Tampereen ammattikorkeakoulu

# I/O ja CAN -laajennusyksikön tuki Multitool Creatorissa

Johannes Lindeman

OPINNÄYTETYÖ
Maaliskuu 2023

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

LINDEMAN, JOHANNES:
I/O ja CAN -laajennusyksikön tuki Multitool Creatorissa

Opinnäytetyö 51 sivua
Maaliskuu 2023

---

Opinnäytetyön päämääränä oli kehittää toimeksiantajayrityksenä toimineen Epec Oy:n ja tämän asiakasyrityksen yhdessä kehittämälle I/O ja CAN -laajennusyksikölle tuki Epec Multitool Creator -konfiguraatiosovellukseen. Laajennusyksikön tuki luo edellytykset konfiguroida yhä kehittyneempien ohjausjärjestelmien sovelluksia Multitool Creatorin avulla. Laajennusyksikön on tarkoitus toimia yhdessä samojen tahojen kehittämän keskusyksikön kanssa ja laajentaa oma I/O- ja CAN-liikenteensä keskusyksikön käyttöön ethernet-kytkennän yli.

Työssä kehitettiin Multitool Creatoriin laitetuki laajennusyksikölle ja tarvittavat uudet ominaisuudet laajennusyksikön ja keskusyksikön yhteistoiminnan mahdollistamiseksi. Uusina ominaisuuksina Multitool Creatoriin kehitettiin vaatimusten mukaisesti muun muassa ethernet-kytkentöjen mahdollistaminen sekä validointi, hallitsevan keskusyksikön valintamahdollisuus ja ethernet-kytkentöjen ketjutusominaisuudet.

Työssä hyödynnettiin laajasti Microsoftin ohjelmistokehitys- ja projektinhallintatyökaluja. Ohjelmoinnissa hyödynnettiin Visual Studio 2022 -ohjelmointiympäristöä ja .NET Frameworkiä. Projektinhallintaan käytettiin Azure DevOpsia.

Projektin kehitystyön aikataulu ei sisältänyt tarkkaa takarajaa, vaan kehittämiseen oli varattu varsin laaja aikaikkuna. Kehitystyön tuotokset on määrä julkaista osana Epec Multitool Creatoria vuoden 2023 aikana. Työn tuloksena syntyneitä tuotoksia tullaan jatkokehittämään edellä mainitun keskusyksikön tuen kehittämisen yhteydessä.

---

Asiasanat: can, i/o, ethernet, multitool creator, azure devops

## ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

LINDEMAN, JOHANNES:
I/O & CAN Extension Unit Support in Multitool Creator

Bachelor's thesis 51 pages
March 2023

---

The target of this bachelor’s thesis was to develop a Multitool Creator support for an I/O & CAN extension unit for Epec Oy. The support for the extension unit will make it possible to configure highly developed control system applications using the Multitool Creator. The extension unit will complement a central unit that is being developed by the same parties as the extension unit. The purpose of the extension unit is to extend its I/O and CAN traffic over ethernet for the use of the central unit.

The development project consisted of developing the extension unit’s device support and all the required new features to allow the extension unit and the central unit to function together in the Multitool Creator.

New features developed for Multitool Creator consisted of the ability to make and validate ethernet connections, choose a controlling central unit for each extension unit, and chaining multiple devices over ethernet.

The products of the development project will be further evolved during the development of the previously mentioned central unit and are to be published as a part of a new release of the Epec Multitool Creator in the year 2023.

---

Key words: can, i/o, ethernet, multitool creator

**SISÄLLYS**

**ERITYISSANASTO**

| | |
|---|---|
| Azure DevOps | Projektinhallintapalvelu |
| Backlog item | Sovellukseen liittyvä tehtävä Azure DevOpsissa |
| Binding | Datajäsenen sidonta |
| Build | Ohjelmiston versio käännösvaiheessa |
| C# | Ohjelmointikieli |
| CAN | Tiedonsiirtoväylä (Controller Area Network) |
| CANopen | Tiedonsiirtoprotokolla |
| CODESYS | Ohjausjärjestelmien kehitystyökalu |
| ECU | Elektroninen hallintayksikkö (Electronic Control Unit) |
| Ethernet | Tiedonsiirtoväylä |
| Exception | Poikkeus |
| Feature | Sovellukseen toteutettava ominaisuus |
| Framework | Kehitysympäristö |
| Kanban | Projektinhallintamenetelmä (jap. taulu) |
| MultiBinding | Usean datajäsenen sidonta |
| NetworkChain | Yhtenäinen laiteverkosto |
| OD | Tiedonsiirrossa käytettävien muuttujien kokoelma CA-Nopen-protokollassa (Object Dictionary) |
| PDO | Lähetettävä/vastaanotettava viesti, dataobjekti CA-Nopen-protokollassa (Process Data Object) |
| Task | Ohjelmiston osan toteutukseen tai korjaukseen liittyvä alitehtävä Azure DevOpsissa |
| WPF | Käyttöliittymien graafisten toteutuksien työkalu (Windows Presentation Foundation) |
| XAML | Ohjelmointikieli |

# 1 JOHDANTO

Teknologian kehittyessä syntyy jatkuvasti tarvetta myös yhä kehittyneemmille ja monipuolisimmille ohjausjärjestelmien ohjelmistokehitysratkaisuille. Tarvitaan ratkaisuja ja tekniikoita, joilla voidaan kehittää yhä kompleksisempia ja edistyneempiä järjestelmiä. Yhä edistyneempien ohjausjärjestelmien ohjelmistojen kehittäminen vaatii myös uudenlaisia ratkaisuja näiden ohjelmistojen kehitystyökaluilta.

Tässä opinnäytetyössä käydään läpi työn toimeksiantajayrityksen sekä tämän asiakasyrityksen yhteistyössä kehittämän I/O ja CAN -laajennusyksikön tuen kehittämistä Epec Multitool Creator -konfiguraatiosovellukseen. Työn toimeksiantajayrityksenä toimi Epec Oy, joka on vuonna 1978 perustettu sulautettuihin ohjausjärjestelmiin ja työkoneiden tietoliikenteeseen erikoistunut järjestelmätoimittaja. (Epec 2022.)

Reaalimaailmassa uuden I/O ja CAN -laajennusyksikön on tarkoitus toimia yhdessä ohjelmoitavan keskusyksikön kanssa ja laajentaa keskusyksikön CAN- ja I/O-liikennettä ethernet-kytkennän yli.

Työssä käytettiin ohjelmointikielenä C#-kieltä ja ohjelmointiympäristönä toimi Visual Studio 2022. Graafisen käyttöliittymän toteutuksissa hyödynnettiin Windows Presentation Foundation (WPF) -kehitysympäristöä ja XAML-ohjelmointikieltä. Projektinhallintaan käytettiin Azure DevOps Server -palvelua ja versionhallintaan Azure DevOpsiin kuuluvaa Team Foundation Version Controlia. Tekniikoista kerrotaan tarkemmin myöhemmissä kappaleissa.

Tämän raportin pyrkimyksenä on antaa lukijalleen kokonaisvaltainen käsitys I/O ja CAN -laajennusyksikön toiminnoista Multitool Creatorissa sekä vaadittujen ominaisuuksien suunnittelusta ja toteuttamisesta C#-ohjelmointiprojektissa.

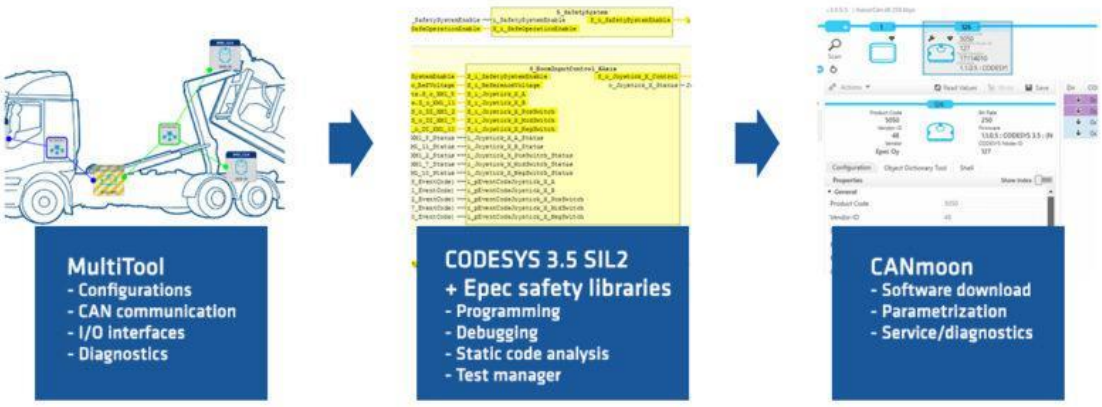
# 2 MULTITOOL CREATOR -KONFIGURAATIOTYÖKALU

Tässä luvussa kuvaillaan Multitool Creator -konfiguraatiotyökalun toimintaperiaatteita, ominaisuuksia sekä tärkeimpien ominaisuuksien taustoja.

## 2.1 Yleisesti

Epec Multitool on ohjelmistokehitystyökalu, jonka tarkoituksena on helpottaa ja nopeuttaa ohjausjärjestelmien ohjelmoinnin konfiguraatiovaihetta. Multitool Creatorin käyttäminen säästää ohjausjärjestelmäsovelluksen konfiguraatiovaiheeseen käytettävää aikaa jopa 90 %. Multitool Creator tarjoaa käyttäjälleen graafisen käyttöliittymän muun muassa kommunikaatioprotokollien ja I/O:n konfigurointiin. (Epec 2021). Kun käyttäjä on Multitool Creatorissa määritellyt järjestelmään haluamansa laitteet ja niiden konfiguraatiot, voidaan luoda näihin konfiguraatioihin perustuva CODESYS-projekti.

Multitool Creatoriin hyvin oleellisesti liittyvä CODESYS on ohjelma, joka toimii kehitysympäristönä ohjausjärjestelmäsovellusten ohjelmointiin. Se kattaa kaikki viisi PLC-ohjelmoinnin (Programmable Logic Controller) IEC 61131-3 -standardiin kuuluvaa ohjelmointikieltä. CODESYSissä käyttäjä voi kehittää ohjausjärjestelmäsovelluksia Epecin ohjelmoitaville laitteille ja kaikkiaan yli miljoonalle eri laitteelle maailmanlaajuisesti. (Motion Ai 2019.) Kun CODESYSissä on luotu valmis sovellus, voidaan se ladata ohjelmoitavaan yksikköön joko suoraan CODESYSistä tai CANmoonin avulla. CANmoon on Multitool Creatorin tavoin Epecin kehittämä ohjelmistotyökalu ja sitä voidaan sovellusten lataamisen lisäksi käyttää mm. ohjelmoitavien yksiköiden laiteohjelmistojen päivittämiseen ja CAN-liikenteen monitorointiin (Epec 2015). Multitool Creatorin, CODESYSin ja CANmoonin avulla ohjausjärjestelmäsovellus voidaan konfiguroida, toteuttaa ja ottaa käyttöön kolmessa vaiheessa (kuva 1).




KUVA 1. Ohjausjärjestelmäsovelluksen vaiheet (Epec n.d.).

## 2.2 Laitetuki Multitool Creatorissa

Multitool Creatorissa yksiköiden laitetuki pohjautuu kunkin laitteen yksilöllisiin ominaisuuksiin. Ominaisuuksien pohjalta on luotu laitekohtaiset XML-tiedostot, joissa määritellään muun muassa laitekohtaiset tunnistetiedot, kuten laitenumero ja -tyyppi, CAN-väylien ja I/O-pinnien lukumäärät sekä laitteen muut ominaisuudet. Kun käyttäjä lisää projektiin laitteen, luetaan Multitool Creatorin ohjelmakoodissa kyseisen laitteen XML-tiedosto ja luodaan laitteelle käyttöliittymänäkymä, jossa näytetään laitteen ominaisuuksiin perustuvat konfiguraatiovälilehdet.

Multitool Creator tarjoaa laitetuen useille ohjelmoitaville yksiköille sekä CANopen slave -yksiköille. Ohjelmoitaviin yksiköihin kuuluu sekä tavallisia että safety-yksiköitä. Ohjelmoitavilla yksiköillä tarkoitetaan Epecin ECU:ita, joista voidaan generoida Multitool Creatorissa CODESYS-projekti ja joihin voidaan ladata valmis CODESYS-sovellus. Safety-yksiköt (esim. Epec SL84) ovat muuten samankaltaisia kuin tavalliset ohjelmoitavat yksiköt, mutta ne tukevat useita laiteturvallisuuteen liittyviä toiminnallisuuksia, joilla voidaan esimerkiksi estää odottamattomia liikkeitä tai tarvittaessa pysäyttää laitteen toiminta (Epec 2022).

CANopen slave -yksiköllä tarkoitetaan yksikköä, joka ottaa käskyjä vastaan joltain ohjelmoitavalta yksiköltä ja toteuttaa ne. Ohjelmoitava CANopen master -yksikkö siis määrittää CANopen slave -yksikön toiminnan. Multitool Creatoriin on toistaiseksi kehitetty tuki Epecin omalle GL84 CANopen slave -yksikölle ja sen lisäksi on mahdollisuus luoda kustomoituihin määrityksiin perustuva geneerinen

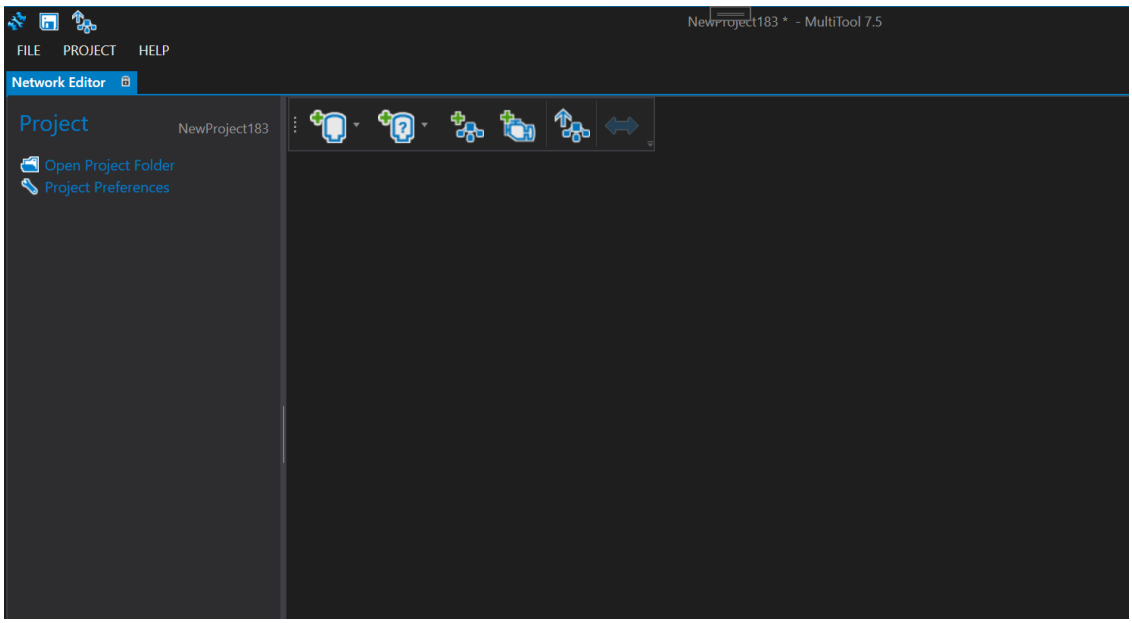
CANopen slave -yksikkö. Geneerisellä CANopen slave -yksiköllä tarkoitetaan Multitool Creatorissa yksikköä, joka ei edusta mitään Epecin todellista laitetta, vaan sillä simuloidaan periaatteessa minkä tahansa CANopen slave -yksikön toimintaa. Reaalimaailmassa CANopen slave on mikä tahansa CANopen-protokollan toteuttava laite, joka vastaanottaa käskyjä ja toteuttaa ne. Geneerinen CANopen slave luodaan Multitool Creatorissa lukemalla geneerisen yksikön xml-tiedosto sekä slaven kommunikaatioparametreja sekä PDO- ja OD-määrityksiä varten luotu eds-, dcf-, xdd-, tai xdc-tiedosto, jonka käyttäjä valitsee. Geneerisen CANopen slaven luontiin käytettävän tiedoston voi luoda esim. Multitool Creatorissa exportoimalla sellaisen haluamillaan PDO- ja OD-määrityksillä jonkin ohjelmoitavan yksikön Object Dictionarysta. Tässä työssä käsiteltävä laajennusyksikkö on CANopen slave -tyyppinen laite.

## 2.3 Käyttöliittymä

Multitool Creatorin käyttöliittymän keskeisimmät elementit ovat verkkoeditori sekä laiteasetusnäkymä. Tässä kappaleessa kuvaillaan Multitool Creatorin käyttöliittymää yleisellä tasolla, eikä jokaista komponenttia ja toiminnallisuutta sanallisteta erikseen. Käyttöliittymänäkymiä käsitellään tarkemmin seuraavassa luvussa laitteen ominaisuuksien tarkastelun yhteydessä.

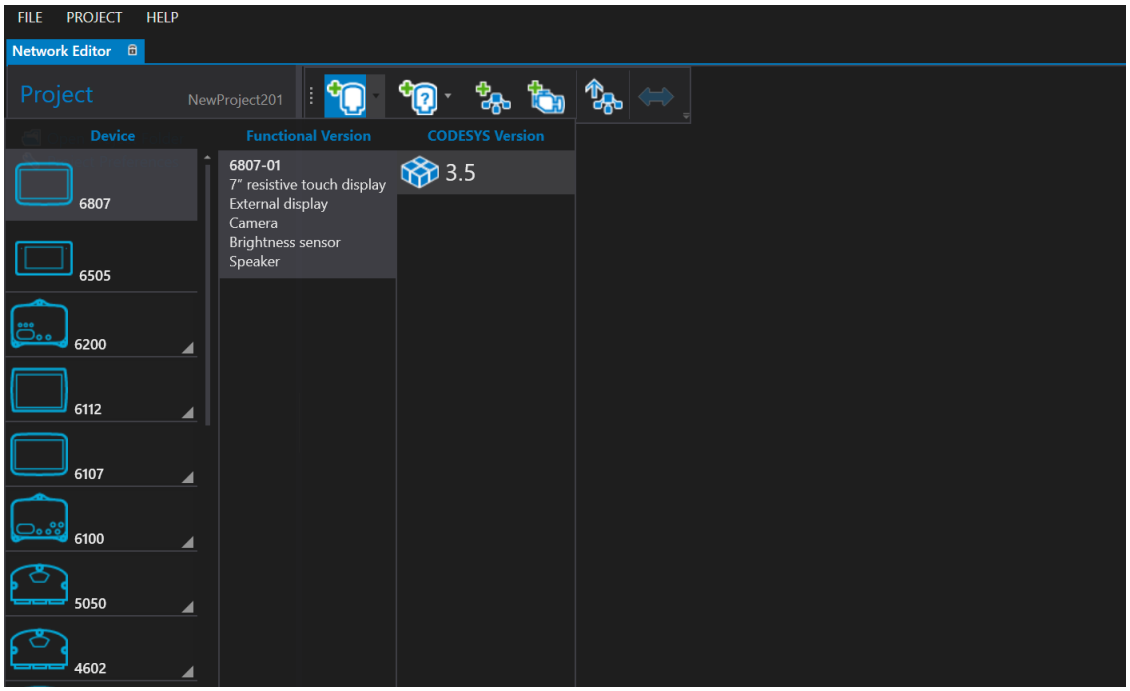
### 2.3.1 Verkkoeditori

Verkkoeditorissa (kuva 2) käyttäjä voi muun muassa lisätä projektiin haluamansa laitteet ja yhdistää ne toisiinsa CAN-liitoksella. Tässä työssä toteutettiin täysin uutena ominaisuutena myös ethernet-liitoksen mahdollisuus, jota käsitellään myöhemmin.



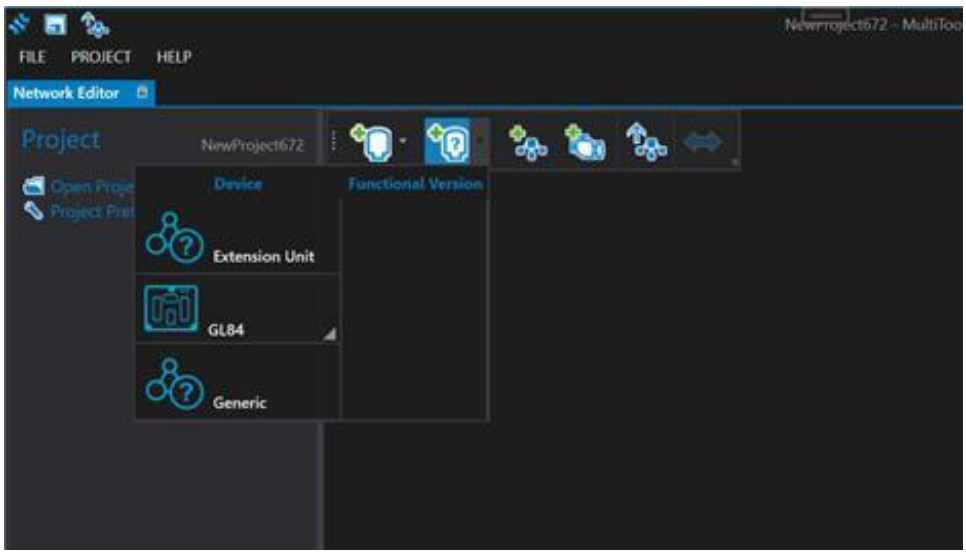

KUVA 2. Multitool Creatorin verkkoeditori.

Käyttäjälle esitetään yläpalkissa alasvetovalikoita, joista käyttäjä pääsee valitsemaan projektiin haluamansa laitteet (kuva 3).




KUVA 3. Alasvetovalikko ohjelmoitavien yksiköiden valintaan Multitool Creatorissa.

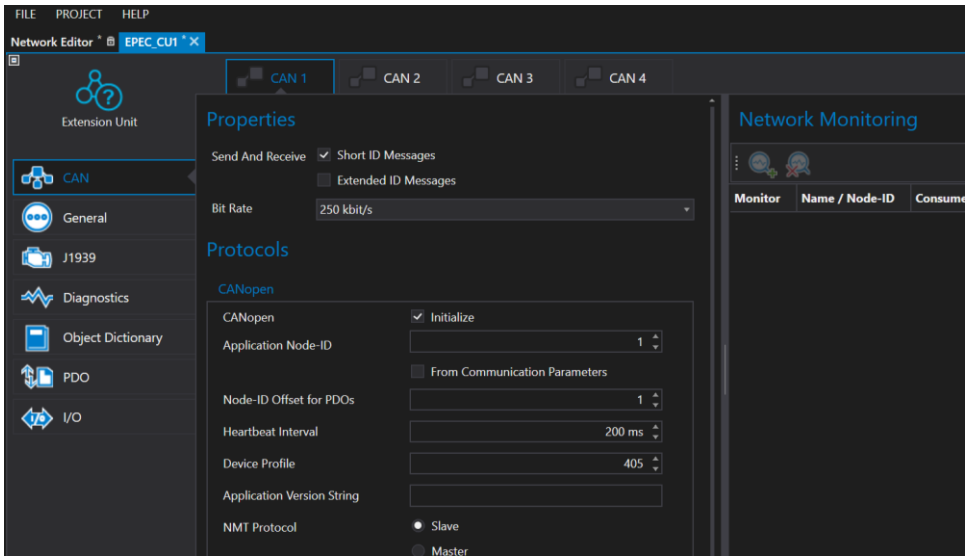
CANopen slave -laitteet, joihin laajennusyksikkökin kuuluu, näytetään omassa erillisessä valikossaan (kuva 4).




KUVA 4. CANopen slave -yksikköjen alasvetovalikko Multitool Creatorissa.

### 2.3.2 Laiteasetukset

Laiteasetuksissa käyttäjälle näytetään kyseisen laitteen ominaisuuksiin kuuluvat konfiguraatiovälilehdet (kuva 5). Ohjelmoitavilla yksiköillä ominaisuuksia on yleensä huomattavasti enemmän kuin CANopen slave -yksiköillä. Laajennusyksikön laiteasetuksiin kuuluu CAN-näkymä sekä General-, J1939-, Diagnostics-, PDO-, I/O- ja Object Dictionary -välilehdet. Yhteistä suurimmalle osalle laitteista on CAN-näkymä, joka esitetään laitteen jokaiselle CAN-väylälle, mahdollistaen verkkojen monitoroinnin ja eri konfiguraatiot laitteen kullekin väylälle.




KUVA 5. Laajennusyksikön konfiguraatiovälilehtiluettelo ja CAN-näkymä Multitool Creatorissa.

# 3 VAATIMUKSET, SUUNNITTELU JA VERSIONHALLINTA

Tässä luvussa käsitellään kehitysprojektin toteutuksen vaatimuksia vaatimusmäärittelyn mukaisesti sekä esitellään projektin suunnitteluun ja versionhallintaan käytettyjä työkaluja.

## 3.1 Vaatimukset

Vaatimusmäärittelyn mukaan laajennusyksikön CAN-väylät ja I/O tulee käsitellä keskusyksikössä kuin ne olisivat lokaaleja, eli keskusyksikön omia. CAN-väylät ja I/O tulee välittää keskusyksikölle ethernetin yli. Esimerkiksi laajennusyksikön CAN-väylät välitetään ethernetin yli ja laiteohjelmisto abstraktoi CAN-väylät näyttäytymään tavallisina, lokaaleina CAN-väylinä.

### 3.1.1 Keskus- ja laajennusyksiköt Multitool Creatorin käyttöliittymässä

Multitoolin verkkoeditori näyttää kaikki projektiin lisätyt laitteet. Jokaiselle yksikölle näytetään yksikön CAN-väylät, jotka voidaan kytkeä verkkoon kokonaisten järjestelmien luontia varten. Keskusyksikön ja laajennusyksikön tukea varten on käyttöliittymässä näytettävä myös niiden ethernet-väylät.

Väylien topologia näytetään CAN- ja ethernet-väylille. Laajennusyksikön CAN-väylät näytetään osana laajennusyksikköä, kuten ne ovat reaalimaailmassa. Ethernet-väylät erottuvat CAN-väylistä käyttöliittymässä muotonsa puolesta.

Verkko luodaan automaattisesti, kun yksikön ethernet yhdistetään toisen yksikön ethernetiin. Ethernet- ja CAN-väyliä ei voi yhdistää keskenään. Ensimmäinen tyhjään verkkoon yhdistetty väylä määrittää verkon tyypin: jos CAN-väylä yhdistetään tyhjään verkkoon, verkko on CAN-verkko; jos ethernet-väylä yhdistetään tyhjään verkkoon, verkko on ethernet-verkko.

Ethernet-väyliä tulee voida ketjuttaa laajennusyksiköiden yli. Esimerkiksi kun keskusyksikkö on kytketty ethernetin yli laajennusyksikköön ja tämä laajennusyksikkö on kytketty ethernetin yli toiseen laajennusyksikköön, keskusyksikkö näkee molemmat yksiköt samalla tavalla, vaikka suoraa kytkentää toiseen yksikköön ei ole. Keskusyksikön vaatimusten mukaan ainakin 15 laajennusyksikköä tulee voida kytkeä yhteen keskusyksikön ethernet-väylään.

Laajennusyksikön asetukset sisältävät keskusyksikön valintamahdollisuuden, eli kunkin laajennusyksikön asetuksista valitaan, mikä verkostossa oleva keskusyksikkö laajennusyksikköä hallitsee. Laajennusyksikön CAN-konfiguraatiovälilehdellä näytetään CAN-väylälle laajennusyksikön lokaali CAN-numero sekä hallitsevan keskusyksikön laajennettu CAN-numero.

Abstraction-, Events- ja Library Manager -konfiguraatiovälilehtiä ei näytetä laajennusyksikölle, koska laajennusyksikölle itselleen ei luoda applikaatiota, vaan se luodaan keskusyksikölle ja se käyttää keskusyksikön kirjastoja. Muutoin laajennusyksiköllä on täsmälleen samat konfiguraatiovälilehdet kuin Epecin ohjelmoitavilla yksiköillä.

Laajennusyksikön CAN-viestejä ja I/O:ta käytetään keskusyksikön applikaatiossa samoin kuin keskusyksikön omia lokaaleja CAN-viestejä ja I/O:ta. Täten keskusyksikössä laajennusyksikön muuttujat validoidaan kuin ne olisivat sen omia, eli identtisiä muuttujanimiä ei sallita.

### 3.1.2 Koodipohjan generointi

Laajennusyksikkö ei ole ohjelmoitava yksikkö, joten sille ei luoda koodipohjaa, vaan sen asetukset generoidaan keskusyksikön koodipohjaan. Keskusyksikön koodipohjassa kaikki laajennusyksikön CAN-väylät ja I/O käsitellään samoin kuin keskusyksikön omat CAN-väylät ja I/O. Ohjelmoija näkee laajennusyksikön I/O-muuttujat ja -pinnit kuin ne olisivat keskusyksikön omaa I/O:ta. Laajennusyksikön CAN-viestit lähetetään ja vastaanotetaan kuin ne olisivat keskusyksikön omaa CAN-liikennettä.

### 3.1.3 Avoimet kysymykset

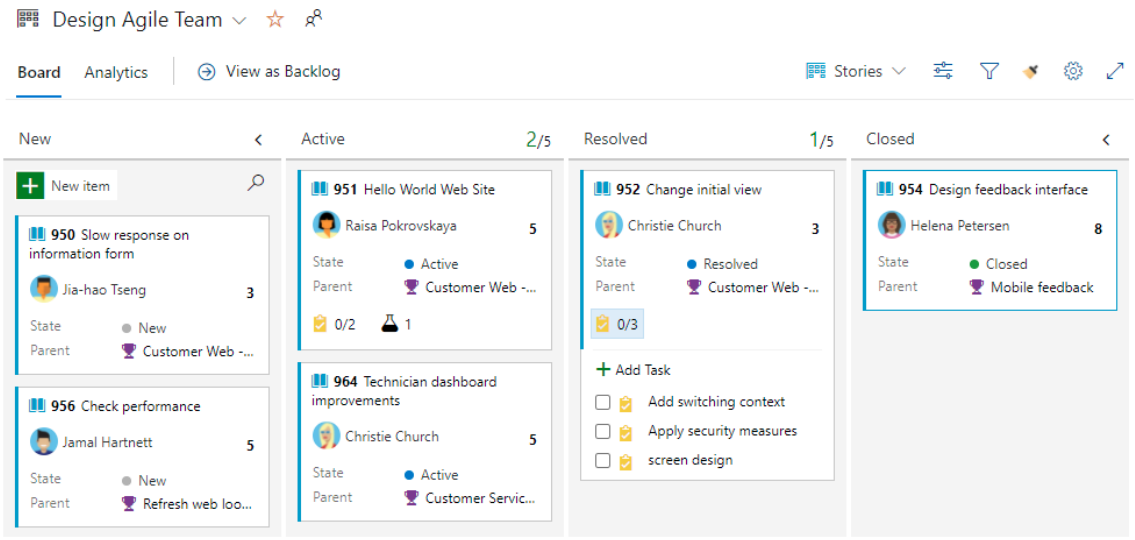
Raportin kirjoitushetkellä on vielä epäselvää, tuleeko keskus- ja laajennusyksiköiden välisiin kytkentöihin rajoituksia. Kirjoitushetkellä on myös määrittelemättä, tuleeko ketjutuksen tapahtua laajennusyksiköillä oletuksena vai tuleeko ominaisuutta hallita erikseen. Myös ketjutettujen yksiköiden lopullinen maksimilukumäärä on toistaiseksi kyseenalainen. Laajennusyksikön abstraktointi keskusyksikön laiteohjelmistossa tullaan toteuttamaan myöhemmin määriteltävänä ajankohtana ja siihen liittyvät kysymykset pysyvät toistaiseksi avoimina.

## 3.2 Suunnittelu

Laadukkaan ja yksityiskohtaisen vaatimusmäärittelyn ansiosta projektin suunnittelu oli kohtuullisen suoraviivaista ja helppoa. Vaatimusdokumentaatiossa määritellyt asiat yksinkertaisesti jaettiin Azure DevOps Server -palveluun kuuluvassa Azure Boardsissa erityyppisiksi tehtäviksi. Azure Boardsissa erilaisten tehtävien seurannassa ja hallinnassa hyödynnetään kanban-menetelmää, joka selkeyttää ja visualisoi kehitystyön vaiheita. Kanban-menetelmä tarkoittaa yksinkertaisuudessaan sitä, että työt pilkotaan pienemmiksi tehtäviksi ja ne kirjataan näkyviin nimikkeisiin, jotka asetetaan kanban-taululle. Kanban-taululla on työn vaiheita kuvaavia sarakkeita ja tehtävänimikkeitä siirretään taululla etenemisen mukaan. (Lekman 2009.)

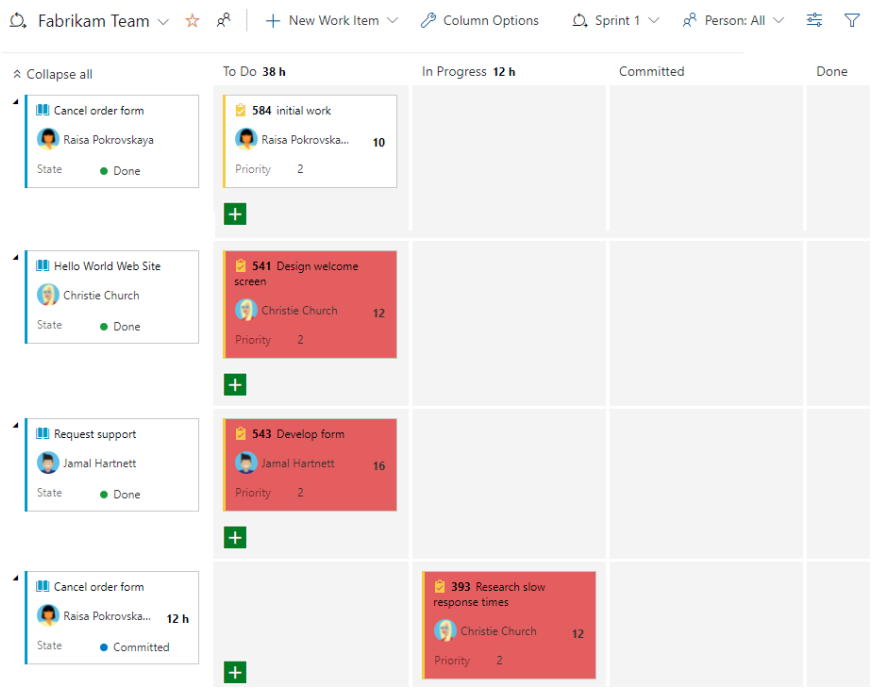
Työn suunnittelu aloitettiin luomalla Azure Boardsissa ensin koko projektia edustava feature-tyyppinen tehtävä, johon linkitettiin vaatimusmäärittely ja kirjattiin projektin yleinen kuvaus. Featuren alle projektia pilkottiin pienemmiksi osiksi, backlog item -tyyppisiksi tehtäviksi. Azure Boardsissa tehtäville voi asettaa monia erilaisia relaatioita, mutta tälle projektille hyödyllisin ja eniten käytetty relaatio oli tavallinen parent-child-relaatio. Feature toimi koko projektin ylimmän tason tehtävänä ja backlog itemeista tehtiin featuren childeja, eli alemman tason tehtäviä. Kun backlog itemit puolestaan palasteltiin yhä pienemmiksi task-tyyppisiksi tehtäviksi, taskit asetettiin backlog itemien alemman tason tehtäviksi. Kaikille tehtävätyypeille voidaan määrittää tehtävään käytettävä arvioitu työaika.

Backlog itemit ja bugit näytetään boards-näkymässä seinällä (kuva 6), jossa on neljä kolumnia: New, Approved, Resolved ja Closed. Kolumnit voidaan nimetä myös eri tavalla tarvittaessa. Tehtäviä siirretään tilasta toiseen vasemmalta oikealle sitä mukaa kuin kukin tehtävä etenee. Dokumentaation kannalta backlog itemeihin ja bugeihin on tärkeää kirjata tarpeeksi tietoa tehtävän sisällöstä ja liittää versionhallinnassa lähdekoodimuutokset (changeset) osaksi tehtävää.




KUVA 6. Boards-näkymä ja backlog itemien elinkaari (Microsoft 2022).

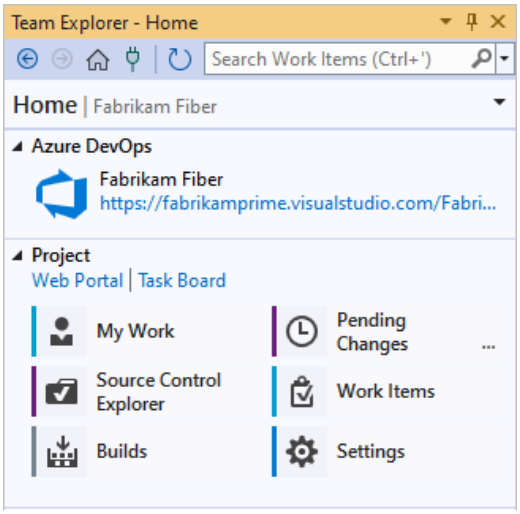
Backlog itemien ja bugien alle kuuluvat task-tyyppiset tehtävät näkyvät samankaltaisella seinällä Sprint-näkymässä (kuva 7). Taskeja voi käyttää ikään kuin vain työn seuraamisen apuvälineenä, mutta niihinkin voi ja on suotavaa kirjata ylös tietoa tehtävän yhteydessä suoritetuista toimenpiteistä.




KUVA 7. Sprint-näkymä ja taskien elinkaari (Microsoft 2022).

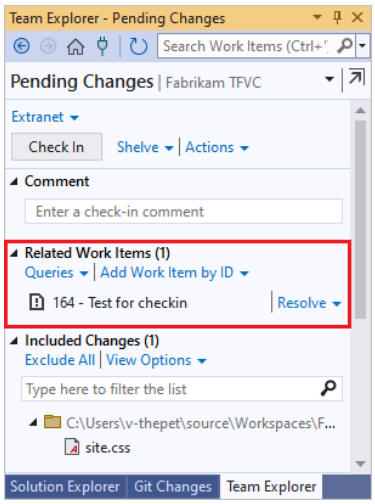
## 3.3 Versionhallinta

Projektin versionhallinnassa hyödynnettiin Teams Foundation Version Controlia, joka on myös osa Azure DevOps Server -palvelua. Teams Foundation Version Control ja Azure DevOps Server tarjoavat monipuolisen ja helppokäyttöisen Visual Studio -integraation, jonka avulla kaikki versionhallintaan liittyvät tehtävät voidaan suorittaa suoraan Visual Studion käyttöliittymän Team Explorer -näkymässä (kuva 8) (Microsoft 2022).



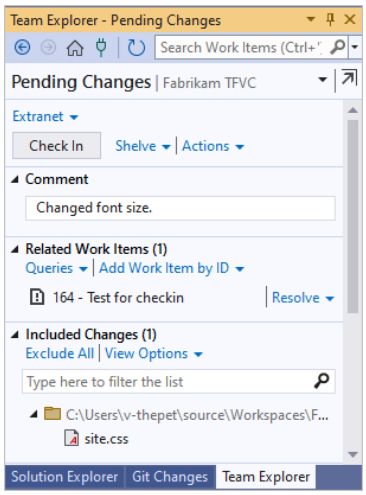

KUVA 8. Team Explorer -näkymä Visual Studiossa (Microsoft 2022).

Team Foundation Version Control mahdollistaa myös muiden Azure DevOps Server -palveluiden laajamittaisen hyödyntämisen versionhallinnassa. Visual Studio -integraatiossa voidaan esimerkiksi suoraan liittää lähdekoodiin tehdyt muutokset Azure DevOpsissa määriteltyihin tehtäviin (kuva 9) ja tarkastella lähdekooditiedostojen versiohistoriaa. (Microsoft 2022.)




KUVA 9. Azure DevOps -tehtävän liittäminen lähdekoodin päivitykseen (Microsoft 2022).

Palvelimella sijaitsevien lähdekooditiedostojen päivittäminen tapahtuu TFVC:ssä ”check in” -komennolla, joka lähettää muutokset palvelimelle. Hyvien ohjelmointikäytäntöjen mukaisesti jokainen päivitys tulee kommentoida muutosta kuvaavalla tavalla (kuva 10). (Microsoft 2022.)




KUVA 10. Palvelimelle lähettämistä odottava valmis muutospaketti (Microsoft 2022).

# 4 LAAJENNUSYKSIKKÖ

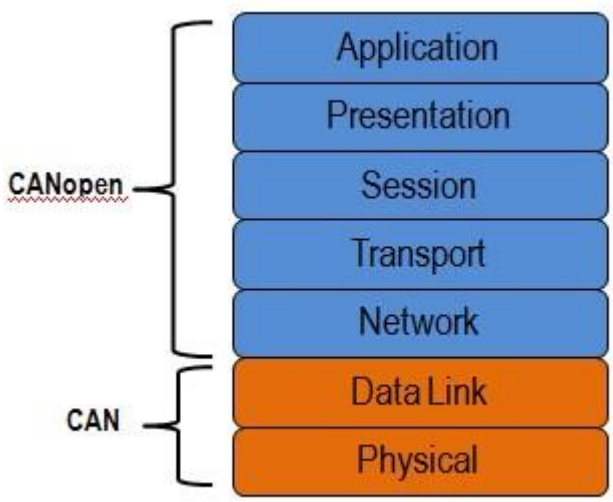
Työssä kehitettyjen uusien ominaisuuksien lisäksi laajennusyksikkö tukee useita Multitool Creatorin yleisiä ominaisuuksia. Laajennusyksikön yleiset ominaisuudet keskittyvät pääasiassa laitteidenväliseen tiedonsiirtoon ja niiden avulla käyttäjä voi määrittää tiedonsiirrossa käytetyt muuttujat sekä parametrit.

## 4.1 Yleiset ominaisuudet

Tässä kappaleessa käydään läpi laajennusyksikön oleellisimmat yleiset ominaisuudet, eli sellaiset ominaisuudet, jotka löytyvät entuudestaan muiltakin laitteilta Multitool Creatorissa. Laajennusyksikön keskeisimpiin yleisiin ominaisuuksiin kuuluu CAN-, PDO-, Object Dictionary- ja I/O-konfiguraatiovälilehdet. Nämä ominaisuudet toimivat aivan kuten ohjelmoitavilla yksiköillä.

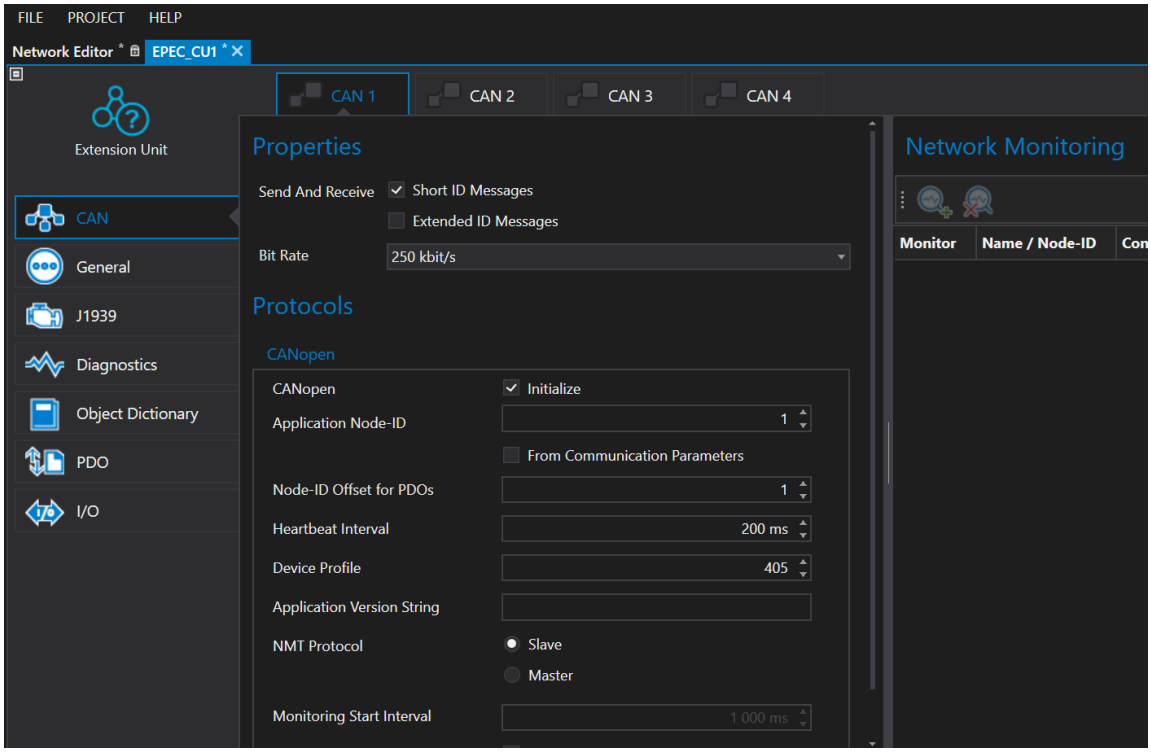
### 4.1.1 CAN

CAN (Controller Area Network) on elektronisten ohjausyksiköiden (Electronic Control Unit, ECU), mikrokontrollerien ja muiden laitteiden viestiliikennettä varten kehitetty protokolla. CAN-väylä koostuu fyysisellä tasolla kahdesta johdosta, jotka välittävä tietoa laitteiden välillä. (Javatpoint 2021) CANopen on CAN:iin pohjautuva protokolla, joka standardoi laitteiden ja sovellusten välisen viestiliikenteen. CANopen täydentää CAN-protokollan OSI-mallin mukaiseksi (kuva 11). (National Instruments 2022.)




KUVA 11. CAN ja CANopen OSI-mallissa (National Instruments 2022).

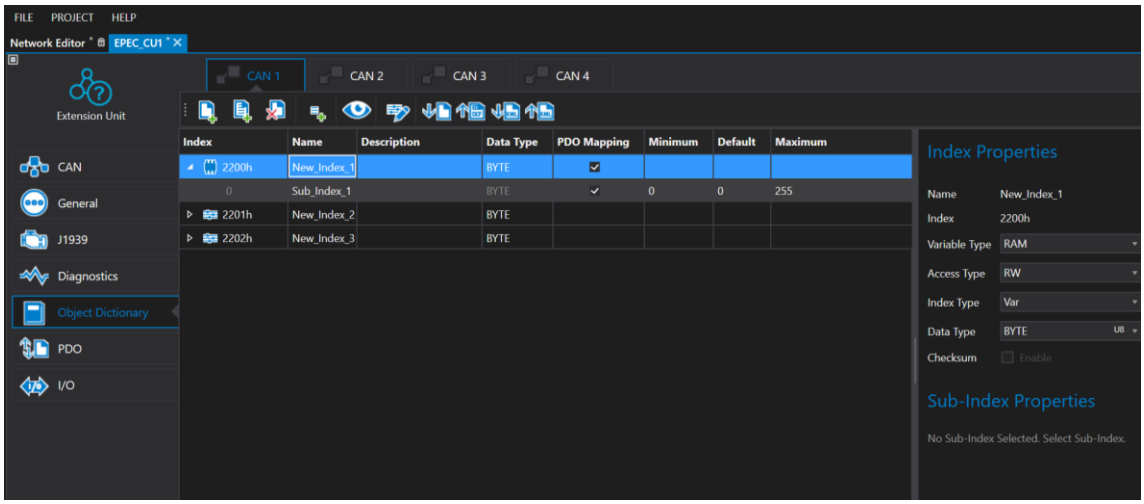
CAN-välilehdellä (kuva 12) käyttäjä voi määrittää mm. CANopen-parametrit ja tarkastella väylään liitetyn verkon liikennettä. Käyttäjä voi valita näkymän ylä-osasta CAN-väylän, jonka näkymää haluaa käsitellä.




KUVA 12. CAN-konfiguraatiovälilehti Multitool Creatorissa.

### 4.1.2 Object Dictionary

Object Dictionary (OD) on CANopen-protokollan applikaatiokerrokseen kuuluva konfiguraatio- ja prosessidatan varastointiin käytettävä datataulukko. Object Dictionary on välttämätön kaikille CANopen -laitteille. CANopen -standardin mukaan OD-indeksin koko on 16 bittiä ja ali-indeksien koko on 8 bittiä. Jokaisella indeksillä voi olla maksimissaan 256 ($2^8$) ali-indeksiä ja indeksejä voi olla OD:ssä 65536 ($2^{16}$) kappaletta. Joillekin yksittäisille indekseille ja indeksialueille on standardissa määritellyt parametrit, kun taas joihinkin käyttäjä voi itse vapaasti määritellä indeksejä ja ali-indeksejä. Jotta laitteen voidaan katsoa olevan CANopen -laite, sen OD:ssa tulee olla tietyt standardissa määritellyt pakolliset indeksit. (National Instruments 2022.) Multitool Creatorin Object Dictionary -välilehdellä (kuva 13) käyttäjä voi lisätä laitekohtaisia ennalta määriteltyjä (pre-defined) indeksejä tai luoda omia, kustomoituja indeksejä, joille käyttäjä voi valita haluamansa ominaisuudet. Object Dictionary -välilehdeltä indekseille voi asettaa PDO-kartoituksen (PDO mapping), jolloin indekseistä luodaan muuttujat PDO-välilehdelle ja ne voidaan sijoittaa PDO-viestiin.

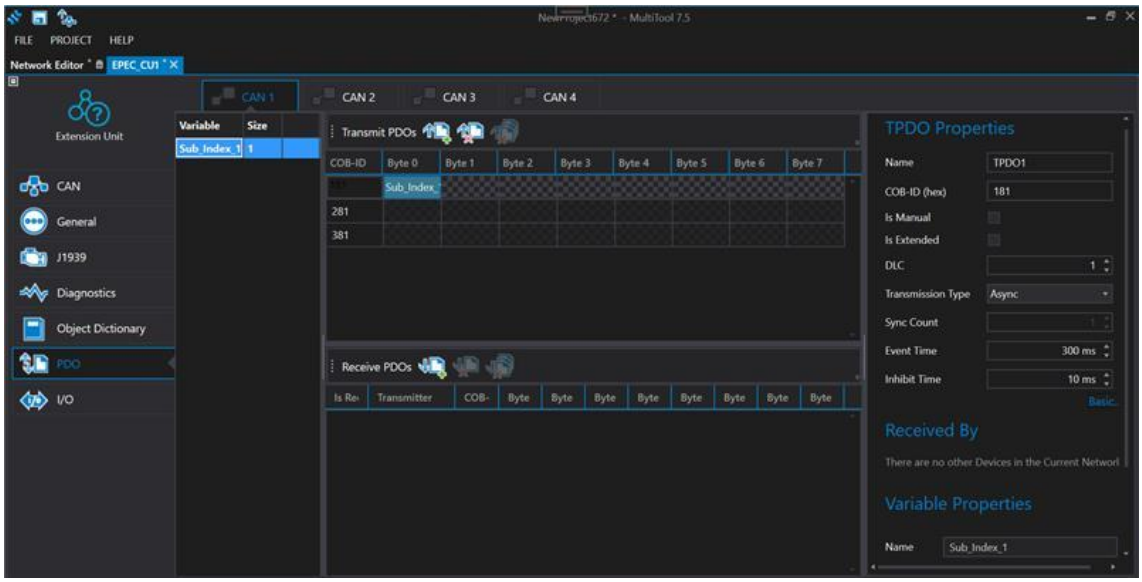



KUVA 13. Object Dictionary -konfiguraatiovälilehti Multitool Creatorissa.

### 4.1.3 PDO

PDO (Process Data Object) tarkoittaa käytännössä jollekin toiselle laitteelle lähetettävää viestiä, joka sisältää yhden CAN-kehyksen. Yksi PDO-viesti voi sisältää

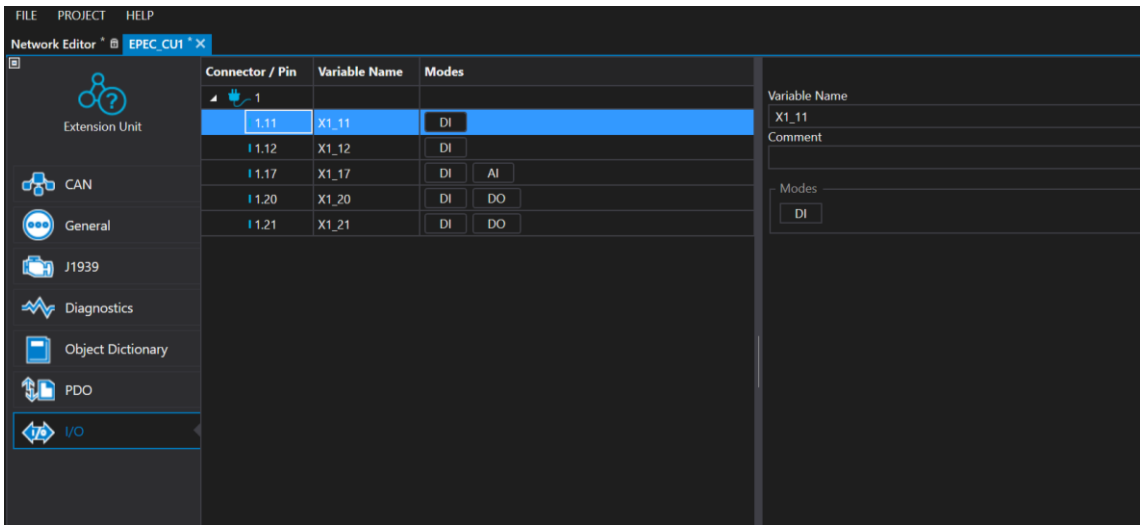
korkeintaan 8 tavua dataa. PDO:ita käytetään korkean prioriteetin hallintainformaation viestittämiseen laitteiden välillä. (CAN in Automation n.d.) Multitool Creatorin PDO-välilehdellä (kuva 14) käyttäjä voi lisätä ja poistaa PDO-viestejä sekä määrittää viestien kommunikaatioparametrit ja viestiin sisällytetyt muuttujat. PDO-viestiin voidaan sisällyttää itse luotuja muuttujia ja Object Dictionarysta kartoitettuja indeksejä.




KUVA 14. PDO-konfiguraatiovälilehti Multitool Creatorissa.

### 4.1.4 I/O

I/O (input/output) tarkoittaa yksinkertaisesti tiedon vastaanottamista ja lähettämistä jonkin tietoliikennejärjestelmän komponenttien välillä. Multitool Creatorin I/O-konfiguraatiovälilehdellä (kuva 15) käyttäjä voi tehdä määrityksiä laitteen I/O-pinneille. I/O-muuttujille voidaan valita nimi ja moodi (digitaalinen/analoginen, lähtö/tulo) sekä lisätä tarvittaessa kommentti.




KUVA 15. I/O-konfiguraatiovälilehti Multitool Creatorissa.

## 4.2 Uudet ominaisuudet

Uusina ominaisuuksina Multitool Creatoriin kehitettiin ethernet-kytkentöjen toteutus (logiikka, käyttöliittymä ja validointi), hallitsevan keskusyksikön valinta ja laajennetut CAN-väylät.

### 4.2.1 Ethernet

Laajennusyksikköön kuuluu kaksi ethernet- ja neljä CAN-väylää. Ethernetin tarkoitus laitteessa on laajentaa laitteen oma CAN- ja I/O-liikenne keskusyksikölle. Ethernet-väylien toteuttaminen Multitool Creatorissa rakennettiin jo olemassa olevaa CAN-väylien toteutusta hyödyntäen. Multitool Creatorin verkot on kehitetty toteuttamaan CAN-kytkentöjä, joten rajapintoja hyödyntämällä ethernet-kytkennöille ei tarvinnut rakentaa kokonaan uutta verkotuslogiikkaa. Samojen verkkorakenteiden käyttäminen vaati kuitenkin muutoksia esimerkiksi verkotusten validointiin, sillä on selvää, ettei CAN-väyliä voida kytkeä ethernet-verkkoon tai toisinpäin.

C#-ohjelmointikielessä rajapinnat määrittelevät jäsenet, joita rajapinnan toteuttava luokka käyttää. Itse rajapinnassa ei yleensä toteuteta jäsenten sisältöä. Rajapinnat itsessään voivat periä toisia rajapintoja ja luokat voivat toteuttaa useita

rajapintoja. Kun rajapinta perii toisen rajapinnan, se myös perii kaikki perityn rajapinnan jäsenet. Rajapinnan toteuttaminen luokassa tarkoittaa sitä, että rajapinnan jäsenille määritellään sisältö (kuva 16). (Microsoft 2022.)

```
interface ISampleInterface
{
    void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
    }

    static void Main()
    {
        // Declare an interface instance.
        ISampleInterface obj = new ImplementationClass();

        // Call the member.
        obj.SampleMethod();
    }
}
```

KUVA 16. Esimerkki rajapinnasta ja sen toteutuksesta (Microsoft 2022).

IEthernet-rajapinta (kuva 17) perii mm. ICAN-rajapinnan ja sen jäsenet, joten esimerkiksi ICAN-rajapinnassa määriteltyjä funktiota ei tarvitse määritellä IEthernet-rajapinnassa uudelleen, vaan funktioille voi määritellä uuden sisällön suoraan Ethernet-luokkaan, joka toteuttaa IEthernet-rajapinnan. IEthernet-rajapintaan tarvitsee määritellä vain uudet tarvittavat jäsenet.

```
public interface IEthernet : IList<ICAN>, INotifyCollectionChanged, IDataObject, IXmlSerializable,
    INotifyDataErrorInfo, IParentContainer, ITraverseValidations, ICAN
{
    2 references | Lindeman Johannes, 37 days ago | 1 author, 1 change | 2 work items
    IEthernet GetNextEthernet(IEthernet ethernet);
    3 references | Lindeman Johannes, 49 days ago | 1 author, 1 change | 2 work items
    List<INetwork> GetNetworkChain(IEthernet ethernet);
    2 references | Lindeman Johannes, 37 days ago | 1 author, 1 change | 2 work items
    void SplitNetworkChain();
}
```

KUVA 17. IEthernet-rajapinta.

Ethernet-luokkaa varten toteutettiin sille ensin isäntäluokka Gateway ja isäntäluokalle rajapinta IGateway (kuva 18), joka perii mm. ICollection-rajapinnan. Perimällä nämä rajapinnat, IGatewayn toteuttava Gateway-luokka toimii rajapinnalle määritetyn tyypin kokoelmana sekä mahdollistaa ja helpottaa tiedonvälitystä, kun

kokoelman jäsenissä tapahtuu muutoksia. ICollection-rajapinta toteuttaa geneerisen tyypin kokoelman, eli kokoelman tyyppi voidaan määritellä vapaasti. Esimerkiksi tässä tapauksessa tyypiksi määriteltiin ICollection<IEthernet>, jolloin kokoelman jäsenten tulee toteuttaa IEthernet-rajapinta. ICollection-rajapinta määrittelee geneeristen kokoelmien muokkaamiseen käytettävät metodit, kuten kokoelmaan lisääminen ja kokoelmasta poisto. (Microsoft n.d.)

```
public interface IGateway : ICollection<IEthernet>, INotifyCollectionChanged,
    INotifyDataErrorInfo, IParentContainer,
    INotifyPropertyChanged
{
    1 reference | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    IProject Project { get; }
    98 references | Lindeman Johannes, 71 days ago | 1 author, 1 change | 6 work items
    ObservableCollection<INetwork> NetworkChain { get; set; }
    7 references | Lindeman Johannes, 50 days ago | 1 author, 1 change | 2 work items
    ObservableCollection<ICAN> ExtensionCANs { get; set; }
    8 references | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    IDevice Device { get; set; }
    2 references | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    string Name { get; set; }
    1 reference | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    bool DisableUi { get; set; }
    1 reference | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    bool IsUnsupported { get; set; }
    6 references | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    bool IsExtended { get; set; }
    14 references | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    IDevice Master { get; set; }
    2 references | Lindeman Johannes, 71 days ago | 1 author, 1 change | 6 work items
    IEthernet FindEthernet(IDevice device, int index);
    2 references | Lindeman Johannes, 82 days ago | 1 author, 1 change | 2 work items
    void Initialize(IDevice device);
}
```

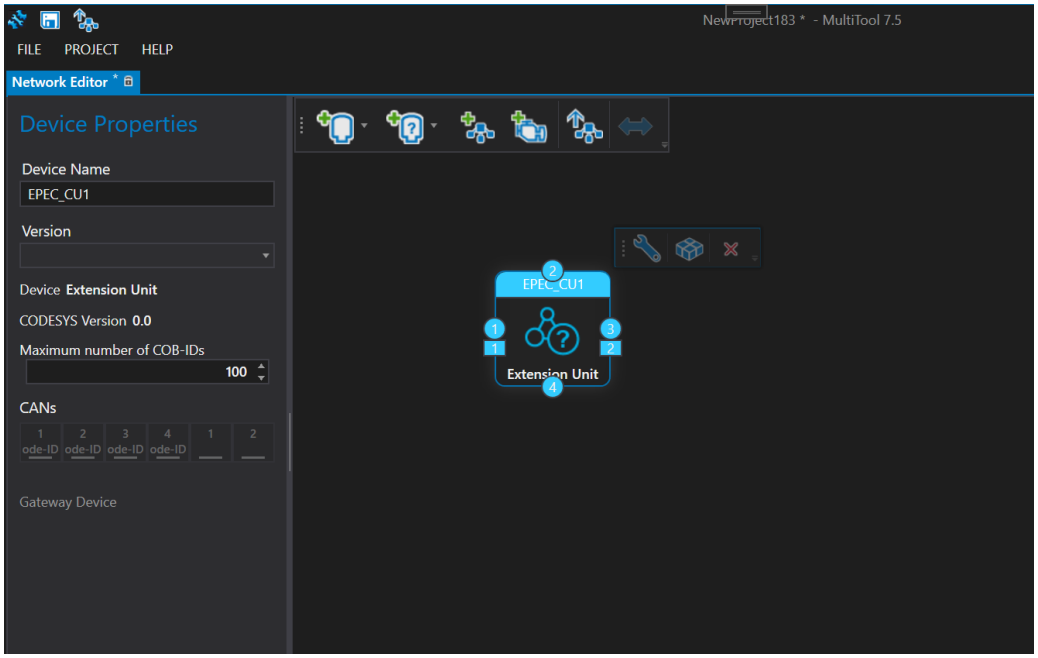
KUVA 18. IGateway-rajapinnan määrittely.

### 4.2.2 Ethernet käyttöliittymässä

Ethernet-väylien – kuten myös CAN-väylien – käyttöliittymätoiminnallisuuksien toteutuksessa hyödynnettiin ”Windows.UI.Xaml.Controls.Primitives”-nimiavaruuteen kuuluvaa Thumb-luokkaa. Thumb-elementin tärkeimpiä ominaisuuksia on komponentin raahaamisen mahdollistaminen käyttöliittymässä. (Microsoft n.d.)

CAN-väyliä edustetaan käyttöliittymästä pallon muotoisilla thumb-elementeillä, joiden sijaintia käyttäjä voi hallita raahaamalla laitteen reunoja pitkin. Raahaamalla elementtiä laitteesta ulospäin, verkkoeditoriin piirtyy viiva, jonka raahaaminen joko verkkoon tai toisen laitteen CAN-väylään yhdistää laitteen verkkoon. Nämä käyttöliittymän perustoiminnot toteutettiin myös Ethernet-väylille. Ethernet-

väylät eroavat käyttöliittymässä CAN-väylistä muotonsa puolesta, kuten ne eroavat reaalimaailmassakin. Ethernet-väylistä luodaan laitteen reunoille suorakaiteen muotoiset Thumb-elementit (kuva 19).




KUVA 19. Laajennusyksikkö Multitool Creatorin käyttöliittymässä.

### 4.2.3 Verkkojen ominaisuudet

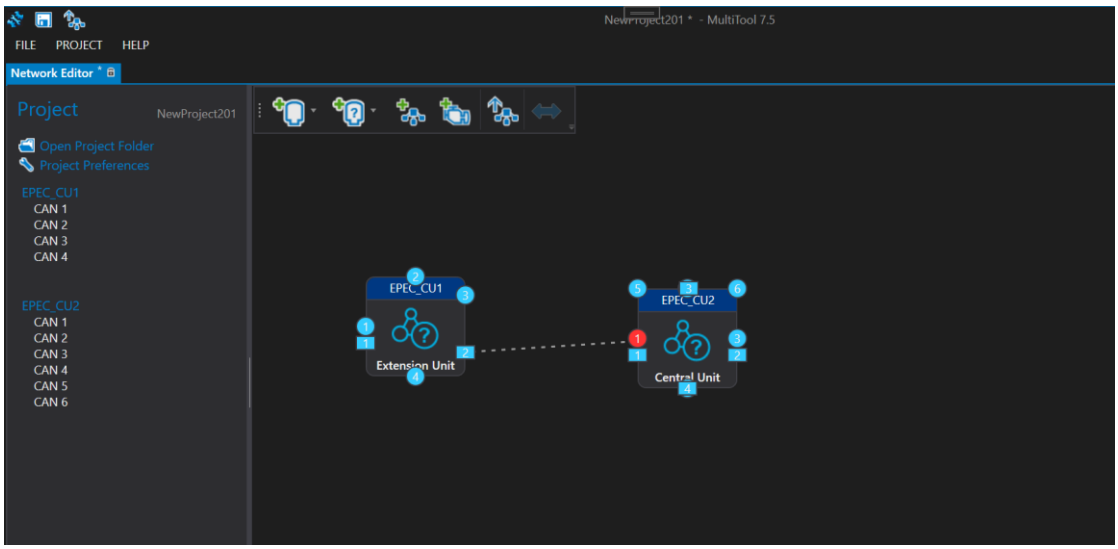
Koska IEthernet-rajapinta toteuttaa ICAN-rajapinnan, rajapintojen periytyvyyttä hyödyntämällä Ethernet-verkkoihin voitiin käyttää vanhaa CAN-kytkentöjä varten toteutettua Network-luokkaa. Network-luokka hyödyntää ObservableCollection-luokkaa, jossa collectionin tyypiksi on määritelty ICAN, eli kokoelmaan voidaan lisätä jäseniä, jotka toteuttavat ICAN-rajapinnan.

Vaikka sekä Ethernet- että CAN-luokka toteuttavat saman rajapinnan, ei niiden kuitenkaan voida sallia yhdistyä toisiinsa tai samaan verkkoon, sillä se ei ole reaalimaailmassakaan mahdollista. Väärien kytkentöjen estäminen toteutettiin validointimetodilla (kuva 20), joka keskeyttää vääränlaiset kytkennät ja asettaa käyttäjälle esitettävän virheviestin, jossa käyttäjälle kerrotaan, mitä tehtiin väärin.

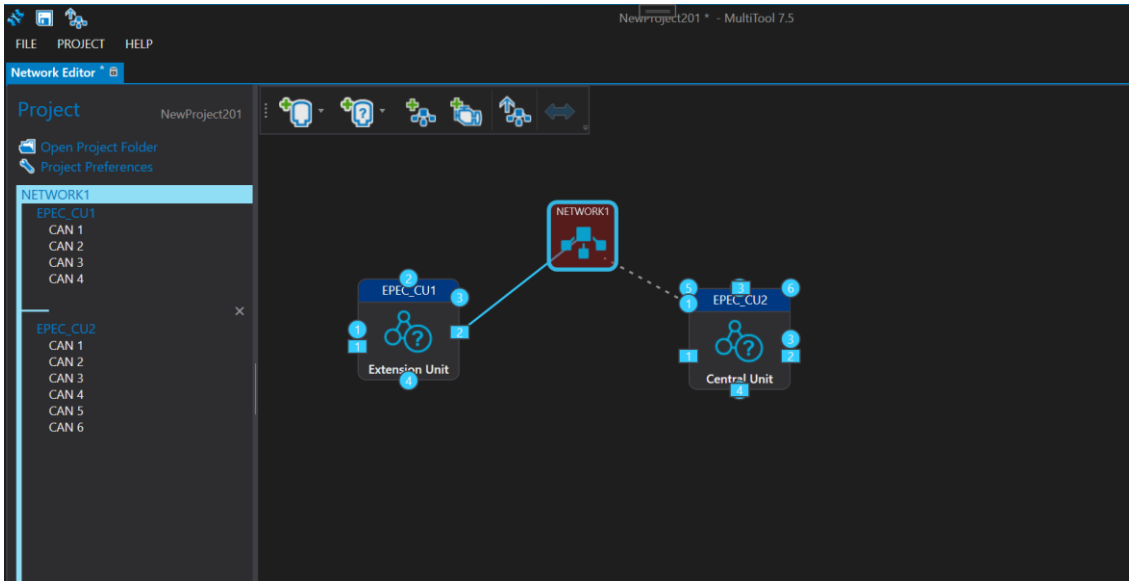
```
public CanConnectResult CanConnect(IConnectableEditorNode target)
{
    if (ReferenceEquals(this, target) || target is EthernetEditorNode)
    {
        if (target is EthernetEditorNode)
        {
            var ethernet = this.Ethernet as IEthernet;
            var targetEthernet = (target as EthernetEditorNode).Ethernet as IEthernet;
            if (ethernet != null && targetEthernet != null && !ReferenceEquals(ethernet.Parent, targetEthernet.Parent))
            {
                return (CanConnectResult)true;
            }
        }
        return (CanConnectResult)false;
    }
    var network = target as NetworkEditorNode;
    if (network != null)
    {
        var ethernet = this.Ethernet as IEthernet;
        if (ethernet != null)
        {
            if ((network.Connection as EthernetEditorNode)?.Ethernet != null)
            {
                if (network.Network.Any(a => a.Device.Guid == ethernet.Device.Guid))
                {
                    return new CanConnectResult(false, "Network can hold only one ethernet connection from each device...");
                }
                if (network.Network.Count > 1)
                {
                    return new CanConnectResult(false, "Ethernet Network can hold only two ethernet connections...");
                }

            }
            if (network.Network.Any(a => a is CAN))
            {
                return new CanConnectResult(false, "Can't connect to CAN network.");
            }
            if (network.Network.OfType<IEthernet>().Any(a =>
            {
                var device = a.GetParentOfType<IDevice>();
                var targetDevice = ethernet.GetParentOfType<IDevice>();
                if (device != null && targetDevice != null)
                {
                    return ReferenceEquals(device, targetDevice);
                }
                return false;
            }))
            {
                return new CanConnectResult(false, "Another Ethernet from this device is already connected to that network...");
            }
        }
        else
        {
            return (CanConnectResult)false;
        }
    }
    if (target is CANEditorNode)
    {
        return new CanConnectResult(false, "Ethernet can not be connected to CAN...");
    }
    return (CanConnectResult)true;
}
```

KUVA 20. Kytkentöjen validoinnin ohjelmalogiikkaa.

Kun käyttäjä pyrkii suorittamaan vääränlaista kytkentää, kytkennän virheellisyydestä indikoidaan punaisella värillä kohde-elementissä, oli se sitten CAN-väylä (kuva 21) tai verkko (kuva 22).




KUVA 21. Virheellisen ethernet-CAN-kytkennän indikointi kohde-elementin värimuutoksella.




KUVA 22. Virheellisen verkkokytkennän indikointi kohde-elementin värimuutoksella.

Kun kytkentä on todettu ohjelmakoodissa validiksi, kutsutaan kytkettävän ethernetin AddToNetwork-metodia, joka saa parametrinaan verkon, johon ethernet lisätään. Ennen kuin ethernet lisätään verkkoon, varmistetaan vielä metodin sisällä kytkennän lähtökohtien oikeellisuus ja mikäli moitittavaa ei löydy, asetetaan parametrina saatu verkko kyseisen ethernetin verkoksi (kuva 23).

```
/// <summary>
/// Adds an ethernet to a network if connection is valid. Calls for network chain updates when appropriate.
/// </summary>
/// <param name="network"></param>
/// <returns></returns>
62 references | Lindeman Johannes, 64 days ago | 1 author, 9 changes | 13 work items
public bool AddToNetwork(INetwork network)
{
    if (network == null)
    {
        return false;
    }
    if (this.Device != null)
    {
        var otherEthernets = from ethernet in this.Device.Gateway
                             where !ReferenceEquals(ethernet, this)
                             select ethernet;

        foreach (var ethernet in otherEthernets)
        {
            if (network.Contains(ethernet))
            {
                throw new CanConnectedException("Same devices' other ethernet already connected to network");
            }
        }
    }
    if (this.Network != null)   // only one connection per Ethernet
    {
        throw new CanConnectedException("Ethernet already connected to network");
    }
    if (network.Any(a => (a is CAN)))   // only one connection per Ethernet
    {
        throw new CanConnectedException("Can't connect to CAN network"); // Network can only hold one type of connection
    }
    if (network.Count > 1)   // cant connect if two devices already in network.
    {
        throw new CanConnectedException("Ethernet network can only have two connections"); // Network can only hold one type of connection
    }

    this.Network = network;
```

KUVA 23. AddToNetwork-metodin validointivaihe.

Laajennusyksikön verkkojen ketjutusominaisuutta varten luotiin ObservableCollection-tyyppinen NetworkChain-ominaisuus, jonka avulla voidaan hallita ja tarkastella kunkin laitteen verkostoa. NetworkChain-ominaisuus sijoitettiin Ethernet-luokan Gateway-isäntäluokkaan. AddToNetwork-metodissa validointivaiheen jälkeen tarkastetaan, onko verkossa jo toinen laite kytkettynä. Mikäli on, lisätään verkon toisen laitteen verkosto käsiteltävän laitteen verkostoon (kuva 24).

```
if (network.Any())
{
    // Add connected network chain to device
    foreach (var net in network.FirstOrDefault().Device.Gateway.NetworkChain)
    {
        if (!this.Device.Gateway.NetworkChain.Contains(net))
            this.Device.Gateway.NetworkChain.Add(net);
    }
}
```

KUVA 24. Käsiteltävän laitteen verkoston päivitys.

Kun käsiteltävän laitteen verkostoon on lisätty mahdollisen toisen laitteen verkosto, täytyy tarkistaa käsiteltävän laitteen toinen ethernet-väylän kytkentä (kuva 25). Mikäli laitteen toinen ethernet-väylä on kytketty verkkoon, täytyy kaikki laitteen verkostoon kuuluvien laitteiden verkostot päivittää kutsumalla UpdateNetworkChain-metodia.

```
var otherEthernet = this.Device.Gateway.FirstOrDefault(a => !ReferenceEquals(a, this));
if (otherEthernet.Network != null && otherEthernet.Network.Any())
{
    // Check all devices' network chains and update if necessary
    this.UpdateNetworkChain();
}
```

KUVA 25. Käsiteltävän laitteen toisen ethernet-väylän kytkennän tarkistus.

UpdateNetworkChain-metodissa (kuva 26) lisätään kaikki käsiteltävän laitteen verkostoon kuuluvat laitteet listaan ja päivitetään laitteiden verkostot.

```
/// <summary>
/// Scans all devices in the network chain and updates them accordingly.
/// </summary>
1 reference | Lindeman Johannes, 74 days ago | 1 author, 2 changes | 2 work items
private void UpdateNetworkChain()
{
    List<IDevice> devices = new List<IDevice>();
    foreach (var net in this.Device.Gateway.NetworkChain)
    {
        foreach (var ethernet in net)
        {
            if (!devices.Contains(ethernet.Device))
                devices.Add(ethernet.Device);
        }
    }
    foreach (var device in devices)
    {
        if (device.Gateway.NetworkChain.Count() < this.Device.Gateway.NetworkChain.Count())
        {
            var nets = this.Device.Gateway.NetworkChain.Where(a => !device.Gateway.NetworkChain.Contains(a)).ToList();
            foreach (var net in nets)
            {
                device.Gateway.NetworkChain.Add(net);
            }
        }
    }
}
```

KUVA 26. UpdateNetworkChain-metodi.

Kun laitteet on päivitetty, palataan AddToNetwork-metodiin ja sen viimeiseen vaiheeseen (kuva 27), jossa lisätään käsiteltävä ethernet-väylä verkkoon, lisätään käsiteltävä verkko oikeisiin verkostoihin ja määritellään vapaille yksiköille hallitseva yksikkö kutsumalla AssignMaster-metodia. Tässä vaiheessa kutsutaan myös OnPropertyChanged-metodia, jotta hallitsevan keskusyksikön valinnasta vastaava MultiValueConverter päivittyy ajan tasalle ja osaa tarjota valikkoon oikeat keskusyksiköt. IMultiValueConverter on WPF:ssä käytettävä sidotun datan kääntämiseen käytettävä rajapinta, jolla voidaan kääntää lähdedata haluttuun muotoon (Microsoft n.d.).

```
        network.Add(this);
        // Add the new network to the chain
        if (!this.Device.Gateway.NetworkChain.Contains(network))
            this.Device.Gateway.NetworkChain.Add(network);

        // Add the new network to other chains
        foreach (var net in this.Device.Gateway.NetworkChain)
        {
            foreach (var ethernet in net)
            {
                if (!ethernet.Device.Gateway.NetworkChain.Contains(network))
                    ethernet.Device.Gateway.NetworkChain.Add(network);

                // Handle master assignments for idle gateways
                if (ethernet.Device.IsCentralUnit())
                    AssignMaster((IEthernet)ethernet);

                (ethernet as Ethernet).OnPropertyChanged("Network");
            }
        }

        if (this.Network is ObservableCollection<IEthernet> xxy)
        {
            xxy.CollectionChanged += this.Network_CollectionChanged;
        }
        return true;
    }
```

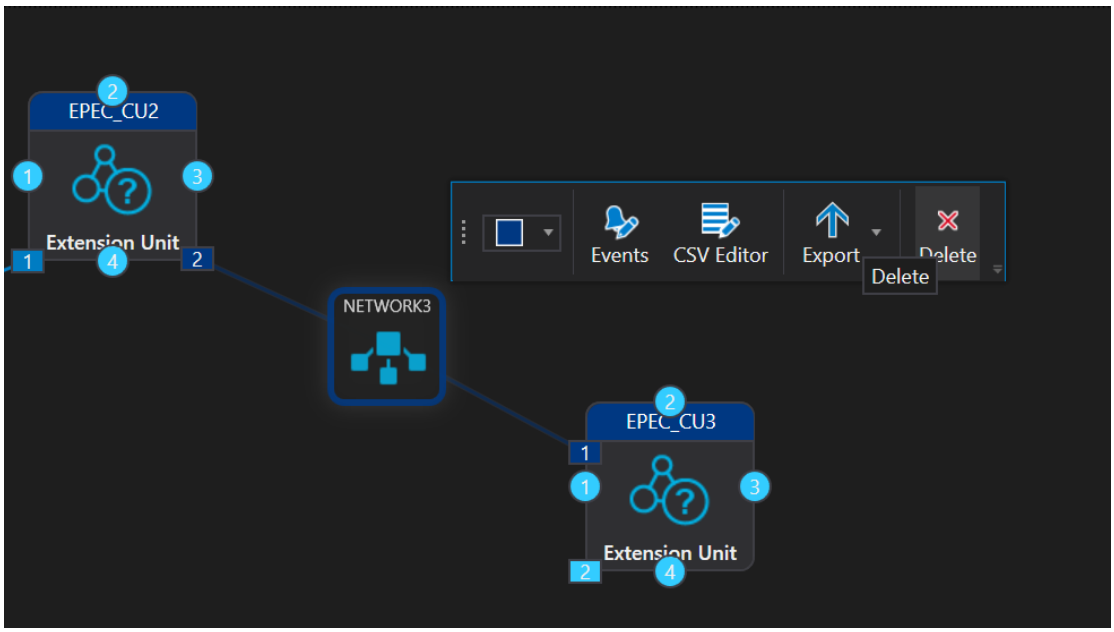
KUVA 27. AddToNetwork-metodin viimeinen vaihe: käsiteltävän väylän lisääminen verkkoon ja verkon lisääminen verkostoihin.

AssignMaster-funktiossa parametrina saadun keskusyksikön ethernetin verkosto skannataan ja mikäli verkostosta löytyy laite, jolla ei ole hallitsevaa keskusyksikköä, asetetaan parametrina saadun ethernetin isäntälaite hallitsevaksi keskusyksiköksi (kuva 28).

```
    /// <summary>
    /// Assigns Master for Gateway if there is a Central Unit in the remaining chain.
    /// </summary>
    /// <param name="ethernet"></param>
    3 references | Lindeman Johannes, 62 days ago | 1 author, 5 changes | 8 work items
    private void AssignMaster(IEthernet ethernet)
    {
        foreach (var net in ethernet.Device.Gateway.NetworkChain)
        {
            foreach (var e in net)
            {
                if (e.Device.Gateway.Master == null)
                {
                    e.Device.Gateway.Master = ethernet.Device;
                }
            }
        }
    }
```

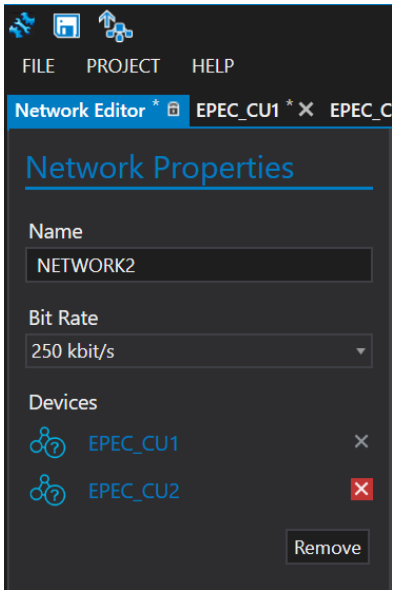
KUVA 28. AssingMaster-funktio.

Yksikön poistamisen verkosta voi tehdä Multitool Creatorissa kahdella tavalla. Ensimmäinen tapa on poistaa koko verkko verkkoeditorin valikosta (kuva 29).



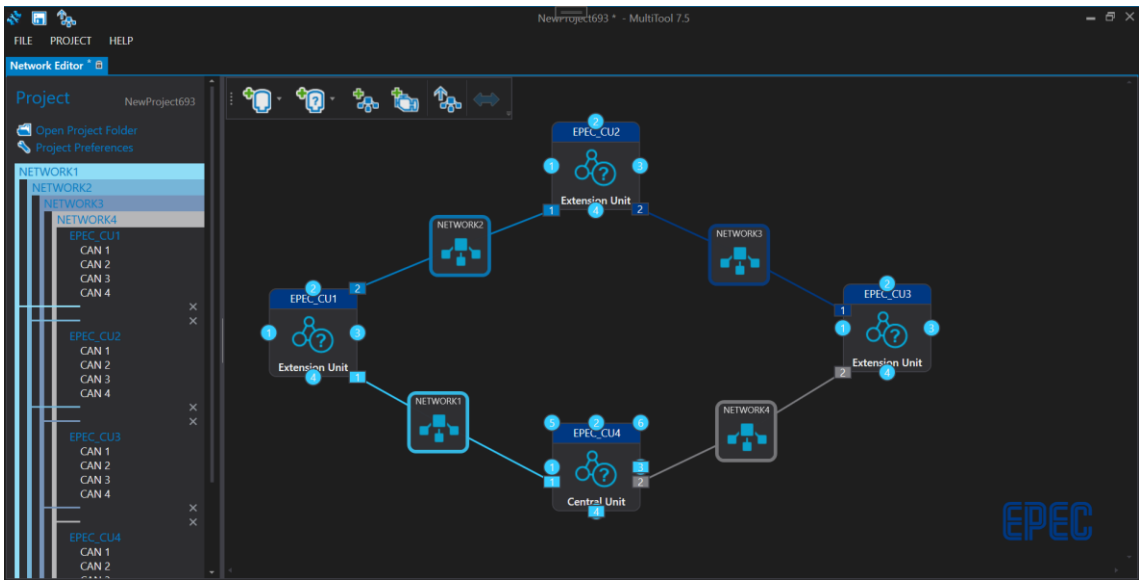

KUVA 29. Verkon poistaminen Multitool Creatorissa.

Toinen tapa on poistaa kytkentä verkon ominaisuuksista löytyvästä väylävalikosta (kuva 30).




KUVA 30. Verkon väylävalikko Multitool Creatorissa.

Kun yksikkö poistetaan verkosta, on käytävä läpi erilaiset tilanteet, joissa kytkennän poisto vaikuttaa verkoston muihin laitteisiin. Esimerkiksi jos ketjutetut laitteet muodostavat suljetun silmukan (kuva 31), kytkennän poisto ei vaikuta verkostoon merkittävällä tavalla, mutta muissa tapauksissa verkosto puolittuu.




KUVA 31. Ketjutetut laitteet suljetussa silmukassa.

Ohjelmakoodissa suljetun silmukan havaitsemiseen luotiin IsLooped-metodi (kuva 32), joka tarkistaa, onko kaikilla verkoston laitteilla enemmän kuin yksi kytketty ethernet-väylä.

```
/// <summary>
/// Checks if network chain is a closed loop.
/// </summary>
/// <param name="chain"></param>
/// <returns></returns>
1 reference | Lindeman Johannes, 81 days ago | 1 author, 2 changes | 4 work items
private bool IsLooped(List<INetwork> chain)
{
    // The loop is closed if there are no devices in the chain with only one network connection.
    if (chain.All(a => a.All(b => b.Device.Gateway.Count(c => c.Network != null) > 1)))
        return true;
    else
        return false;
}
```

KUVA 32. IsLooped-metodi.

Kytkennän poistoon käytetyssä RemoveFromNetwork-metodissa (kuva 33) poistetaan ensin käsiteltävä ethernet-väylä verkosta, jonka jälkeen tarkistetaan muodostaako ketjutetut laitteet suljetun silmukan. Mikäli laitteet eivät muodosta silmukkaa, katkaistaan verkosto puoliksi SplitNetworkChain-metodilla.

```
/// <summary>
/// Removes ethernet from network.
/// </summary>
15 references | Lindeman Johannes, 62 days ago | 1 author, 10 changes | 14 work items
public void RemoveFromNetwork()
{
    if (this.Network != null)
    {
        this.Network.Remove(this);

        // Network chain is cut only if the chain is not a closed loop.
        // If the chain is a loop, its enough to just remove the current network from it.
        var thisChain = this.Device.Gateway.NetworkChain.ToList();
        var otherChain = this.Network.FirstOrDefault(a => !ReferenceEquals(a, this))?.Device.Gateway.NetworkChain.ToList();

        if (otherChain != null && !IsLooped(thisChain))
            this.SplitNetworkChain();
```

KUVA 33. RemoveFromNetwork-metodin ensimmäinen vaihe.

SplitNetworkChain-metodissa (kuva 34) jaetaan toisistaan irti kytketyt ketjut kahdeksi ketjuksi ja poistetaan kunkin ketjun laitteiden verkostoista toisen ketjun verkot. Jälleen joka kerta, kun verkosto muuttuu, kutsutaan OnPropertyChanged-metodia, jotta hallitsevan keskusyksikön valinnasta vastaava MultiValueConverter laukeaa ja lista verkoston keskusyksiköistä päivittyy ajan tasalle.

```
public void SplitNetworkChain()
{
    List<INetwork> firstHalfChain = this.GetNetworkChain(this);
    List<INetwork> secondHalfChain = this.GetNetworkChain(this.Device.Gateway.FirstOrDefault(a => (!ReferenceEquals(a, this) && a.Network != null)));

    foreach (var network in firstHalfChain)
    {
        foreach (var ethernet in network)
        {
            if (secondHalfChain.Any(a => a.Any(b => (b.Device == ethernet.Device.Gateway.Master))))
            {
                ethernet.Device.Gateway.Master = null;
            }
            ethernet.Device.Gateway.NetworkChain.RemoveAll(a => secondHalfChain.Contains(a));
            (ethernet.Device.Gateway as Gateway).OnPropertyChanged("NetworkChain");
            if (ethernet.Device.IsCentralUnit())
            {
                AssignMaster((IEthernet)ethernet);
            }
        }
    }
    foreach (var network in secondHalfChain)
    {
        foreach (var ethernet in network)
        {
            if (firstHalfChain.Any(a => a.Any(b => (b.Device == ethernet.Device.Gateway.Master))))
            {
                ethernet.Device.Gateway.Master = null;
            }
            ethernet.Device.Gateway.NetworkChain.RemoveAll(a => firstHalfChain.Contains(a));

            // Remove current network from chain
            ethernet.Device.Gateway.NetworkChain.Remove(this.Network);
            (ethernet.Device.Gateway as Gateway).OnPropertyChanged("NetworkChain");
            if (ethernet.Device.IsCentralUnit())
            {
                AssignMaster((IEthernet)ethernet);
            }
        }
    }
    // If there is no Core in the chain, Gateway's master is set to null
    if (!secondHalfChain.Any(a => a.Any(b=> b.Device.IsCentralUnit())))
    {
        foreach (var network in secondHalfChain)
        {
            foreach (var ethernet in network)
            {
                ethernet.Device.Gateway.Master = null;
            }
        }
    }
    if (!firstHalfChain.Any(a => a.Any(b => b.Device.IsCentralUnit())))
    {
        foreach (var network in firstHalfChain)
        {
            foreach (var ethernet in network)
            {
                ethernet.Device.Gateway.Master = null;
            }
        }
    }
}
```

KUVA 34. SplitNetworkChain-metodi.

Kun verkostot on tarvittaessa päivitetty, palataan RemoveFromNetwork-metodin viimeiseen vaiheeseen, jossa poistettava verkko poistetaan käsiteltävän laitteen jäljellä olevasta verkostosta ja käsiteltävän ethernet-väylän verkolle asetetaan null-arvo (kuva 35).

```
// Remove the selected network from all chains
foreach (var net in this.Device.Gateway.NetworkChain.ToList())
{
    foreach (var ethernet in net)
    {
        if (ethernet.Device.Gateway.NetworkChain.Contains(this.Network))
        {
            ethernet.Device.Gateway.NetworkChain.Remove(this.Network);
            (ethernet as Ethernet).OnPropertyChanged("Network");
        }
    }
}

if (this.Device.Gateway.NetworkChain.Contains(this.Network))
{
    this.Device.Gateway.NetworkChain.Remove(this.Network);
}

this.Network = null;
this.IsExtended = false;
```

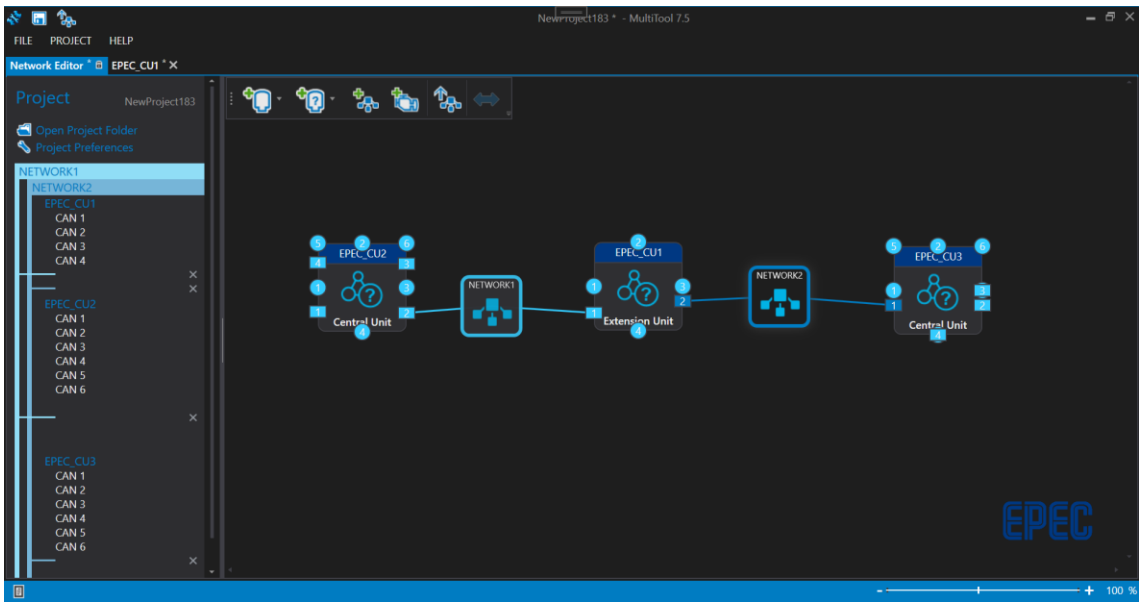
KUVA 35. RemoveFromNetwork-metodin viimeinen vaihe.

### 4.2.4 Suhde keskusyksikköön

Laajennusyksikön tarkoitus on toimia keskusyksikön jatkeena. Keskusyksikkö on laite, josta luodaan koodipohja CODESYS-projektiin ja se käyttää laajennusyksikön CAN- ja I/O-liikennettä kuin ne olisivat sen omia. CAN- ja I/O-määritykset laajennetuille väylille konfiguroidaan kunkin laajennusyksikön asetuksissa. Useita laajennusyksiköitä voidaan ”ketjuttaa” ethernetin välityksellä samaan verkostoon, jolloin keskusyksikkö saa käyttöönsä useita eri laajennusyksiköitä ja kaikki niiden CAN-väylät ja I/O:n.

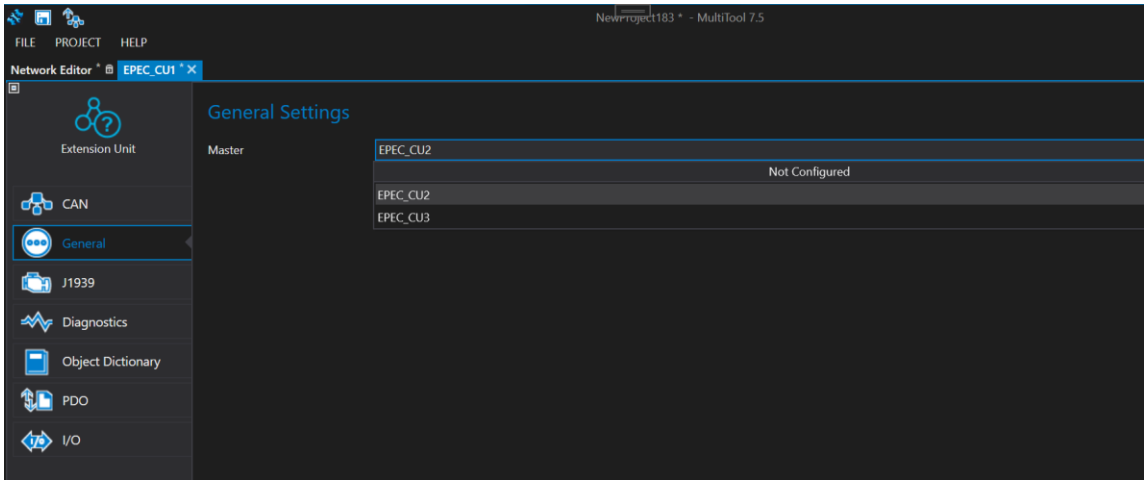
### 4.2.5 Hallitsevan keskusyksikön valinta

Kun samaan verkostoon on kytketty useampi kuin yksi keskusyksikkö (kuva 36), sallitaan käyttäjän valita kullekin laajennusyksiköllle hallitseva keskusyksikkö (master).



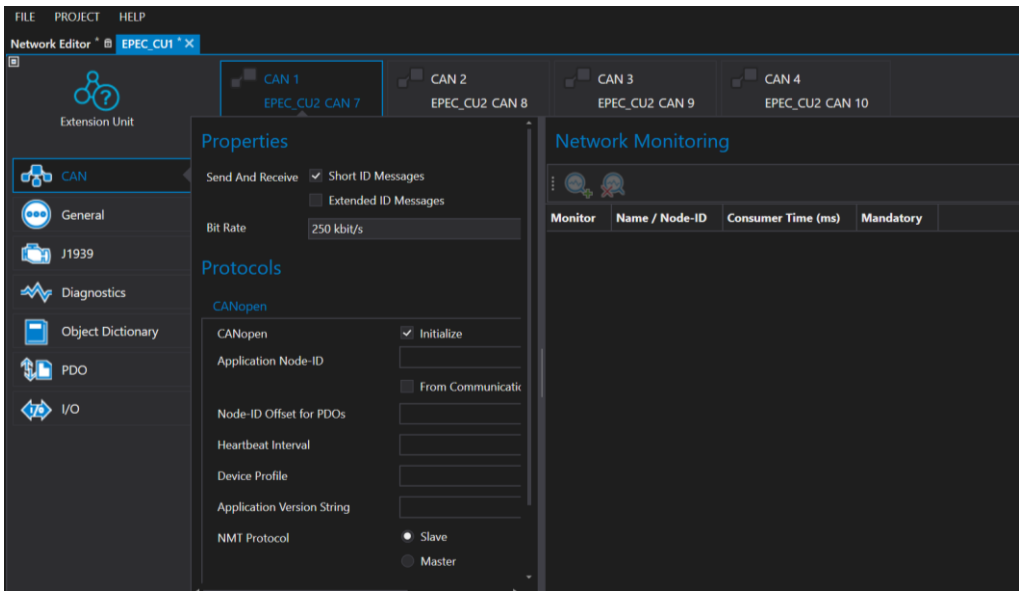

KUVA 36. Useamman keskusyksikön verkosto.

Hallitsevan yksikön valinta tapahtuu alasvetovalikosta laajennusyksikön General-välilehdellä (kuva 37). Vaihtoehdoksi tarjotaan myös ”not configured”, jonka valitsemalla laajennusyksikkö ei laajenna itseään minkään keskusyksikön käyttöön.




KUVA 37. Laajennusyksikön General-välilehti ja masterin valinta.

Kun laajennusyksiköllе valitaan hallitseva keskusyksikkö, CAN-välilehden käyttöliittymässä näytetään CAN-väylien lokaalit numerot sekä minkä laitteen CAN-väyliksi CAN-väylät on laajennettu ja laajennetut numerot. Kuvassa 38 hallitseva laite on EPEC_CU2, ja kun keskusyksiköllä itsellään on kuusi CAN-väylää, laajennusyksikön neljä CAN-väylää saavat jatketuiksi numeroikseen numerot 7-10.




KUVA 38. Laajennusyksikön CAN-välilehti, kun hallitseva keskusyksikkö on valittu.

Hallitsevan keskusyksikön valinta kaikista verkoston keskusyksiköistä toteutettiin hyödyntämällä Ethernet-luokan isäntäluokan (Gateway) NetworkChain-ObservableCollectionia sekä MultiValueConverteria, joka skannaa verkoston ja palauttaa verkoston kaikki keskusyksiköt valittavaksi laajennusyksikön General-välilehden alasvetovalikkoon (kuva 39).

```
class GetCentralUnitsFromNetworkValueConverter : IMultiValueConverter
{
    0 references | 0 changes | 0 authors, 0 changes
    public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
    {
        if (values.Length >= 2 && (values[0] is INetwork || values[1] is INetwork))
        {
            var network = (values[0] is INetwork) ? (INetwork)values[0] : (INetwork)values[1];
            if (network != null)
            {
                var returnableCans = new ObservableCollection<IDevice>();
                var list = new List<IEthernet>();

                IDevice device = null;
                var ethernet1 = (IEthernet)values[2];
                var ethernet2 = (IEthernet)values[3];
                var connectedEthernet = (ethernet1.Network != null) ? ethernet1 : ethernet2;
                if (connectedEthernet != null)
                {
                    list.Add(connectedEthernet);
                    device = connectedEthernet.Device;
                }

                if (device.IsExtensionUnit())
                {
                    foreach (var net in device.Gateway.NetworkChain)
                    {
                        foreach (var ethernet in net)
                        {
                            if (!returnableCans.Contains(ethernet.Device) && ethernet.Device.IsCentralUnit())
                                returnableCans.Add(ethernet.Device);
                        }
                    }
                }
                return returnableCans;
            }
        }
        return Enumerable.Empty<IEthernet>();
    }
```

KUVA 39. MultiValueConverter, joka palauttaa NetworkChainiin kuuluvat keskusyksiköt.

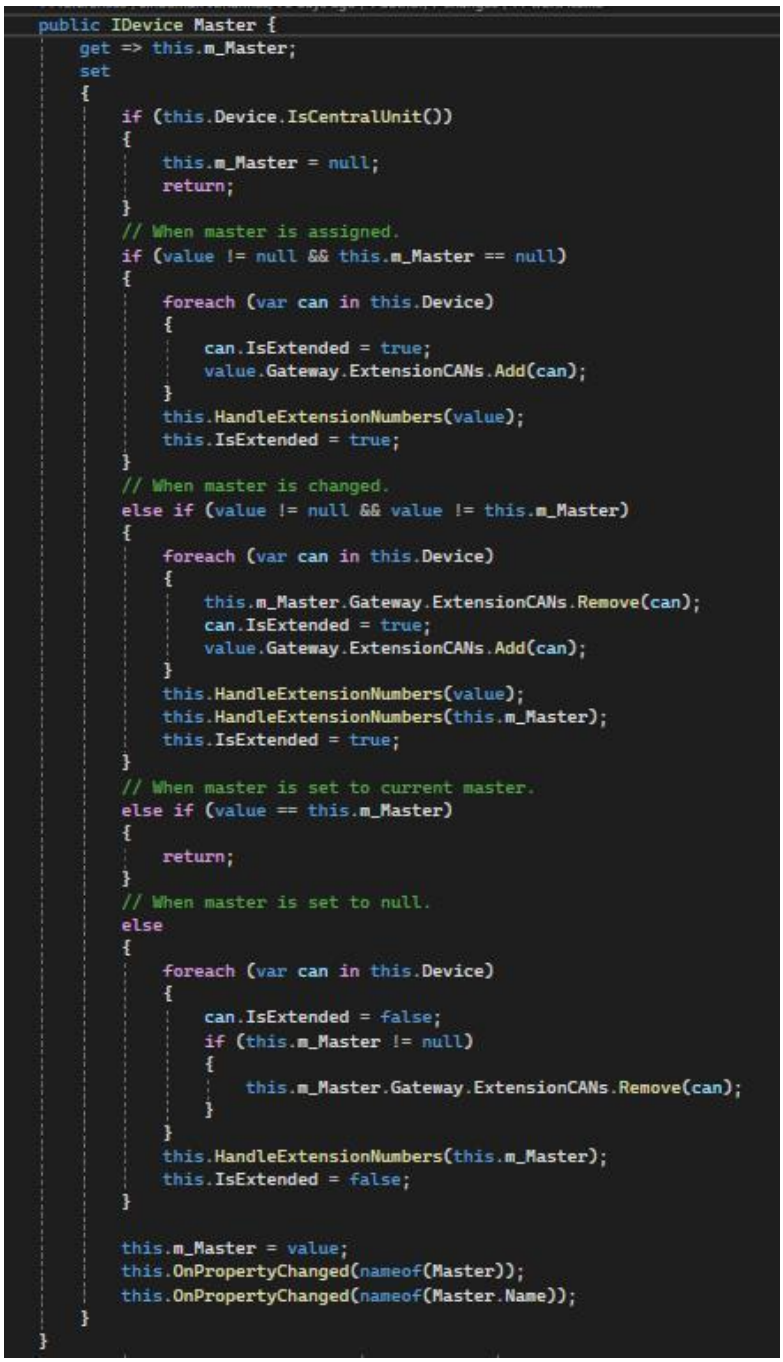
MultiValueConverterille toimitetaan datajäsenet hyödyntämällä MultiBindingia. MultiBindingin avulla MultiValueConverter saadaan myös laukaistua aina, kun jokin määritellyistä datajäsenistä muuttuu. MultiBinding-luokka mahdollistaa kohde-elementin sitomisen useaan eri dataobjektiin sekä kaksisuuntaisen tietoliikenteen sidottujen objektien ja kohde-elementin välillä (Microsoft n.d.). MultiBindingissa MultiValueConverterin datajäsenet (kuva 40) määritellään Path-parametrin avulla, johon syötetään datajäsenen polku suhteessa datakontekstiin.

```
<telerik:RadComboBox.ItemsSource>
    <MultiBinding Converter="{StaticResource GetCentralUnitsFromNetworkConverter}">
        <Binding Path="Generic.Gateway[0].Network"/>
        <Binding Path="Generic.Gateway[1].Network"/>
        <Binding Path="Generic.Gateway[0]"/>
        <Binding Path="Generic.Gateway[1]"/>
        <Binding Path="Generic.Gateway.NetworkChain"/>
    </MultiBinding>
</telerik:RadComboBox.ItemsSource>
<telerik:RadComboBox.ItemTemplate>
    <DataTemplate>
        <TextBlock>
            <TextBlock.Text>
                <Binding Path="Name" />
            </TextBlock.Text>
        </TextBlock>
    </DataTemplate>
</telerik:RadComboBox.ItemTemplate>
</telerik:RadComboBox>
```

KUVA 40. Datan välitys MultiValueConverterille MultiBindingin avulla.

### 4.2.6 CAN- ja I/O-liikenteen laajentaminen

Laajennusyksikön CAN- ja I/O-liikenteen laajentaminen toteutettiin Gateway-luokkaa hyödyntäen. Kun laajennusyksikölle valitaan ”master”, eli hallitseva keskusyksikkö, lisätään laajennusyksikön CAN-väylät keskusyksikön Gatewayn ExtensionCANs-kokoelmaan. Mikäli masteria vaihdetaan, poistetaan kyseisen laajennusyksikön CAN-väylät vanhalta masterilta ja lisätään ne uudelle (kuva 41).



```
public IDevice Master {
    get => this.m_Master;
    set
    {
        if (this.Device.IsCentralUnit())
        {
            this.m_Master = null;
            return;
        }
        // When master is assigned.
        if (value != null && this.m_Master == null)
        {
            foreach (var can in this.Device)
            {
                can.IsExtended = true;
                value.Gateway.ExtensionCANs.Add(can);
            }
            this.HandleExtensionNumbers(value);
            this.IsExtended = true;
        }
        // When master is changed.
        else if (value != null && value != this.m_Master)
        {
            foreach (var can in this.Device)
            {
                this.m_Master.Gateway.ExtensionCANs.Remove(can);
                can.IsExtended = true;
                value.Gateway.ExtensionCANs.Add(can);
            }
            this.HandleExtensionNumbers(value);
            this.HandleExtensionNumbers(this.m_Master);
            this.IsExtended = true;
        }
        // When master is set to current master.
        else if (value == this.m_Master)
        {
            return;
        }
        // When master is set to null.
        else
        {
            foreach (var can in this.Device)
            {
                can.IsExtended = false;
                if (this.m_Master != null)
                {
                    this.m_Master.Gateway.ExtensionCANs.Remove(can);
                }
            }
            this.HandleExtensionNumbers(this.m_Master);
            this.IsExtended = false;
        }

        this.m_Master = value;
        this.OnPropertyChanged(nameof(Master));
        this.OnPropertyChanged(nameof(Master.Name));
    }
}
```

KUVA 41. Laajennusyksikön master-ominaisuus.

Hallitsevan yksikön määrityksessä käydään läpi kaikki ethernet-kytkentöjä tukevan laitteen mahdolliset lähtötilanteet:

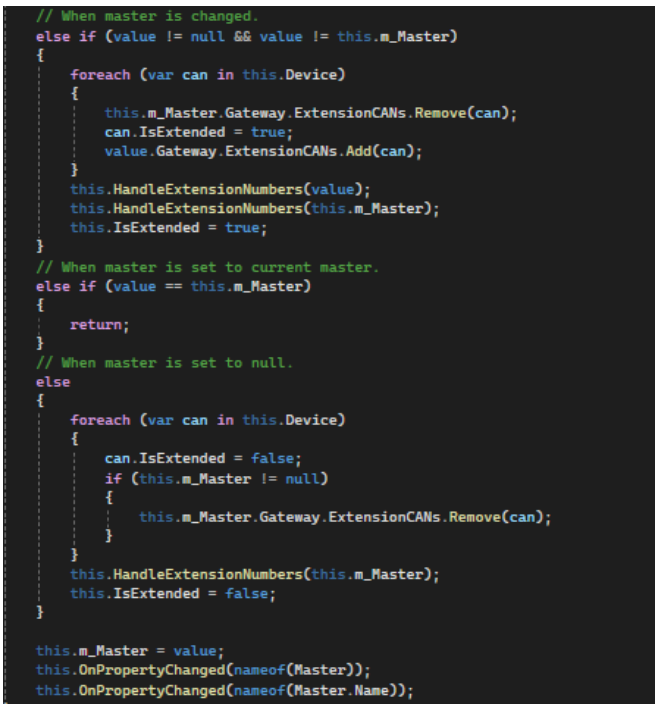
1. laite on itse keskusyksikkö
2. laitteella ei vielä ole hallitsevaa yksikköä
3. laitteella on jo hallitseva yksikkö ja se vaihdetaan
4. laitteella on hallitseva yksikkö ja se poistetaan kokonaan.

Mikäli laite on itse keskusyksikkö, hallitsevaa yksikköä ei aseteta lainkaan, koska keskusyksikön CAN- ja I/O-liikennettä ei voi laajentaa toisen keskusyksikön käyttöön. Mikäli laitteella ei ole ennestään hallitsevaa yksikköä ja sellainen määritetään, asetetaan laitteen CAN-väylät hallitsevan yksikön ExtensionCANs-kokoelmaan ja käsitellään laajennettujen CAN-väylien numerot (kuva 42).

```
set
{
    if (this.Device.IsCentralUnit())
    {
        this.m_Master = null;
        return;
    }
    // When master is assigned.
    if (value != null && this.m_Master == null)
    {
        foreach (var can in this.Device)
        {
            can.IsExtended = true;
            value.Gateway.ExtensionCANs.Add(can);
        }
        this.HandleExtensionNumbers(value);
        this.IsExtended = true;
    }
```

KUVA 42. Hallitsevan yksikön määritys, kun sellaista ei ole ennestään.

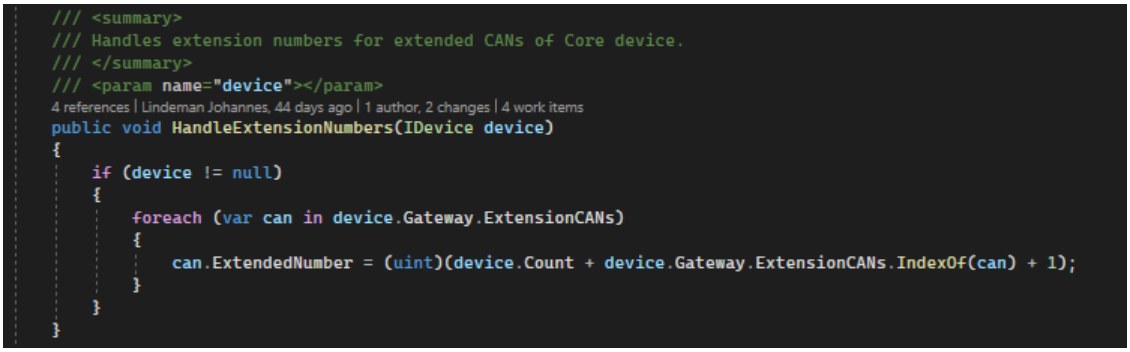
Jos hallitseva yksikkö poistetaan laajennusyksiköltä kokonaan, riittää kun poistaa vanhan hallitsevan yksikön kokoelmasta kyseisten laajennusyksikön CAN-väylät. Mikäli hallitseva yksikkö vaihdetaan toiseen, täytyy jatketut CAN-väylät poistaa vanhan hallitsevan yksikön kokoelmasta ja lisätä uuden hallitsevan yksikön kokoelmaan (kuva 43). Kun kahden hallitsevan yksikön kokoelmia on muutettu, täytyy molemmille yksiköille suorittaa jatkettujen CAN-väylien numeroiden käsittely.



```
// When master is changed.
else if (value != null && value != this.m_Master)
{
    foreach (var can in this.Device)
    {
        this.m_Master.Gateway.ExtensionCANs.Remove(can);
        can.IsExtended = true;
        value.Gateway.ExtensionCANs.Add(can);
    }
    this.HandleExtensionNumbers(value);
    this.HandleExtensionNumbers(this.m_Master);
    this.IsExtended = true;
}
// When master is set to current master.
else if (value == this.m_Master)
{
    return;
}
// When master is set to null.
else
{
    foreach (var can in this.Device)
    {
        can.IsExtended = false;
        if (this.m_Master != null)
        {
            this.m_Master.Gateway.ExtensionCANs.Remove(can);
        }
    }
    this.HandleExtensionNumbers(this.m_Master);
    this.IsExtended = false;
}

this.m_Master = value;
this.OnPropertyChanged(nameof(Master));
this.OnPropertyChanged(nameof(Master.Name));
```

KUVA 43. Hallitsevan yksikön vaihtaminen toiseen tai poistaminen.

Kun hallitsevaa yksikköä vaihdetaan, sekä uuden että vanhan hallitsevan yksikön laajennettujen CAN-väylien lukumäärä muuttuu. Mikäli vanhan keskusyksikön ensimmäisten laajennettujen CAN-väylien (laajennusväylät 7–10) hallitseva yksikkö muuttuu, mutta keskusyksiköllä on vielä muiden laajennusyksiköiden CAN-väyliä käytössä, täytyy jäljelle jäävien CAN-väylien laajennusnumerot muuttaa. Uudelle keskusyksikölle uudet laajennetut CAN-väylät sijoittuu jonon perälle. Tämä toiminnallisuus toteutettiin funktiolla (kuva 44), joka saa parametrinaan keskusyksikön ja laskee ExtensionCANs-kokoelman avulla laajennetuille CAN-väylille oikeat numerot.



```
/// <summary>
/// Handles extension numbers for extended CANs of Core device.
/// </summary>
/// <param name="device"></param>
4 references | Lindeman Johannes, 44 days ago | 1 author, 2 changes | 4 work items
public void HandleExtensionNumbers(IDevice device)
{
    if (device != null)
    {
        foreach (var can in device.Gateway.ExtensionCANs)
        {
            can.ExtendedNumber = (uint)(device.Count + device.Gateway.ExtensionCANs.IndexOf(can) + 1);
        }
    }
}
```

KUVA 44. Laajennettujen CAN-väylien numeroiden käsittely.

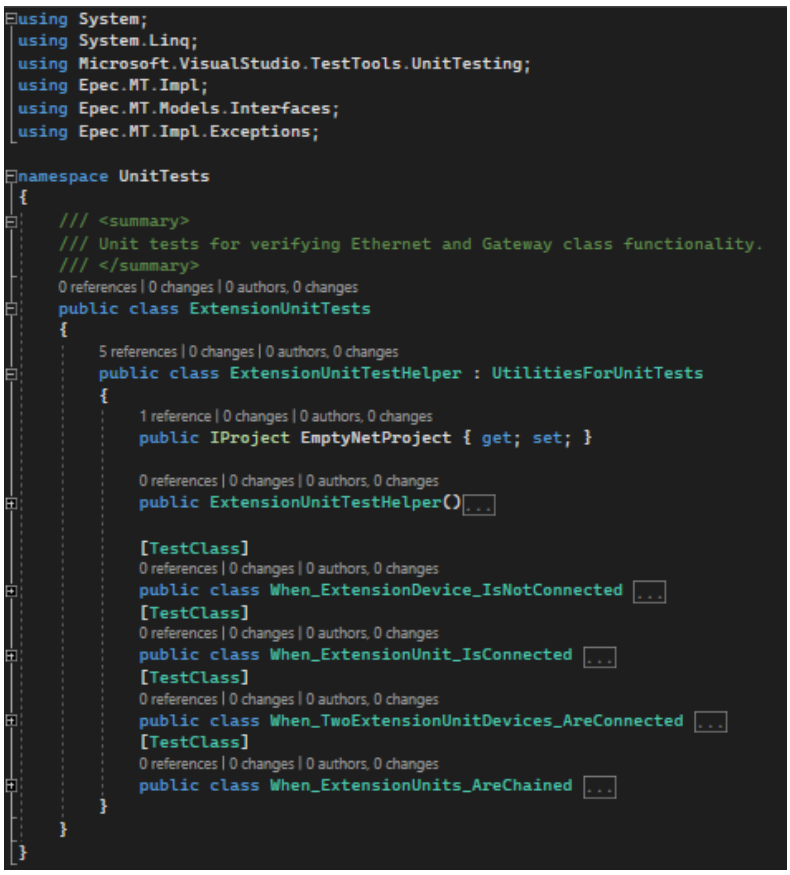
# 5 LAADUNVARMISTUS JA DOKUMENTAATIO

Laadunvarmistus laajennusyksikölle ja muille Multitool Creatorin ominaisuuksille perustuu yksikkötesteihin ja manuaalisiin toiminnallisiin testeihin. Multitool Creatorin yksikkötestit on automatisoitu ja ne ajetaan palvelimella jokaisen buildin yhteydessä. Tässä luvussa käydään läpi oleellisimmat yksikkötestit ja käsitellään toiminnallista testaamista yleisellä tasolla.

Käyttäjälle relevantti dokumentaatio yleisesti koko Multitool Creatorille löytyy sekä Multitool Manualista että Epecin Extranetistä. Yrityksen sisäisen kehitykseen liittyvään dokumentaatioon käytetään Microsoft SharePointia sekä Azure DevOpsia.

## 5.1 Yksikkötestit

Laajennusyksikön yksikkötestit toteutettiin MSTest-testiympäristöä hyödyntäen. Kaikki Multitool Creatorin yksikkötestit on toteutettu MSTest-testiympäristöllä, eikä tässä työssä ollut syytä poiketa käytännöstä. Testien toteuttaminen MSTestissä perustuu testiluokkiin ja testimetodeihin. Multitool Creatorin yksikkötestien nimeämiskäytäntöjen mukaisesti testiluokat (kuva 45) nimetään testiasetelmien mukaan ja testimetodit odotettujen tulosten mukaan.



```
using System;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Epec.MT.Impl;
using Epec.MT.Models.Interfaces;
using Epec.MT.Impl.Exceptions;

namespace UnitTests
{
    /// <summary>
    /// Unit tests for verifying Ethernet and Gateway class functionality.
    /// </summary>
    0 references | 0 changes | 0 authors, 0 changes
    public class ExtensionUnitTests
    {
        5 references | 0 changes | 0 authors, 0 changes
        public class ExtensionUnitTestHelper : UtilitiesForUnitTests
        {
            1 reference | 0 changes | 0 authors, 0 changes
            public IProject EmptyNetProject { get; set; }

            0 references | 0 changes | 0 authors, 0 changes
            public ExtensionUnitTestHelper()...

            [TestClass]
            0 references | 0 changes | 0 authors, 0 changes
            public class When_ExtensionDevice_IsNotConnected ...
            [TestClass]
            0 references | 0 changes | 0 authors, 0 changes
            public class When_ExtensionUnit_IsConnected ...
            [TestClass]
            0 references | 0 changes | 0 authors, 0 changes
            public class When_TwoExtensionUnitDevices_AreConnected ...
            [TestClass]
            0 references | 0 changes | 0 authors, 0 changes
            public class When_ExtensionUnits_AreChained ...
        }
    }
}
```

KUVA 45. Testiluokat laajennusyksikön yksikkötesteille.

Testimetodille voidaan syöttää dataa parametreina käyttämällä tavallisen testimetodin sijasta DataTestMethod-tyyppistä testimetodia, jolloin dataa voidaan syöttää yhden tai useamman DataRow:n muodossa (kuva 46).

```
[TestClass]
0 references | 0 changes | 0 authors, 0 changes
public class When_ExtensionDevice_IsNotConnected : ExtensionUnitTestHelper
{
    [DataTestMethod]
    [DataRow("ExtensionUnit", "Default", "0.0")]
    0 references | 0 changes | 0 authors, 0 changes
    public void ExtensionUnit_ShouldHave_NoConnections(string productCode, string functionalVersion, string csVersion)
    {
        // Create ExtensionUnit device.
        var device = CreateDevice(productCode, functionalVersion, csVersion);
        // Network chain should have no networks
        Assert.IsTrue(device.Gateway.NetworkChain.Count.Equals(0));
        // Ethernets should not be connected to any networks.
        foreach (var ethernet in device.Gateway.OfType<IEthernet>())
        {
            Assert.IsNull(ethernet.Network);
        }
    }
}
```

KUVA 46. MSTest -testiympäristöllä toteutettu testiluokka ja testimetodi.

Päähuomio laajennusyksikön yksikkötesteissä kiinnitettiin verkkojen ja verkostojen toimintaan. Kytkentöjen perustoimintoja testattiin mm. luomalla laajennusyk-

sikkö ja kytkemällä se verkkoon, jonka jälkeen testattiin, että NetworkChain-kokoelmassa on sisältöä ja jokin laitteen etherneteistä on kytketty verkkoon (kuva 47).

```
[TestClass]
0 references | 0 changes | 0 authors, 0 changes
public class When_ExtensionUnit_IsConnected : ExtensionUnitTestHelper
{
    [DataTestMethod]
    [DataRow("ExtensionUnit", "Default", "0.0")]
    ● | 0 references | 0 changes | 0 authors, 0 changes
    public void ExtensionUnit_ShouldHave_Connections(string productCode, string functionalVersion, string csVersion)
    {
        // Create ExtensionUnit device.
        var device = CreateDevice(productCode, functionalVersion, csVersion);
        var ethernet = device.Gateway.FirstOrDefault();

        INetwork network = new Network();
        ethernet.AddToNetwork(network);

        // Network chain should have a network
        Assert.IsFalse(device.Gateway.NetworkChain.Count.Equals(0));
        // At least one ethernet should be connected to a network.
        Assert.IsTrue(device.Gateway.Any(a => a.Network != null));
    }
```

KUVA 47. Yksikkötesti ethernetin kytkemiselle verkkoon.

Ethernet-kytkentöjen validointeja testattiin muun muassa luomalla kaksi laitetta ja yrittämällä yhdistää yhden laitteen CAN-väylä toisen laitteen ethernetiin sekä päinvastoin (kuva 48). Testit määritetään onnistuneiksi, mikäli vääränlainen yhdistäminen tuottaa oikeanlaisen poikkeuksen ja jälkimmäinen kytkentä epäonnistuu, sillä ensimmäinen verkkoon kytketty väylä määrittää verkon tyypin.

```
    [DataTestMethod]
    [DataRow("ExtensionUnit", "Default", "0.0")]
    ● | 0 references | 0 changes | 0 authors, 0 changes
    public void Networks_ShouldNotAllow_CanToEthernetConnections(string productCode, string functionalVersion, string csVersion)
    {
        // Create ExtensionUnit device.
        var device1 = CreateDevice(productCode, functionalVersion, csVersion);
        var device2 = CreateDevice(productCode, functionalVersion, csVersion);
        var ethernet = device1.Gateway.FirstOrDefault();
        var can = device2.FirstOrDefault();

        INetwork network = new Network();
        ethernet.AddToNetwork(network);

        // Connecting can to network that already has Ethernet should throw exception.
        Assert.ThrowsException<CanConnectedException>(() => can.AddToNetwork(network));
        // Network should contain only one item.
        Assert.IsTrue(network.Count.Equals(1));
    }
    [DataTestMethod]
    [DataRow("ExtensionUnit", "Default", "0.0")]
    ● | 0 references | 0 changes | 0 authors, 0 changes
    public void Networks_ShouldNotAllow_EthernetToCanConnections(string productCode, string functionalVersion, string csVersion)
    {
        // Create ExtensionUnit device.
        var device1 = CreateDevice(productCode, functionalVersion, csVersion);
        var device2 = CreateDevice(productCode, functionalVersion, csVersion);
        var ethernet = device1.Gateway.FirstOrDefault();
        var can = device2.FirstOrDefault();

        INetwork network = new Network();
        can.AddToNetwork(network);

        // Connecting can to network that already has Ethernet should throw exception.
        Assert.ThrowsException<CanConnectedException>(() => ethernet.AddToNetwork(network));
        Assert.IsTrue(network.Count.Equals(1));

    }
}
```

KUVA 48. Ethernet- ja CAN-kytkentöjen validointitestit.

Kytkentöjen validointitesteissä testattiin myös muun muassa, että ethernetin pystyy yhdistämään vain yhteen verkkoon kerrallaan (kuva 49). Testi määritellään onnistuneeksi, kun verkkoon jo kytketyn ethernetin yhdistäminen toiseen verkkoon epäonnistuu ja tuottaa poikkeuksen.

```
[DataTestMethod]
[DataRow("ExtensionUnit", "Default", "0.0")]
0 references | 0 changes | 0 authors, 0 changes
public void Ethernet_ShouldBe_AbleToConnectToOnlyOneNetwork(string productCode, string functionalVersion, string csVersion)
{
    // Create ExtensionUnit device.
    var device1 = CreateDevice(productCode, functionalVersion, csVersion);
    var ethernet1 = device1.Gateway.FirstOrDefault();
    var ethernet2 = device1.Gateway.LastOrDefault();
    INetwork network1 = new Network();
    INetwork network2 = new Network();
    ethernet1.AddToNetwork(network1);

    // Connecting one ethernet to many networks should throw exception.
    Assert.ThrowsException<CanConnectedException>(() =>  ethernet1.AddToNetwork(network2));
}
```

KUVA 49. Ethernetin kytkentä toiseen verkkoon.

Toisin kuin CAN-verkko, ethernet-verkko saa pitää sisällään vain kaksi yhdistettyä ethernetiä. Tämän testaamiseksi luotiin neljä laitetta ja yritettiin yhdistää kaikki laitteet samaan verkkoon (kuva 50). Testi määritellään onnistuneeksi, kun vain kaksi ensimmäistä kytkentää onnistuu.

```
[DataTestMethod]
[DataRow("ExtensionUnit", "Default", "0.0")]
0 references | 0 changes | 0 authors, 0 changes
public void Network_ShouldAllow_OnlyTwoEthernetConnections(string productCode, string functionalVersion, string csVersion)
{
    var device1 = CreateDevice(productCode, functionalVersion, csVersion);
    var device2 = CreateDevice(productCode, functionalVersion, csVersion);
    var device3 = CreateDevice(productCode, functionalVersion, csVersion);
    var device4 = CreateDevice(productCode, functionalVersion, csVersion);

    var deviceethernet1 = device1.Gateway.FirstOrDefault();
    var device2ethernet1 = device2.Gateway.FirstOrDefault();
    var device2ethernet2 = device2.Gateway.LastOrDefault();
    var device3ethernet1 = device3.Gateway.FirstOrDefault();
    var device3ethernet2 = device3.Gateway.LastOrDefault();
    var device4ethernet1 = device4.Gateway.FirstOrDefault();
    var device4ethernet2 = device4.Gateway.FirstOrDefault();

    INetwork network = new Network();

    try
    {
        // Two first connection attempts should be succesful.
        deviceethernet1.AddToNetwork(network);
        device2ethernet1.AddToNetwork(network);

        Assert.IsTrue(network.Count.Equals(2));

        // Other efforts to connect should throw exceptions and network count should remain untouched.
        Assert.ThrowsException<CanConnectedException>(() => device3ethernet1.AddToNetwork(network));
        Assert.ThrowsException<CanConnectedException>(() => device4ethernet1.AddToNetwork(network));
        Assert.IsTrue(network.Count.Equals(2));
    }
    catch (Exception ex)
    {
        Assert.Fail("Failure in 'Network_ShouldAllow_OnlyOneEthernetFromSameDevice' test! " + ex);
    }
}
```

KUVA 50. Useamman kuin kahden ethernetin kytkentä samaan verkkoon.

Verkostojen päivittymistä testattiin luomalla useita laitteita ja kytkemällä ne verkostoon, jonka jälkeen kytkennät laitteiden 1 ja 2 välillä poistettiin (kuva 51).

```
[DataTestMethod]
[DataRow("ExtensionUnit", "Default", "0.0")]
0 references | 0 changes | 0 authors, 0 changes
public void NetworkChains_ShouldUpdate_OnRemoval(string productCode, string functionalVersion, string csVersion)
{
    var device1 = CreateDevice(productCode, functionalVersion, csVersion);
    var device2 = CreateDevice(productCode, functionalVersion, csVersion);
    var device3 = CreateDevice(productCode, functionalVersion, csVersion);
    var device4 = CreateDevice(productCode, functionalVersion, csVersion);

    var device1ethernet1 = device1.Gateway.FirstOrDefault();
    var device2ethernet1 = device2.Gateway.FirstOrDefault();
    var device2ethernet2 = device2.Gateway.LastOrDefault();
    var device3ethernet1 = device3.Gateway.FirstOrDefault();
    var device3ethernet2 = device3.Gateway.LastOrDefault();
    var device4ethernet1 = device4.Gateway.FirstOrDefault();
    var device4ethernet2 = device4.Gateway.FirstOrDefault();

    INetwork network1 = new Network();
    INetwork network2 = new Network();
    INetwork network3 = new Network();

    try
    {
        device1ethernet1.AddToNetwork(network1);
        device2ethernet1.AddToNetwork(network1);
        device2ethernet2.AddToNetwork(network2);
        device3ethernet1.AddToNetwork(network2);
        device3ethernet2.AddToNetwork(network3);
        device4ethernet1.AddToNetwork(network3);
    }
    catch (Exception ex)
    {
        Assert.Fail("Failure in 'NetworkChains_ShouldUpdate_OnRemoval' test! " + ex);
    }

    // The number of networks should be same for all devices.
    Assert.IsTrue(device1.Gateway.NetworkChain.Count.Equals(3));
    Assert.IsTrue(device2.Gateway.NetworkChain.Count.Equals(3));
    try
    {
        device1ethernet1.RemoveFromNetwork();
        device2ethernet1.RemoveFromNetwork();
    }
```

KUVA 51. Verkoston luominen ja sen päivittyminen.

Testi määritellään onnistuneeksi, kun laitteen 1 verkosto on tyhjä, sillä se ei enää ole kytketty mihinkään, ja muiden laitteiden verkostot ovat identtiset (kuva 52).

```
    // device 1 should have no connections in network chain.
    Assert.IsTrue(device1.Gateway.NetworkChain.Count.Equals(0));
    // device 2 should have 2 connections left in the chain.
    Assert.IsTrue(device2.Gateway.NetworkChain.Count.Equals(2));

    // All chains should have same networks except device 1 that is disconnected from chain.
    Assert.IsFalse(device1.Gateway.NetworkChain.SequenceEqual(device2.Gateway.NetworkChain));
    Assert.IsTrue(device2.Gateway.NetworkChain.SequenceEqual(device3.Gateway.NetworkChain)
        && device3.Gateway.NetworkChain.SequenceEqual(device4.Gateway.NetworkChain));
}
```

KUVA 52. Verkoston päivittyminen kaikilla laitteilla muutoksen tapahtuessa.

MSTest-testiympäristö on monipuolinen ja kattava testausympäristö, mutta siitä löytyy myös puutteensa. Kenties suurimpana puutteena voidaan pitää kattavuus-

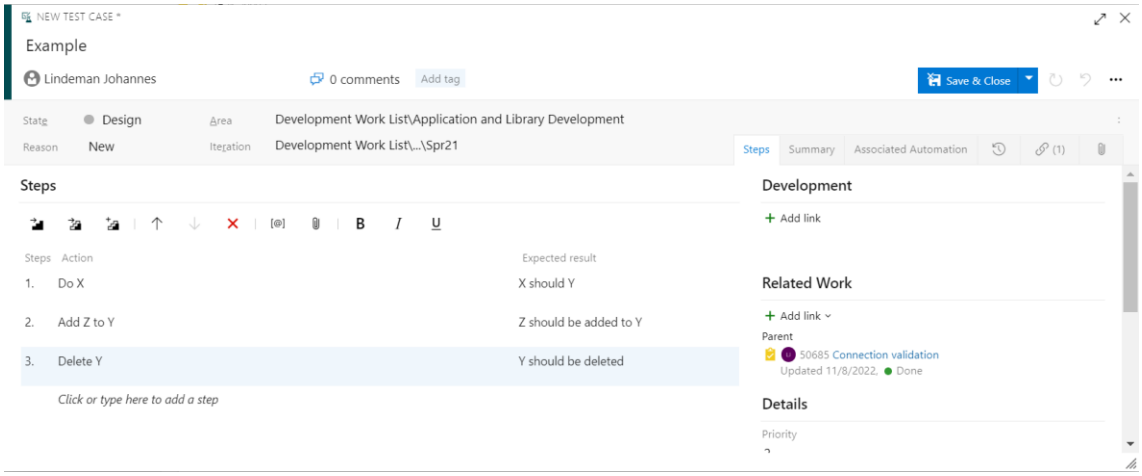
raportoinnin puutetta. Mikäli MSTest:llä luotujen testien kattavuutta haluaa tarkastella, on siihen käytettävä kolmannen osapuolen työkaluja, kuten Coverlet ja ReportGenerator (Microsoft 2023).

## 5.2 Toiminnalliset testit

Multitool Creatorin uusien ominaisuuksien manuaalisista toiminnallisista testeistä vastaa pääasiassa ohjelmistotestaukseen erikoistunut testausspesialisti. Multitool Creatorin kehityksestä vastaavan ohjelmistokehitystiimin testauskäytäntöjen mukaisesti jonkin ominaisuuden kehittäjän ei tulisi itse suorittaa oman tuotoksensa toiminnallisia testejä. Koska laajennusyksikön on määrä kuulua Multitool Creatorin julkaisuun, joka julkaistaan useita kuukausia itse tämän opinnäytetyön jälkeen, aikataulullisten priorisointien vuoksi tämän projektin toiminnallinen testaaminen tapahtuu vasta tulevaisuudessa. Tämän takia tässä raportissa ei kuvailla projektiin liittyviä todellisia toiminnallisia testejä, vaan käsitellään toiminnallista testaamista yleisellä tasolla ja esimerkein.

Toiminnalliset testit kuuluvat niin sanottujen ”musta laatikko” -testien perheeseen. ”Musta laatikko” -testeiksi kutsutaan testejä, joissa testaaja ei tiedä itse ohjelmalogiikasta mitään, vaan testit perustuvat puhtaasti syötteisiin tai toimintoihin ja niiden odotettuihin tuloksiin. Toiminnallisilla testeillä arvioidaan, toimiiko ohjelma vaatimusten määrittämällä tavalla. Testaamisen tavoitteena on varmistaa, että kaikki ohjelman toiminnot toimivat juuri kuten on tarkoitettu. (Codedec n.d.)

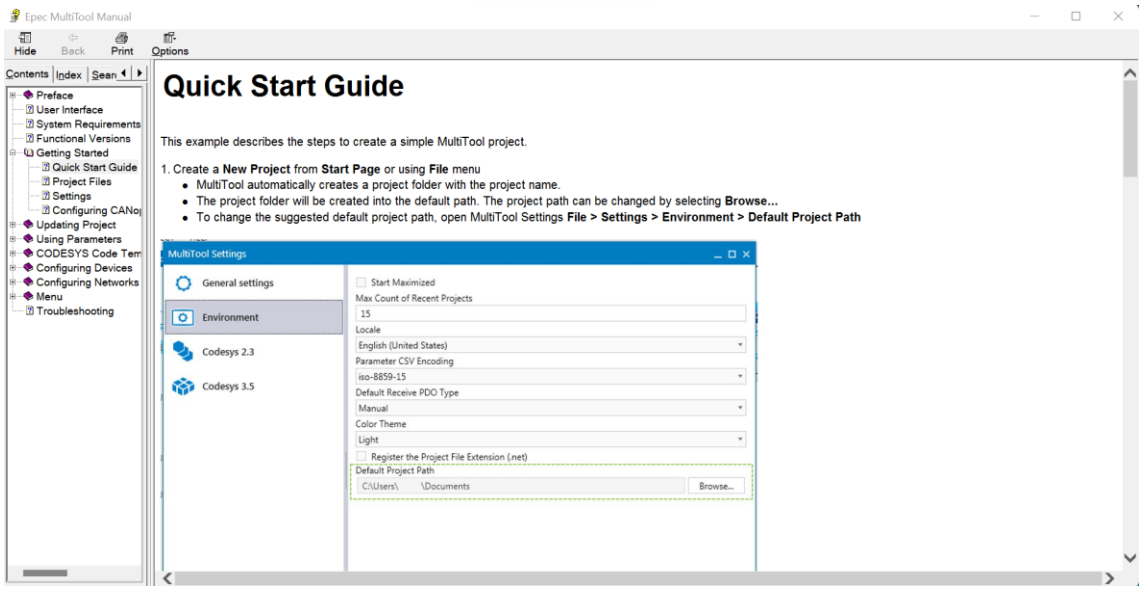
Yleisesti Multitool Creatorin toiminnalliset testit dokumentoidaan hyödyntäen Azure DevOpsia ja sen test-itemeita. Kuhunkin itemiin (kuva 53) kirjataan, millaista tulosta odotetaan jonkun toiminnan tuloksena. Itemiin voidaan kirjata myös muita oleellisia tietoja, kuten vaikka buildin numero.




KUVA 53. Testi-item Azure DevOpsissa.

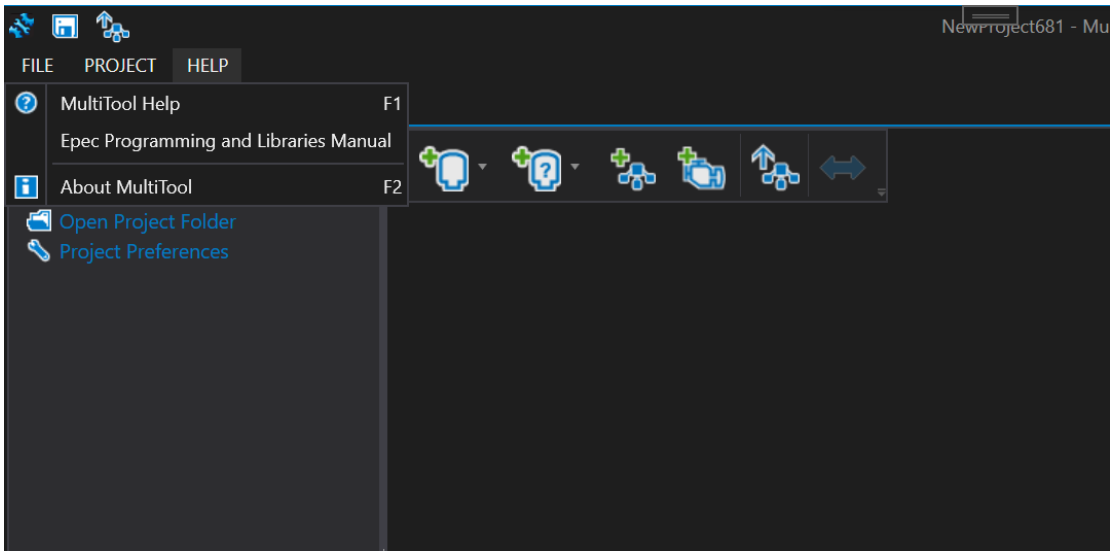
## 5.3 Dokumentaatio

Yleisesti Multitooliin liittyvää dokumentaatiota löytyy laajasti Epec Extranetistä, Multitool Manualista ja Epecin ohjelmointimanuaalista. Epec Extranet on asiakkaille suunnattu verkkosivu, joka sisältää ohjelmointimanuaaleja, käyttöoppaita ja muita Epecin tuotteisiin liittyviä dokumentteja. Multitool Manual (kuva 54) on kattava käyttöopas, joka sisältää selitykset kaikista Multitool Creatorin tärkeimmistä toiminnoista ja yksityiskohtaista opastusta sovelluksen oikeaoppiseen käyttöön.




KUVA 54. Multitool Manual avattuna Multitool Creatorin käyttöliittymästä.

Multitool Manualiin ja Epecin ohjelmointimanuaaliin käyttäjä pääsee käsiksi suoraan Multitool Creatorin käyttöliittymän yläpalkista (kuva 55). Multitool Manualiin tulee käyttöohjeistusta ja dokumentaatiota myös laajennusyksikköön ja keskusyksikköön liittyen myöhemmin päätettävänä ajankohtana.




KUVA 55. Multitool Manualin sijainti käyttöliittymässä.

Ohjelmistokehitykseen liittyvään dokumentaatioon hyödynnettiin SharePointia ja Azure DevOpsia. SharePoint mahdollistaa muun muassa vaatimusdokumenttien jakamisen tiimin kesken ja Azure DevOpsin avulla kaikki tehty kehitystyö dokumentoidaan featureihin, backlog itemeihin, bugeihin ja taskeihin.

# 6 POHDINTA

Tämän työn keskeisimpänä tuloksena kehitettiin onnistuneesti Multitool Creator-konfiguraatiotyökaluun laitetuki uudenlaiselle I/O & CAN -laajennusyksiköille sekä pohja myös mahdollisten tulevien laitteiden ethernet-kytkennöille Multitool Creatorissa. Työ sujui alusta loppuun kohtuullisen hyvin, eikä ylitsepääsemättömiä ongelmia ilmennyt. Työn vaikeustasoa voidaan katsoa nostaneen sen, että Multitool Creator ei ole kuluttajakäyttöön, vaan hallintajärjestelmien ammattilaisten käyttöön tarkoitettu sovellus, joten pelkästään Multitool Creatorin käyttäminen vaatii jonkin verran perehtymistä aihealueeseen.

Työ itsessään oli erittäin opettavaista sekä mielenkiintoista ja tarjosi paljon uutta kokemusta ja syventävää oppia erityisesti C#-ohjelmoinnista, mutta myös ohjelmistokehitystyöstä ja sen menetelmistä yleisellä tasolla. Työ oli myös omiaan havainnollistamaan Azure DevOps -työkalujen hyödyllisyyden projektinhallinnan apuvälineenä käytännön tasolla.

Projektia voidaan jatkokehittää ja tullaan jatkokehittämään läheisesti laajennusyksikköön liittyvän keskusyksikön kehityksen yhteydessä. Multitool Creatorin versio, jossa tämän projektin tuotokset ovat mukana, tullaan julkaisemaan Epecin toimesta arviolta vuoden 2023 aikana.

**LÄHTEET**

CAN in Automation. n.d. PDO protocol. Verkkosivu. Viitattu 3.1.2023. https://can-cia.org/can-knowledge/canopen/pdo-protocol/

Codedec. n.d. Functional Testing in Software Testing. Verkkosivu. Viitattu 9.2.2023. https://codedec.com/tutorials/functional-testing-in-software-testing/

Epec. n.d. Company. Verkkosivu. Viitattu 13.12.2022. https://epec.fi/company/

Epec. n.d. Functional Safety. Verkkosivu. Viitattu 25.1.2023. https://epec.fi/products/functional-safety/

Epec. n.d. Epec SL84 Safety Control Unit. Verkkosivu. Viitattu 12.1.2022. https://epec.fi/products/epec-sl84-safety-control-unit/

Epec. 2015. New version of CANmoon configuration and diagnostics tool available. Verkkosivu. Viitattu 25.1.2023. https://epec.fi/new-version-of-canmoon-configuration-and-diagnostics-tool-available/

Epec. 2021. Optimize Time-To-Market with Epec MultiTool. Verkkosivu. Viitattu 8.12.2022. https://epec.fi/optimize-time-to-market-with-epec-multitool/

Lekman, Lare. 2009. Mikä ihmeen Kanban. Lekman. Verkkosivu. Viitattu 1.2.2023. https://www.lekman.fi/post/mika-ihmeen-kanban

Microsoft. n.d. Thumb Class. Verkkosivu. Viitattu 23.12.2022. https://learn.microsoft.com/en-us/uwp/api/windows.ui.xaml.controls.primitives.thumb?view=winrt-22621

Microsoft. 2022. Develop and share code in TFVC with Visual Studio. Verkkosivu. Viitattu 31.1.2023. https://learn.microsoft.com/en-us/azure/devops/repos/tfvc/share-your-code-in-tfvc-vs?view=azure-devops

Microsoft. 2022. What is Azure Boards. Verkkosivu. Viitattu 1.2.2023. https://learn.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops

Microsoft. 2022. Interface (C# Reference). Verkkosivu. Viitattu 4.1.2023. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

Microsoft. n.d. ICollection<T> Interface. Verkkosivu. Viitattu 4.1.2023. https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.icollection-1?view=net-7.0

Microsoft. n.d. MultiBinding Class. Verkkosivu. Viitattu 9.2.2023. https://learn.microsoft.com/en-us/dotnet/api/system.windows.data.multibinding?view=windowsdesktop-7.0

Microsoft. n.d. IMultiValueConverter Interface. Verkkosivu. Viitattu 13.3.2023. https://learn.microsoft.com/en-us/dotnet/api/system.windows.data.imultivalueconverter?view=windowsdesktop-7.0

Microsoft. 2023. Use code coverage for unit testing. Verkkosivu. Viitattu 13.1.2023. https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-code-coverage?tabs=windows

Motion Ai. 2019. What is CODESYS and Why Should You Use it? Verkkosivu. Viitattu 12.1.2022. https://ammc.com/codesys-why-should-you-use-it/

National Instruments. 2022. The Basics of CANopen. Verkkosivu. Viitattu 20.1.2023. https://www.ni.com/fi-fi/innovations/white-papers/13/the-basics-of-canopen.html