VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Petri Tuomio

MODBUS MAYHEM

MODBUS/TCP-ASIAKASSOVELLUS WINDOWS PHONE 7.1 -YMPÄRISTÖÖN

Tekniikka ja liikenne
2013

VAASAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Petri Tuomio
Opinnäytetyön nimi	Modbus/TCP-asiakassovellus Windows Phone 7.1 -ympäristöön
Vuosi	2013
Kieli	Suomi
Sivumäärä	45
Ohjaaja	Pirjo Prosi

Opinnäytetyössä kuvataan Windows Phone–alustalle ja sen käyttöjärjestelmän versiolle 7.1 kehitettävän Modbus/TCP asiakassovelluksen toteutusta. Sovellus mahdollistaa Modbus palvelimen input ja holding rekistereiden sisältämien lukuarvojen lukemisen puhelimen näytölle sekä arvojen muokkaamisen ja kirjoittamisen holding rekistereihin käyttäen TCP-yhteyttä asiakassovelluksen ja palvelimen välillä. Sovellus on tarkoitettu käytettäväksi pienimuotoiseen palvelinlaitteen tietojen selaamiseen ja/tai ohjaukseen.

Sovelluksen ohjelmistokehitysalustana toimii Microsoft Visual Studio 2012 johon on asennettu Windows Phone SDK 7.8. Ohjelmointikielenä on C# ja XAML. Kehitysalustaa käytetään PC:llä jonka käyttöjärjestelmänä on Windows 8. Sovelluksen testaamiseen käytetään emulaattorin lisäksi myös Nokia Lumia 800 sekä 920 matkapuhelinta.

Modbus protokollan määrittely sisältää useita eri ”funktioita”. Opinnäytetyön asiakassovellus on rajattu käyttämään kolmea yleisesti käytettyä komentoa joilla vaadittu toiminnallisuus voidaan toteuttaa.

Tuloksena syntynyt Modbus Mayhem on helppokäyttöinen ja selkeä sovellus. Sen käytön suurimpana rajoitteena on verkkoyhteyden muodostaminen palvelinlaitteelle sillä teollisuusympäristössä - johon protokollansa myötä sovellus kuuluu - WLAN ei vielä (tätä kirjoitettaessa) ole kovin yleinen. Opinnäytetyönä Modbus Mayhem on mitä parhain, suorastaan fantastinen.

Avainsanat Windows Phone, Modbus/TCP, Modbus Mayhem

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikan koulutusohjelma

ABSTRACT

Author	Petri Tuomio
Title	Modbus/TCP Client Application to Windows Phone 7.1 Environment
Year	2013
Language	Finnish
Pages	45
Name of Supervisor	Pirjo Prosi

This thesis describes the implementation of Modbus/TCP client application to Windows Phone 7.1 platform. This application enables reading and writing of holding registers as well as reading of input registers from Modbus/TCP server to visualize and to allow editing of values, using TCP connection between Modbus Mayhem and server. The application itself is meant to be used for small scale data handling and controlling.

Microsoft's Visual Studio 2012 + SP1 with additional Windows Phone SDK 7.8 and SDK 8.0 was used for the application development. Programming languages were C# and XAML. Development platform was run at workstation equipped with MS Windows 8 Pro operating system. Application was tested not only with WP7.8 and WP8.0 emulators but also with Nokia Lumia 800 and 920 mobile phones.

Definition of Modbus protocol contains several different Function Codes. Application described in this thesis is built to utilize three most commonly used functions to achieve the required operational level.

As a result from this thesis, application called Modbus Mayhem was born. It is an easy to use (and learn) application. What comes to usage of Modbus Mayhem, the biggest problem is that the wireless LAN needed is very seldom used at industry (at the moment). And due to Modbus, industry is the native environment for the Modbus Mayhem. But what comes to thesis itself, Modbus Mayhem was not only a fantastic but also an astonishing way to learn how to create Windows Phone applications.

Keywords Windows Phone, Modbus/TCP, Modbus Mayhem

SISÄLLYS

1 JOHDANTO

Sähkötekniikan huima kehitys viimeisen sadan vuoden aikana on poikinut uskomattoman määrän mitä erilaisempia laitteita, joiden käyttötapaa rajoittaa pikemminkin ihmisen mielikuvitus kuin itse tekniikka. Sähkölaitteiden myötä syntyi myös tarve saada laitteet keskustelemaan keskenään eli vaihtamaan tietoa. Sähkömekaaninen puhelinkeskus tai vanhan pyykinpesukoneen mekaaninen logiikka edustavat järjestelmiä ja laitteita, joissa vielä sijaitsi liikkuvia osia ja joissa valintojen ja ohjauksen - siis tiedon - siirto paikasta toiseen laitteen sisällä tapahtui hammasrattailla ja releillä. Tiedon määrän kasvaessa ja varsinkin puolijohteiden kehittyessä oli luonnollista siirtyä kuljettamaan tarvittavaa tietoa sähköisesti *väylällä*. Terminä ”väylä” on kuvaava. Sen synonyymit ”ura”, ”reitti” ja ”tie” kuvaavat mahdollisuutta siirtyä paikasta A paikkaan B, mennä reittiä X paikkaan Y jne /1/. Juuri niin väylää käytetään. Vaikka väyliä on lukematon määrä, niin yhteistä niille kaikille on se, että niillä siirretään tietoa laitteiden tai laitteen osien välillä. Väylät mahdollistavat järjestelmän hajauttamisen.

Väylä itsessään on abstrakti. Jokainen voi kehittää ja toteuttaa oman väyläratkaisunsa. Väylän määrittelyn ei tarvitse puuttua siirtotien rakenteeseen tai rajapintaan. Se mikä lähtöpäässä sujahti RS-232:een, voi tulla ulos vastaanottajalle RS-485:sta. Tietoa voidaan siirtää niin puukepillä kuin puhelinverkon välityksellä.

2 MODBUS

2.1 Modbus-protokolla

Modbus esiteltiin 1979 yhtiön nimeltä Modicon toimesta /2/. Nimen alku ”Mod” viittaakin sanaan Modicon ja ”bus” englanninkieliseen sanaan jolla tarkoitetaan mm. väylää. Modbus yleistyi nopeasti ja on yksi yleisimmin käytetyistä teollisuusväylistä /3/. Modbus-protokolla on hyvin yksinkertainen ja siten helppo implementoida tietojenkäsittelyteholtaan vaatimattomiinkin laitteisiin. Lisäksi protokolla on avoin ja kuka tahansa halukas voi ilman pelkoa lisenssimaksuista tuotteistaa sen. Protokollan ongelmana voidaankin pitää juuri sen yksinkertaisuutta. Ympäristö johon Modbus suunniteltiin, käytti pelkästään sarjaliikennettä (RS-485/232) kommunikointiin ja oli tämän päivän mittakaavassa tiedonsiirtokapasiteetiltaan hyvin rajoittunut.

Modiconin siirryttyä myöhemmin yrityskaupan myötä Schneider Electricin omistukseen, siirrettiin Modbus-protokollan hallinta itsenäiselle käyttäjien ja kehittäjien muodostamalle ryhmittymälle, jolle jäi standardin kehitys ja ylläpito. Ryhmittymä sai nimekseen The Modbus Organization /4/.

Modbussista on useita eri variaatioita. Yksi vanhimmista on Modbus ASCII, jossa väylällä siirretään ASCII-merkistön mukaisia merkkejä, eli kirjaimiston numeerisia koodeja. Tällä tavalla siirrettävät paketit on helpompi erottaa toisistaan vuonohjauksen käytettyjen merkkien avulla, eikä kommunikointi ole niin riippuvaista kellotuksesta. Toinen sarjaliikenteessä käytetty on tyyppi on Modbus RTU, jossa data on binäärimuodossa. Vaikka itse ASCII-merkin esittäminen vie vain 7 bittiä, tarvitaan jokaista RTU:n vastaavaa hyötykuormatavua kohti 15 bittiä, eli lähes kaksinkertainen määrä siirrettäviä bittejä. Molemmissa tapauksissa paketti sisältää tarkistussumman tiedon muuttumattomuuden varmistamiseen. Modbus ASCII:ssa on käytössä kevyempi LRC (Longitudinal Redundancy Check) ja RTU:ssa CRC (Cyclic Redundancy Check).

Modbus-RTU:n paketin hyötykuorman (Protocol Data Units) maksimikoko on 253 tavua jonka rajoittajana toimii standardin määrittämä suurin pakettikoko 256

tavua (Application Data Unit), josta vähennetään 3 tavua ja käytetään niistä yksi orjalaitteen osoitteelle ja loput kaksi paketin 16-bittiselle CRC-tarkistusluvulle /5/. Useat valmistajat ovat muokanneet standardia omiin laitteisiinsa, tarkoituksena milloin lisätä omia komentoja tai siirtää 32- tai 64 bittisiä kokonais- tai liukulukuja. Luonnollisesti tämä rajaa laitteen kommunikointipartnerit saman valmistajan tuotteisiin. Standardi ei varsinaisesti määrittele rekisterin pituutta, mutta de-facto on että perusyksikkö on 16-bittinen muuttuja, ja jos halutaan siirtää 32-bittisiä lukuja väylällä, niin siihen käytetään kahta rekisteriä (16+16 bittiä), 64-bittiseen lukuun neljää jne. Tästäkin on valitettavasti poikkeuksia. Lisäksi on mainittava suljettu mutta Modbus-nimeä kantava Schneider Electricin Modbus+ ja varsin tunnettu Enronin variaatio nimettynä tekijänsä mukaan, eli Enron-Modbus. Termillä ’rekisteri’ viitataan muuttujan osoitteeseen tai tunnukseen. Rekisteri varastoi lukuarvon, joka voidaan lukea tai kirjoittaa. Modbus-paketti sisältää aina orjalaitteen osoitteen ja tarkistussumman lisäksi toimintokoodin (Function Code, FC), joka määrittelee vaaditun toiminnon vastaanottajalle kuten myös paketin data-osuuden, jonka sisältö voidaan tulkita toimintokoodin perusteella.

Modbus-rekisterit on jaettu neljään eri ryhmään joita ovat:

- digitaaliset sisääntulot (Discrete Inputs), vain lukuoikeudet.
- kelat (Coils), luku- ja kirjoitusoikeudet.
- sisääntulorekisterit (Input registers), vain lukuoikeudet.
- pitorekisterit (Holding registers), luku- ja kirjoitusoikeudet.

Eri ryhmien rekistereillä voi olla sama tunnusnumero ja viittauksen käsiteltäviin rekistereihin tarkentaa Modbus-paketin toimintokoodi, joka määrittelee mistä ryhmästä käsiteltävä rekisteri löytyy. Dokumentoinnissa käytetään yleisesti numeerista prefixiä erottamaan rekisteriryhmiä toisistaan seuraavasti:

- ’0’(xxxx) käytetään rekistereiden ryhmälle ’Coils’
- ’1’(xxxx) ryhmälle käytetään rekistereiden ’Discrete Inputs’
- ’3’(xxxx) ryhmälle käytetään rekistereiden ’Input registers’
- ’4’(xxxx) ryhmälle käytetään rekistereiden ’Holding registers’ /7/

Ryhmien nimet ovat jäänne menneisyydestä ja voivat sisältää tiloja tai arvoja mistä tahansa laitteelta. Sen sijaan käsittelyssä ovat vanhat säännöt edelleen voimassa ja ryhmät ’Input registers’ ja ’Discrete Inputs’ ovat väylän kautta vain luettavissa, kun taas ryhmiin ’Coils’ ja ’Holding registers’ onnistuu sekä luku- että kirjoitus isännältä.

Prefixin merkintätavoissa on myös joitakin hankaluuksia. Kun viitataan esim. rekisteriin 30 001 on kyseessä *oletettavasti* Input-rekisteri numero 1. Merkintätavasta seuraa helposti ongelma, joka syntyy siitä että Modbus-standardi ei tunne tällaista yleistynyttä prefix-käytäntöä ja riippuu täysin merkitsijästä kuinka itse merkintä toteutetaan. Ja kun rekistereitä voi olla 65 536 kappaletta (0 – 65 535), niin yleisesti merkintä 30 001 voi tarkoittaa yhtä hyvin joko Input- tai Holding rekisteriä 30 001 jolloin merkintä olisi prefixiä käyttäen Input-rekisterille 330 001 ja Holding-rekisterille 430 001. Rekisterien merkintätapa myös vaihtelee eri sovelluksissa, joten kyse ei ole pelkästä dokumentoinnista vaan mitätön seikka saa varsin suuret mittasuhteet ihmisten tulkitessa määrittelyjä tavallaan.

Omituinen seikka löytyy myös väylälle lähetettävän rekisterin tunnusnumeron käsittelystä. Osa markkinoiden laitteista pienentää käsiteltävän rekisterin numeroa yhdellä lähettäessään luku- tai kirjoituspaketin väylälle. Esim. laite X on määritelty lukemaan arvo orjaltaan rekisteristä 10. Kun orja vastaanottaa kyselyn, onkin Modbus-pakettiin määritelty rekisterin numeroksi 9. Jos orja toimii vastaavalla tavalla kuin isäntänsä, se nostaa vastaanotetun rekisterin arvoa yhdellä tai paremminkin osaa tulkita kyselyn halutulla tavalla, ja palauttaa arvon rekisteristä 10. Jos se ei sitä tee, laite X vastaanottaa arvon rekisteristä 9. Mikä voi olla tietysti haluttuakin sillä onhan mahdollista että laitteen X tuli lukea arvo rekisteristä 9 mutta laitteen rekisterinumeron käsittelytavan takia jouduttiin määrittelemään kysely rekisteriin 10.

Modbus-väylään liitetyt laitteet eivät keskustele vapaasti keskenään vaan kommunikoinnissa on tiukka isäntä-orja hierarkia jossa yksi väylän laitteista suorittaa niin tietojen lukemisen kuin kirjoittamisen toimien näin ”isäntänä” (host). Orjalaitteiden (slave) maksimimääräksi standardi määrittää 247 kpl

osoitealueelta 1-247 /6/. Osoite 0 on varattu yleislähetykselle (broadcast) isännältä johon orjat eivät vastaa.

Modbus käyttää big-endian-rakennetta tiedonsiirrossa eli jos suurempi kuin yhden tavun mittainen numeerinen tieto siirretään paketissa, sen eniten merkitsevä tavu lähetetään ensin. Esimerkkinä 16-bittinen luku 0x1234 ja 32-bittinen luku 0x12345678:

Arvo	Tavu[0]	Tavu[1]	Tavu[2]	Tavu[3]
0x1234	0x12	0x34	-	-
0x12345678	0x12	0x34	0x56	0x78

2.2 Modbus/TCP

Ethernetin yleistyessä teollisuusympäristössä, syntyi tarve laajentaa Modbus-reviiriä sarjaväylistä Ethernetiin. Tämän tuloksena syntyi Modbus/TCP, jossa Modbus RTU-paketin rakenne lähes sellaisenaan on siirretty kulkemaan TCP:n päällä. RTU:n tarkistussumma voitiin poistaa, koska Ethernet-kehys itsessään sisältää tarkistussumman. Lisäksi lisättiin otsikkotaulu jossa sijaitsevat kentät yksilölliselle viestitunnisteelle (Transaction Identifier), protokollatunnisteelle (Protocol IDentifier) sekä paketin koolle (LEN). Lisäyksen koko on yhteensä kuusi tavua joista jokainen edellämainittu osio vie kaksi tavua.

Kuva 1. RTU ja TCP pakettien rakenteet

Selvyyden vuoksi on kuvaan 1 merkitty Modbus RTU orjaosoitteen kenttään tunnus ”UID”, vaikka lyhenne otettiin käyttöön vasta Modbus/TCP:n myötä. RTU:ssa käytetään kentästä termiä ”Slave ID” tai ”Slave address”. UID:n sisältävä kenttä on hyvin yleisesti käytössä ja tavallisesti laitteet tarkistavat sen sisällön saapuvasta paketista vaikka laite sijaitsisi samassa verkossa kuin asiakas (ei siltausta eikä reititystä laitteiden välillä), jolloin IP-osoite jo itsessään riittää tuomaan paketin oikealle laitteelle. Modbus.org suosittaa että jos siltausta tai reititystä ei käytetä, UID:n arvoksi asetetaan 255. Tällä haetaan lisävarmuutta yhteyksiin laitteiden välillä, ajatuksena että jos tapahtuu vahinko että IP osoitteen muuttuessa jonkin muutoksen yhteydessä palvelimen IP siirtyykin sillalle tai reitittimelle, kyseinen laite tuhoaa paketin päästämättä sitä läpi koska paketti sisältää laittoman UID:n eli luvun joka ei kuulu standardi määrittelemällä alueelle 0-247.

Modbus/TCP:n protokollatunniste (PID) on 0. Koska kaikkien siirrettävien pakettien koko jää alle 256 tavun, on myös pituuskentän (LEN) ensimmäisen tavun arvo aina 0. Kuudennessa tavussa ilmoitetaan sitä *seuraavien* tavujen määrä. Koska siirrossa käytetään TCP:tä, voi paketti saapua useammassa kuin yhdessä erässä. Pituuskentän avulla vastaanottaja osaa odottaa mahdollisesti vielä matkalla olevaa, puuttuvaa paketin osaa.

Viestitunnisteen (TI) arvo voi olla mitä vain. Orja/palvelin on velvoitettu kopioimaan saapunut tunniste vastausviestiin. Serverin vastausviesti ei sisällä mitään tietoa siitä, mikä tai mitkä rekisterit ovat saapuvassa paketissa kyseessä. On täysin mahdollista että serverin vastaukset saapuvat eri järjestyksessä kuin kyselyt on lähetetty ja ainoa tapa jolla voidaan varmistaa että saapuva data siirretään oikealle osoitealueelle on verrata kyselyssä käytettyä ja saapuvaa tunnistetta toisiinsa. Standardi ei ota kantaa onko tunnistetta käytettävä sillä arvohan voidaan pitää vakiona, mutta on järjetöntä toimia näin jos useampaa kuin yhtä rekisterialuetta luetaan. Kirjoitus toimii toki hienosti oli tunniste mikä tahansa, mutta silloinkin on mahdollista että serverin virhetilanteessa ilmoittaman poikkeuksen rekisterialue tulkitaan väärin tai jää epäselväksi. Todennäköisyys moiselle on toki pieni mutta ei nolla.

Kuvassa 1 vihreällä esitetty kenttä ’FC’ on yhden tavun mittainen ja sisältää toimintokoodin (Function Code). Modbus.org määrittelee 19 eri koodia joista osalle erikseen mainitaan toiminnon tarkoitetun vain sarjaliikennekäyttöön, mutta mikään ei estä niiden käyttöä myöskään Ethernet-ympäristössä. Lisäksi valmistajat voivat lisätä omia koodejaan laitteilleen tarpeidensa mukaan.

Taulukossa 1 on lueteltu Modbus.org:n sivustolta löytyvästä ”Modbus application protocol specification V1.1b3”-dokumentissa listatut toimintokoodit ja niiden englanninkieliset kuvaukset.

Taulukko 1. Modbus toimintokoodit.

Toimintokoodi	Heksalukuna	Kuvaus
1	0x01	Read Coils
2	0x02	Read Discrete Inputs
3	0x03	Read Holding Registers
4	0x04	Read Input Registers
5	0x05	Write Single Coil
6	0x06	Write Single Register
7*	0x07	Read Exception Status
8*	0x08	Diagnostics
11*	0x0B	Get Comm Event Counter
12*	0x0C	Get Comm Event Log
15	0x0F	Write Multiple Coils
16	0x10	Write Multiple Registers

17*	0x11	Report Server ID
20	0x14	Read File Record
21	0x15	Write File Record
22	0x16	Mask Write Register
23	0x17	Read/Write Multiple Registers
24	0x18	Read FIFO Queue
43	0x2B	Encapsulated Interface Transport

* Määritelty sarjaliikennekäyttöön.

Toimintokoodille varatusta tavusta eniten merkitsevin bitti (0x80) on varattu poikkeuksesta ilmoittamiseen. Kun palvelin vastaanottaa asiakkaan Modbus-paketin, se luonnollisesti toteuttaa toimintokoodin sille määrittelemän tehtävän kuten varastoimaan lähetetyn datan sille osoitettuihin rekistereihin, tai palauttamaan arvot kysytyistä rekistereistä, muista toiminnoista puhumattakaan. Jos palvelin ei esimerkiksi tunnista toimintokoodia tai rekisteri johon paketissa viitataan ei ole käytettävissä, palvelin palauttaa poikkeusilmoituksen (exception). Tämä tapahtuu palauttamalla vastaanotetun toimintokoodin arvon summattuna arvolla 0x80. Toisin sanoen toimintokoodin tavun eniten merkitsevä bitti on asetettu arvoon ”1”. Palvelin palauttaa toimintokoodin lisäksi vielä yhden tavun, joka kertoo tarkemmin poikkeuksen syyn. Täten esimerkiksi poikkeusilmoitus toimintokoodille 0x03 palautuu lähettäjälle arvolla 0x83. Rakenne on esitetty kuvassa 2. Näin paketin lähettäjä saa tiedon että virhe on tapahtunut ja seuraavan tavun sisältämä arvo antaa yksityiskohtaisempaa tietoa itse poikkeuksesta. Modbus.org listaa yhdeksän tunnistekoodia poikkeuksille. Nämä ovat listattuna taulukkoon 2.

Taulukko 2. Poikkeukset englanninkielisine kuvauksineen.

Poikkeuksen numero	Heksalukuna	Kuvaus
1	0x01	Illegal Function
2	0x02	Illegal Data Address
3	0x03	Illegal Data Value
4	0x04	Server Device Failure
5	0x05	Acknowledge
6	0x06	Server Device Busy
8	0x08	Memory Parity Error
10	0x0A	Gateway Path Unavailable
11	0x0B	Gateway Target Device Failed To Respond

Kuva 2. Paketin rakenne, poikkeuksen ilmoitus

Se mitä dataa Modbus-paketti sisältää, kuinka siihen tulee vastata tai kuinka saapunut komento käsitellään, riippuu toimintokoodista ja luonnollisesti palvelimesta itsestään. Kaikkien standardissa määriteltyjen toimintokoodien tukeminen palvelimessa on toki mahdollista mutta harvinaista. Niinpä normaalisti tuella toimintokoodeille 1, 2, 3, 4, 5 ja 16 voidaan kattaa valtaosa markkinoiden laitteista, jonka lisäksi toimintokoodit 1, 2, 5 yleensä voidaan korvata toimintokoodeilla 3 ja 16 kun rekisterien arvoja käsitellään bittimaskeina, eli näennäisesti eri rekisterialueita käsittelevät komennot ohjataan serverissä samoille

rekistereille. Toimintokoodi 4 on myös yleisesti käytetty jos rekistereille halutaan antaa vain lukuoikeus.

Tämän opinnäytetyön aiheena oleva Windows Phone–sovellus käyttää vain toimintokoodeja 3, 4 ja 16. Seuraavaksi käsitellään Modbus-paketin sisältö näitä toimintokoodeja käytettäessä.

2.3 FC3, FC4 ja FC16

Opinnäytetyön sovellus on nimeltään Modbus Mayhem ja se on suunniteltu käyttämään kolmea toimintokoodia. Näistä ”Read multiple registers” (0x03) ja ”Write multiple registers” (0x10) ovat yleiskäyttöisinä ja myös erittäin yleisesti käytössä. ”Read input registers” (0x04) on myös varsin yleinen. Input- ja holding-rekisterit ovat tyypiltään analogisia mutta holding-rekistereitä käytetään laajasti myös yksittäisten binääritietojen siirtoon, käyttäen analogiarvon sijaan rekisterin jokaista bittiä yksittäisenä tosi-epätosi -tietona. Rakenteeltaan input- ja holding-rekistereiden lukuun käytettävät Modbus-paketit ovat lähes identtiset, vain toimintokoodin numero on eri. Se riittää ohjaamaan kyselyn eri rekisteriosoitteeseen palvelimella, kuten on jo aikaisemmin mainittu.

Tavu[0] : Viestitunnisteen eniten merkitsevä tavu

Tavu[1] : Viestitunnisteen vähiten merkitsevä tavu

Tavu[2] : Protokollatunnisteen eniten merkitsevä tavu

Tavu[3] : Protokollatunnisteen vähiten merkitsevä tavu

Tavu[4] : Paketin pituus, eniten merkitsevä tavu

Tavu[5] : Paketin pituus, vähiten merkitsevä tavu

Tavu[6] : UID

Koska Modbus-paketin otsikko-osuus + UID on aina rakenteeltaan samanlainen riippumatta toimintokoodista, seuraavien toimintokoodien kuvaus alkaa ensimmäisestä UID:n jälkeisestä tavusta. Koska Modbus/TCP on Ethernet-protokolla, käytetään jatkossa ”isäntä” ja ”orja” termien sijaan tutumpia termejä ”asiakas” ja ”palvelin”.

2.3.1 Read multiple registers

Kuva 3. Toimintokoodi 3

Tavu[7] : Toimintokoodi (3)

Tavu[8] : Rekisterin numero, eniten merkitsevä tavu

Tavu[9] : Rekisterin numero, vähiten merkitsevä tavu

Tavu[10] : Luettavien rekistereiden määrä, eniten merkitsevä tavu

Tavu[11] : Luettavien rekistereiden määrä, vähiten merkitsevä tavu

Kuva 3 esittää toimintokoodin 3 sisältämän Modbus-paketin. Toimintokoodin perusteella paketin vastaanottanut serveri tietää, että asiakas haluaa lukea arvoja holding-rekistereistä. Tavut 8 ja 9 sisältävät tiedon mikä on ensimmäinen rekisterin numero josta lukeminen aloitetaan. Koska suurin luettavien rekistereiden määrä on 125, on tavussa 10 aina arvo 0. Tavu 11 siis yksin määrittää monenko rekisterin sisällön palvelin lähettää asiakkaalle.

Palvelin vastaa kyselyyn palauttamalla otsikkotaulun ja UID:n lisäksi toimintokoodin sekä ilmoittamalla vastauksen dataosuuden sisältämän tavumäärän, sekä pyydettyjen rekisterien sisällön. Paluuviestin rakenne on esitetty kuvassa 4.

Tavu[7] : Toimintokoodi (3)

Tavu[8] : *Seuraavien* tavujen määrä (=2 x rekistereiden määrä)

Tavu[9 =>] : Rekisterien sisältö

Kuva 4. Vastaus

Alla esimerkkinä kysely jossa luetaan kolmen rekisterin tiedot alkaen osoitteesta 1 laitteelta jonka UID on 5. Rekistereissä ovat arvot 10, 100 ja 1 000.

Kysely: 42 30 00 00 00 06 05 03 00 01 00 03

Vastaus: 42 30 00 00 00 09 05 03 06 00 0A 00 64 03 E8

Jos kysely suoritetaan rekisteriin jota laitteessa ei ole tai laite ei tunnista toimintokoodia ”3”, poikkeus ilmoitetaan toimintokoodilla 0x83 (0x80 + 0x03) ja sitä seuraavassa tavussa tarkemmin arvoilla 1 tai 2, eli joko ”laiton toimintokoodi” tai ”laiton rekisterinumero”.

2.3.2 Read input registers

Toimintokoodia 4 käytettäessä Modbus paketin rakenne niin kyselyssä kuin vastauksessa on täysin samanlainen kuin käytettäessä toimintokoodia 3, ainoana erona on luonnollisesti eri toimintokoodi, joka ohjaa käsittelyn input-rekistereihin. Myös poikkeukset ovat identtiset, lukuunottamatta palautettavaa toimintokoodia joka on 0x84 (0x80 + 0x04).

2.3.3 Write multiple registers

Kuva 5. Toimintokoodi 16

Tavu[7] : Toimintokoodi (16)

Tavu[8] : Rekisterin numero, eniten merkitsevä tavu

Tavu[9] : Rekisterin numero, vähiten merkitsevä tavu

Tavu[10] : Kirjoitettavien rekistereiden määrä, eniten merkitsevä tavu

Tavu[11] : Kirjoitettavien rekistereiden määrä, vähiten merkitsevä tavu

Tavu[12] : *Seuraavien* tavujen määrä (=2 x rekistereiden määrä)

Tavu[13 =>] : Rekistereihin kirjoitettava sisältö

Kuvan 5 esittämän toimintokoodin 16 perusteella paketin vastaanottanut serveri tietää, että asiakas haluaa kirjoittaa arvoja holding-rekistereihin. Tavut 8 ja 9 sisältävät tiedon mikä on ensimmäinen rekisterin numero, josta kirjoittaminen aloitetaan. Koska suurin kirjoitettavien rekistereiden määrä on 100, on tavussa 10 aina arvo 0. Tavu 11 siis yksin määrittää monenko rekisterin datan serveri on vastaanottanut asiakkaalta. Palvelin siirtää vastaanotetun sisällön rekistereihin ja kuittaa vastaanoton lähettämällä toimintokoodin, alkurekisterin numeron ja montako rekisteriä tuli kirjoitettua, joka on kopio tavuissa 10 ja 11 tulleesta arvosta.

Tavu[7] : Toimintokoodi (16)

Tavu[8] : Rekisterin numero, eniten merkitsevä tavu

Tavu[9] : Rekisterin numero, vähiten merkitsevä tavu

Tavu[10] : Kirjoitettujen rekistereiden määrä, eniten merkitsevä tavu

Tavu[11] : Kirjoitettujen rekistereiden määrä, vähiten merkitsevä tavu

Jos kirjoitus yritetään suorittaa rekisteriin jota laitteessa ei ole tai laite ei tunnista toimintokoodia ”16”, poikkeus ilmoitetaan toimintokoodilla 0x90 (0x80 + 0x10) ja sitä seuraavassa tavussa tarkemmin arvoilla 1 tai 2, eli joko ”laiton toimintokoodi” tai ”laiton rekisterinumero”.

3 SOVELLUS

3.1 Modbus Mayhem

Modbus Mayhemin (referoitu tästä eteenpäin ”MM”) kehitysympäristönä toimi Microsoft Visual Studio 2012 + SP1. Kehitysympäristöön oli asennettu lisäksi Windows Phone 8.0 SDK sekä Windows Phone SDK 7.8. Jälkimmäinen syystä että MM on tarkoitettu käytettäväksi myös Windows Phone 7.1 –käyttöjärjestelmässä, joka tunnetaan ehkä paremmin versionumerolla 7.5 tai nimellä ”Mango”. Kehitykseen käytetyn työaseman käyttöjärjestelmänä toimi Windows 8 Professional.

Windows Phone 7.1 -käyttöjärjestelmän sovelluksia voidaan tehdä joko Silverlight- tai XNA Framework ympäristöihin. Silverlight on lähempänä perinteistä, tapahtumapohjaista Windows-ohjelmointia, kun taas XNA on suunnattu peliohjelmointiin jatkuvan, keskeytymättömän ohjelmakiertonsa ja vuorottelevan laske-piirrä-laske-piirrä –ohjelmasuorituksensa mukaisesti. Luonnollisesti mikään ei estä käyttämästä ympäristöjä mielensä mukaan mutta projektin aloitusvaiheessa on hyvä tietää mitä haluttu ympäristö sillä näistä kahdesta valitaan ohjelman tyyppi uutta projektia luotaessa. XNA on tosin väistyvä ohjelmointiympäristö ja sen työkalujen kehitys on lopetettu Microsoftin toimesta/8/. WP 8.0 kuitenkin sisältää tuen XNA:lle /9/, mutta virallisesti kyse on vain taaksepäin yhteensopivuuden turvaamisesta.

MM:n tapauksessa Silverlight oli luonteva ratkaisu, sillä sovelluksen käyttö on tapahtumapohjaista sisältäen runsaasti valintoja, parametreja ja ikkunoita.

Ohjelmointikielenä toimivat XAML ja C#.

3.2 Vaatimusten määrittely

Koska sovelluksen käyttö tapahtuu puhelimen näyttöä koskettelemalla, on valintapainikkeiden sijoittelulle varattava tilaa. Käyttöliittymän tulee myös

luonnollisesti olla mahdollisimman helppokäyttöinen. Tummien värien käyttö on suositeltavaa akkukeston kannalta, sillä osassa puhelimista on käytössä OLED-näyttö jonka virrankulutus on suoraan verrannollinen näytön kirkkauteen. Ennen työn aloittamista on hyvä listata mitä ohjelman haluaa tekevän, mitä voi jättää pois, ja mitä ominaisuuksia siinä on ehdottomasti oltava. Seuraava listaus selvittää vaatimuksia joita MM:lle asetettiin, sekä selitykset miksi kyseinen ominaisuus sisällytettiin sovellukseen.

1. *Rekistereitä tulee olla mahdollista lisätä poistaa sovelluksen ajon aikana*
   Yhteyden purkaminen ja uudelleenaktivointi jotta rekistereitä voidaan lisätä tai poistaa, on yksinkertaisesti ikävää, ylimääräistä puuhaa.

2. *Rekistereille tulee olla mahdollista määritellä toiminto niitä luodessa*
   Helpottaa käyttöä ja konfigurointi tehdään kerralla loppuun asti.

3. *Rekistereiden toiminnallisuutta tulee olla mahdollista muuttaa sovelluksen ajon aikana*
   Ks. kohta 1.

4. *Rekisterin tyypin tulee ilmetä niin että, input- ja holding rekisterit eivät sekoitu vahingossa*
   Koska rekistereillä voi olla sama numero, on oltava visuaalisesti selkeä tapa kertoa käyttäjälle esitettävän rekisterin tyyppi. MM käyttää eri värejä rekisterin tyypin mukaan.

5. *Jokaisella rekisterillä tulee olla mahdollisuus yksilölliseen kommenttiin*
   Pelkän rekisterin numeron sijaan kuvaava teksti esimerkiksi ”Jännite, vaihe 1” on huomattavasti käyttöä helpottavaa.

6. *Yhdellä silmäyksellä tulee selvitä rekisterin numero ja arvo ja kuinka rekisteriä käytetään (luku/kirjoitus/passiivinen)*
   Rekisterin käytöstä kerrotaan symbolilla. Samasta kehyksestä löytyvät kaikki edellä mainitut tiedot.

7. *On oltava mahdollisuus yksittäiseen, asynkroniseen arvon lukuun tai kirjoittamiseen*

   Jatkuvan kirjoituksen tai luvun sijaan voidaan ”liipaista” haluttaessa kirjoitus tai luku haluttuun rekisteriin. Rekisteriä, jonka luku tai kirjoitus ei ole aktivoitu mutta joka on olemassa, kutsutaan MM:ssa *passiiviseksi*.

8. *Käytettävät rekisterit on pystyttävä tallentamaan omina ryhminään jotta ne ovat käytettävissä ilman uusintakonfiguroimista*

   Käytettävyyden kannalta on olennaista, että rekisterien määrittelyä ei tarvitse tehdä aina uudelleen. MM antaa mahdollisuuden tallentaa rekisterit arvoineen ja kommentteineen sekä yleiset asetellut parametrit ns. *profiiliin*. Profiilien määrää ei ole rajoitettu.

9. *Mahdollinen palvelimen ilmoittama luku- tai kirjoitusvirhe on välitettävä käyttäjälle*

   Serveriltä vastaanotettu poikkeus esitetään huutomerkki-symbolilla rekisterin yhteydessä.

10. *Mahdollisuus säätää pakettien lähetysten välistä aikaa*

    MM tukee kolmea eri lähetystiheyttä. Päivitysnopeudella on merkitystä jos rekistereitä luetaan hitaan yhteyden yli esim. jos käytössä on Modbus/TCP <-> Modbus RTU –silta.

11. *Mahdollisuus säätää paketin sisältämien rekisterien määrää*

    Toisinaan serverin resurssit ovat niin rajoitetut, että luettavien ja kirjoitettavien rekisterien määrää on rajoitettava.

12. *Tuki vapaalle rekisterivalinnalla*

    Tällä tarkoitetaan sitä, että sovelluksen tulee mahdollistaa mistä tahansa rekisteriavaruudesta valittavan rekisterin käyttö ilman rajoitteita rekisterien sijainnille toistensa suhteen.

13. *Offset*

    Mahdollisuus pienentää väylälle kirjoitettavan toimintokoodin rekisterin arvoa yhdellä yhteensopivuuden parantamiseksi.

14. *Tuki muuttujatyypeille uint ja int*

    Tapauksesta riippuen serveriltä saatavan rekisterin arvo voi olla etumerkillinen tai etumerkitön. Käyttäjän on voitava valita palautettavan arvon muuttujatyyppi.

Seuraava listaus kertoo mitkä ominaisuudet olisivat olleet hyödyllisiä mutta jotka rajattiin pois:

1. *Tuki 32-bittisille luvuille*

   Niin hyödyllistä kuin 16-bittisten rekisterien yhdistely 32-bittisiksi kokonais- tai liukuluvuiksi olisikin, projektin koon rajoittamiseksi 32-bittisten lukujen käsittely jätettiin pois.

2. *Tuki useammalle toimintokoodille*

   MM:n tukemat toimintokoodit katsottiin projektille riittäväksi.

3. *Tarkempi ilmoitus vastaanotetun poikkeuksen syystä*

   Poikkeuksen varsinaisen syyn diagnosointiin voidaan käyttää muita sovelluksia. Ilmoitus rekisterin käsittelyn epäonnistumisesta riittää.

4. *Heksaluvut*

   Rekisterin arvo: Binääritilojen luku helpottuu huomattavasti heksadesimaaleja käytettäessä.

   Rekisterin numero: Harvemmin, mutta joskus kuitenkin, rekisterien numerot esitetään dokumentoinnissa heksamuodossa. Tämä kenties lisätään tulevaisuudessa.

5. *Bittimaskit, bittikohtainen muokkaus ja luku*
   Rekisterin arvon bittikohtainen muokkaus on joissakin tapauksissa tarpeen. Ominaisuus jätettiin pois projektin koon rajoittamiseksi.

6. *Tarkempi ilmoitus mahdollisista vastaanotetun paketin virheistä, kuten viallinen otsikkotaulu tai väärä viestitunnus*
   MM tarkistaa vastaanotetun paketin otsikkokentän rakenteen sekä vertaa viestitunnusta lähetettyyn. Jos tunnusta ei löydy tai otsikon rakenne ei ole standardin mukainen, paketti hylätään. MM ei kuitenkaan ilmoita käyttäjälle havainneensa viallisen paketin. Koska tarkempi ongelmadiagnosointi vaatisi lokin ja tietojen keräämisen tai vähintäänkin ilmoituksen, ominaisuus jätettiin pois projektin koon rajoittamiseksi.

3.3 Toiminnallisuus

MM kykenee käsittelemään yhteensä 131 070 rekisteriä (65 535 input-rekisteriä sekä 65 535 holding-rekisteriä), joista jokainen voidaan valita erikseen käyttöön muista rekistereistä riippumatta. Tiedonsiirron ollessa aktiivinen, MM kykenee muokkaamaan Modbus viestit käyttäjän toimien mukaan niin, että esimerkiksi yhteen Modbus pakettiin mahtunut rekisterien sarja pilkotaan useammaksi paketiksi jos yhden tai useamman sarjan jäsenen toiminnallisuus muuttuu, tai jos rekistereitä lisätään tai poistetaan yhteyden aikana. Eli jos luetaan esim. rekistereitä 1, 2 ja 3, voidaan luku suorittaa yhdellä Modbus-paketilla. Mutta jos rekisterin 2 käsittely muutetaankin kirjoitukseksi, MM pilkkoo osoitealueen käsittelyn kolmeen Modbus-pakettiin, jotka ovat luku 1, kirjoitus 2 ja luku 3.

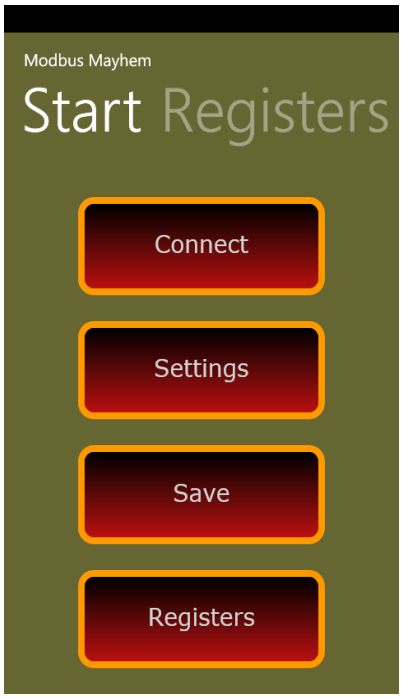
Käytännössä sovelluksessa yhtä aikaa käytettävien rekisterien määrä on varsin pieni, kymmeniä, korkeintaan satoja palvelinta kohti. Käyttöliittymää ei ole suunniteltu suurten rekisterimäärien käsittelyyn, mutta suurtenkaan rekisterimäärien valintaa ei ole haluttu keinotekoisesti estää. Ongelmaksi nousee käytettävän alustan varsin vaatimaton suorituskyky suuria tietomääriä

käsiteltäessä. Kymmenien tuhansien rekistereiden luominen, tallentaminen ja lukeminen muistista pysäyttää sovelluksen käyttöliittymän vasteen pitkäksi aikaa. Tämä johtuu siitä, että MM ei luo omia säikeitä tietojen käsittelyyn vaan ohjelman ’moottori’ ja käyttöliittymä ovat samassa säikeessä. Niinpä jokaisessa ohjelman silmukassa käytetty aika on suoraan pois ohjelman vasteelta. Niillä rekisterimäärillä joita MM:ssa on tarkoitettu käytettävän, tämä ei kuitenkaan ole ongelma ja kuulostaa paljon pahemmalta kuin mitä todellisuudessa on.

4 KÄYTTÖLIITTYMÄ

4.1 Päänäkymä

Käyttöliittymä koostuu päänäkymästä sekä alisivuista, joissa määritellään ohjelman parametreja sekä lisätään ja poistetaan rekistereitä tai muutetaan niiden määrittelyitä.



Kuva 6. Aloitusnäkymä

Sovelluksen päänäkymä on kaksiosainen pivot-tyyppinen sivu, eli sivua voidaan vierittää molempiin suuntiin siirtyen näin sivulta toiselle. Sovelluksen käynnistyessä sivun ”Start”-osio tulee näkyviin. Tämä sisältää neljä toimintopainiketta joilla sovellusta hallitaan; ”Connect”, ”Settings”, ”Save” ja ”Registers”.

4.2 Connect

Tämä painike avaa valintaikkunan jossa määritellään palvelimen IP-osoite sekä käytettävä portti. Oletuksena on portti numero 502, joka on varattu Modbus/TCP palvelimen oletusportiksi /10/.

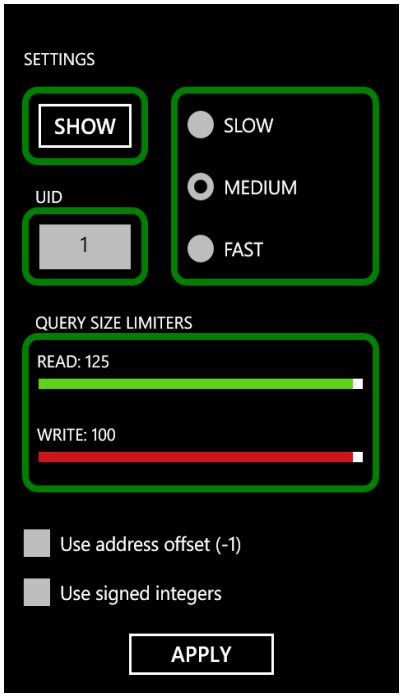
Kuva 7. Connect-sivu

Sivulta löytyy myös mahdollisuus valita tallennettu ”profiili”. Profiili on tallennus joka sisältää käyttäjän määrittelemät rekisterit ja asetukset. Profiilin avulla käyttäjä voi helposti valita esiasetellun sisällön sovellukseen, eikä jokaista käyttökertaa varten tarvitse erikseen valita rekistereitä, yhteysmäärittelyitä tai muita parametreja. Profiili luodaan pääikkunan painikkeella ”Save” josta lisää kohdassa 3.3. Kun listasta on valittuna profiili myös painikkeet ”Activate” sekä ”Remove” ovat käytettävissä. Painettaessa tällöin painiketta ”Activate”, MM lataa tallennetut rekisterit sisältöineen ja esiasettaa IP-osoitteen sekä määritellyn portin. Valittaessa ”Remove”, valittu profiili poistetaan. Tallennettavien profiilien määrää ei ole rajoitettu. Luonnollisesti vain yksi profiili kerrallaan voi olla valittuna ja aktivoituna.

Valittaessa ”Connect”, MM luo yhteyden määriteltyyn IP-osoitteeseen ja porttiin. Aktiivinen yhteys puretaan valitsemalla ”Disconnect”.

4.3 Settings

Tämä painike avaa valintaikkunan jossa määritellään Modbus-paketin rajoituksia ja parametreja sekä määritellään kyselynopeus ja rekistereissä käytetty muuttujatyyppi.



Kuva 8. Yleisasetukset

Kenttä ”UID” hyväksyy arvot 0 – 255 käytettäväksi Modbus viestin UID-osiossa. Modbus-pakettien lähetysnopeutta voidaan säätää valitsemalla yksi kolmesta parametrista; Slow, Medium tai Fast, joista Slow vastaa ~1500 ms viivettä kyselyjen välillä, Medium ~750 ms ja Fast ~200 ms. Kentässä ”Query size limiters” käyttäjä voi asettaa montako rekisteriä määritetään Modbus paketissa maksimissaan käsiteltävän, luku- ja kirjoitusoperaatiossa erikseen. Jos ”Use address offset (-1)” on valittuna, väylälle lähetettävää Modbus-referenssiä vähennetään yhdellä. Esim. luettaessa rekisteriä 100, näkyy väylällä arvo 99 jne.

Tämän takia MM:ssa ei ole rekisteriä 0 (nolla) sillä tämä valinta aiheuttaisi virheen offset aktivoituna referenssin ollessa -1. Kun ”Use signed integers” on valittuna, rekisterin arvo ilmoitetaan välillä -32 767 - 32 768 ja eniten merkitsevä bitti käytetään etumerkin osoittamiseen, eli muuttujatyyppi on 16-bittinen etumerkillinen kokonaisluku. Ilman valintaa käytössä on 16-bittinen etumerkitön kokonaisluku eli muuttujan arvo on välillä 0 - 65 535.

Painamalla ”Show”-kentästä painiketta ”About” (kentän otsikkoa ei kuvassa 8 näy) avautuu ikkuna jossa näkyy MM:n versionumero ja josta löytyy linkki sivulle josta löytyy sovelluksen manuaali, tai paremminkin pikaopas pdf-formaatissa.

4.4 Save

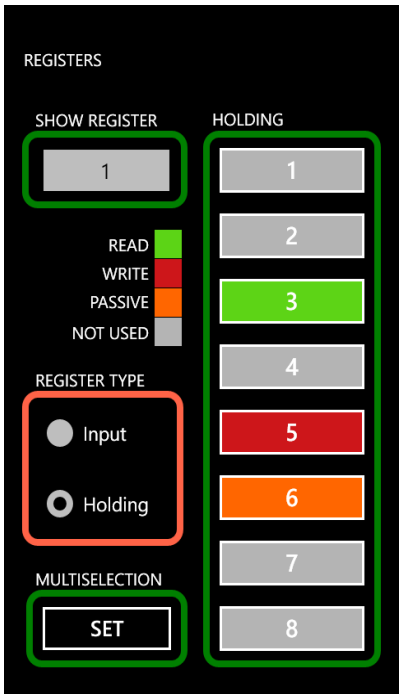
Save-painike avaa kyselyikkunan jossa pyydetään syöttämään nimi tallennettavalle profiilille (kuva 9).

Kuva 9. Tallennettavan profiilin nimen syöttö

Profiilin nimeämisen jälkeen painettaessa ”OK”, tallentuu valitulla nimellä sovelluksen Settings-sivulla määritellyt asetukset sekä valitut rekisterit arvoineen ja kommentteineen. Jos samalla nimellä on jo olemassa profiili, varmistetaan käyttäjältä haluaako hän ylikirjoittaa jo olemassaolevan profiilin. Huomionarvoista on että profiileja - siis niiden sisältämiä rekistereitä ja parametreja - voidaan käsitellä ja tallentaa yhteydettömässä tilassa. On siis mahdollista ”esiladata” rekisterit halutuilla arvoilla jotka yhteyden luomisen jälkeen automaattisesti siirtyvät palvelimelle jos kirjoitus on aktivoitu, tai vain valmistella ryhmä rekistereitä luettavaksi yhteydenoton jälkeen.

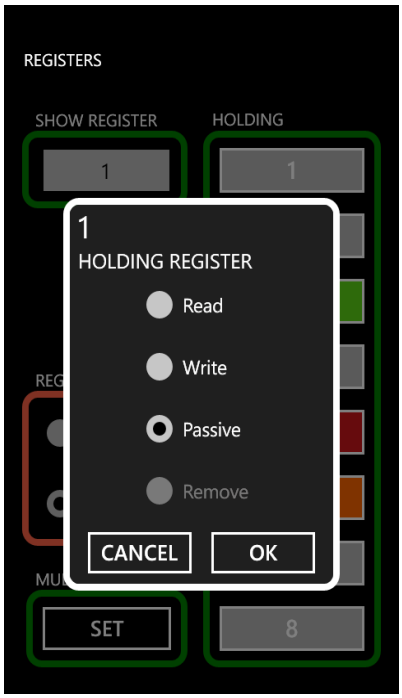
4.5 Registers

Registers-painike avaa valintaikkunan, jossa määritetään käytettävät rekisterit sekä niiden tyyppi ja toiminto.



Kuva 10. Rekistereiden valintaikkuna

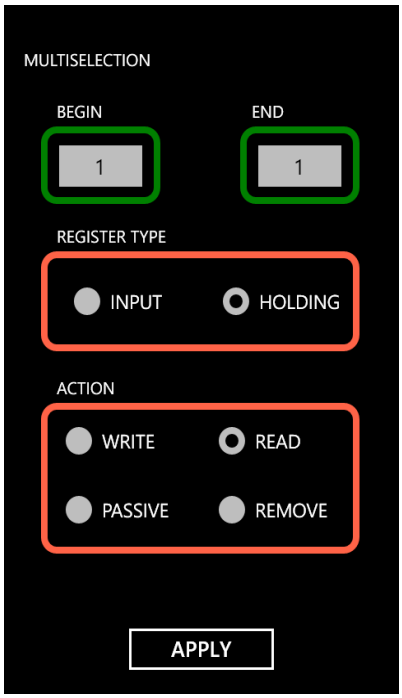
Kentässä ”Register Type” valitaan kentässä ”Input”/”Holding” (nimi vaihtuu valinnan mukaan) näkyvät rekisterit. Kuvan 10 oikeanpuoleisessa listassa näkyvät nelikulmiot ovat painikkeita, joita painamalla avautuu valintaikkuna jossa valitulle rekisterille voidaan määritellä tarkempi toiminnallisuus. Valinnan mukaan muuttuva väri kertoo selvästi kuinka rekisteri on määritelty käytettäväksi ilman, että sen parametreja tarvitsee tarkastella valintaikkunaa avaamalla.



Kuva 11. Toimintovalinta

Rekisterille voidaan määrittää kuvan 11 esittämällä, yksinkertaisella valinnalla kuinka sitä tullaan käsittelemään. Joko luetaan, kirjoitetaan tai rekisteri otetaan käyttöön niin, että siihen voidaan kirjoittaa arvo mutta sitä ei *jatkuvasti* lueta eikä kirjoiteta. Jos valittu rekisteri on jo olemassa, on myös mahdollista muuttaa sen valittua toiminnallisuutta sekä poistaa rekisteri. On huomattava että kun MM:n yhteydessä käytetään termiä ”rekisteri”, tarkoitetaan rekisterioliota, joka sisältää rekisterin numeron, tyypin, arvon, mitä sillä tehdään sekä kommentin. Kyse on siis oliosta, joka on enemmän kuin pelkkä Modbus-rekisteri.

Koska yksittäisen rekisterin konfigurointi on auttamattoman hidasta, suuria rekisterijoukkoja käsiteltäessä on mahdollista siirtyä Multiselection-kentän sisältämän SET-painikkeen kautta Multiselection-sivulle, jossa voidaan syöttää rekisterijoukon alku- ja loppuosoite sekä määritellä mitä asetuksia rekisterille halutaan. Vastaavasti kuten yksittäisellekin rekisterille.



Kuva 12. Ryhmävalintaikkuna

Jos valittuun ryhmään kuuluu jo olemassaoleva rekisteri, se saa valitun toiminnon tai se poistetaan valinnan mukaan. Jos rekisteriä ei ole se luodaan ja sen toiminto asetetaan valittua vastaavaksi. Käyttäjän valinnat astuvat voimaan vasta kun APPLY-painiketta on painettu. Sama koskee muitakin sivuja joille APPLY-painike on sijoitettu.

4.6 Rekisterinäkymä

Rekisterivalinnat näkyvät pääikkunan rekisteriosiossa, johon valitut rekisterit ilmestyvät. Tämä on lista jossa ylimpänä sijaitsevat Input-rekisterit. Määritellyt holding-rekisterit sijaitsevat heti niiden alapuolella. Lisäksi rekisterit ovat omissa, tyyppinsä mukaisissa ryhmissään rekisterin numeron perusteella ja suuruusjärjestyksessä. Input-rekistereiden kehys on sininen ja holding-rekistereiden kehys on vaaleanharmaa. Näin listan eri rekisterityypit voidaan helposti erottaa toisistaan silmämääräisesti.

Kuva 13. Kehysvärit

Kuvassa 13 on ylimpänä Input-rekisteri numero 2 559, jonka alapuolella Holding-rekisteri numero 4.

Rekisterit voidaan myös yksilöllisesti kommentoida. Tämä helpottaa käytettävyyttä huomattavasti jos käytössä on useita rekistereitä.

Kuva 14. Rekisterilista

Riippuen rekisterin konfiguraatiosta, sille annetaan myös symboli joka kuvaa sille valittua toimintoa. Vihreä nuoli tarkoittaa arvon lukua, punainen nuoli taas kirjoitusta ja oranssi ”X” kertoo rekisterin määritetyn passiiviseksi. Rekisterin sisältöön ja sen parametreihin päästään käsiksi painamalla listassa rekisteriä kuvaavaa neliötä. Neliötä, joka on tavallinen painonappi. Rekistereiden luku tai kirjoitus ei pysähdy vaikka valintaikkuna aktivoidaankin, kuten ei myöskään

mikäli avataan mikä tahansa parametrointi-ikkuna tai niiden alisivu, vaan kommunikointi jatkuu taustalla. Rekisteriin voidaan syöttää uusi arvo kentän ”SET VALUE” kautta. Jos kenttä jätetään tyhjäksi ja painetaan OK-painiketta, rekisterin arvo säilyy koskemattomana. Muussa tapauksessa kenttään kirjoitettu arvo siirretään rekisteriin.

Kuva 15. Rekisterin valintaikkuna

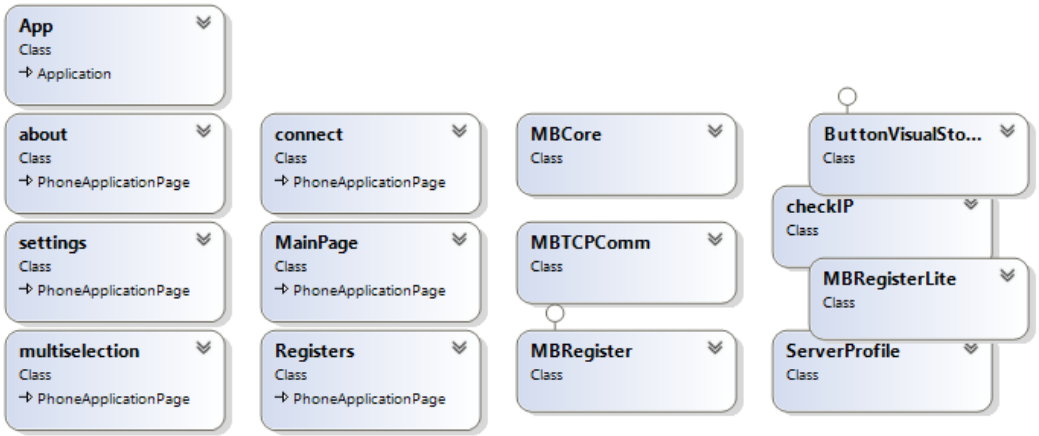
”Commit”-osiossa on kaksi valintaa; ”Read” eli luku ja ”Write” eli kirjoitus. Kumpi tahansa valinnoista on aktivoituna ja painetaan OK, suorittaa MM välittömästi kyseiselle rekisterille valinnan määrittelemän toiminnon. Toiminto saa korkeamman prioriteetin kuin ”normaali” kommunikointi, joka jatkuu normaalisti sen jälkeen kun valittu luku tai kirjoitus on suoritettu. Commit-valinnoilla on mahdollista myös käskyttää passiivista rekisteriä lukemaan tai kirjoittamaan. Tämä onkin tapa jolla passiivista rekisteriä on ajateltu käytettävän. Valinnat toimivat myös rekistereille joiden luku tai kirjoitus on aktivoituna. Käyttö yhteydettömässä tilassa ”varastoi” yksittäisiä luku- ja kirjoitusoperaatioita jotka suoritetaan yhteyden synnyttyä. Jos samalle rekisterille suoritetaan yhteydettömässä tilassa useampi valinta, vimeksi tehty valinta kumoaa aina edellisen.

5 OHJELMAN RAKENNE

5.1 Koodi ja luokat

Suoritettaessa ”Calculate Code Metrics”-toiminto Visual Studiossa MM:lle koodianalyysi ilmoittaa koodirivien määräksi 1 259 kpl (Versio 1.0.3). Varsin vähäinen koodimäärä selittyy pitkälti XAML-koodin osuudesta käyttöliittymän hallintaan. Tuo XML-pohjainen käyttöliittymän rakentamiseen käytettävä kuvauskieli automatisoi paljon toimintoja, jolloin esimerkiksi komponenttien sisällön suoran ohjaamisen sijaan määritellään komponenttien sisällön lähde. Tällä tavalla kuvataan lähteen elementtien sisältöä sen sijaan, että varsinaista ohjelmointikieltä käytettäisiin käyttöliittymän hallintaan ja tiedon päivittämiseen. Toki käyttöliittymän toiminnallisuus on mahdollista määritellä ja suorittaa myös perinteisesti koodaamalla.

MM koostuu yhteensä 14 luokasta, joista luokka App, joka on sovelluksen kantaluokka, ja MainPage syntyvät automaattisesti projektia luotaessa.

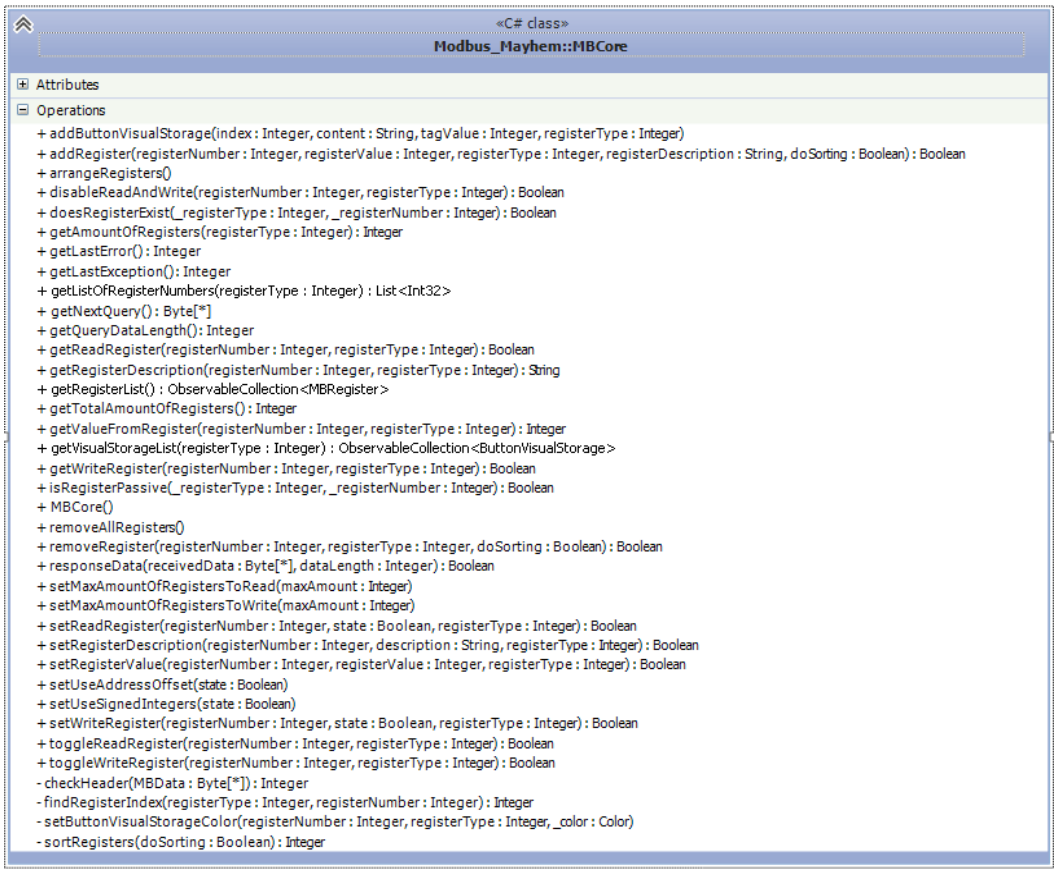


Kuva 16. Modbus Mayhem luokkineen

MainPage, connect, Registers, about, settings ja multiselection ovat sovelluksessa käytettäviä ikkunoita, joiden nimet viittaavat suoraan kappaleessa 3 käsiteltyihin sivuihin. Luokat MBCore, MBTCPComm, MBRegister, ButtonVisualStorage, checkIP, MBRegisterLite ja ServerProfile ovat näkymättömiä ohjelman luokkia. Yksikään mainituista luokista ei periydy ryhmän toisesta luokasta.

5.2 MBCore

MBCore on MM:n varsinainen moottori, sillä kaikki Modbus-protokollaan liittyvä toiminta sisältyy MBCoreen.



Kuva 17. MBCoren metodit

Luokka sisältää kuvassa 17 näkyvät metodit, joilla käsitellään rekistereitä sekä Registers-ikkunan painikkeita. Lisäksi luokka rakentaa Modbus-kyselyt lähetettäväksi palvelimelle, sekä tarkistaa ja purkaa vastaanotetut paketit.

Muutama lähinnä virheenkäsittelyyn luotu metodi on myöhemmin hylätty eivätkä ole käytössä, mutta ne on säilytetty luokassa, mahdollisten tulevien tarpeiden varalta. Lyhyt kuvaus metodeista aakkosjärjestyksessä:

- *AddButtonVisualStorage* – luo rekistereille Registers-ikkunassa näytettävän painikelistan elementit tyyppiä ButtonVisualStorage. Tämä

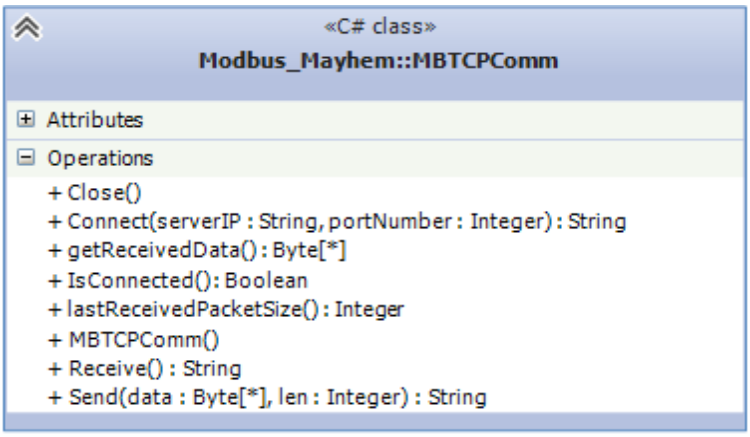
metodi on käytössä vain MBCoren konstruktorissa sovelluksen käynnistyessä.

- *addRegister* – luo halutun tyyppisen rekisterin ja alustaa sen.
- *arrangeRegisters* – järjestelee rekisterit numeron ja tyypin mukaan.
- *disableReadAndWrite* – asettaa rekisterin passiiviseksi
- *doesRegisterExist* – palauttaa tosi/epätosi riippuen siitä onko rekisteri jo luotu.
- *getAmountOfRegisters* – palauttaa valitun tyyppisten rekisterien lukumäärän.
- *getLastError* – Palauttaa viimeisimmän havaitun virheen koodin. MM ei käytä tätä metodia.
- *getLastException* – Palauttaa viimeisimän Modbus-poikkeuksen arvon. MM ei käytä tätä metodia.
- *getListOfRegisterNumbers* – palauttaa listan joka sisältää pyydetyn rekisterityypin olemassa olevien rekistereiden numerot.
- *getNextQuery* – palauttaa Modbus-paketin lähetettäväksi palvelimelle.
- *getQueryDataLength* – palauttaa edellisen getNextQuery-metodin generoiman paketin tavumäärän.
- *getReadRegister* – palauttaa kysytyn rekisterin luku-toiminnon tilan (tosi/epätosi).
- *getRegisterDescription* – palauttaa kysytyn rekisterin kommenttitekstin.
- *getRegisterList* – palauttaa rekisterilistan osoitteen.
- *getTotalAmountOfRegisters* – palauttaa kaikkien rekisterien määrän.
- *getValueFromRegister* – palauttaa rekisterin arvon.
- *getVisualStorageList* – palauttaa luodun ButtonVisualStorage-listan osoitteen.
- *getWriteRegister* – palauttaa kysytyn rekisterin kirjoitus-toiminnon tilan (tosi/epätosi).
- *isRegisterPassive* – palauttaa tiedon ovatko rekisterin sekä luku- ja kirjoitustoiminto pois päältä.
- *MBCore* – konstruktori. ButtonVisualStorage-lista luodaan täällä.
- *removeAllRegisters* – poistaa istunnon kaikki rekisterit.

- *removeRegister* – poistaa yksittäisen rekisterin.
- *responseData* – vastaanottaa MBTCPComm-luokan palauttaman Modbus paketin käsiteltäväksi.
- *setMaxAmountOfRegistersToRead* – määrittelee suurimman määrän luettavia rekistereitä per Modbus-kysely.
- *setMaxAmountOfRegistersToWrite* – määrittelee suurimman määrän kirjoitettavia rekistereitä per Modbus-kysely.
- *setReadRegister* – asettaa rekisterin luku-toiminnon päälle/pois.
- *setRegisterDescription* – tallentaa rekisterin kommentin rekisteriin.
- *setRegisterValue* – asettaa rekisterin arvon.
- *setUseAddressOffset* – asettaa ”Address Offset”-toiminnon päälle/pois.
- *setUseSignedIntegers* – asettaa esitykseen käytettävän kokonaislukumuuttujan tyypin päälle/pois.
- *setWriteRegister* – asettaa rekisterin kirjoitus-toiminnon päälle/pois.
- *toggleReadRegister* – asettaa rekisterin yksittäisen kirjoituksen aktiiviseksi. Tämä on ”Commit”-kentän ”Read”-valinnan asetus.
- *toggleWriteRegister* – asettaa rekisterin yksittäisen kirjoituksen aktiiviseksi. Tämä on ”Commit”-kentän ”Write”-valinnan asetus, käytössä vain Holding-rekistereille.
- *checkHeader* – luokan sisäinen metodi jota käytetään vastaanotetun Modbus-paketin otsikkotaulun tarkastamiseen.
- *findRegisterIndex* – luokan sisäinen metodi joka palauttaa kysytyn rekisterin indeksin rekisterit sisältävässä listassa. Ehdottomasti luokan eniten kutsuttu metodi.
- *setButtonVisualStorageColor* – luokan sisäinen metodi. Registers-sivulla näytettävien painikkeiden taustavärin määrittely.
- *sortRegisters* – luokan sisäinen metodi. Järjestää rekisterit listassa tyyppi- ja numerojärjestykseen.

5.3 MBTCPComm

MBTCPComm on luokka, joka luo ja sulkee tarvittaessa (asynkronisen) TCP-yhteyden palvelimelle, ja huolehtii Modbus-pakettien lähetyksestä ja vastaanotosta.



Kuva 18. MBTCPComm metodeineen

MBTCPComm luo asynkronisen yhteyden, mikä tarkoittaa sitä että ohjelman suoritus ei keskeydy odottamaan vastausta vastapuolelta lähetyksen jälkeen, vaan suoritus palautuu pääohjelmaan. Itse pääohjelma käyttää luokan dataAvailable-attribuuttia vastauksen saapumisen havaitsemiseen ja vastaanotetun paketin sisällön välittämiseen MBCore-luokalle. MBTCPComm ja MBCore vaihtavat vastaanotetun paketin tiiviillä kutsulla:

```
MyApp.mbc.responseData(MyApp.mbComm.getReceivedData(),
MyApp.mbComm.lastReceivedPacketSize());
```

jolle vetää vertoja vain yhtä tiivis kutsu jolla uusi lähetettävä paketti siirtyy MBCoresta MBTCPComm:lle:

```
MyApp.mbComm.Send(MyApp.mbc.getNextQuery(),
MyApp.mbc.getQueryDataLength());
```

Tämä on kaikki se vähä mitä itse pääohjelma osallistuu itse protokollan käsittelyyn – siirtää paketin luokalta toiselle ja takaisin.

”MyApp”-prefix metodin kutsussa tarvitaan koska molemmat luokat, mbc ja mbComm, kuuluvat ylemmälle App-luokan oliolle (ks. kuva 16), jonne viitatessa polku tarvitaan.

Luokan metodit selitettyinä aakkosjärjestyksessä:

- *Close* – sulkee TCP-yhteyden.
- *Connect* – luo TCP-yhteyden haluttuun IP-osoitteeseen ja porttiin.
- *getReceivedData* – palauttaa vastaanotetun Modbus-paketin.
- *IsConnected* – palauttaa yhteyden tilan, aktiivinen vaiko ei (tosi/epätosi).
- *lastReceivedPacketSize* – palauttaa vastaanotetun Modbus-paketin tavujen määrän.
- *MBTCPComm* – konstruktori.
- *Receive* – aktivoi vastaanoton. Vaikka metodi on julkinen sitä ei käytetä luokan ulkopuolelta.
- *Send* – vastaanottaa lähetettävän Modbus-paketin ja lähettää sen palvelimelle.

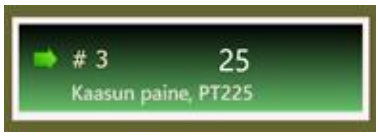
5.4 MBRegister

Vaikka Modbus-rekisteri itsessään on vain rekisterin numero ja arvo, on MM:n jokaisella rekisterillä oma instanssi luokasta MBRegister, joka taas sisältää huomattavasti enemmän tietoa kuin vain edellä mainitut. Luonnollisesti mukana on myös rekisterin tyyppi, mutta lisäksi luokkaan kuuluu rekisterin käyttötapa (luku, kirjoitus, passiivinen), rekisterin yhteydessä käytettävä pääikkunan painikkeen reunan väri, onko rekisterin käsittelyn yhteydessä palvelimelta vastaanotettu poikkeus, polku painikkeessa näytettävään symboliin, rekisterin selitysteksti, käytetäänkö etumerkillistä vaiko etumerkitöntä muuttujaa sekä onko rekisteri kenties valittu kirjoitettavaksi tai luettavaksi rekisterin valintaikkunan Commit-osuudessa. Osa attribuuteista sijaitsee luokassa vain käyttöliittymän takia, kuten polku symbolin sisältävään tiedostoon.

5.4.1 Listbox

XAML antaa mahdollisuuden rakentaa määrittely käyttöliittymän komponentissa käytettävälle elementille. Komponentti on tyyppiä ”ListBox”, ja elementti taas rekisteri-olio. Määritellä siis kuinka ListBoxin sisältämä yksittäinen rekisterielemetti näytetään. ”<ListBox.ItemTemplate>” kertoo, että seuraavaksi tulee määrittely, joka taas aloitetaan käyttämällä <DataTemplate> -elementtiä. ListBox-komponentin rekisterin esittämiseen käytetty kuvaus onkin seuraavanlainen:

```
<ListBox.ItemTemplate>
<DataTemplate>
<Grid Height="145">
<Button Width="460" Height="145" Tag="{Binding tagRegData, Mode=OneTime}" Border-
Brush="{Binding borderColorUsed, Mode=OneTime}" BorderThickness="8"
Tap="reg_list_Button_Click">
<Button.Background>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="Black" Offset="0"/>
<GradientStop Color="White" Offset="1"/>
<GradientStop Color="#FF64BD68" Offset="0.959"/>
</LinearGradientBrush>
</Button.Background>
<Button.Content>
<StackPanel Orientation="Horizontal" Width="400" Height="110">
<StackPanel>
<Image Width="40" Height="40" Source="{Binding logoPath, Mode=OneWay}" Margin="2,18"/>
</StackPanel>
<StackPanel></StackPanel>
<StackPanel Orientation="Vertical" Margin="0,20,0,0">
<StackPanel Orientation="Horizontal" Height="40">
<TextBlock FontFamily="Tahoma" Padding="10,0" Width="30" FontSize="28" Text="#"
Height="40" Foreground="Bisque"/>
<TextBlock FontFamily="Tahoma" Padding="10,0" Width="110" FontSize="28" Text="{Binding
registerNumber, Mode=OneTime}" Height="40" Foreground="Bisque"/>
<TextBlock Padding="50,0" Width="200" FontSize="36" Text="{Binding registerValue,
Mode=OneWay}" FontFamily="Segoe UI Semibold"/>
</StackPanel>
<StackPanel Orientation="Horizontal" Height="40">
<TextBlock Padding="10,0" Width="359" FontSize="24" Foreground="#FFE2DBD2" Text="{Binding
registerDescription, Mode=OneWay}" Height="40"/>
</StackPanel>
</StackPanel>
</StackPanel>
</Button.Content>
</Button>
</Grid>
</DataTemplate>
</ListBox.ItemTemplate>
```



Kuva 19. Mitä XAML kuvasi, se on tässä.

Kuvan 19 vihreä nuoli haetaan oliolta näin:

```
<Image Width="40" Height="40" Source="{Binding logoPath, Mode=OneWay}" Margin="2,18"/>
```

Jos aktiivisena olisi rekisterin kirjoitus, olisi rekisterin logoPath-muuttuja tarjonnut polun punaiseen nuoleen ja passiivisen rekisterin vastaavasti oman ”X”-symbolinsa kuvatiedostoon.

”#” on lisätty näin:

```
TextBlock FontFamily="Tahoma" Padding="10,0" Width="30" FontSize="28" Text="#" Height="40"
Foreground="Bisque"/>
```

Merkki on siis kiinteä, vakio. Mutta sitä seuraavat rekisterin numero, rekisterin sisältämä arvo ja -kommentti on tuotu näin:

```
<TextBlock FontFamily="Tahoma" Padding="10,0" Width="110" FontSize="28" Text="{Binding
registerNumber, Mode=OneTime}" Height="40" Foreground="Bisque"/>

<TextBlock Padding="50,0" Width="200" FontSize="36" Text="{Binding registerValue,
Mode=OneWay}" FontFamily="Segoe UI Semibold"/>

<TextBlock Padding="10,0" Width="359" FontSize="24" Foreground="#FFE2DBD2" Text="{Binding
registerDescription, Mode=OneWay}" Height="40"/>
```

Mikä tässä nyt on niin hienoa? Se, että edellä esitetty koodi huolehtii, että rekisterit näkyvät aina oikein sisältöineen pääsivulla, riippumatta oliko rekistereitä käytössä 1 vaiko kaikki 131 070 kappaletta. Arvot ”liitetään” näyttöön ”{Binding..}” –komennolla. Esittämiseen käytetty ListBox-komponentti yhdistetään MBCoren sisältämään rekisteriluetteloon antamalla sille nimi;

```
<ListBox Name="MixedListBox"..>
```

ja ohjelmallisesti kertomalla missä sen näytettäväksi tarkoitettu sisältö on tarjolla, juuri tälle ”MixedListBox” nimiselle komponentille;

```
MixedListBox.ItemsSource = MyApp.mbc.getRegisterList();
```

Esimerkin “mbc” on MBCoren instanssi.

5.5 ServerProfile, MBRegisterLite

ServerProfile-luokkaa käytetään vain tallennettaessa tai luettaessa tallennettu profiili ohjelmaan. Luokan konstruktori ajaa keräysrutiinin, jolla se hakee tiedot käytössä olevista rekistereistä MBCorelta ja tallentaa yksittäisen rekisterin tiedot MBRegister-luokkaa kevyempään luokkaan nimeltä MBRegisterLite. Tätä luokkaa käyttäen se taas rakentaa sisäiset listat (List<MBRegisterLite>) ohjelmassa käytettyjen rekistereiden mukaisesti. ServerProfile myös varastoi käytössä olevat parametrit omiin attribuutteihinsa. MBRegisterLite ei sisällä polkuja symboleihin tai sisäisiä tapahtumakäsittelijöitä muuttuneiden arvojen päivittämiseksi käyttöliittymässä, vaan pelkästään tiedot joiden avulla voidaan palauttaa tallennetun profiilin asetukset rekistereihin. Tämä siksi että tallennettava tietomäärä pysyisi mahdollisimman pienenä. Kun konstruktori on suoritettu, pääohjelma serialisoi luokan ja tallentaa sen valitulla profiilinimellä (sekä lisäten nimen loppuun tekstin ”MMData”, kuriositeettinä kerrottakoon). Palautus profiilia luettaessa tapahtuu yksinkertaisesti de-serialisoimalla luokka, jonka jälkeen pääohjelma poistaa nykyiset käytössä olevat rekisterit, purkaa varastoidut rekisterien tiedot ja parametrit ja ohjaa MBCorea luoden uudet, tallennettuja vastaavat rekisterit sekä ottamalla profiiliin tallennetut ohjelman parametrit käyttöön.

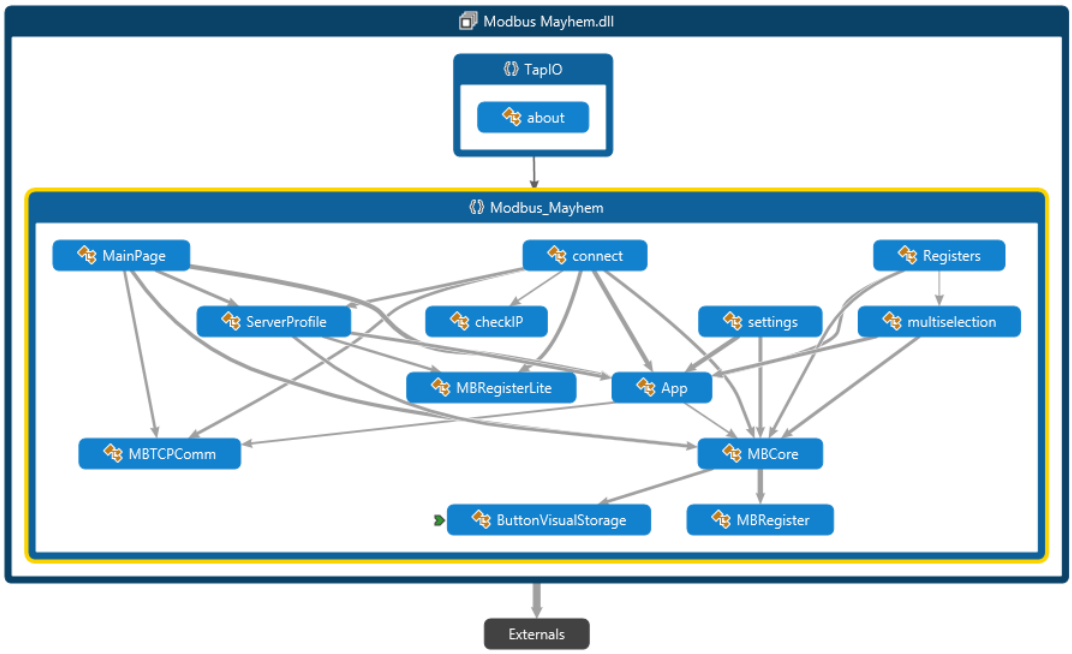
5.6 ButtonVisualStorage

ButtonVisualStorage-luokka on vain ja ainoastaan Registers-sivun listauksen painikkeiden värin ja numeron varastoimista varten. Paljastettakoon, että latauskuvan näkyessä luodaan instanssi luokasta jokaiselle rekisterille ja se vie hiukan aikaa.

5.7 checkIP

Luokka checkIP ei tee muuta kuin tarkistaa onko Connection-ikkunassa syötetty IP-osoite oikein koostettu. Luokka täydentää jo valmiiksi löytyvää IPAddress.TryParse-metodia mahdollistamalla pilkkujen ”,” käytön osoiteessa. Tällä pyritään vain varmistamaan, että mahdollisesti tulevaisuudessa puhelimen käyttöalueen mukaan muuttuvan desimaalierottimen kanssa ei synny ongelmia.

5.8 Luokkien riippuvuudet



Kuva 20. Luokkariippuvuudet

Kuva 20 esittää luokkariippuvuuksia, eli kuinka eri luokat käyttävät toisten luokkien resursseja tai luovat luokan instansseja omaan tarpeeseensa. Kuvasta voidaan helposti poimia edellisissä kappaleissa kerrottuja toiminnallisuuksia. Esimerkiksi luokka MainPage käyttää luokkaa ServerProfile profiilin tallennuksen yhteydessä, joka taas käyttää luokkaa MBRegisterLite käytettyjen rekistereiden tietojen tallentamiseen. Myös luokka connect käyttää ServerProfilea profiilin haun

yhteydessä. Lisäksi purkaessaan tiedot, se luo MBRegisterLite-luokan foreach-käsittelyään varten. Huomionarvoista on että kaikki rekistereihin liittyvä käsittely kulkee vain ja ainoastaan luokan MBCore kautta, samoin kuin Registers-sivun painikelistan ulkoasun muokkaus. Modbus-osio tietovarastoineen on tiukasti erotettu muun ohjelman toiminnasta ja rekistereiden käsittely tapahtuu pelkästään MBCoren tarjoamien palveluiden kautta. Kuvan ulkopuolisena keikkuva about-luokka on ulkona muun ohjelman osiosta, sillä sen nimiavaruus ei ole ”Modbus_Mayhem” vaan ”TapIO”. Se on kopioitu eri projektista osaksi MM:ää, siinä selitys.

LOPPUSANAT

Tämä lopputyö antoi minulle paljon. Sen avulla hankin suuren määrän tietoa ja kokemusta käytettäväksi sovellusten ohjelmoimiseen Windows Phone-alustalle Silverlight-ympäristössä. Taitoja, joita pyrin hyödyntämään kaupallisesti tulevaisuudessa. Modbus Mayhem ei tule olemaan kaupallisesti merkittävä, mutta siinä käytetyt tekniikat tulevat periytymään tuleviin sovelluksiini. Jotka toivottavasti ovat kaupallisesti merkittäviä.

Modbus Mayhemin koodaamiseen käytin noin kahden kuukauden ajan arki-illat, yksittäisen illan session keston ollessa luokkaa 5-6 tuntia. Suuri tuntimäärä selittyy ainakin osittain sillä, että taitojen kehittyessä poistin vanhaa koodia ja tein toiminnot yksinkertaisemmin tai hyödyntäen löytämääni uutta tekniikkaa. Lisäksi alussa sovellus oli laajempi pääsivun sisältäessä peräti viisi näkymää nykyisen kahden sijasta. Uskon, että yksinkertaistaminen osittain teknisten rajoitusten takia kuitenkin johti parempaan lopputulokseen. Sen ansiosta Modbus Mayhem ei sortunut ylenmääräiseen valikoiden ja valintojen runsauteen. Se ei saanut käyttöliittymää ”jonka on suunnitellut joku pahuksen insinööri”.

Olen tyytyväinen sovellukseen. Työni takia käytän Modbus-protokollaa käyttäviä ohjelmoitavia logiikkoja ja Modbus Mayhemiä voin aidosti hyödyntää testaamisessa. Käyttäminen on helppoa ja todellinen helmi on mahdollisuus

tallentaa profiileja. Ominaisuuden arvoa en täysin ymmärtänyt tehdessäni päätöksen sen sisällyttämisestä sovellukseen, mutta jollei sitä jo olisi se täytyisi lisätä. Sovelluksen potentiaalinen ostajakunta on pieni, mutta ne jotka sovelluksen hankkivat saavat erinomaisen Modbus-työkalun puhelimeensa.

Sovellusta voi parantaa lisäämällä kappaleessa 3.2 listattuja ominaisuuksia, jotka eivät päässeet mukaan hyödyllisyydestään huolimatta. Itse koodi pysyi varsin hyvin kasassa, johtuen ehkä osittain siitä että tiukka hierarkia, jossa rekistereiden käsittely kulkee vain ja ainoastaan luokan MBCore kautta, esti toiminnallisuuden hajoamisen niin että rekistereitä käsiteltäisiin ympäri sovellusta. Niiden ympärillähän kaikki sovelluksessa pyörii. En ole tyytyväinen VisualButtonStorage-luokan käyttöön. Se on hiukan ”päälleliimattu”, paremminkin hätäratkaisu kuin loppuun asti ajateltu. MBRegisters-luokka taas on turhankin iso ja osa sen sisältämistä parametreista voitaisiin kenties hajauttaa. Tällä tavoittelisin rekistereiden käsittelyn keventymistä. Suurimmaksi virheeksi koen kuitenkin sen että en käyttänyt säikeitä ohjelmassa ja tästä seuraa että ohjelma ”jäätyy” hetkeksi kun käsitellään tuhansia rekistereitä kerralla. Vasta loppuvaiheessa löytämäni BackgroundWorker-luokka tuli yksinkertaisesti liian myöhään.

Ensimmäinen kosketukseni olio-ohjelmointiin tapahtui Java-ohjelmointikielen kautta kolme vuotta sitten. C# ja Java muistuttavat paljon toisiaan ja siten kynnys aloittaa ja tutustua C#:n käyttöön jäi siten varsin matalaksi. Windows Phone – ohjelmointiin tutustuin lukemalla Charles Petzoldin kirjan ”Programming Windows Phone – Microsoft Silverlight Edition” /11/. Kirja on kirjoitettu Windows Phone versiolle 7.0 mutta on edelleen käyttökelpoinen. Kirja on ladattavissa ilmaiseksi sähköisessä muodossa Microsoft Pressin sivuilta. Kirjan saa myös paperisena, jollaisena omani hankin.

LÄHTEET

/1/ SuomiSanakirja.fi. Viitattu 16.1.2013.
http://synonyymit.fi/v%C3%A4yl%C3%A4

/2/ Lammert Bies. Viitattu 16.1.2013.
http://www.lammertbies.nl/comm/info/modbus.html

/3/ Industrial Equipment News. Viitattu 16.1.2013.
http://www.ien.com/article/eight-popular-open/562

/4/ Modbus Organization. Viitattu 17.1.2013.
http://modbus.org/faq.php.

/5/ Modbus Organization. Viitattu 16.1.2013.
http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.

/6/ Simply Modbus. Viitattu 16.1.2013.
http://www.simplymodbus.ca/faq.htm

/7/ MBLogic. Viitattu 17.1.2013.
http://mblogic.sourceforge.net/mbapps/ModbusBasics-en.html

/8/ The Escapist. Viitattu 08.04.2013
http://www.escapistmagazine.com/news/view/121843-Microsoft-Confirms-the-End-of-XNA

/9/ Windows Phone Dev Center. Viitattu 08.04.2013.
http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207003(v=vs.105).aspx

/10/ Uniblue PC-Library. Viitattu 22.04.2013.
http://www.pc-library.com/ports/tcp-udp-port/502/

/11/ Petzold Charles. 2010. Programming Windows Phone 7. Microsoft Press. Viitattu 01.05.2013.
http://blogs.msdn.com/b/microsoft_press/archive/2010/10/28/free-ebook-programming-windows-phone-7-by-charles-petzold.aspx