-us/ Viitattu 10.10.2020.

[9] https://www.jadaktech.com/products/thingmagic-rfid/thingmagic-uhf-fixed-finished-reader-development-kit/ Viitattu 10.10.2020.

[10] https://www.jadaktech.com/resources/rfid-document-library/ Viitattu 10.10.2020.

[11] https://www.jadaktech.com/products/thingmagic-rfid/thingmagic-mercury-api/ Viitattu 10.10.2020.

[12] https://toptunniste.fi/helsinki-vantaan-lentokentalla-tyoskentelevia-koronakoiria-koulutetaan-rfid-teknologian-avulla/ Viitattu 10.10.2020.

[13] https://toptunniste.fi/ratkaisut/smart-cabinet/ Viitattu 14.10.2020.

[14] http://www.netti-lakka.com/Ohjelmoimaan/CS_Olio-opas.pdf Viitattu 14.10.2020.

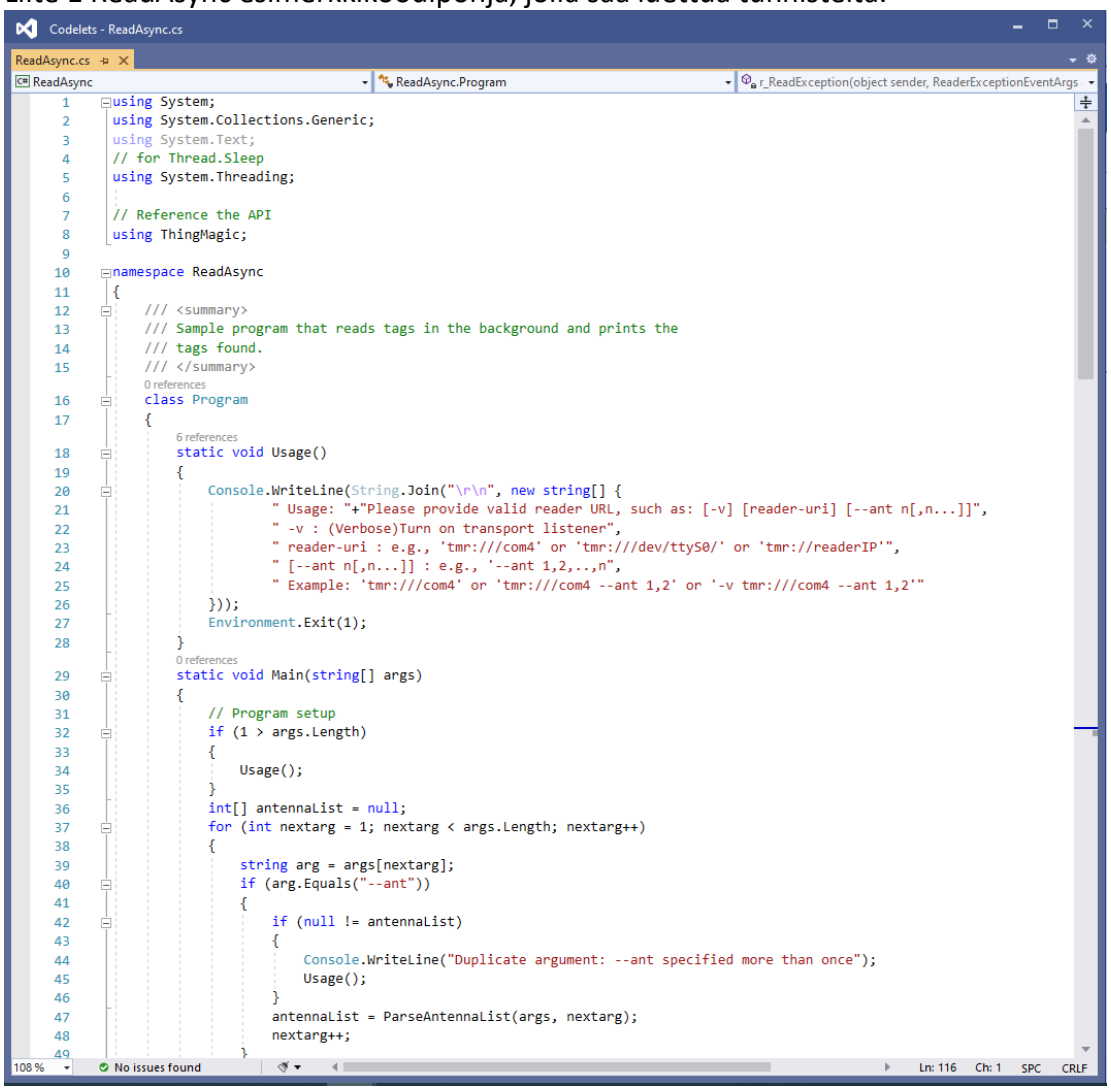
[15] https://www.cs.helsinki.fi/u/ahslaaks/kesa11/ohja/ Viitattu 14.10.2020.

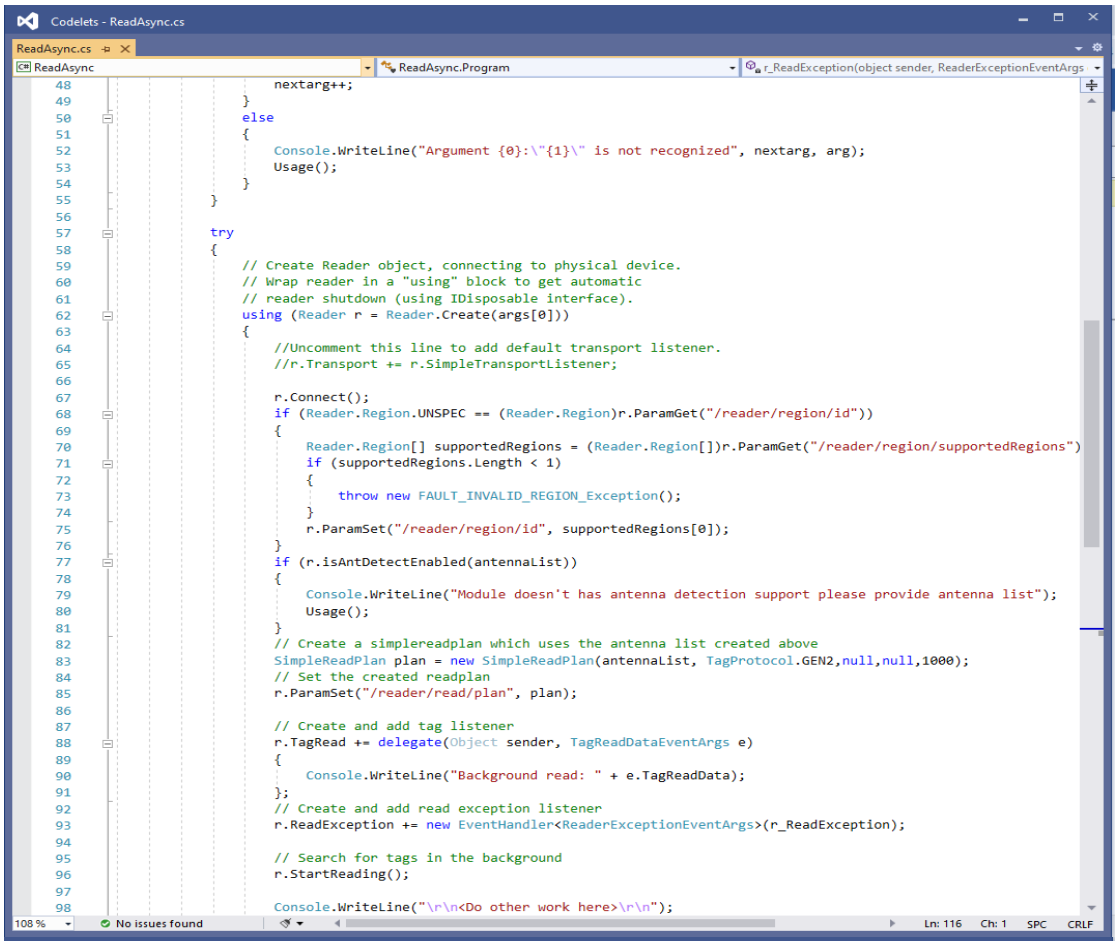
[16] https://en.wikipedia.org/wiki/Library_(computing)#Shared_libraries Viitattu 14.10.2020.

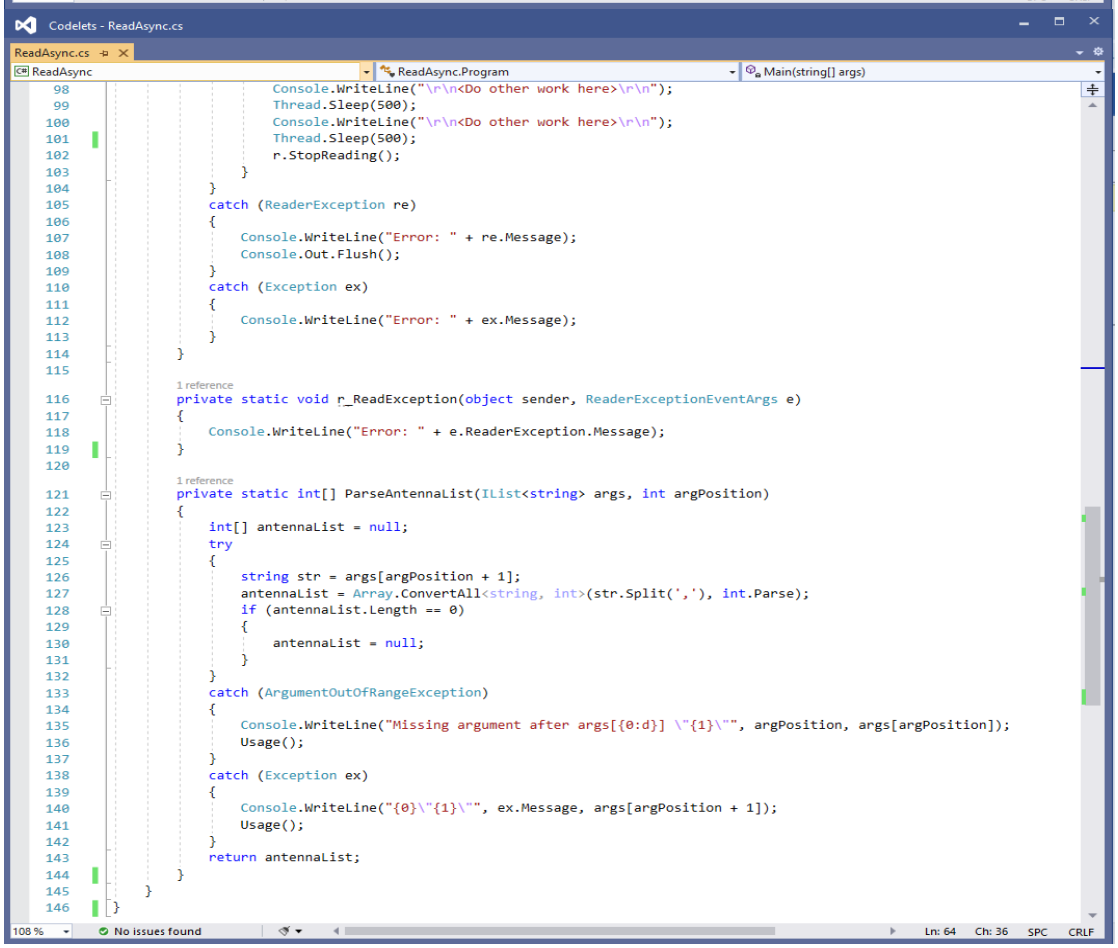
[17] https://fi.wikipedia.org/wiki/Muuttuja_(ohjelmointi) Viitattu 14.10.2020.

LIITTEET

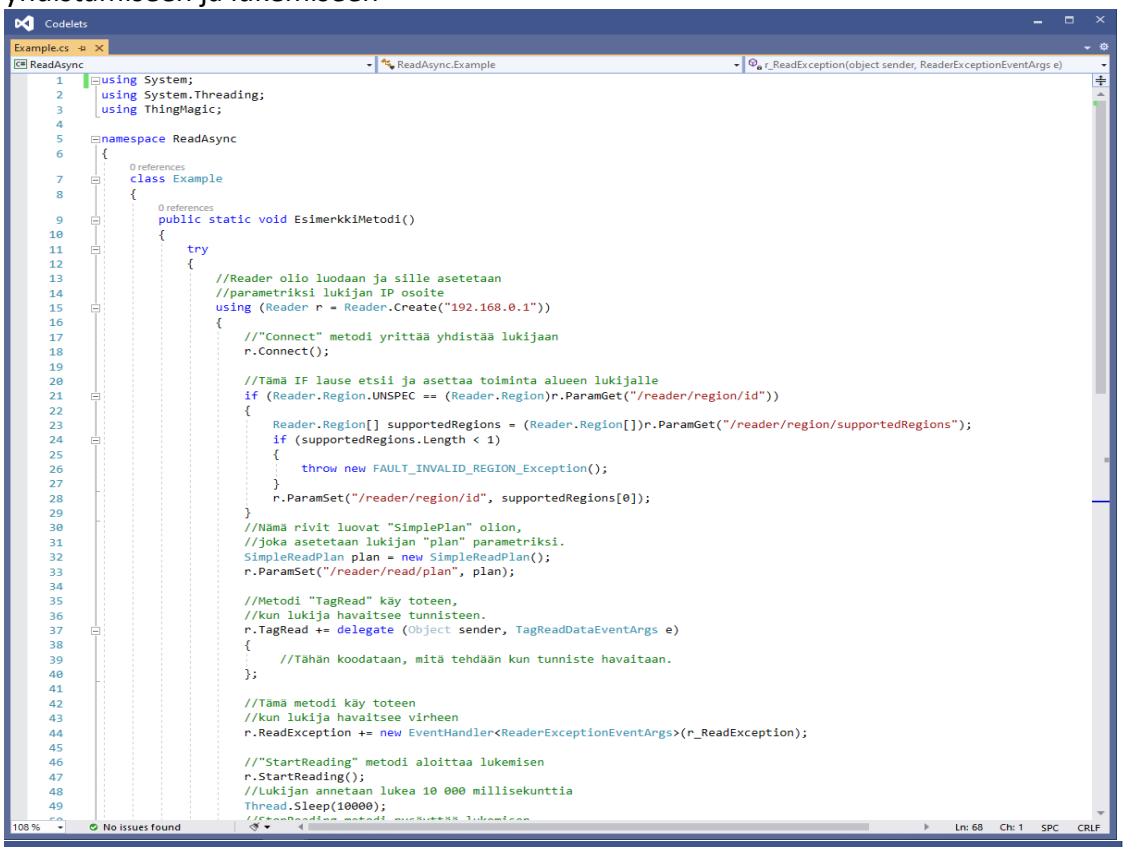
Liite 1 ReadAsync esimerkkikoodipohja, jolla saa luettua tunnisteita.




Codelets - ReadAsync.cs
ReadAsync.cs
ReadAsync
ReadAsync.Program
r_ReadException(object sender, ReaderExceptionEventArgs
nextarg++;
}
else
{
Console.WriteLine("Argument {0}:\"{1}\" is not recognized", nextarg, arg);
Usage();
}
}
try
{
// Create Reader object, connecting to physical device.
// Wrap reader in a "using" block to get automatic
// reader shutdown (using IDisposable interface).
using (Reader r = Reader.Create(args[0]))
{
//Uncomment this line to add default transport listener.
//r.Transport += r.SimpleTransportListener;
r.Connect();
if (Reader.Region.UNSPEC == (Reader.Region)r.ParamGet("/reader/region/id"))
{
Reader.Region[] supportedRegions = (Reader.Region[])r.ParamGet("/reader/region/supportedRegions")
if (supportedRegions.Length < 1)
{
throw new FAULT_INVALID_REGION_Exception();
}
r.ParamSet("/reader/region/id", supportedRegions[0]);
}
if (r.isAntDetectEnabled(antennaList))
{
Console.WriteLine("Module doesn't has antenna detection support please provide antenna list");
Usage();
}
// Create a simplereadplan which uses the antenna list created above
SimpleReadPlan plan = new SimpleReadPlan(antennaList, TagProtocol.GEN2,null,null,1000);
// Set the created readplan
r.ParamSet("/reader/read/plan", plan);
// Create and add tag listener
r.TagRead += delegate(Object sender, TagReadDataEventArgs e)
{
Console.WriteLine("Background read: " + e.TagReadData);
};
// Create and add read exception listener
r.ReadException += new EventHandler<ReaderExceptionEventArgs>(r_ReadException);
// Search for tags in the background
r.StartReading();
Console.WriteLine("\r\n<Do other work here>\r\n");
108 %
No issues found
Ln: 116
Ch: 1
SPC
CRLF

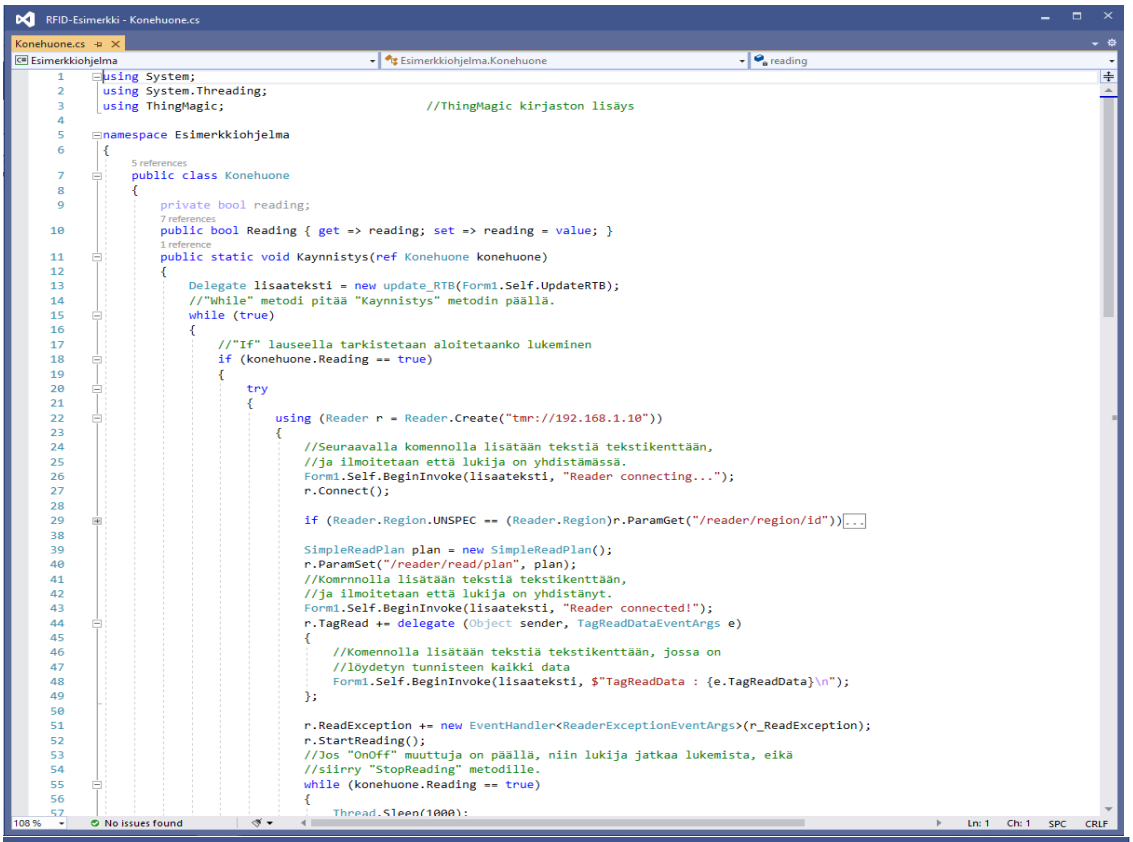

Codelets - ReadAsync.cs
ReadAsync.cs
ReadAsync
ReadAsync.Program
Main(string[] args)
Console.WriteLine("\r\n<Do other work here>\r\n");
Thread.Sleep(500);
Console.WriteLine("\r\n<Do other work here>\r\n");
Thread.Sleep(500);
r.StopReading();
}
}
catch (ReaderException re)
{
Console.WriteLine("Error: " + re.Message);
Console.Out.Flush();
}
catch (Exception ex)
{
Console.WriteLine("Error: " + ex.Message);
}
}
1 reference
private static void r_ReadException(object sender, ReaderExceptionEventArgs e)
{
Console.WriteLine("Error: " + e.ReaderException.Message);
}
1 reference
private static int[] ParseAntennaList(IList<string> args, int argPosition)
{
int[] antennaList = null;
try
{
string str = args[argPosition + 1];
antennaList = Array.ConvertAll<string, int>(str.Split(','), int.Parse);
if (antennaList.Length == 0)
{
antennaList = null;
}
}
catch (ArgumentOutOfRangeException)
{
Console.WriteLine("Missing argument after args[{0:d}] \"{1}\"", argPosition, args[argPosition]);
Usage();
}
catch (Exception ex)
{
Console.WriteLine("{0}\"{1}\"", ex.Message, args[argPosition + 1]);
Usage();
}
return antennaList;
}
}
}
108 %
No issues found
Ln: 64
Ch: 36
SPC
CRLF

Liite 2. ReadAsync esimerkki koodipohjasta otettu tarvittavat metodit yhdistämiseen ja lukemiseen

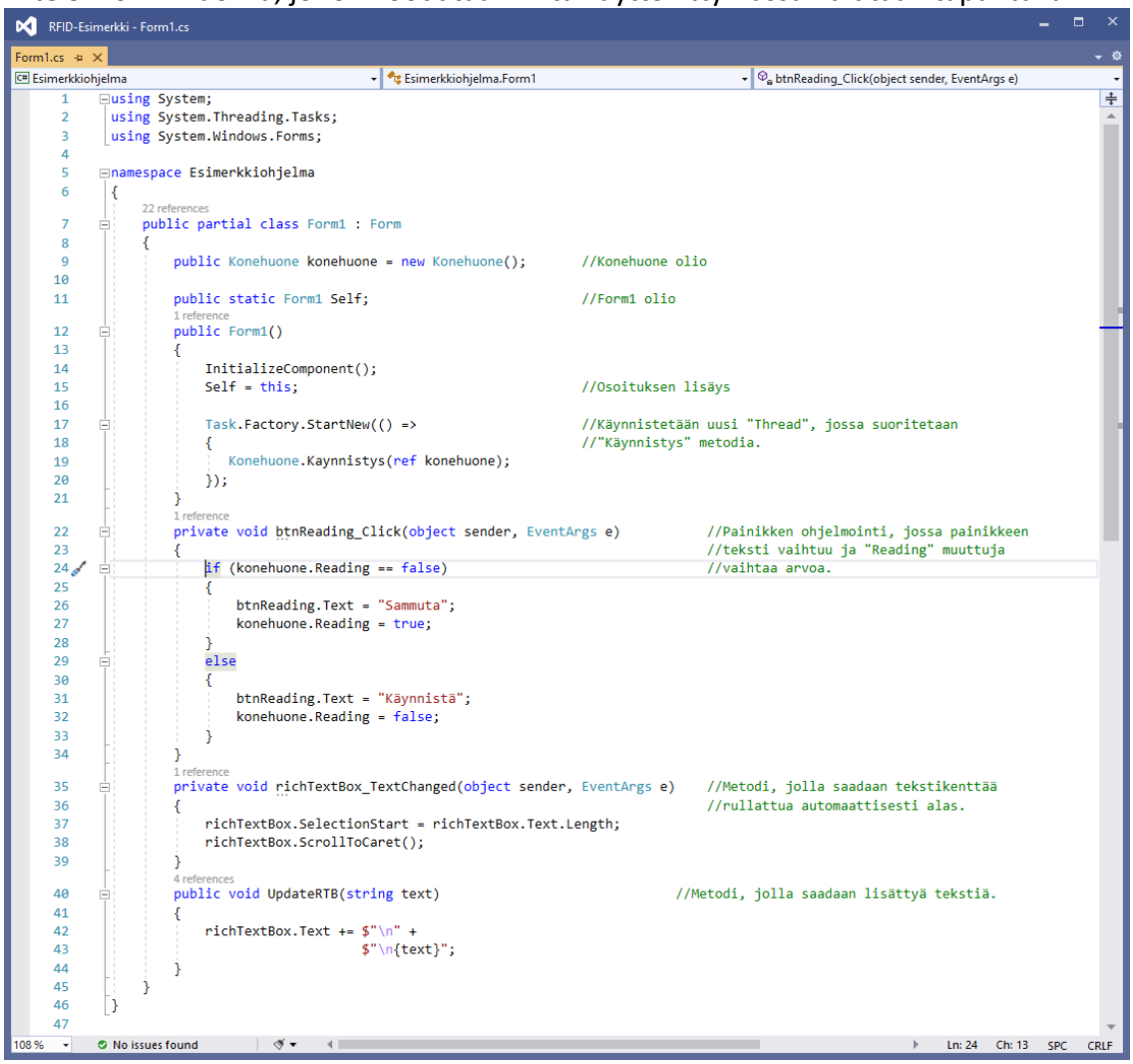


```
using System;
using System.Threading;
using ThingMagic;

namespace ReadAsync
{
    0 references
    class Example
    {
        0 references
        public static void EsimerkkiMetodi()
        {
            try
            {
                //Reader olio luodaan ja sille asetetaan
                //parametriksi lukijan IP osoite
                using (Reader r = Reader.Create("192.168.0.1"))
                {
                    //"Connect" metodi yrittää yhdistää lukijaan
                    r.Connect();

                    //Tämä IF lause etsii ja asettaa toiminta alueen lukijalle
                    if (Reader.Region.UNSPEC == (Reader.Region)r.ParamGet("/reader/region/id"))
                    {
                        Reader.Region[] supportedRegions = (Reader.Region[])r.ParamGet("/reader/region/supportedRegions");
                        if (supportedRegions.Length < 1)
                        {
                            throw new FAULT_INVALID_REGION_Exception();
                        }
                        r.ParamSet("/reader/region/id", supportedRegions[0]);
                    }
                    //Nämä rivit luovat "SimplePlan" olion,
                    //joka asetetaan lukijan "plan" parametriksi.
                    SimpleReadPlan plan = new SimpleReadPlan();
                    r.ParamSet("/reader/read/plan", plan);

                    //Metodi "TagRead" käy toteen,
                    //kun lukija havaitsee tunnisteen.
                    r.TagRead += delegate (Object sender, TagReadDataEventArgs e)
                    {
                        //Tähän koodataan, mitä tehdään kun tunniste havaitaan.
                    };

                    //Tämä metodi käy toteen
                    //kun lukija havaitsee virheen
                    r.ReadException += new EventHandler<ReaderExceptionEventArgs>(r_ReadException);

                    //"StartReading" metodi aloittaa lukemisen
                    r.StartReading();
                    //Lukijan annetaan lukea 10 000 millisekunttia
                    Thread.Sleep(10000);
                    //StopReading metodi pysäyttää lukemisen
                    r.StopReading();
                }
            }
            //CATCH jos lukemisessa sattuu virhe
            catch (ReaderException re) { Console.WriteLine(re.ToString()); }
            //CATCH jos TRY lauseessa sattuu virhe
            catch (Exception ex) { Console.WriteLine(ex.ToString()); }


        }
        1 reference
        private static void r_ReadException(object sender, ReaderExceptionEventArgs e)
        {
            Console.WriteLine("Error: " + e.ReaderException.Message);
        }
    }
}
```

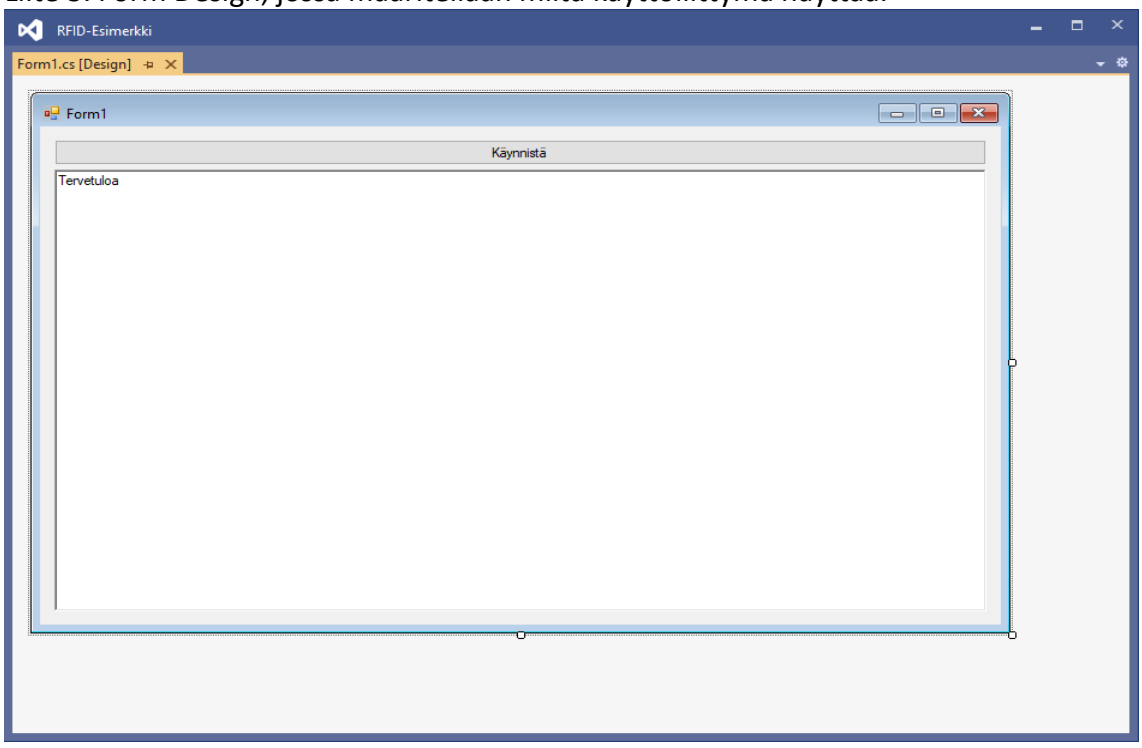
Liite 3. Konehuone luokka, jota on käytetty esimerkki ohjelmassa.


RFID-Esimerkki - Konehuone.cs
Konehuone.cs
Esimerkkiohjelma
Esimerkkiohjelma.Konehuone
reading
using System;
using System.Threading;
using ThingMagic;
//ThingMagic kirjaston lisäys
namespace Esimerkkiohjelma
{
5 references
public class Konehuone
{
private bool reading;
7 references
public bool Reading { get => reading; set => reading = value; }
1 reference
public static void Kaynnistys(ref Konehuone konehuone)
{
Delegate lisaateksti = new update_RTB(Form1.Self.UpdateRTB);
//"While" metodi pitää "Kaynnistys" metodin päällä.
while (true)
{
//"If" lauseella tarkistetaan aloitetaanko lukeminen
if (konehuone.Reading == true)
{
try
{
using (Reader r = Reader.Create("tmr://192.168.1.10"))
{
//Seuraavalla komennolla lisätään tekstiä tekstikenttään,
//ja ilmoitetaan että lukija on yhdistämässä.
Form1.Self.BeginInvoke(lisaateksti, "Reader connecting...");
r.Connect();
if (Reader.Region.UNSPEC == (Reader.Region)r.ParamGet("/reader/region/id"))...
SimpleReadPlan plan = new SimpleReadPlan();
r.ParamSet("/reader/read/plan", plan);
//Komrnnolla lisätään tekstiä tekstikenttään,
//ja ilmoitetaan että lukija on yhdistänyt.
Form1.Self.BeginInvoke(lisaateksti, "Reader connected!");
r.TagRead += delegate (Object sender, TagReadDataEventArgs e)
{
//Komennolla lisätään tekstiä tekstikenttään, jossa on
//löydetyn tunnisteen kaikki data
Form1.Self.BeginInvoke(lisaateksti, $"TagReadData : {e.TagReadData}\n");
};
r.ReadException += new EventHandler<ReaderExceptionEventArgs>(r_ReadException);
r.StartReading();
//Jos "OnOff" muuttuja on päällä, niin lukija jatkaa lukemista, eikä
//siirry "StopReading" metodille.
while (konehuone.Reading == true)
{
Thread.Sleep(1000);
}
r.StopReading();
//--Komrnnolla lisätään tekstiä tekstikenttään,
//--ja ilmoitetaan että lukija on yhdistämässä.
Form1.Self.BeginInvoke(lisaateksti, "Reading stopped");
}
}
catch (ReaderException re) { Form1.Self.BeginInvoke(lisaateksti, $"ReaderException : {re}"); }
catch (Exception ex) { Form1.Self.BeginInvoke(lisaateksti, $"Exception : {ex}"); }
}
else
{
//Jos "Reading" muuttuja on epätosi, Thread jää odottamaan 1 sekunnin ajaksi
//ja katsoo sen uudelleen.
Thread.Sleep(1000);
}
}
}
//Metodi, jolla saadaan näkyviin lukemisessa tapahtuneet virheet
//tekstikenttään.
1 reference
private static void r_ReadException(object sender, ReaderExceptionEventArgs e)
{
Delegate lisaateksti = new update_RTB(Form1.Self.UpdateRTB);
Form1.Self.BeginInvoke(lisaateksti, $"Error: {e.ReaderException.Message}");
}
//Delegate metodi, joka asetetaan delegate muuttujaan.
private delegate void update_RTB(string x);
}
}
108 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF

Liite 3. Form1 luokka, johon koodataan mitä käyttöliittymässä halutaan tapahtuvan.



Liite 3. Form Design, jossa määritellään miltä käyttöliittymä näyttää.



Liite 4. ThingMagic UHF Fixed/Finished Reader Development Kit, josta löytyy lukija, antenni, tarvittavat kaapelit, “Mercury Api” kirjasto, “Universal Reader Assistant” ohjelma, ohjeita sekä vuoden tekninen tuki.

https://www.jadaktech.com/products/thingmagic-rfid/thingmagic-uhf-fixed-finished-reader-development-kit/