Tampereen ammattikorkeakoulu

TwinCAT 3 -ohjelmakirjasto pienoisrautatien ohjaukseen

Antti Seitsenlinna

OPINNÄYTETYÖ
Toukokuu 2024

Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

SEITSENLINNA, ANTTI:
TwinCAT 3 -ohjelmakirjasto pienoisrautatien ohjaukseen

Opinnäytetyö 77 sivua, joista liitteitä 22 sivua
Toukokuu 2024

Tampereen ammattikorkeakoulussa kehitetään pienoisrautatiehen pohjautuvaa sähkö- ja automaatiotekniikan oppimisympäristöä. Sen tarkoituksena on yhdistää monenlaisia erityisesti käytännön oppimismahdollisuuksia toiminnallisuuksiksi eli asemapaikoiksi radan varrelle. Niissä voidaan teettää harjoitustöitä, minkä lisäksi oppimisympäristöä voidaan kehittää opiskelijavetoisesti esimerkiksi projekteilla. Kehitystyössä on mahdollista hyödyntää osaamista myös muilta aloilta, ja vain mielikuvitus onkin rajana oppimisympäristön edistämisessä.

Tässä opinnäytetyössä luotiin pienoisrautatien Märklin-laitteiden ohjaamiseksi CAN-rajapinta ja ohjelmakirjasto TwinCAT 3 -kehitysympäristöön. Niille laadittiin myös kattava dokumentaatio tulevan käytön helpottamiseksi. Opinnäytetyöllä muodostettiin perusta pienoisrautatien ominaisuuksien kehittämiselle jatkossa. Työssä esitellään pienoisrautatieympäristöä sekä perehdytään CAN-väylään ja sen käyttöön Märklin Digitalissa. Lisäksi työssä käydään läpi rajapintaohjelman toteutuksessa kohdattuja haasteita.

Valmis ohjelmakirjasto sisältää kaikki keskeiset ohjelmalohkot Märklin-pohjaisten veturien ja lisälaitteiden ohjaamiseksi. Laadittu dokumentaatio on perusteellinen ja ohjelmakoodin kommentointi kattava. Työ edistää oppimisympäristön kehitystä merkittävästi ja muodostaa perustan ohjausjärjestelmän ja muiden sovellusten kehittämiselle tulevaisuudessa. Kehityskohteena saattaa olla jatkossa joidenkin konfigurointi- ja erityistoimintojen lisääminen, mikäli ne koetaankin tarpeellisiksi. Ohjelmakirjaston dokumentaatio on kokonaisuudessaan työn liitteissä.

Asiasanat: pienoisrautatie, ohjelmakirjasto, CAN, TwinCAT 3, automaatio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Electrical Engineering
Automation Engineering

SEITSENLINNA, ANTTI:
TwinCAT 3 Program Library for Model Railway Control

Bachelor's thesis 77 pages, appendices 22 pages
May 2024

A learning environment based on model railway is being developed by Tampere University of Applied Sciences for electrical and automation students. It combines a wide variety of primarily practical learning opportunities into stations along the track. They can be used for various types of hands-on exercises. The learning environment can also be developed through student-led projects, and students from other fields can also participate in the development process. Ultimately, only imagination limits all the possibilities.

The purpose of this thesis was to implement a CAN interface and program library for controlling Märklin model railway devices in TwinCAT 3 environment. To make the library easier to use, a comprehensive documentation was also created. The thesis formed the basis for the future development of the features of the model railway. The thesis presents the model railway environment with its devices and provides information about the CAN bus and its use in Märklin Digital. In addition, the challenges faced in implementing the interface program are highlighted.

The resulting program library includes all the most essential function blocks for controlling Märklin-based locomotives and accessories. A thoroughly commented program library, along with comprehensive documentation, facilitates the future use of the library. Thus, the thesis fulfills its objectives and contributes notably to the development of the learning environment. In the future, it may be necessary to add some configuration and special functionalities if those are needed. The complete documentation is included in the appendices.

Key words: model railway, program library, CAN, TwinCAT 3, automation

SISÄLLYS

LYHENTEET JA TERMIT

CAN	Controller Area Network; Väyläprotokolla
CNC	Computer Numerical Control; Tietokoneohjattu numeerinen ohjaus
DEL	Delimiter; Erotin, erilliset kentät toisistaan erottava osa
DLC	Data Length Code; Pituuskoodi, datatavujen määrä
EOF	End-of-frame; Kehyksen viimeinen, lopettava osa
IDE	Identifier Extension Bit; Kehysmuodon määrittävä bitti
IEC	International Electrotechnical Commission; Sähköalan kansainvälinen standardisoimisjärjestö
ISO	International Organization for Standardization; Kansainvälinen standardisoimisjärjestö
NC	Numerical Control; Numeerinen ohjaus
NRZ	Non-return-to-zero; Binäärikoodaus, jossa molemmat bitin tilat esitetään merkitsevinä signaalitasoina
PLC	Programmable Logic Controller; Ohjelmoitava logiikka
RTR	Remote Transmission Request; Erottaa datakehyksen ja datapyyntökehyksen toisistaan
SAE	Society of Automotive Engineers; Yhdysvaltalainen autoalan standardisointijärjestö
SOF	Start-of-frame; Kehyksen ensimmäinen, aloittava osa
SRR	Substitute Remote Request; Tarvittaessa RTR-osan paikalla oleva korvaava osa
ST	Structured Text; Strukturoitu teksti, tekstipohjainen ohjelmointikieli ohjelmoitaville logiikoille
TAMK	Tampereen ammattikorkeakoulu
TwinCAT	The Windows Control and Automation Technology; Beckhoffin kehittämä automaatioalan kehitysympäristö

1 JOHDANTO

Vain mielikuvitus on rajana siinä, mitä pienoisrautateillä voidaan toteuttaa. Yksi esimerkki tästä on Tampereen ammattikorkeakoulun kehitteillä oleva sähkö- ja automaatiotekniikan oppimisympäristö, jossa yhdistetään pienoisrautatien avulla erilaisia yksittäisiä toiminnallisuuksia asemapaikoiksi radan varrelle. Ympäristö luo teoriaopetuksen rinnalle mahdollisuuden teettää monenlaisia eri käytännön harjoitus- ja projektitöitä sekä radan valmiilla asemapaikoilla että kehitystyössä, jossa on mahdollisuuksia eri alojen opiskelijoille tietotekniikasta arkkitehteihin.

Yhteen alkuvaiheen keskeisistä kehitystarpeista lähdettiin vastaamaan tässä opinnäytetyössä, kun pienoisrautatielle luotiin CAN-rajapinta ja ohjelmakirjasto Beckhoffin TwinCAT 3 -kehitysympäristöön. Ohjelmakirjaston tavoitteena oli muodostaa perusta ohjausjärjestelmän ja muiden ominaisuuksien kehittämiselle jatkossa, mikä veisi oppimisympäristöä merkittävästi eteenpäin.

Työssä esitellään pienoisrautatieympäristöä sekä perehdytään CAN-väylään ja sen käyttöön Märklin Digitalissa. Työssä käydään läpi myös rajapintaohjelman toteutuksessa kohdattuja haasteita ja selvitystyötä. Lisäksi tutustutaan testeihin, joita rajapintaohjelmalle ja ohjelmakirjastolle tehtiin niiden oikeanlaisen toiminnan varmistamiseksi. Työssä laadittiin myös kattava dokumentaatio tulevan käytön helpottamiseksi ja pohdittiin tavoitteiden toteutumista sekä jatkokehityskohteita.

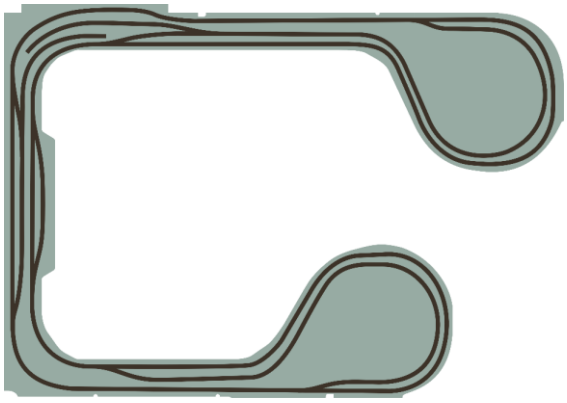
2 TOTEUTUSYMPÄRISTÖ JA LAITTEISTO

2.1 Pienoisrautatieympäristö

Työ toteutettiin Tampereen ammattikorkeakoulun pienoisrautatieympäristöön, joka on kehitteillä oleva sähkö- ja automaatioalan oppimisympäristö erityisesti käytännön opetusta varten. Siellä on tarkoitus järjestää kursseja syksystä 2024 alkaen. Radan varrelle kehitetään erilaisia toiminnallisuuksia eli asemapaikkoja, joissa voidaan teettää harjoitustöitä. Asemapaikkoja ja muuta oppimisympäristöä on määrä kehittää opiskelijavetoisesti erilaisilla projektitöillä. Kuvassa 1 esitetään panoraamanäkymä oppimisympäristöstä.

KUVA 1. Kehitteillä oleva pienoisrautatieympäristö. (Kuva: Antti Seitsenlinna)

Pienoisrautatie on rakennettu noin 60 neliömetrin kokoiseen tilaan vaneritasolle, joka kiertää tilaa kaapelihyllyn päällä kolmella seinällä. Kaapelihyllyn ansiosta eri laitteiden, anturien ja muiden rataverkon toimintojen kaapelointi on helpompaa. Raidetta pienoisrautatiessä on tällä hetkellä yhteensä noin 80 metriä. Ympäristön rataverkossa on useampi rinnakkainen raide ja yhteensä kymmenen vaihdetta eli risteyskohtaa. Rataverkkoa ja raiteiden asettelua havainnollistetaan kuviossa 1.



KUVIO 1. Pienoisrautatieympäristön rataverkko. (Kuvio: Antti Seitsenlinna)

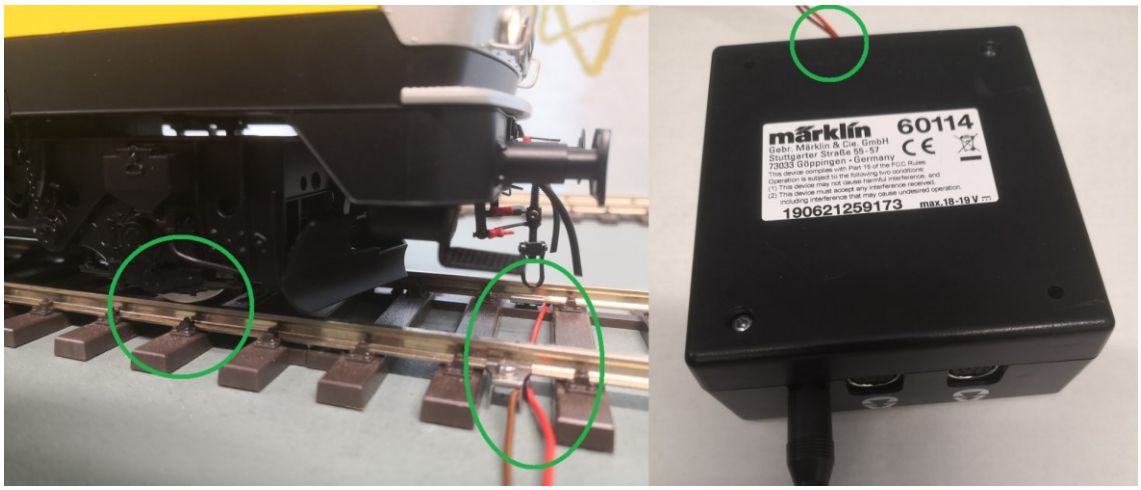
Oppimisympäristöön on tällä hetkellä kaavailtu noin kymmentä eri asemapaikkaa radan varrelle. Niiden on tarkoitus yhdistyä junan myötä kokonaisuudeksi siten, että esimerkiksi raaka-aineita tai tuotteita voisi kuljettaa asemapaikalta toiselle. Asemapaikkoja on tarkoitus tehdä monesta eri osa-alueesta, ja suunnitteilla ovat esimerkiksi kaivos, jäteasema, voimalaitos sekä prosessitehdas. Ympäristössä on myös mahdollisuus tarjota näkyvyyttä eri alojen yrityksille. Kaksi suunniteltua asemapaikkaa on esitetty kuvassa 2. Asemapaikkojen lisäksi on kaavailtu muun muassa Tampereen rautatieasemaa siluetteineen sekä kaukolämpöverkkoa.

KUVA 2. Suunniteltuja asemapaikkoja. (Kuva: Antti Seitsenlinna)

Junakaluston ja raiteet on valmistanut Märklin. Raiteet rakentuvat 1-kokoisista raidepaloista. Tällä hetkellä kalusto koostuu veturista ja neljästä vaunusta. Veturi on 71,9 senttimetriä pitkä ja painaa 9,2 kiloa. Se on mittakaavan 32:1 pienoismalli aidosta SerFer V320 -dieselveturista. (Product description n.d.) Neljä vaunua ovat kaikki 37,5 senttimetriä pitkiä lavettivaunuja. Niihin voidaan tehdä muutoksia tarpeiden mukaan, ja yhdestä vaunusta onkin jo hahmoteltu radan anturivaunua. Käytössä oleva junakalusto esitetään kuvassa 3.

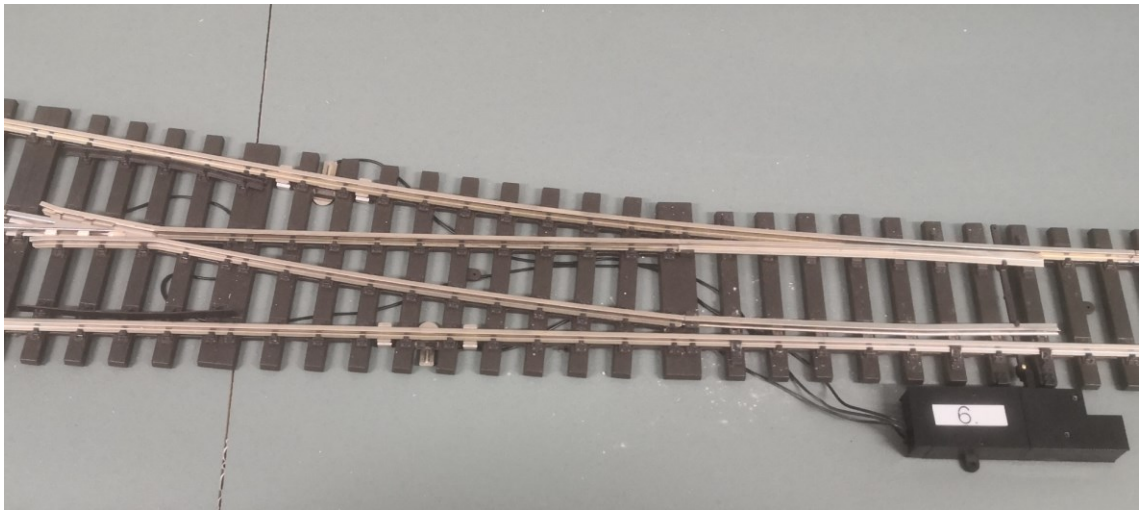
KUVA 3. Nykyinen junakalusto: veturi ja neljä vaunua. (Kuva: Antti Seitsenlinna)

Veturit ja muut radan laitteet saavat toimintavirtansa ja ohjauksensa kiskojen kautta ratayksiköstä. Noin 16–18 voltin vaihtojännite ja ohjaussignaali kulkevat päällekkäin samoissa johtimissa ja kiskoissa. Ratayksikkö osaa kommunikoida radan laitteiden kanssa montaa eri protokollaa käyttäen, kun taas ratayksikön kanssa keskustellaan CAN-väylällä. Kuvassa 4 on esitetty ratayksikkö ja kaksi kiskoihin menevää johdinta, joista toimintavirta ja ohjaussignaali välittyvät veturin metallisille pyörille.



KUVA 4. Ratayksikkö ja kiskojen johtimet. (Kuva: Antti Seitsenlinna)

Tässä työssä ainoat kiskojen kautta toimivat radan laitteet ovat veturia lukuun ottamatta vaihteet. Niille saatetaan jatkossa kuitenkin kehittää toinen ohjaustapa, sillä niiden käyttö ratayksikön kautta on epävarmempaa ja tilatieto jää saamatta takaisinkytkennän puuttuessa. Takaisinkytkentä puuttuu myös useimmilta muilta Märklinin lisälaitteilta. Yksi radan vaihteista on esitetty kuvassa 5.



KUVA 5. Yksi rataverkon vaihteista. (Kuva: Antti Seitsenlinna)

Työssä on tavoitteena luoda CAN-rajapinta, jonka avulla voidaan kommunikoida ratayksikön kanssa. Rajapintaohjelman päälle toteutetaan veturin ja lisälaitteiden ohjaukseen ohjelmakirjasto. Tavoitteena on tuoda ohjaus kuvassa 6 näkyvältä käsiohjaimelta TwinCAT 3 -kehitysympäristöön, jotta pienoisrautatielle voitaisiin tulevaisuudessa kehittää sen pohjalta ohjausjärjestelmä ja muita ominaisuuksia.

KUVA 6. Märklin Mobile Station 2 -käsiohjain. (Kuva: Antti Seitsenlinna)

Ohjelmakirjastoon on tarkoituksena tuoda mahdollisimman moni käsiohjaimen ominaisuus. Sillä voidaan ohjata esimerkiksi veturin nopeutta ja suuntaa, valoja sekä antaa äänimerkkejä. Lisäksi voidaan ohjata lisälaitteiden kuten vaihteiden tilaa. Tavoitteena ohjelmakirjastossa on mahdollistaa erityisesti veturin ohjaus, mutta myös lisälaitteiden ohjaus on tarkoitus toteuttaa ohjaustavan mahdollisesta muuttumisesta huolimatta. Käsiohjaimen konfigurointitoimintoja ei niiden vähäisen tarpeen takia tässä vaiheessa tuoda ohjelmakirjastoon. Tällaisia toimintoja ovat muun muassa veturin enimmäisnopeus tai äänenvoimakkuuden taso.

2.2 Märklin Digital

Märklin on vuonna 1859 perustettu saksalainen leluvalmistaja. Se valmisti alun perin muun muassa nukkekotitarvikkeita, nukenrattaita ja miniatyyrihevoskärryjä. Nykyään yritys tunnetaan pienoisrautatietarvikkeistaan, joita se on valmistanut 1800-luvun loppupuolelta alkaen. (Über Märklin n.d.)

Märklin Digital on Märklinin kehittämä digitaalijärjestelmä pienoisrautateiden ohjaukseen. Digitaalijärjestelmän etuna on, että eri vetureita ja lisälaitteita voidaan

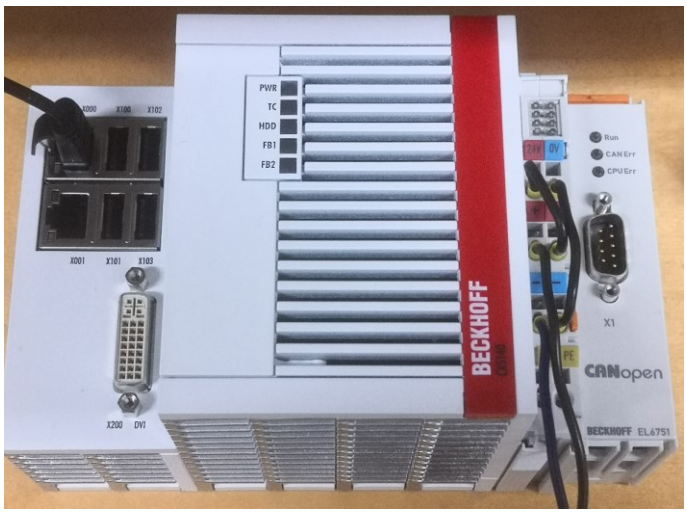
ohjata toisistaan riippumatta. Sen lisäksi se vähentää johdotuksen ja kytkentöjen tarvetta. Perinteinen vaihtoehto on veturien ohjaaminen jännitettä säätelemällä siten, että veturin nopeus on suhteessa jännitteen suuruuteen. Jokaiselle veturille jouduttaisiin tällöin varaamaan omat raiteensa ja pysähtyneille junille tekemään erillisiä raideosuuksia. Digitaalijärjestelmässä koko rataverkolla voi ajaa kaikilla yhteensopivilla vetureilla. (Märklin Digital antaa mahdollisuuksia n.d.)

Digitaalijärjestelmään voidaan kytkeä lukuisia erilaisia laitteita. Vetureilla voi olla useita toiminnallisuuksia, kuten valojen ohjaus tai äänimerkit. Tämän työn veturi tukee yhteensä 32:ta eri toimintoa (Product description n.d.). Mahdollisia radan laitteita ovat vaihteiden lisäksi muun muassa opastimet, nosturit ja kääntöpöydät. Lisäksi digitaalinen pienoisrautatie vaatii ohjauslaitteen, joista yksi esimerkki on tämän työn käsiohjain tai kehitettävä ohjelmakirjasto. Niiden ohella on erilaisia näppäinpöytiä, kosketusnäytöllisiä laitteita sekä releyksiköitä. Digitaalijärjestelmä tarvitsee myös keskusyksikön, joka keskustelee radan laitteiden kanssa. Tässä työssä sen virkaa hoitaa ratayksikkö, mutta eri vaihtoehtoja on Märklinillä useita. (Märklin Digital antaa mahdollisuuksia n.d.)

2.3 Beckhoff-laitteisto ja TwinCAT 3

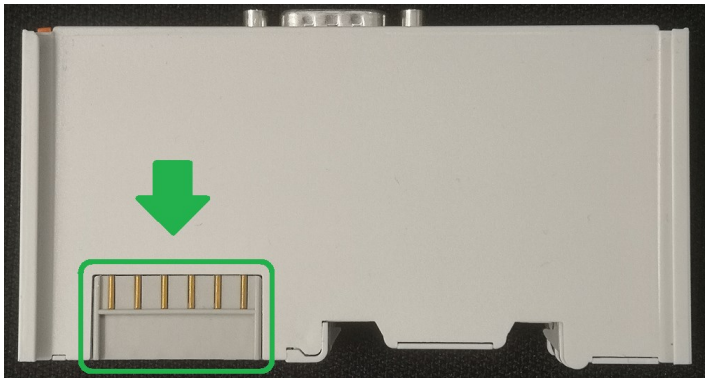
Työ toteutettiin TwinCAT 3 -kehitysympäristöön Beckhoff-laitteistoa käyttäen. Beckhoff on saksalainen automaatioalan yritys, jonka valikoimiin kuuluu muun muassa teollisuustietokoneita, I/O- ja kenttäväyläkomponentteja, ohjelmistoja sekä liikkeenohjaus- ja konenäkötuotteita (Company n.d.). Beckhoffin laitteisto valittiin, koska se oli jo ennestään valmiina TAMKin automaatiolaboratoriossa. Se soveltui myös hyvin työn rajapintaohjelman ja ohjelmakirjaston toteuttamiseen.

Laitteisto koostuu CX5140-teollisuustietokoneesta sekä siihen liitetystä EL6751-CANopen-moduulista. Tietokoneeseen on asennettu TwinCAT 3 -järjestelmä, jonka myötä se täyttää PLC-standardin IEC 61131-3 vaatimukset ja tukee neljää samanaikaista tehtävää. Sivussa on jousiliittimet EtherCAT-moduulien ja väylälaitteiden liittämiseksi. Tietokoneessa on kaksi Ethernet-porttia. (CX51x0 Manual 8.1.2024, 14–15) Toista porteista käytetään tässä työssä yhteyden muodostamisessa tietokoneeseen. Kuvassa 7 on esitetty tietokone ja siihen liitetty moduuli.



KUVA 7. Beckhoff CX5140 -teollisuustietokone, jonka oikeassa reunassa siihen liitetty Beckhoff EL6751 -moduuli. (Kuva: Antti Seitsenlinna)

Työssä käytettävä EL6751 -moduuli on tehty ensisijaisesti CANopen-laitteille, joka on CAN-väylään pohjautuva protokolla. Moduulia voidaan kuitenkin käyttää myös tavallisten CAN-viestien lähetykseen ja vastaanottoon. (EL6751 Manual 9.1.2024, 16) Sekä moduulissa että teollisuustietokoneessa on diagnostiikka-merkkivaloja, jotka kertovat niiden tilasta. Moduulissa olevat liittimet on esitetty kuvassa 8. Ne ovat vastakappale teollisuustietokoneen jousiliittimille.

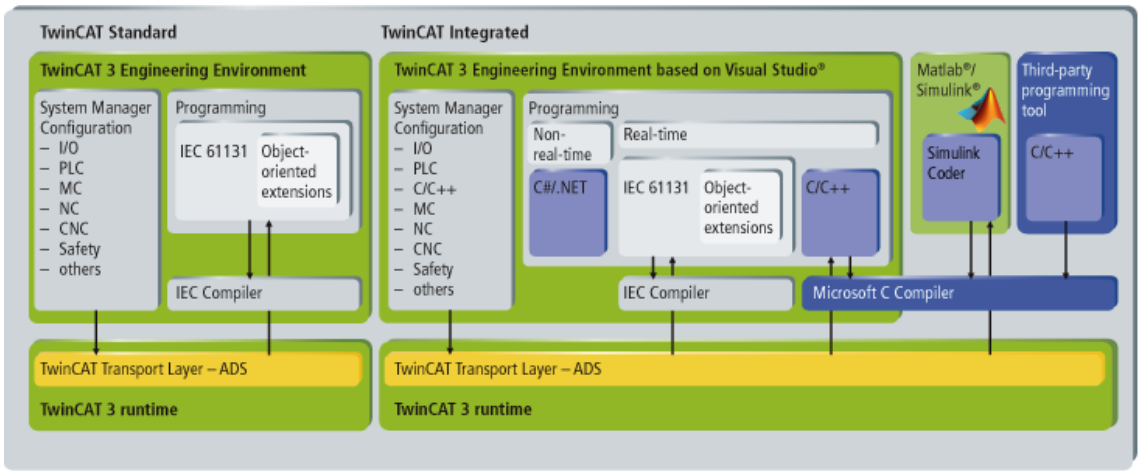


KUVA 8. Beckhoff EL6751 -moduulin liittimet. (Kuva: Antti Seitsenlinna)

TwinCAT on Beckhoffin kehittämä ohjelmistojärjestelmä automaatiosovelluksiin. Se on mahdollista asentaa lähes kaikkiin PC-pohjaisiin järjestelmiin. Järjestelmää voidaan käyttää ohjausjärjestelmänä esimerkiksi PLC-, NC-, CNC- ja robotiikka-sovelluksissa. Yksi TwinCAT-järjestelmän eduista on modulaarisuus: muutoksia ja lisäyksiä voidaan tehdä milloin tahansa lisäämällä erilaisia moduuleita. Lisäksi siihen voidaan tuoda kolmansien osapuolien komponentteja ja tiettyä tarkoitusta tai laitteita varten räätälöityjä ratkaisuja. (TwinCAT automation software n.d.)

Uusin järjestelmän versio on TwinCAT 3, jonka yksi tärkeimmistä tavoitteista on ohjelmistotuotannon yksinkertaistaminen. TwinCAT 3 onkin toteutettu Microsoft Visual Studio -ohjelmointiympäristön pohjalle laajennettavuuden sekä kestävän tulevaisuuden edistämiseksi. Järjestelmän etuna on myös, että kehittäjä tarvitsee vain yhden kehitystyökalun- eli -ympäristön. Aikaisempi versio TwinCAT 2 ja eri moduulit voidaan kaikki liittää yhteen ohjelmaan helposti projektipuun avulla. (Philosophy n.d.)

Järjestelmän arkkitehtuurissa kaikkia moduuleita käsitellään erillisinä osinaan. Osien välinen tiedonsiirto tapahtuu ADS:n (Automation Device Specification) kautta keskitetysti niin sanotun viestireitittimen kautta, joka hallitsee viestejä ja jakaa niitä järjestelmässä ja siihen yhdistetyssä laitteissa. Jokaisessa TwinCAT-laitteessa on viestireititin. (ADS device consept n.d.) TwinCAT 3 -järjestelmän toimintaa ja joitakin mahdollisia moduuleita havainnollistetaan kuviossa 2.



KUVIO 2. TwinCAT 3 -kehitysympäristön toiminta. (Philosophy n.d.)

TwinCAT 3 täyttää PLC-standardin IEC 61131-3 vaatimukset ohjelmointikieliin liittyen. (TwinCAT automation software n.d.) Standardissa määritetään syntaksi ja semantiikka ohjelmoitavissa logiikoissa käytettäville ohjelmointikielille. Yksi standardissa määritetty ohjelmointikieli on tekstipohjainen strukturoitu teksti (engl. structured text), jota käytetään tässä työssä rajapintaohjelman ja ohjelma-kirjaston toteutuksessa. (SFS-EN IEC 61131-3:2013, 9, 61)

3 CAN-VÄYLÄ

3.1 Historia

Boschilla tarkasteltiin 1980-luvun alussa olemassa olevia sarjaväyläjärjestelmiä ja niiden mahdollista käyttöä henkilöautoissa. Koska mikään verkkoprotokolla ei kyennyt täyttämään autoinsinöörien vaatimuksia, alettiin vuonna 1983 kehittää Uwe Kiencken johdolla uutta sarjaväylää. Tavoitteena oli erityisesti mahdollistaa uusien toimintojen kehittäminen, eikä niinkään esimerkiksi vähentää johdotusta. Määrittelyvaiheeseen osallistuivat myös Mercedes-Benz ja puolijohdetoimittaja Intel. Konsultiksi palkattu professori Wolfhard Lawrenz antoi uuden protokollan nimeksi Controller Area Network eli CAN. (History of CAN technology n.d.)

Uusi CAN-väylä lanseerattiin helmikuussa 1986 SAE:n kongressissa Detroitissa. Se toimi ilman keskitettyä väylänhallintaa ja perustui kilpavarausjärjestelmään, jossa suurimman prioriteetin kehys saa varata väylän. 1990-luvun alkupuolella Boschin CAN 2.0 -spesifikaatiota esitettiin kansainvälisesti standardoitavaksi. Useiden, erityisesti ranskalaisten autovalmistajien kehittämään VAN-väylään (Vehicle Area Network) liittyneiden poliittisten kiistojen jälkeen julkaistiin vuoden 1993 marraskuussa ISO 11898 -standardi. Standardia laajennettiin vuonna 1995, kun siihen lisättiin kuvaus laajennetusta 29-bittistä CAN-tunnistetta käyttävästä kehysmuodosta. (History of CAN technology n.d.)

Vaikka CAN-väylä kehitettiin alun perin käytettäväksi henkilöautoissa, sen monet ensimmäiset sovellukset tulivat muilta aloilta. Pohjois-Euroopassa suomalainen hissivalmistaja Kone ja ruotsalainen suunnittelutoimisto Kvaser hyödynsivät sitä jo sen alkuaikoina. (History of CAN technology n.d.) Monet valmistajat ja tahot ovat kehittäneet CAN-pohjaisia korkeamman tason protokollia, jotka lisäävät puuttuvia ominaisuuksia. Esimerkiksi teollisuusautomaatiossa käytetyt CANopen ja DeviceNet pohjautuvat CAN-protokollaan. Lisäksi CAN-väylää sovelletaan muun muassa energianhallinnassa, merenkulussa, ilmailussa sekä satelliiteissa. (CiA and international standardization n.d.)

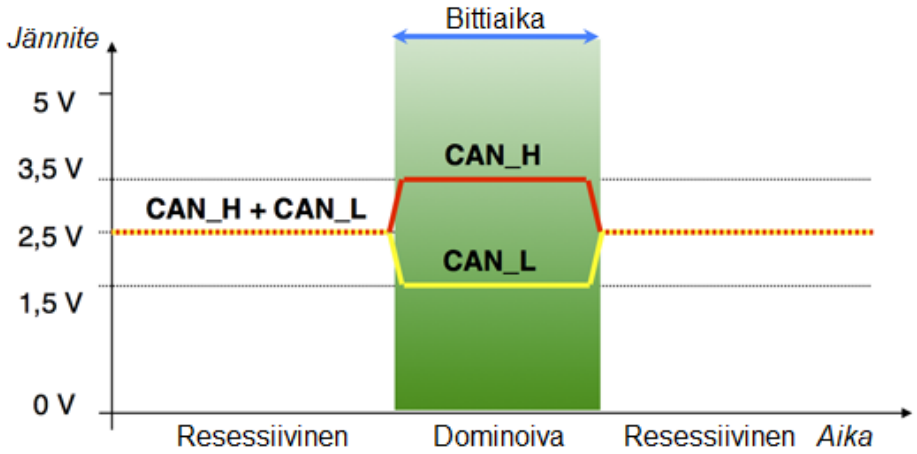
3.2 Toimintaperiaate

CAN-väylää käytetään joustavuutensa takia monenlaisissa eri käyttökohteissa. Sen kaapelointi on yksinkertainen ja väylään voidaan liittää rakenteensa ansiosta teoriassa rajaton määrä laitteita eli solmuja, minkä lisäksi CAN-väylä kestää hyvin ulkoisia häiriöitä. (Classical Controller Area Network (CAN) n.d.)

Seuraavissa alaluvuissa käydään läpi CAN-väylän rakennetta sekä tiedonsiirron toimintaa ja kehystyyppejä. Yksityiskohdat ja vaatimukset väylälle on määritetty standardissa ISO 11898 ja sen laajennuksissa. Lisäksi käydään läpi väylän eri virheentunnistus ja -hallintamenetelmiä.

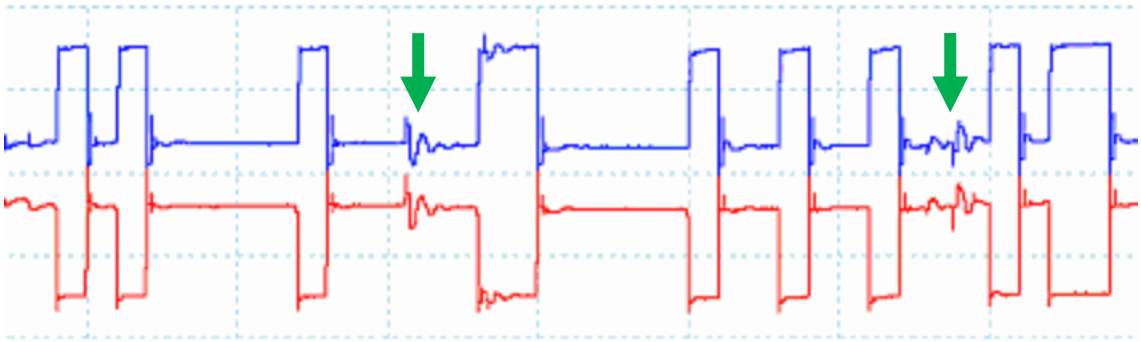
3.2.1 Rakenne

CAN-väylässä käytetään non-return-to-zero (NRZ) -linjakoodausta, eli siinä ei ole erikseen lepotilaa. Se tarkoittaa, että sekä väylän dominantti että resessiivinen tila esitetään tiettyinä signaalitasoina. Loogiset tilat ovat päinvastaiset tyypilliseen loogiseen toimintaan nähden. Dominanttia tilaa vastaa CAN-väylässä looginen 0 ja resessiivistä looginen 1. Tyhjänä väylä on resessiivisessä tilassa. Jo yhden solmun dominantti tila vaikuttaa dominoivasti koko väylän tilaan riippumatta sen resessiivisten solmujen määrästä. Perinteisessä CAN-väylässä on kaksi johdinta, CAN_High ja CAN_Low, joiden välisen jännite-eron perusteella väylän looginen tila määräytyy kuvion 3 mukaisesti. (The CAN Bus Protocol Tutorial n.d.)



KUVIO 3. CAN-väylän signaalitasojen toiminta. (CAN high-speed transmission n.d., muokattu)

Resessiivisessä tilassa molemmat johtimet ohjataan 2,5 V jännitetasoon, jolloin niiden välillä ei ole jännite-eroa. Dominantissa tilassa CAN_High ohjataan 3,5 V ja CAN_Low 1,5 V jännitetasoon, minkä seurauksena johtimien välille muodostuu 2 V jännite-ero. Jännite-eroon perustuva rakenne vähentää sähkömagneettisen häiriön vaikutusta. Mahdollinen häiriö kohdistuu kumpaankin johtimeen, jolloin jännite-ero säilyy samana. Kuviossa 4 esitetään CAN-väylään kohdistuva sähkömagneettinen häiriö ja sen vaikutus johtimiin. (CAN high-speed transmission n.d.)



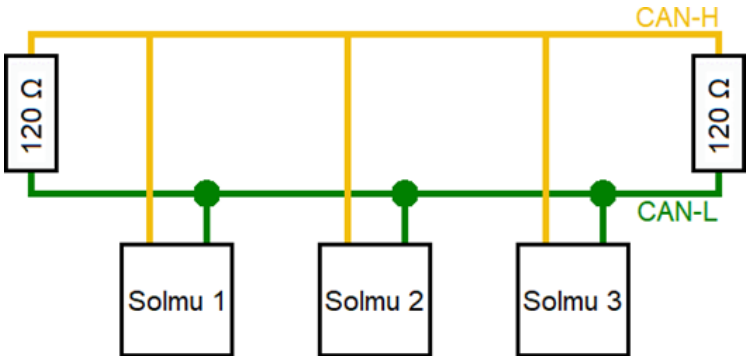
KUVIO 4. Oskilloskooppikuvio CAN-väylään kohdistuvasta sähkömagneettisesta häiriöstä. Häiriö vaikuttaa kumpaankin johtimeen. (Mark Banks 2001, muokattu)

CAN-väylän tiedonsiirtonopeus on standardin mukaan enintään 1 Mbit/s. Tällä nopeudella kaapelin pituus saa olla enintään 40 metriä, mikä johtuu seuraavassa alaluvussa käsiteltävästä sovittelumenettelystä. Se edellyttää, että signaalin on kyettävä etenemään kaukaisimpaan solmuun ja takaisin ennen signaalin bittien käsittelyä. Kaapelin pituutta rajoittaa toisin sanottuna valon nopeus. Taulukossa 1 on lueteltu kaapelien enimmäispituuksia erilaisilla yleisillä väylänopeuksilla. (The CAN Bus Protocol Tutorial n.d.)

TAULUKKO 1. Kaapelien enimmäispituudet eri väylänopeuksilla. (The CAN Bus Protocol Tutorial n.d.)

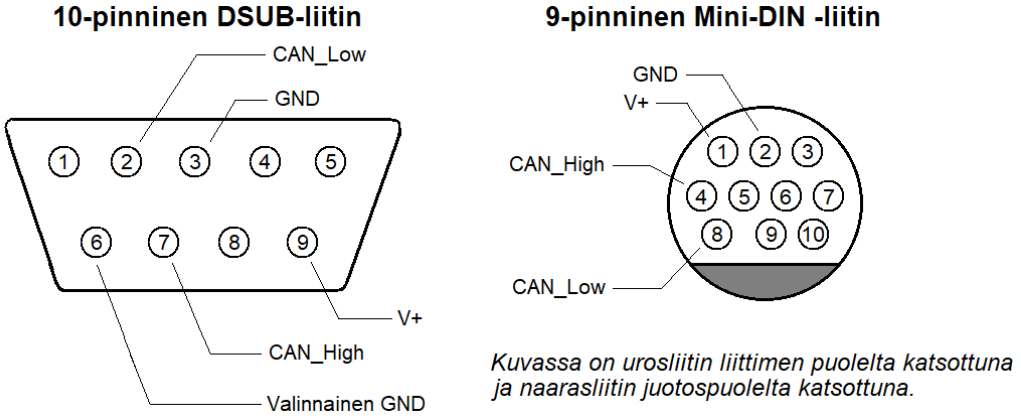
Nopeus (kbit/s)	**Kaapelin pituus (m)**
1000	40
500	100
250	250
125	500
50	1000
10	6000

Johdinpari on ISO 11898 -standardin mukaan kierrettävä riippumatta siitä, onko se häiriösuojattu vai ei. Väylä on päätettävä 120 ohmin vastuksella kummastakin päästä signaalin heijastusten poistamiseksi ja jännitetasojen varmistamiseksi. Kaapelin nimellisimpedanssin on oltava 120 ohmia. Kuviossa 5 on esitetty esimerkki väylämuotoisesta topologiasta. Muita mahdollisuuksia ovat esimerkiksi rengas- ja tähtitopologiat. (The CAN Bus Protocol Tutorial n.d.)



KUVIO 5. Esimerkki CAN-väylän topologiasta. (Kuvio: Antti Seitsenlinna)

Liittimen mallia ei standardissa ole määritelty (The CAN Bus Protocol Tutorial n.d.). Tämän työn kaapelissa on EL6751:n päässä 9-pinninen DSUB-naarasliitin ja Märklin-ratayksikön päässä 10-pinninen Mini-DIN-urosliitin. Liittimet on esitetty pinneineen kuviossa 6. Vaikka liittimessä olisi useampikin pinni, käytetään niistä vain neljää. Loput ovat tyhjiä pinnejä tai varalla.

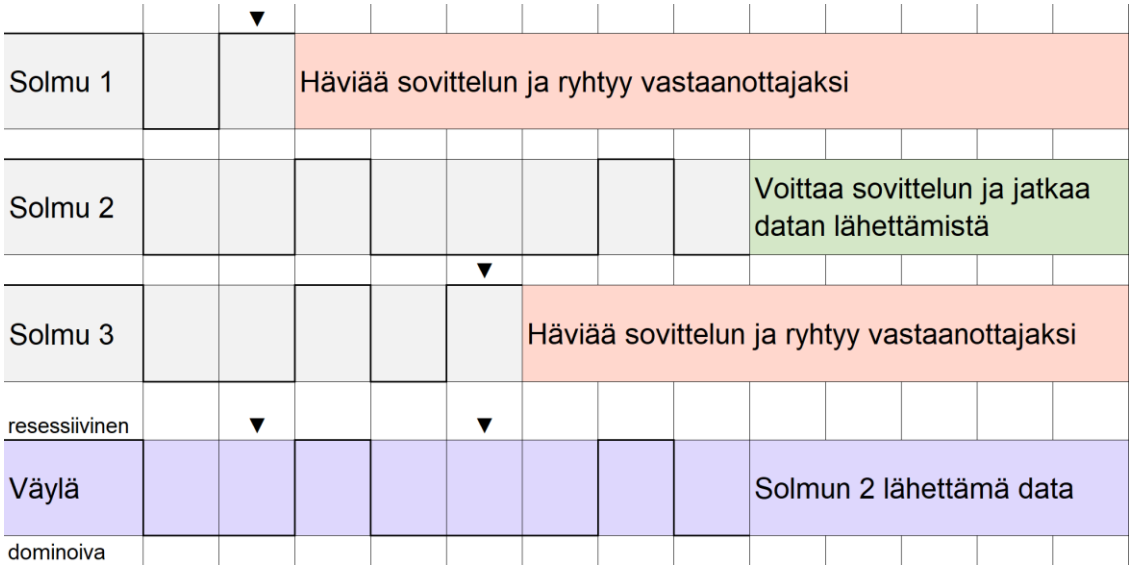


KUVIO 6. DSUB- ja Mini-DIN-liittimet pinneineen. (Kuvio: Antti Seitsenlinna)

Neljästä pinnistä kahta käytetään CAN_High- ja CAN_Low-datajohtimille ja yhtä GND-johtimelle. Datajohtimien jännite mitataan suhteessa GND-johtimeen. Urosliittimiin syötetään pinniin V+ väylän syöttöjännite. Se voidaan tarvittaessa jättää pois, jos väylän laitteet eivät tarvitse sitä (The CAN Bus Protocol Tutorial n.d.).

3.2.2 Tiedonsiirto väylässä

Kaikki väylän solmut voivat lähettää ja vastaanottaa dataa. CAN-väylä perustuu yleislähetykseen, jossa jokainen solmu vastaanottaa väylään lähetetyn datan eli kehyksen. Viesteillä ei ole varsinaista osoitetta, vaan kukin solmu päättää sisällön perusteella, onko vastaanotettu kehys sille tarpeellinen. Kehyksellä on tunniste, jonka tulisi olla yksilöllinen koko verkossa. Se paitsi kertoo kehyksen sisällöstä, määrittää myös sen prioriteetin. CAN-väylässä ei ole keskitettyä väylänhallintaa tai sovittelua, vaan se perustuu kilpavarausjärjestelmään kuvion 7 mukaisesti. (Classical Controller Area Network (CAN) n.d.)



KUVIO 7. Sovittelu kilpavarausjärjestelmässä. (Kuvio: Antti Seitsenlinna)

Tilanteessa, jossa useampi solmu lähettää kehyksen samanaikaisesti eli kilpailee väylään pääsystä, kukin solmu tarkkailee kehystä lähettäessään samalla väylän signaalitasoa bitti kerrallaan. Jos solmu havaitsee väylässä dominantin bitin ja on itse lähettämässä resessiivistä bittiä, lopettaa se oman lähetyksensä ja siirtyy vastaanottajaksi. Suurimman prioriteetin kehyksen dominantit bitit voittavat muut kehykset. Kun tämä bittikohtainen sovittelu on ohi, sovittelun hävinneet solmut yrittävät lähettää kehyksensä uudelleen väylän ollessa jälleen vapaa. (Classical Controller Area Network (CAN) n.d.) Sovittelussa aina yksi solmu voittaa ja jatkaa datan lähettämistä ilman viiveitä. Bittikohtaisen sovittelun onnistumisen kannalta on tärkeää, ettei useammalla solmulla ole samaa prioriteettia eli sovittelukehystä. Kehyksen, joka ei sisällä datatavuja, voi kuitenkin lähettää mikä tahansa solmu. (The CAN Bus Protocol Tutorial n.d.)

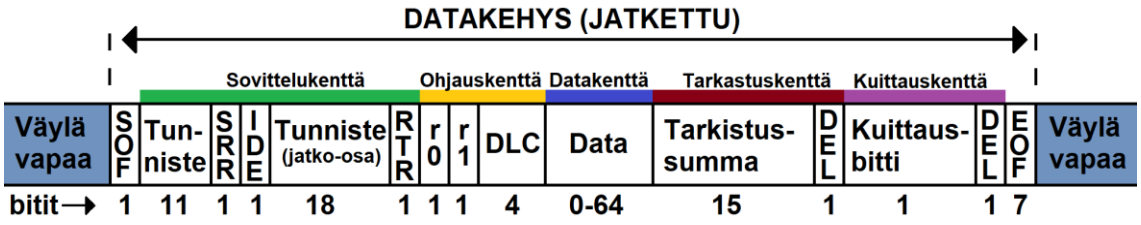
Keskeinen osa CAN-protokollaa on virheenkäsittely eli väylän virhetilanteiden tunnistaminen ja hallinta. Virheenkäsittelyn tavoitteena on havaita virheet, jotta solmu voi yrittää lähettää virheellisen kehyksen uudelleen. Jokainen solmu pyrkii kehyksen vastaanottaessaan havaitsemaan siinä mahdollisesti olevat virheet. Jos solmu löytää virheen, se lähettää väylälle virhemerkin (engl. error flag), josta muut solmut tietävät hylätä kehyksen. (The CAN Bus Protocol Tutorial n.d.)

Virheentunnistustapoja on määritelty protokollassa viisi erilaista. Virhemerkillä voidaankin kertoa seuraavanlaisista eri virhetilanteista:

- Solmut tarkkailevat kehystä lähettäessään samalla väylän signaalitasoa. Jos väylän signaalitaso eroaa lähetetystä bitistä, lähettää solmu bittivirheen.
- Solmut lisäävät jokaisen lähettämänsä viiden samantasoisen bitin jälkeen vastakkaisen tason täytebitin (engl. stuff bit). Jos väylällä havaitaan enemmän kuin viisi perättäistä saman tasoista bittiä, havaitsija lähettää täytebittivirheen.
- Tietyt kehyksen osat ovat kaikissa kehyksissä samanlaiset ja ne on määritetty standardissa. Vääränlaisen osan havaitsija lähettää kehysvirheen.
- Kaikkien kehyksen vastaanottavien solmujen odotetaan lähettävän kuittaus vastaanotosta. Jos lähettäjä ei havaitse erityistä kuittausbittiä vastauksessa, se lähettää kuittausvirheen.
- Jokaisessa kehyksessä on 15-bittinen tarkastussumma. Jos kehyksen oma tarkastussumma eroaa vastaanottajan sille laskemasta tarkastussummasta, lähettää vastaanottaja tarkastusvirheen. (The CAN Bus Protocol Tutorial n.d.)

3.2.3 Kehystyypit

Perinteisessä CAN-väylässä on neljä eri kehystyyppiä: datakehys, datapyyntökehys, vikakehys sekä ylikuormituskehys. Niistä yleisimmin käytetään data- ja datapyyntökehystä, joista on kahta eri kehysmuotoa. Niin kutsuttu normaalikehys sisältää 11-bittisen ja jatkettu kehys 29-bittisen tunnisteen. Kehysmuodoista ja niitä hyödyntävistä laitteista käytetään nimiä CAN 2.0A ja CAN 2.0B. Datakehys koostuu sovittelu-, ohjaus-, data-, tarkastus- sekä kuittauskentästä. Rakenne on sama myös datapyyntökehyksellä, jolta vain puuttuu datakenttä. Kehyksen alussa ja lopussa on resessiiviset SOF- ja EOF-osat. Jatkettu datakehys ja sen kentät on esitetty kuviossa 8. (The CAN Bus Protocol Tutorial n.d.)



KUVIO 8. Jatkettu datakehys ja sen kentät. (Kuvio: Antti Seitsenlinna)

Sovittelukenttä koostuu 11- tai 29-bittisestä tunnisteesta sekä RTR-bitistä, joka on datakehyksellä dominantti ja datapyyntökehyksellä resessiivinen. Tunnisteen 18-bittinen jatko-osa on jatketussa kehyksessä erillisenä osana, minkä ansiosta eri kehysmuotojen alku on samanlainen ja ne tukevat samaa sovittelumenettelyä. Tästä syystä myös jatko-osan takia siirtyvä RTR-bitti on korvattu resessiivisellä SRR-bitillä. Normaalikehyksessä IDE-bitti on ohjauskentässä ja dominantti, kun taas laajennetussa kehyksessä se sijaitsee sovittelukentässä ja on resessiivinen. (Classical Controller Area Network (CAN) n.d.)

Ohjauskentässä on myös 4-bittinen DLC-pituuskoodi, joka kertoo datakentässä olevien datatavujen määrän. Datatavuja voi datakentässä olla 0…8. Tarkastus-kentässä on 15-bittinen tarkistussumma kehyksen eheyden varmistamiseksi. Kuittausbitti on bitti, jolla solmut kuittaavat vastaanotetun viestin. Jos kehystä ei kuitata, lähettäjä voi yrittää lähettää sen uudelleen. Bitit r0 ja r1 eivät ole käytössä ja ovat varalla. DEL-bitit toimivat erottimina lyhyiden kenttien välissä. (Classical Controller Area Network (CAN) n.d.; The CAN Bus Protocol Tutorial n.d.)

Kahta muuta tyyppiä, vika- ja ylikuormituskehystä, käytetään poikkeustilanteissa. Solmu lähettää vian havaitessaan vikakehyksen, jonka seurauksena myös muut solmut havahtuvat vikaan. Lähettäjä yrittää sen jälkeen automaattisesti lähettää kehyksensä uudelleen. Vikakehys sisältää 6-bittisen virhemerkin ja 8-bittisen DEL-osan eli erottimen. Erotin antaa väylässä muille solmuille tilaa lähettää omat virhemerkkinsä.

Ylikuormituskehys lähetetään solmun ollessa ylikuormittunut. Kehystä käytetään kuitenkin enää harvoin ohjainten kehityttyä. Yksi harvoista ylikuormituskehystä käyttävistä CAN-ohjaimista on Intel 82526. Kuten vikakehys, ylikuormituskehys koostuu vastaavasti 6-bittisestä ylikuormitusmerkistä ja 8-bittisestä erotusosasta. (The CAN Bus Protocol Tutorial n.d.)

3.3 Käyttö Märklin Digitalissa

Märklin Digital hyödyntää CAN-väylää rata- ja ohjausyksiköiden välisessä tiedonsiirrossa. Tällä yhtenäistetään kommunikaatiota, sillä ratayksiköt puolestaan käyttävät useita eri rataprotokollia keskustellessaan edelleen radan laitteiden kanssa. Väylän nopeudeksi on määritetty 250 kbit/s. Tiedonsiirrossa käytetään vain datakehystä ja tarkemmin sen jatkettua muotoa, eikä muita kehystyyppejä tueta. Ohjauskäskyt ja erilaiset päivitys- ja konfigurointitiedot välitetään viesteillä, jotka CAN-protokollan mukaan koostuvat 29-bittisestä tunnisteesta, 4-bittisestä pituuskoodista sekä enintään kahdeksasta datatavusta. Märklin jakaa tunnisteen neljään osaan: prioriteettiin, komentoon, R-bittiin sekä hajautus- eli hash-arvoon. (Kommunikationsprotokoll 7.2.2012, 5, 7) Viestin muoto on esitetty taulukossa 2.

TAULUKKO 2. CAN-viestin muoto Märklin Digitalissa. (Kommunikationsprotokoll 7.2.2012, 5)

29-bittinen tunniste				**Pituuskoodi**	**Datatavut**
Prioriteetti	Komento	R-bitti	Hajautusarvo	Pituuskoodi	0…8 tavua
2+2 bittiä	8 bittiä	1 bitti	16 bittiä	4 bittiä	8 bittiä/tavu

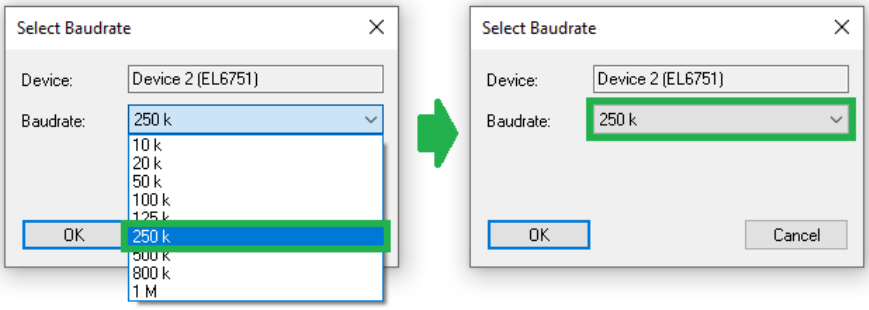
Tunnisteen prioriteetti osallistuu viestin kehyksen prioriteetin muodostumiseen väylässä. Prioriteettia seuraa kahdeksan bitin komento-osa, jossa on suoritettava tai suoritettu komento. R-bitti eli vastausbitti kertoo, onko viesti pyyntö vai vastaus aikaisempaan pyyntöön: kun vastaanottaja on suorittanut pyynnön, se lähettää viestin kopion tai pyydetyt arvot dominantin vastausbitin kanssa vahvistuksena. Vastausbitin rooli on merkittävä, sillä CAN-väylässä viestin vastaanottoa ei voida muuten vahvistaa. Hajautusarvon päätarkoitus on estää ristiriitoja eri laitteiden ja viestien välillä. Se voi myös sisältää datan sekvenssinumeron, jos datan siirtoon tarvitaan useampi viesti. Hajautusarvon tulisi olla uniikki kullakin väylän laitteella.

Pituuskoodi kertoo luvun 3.2.3 mukaisesti datatavujen määrän. Datatavuja voi olla enintään kahdeksan, ja ne sisältävät tietoa suoritettavasta komennosta tai toiminnosta. Tällaisia tarkempia tietoja ovat esimerkiksi asetusarvo tai veturin osoite. (Kommunikationsprotokoll 7.2.2012, 5–7) Viestin muodon tunteminen on tärkeää työn myöhemmässä vaiheessa rajapintaohjelmaa luodessa, sillä kaikki komennot, ohjaukset ja pyynnöt lähetetään samaa viestimuotoa käyttäen.

4 RAJAPINTAOHJELMA

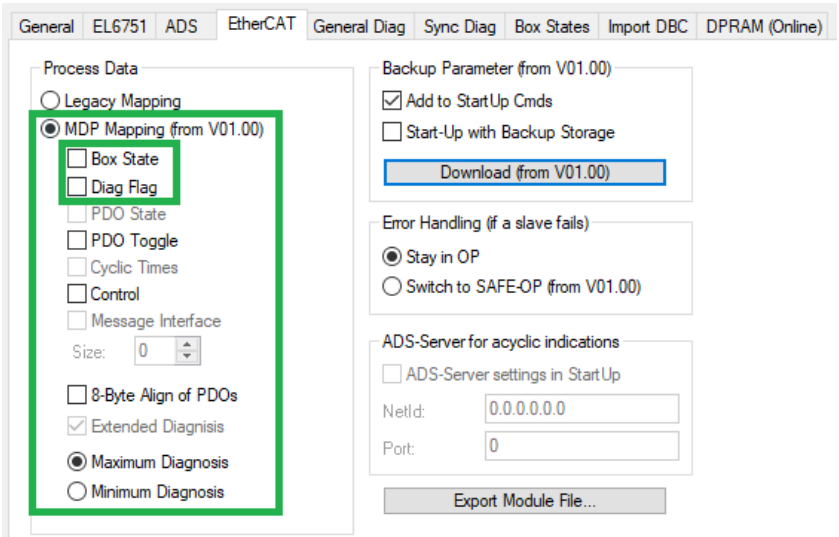
4.1 Moduulin konfigurointi

Työ aloitettiin luomalla tyhjä projekti TwinCAT 3 -kehitysympäristöön. Työssä käytettävä kohdelaite CX5140 valittiin yläpalkin pudotusvalikosta, minkä jälkeen siihen liitetty EL6751-moduuli saatiin lisättyä projektipuun I/O-kohdan skannaustoiminnolla. Lisäyksen yhteydessä valittiin väylän nopeudeksi 250 kbit/s Märklinin määritelmän mukaisesti. Valinta on esitetty kuvassa 9.



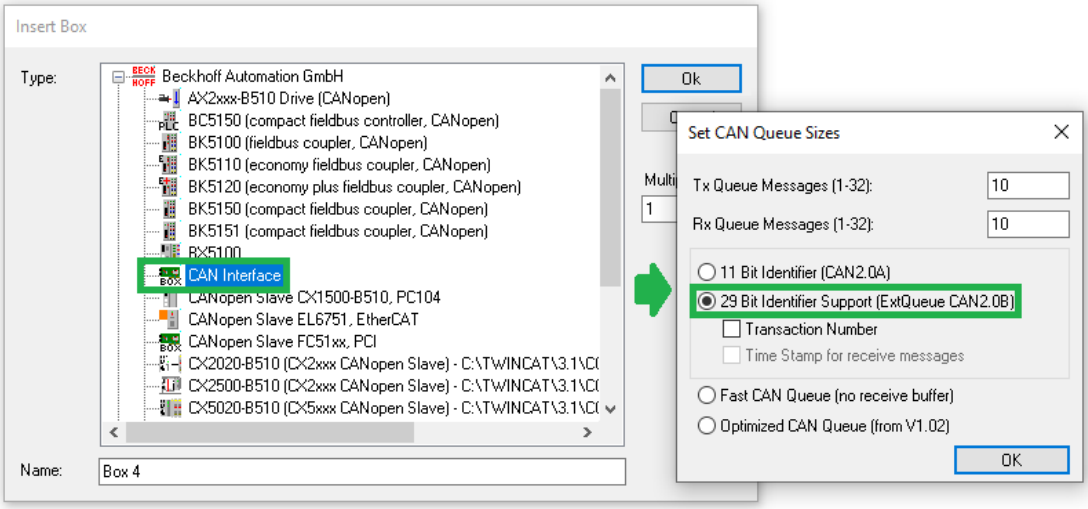
KUVA 9. Väylän nopeuden valitseminen. (Kuva: Antti Seitsenlinna)

Lisätyn EL6751-moduulin asetukset avattiin kaksoisnapauttamalla sitä projektipuussa. Sitten valittiin EtherCAT-välilehdellä ”MDP Mapping (from V01.00)” ja poistettiin sen alta valinnat ”Box State” ja ”Diag Flag” kuvan 10 mukaisesti. Muut asetukset säilytettiin ennallaan.



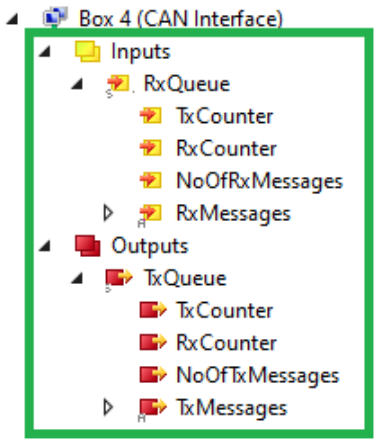
KUVA 10. Moduulin asetusmuutokset. (Kuva: Antti Seitsenlinna)

Moduulille luotiin CAN-rajapinta napauttamalla sitä hiiren oikealla ja valitsemalla ”Add New Item”. Avautuvasta valikosta valittiin ”CAN Interface”. Seuraavassa asetusnäkymässä valittiin ”29 Bit Identifier Support (ExtQueue CAN 2.0B)”, koska Märklinin laitteet käyttävät jatkettua kehysmuotoa. Jonon eli puskurin kokona oli oletuksena 10 viestiä, mikä on tämän työn tarkoitukseen sopiva. CAN-rajapinnan lisääminen ja sen asetukset on esitetty kuvassa 11.



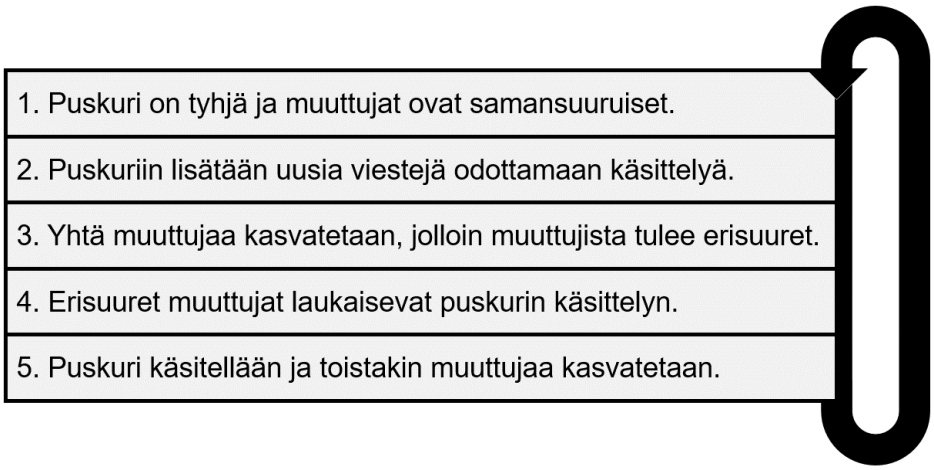
KUVA 11. CAN-rajapinnan lisääminen. (Kuva: Antti Seitsenlinna)

Moduuli sai lisätyn CAN-rajapinnan myötä tulo- ja lähtömuuttujat. Nämä muuttujat linkitetään myöhemmin rajapintaohjelman muuttujiin, ja niitä käytetään viestien ja moduulin tilan välittämiseen ohjelman ja moduulin välillä. Tulo- ja lähtöpuolien samannimiset muuttujat toimivat yhdessä ikään kuin muuttujapareina. Moduuliin saapuvat viestit ovat RxMessages-muuttujassa tulopuolella ja lähtevät viestit TxMessages-muuttujassa lähtöpuolella. Tulot ja lähdöt on esitetty kuvassa 12.



KUVA 12. CAN-rajapinnan tulo- ja lähtömuuttujat. (Kuva: Antti Seitsenlinna)

Tulopuolen NoOfRxMessages-muuttuja kertoo vastaanottopuskurissa käsittelyä odottavien viestien määrän. Vastaavasti taas lähtöpuolella NoOfTxMessages-muuttujaan kirjoitetaan rajapintaohjelmassa puskuriin lisättyjen viestien määrä. RxMessages- ja TxMessages ovat puskureita käsittelyä odottaville viesteille. CAN-rajapinnan lisäysvaiheessa niiden kooksi määritettiin kymmenen viestiä, eli RxMessages voi sisältää enintään kymmenen lähettyvää viestiä ja TxMessages saman määrän vastaanotettuja viestejä. Kun puskurissa olevat lähtevät viestit on lähetetty ja vastaanotetut viestit käsitelty, siitä kertovat tulo- ja lähtöpuolen TxCounter- ja RxCounter-muuttujaparit kuviossa 9 esitetyllä tavalla. Ne pitävät lukua lähetetyistä ja vastaanotetuista viesteistä ja kertovat puskurien tilasta ollen joko saman- tai erisuuruiset.



KUVIO 9. TxCounter- ja RxCounter-muuttujaparien toimintaperiaate. (Kuvio: Antti Seitsenlinna)

TxCounter-muuttujaa kasvatetaan lähtöpuolella yhdellä, kun puskurissa on uusia viestejä lähetettäväksi. Vastaavasti sitä kasvatetaan tulopuolella, kun puskurissa olevat viestit on käsitelty. Muuttujia voidaan verrata rajapintaohjelmassa, ja kun ne ovat samansuuruiset, tiedetään puskurin olevan tyhjä uusia viestejä varten. RxCounter-muuttujat toimivat myös parina kertoen vastaanottopuskurin tilasta. Muuttujat ovat yhtä suuret puskurin ollessa tyhjä. Kun puskuriin vastaanotetaan uusia viestejä, kasvatetaan tulopuolen muuttujaa, jolloin muuttujat ovat erisuuret ja rajapintaohjelma tietää käsitellä puskurin. Kun viestit on käsitelty, lähtöpuolen muuttujaa kasvatetaan ja muuttujat ovat jälleen yhtä suuret.

4.2 Rajapinnan perustoiminnallisuus

Koska EL6751-moduuli on tehty ensisijaisesti CANopen-käyttöön, sen käytöstä perinteisessä CAN-väylässä löytyi vain vähän tietoa ja esimerkkejä. Varsinainen rajapintaohjelma eli tulevan ohjelmakirjaston viestit lähettävä ja vastaanottava toiminnallisuus piti rakentaa itse. Märklinin kattavan dokumentaation ansiosta oli jo alusta asti selvää, millaisia viestejä moduulista halutaan ratayksikölle lähettää. Ongelmaksi muodostuikin lähinnä se, miten rajapintaohjelmasta saadaan dataa lähetettyä ja miten siitä saadaan halutun muotoista. Lisähaastetta toi käytettävä jatkettu kehysmuoto ja se, ettei aikaisempaa kokemusta aiheesta juuri ollut.

Ongelman ratkaisemiseksi lähdettiin etsimään käyttöesimerkkejä Beckhoffin oman dokumentaation tueksi. Ratkaisevaan asemaan nousi saksalainen PLC-foorumi SPS-Forum, jossa aihetta oli puitu muutamassa eri keskusteluketjussa. Ketjuissa oli jaettu myös esimerkkitiedosto, jonka pohjalta oli mahdollista oppia yrityksen ja erehdyksen kautta avainasioita rajapintaohjelmaa varten. Sen lisäksi paljon hyötyä oli myös Beckhoffin sähköpostitse toimittamasta esimerkistä, vaikka kyseinen ohjelma olikin tehty hieman erilaiseen käyttötarkoitukseen.

Kun esimerkkien pohjalta oli saatu muodostettua ohjelma viestien lähettämiseksi, sitä testattiin ja kehitettiin seuraamalla väylää National Instrumentsin NI-9862 -moduulin ja NI-XNET Bus Monitor -analyysityökalun avulla. Niiden avulla saatiin hiottua datasta juuri oikeanlaista Märklinin ratayksikköä varten. Kuvassa 13 on esitetty NI-9862-moduuli ja sen kanssa käytetty NI cDAQ-9171-USB-runko.

KUVA 13. NI-9862-moduuli ja NI cDAQ-9171-runko (Kuva: Antti Seitsenlinna)

Rajapintaohjelman perustoiminnallisuus osoittautui jälkikäteen yksinkertaiseksi ja vastasi Beckhoffin dokumentaatiossa annettua esimerkkiä. Alussa oli kuitenkin haastavaa kokemattomuuden takia ymmärtää kaikkia yksityiskohtia. Petollisinta oli, kun luuli virheellisesti ymmärtäneensä tietyn asian toiminnan. Tämä pitkitti välillä turhaan selvittelyä. Ohjelmakoodi viestien lähetyksen perustaksi oli lopulta

```
IF Outputs.TxCounter = Inputs.TxCounter THEN
    Outputs.TxMessages[0].cobId := 16#80086751;
    Outputs.TxMessages[0].length := 5;
    Outputs.TxMessages[0].data := dataOut;

    Outputs.TxCounter := Outputs.TxCounter + 1;
    Outputs.NoOfTxMessages := 1;
END_IF
```

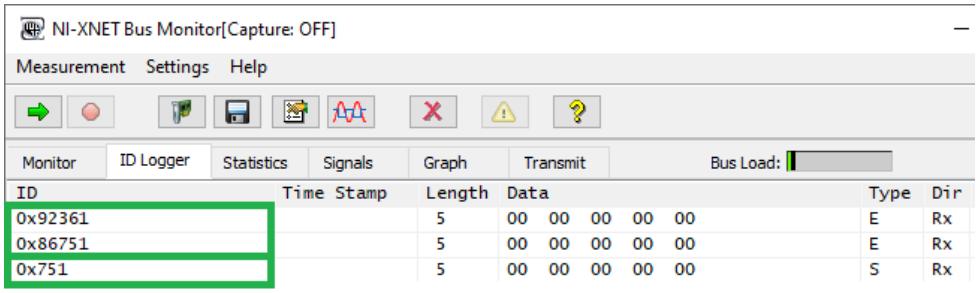
jossa tunnistetaan TxCounter-muuttujaparin avulla puskurin tyhjä tila. CAN-viesti rakennetaan tämän jälkeen cobId-, length-, ja data-lähtömuuttujilla. Nimiensä mukaisesti length-muuttujaan kirjoitetaan pituuskoodi ja data-muuttujaan viestin datatavut. Tunniste on 32-bittisen cobId-muuttujan ensimmäisissä 29:ssä bitissä kolmen viimeisen bitin jäädessä ylimääräisiksi. Kun viesti on rakennettu ja valmis lähetettäväksi, TxCounter-muuttujaa kasvatetaan yhdellä käsittelyn odottamisen merkiksi ja NoOfTxMessages-muuttujalla kerrotaan puskurissa olevan yksi viesti. Vastaanoton perustana toimivaksi ohjelmakoodiksi taas muodostui vastaavasti

```
IF Outputs.RxCounter <> Inputs.RxCounter THEN
    FOR i := 0 TO (Inputs.NoOfRxMessages-1) DO
        ReceivedMessage[i] := Inputs.RxMessages[i];
    END_FOR

    Outputs.RxCounter := Outputs.RxCounter+1;
END_IF
```

jossa RxCounter-muuttujapari kertoo puskurissa olevan käsiteltävää. Puskurista jokainen vastaanotettu viesti siirretään ReceivedMessage-taulukkomuuttujaan For-silmukan avulla. Puskuri vapautetaan kasvattamalla lähtöpuolen RxCounter-muuttujaa yhdellä, jolloin siitä tulee samansuuruinen parinsa kanssa.

Tunnisteesta ylimääräisiksi jääneistä cobId-lähtömuuttujan biteistä viimeinen on viestin kehysmuodon kertova bitti. Se asetetaan normaalikehystä käytettäessä arvoon 0 ja jatkettua kehystä käytettäessä arvoon 1, jotta moduuli tietää lähettää pidemmän 29-bittisen tunnisteen. Tämän bitin toiminta opittiin kantapään kautta

väylää seuraamalla. Se jätettiin aluksi arvoon 0, minkä seurauksena tunnisteet olivat liian lyhyitä. Kuvassa 14 on esitetty NI-XNET Bus Monitor -työkalun näkymä lähetysohjelmaa suoritettaessa, kun kehyksen määrittävä bitti on arvoissa 0 ja 1.



KUVA 14. Näkymä NI-XNET Bus Monitor -analyysityökalusta, kun lähetetään viesti kehysmuodosta kertovan bitin eri arvoilla. (Kuva: Antti Seitsenlinna)

Testiohjelman lähettämät viestit tunnistaa niiden 751-loppuisesta tunnisteesta, mikä on asetetun heksadesimaalimuotoisen tunnisteen loppuosan hajautusarvo. Kuvasta 13 huomataan, että alin tunniste 751 lyhyempi kuin sen yläpuolella oleva tunniste 86751. Tämä johtuu juuri kehyksen määrittävän bitin väärästä arvosta. Ylempi viesti sai myös tunnisteella 92361 vastauksen Märklinin ratayksiköltä, mikä on hyvä merkki viestin oikeasta muodosta.

4.3 Rajapintaohjelman luominen

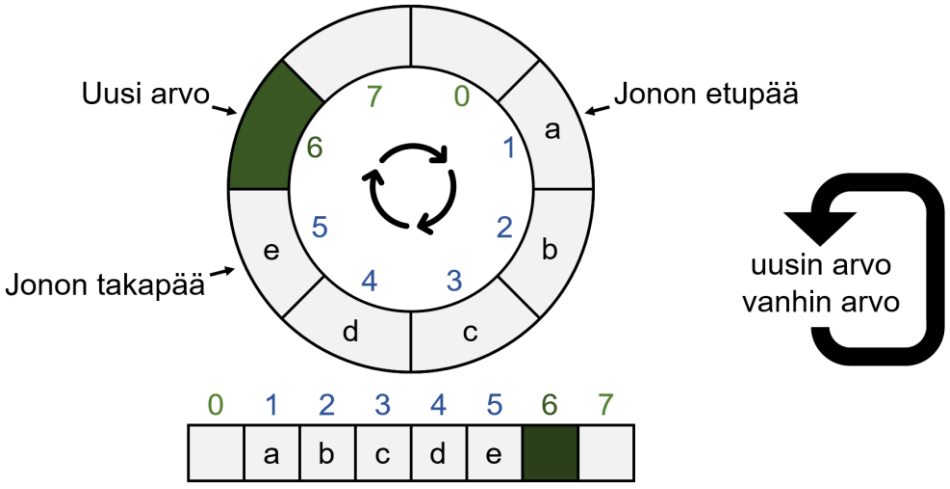
Ohjelmakirjaston taustalla toimii CAN-liikenteestä huolehtiva rajapintaohjelma. Sen tehtävänä on lähettää ja vastaanottaa viestejä ja tarvittaessa myös vahvistaa niiden perillemeno. Ohjelman tulee toimia luotettavasti, koska se muodostaa koko ohjelmakirjaston ja Märklin-pohjaisten laitteiden ohjauksen perustan. Siksi sille tehtiin myös toimintatestejä, jotta voitiin varmistua riittävän tehokkaasta ja varmasta toiminnasta.

Varsinainen rajapintaohjelma muodostui haasteiden ratkettua toimivien lähetys- ja vastaanotto-ohjelmakoodien pohjalta, kun niitä lähdettiin viemään eteenpäin. Seuraavissa alaluvuissa käydään läpi rajapintaohjelman keskeiset rakenteet ja muuttujat sekä tutustutaan ohjelmointi- ja testausprosessiin. Rajapintaohjelmasta saatiin lopulta tehtyä toimiva ja tehokas kokonaisuus, jonka päälle on hyvä alkaa rakentamaan ohjelmakirjastoa.

4.3.1 Rakenne ja toiminta

Rajapintaohjelmasta pyrittiin tekemään modulaarinen eli jakamaan sen erilaiset toiminnallisuudet järkevästi hallittaviin osiin. Tarkoituksena oli helpottaa testausta ja myöhemmin ylläpitoa, kun ohjelma jakautuu selkeästi eri osiin, joilla on omat tehtävänsä. Huomiota kiinnitettiin myös muuttujien nimeämiseen, ohjelmakoodin kommentointiin sekä johdonmukaiseen rakenteeseen. Rajapintaohjelma tehtiin strukturoitua tekstiä (ST) käyttäen, sillä se on tekstipohjainen ja sopi graafisia tikapuukaavio- ja toimilohkomuotoja paremmin monimutkaiselle kokonaisuudelle.

Tärkein rajapintaohjelman tehtävä on lähettää CAN-viestejä EL6751-moduulille ja vastaanottaa niitä siltä. Moduulissa on vain rajallisesti puskuritilaa viesteille, joten rajapintaohjelmaan piti rakentaa myös omat lähetys- ja vastaanottojonot. Ne päätettiin toteuttaa rengasmuotoisella rakenteella (eng. circular queue), jonka toimintaperiaate on esitetty kuviossa 10.

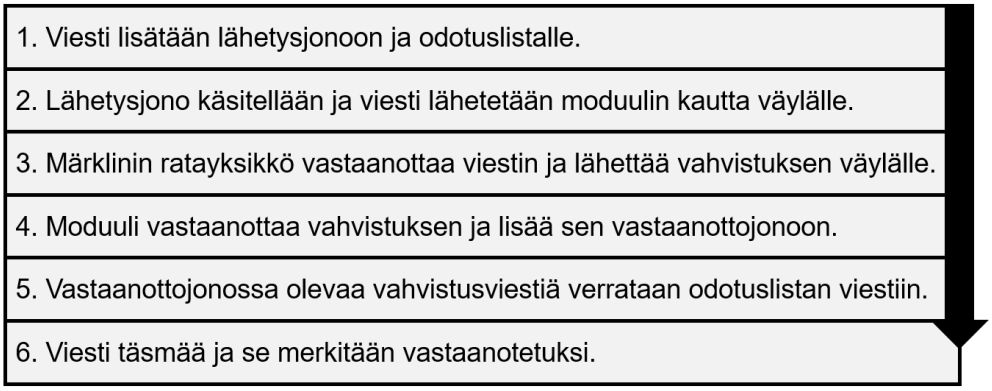


KUVIO 10. Rengasmuotoisen jonon toiminta. (Kuvio: Antti Seitsenlinna)

Tämän työn rajapintaohjelmassa rengasmuotoinen jono on toteutettu tavallisen yksiulotteisen taulukon pohjalle. Taulukon lisäksi siinä on kaksi osoitinmuuttujaa, yksi osoittaa etupään ja toinen takapään position. Jonoa voidaan käsitellä vain sen päistä: jonon etupäästä poistetaan arvoja ja takapäätä seuraavaan positioon lisätään arvoja. Juuri se on rengasmuotoisen jonon etu, sillä koko jonoa ei tarvitse päivittää tai järjestellä sitä käsiteltäessä. Jonon etupää siirtyy eteenpäin aina, kun siitä poistetaan arvo. Vastaavasti takapää siirtyy eteenpäin, kun jonoon lisätään arvo. Jono käsitellään rakenteensa ansiosta myös aina saapumisjärjestyksessä.

Jonon tila voidaan päätellä sen osoitinmuuttujista. Kun jono on täynnä, takapään positiota seuraavana on heti etupään positio. Jonoa alustettaessa tai viimeistä elementtiä poistettaessa sen molemmat osoittimet asetetaan arvoon -1 merkiksi tyhjästä jonosta. Lähetysjonossa lähetettävät viestit lisätään jonon perälle. Jonon etupäästä käsitellään puskurin koon verran viestejä kerrallaan, kunnes jono on tyhjä. Vastaanottojonossa uudet viestit lisätään jonon perälle käsittelyä varten.

Vastaanotetuista viesteistä ovat tärkeitä vain ne, jotka vastaavat ohjelmakirjaston lähettämiin viesteihin. Tällaisia viestejä ovat vahvistukset käskyn suorittamisesta tai pyydetyt arvot. Koska kaikki tarpeelliset vastaanotettavat viestit ovat tavalla tai toisella vastauksia, niitä tiedetään odottaa etukäteen. Rajapintaohjelmaan luotiin kaksi viestinlähetystoimintoa, joista toinen viestin lähettämisen lisäksi lisää sen myös erityiselle odotuslistalle. Kaikkia vastaanotettuja viestejä verrataan odotuslistalla oleviin viesteihin. Jos vastaanotetulle viestille löytyy odotuslistalta vastine, se kopioidaan odotuslistan viestin tilalle ja merkitään vastaanotetuksi. Muussa tapauksessa vastaanotettu viesti todetaan tarpeettomaksi ja poistetaan. Kuvio 11 havainnollistaa rajapintaohjelman toimintaa, kun lähetetään viestiä niin kutsutun vahvistetun lähetystoiminnon kautta, eli odotuslistaa käyttäen.



KUVIO 11. Rajapintaohjelman toimintavaiheet vahvistettua viestiä lähetettäessä. (Kuvio: Antti Seitsenlinna)

Odotuslistaa käytetään siis viestin vastaanoton vahvistamiseksi. Jos viestiin ei kuulu vastausta tietyn ajan kuluessa, se poistetaan odotuslistalta ja laukaistaan virhetila. Tämä aika on yleensä joitakin kymmeniä millisekunteja. Käytännössä Märklin-ratayksikkö vastaa usein jo muutaman millisekunnin sisällä, mutta näin nopealle reaktiolle ei ole tarvetta. Siksi odotusaikaan on päätetty jättää hieman varaa mahdollisten väylän ylikuormitustilanteiden ja muiden ongelmien varalta.

4.3.2 Toteutus

Rajapintaohjelmaa lähdettiin toteuttamaan suunnittelemalla ensin suuripiirteisesti sen rakenne eli määrittelemällä käytettävät datatyypit, muuttujat ja ohjelmalohkot. Aluksi rajapintaohjelman tarvitsemista tulo- ja lähtömuuttujista muodostettiin omat CAN_Inputs ja CAN_Outputs structure-muotoiset datatyyppinsä. Tämän jälkeen luotiin globaalit muuttujat GVL_CanRxTx, jotka on esitetty taulukossa 3.

TAULUKKO 3. Rajapintaohjelman globaalit muuttujat listassa GVL_CanRxTx.

Nimi	**Tyyppi**	**Kuvaus**
CanInputs	CAN_Inputs	Tulomuuttujat
CanOutputs	CAN_Outputs	Lähtömuuttujat
arrCanRxQueue	array	Vastaanottojono
nCanRxQueueFront	int	Jonon etupään osoitin
nCanRxQueueRear	int	Jonon takapään osoitin
arrCanTxQueue	array	Lähetysjono
nCanTxQueueFront	int	Jonon etupään osoitin
nCanTxQueueRear	int	Jonon takapään osoitin
arrCanPendingList	array	Odotuslista
nCanPendingListSize	unsigned int	Odotuslistan viestimäärä
nCAN_IO_QUEUE_SIZE	unsigned int	Moduulin puskurien koko
nCAN_RX_QUEUE_SIZE	unsigned int	Lähetysjonon koko
nCAN_TX_QUEUE_SIZE	unsigned int	Vastaanottojonon koko
nCAN_PENDING_LIST_SIZE	unsigned int	Odotuslistan koko
nHASH	word	Moduulin käyttämä hash
nDefaultTimeOut	time	Odotuslistan odotusaika

Muuttujien nimien alku kertoo muuttujan tyypistä. Pieni n-kirjain kertoo muuttujan olevan numeerinen ja arr-etuliite kertoo sen olevan array- eli taulukkotyyppinen. Isoilla kirjaimilla kirjoitetut muuttujat ovat vakioarvoja, eli ne eivät enää muutu määrittelynsä jälkeen ohjelman aikana. Muuttujat kannattaa määrittää vakioiksi silloin, kun niitä käytetään vain asetusarvoina ilman, että niitä tarvitsee muuttaa. Globaali muuttuja tarkoittaa, että muuttujaa voi käyttää kaikkialla ohjelmassa. Sen vastakohta on paikallinen muuttuja, jota voi käyttää vain tietyn lohkon sisällä.

Kun datatyypit ja globaalit muuttujat oli määritetty, alettiin luomaan varsinaista rajapintatoiminnallisuutta. Viestien siirtoon moduulin puskurien ja lähetys- sekä vastaanottojonon välillä luotiin funktio F_CanRxTx. Se toimii varsinaisena rajapintana, joka aikaisemmin kuvatun mukaisesti lähettää lähetysjonosta tietyn määrän viestejä kerrallaan ja lisää vastaanotetut vastaukset vastaanottojonon perälle. Jonojen hallintaan toteutettiin funktiot F_EnQueue ja F_DeQueue, joita voidaan kutsua, kun lähetys- tai vastaanottojonoa täytyy käsitellä. Lähetykseen luotiin vahvistamaton lähetystoiminto F_CanSendUnVerified sekä vahvistettu F_CanSend. Vahvistettu toiminto lisää lähetysjonon lisäksi viestin odotuslistalle ja odottaa siihen vastausta. Vastaanottojonon viestit käsitellään ja niitä vertaillaan odotuslistaan F_CanProcessReceiveQueue-funktiossa. Lopuksi käsitellyt viestit poistetaan jonosta.

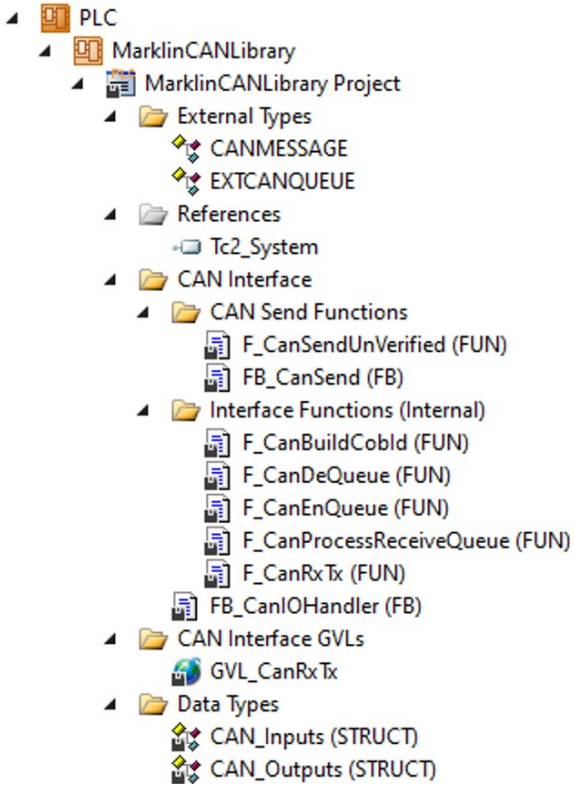
Lähetystoimintoihin syötetään sisään lähetettävä Märklin-komento, pituuskoodi sekä datatavut. Lisäksi syötetään odotusaika tai vastausbitin tila sen mukaan, kumpi lähetystoiminto on kyseessä. Varsinainen tunniste rakennetaan funktiolla F_CanBuildCobid, johon syötetään komento ja vastausbitin tila, joka vahvistetulla viestillä on aina resessiivinen. Funktio yhdistää komennon ja vastausbitin sekä aiemmin määritetyn globaalin hash-muuttujan 29-bittiseksi tunnisteeksi. Viestien prioriteettikenttä on aina sama, ”0000”. Rajapintaohjelman suorittamiseen luotiin funktio F_CanIOHandler, jota kutsutaan ohjelmakirjaston käytön yhteydessä. Kaiken kaikkiaan rajapintaohjelmaan luotiin kahdeksan eri funktiota, jotka on vielä koottu kuvauksineen taulukkoon 4.

TAULUKKO 4. Rajapintaohjelman funktiot.

Nimi	**Kuvaus**
F_CanRxTx	Siirtää viestejä moduulin ja jonojen välillä
F_EnQueue	Lisää viestin jonon takapäähän
F_DeQueue	Palauttaa ja poistaa viestin jonon etupäästä
F_CanSendUnVerified	Lähettää viestin ilman vahvistusta
F_CanSend	Lähettää viestin ja kuittaa vahvistuksen
F_CanProcessReceiveQueue	Käsittelee vastaanottojonon
F_CanBuildCobid	Rakentaa CAN-tunnisteen syötteestä
F_CanIOHandler	Suorittaa rajapintaohjelman

Aiemmin luvussa 4.2 todettiin, että CAN-viestin tunnisteesta jää kolme bittiä yli. Niistä yhtä käytettiin merkitsemään jatkettua kehysmuotoa. Kahdella muulla bitillä ei kuitenkaan ole määrättyä käyttöä. Tässä ohjelmassa niitä käytetäänkin viestin tilan määräytymiseen odotuslistalla. Molempien bittien resessiivinen tila tarkoittaa tyhjää viestiä. Kun molemmat ovat dominantteja, tarkoittaa se viestin odottavan vastausta. Vastatulla viestillä vain toinen biteistä on dominantti.

Valmiit funktiot sijoitettiin CAN Interface -kansioon, jonka sisälle luotiin vielä kaksi alikansiota selkeyden vuoksi. Lähetystoiminnot sijoitettiin kumpikin alikansioon nimeltä ”CAN Send Functions”. Rajapintaohjelman sisäiset toiminnot erotettiin alikansioon ”Interface Functions (Internal)”, koska niitä ei ole tarkoitus suorittaa ohjelmakirjastosta. Funktiota FB_CanIOHandler ei sijoitettu alikansioihin. Valmiin rajapintaohjelman rakenne esitetään kuvassa 15.



KUVA 15. Valmiin rajapintaohjelman rakenne. (Kuva: Antti Seitsenlinna)

Myös globaalit muuttujat ja datatyypit sijoitettiin omiin kansioihinsa. Viestien muuttujatyypiksi määritettiin EXTCANQUEUE ja datatavujen CANMESSAGE, mitkä ovat ulkoisia datatyyppejä. Ne ovat myös omassa kansiossaan. Ohjelma viittaa ulkoiseen Tc2_System PLC-kirjastoon. Siitä hyödynnetään joitakin perus-toimintoja kuten bittioperaatioita.

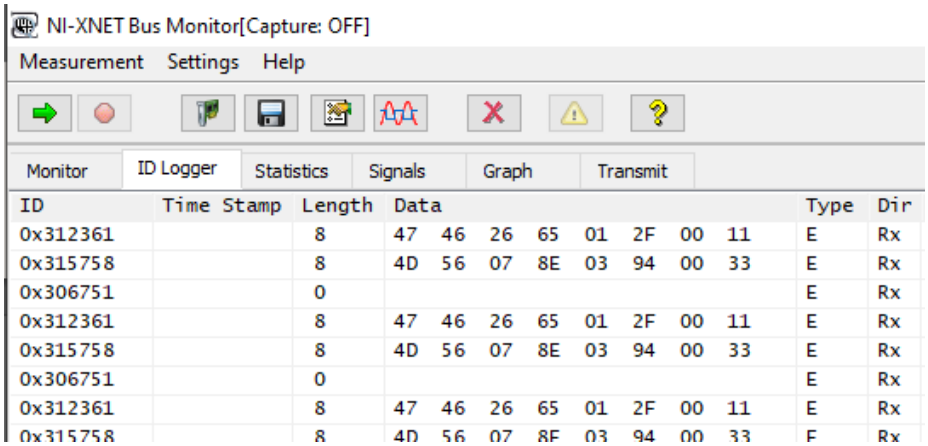
4.3.3 Valmiin ohjelman testaus

Testaus aloitettiin linkittämällä rajapintaohjelman tulo- ja lähtömuuttujat moduulin vastaaviin muuttujiin. Ohjelmaa suorittavan tehtävän sykliajaksi määriteltiin 1 ms, jotta viestit voitaisiin käsitellä tarpeeksi nopeasti. Ohjelman testaamisella pyrittiin varmistumaan sen oikeasta ja tehokkaasta toiminnasta. Testauksessa käytettiin apuna jo rajapinnan selvittelytyössä käytettyä National Instrumentsin NI-9862 -moduulia ja NI-XNET Bus Monitor -analyysityökalua. Sen avulla voitiin tarkastaa, mitä väylälle lähetetään ja saadaanko kaikkiin lähetettyihin viesteihin vastaus. Testiohjelmakoodina oli

```
FB_CanIOHandler();

nIntervalTime := nIntervalTime+1;
IF nIntervalTime-nOldIntervalTime >= 2
 AND (receiveCounter.CV OR errorCounter.CV) < 10 THEN
    bSending := TRUE;
    nOldIntervalTime := nIntervalTime;
ELSE
    bSending := FALSE;
END_IF

FB_CanSend(IN := nSending, nCommandIn := 16#18, nTimeoutIn := T#2MS);

receiveCounter(CU := FB_CanSend.RECEIVED);
nReceiveCounterValue := receiveCounter.CV;
errorCounter(CU := FB_CanSend.ERROR);
nErrorCounterValue := errorCounter.CV;
```

joka lähettää viestejä, kunnes yhteensä kymmenen viestiä on vastaanotettu tai todettu epäonnistuneeksi. Odotusaika nTimeoutIn on asetettu arvoon 2 ms, eli pienimpään mahdolliseen aikaan ottaen huomioon sykliajan 1 ms. Ensimmäisellä ohjelmakierrolla lähetetään viesti ja toisella se todetaan vastaanotetuksi. Mikäli vastaanottoa ei toisella kierrolla kuulu, todetaan kolmannella kierrolla jo viestin lähetyksen epäonnistuneen.

Tiukalla odotusajalla varmistetaan ohjelman tehokas toiminta. Viestit lähetetään myös kahden millisekunnin välein. Testikomentona oli heksadesimaalimuodossa 18 eli niin kutsuttu yhteydentestauskomento. Komento on määritetty Märklinin protokollassa. Ratayksikkö vastaa siihen lähettämällä käyttöliittymänsä version. Komento ei tee mitään muuta, eli se on hyvä testikomento. Kuvassa 16 on esitetty kuva analyysityökalusta testiohjelmaa suoritettaessa. Kuvankaappauksessa on näytetty vain muutama ensimmäinen rivi onnistuneesta testistä.



KUVA 16. Kuva analyysityökalusta testauksen aikana. (Kuva: Antti Seitsenlinna)

Vastaava testi toistettiin myös suuremmalla viestimäärällä jonojen testaamiseksi sekä varmatoimisuuden toteamiseksi. Lisäksi suoritettiin testi, jossa CAN-väylän johdin irrotettiin Märklinin ratayksiköstä ja katsottiin, seuraako siitä virhetilannetta. Kumpikin testi todettiin onnistuneeksi. Rajapintaohjelma palautui automaattisesti, kun johdin kytkettiin takaisin ratayksikköön sen irrottamisen jälkeen. Käytössä oli kaapeli, jonka toisessa päässä ei ollut liitintä. Näin johtimet voitiin irrottaa testiä varten.

Onnistuneiden testien jälkeen rajapintaohjelma todettiin valmiiksi ja toimivaksi ohjelmakirjastoa varten. Rajapintaohjelma täytti sille asetetut tavoitteet oikeasta ja tehokkaasta toiminnasta. Se sisältää tarvittavat toiminnallisuudet, jotta viestejä voidaan välittää ohjelmakirjastolta ja takaisin. Lisäksi rajapintaohjelma palauttaa testatusti kuittauksen onnistumisesta tai tiedon virheistä, mikä on tärkeää.

5 OHJELMAKIRJASTO

5.1 Määrittely

Ohjelmakirjaston rakentaminen alkoi rajapintaohjelman tapaan suunnittelulla, kun kartoitettiin siinä tarvittavat eri toiminnot. Apuna suunnittelussa hyödynnettiin Joerg Pleumannin kehittämää Railuino-kirjastoa, jolla Märklin-laitteita voi ohjata Arduino-kehitysalustalla. Myös se on tehty korvaamaan Märklinin käsiohjain, jotta pienoisrautatien toimintoja voitaisiin ohjata ohjelmallisesti. Kirjasto on jo vanha, mutta siitä voitiin kuitenkin poimia tärkeitä toimintoja toteutettavaksi tämän työn ohjelmakirjastoon. Sen ohjelmakoodia päästiin lukemaan GitHub-sivustolla.

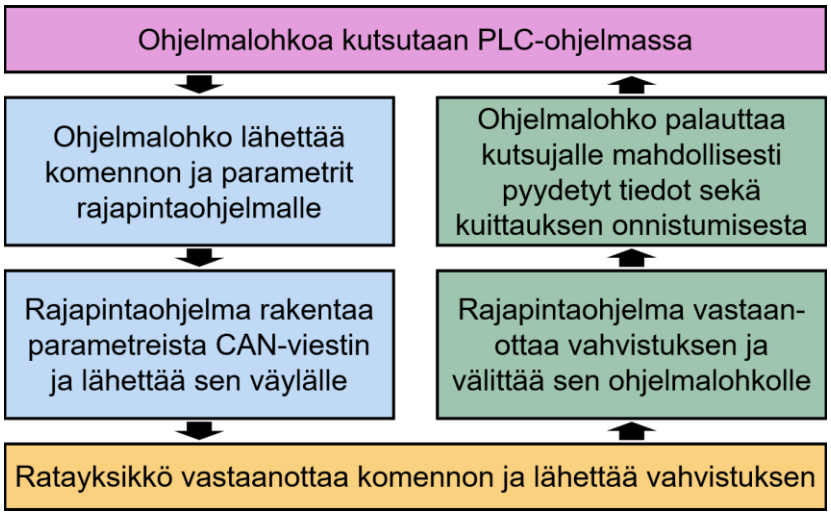
Kirjaston ohjelmakoodia ei tarvinnut suorittaa, vaan ohjelmakoodin lukeminen riitti toimintojen kartoittamiseksi. Kaikki Railuino-kirjaston sisältämät toiminnot koottiin listaksi, minkä jälkeen niistä karsittiin tarpeettomat pois. Sellaisia olivat etenkin konfigurointikomennot, joita ei päätetty toteuttaa ohjelmakirjastoon niiden vähäisen käytön takia. Kun tarpeelliset toiminnot oli listattu, yritettiin vielä pohtia mahdollisia muita tarvittavia toimintoja. Niitä ei kuitenkaan enää keksitty, vaan kaikki ohjelmakirjastoon toteutettavat toiminnot saatiin muodostettua jo kattavan Railuino-kirjaston avulla. Tuosta kirjastosta ei siis ollut unohdettu mitään tärkeää.

Ensisijaisesti toiminnot valittiin siitä näkökulmasta, että niillä voitaisiin korvata mahdollisimman hyvin käsiohjaimen toiminnallisuudet, joita ovat muun muassa veturien nopeuden, suunnan ja toimintojen ohjaus sekä lisätarvikkeiden ja radan virran ohjaus. Jokainen valittu toiminnallisuus on tarpeellinen tämänhetkisessä pienoisrautatiessä. Mitään toimintoja ei siis lähdetty suunnittelemaan varalle, ja esimerkiksi valitut veturitoiminnot sopivat jo valmiiksi laajasti eri veturimalleille.

Toimintojen kartoittamisen jälkeen alettiin selvittää Märklinin saksankielisestä Kommunikationsprotokoll-dokumentista, mitä ratayksikön komentoja toiminnot vastaavat sekä mitä parametreja toiminnot vaativat. Kuten aiemmin on jo opittu, viestit koostuvat komennosta, pituuskoodista sekä datatavuista. Kun ne saatiin selville, tiedettiin tarkalleen rajapintaohjelmalle kussakin toiminnossa syötettävät tiedot. Seuraavaksi ryhdyttiin pohtimaan ohjelmakirjaston rakennetta ja toimintaa.

Jokainen toiminto haluttiin rakentaa omaksi ohjelmalohkoksi, jotta niitä voitaisiin helposti kutsua graafisessa ohjelmointikielessä. Toisaalta modulaarinen rakenne tulee helpottamaan myös päivitysten ja kehitystoimenpiteiden tekemistä jatkossa. Kuten rajapintaohjelma, myös ohjelmakirjasto päätettiin tehdä rajapintaohjelman tapaan strukturoitua tekstiä käyttäen, koska ohjelmalohkot on vaivattomampi tehdä tekstipohjaisella ohjelmointikielellä.

Rajapintaohjelman funktio FB_CanSend toteutettiin erityisesti ohjelmakirjastoa silmällä pitäen. Siihen tehtiin toiminto, joka vastaanottaa lähetettyyn komentoon saatavan vahvistuksen ratayksiköltä. Sen lisäksi siinä on valmiit parametrit, joilla voidaan raportoida onnistunut komennon lähetys. Suunniteltu ohjelmakirjaston ohjelmalohkojen toimintaperiaate on havainnollistettu kuviossa 12.



KUVIO 12. Ohjelmalohkon toiminta sitä kutsuttaessa. (Kuvio: Antti Seitsenlinna)

Suunnitelmassa tiettyä toimintoa vastaavaa ohjelmalohkoa kutsutaan käyttäjän PLC-ohjelmassa, minkä seurauksena ohjelmalohko lähettää toimintoa vastaavan komennon ja sen parametrit rajapintaohjelmalle. Rajapintaohjelma muodostaa niistä CAN-viestin ja lähettää sen väylälle. Ratayksikkö vastaanottaa komennon väylältä ja vahvistaa vastaanoton lähettämällä vahvistuksen. Rajapintaohjelma vastaanottaa vahvistusviestin ja välittää sen ohjelmalohkolle, joka taas palauttaa sitä kutsuneelle PLC-ohjelmalle kuittauksen onnistuneesta suorittamisesta. Jos suoritettu toiminto oli esimerkiksi tietopyyntö, välitetään kutsujalle myös pyydetty tieto kuten veturin nopeus tai kulkusuunta. Lisäksi ohjelmakirjastolle päätettiin tehdä globaaleita muuttujia käyttäen osoitekirja, jossa laitteiden osoitteet on koottu yhteen paikkaan. Niitä ei näin tarvitsisi muistaa tai tarkistaa erikseen.

5.2 Ohjelmalohkojen esittely

Ohjelmakirjastoon toteutettiin lopulta kymmenen toimintoa vetureille, kaksi lisätarvikkeille ja kaksi muille toiminnoille. Toiminnot on listattu taulukkoon 5, minkä lisäksi ne esitellään yksitellen seuraavissa alaluvuissa 5.2.1, 5.2.2 ja 5.2.3.

TAULUKKO 5. Ohjelmakirjaston ohjelmalohkot.

Toiminnon tyyppi	**Nimi**	**Kuvaus**
Veturien toiminnot	FB_SetLocoSpeed	Aseta veturin nopeusohje
	FB_AccelerateLoco	Kasvata veturin nopeutta
	FB_DecelerateLoco	Hidasta veturin nopeutta
	FB_GetLocoSpeed	Kysy veturin nopeutta
	FB_SetLocoDirection	Aseta veturin kulkusuunta
	FB_ToggleLocoDirection	Vaihda veturin kulkusuunta
	FB_GetLocoDirection	Kysy veturin kulkusuuntaa
	FB_SetLocoFunction	Aseta veturin toiminnon tila
	FB_GetLocoFunction	Kysy veturin toiminnon tilaa
	FB_ToggleLocoFunction	Vaihda veturin toiminnon tila
Lisätarvikkeiden toiminnot	FB_SetAccessory	Aseta lisätarvikkeen tila
	FB_SetTurnout	Aseta vaihteen asento
Muut toiminnot	FB_SetTrackPower	Kytke raiteiden virta
	FB_VersionPinger	Testaa yhteys ja kysy versio

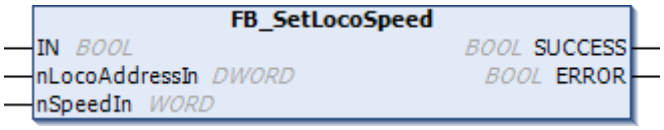
Veturitoiminnot sisältävät sisääntulomuuttujan nLocoAddressIn, jolla määritetään kohdeveturin osoite. Vastaavanlainen muuttuja on lisätarvikkeiden toiminnoissa. Ohjelmalohkoissa on IN-sisääntulomuuttuja. Kun se on ”tosi”, ohjelmalohko suoritetaan ja sen tiedot säilytetään, kunnes muuttuja on ”epätosi”. Ulostulomuuttujat vaihtelevat ohjelmalohkoittain sen mukaan, mikä on kunkin lohkon tehtävä.

Kaikkiin ohjelmalohkoihin luotiin ulostulomuuttujat SUCCESS ja ERROR, jotka kertovat suorittamisen onnistumisesta. Jos ohjelmalohko ei vastaanota ajoissa vahvistusta ratayksiköltä tai se vastaanottaa vääränlaisen vastauksen, asettaa se ERROR-muuttujan arvoksi ”tosi”. Vastaavasti onnistuneen suorituksen jälkeen ohjelmalohko palauttaa SUCCESS-muuttujan arvon ”tosi”.

5.2.1 Veturitoiminnot

FB_SetLocoSpeed

Veturin nopeuden asettamiseksi luotiin ohjelmalohko FB_SetLocoSpeed, jossa ratayksikölle lähetetään nopeuskomento. Komennossa nopeusparametrin koko on 10 bittiä, sillä nopeusohje voidaan antaa välillä 0…1000. Nopeusohje 0 vastaa pysähtynyttä veturia. Muut arvot skaalautuvat veturille erikseen konfiguroidun enimmäisnopeuden mukaan siten, että 1000 vastaa suurinta nopeutta. Kuvassa 17 on esitetty ohjelmalohkon graafinen muoto.

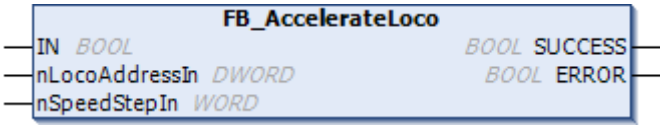


KUVA 17. Ohjelmalohko FB_SetLocoSpeed. (Kuva: Antti Seitsenlinna)

Ohjelmalohkon sisääntulomuuttujat ovat sen aktivoiva IN-totuusarvo, kohdeveturi nLocoAddressIn (dword) sekä nopeusohjeen sisältävä nSpeedIn (word), joka jaetaan ohjelmassa kahteen osaan siten, että nopeusohjeen viidenteen tavuun on sijoitettu siitä kahdeksan tavua ja kuudenteen datatavuun jäljelle jäävät kaksi bittiä. Ulostulomuuttujia ei ole SUCCESS- ja ERROR-totuusarvojen lisäksi.

FB_AccelerateLoco

Veturin nopeutta voidaan ohjata myös ohjelmalohkolla FB_AccelerateLoco. Se luotiin, jotta nopeutta voitaisiin kiihdyttää askelmaisesti. Ohjelmalohkossa ensin kysellään veturille asetettua nopeusohjetta ohjelmalohkolla FB_GetLocoSpeed. Sitä kasvatetaan määritetyn nopeusaskeleen verran, minkä jälkeen uusi arvo asetetaan veturille ohjelmalohkoa FB_SetLocoSpeed käyttäen. Enimmäisarvo nopeusohjeelle on kuitenkin 1000, eikä sitä ylitetä. Ohjelmalohkon graafinen muoto on esitetty kuvassa 18. Ohjelmalohkon sisääntulomuuttuja on aktivoinnin ja veturin osoitteen lisäksi nopeusohje nSpeedStepIn (word). Erityisiä ulostulomuuttujia ei tälläkään ohjelmalohkolla ole.



KUVA 18. Ohjelmalohko FB_AccelerateLoco. (Kuva: Antti Seitsenlinna)

FB_DecelerateLoco

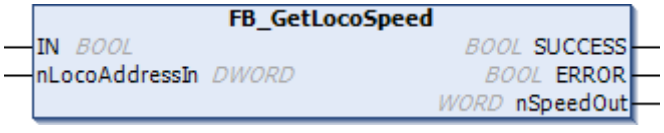
Veturin kiihdytystoiminnon pariksi luotiin ohjelmalohko FB_DecelerateLoco, jolla veturin nopeutta voidaan hidastaa askelmaisesti. Kiihdytystoiminnon tavoin ensin kysellään veturille asetettua nopeusohjetta ohjelmalohkolla FB_GetLocoSpeed, jonka jälkeen sitä vähennetään nopeusaskeleen verran. Sen jälkeen uusi arvo asetetaan veturille ohjelmalohkoa FB_SetLocoSpeed käyttäen. Nopeusohje ei voi kuitenkaan mennä nollan alapuolelle. Valmiin ohjelmalohkon graafinen muoto on esitetty kuvassa 19. Ohjelmalohkon sisään- ja ulostulomuuttujat vastaavat aiemman kiihdytysohjelmalohkon muuttujia.

KUVA 19. Ohjelmalohko FB_DecelerateLoco. (Kuva: Antti Seitsenlinna)

FB_GetLocoSpeed

Veturille asetettua nopeusohjetta kysellään ohjelmalohkolla FB_GetLocoSpeed. Se lähettää ratayksikölle nopeuskomennon ilman nopeusohjetta. Ratayksikkö vastaa kyselyyn palauttamalla sillä hetkellä veturille asetetun nopeusohjeen, jonka arvo on välillä 0…1000. Palautetun nopeusparametrin koko on myös 10-bittiä lähetettävän nopeuskomennon tavoin.

Ohjelmalohkoa käytettäessä on tärkeää huomata, että nopeusohje ei ole sama asia kuin veturin todellinen nopeus. Ratayksikkö ei tiedä veturin todellista nopeutta tai sitä, onko nopeusohje todella saavutettu. Siksi todellista nopeutta ei voida kysellä ohjelmakirjaston kautta, vaan tarvittaessa se on mitattava toisella tavalla. Valmis ohjelmalohko on esitetty kuvassa 20.

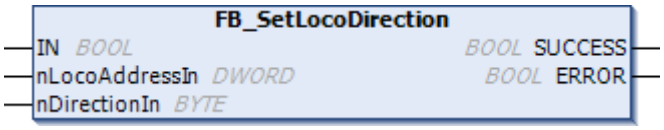


KUVA 20. Ohjelmalohko FB_GetLocoSpeed. (Kuva: Antti Seitsenlinna)

Ohjelmalohkossa ei ole erityisiä sisääntulomuuttujia aktivoinnin ja kohdeveturin lisäksi. Sen sijaan ulostulomuuttujana on SUCCESS- ja ERROR-totuusarvojen lisäksi ratayksikön palauttama nopeusohjeen arvo nSpeedOut (word).

FB_SetLocoDirection

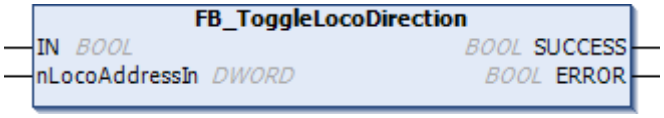
Veturin kulkusuunnan ohjaamiseksi luotiin ohjelmalohko FB_SetLocoDirection. Se lähettää ratayksikölle suuntakomennon, jonka viides datatavu määrittelee suuntaparametrin. Parametrin arvo voi olla 0, 1, 2 tai 3. Kun arvo on 0, säilytetään senhetkinen veturin kulkusuunta. Arvo 1 tarkoittaa kulkusuuntaa eteenpäin ja 2 taaksepäin. Parametrin arvolla 3 vaihdetaan veturin senhetkistä kulkusuuntaa. Ohjelmalohkon sisääntulomuuttujana on aktivoinnin ja veturin osoitteen lisäksi suuntaparametri. Erityisiä ulostulomuuttujia ei ole. Kuvassa 21 on esitetty valmis ohjelmalohko.



KUVA 21. Ohjelmalohko FB_SetLocoDirection. (Kuva: Antti Seitsenlinna)

FB_ToggleLocoDirection

Veturin kulkusuunnan vaihtamiseksi luotiin myös FB_ToggleLocoDirection, jossa ratayksikölle lähetettävän suuntakomennon parametri asetettu ennalta arvoon 3. Ohjelmalohko vastaa muuten edellistä ohjelmalohkoa FB_SetLocoDirection, mutta tarjoaa selkeämmän toiminnan pelkästään veturin suunnan vaihtamiseksi. Graafinen esitysmuoto ohjelmalohkosta on kuvassa 22. Ohjelmalohkolla ei ole varsinaisia sisääntulomuuttujia aktivoinnin ja kohdeveturin osoitteen lisäksi, eikä myöskään erityisiä ulostulomuuttujia. Suuntaparametri asetettiin jo valmiiksi, eli sitä ei tarvita.

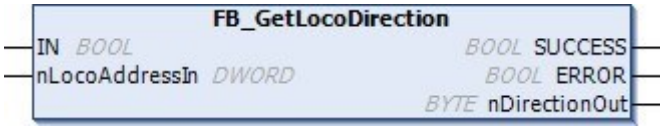


KUVA 22. Ohjelmalohko FB_ToggleLocoDirection. (Kuva: Antti Seitsenlinna)

FB_GetLocoDirection

Veturin kulkusuuntaa voidaan kysellä ohjelmalohkolla FB_GetLocoDirection. Siinä ratayksikölle lähetetään suuntakomento ilman parametria, ja ratayksikkö vastaa palauttamalla viestin senhetkisen kulkusuunnan suuntaparametrina. Sen arvo on eteenpäin kulkevalla veturilla 1 ja taaksepäin kulkevalla veturilla 2, kuten suuntakomentoa lähetettäessäkin. Valmis ohjelmalohko on esitetty kuvassa 23.

Ohjelmalohkossa ei ole erityisiä tulomuuttujia. Ratayksikön palauttaman suunta-parametrin arvo palautetaan ulostulomuuttujana nDirectionOut (word).



KUVA 23. Ohjelmalohko FB_GetLocoDirection. (Kuva: Antti Seitsenlinna)

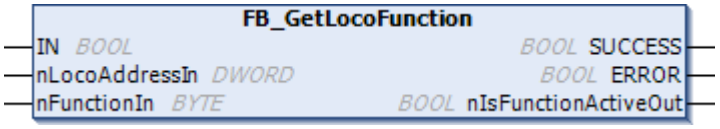
FB_SetLocoFunction

Veturin valojen, äänimerkkien ja muiden toimintojen ohjaukseen luotiin ohjelmalohko FB_SetLocoFunction, jonka ratayksikölle lähettämässä komennossa on toiminnon numero viidennessä ja sen arvo kuudennessa datatavussa. Toiminnon numero voi olla 0…31 ja arvoparametri välillä 0…31. Toiminnoilla, joilla on vain kaksi arvoa, arvo 0 kytkee toiminnon pois päältä ja arvo 1 kytkee sen päälle. Muut arvot säätävät toiminnon intensiteettiä, kuten valojen kirkkautta. Ohjelmalohkon graafinen muoto on esitetty kuvassa 24. Ohjelmalohkolle luotiin erityisiksi sisääntulomuuttujiksi toiminnon numerolle nFunctionIn (byte) ja sen arvoparametrille nFunctionValueIn (byte). Erityisiä ulostulomuuttujia ei ole.

KUVA 24. Ohjelmalohko FB_SetLocoFunction (Kuva: Antti Seitsenlinna)

FB_GetLocoFunction

Veturin toimintojen tilaa voidaan kysellä ohjelmalohkolla FB_GetLocoFunction. Se lähettää ratayksikölle samanlaisen komennon kuin toimintoa ohjatessa, mutta ilman toiminnon arvoparametria. Ratayksikkö vastaa komentoon palauttamalla totuusarvon 0 tai 1 eli tiedon, onko toiminto aktiivinen. Toiminnon intensiteetistä ei ole mahdollista saada tietoa, koska ratayksikössä ei säilötä siitä informaatiota. Kuvassa 25 on esitetty ohjelmalohkon graafinen esitysmuoto.

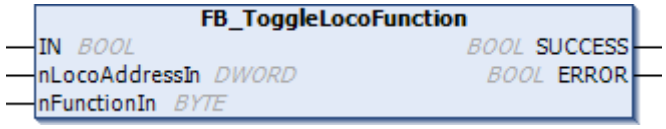


KUVA 25. Ohjelmalohko FB_GetLocoFunction. (Kuva: Antti Seitsenlinna)

Myöskään tässä ohjelmalohkossa ei ollut tarvetta erityisille sisääntulomuuttujille lukuun ottamatta muuttujaa nFunctionIn (byte), jossa määritetään kyselyn kohde eli toiminnon numero. Ulostulomuuttujaksi luotiin ratayksikön palauttamalle arvo-parametrille totuusarvo nIsFunctionActiveOut.

FB_ToggleLocoFunction

Veturin toimintojen tilaa voidaan vaihtaa ohjelmalohkoa FB_ToggleLocoFunction käyttäen. Ohjelmalohkon toiminnallisuus toteutettiin käyttämällä ohjelmalohkoja FB_GetLocoFunction ja FB_SetLocoFunction, koska erillistä komentoa ei ole. Ohjelmalohkon graafinen esitysmuoto on kuvassa 26.



KUVA 26. Ohjelmalohko ToggleLocoFunction. (Kuva: Antti Seitsenlinna)

Ohjelmalohkossa kysellään ensin veturin toiminnon tila. Tila käännetään päinvastaiseksi ja uusi komento lähetetään sen jälkeen ratayksikölle. Vain toiminnon numero tarvitaan sisääntulomuuttujaksi, ja erityisiä ulostulomuuttujia ei ole.

5.2.2 Lisätarvikkeiden toiminnot

FB_SetAccessory

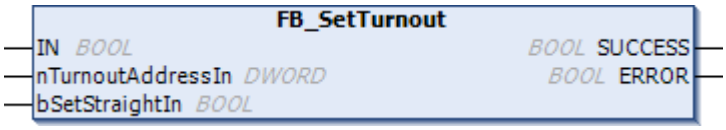
Lisätarvikkeiden tilan ohjaamiseen luotiin ohjelmalohko FB_SetAccessory, jonka ratayksikölle lähettämässä komennossa määritellään viidennessä datatavussa asetettava tila tai asetus ja kuudennessa datatavussa asetuksen intensiteetti. Asetus voidaan valita välillä 0…31 ja arvoparametri välillä 0…31. Toiminnoilla, joilla on vain kaksi arvoa, arvo 0 kytkee toiminnon pois päältä ja arvo 1 päälle. Kuvassa 27 esitetään valmiin ohjelmalohkon graafinen esitysmuoto.

KUVA 27. Ohjelmalohko FB_SetAccessory. (Kuva: Antti Seitsenlinna)

Ohjelmalohkon sisääntulomuuttujat ovat sen aktivoiva IN-totuusarvo, kohdelaite nAccessoryAddressIn (dword), asetettava tila nAccessorySettingIn (byte) sekä arvoparametri nAccessoryIntensityIn (byte). Ulostulomuuttujia ei ole lukuun ottamatta SUCCESS- ja ERROR-totuusarvoja.

FB_SetTurnout

Vaihteiden ohjaamiseen luotiin oma ohjelmalohkonsa FB_SetTurnout. Se vastaa ohjelmalohkoa FB_SetAccessory sillä erotuksella, että vaihteiden asennolle on luotu valmis parametri. Se asetetaan arvoon ”tosi”, kun vaihde halutaan kääntää suoralle vaihteelle ja arvoon ”epätosi”, kun vaihde halutaan poikkeavalle raiteelle. Valmis ohjelmalohko on esitetty graafisessa esitysmuodossa kuvassa 28. Ainut sisääntulomuuttuja on vaihteen asennon parametri bSetStraightIn (bool).

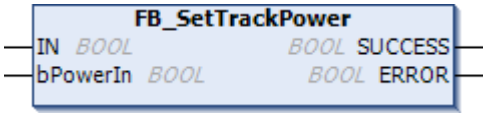


KUVA 28. Ohjelmalohko FB_SetTurnout. (Kuva: Antti Seitsenlinna)

5.2.3 Muut toiminnot

FB_SetTrackPower

Raiteiden virtaa ohjataan ohjelmalohkolla FB_SetTrackPower. Sen ratayksikölle lähettämä komento vaihtelee sen mukaan, kytketäänkö virrat päälle vai pois päältä. Kun valintaparametri on ”epätosi” eli virrat kytketään pois päältä, lähettää ohjelmalohko vain virransammutuskomennon. Jos virrat taas kytketään päälle eli valintaparametri on ”tosi”, määritetään ensin käytettävät rataprotokollat ja vasta sitten kytketään virta. Tähän toimintamalliin otettiin mallia Märklinin käsiohjaimen vastaavasta toiminnosta. Kuvassa 29 esitetään ohjelmalohkon graafinen muoto. Sisääntulomuuttujana on IN-totuusarvo sekä radan virran asetusparametrille bPowerIn-totuusarvo. Erityisiä ulostulomuuttujia ei ole.



KUVA 29. Ohjelmalohko FB_SetTrackPower. (Kuva: Antti Seitsenlinna)

FB_VersionPinger

Ohjelmalohkolla FB_VersionPinger voidaan kysellä ratayksikön käyttöliittymän versiota. Lisäksi sitä voidaan käyttää selvittämään ratayksikön vasteaika: samalla kun ohjelmalohko lähettää ratayksikölle versiokomennon, se katsoo järjestelmän ajan. Vastauksen saapuessa aika tarkastetaan uudestaan. Aikojen erotuksesta voidaan päätellä, kuinka kauan vastauksen saamisessa kesti. Valmis ohjelmalohko on esitetty graafisessa muodossa kuvassa 30.

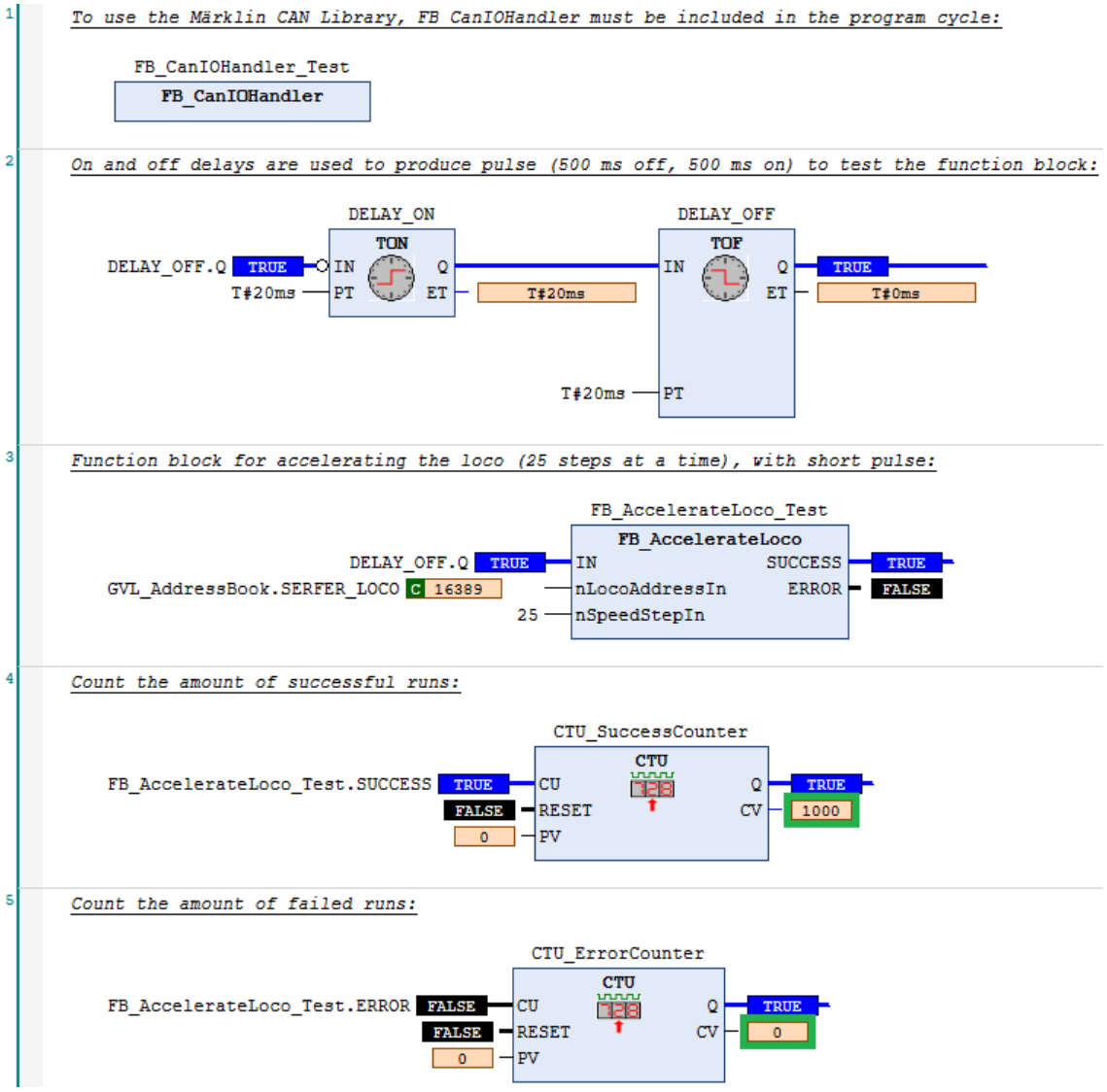
KUVA 30. Ohjelmalohko FB_VersionPinger. (Kuva: Antti Seitsenlinna)

Ohjelmakirjastossa ei ole sisääntulomuuttujia IN-totuusarvon lisäksi. Ulostulomuuttujina on vasteaika millisekunteina muuttujassa nPingMsOut (unsigned int) sekä ratayksikön palauttama käyttöliittymän versio nVersionOut (word). Koska ohjelmalohkon käyttämä versiokysely lähetetään kaikille Märklin-laitteille, siihen vastaa myös esimerkiksi väylään mahdollisesti liitetty Märklinin käsiohjain. Sen takia rajapintaohjelmaan rakennettiin erityinen suodatin, joka suodattaa tämän komennon nähdessään muut kuin ratayksikön vastaukset pois. Näin väylän muut laitteet eivät sekoita versiokyselyä ja vasteajan mittausta.

5.3 Ohjelmalohkojen testaus

Kun ohjelmakirjaston ohjelmalohkot saatiin valmiiksi, oli tärkeää testata niiden toimivuus. Testitoiminnallisuus rakennettiin graafiseen toimilohko-ohjelmaan. Testiohjelman alkuun sijoitettiin funktio FB_CanIOHandler_Test, joka suorittaa ohjelmakirjaston toiminnan kannalta välttämättömät rajapintatoiminnallisuudet. Sen jälkeen toteutettiin veto- ja päästöhidastusfunktioita käyttäen pulssitoiminto, jonka tuottama pulssi suorittaa testattavan ohjelmalohkon tietyin väliajoin. Sekä onnistuneiden että epäonnistuneiden suorituskertojen laskemista varten luotiin laskurit, joihin syötetään ohjelmalohkon SUCCESS- ja ERROR-ulostulot. Samaa testiohjelmaa käytettiin kaikkien ohjelmalohkojen testaamiseksi. Testiohjelmassa testataan ohjelmalohkoa FB_AccelerateLoco.

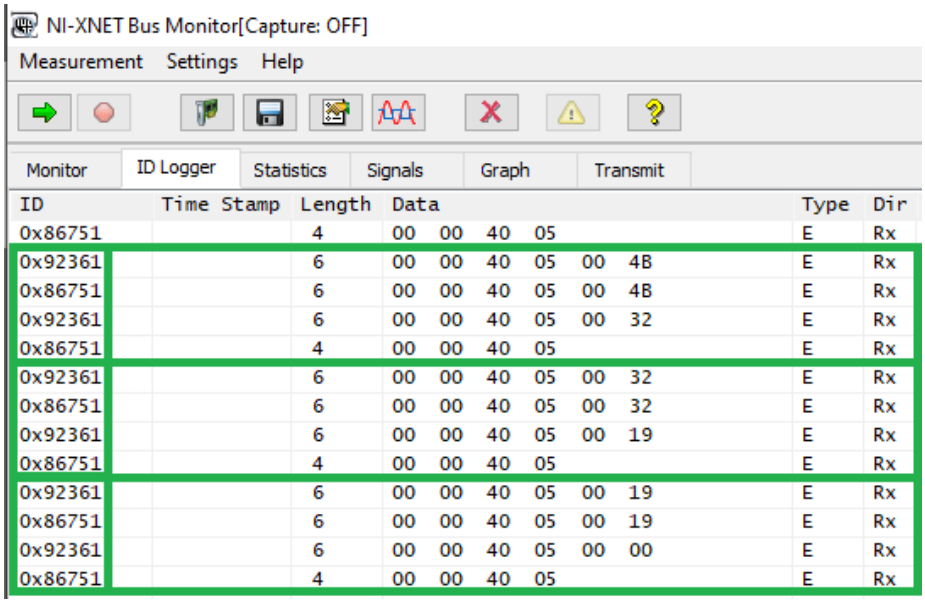
Aluksi käytettiin yhden sekunnin pulssia, jossa signaali on 500 ms päällä ja 500 ms pois päältä. Testit suoritettiin kuitenkin myös 20 ms välein toistuvalla pulssilla. Se valittiin, koska ohjelmalohkojen odotusaika on myös 20 ms, jonka verran ne odottavat vastausta. Käytännössä onnistunut suorituskerta kestää joitakin millisekunteja, mutta odotusaikaan on päätetty jättää hieman varaa. Ohjelmalohkoa testattiin myös lyhyellä yhden ohjelmakierron pituisella pulssilla lisäämällä sen eteen nousevan reunan tunnistus. Nousevalla reunalla tarkoitetaan signaalin muuttumista aktiiviseksi. Käynnissä oleva testi on esitetty kuvassa 31.



KUVA 31. Käynnissä oleva ohjelmalohkon testaus. (Kuva: Antti Seitsenlinna)

Kuvasta voidaan huomata, että onnistuneita suorituskertoja laskevan laskurin CTU_SuccessCounter arvo on 1000 samalla kun epäonnistuneita suorituskertoja ei ole yhtään. Ohjelmalohko siis ainakin väittää lähettäneensä tuhat komentoa epäonnistumatta kertaakaan. Tästä ei kuitenkaan voitu olla täysin varmoja, joten

asia varmistettiin vielä aiemmin tutuksi tulleella National Instrumentsin analyysityökalulla. Kuvassa 32 on esitetty analyysityökalu testitilanteen aikana sekä siinä näkyvät viestit.



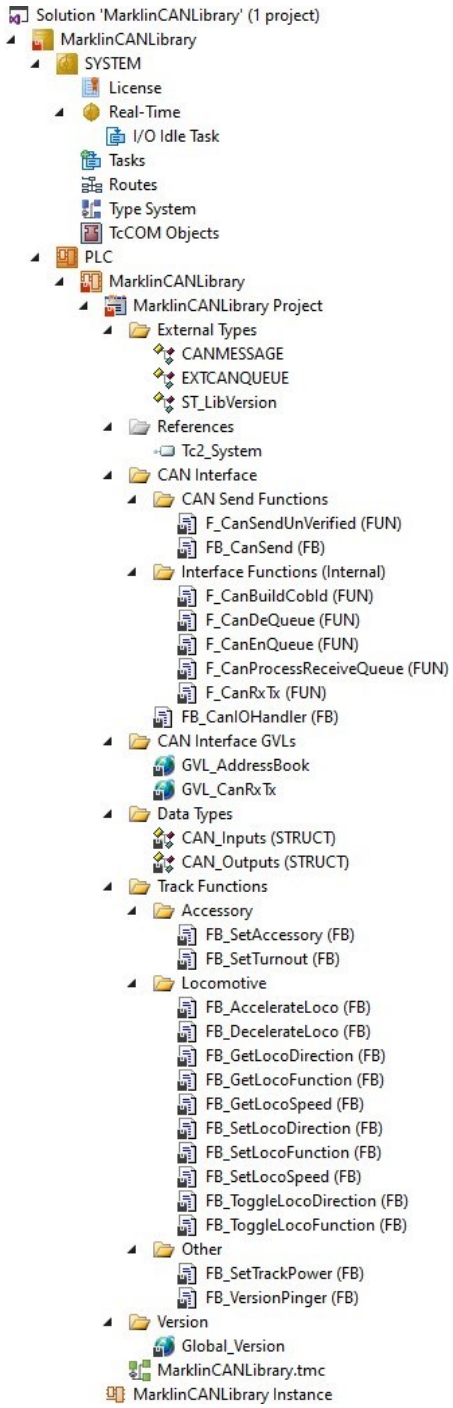
KUVA 32. Analyysityökalun näkymä testitilanteessa. (Kuva: Antti Seitsenlinna)

Analyysityökalulla voitiin varmistaa ohjelmakirjaston toimivan oikein. Moduuli (tunniste 0x86751) lähettää ensin nopeuskomennon ilman nopeusohjetta, johon ratayksikkö (tunniste 0x92361) vastaa veturin nykyisellä nopeusohjeella. Veturi on pysähtynyt, joten nopeusohje on 0. Ohjelmalohko lisää tähän testiohjelmassa määritetyn askeleen 25 (heksadesimaalimuodossa 19) ja lähettää sen moduulin kautta ratayksikölle. Ratayksikkö vahvistaa komennon suorittamisen lähettämällä kopion viestistä. Sama toistuu nopeusaskeleen kasvaessa. Kuvassa on esitetty nopeusaskeleen arvot 50 ja 75, mutta ne jatkoivat kasvuaan aina tuhanteen asti.

Ohjelmalohkon toimivuus varmistettiin myös seuraamalla veturia. Kun suoritettiin testiohjelmaa, veturin nopeus kasvoi tasaisesti odotetun mukaisesti. Vikatilannetestinä CAN-väylän johdinvikaa simuloitiin irrottamalla väylän toinen datajohdin. Onnistuneiden suorituskertojen laskuri kasvoi aluksi, kunnes se pysähtyi datajohtimen irrottua, minkä jälkeen epäonnistuneiden suorituskertojen laskuri alkoi heti kasvaa. Kun datajohdin kytkettiin takaisin, onnistuneiden suorituskertojen laskuri alkoi taas kasvaa. Testi onnistuikin hyvin, sillä ohjelmalohko paitsi tunnisti virhetilanteen, myös palautui itsenäisesti vian korjaannuttua. Kaikille ohjelmalohkoille suoritettiin samat testit, jotta voitiin varmistua, ettei niissä ole ohjelmointivirheitä.

5.4 Kirjaston viimeistely

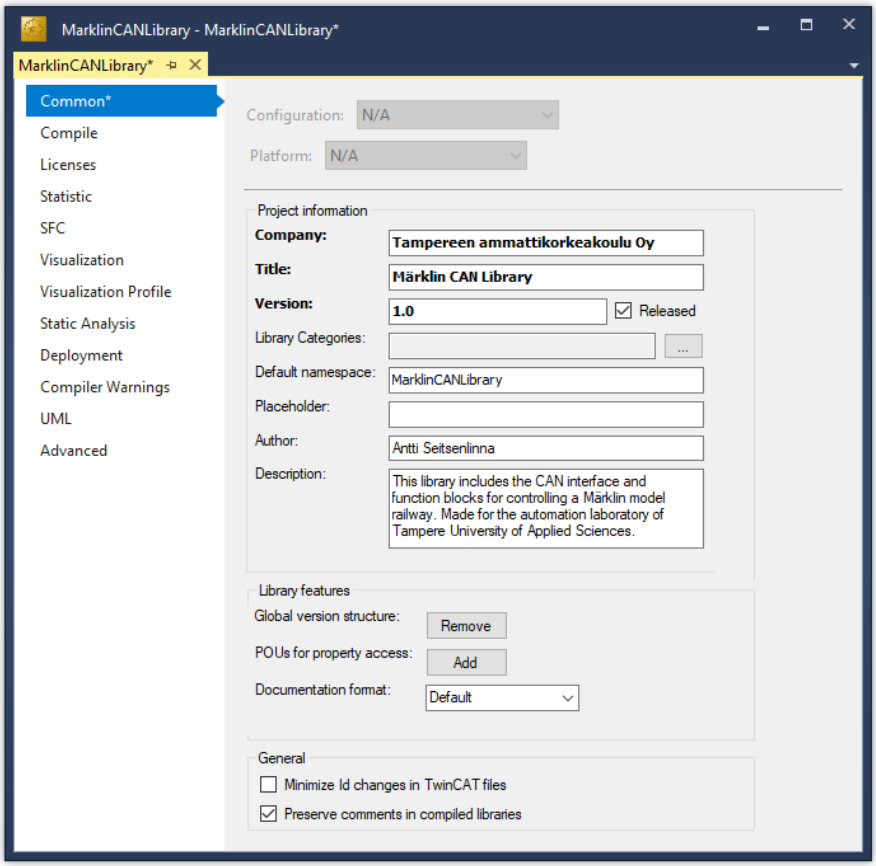
Ohjelmakirjaston valmistuttua sen projektipuusta siivottiin kaikki turhat kohteet pois. Lisäksi poistettiin kaikki kehitysvaiheen testiohjelmat ja tehtävät. Tarkoitus oli tehdä ohjelmakirjaston projektista mahdollisimman karsittu, koska sen ei ole tarkoitus toimia itsenäisenä ohjelmana. Kuvassa 33 on esitetty lopullinen näkymä karsitun ohjelmakirjaston projektipuusta. Siihen ei jäänyt enää rajapintaohjelman ja ohjelmakirjaston lisäksi montaakaan ylimääräistä asiaa.



KUVA 33. Valmis ohjelmakirjasto projektipuussa. (Kuva: Antti Seitsenlinna)

Siistimisen jälkeen käytiin vielä läpi kaikki paitsi ohjelmakirjaston, myös aiemmin pohjalle toteutetun rajapintaohjelman kommentoinnit. Kaikkien ohjelmien alkuun lisättiin lyhyt kuvaus niiden toiminnasta ja kuvaukset sisään- ja ulostulomuuttujille sekä globaaleille muuttujille. Nämä kuvaukset näkyvät kirjastoa käytettäessä ja helpottavat siten käyttöä. Kommentoinneissa noudatettiin koko projektin laajuista yhtenäistä muotoilutyyliä.

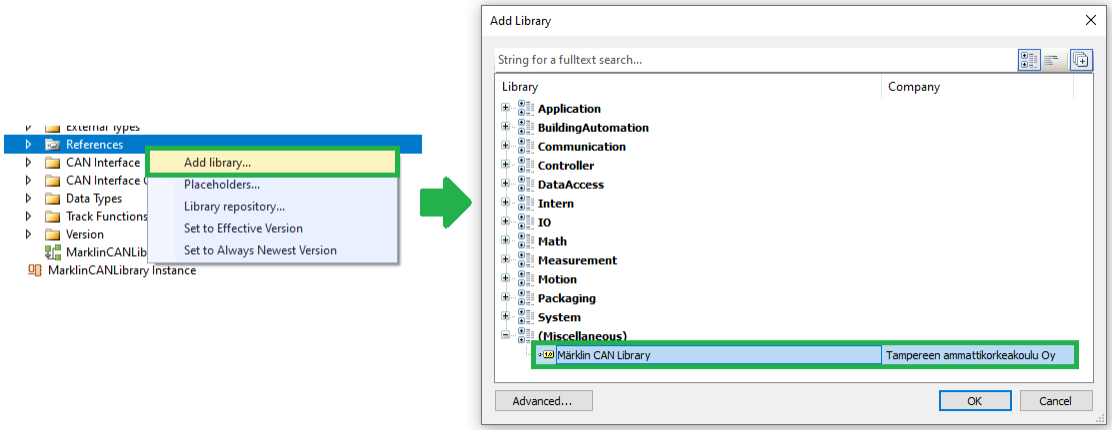
Kun ohjelmakirjasto oli viimeistelty, piti sille täyttää vielä projektin tiedot. Se tehtiin napauttamalla PLC-kohdassa projektin nimeä ”MarklinCANLibrary Project” hiiren oikealla ja valitsemalla ”Properties”. Avautuvassa ikkunassa voitiin täyttää tietoja projektille. Kenttiin syötettiin muun muassa projektin otsikko, versio sekä kuvaus. Syötetyt tiedot on esitetty kuvassa 34.



KUVA 34. Projektille täytetyt tiedot. (Kuva: Antti Seitsenlinna)

Tietoikkunassa napautettiin myös kohdan ”Global version structure” vierestä ”Add”, jolla lisättiin versio globaaliksi muuttujaksi projektiin. Sillä mahdollistettiin esimerkiksi projektin version tarkastaminen ohjelmallisesti. Tietojen lisäämisen jälkeen projekti oli valmis kirjastoksi tallentamista varten. Tallentaminen tehtiin

napauttamalla taas projektin nimeä hiiren oikealla ja valitsemalla ”Save as library and install …”. Valinnan jälkeen avautui ikkuna, jossa kysyttiin tallennettavan kirjaston nimeä ja tallennuspaikkaa. Tallennustoiminto asensi kirjaston samalla myös kehitysympäristöön tulevien projektien käyttöön. Tämä varmistettiin vielä napauttamalla References-kohtaa hiiren oikealla ja valitsemalla ”Add library…”, jolloin ohjelmakirjasto näkyi kirjastolistauksessa kuvan 35 osoittamalla tavalla. Ohjelmakirjasto otettiin myös talteen tallennussijainnistaan sen jakamista varten.



KUVA 35. Ohjelmakirjasto näkyy kirjastolistauksessa. (Kuva: Antti Seitsenlinna)

Lopputuloksena syntynyt ohjelmakirjasto täyttää kaikki sille asetetut vaatimukset ja todettiin monilla eri testeillä toimivaksi. Kokonaisuus on hyvin kommentoitu ja sisältää kaikki keskeiset toiminnot eri Märklin-pohjaisten laitteiden ohjaamiseksi. Valmista ohjelmakirjastoa on helppo hyödyntää PLC-ohjelmoinnissa valmiiden ohjelmalohkojen ansiosta, kun pienoisrautatielle kehitetään sovelluksia jatkossa.

6 DOKUMENTAATIO

6.1 Suunnittelu ja tavoitteet

Ohjelmakirjastoa tullaan käyttämään tulevaisuudessa pienoisrautatieympäristön ohjausjärjestelmän ja muiden sovellusten perustana. Onkin tärkeää, että kirjaston hyödyntäminen on mahdollisimman helppoa, minkä takia sille oli tarpeen laatia kattava dokumentaatio. Hyvä dokumentaatio paitsi helpottaa ohjelmakirjaston käyttöä, myös tulee edistämään sen pitkäikäisyyttä ja mahdollista jatkokehitystä.

Dokumentaation suunnittelu aloitettiin pohtimalla, mitkä ohjelmakirjaston käytön vaiheet ovat erityisen haastavia ja vaativat opastusta. Lisäksi pyrittiin arvioimaan keskimääräistä ohjelmakirjaston käyttäjän tietotasoa ja kartoittamaan asioita, joista olisi tarpeellista tarjota lisätietoa. Dokumentaatiolle asetettiin pohdinnan ja opinnäytetyön aikana kertyneen käyttökokemuksen perusteella lopulta seuraavat tavoitteet, jotka pyrittiin täyttämään dokumentaatiota laadittaessa:

- Opastaa EL6751-moduulin konfiguroinnissa.
- Opastaa ohjelmakirjaston saattamisessa käyttökuntoon moduulin kanssa.
- Perustella kunkin ohjelmalohkon tarkoitus pienoisrautatien ohjauksessa.
- Perustella ohjelmakirjaston ohjelmoinnissa tehtyjä ratkaisuja.
- Kertoa ohjelmakirjaston käyttöön liittyvistä tärkeistä yksityiskohdista.
- Sisältää versionumeron, jotta uudet versiot voidaan erottaa vanhoista.

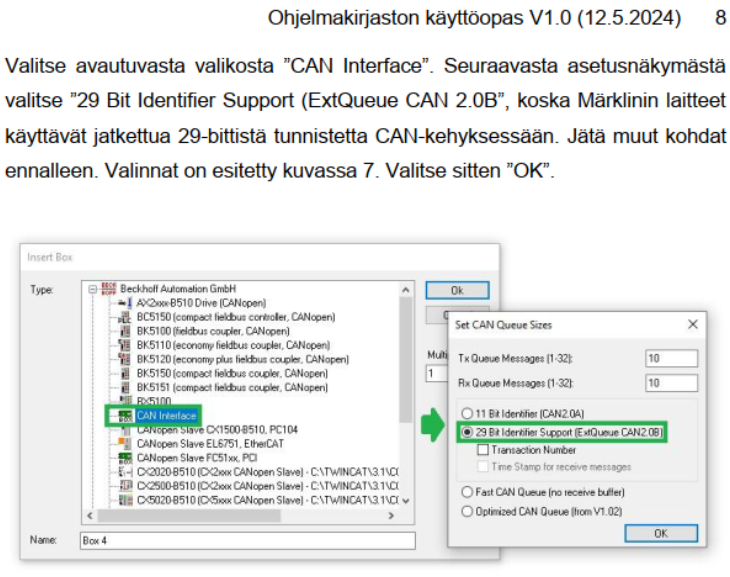
Tavoitteissa oletetaan todennäköisellä ohjelmakirjaston käyttäjällä olevan edes kohtalaisesti aikaisempaa kokemusta TwinCAT 3 -kehitysympäristön käytöstä. Yksi haastavimmista asioista ohjelmakirjaston käytössä tulee olemaan kirjaston saattaminen käyttökuntoon, minkä takia siihen päätettiin kiinnittää huomiota. Koska ohjelmakirjaston käyttö PLC-ohjelmoinnissa on tehty intuitiiviseksi sekä kommentoitu hyvin ohjelmakoodissa, se ei vaadi paljon opastusta. On kuitenkin tärkeää tiedostaa, että ohjelmakirjaston laatijalle sen käyttökin on helpompaa. Olennaista on auttaa dokumentaation lukijaa ymmärtämään, miksi ratkaisuihin on päädytty ja mitä tarkoitusta kukin ohjelmalohko palvelee. Joka ikistä pientä yksityiskohtaa ei tarvitse selittää, jos ne on avattu jo kommentoinnissa.

6.2 Laatiminen

Dokumentaatiota lähdettiin toteuttamaan TAMKin dokumenttipohjalle, koska se soveltui hyvin käyttötarkoitukseen. Otsikoksi dokumentaatiolle annettiin sitä hyvin kuvaava ”Ohjelmakirjaston käyttöopas”. Kansilehdelle kirjoitettiin otsikon lisäksi ohjelmakirjaston kuvaus alaotsikoksi sekä versionumero ja päivitysajankohta. Tämä on dokumentaation ensimmäinen versio, joten versionumeroksi annettiin luonnollisesti 1.0.

Käytettävässä dokumenttipohjassa oli valmiina sisällysluettelo, jonka katsottiin olevan tarpeellinen. Ensimmäisessä tekstiluvussa kerrottiin ohjelmakirjastosta ja sen ohjelmalohkojen toiminnasta lyhyesti. Lisäksi käytiin johdannon tapaisesti läpi dokumentaatiossa käsiteltävät asiat ja esiteltiin moduuli, jolla ohjelmakirjasto toteutettiin. Luvun lopussa esiteltiin vielä, miten ohjelmakirjastoon voi tarvittaessa tehdä muutoksia.

Toisessa luvussa opastettiin moduulin konfigurointi sekä ohjelmakirjaston lisäys projektiin. Kaikki olennaiset asetukset ja muut yksityiskohdat käytiin myös läpi ohjelmakirjaston oikean toiminnan varmistamiseksi. Kaikki vaiheet esitettiin siinä järjestyksessä, jossa ne kannattaa tehdä tositilanteessa. Lisäksi luvussa esiteltiin moduulin ja ohjelmakirjaston sisään- ja ulostulomuuttujien linkitys toisiinsa. Ote dokumentaation luvusta 2 on esitetty kuvassa 36.



Ohjelmakirjaston käyttöopas V1.0 (12.5.2024) 8

Valitse avautuvasta valikosta ”CAN Interface”. Seuraavasta asetusnäkymästä valitse ”29 Bit Identifier Support (ExtQueue CAN 2.0B”, koska Märklinin laitteet käyttävät jatkettua 29-bittistä tunnistetta CAN-kehyksessään. Jätä muut kohdat ennalleen. Valinnat on esitetty kuvassa 7. Valitse sitten ”OK”.

KUVA 7. CAN-rajapinnan lisääminen.

KUVA 36. Ote dokumentaation luvusta 2. (Kuva: Antti Seitsenlinna)

Seuraavassa luvussa esiteltiin ohjelmakirjaston ohjelmalohkot ja kerrottiin, mitä ne ylipäätään ovat. Lisäksi kerrottiin ohjelmalohkosta FB_CanIOHandler, joka on oleellinen rajapintaohjelman toiminnan kannalta. Jokainen ohjelmalohko käytiin läpi ja esiteltiin muuttujat sekä ohjelmalohkon tarkoitus. Luvun lopussa kerrottiin, mitä lähetysfunktiot FB_CanSend ja FB_CanSendUnVerified ovat, ja miten niillä voidaan luoda uusia ohjelmalohkoja tai lähettää muokattuja Märklin-komentoja. Lisäksi kerrottiin rajapintaohjelman sisäisistä funktioista, joihin ei ohjelmakirjaston käytössä pitäisi sekaantua.

Viimeisessä luvussa esitettiin yksinkertainen käyttöesimerkki, jossa asetetaan veturin ajovalot päälle ja nopeusohje arvoon 500. Käyttöesimerkin tarkoituksena on havainnollistaa, miten ohjelmalohkoa FB_CanIOHandler ja toiminnallisuuksia voidaan hyödyntää PLC-ohjelmoinnissa. Lisäksi havainnollistettiin osoitekirjan toimintaa sekä esitettiin ehdotus ERROR-ulostulomuuttujan toiminnallisuuksille.

Dokumentaatiosta saatiin lopulta tehtyä tarpeeksi kattava ja se täyttää hyvin sille asetetut tavoitteet, eli se helpottaa ohjelmakirjaston käyttöä ja selventää osien tarkoitusta. Dokumentaatiossa saatiin myös tuotua esille tärkeitä yksityiskohtia, kuten sykliajan muuttaminen. Sen visuaalinen ilme on dokumenttipohjan ansiosta yhtenäinen ja siisti. Valmis dokumentaatio on kokonaisuudessaan työn liitteissä.

7 POHDINTA

Tämän opinnäytetyön tarkoituksena oli luoda CAN-rajapinta ja ohjelmakirjasto Tampereen ammattikorkeakoulun kehitteillä olevaan pienoisrautatieympäristöön. Pienoisrautatien ohjaus tuotiin käsiohjaimelta TwinCAT 3 -kehitysympäristöön ja muodostettiin pohja ohjausjärjestelmän ja muiden ominaisuuksien kehittämiselle jatkossa. Tuloksena syntynyt ohjelmakirjasto hyödyntää tehokkaasti toteutettua rajapintaohjelmaa ja kykenee korvaamaan keskeiset käsiohjaimen toiminnot Märklin-pohjaisten veturien ja lisälaitteiden ohjaamiseksi. Jotta ohjelmakirjaston hyödyntäminen olisi mahdollisimman vaivatonta, sille laadittiin kattava liitteistä kokonaisuudessaan löytyvä dokumentaatio. Se sisältää kaikki tarpeelliset tiedot ohjelmakirjaston käytöstä ja yksityiskohdista, kuten tärkeistä asetuksista.

Opinnäytetyön ensimmäisissä luvuissa esiteltiin työn toteutuspaikkana toimivaa pienoisrautatieympäristöä sekä siinä käytettävää laitteistoa ja kehitysympäristöä. Lisäksi perehdyttiin CAN-väylän historiaan ja toimintaan sekä käyttöön Märklin Digitalissa. Teoriapohjan jälkeen tutustuttiin rajapintaohjelman rakenteeseen ja sen toteutusprosessiin. Kun rajapinta oli saatu muodostettua, ohjelmakirjastoa ryhdyttiin suunnittelemaan kartoittamalla tarvittavat ohjelmalohkot ja pohtimalla kirjaston toimintaa. Valmiit ohjelmalohkot esiteltiin ja testattiin. Lopuksi käytiin läpi dokumentaation laatimisprosessia ja asetettuja vaatimuksia.

Vaikka ohjelmakirjasto on kattava ja edistää huomattavasti oppimisympäristön kehitystä, siitä päätettiin kuitenkin jättää joitakin konfigurointi- ja erityistoimintoja pois niiden vähäisen tarpeen takia. Jatkokehityskohteena saattaakin olla näiden ominaisuuksien lisääminen myöhemmin, mikäli ne koetaan tarpeellisiksi. Työtä tullaan hyödyntämään jo pian lähitulevaisuudessa pienoisrautatien sovellusten kehittämisessä. Opinnäytetyö valmistuikin juuri ajoissa sitä varten, että syksyksi 2024 on mahdollista tehdä ohjausjärjestelmä pienoisrautatielle rajapintaohjelman ja ohjelmakirjaston pohjalta.

LÄHTEET

ADS device consept. n.d. Beckhoff Automation GmbH & Co. KG. Verkkosivu. Viitattu 11.5.2024. https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/11291871243.html&id=

CAN high-speed transmission. n.d. CAN in Automation. Verkkosivu. Viitattu 18.4.2024. https://www.can-cia.org/de/can-knowledge/can/high-speed-transmission/

CiA and international standardization. n.d. CAN in Automation. Verkkosivu. Viitattu 17.4.2024. https://www.can-cia.org/groups/international-standardization/

Classical Controller Area Network (CAN). n.d. CAN in Automation. Verkkosivu. Viitattu 17.4.2024. https://www.can-cia.org/can-knowledge/can/classical-can/

Company. n.d. Beckhoff Automation GmbH & Co. KG. Verkkosivu. Viitattu 9.5.2024. https://www.beckhoff.com/en-en/company/

CX51x0 Manual, V2.8. 8.1.2024. Beckhoff Automation GmbH & Co. KG. Pdf-dokumentti. Viitattu 26.4.2024. https://download.beckhoff.com/download/Document/ipc/embedded-pc/embedded-pc-cx/cx5100_en.pdf

EL6751 Manual, V3.9. 9.1.2024. Beckhoff Automation GmbH & Co. KG. Pdf-dokumentti. Viitattu 26.4.2024. https://download.beckhoff.com/download/Document/io/ethercat-terminals/el6751en.pdf

History of CAN technology. n.d. CAN in Automation. Verkkosivu. Viitattu 17.4.2024. https://www.can-cia.org/can-knowledge/can/can-history/

Kommunikationsprotokoll, V2.0. 7.2.2012. Gebr. Märklin & Cie GmbH. Pdf-dokumentti. Viitattu 23.4.2024. https://streaming.maerklin.de/public-media/cs2/cs2CAN-Protokoll-2_0.pdf

Mark Banks. 2001. Interworld Electronics & Computer Industries Inc. Verkkosivu. Viitattu 22.4.2024. https://www.interworldna.com/pico/automotive/audi-can-bus.php

Märklin Digital antaa mahdollisuuksia. n.d. Märklin Club of Finland. Verkkosivu. Viitattu 26.4.2024. http://www.marklinclub.fi/digital.htm

Philosophy. n.d. Beckhoff Automation GmbH & Co. KG. Verkkosivu. Viitattu 11.5.2024. https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/index.html&id=2

Product description. n.d. Märklin & Cie GmbH. Verkkosivu. Viitattu 26.4.2024. https://www.maerklin.de/en/products/details/article/55324

SFS-EN IEC 61131-3:2013. 2013. Programmable controllers. Part 3: Programming languages. Helsinki: Suomen Standardisoimisliitto SFS. Viitattu 27.4.2024. Vaatii käyttöoikeuden. https://online.sfs.fi/fi/index/tuotteet/SFSsahko/CENELEC/ID2/6/285253.html.stx

The CAN Bus Protocol Tutorial. n.d. Kvaser. Verkkosivu. Viitattu 17.4.2024. https://www.kvaser.com/can-protocol-tutorial/

TwinCAT automation software. n.d. Beckhoff Automation GmbH & Co. KG. Verkkosivu. Viitattu 27.4.2024. https://www.beckhoff.com/en-en/products/automation/twincat/

Über Märklin. n.d. Gebr. Märklin & Cie GmbH. Verkkosivu. Viitattu 26.4.2024. https://www.maerklin.de/de/unternehmen/ueber-maerklin

LIITTEET

Liite 1. Ohjelmakirjaston dokumentaatio

Tampereen ammattikorkeakoulu

Ohjelmakirjaston käyttöopas

TwinCAT 3 -ohjelmakirjasto pienoisrautatieympäristön Märklin-laitteiden ohjaukseen

V1.0

Päivitetty 14.5.2024

SISÄLLYS

1 OHJELMAKIRJASTO

1.1 Johdanto

Tämä käyttöopas on tehty TwinCAT 3 -ohjelmakirjastolle, jolla voidaan ohjata Tampereen ammattikorkeakoulun automaatiolaboratorion pienoisrautatiehen pohjautuvan oppimisympäristön Märklin-laitteita. Oppaassa käydään läpi ohjelmakirjaston käyttöönotto ja sen sisältämät ohjelmalohkot sekä perehdytään kirjaston yksityiskohtiin ja käyttöesimerkkiin.

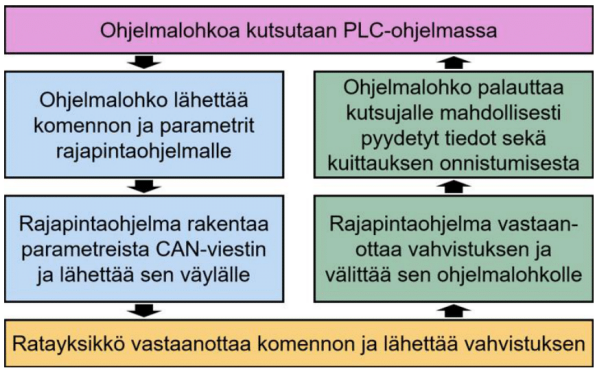
Oppaan tarkoituksena on edistää ohjelmakirjaston oman kommentoinnin rinnalla ohjelmakirjaston käytön helppoutta ja tehdä sen hyödyntämisestä vaivatonta. Ohjelmakirjaston käyttäjällä oletetaan olevan edes kohtalaisesti aikaisempaa kokemusta TwinCAT 3 -kehitysympäristön käytöstä ja PLC-ohjelmoinnista. Opas on toteutettu osana opinnäytetyötä, jonka tuloksena käsiteltävä ohjelmakirjasto ja sen rajapintaohjelma syntyivät.

1.2 Toiminta

Ohjelmakirjaston toteutuksessa on käytetty EL6751-moduulia, jonka pohjalta myös opas on kirjoitettu. Se kommunikoi pienoisrautatien Märklin-laitteiden kanssa ratayksikön välityksellä CAN-väylää käyttäen. Ohjelmakirjaston pohjalla on rajapintaohjelma, joka muuntaa lähetettävät Märklin-komennot CAN-viesteiksi ja lähettää ne moduulin kautta väylälle. Rajapintaohjelma myös välittää kaikki vastaanotetut viestit moduulilta ohjelmakirjastolle.

Käytännössä vain sellaiset vastaanotetut viestit käsitellään, jotka ovat vastauksia aiemmin lähetettyihin pyyntöihin tai vastaanottokuittauksia. Vastaanottokuittaus on ratayksikön lähettämä vahvistus siitä, että viesti lähetettiin onnistuneesti ja ratayksikkö suoritti mahdollisen komennon. Ohjelmakirjasto muodostuu lohkoista, joita voidaan kutsua PLC-ohjelmassa minkä tahansa muun funktion tapaan. Varsinaista taustatoimintaa ei siis tarvitse välttämättä tuntea.

Ohjelmakirjaston ohjelmalohkojen toimintaperiaate on esitetty kuviossa 1. Kun ohjelmalohkoa kutsutaan, sille syötetään vaadittavat parametrit kuten veturin tai lisälaitteen osoite, asetusarvo tai muu data. Ohjelmalohko lähettää ne eteenpäin rajapintaohjelmalle ja lopulta ratayksikkö vastaanottaa komennon ja lähettää vahvistuksen.



KUVIO 1. Ohjelmalohkon toiminta sitä kutsuttaessa.

Rajapintaohjelma odottaa ratayksikön vahvistusta ja välittää sen ohjelmalohkolle. Ohjelmalohko puolestaan välittää kutsujalle pyydetyt tiedot tai vähintään kuittauksen viestin lähettämisen onnistumisesta. Kuittausta odotetaan odotusajan verran, joka on yleensä 20 ms. Tämän jälkeen viestin lähetys todetaan epäonnistuneeksi, jos vastausta ei kuulu.

Lähes kaikki ohjelmalohkot vaativat veturin tai lisälaitteen osoitteen. Jotta niitä ei tarvitsisi muistaa ulkoa, ohjelmakirjaston globaaleihin muuttujiin on toteutettu kuvassa 1 esitetty osoitekirja. Se sisältää kaikkien tarvittavien laitteiden osoitteet, jolloin osoitteiden ulkoa muistamisen sijaan voidaan viitata osoitemuuttujaan.

Name	Type	Inherited from	Address	Initial	Comment
V111_TURNOUT	DWORD			16#3800	Address of V111 turnout
V112_TURNOUT	DWORD			16#3801	Address of V112 turnout
V213_TURNOUT	DWORD			16#3802	Address of V213 turnout
V214_TURNOUT	DWORD			16#3803	Address of V214 turnout
V315_TURNOUT	DWORD			16#3804	Address of V315 turnout
V316_TURNOUT	DWORD			16#3805	Address of V316 turnout
V317_TURNOUT	DWORD			16#3806	Address of V317 turnout
V218_TURNOUT	DWORD			16#3807	Address of V218 turnout
V219_TURNOUT	DWORD			16#3808	Address of V219 turnout
V230_TURNOUT	DWORD			16#3809	Address of V230 turnout
SERFER_LOCO	DWORD			16#4005	Address of the Serfer 320-001 locomotive
CONNECTOR_BOX	DWORD			16#47462665	Address of the 60114 track box

KUVA 1. Ohjelmakirjaston osoitekirja.

Osoitekirjasta löytyvät nykyhetkellä käytössä olevan yhden veturin lisäksi kaikkien vaihteiden osoitteet. Lisäksi siinä on ratayksikön osoite. Osoitekirjan nimi on GVL_AddressBook.

1.3 Ohjelmakirjaston ja käyttöoppaan muutokset

Sekä ohjelmakirjasto että tämä käyttöopas ovat versionumeroituja. Muutoksia ja päivityksiä tehdessä on tärkeää muistaa kasvattaa tätä versionumeroa, jotta voidaan erottaa eri versiot uusimmasta versiosta. Ohjelmakirjasto on rakennettu modulaarisesti ja uusien ohjelmalohkojen lisääminen tarvittaessa on helppoa.

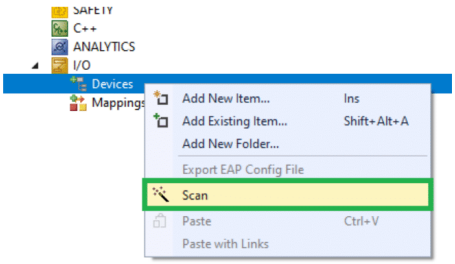
Uusien ohjelmalohkojen lisääminen on neuvottu tässä käyttöoppaassa aliluvussa 3.4. Kevyempänä vaihtoehtona uuden ohjelmalohkon lisäämiselle on myös yksittäisen komennon lähettäminen lähetysfunktioita käyttämällä.

2 LISÄÄMINEN PROJEKTIIN

Seuraavissa aliluvuissa opastetaan ohjelmakirjaston lisääminen projektiin ja sen käyttöönotto. Samalla käydään läpi olennaiset asetukset ja muut yksityiskohdat ohjelmakirjaston oikean toiminnan varmistamiseksi. Jos ohjelmakirjasto ei tunnu toimivan, kannattaa varmistaa, että se on varmasti asennettu projektiin oikein.

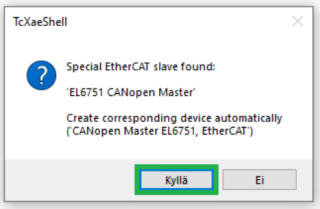
2.1 EL6751-moduulin konfigurointi

Avaa tai luo uusi projekti, johon ohjelmakirjasto on tarkoitus lisätä. Varmista, että kohdelaite on yhdistetty ja moduuli on liitetty. Avaa projektipuusta I/O-kohta ja napauta Devices-kohtaa hiiren oikealla. Valitse ”Scan” kuvan 2 mukaisesti.



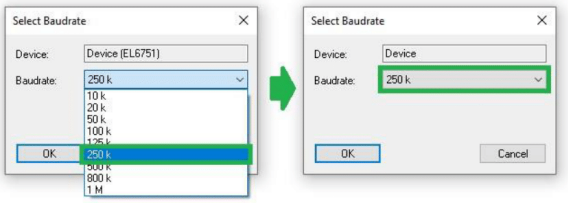
KUVA 2. Laitteiden skannaus projektipuun Devices-kohdassa.

Valitse avautuvissa ikkunoissa ”OK” tai ”Kyllä”, ellei ole tarpeen jostain syystä valita toisin. Jatka, kunnes näet kuvassa 3 esitetyn ikkunan, joka ilmoittaa EL6751-moduulin löytymisestä.



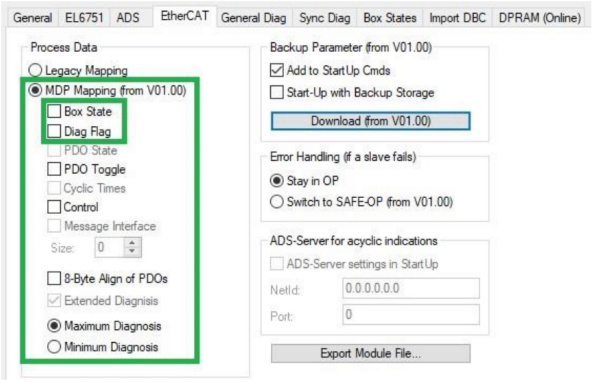
KUVA 3. Skannaustoiminto löysi EL6751-moduulin.

Valitse ”Kyllä”, minkä jälkeen kysytään käytettävää tiedonsiirtonopeutta. Se on Märklinin laitteissa 250 kbit/s, jolloin alasvetovalikosta valitaan ”250 k” (kuva 4). Kun tiedonsiirtonopeus on valittu, valitse ”OK”.



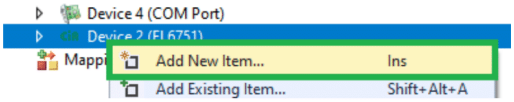
KUVA 4. Tiedonsiirtonopeuden asetus.

Moduuli on nyt lisätty projektiin ja sen pitäisi näkyä projektipuussa. Seuraavaksi kaksoisnapauta lisättyä moduulia avataksesi asetusnäkymän. Valitse EtherCAT-välilehdellä ”MDP Mapping (from V01.00)” ja poista sen alta valinnat ”Box State” ja ”Diag Flag” kuvan 5 mukaisesti.



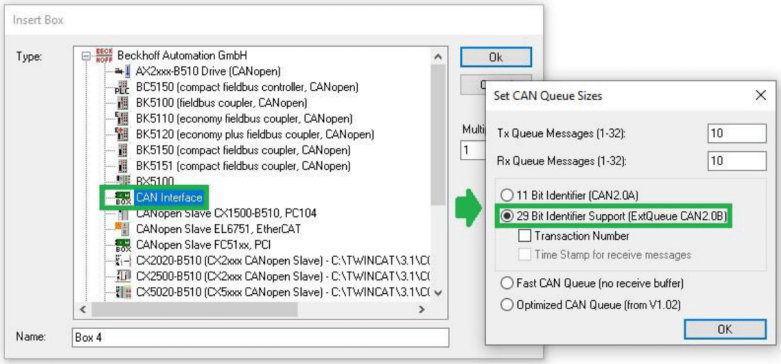
KUVA 5. EtherCAT-välilehden asetusmuutokset.

Luodaan seuraavaksi moduulille CAN-rajapinta. Napauta moduulia hiiren oikealla ja valitse ”Add New Item…” kuvan 6 osoittamalla tavalla.



KUVA 6. Uuden kohteen lisääminen moduuliin.

Valitse avautuvasta valikosta ”CAN Interface”. Seuraavasta asetusnäkymästä valitse ”29 Bit Identifier Support (ExtQueue CAN 2.0B”, koska Märklinin laitteet käyttävät jatkettua 29-bittistä tunnistetta CAN-kehyksessään. Jätä muut kohdat ennalleen. Valinnat on esitetty kuvassa 7. Valitse sitten ”OK”.

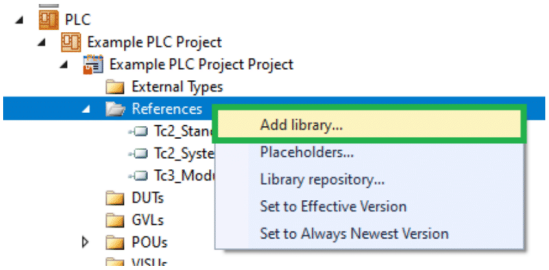


KUVA 7. CAN-rajapinnan lisääminen.

Moduuli on nyt lisätty projektiin ja siinä on CAN-rajapinnan myötä tulo- ja lähtömuuttujat. Nämä muuttujat linkitetään seuraavassa aliluvussa ohjelmakirjaston muuttujiin.

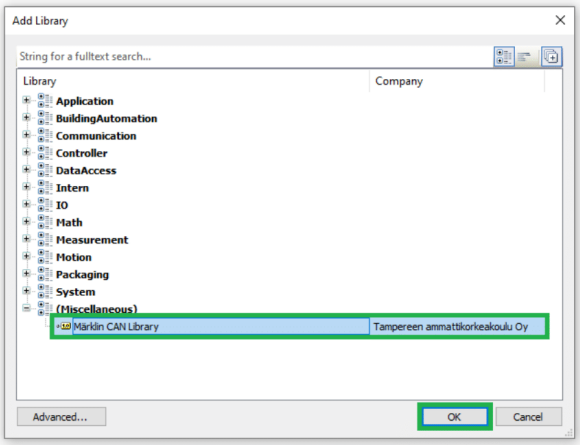
2.2 Ohjelmakirjaston lisääminen

Ohjelmakirjastoa varten avaa olemassa oleva tai luo uusi PLC-projekti tarpeen mukaisesti. Napauta References-kohtaa hiiren oikealla ja valitse ”Add Library…” kirjaston lisäämiseksi. Valinta on esitetty kuvassa 8.



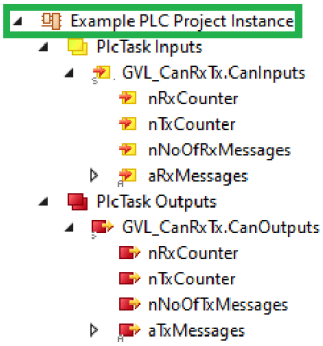
KUVA 8. Uuden kirjaston lisääminen PLC-projektiin.

Valitse avautuvasta valikosta Miscellaneous-kohdan alta ”Märklin CAN Library” ja valitse ”OK”. Varmista, että ohjelmakirjasto on asennettu järjestelmään, jos et löydä sitä valikosta. Valinta näytetään kuvassa 9.



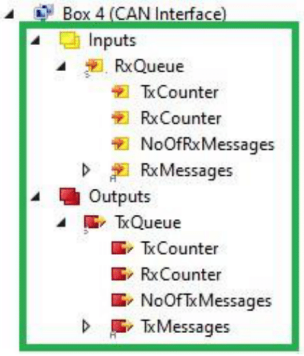
KUVA 9. Ohjelmakirjaston valinta.

Valitse vielä ”Build Solution”, jolloin lisätyn ohjelmakirjaston tulo- ja lähtömuuttujat luodaan projektiin sen instanssin alle. Muuttujat on esitetty kuvassa 10.



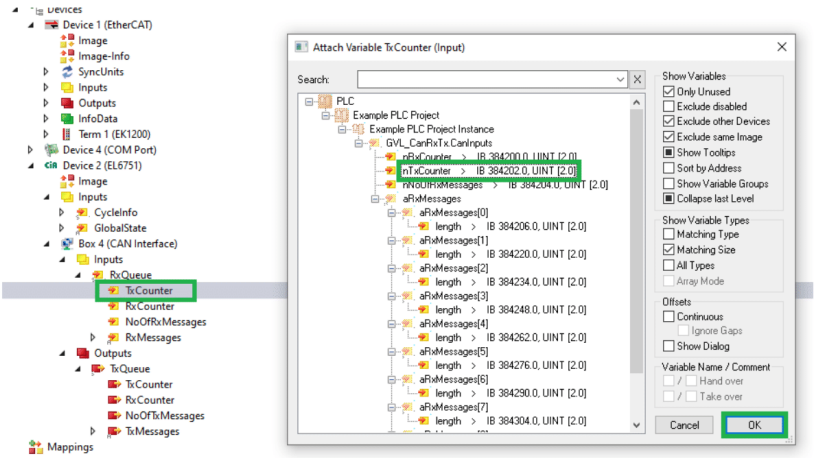
KUVA 10. PLC-projektin tulo- ja lähtömuuttujat, kun ohjelmakirjasto on lisätty.

PLC-projektin tulo- ja lähtömuuttujat pitää liittää moduulin luomiin vastaaviin muuttujiin, jotta tiedonsiirto niiden välillä toimisi. Moduulin tulo- ja lähtömuuttujat löytyvät CAN-rajapinnan alta kuvan 11 osoittamalla tavalla.



KUVA 11. Moduulin tulo- ja lähtömuuttujat.

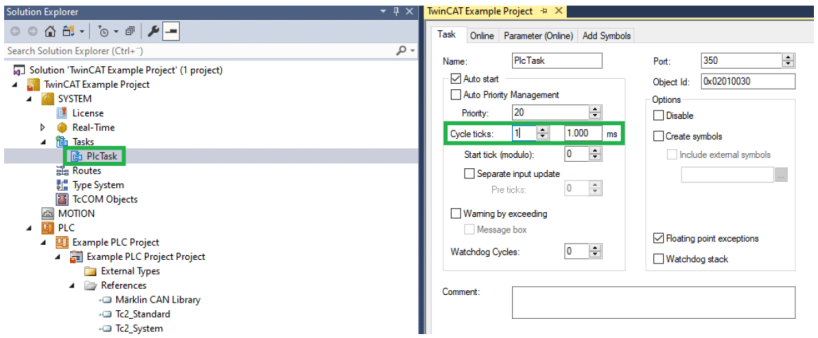
Linkitä kukin moduulin tulo- ja lähtömuuttuja vastaavan nimiseen muuttujaan PLC-projektissa. Linkitys on esitetty kuvassa 12. Linkitys onnistuu esimerkiksi kaksoisnapauttamalla muuttujaa projektipuussa ja valitsemalla sen jälkeen siihen linkitettävä muuttuja. Lopuksi valitaan ”OK”.



KUVA 12. Moduulin ja ohjelmakirjaston muuttujien linkitys toisiinsa.

Toista sama sekä tulo- että lähtömuuttujien kullekin neljälle muuttujalle. Muuttujat RxMessages ja TxMessages kannattaa linkittää yhtenä taulukkomuuttujana sen sijaan, että linkittäisi jokaisen sen sisältämän muuttujan erikseen. Lopuksi on vielä tärkeää vaihtaa **ohjelmakirjastoa suorittavan tehtävän sykliajaksi 1 ms**.

Se tehdään SYSTEM-kohdan alta Task-kohdasta. Kaksoisnapauta kirjaston käyttämää tehtävää ja syötä ”Cycle ticks” kohtaan arvo 1 kuvan 13 mukaisesti.



KUVA 13. Sykliajan muuttaminen.

Ohjelmakirjasto on nyt valmis käytettäväksi. Se on lisätty projektiin ja linkitetty EL6751-moduuliin. Seuraavissa luvuissa tutustutaan tarkemmin sen sisältämiin ohjelmalohkoihin ja perehdytään ohjelmakirjaston käyttöön.

3 OHJELMALOHKOT JA NIIDEN KÄYTTÖ

Ohjelmalohkoilla tarkoitetaan eri toimintoja, joilla pienoisrautatietä voidaan ohjata. Kaikki ohjelmakirjaston ohjelmalohkot esitetään seuraavilla sivuilla. Ohjelmalohkoissa on IN-sisääntulomuuttuja sekä SUCCESS- ja ERROR-ulostulomuuttujat, jotka kaikki ovat totuusarvoja. Sisääntulomuuttuja toimii ohjelmalohkon herätteenä, eli ohjelmalohko suoritetaan sen ollessa TOSI.

Ohjelmalohkon pyytämät tulokset ja tiedot kuten ulostulomuuttujien arvot, senhetkinen veturin suunta tai nopeusohje säilytetään, kunnes IN-muuttuja on EPÄTOSI. Ulostulomuuttujat SUCCESS ja ERROR kertovat toiminnon suorittamisen onnistumisesta. Jos ohjelmalohko suoritetaan onnistuneesti ja ratayksiköltä vastaanotetaan kuittaus, SUCCESS-muuttuja on TOSI. Vastaavasti kuittauksen puuttuessa tai muussa virhetilanteessa ERROR-muuttuja on TOSI.

Tärkeä ohjelmalohko ohjelmakirjaston toiminnan kannalta on FB_CanIOHandler, joka täytyy lisätä ohjelmakiertoon rajapintaohjelman toimimiseksi. Siihen on koottu kaikki tarvittavat alafunktiot, kuten viestien lähettäminen ja vastaanottaminen sekä jonojen käsittely.

Kaikilla ohjelmalohkoilla yhteiset sisään- ja ulostulomuuttujat:

Sisääntulomuuttujat

IN	*BOOL*	Kun TOSI, ohjelmalohko suoritetaan ja sen tulokset säilytetään, kunnes taas EPÄTOSI.

Ulostulomuuttujat

SUCCESS	BOOL	Palauttaa arvon TOSI, jos ohjelmalohkon suorittaminen onnistui ja saatiin vastaanottokuittaus. Muussa tapauksessa EPÄTOSI.
ERROR	BOOL	Palauttaa arvon TOSI, jos ohjelmalohkon suorittaminen epäonnistui tai vastaanottokuittausta ei saatu. Muussa tapauksessa EPÄTOSI.

3.1 Veturitoiminnot

FB_SetLocoSpeed

Aseta veturin nopeusohje.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nSpeedIn	*WORD*	Veturille asetettava uusi nopeusohje. Lukuarvo välillä 0…1000.

Selitys

Veturin nopeus määräytyy nopeusohjeen ja sen erikseen konfiguroidun enimmäisnopeuden mukaan. Nopeusohje voidaan asettaa välillä 0…1000. Nopeusohje 0 vastaa pysähtynyttä veturia ja 1000 veturin enimmäisnopeutta.

FB_AccelerateLoco

Kasvata veturin nopeusohjetta tietyn askeleen verran.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nSpeedStepIn	*WORD*	Askeleen suuruus lukuarvona, jonka verran nopeusohjetta kasvatetaan.

Selitys

Veturin nopeusohjetta voidaan kasvattaa enimmäisarvoon 1000 saakka. Ohjelmalohko kyselee ensin veturin senhetkisen nopeusohjeen, minkä jälkeen veturille annetaan nopeusaskeleen verran kasvatettu nopeusohje.

FB_DecelerateLoco

Vähentää veturin nopeusohjetta tietyn askeleen verran.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nSpeedStepIn	*WORD*	Askeleen suuruus lukuarvona, jonka verran nopeusohjetta vähennetään.

Selitys

Veturin nopeusohjetta voidaan vähentää arvoon 0 saakka. Ohjelmalohko kyselee ensin veturin senhetkisen nopeusohjeen, minkä jälkeen veturille annetaan nopeusaskeleen verran pienennetty nopeusohje.

FB_GetLocoSpeed

Kysy veturille asetettua nopeusohjetta.

Yhteisten lisäksi yksi sisääntulomuuttuja ja yksi ulostulomuuttuja.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.

Ulostulomuuttujat

nSpeedOut	*WORD*	Veturille asetettu nopeusohje 0…1000.

Selitys

Ratayksikkö palauttaa veturille sillä hetkellä asetetun nopeusohjeen. On tärkeää huomata, että nopeusohje ei ole sama asia kuin veturin todellinen nopeus. Todellista nopeutta ei voida kysellä, sillä ratayksikkö ei tiedä sitä.

FB_SetLocoDirection

Aseta veturin kulkusuunta.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nDirectionIn	*BYTE*	Veturille asetettava kulkusuunta. Lukuarvo välillä 0…3.

Selitys

Veturin kulkusuunta voidaan asettaa seuraavasti:

0 = säilytä veturin kulkusuunta samana

1 = aseta kulkusuunta eteenpäin

2 = aseta kulkusuunta taaksepäin

3 = vaihda veturin kulkusuuntaa

Jos uusi ja nykyinen kulkusuunta ovat samat, kulkusuunta säilyy samana.

FB_ToggleLocoDirection

Vaihda veturin kulkusuuntaa.

Yhteisten lisäksi yksi sisääntulomuuttuja ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.

Selitys

Yksinkertaistettu versio ohjelmalohkosta FB_SetLocoDirection, kun halutaan vain vaihtaa kulkusuuntaa.

FB_GetLocoDirection

Kysy veturin kulkusuuntaa.

Yhteisten lisäksi yksi sisääntulomuuttuja ja yksi ulostulomuuttuja.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.

Ulostulomuuttujat

nDirectionOut	*BYTE*	Veturin kulkusuunta. Lukuarvo 1 tai 2.

Selitys

Ratayksikkö palauttaa veturin senhetkisen kulkusuunnan seuraavasti:

1 = eteenpäin

2 = taaksepäin

FB_SetLocoFunction

Aseta veturin toiminnon tila.

Yhteisten lisäksi kolme sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nFunctionIn	*BYTE*	Määrittää toiminnon numeron. Lukuarvo välillä 0…31.
nFunctionValueIn	*BYTE*	Määrittää toiminnon intensiteetin. Lukuarvo välillä 0…31.

Selitys

Veturin toiminnot, niiden numerot sekä intensiteetin arvot löytyvät usein veturin käyttöoppaasta. Toimintoja ovat esimerkiksi valot ja äänimerkit. Toiminnon intensiteetillä tarkoitetaan toiminnon voimakkuutta, kuten valojen kirkkautta. Jos toiminnolla on vain kaksi arvoa, 0 kytkee toiminnon pois päältä ja arvo 1 päälle.

FB_GetLocoFunction

Kysy veturin toiminnon tilaa.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja yksi ulostulomuuttuja.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nFunctionIn	*BYTE*	Määrittää toiminnon numeron. Lukuarvo välillä 0…31.

Ulostulomuuttujat

nIsFunctionActiveOut	*BOOL*	Veturin toiminnon tila.

Selitys

Ratayksikkö palauttaa veturin toiminnosta vain tiedon, onko se aktiivinen vai ei.

FB_ToggleLocoFunction

Vaihda veturin toiminnon tila päinvastaiseksi.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nLocoAddressIn	*DWORD*	Määrittää kohdeveturin osoitteen.
nFunctionIn	*BYTE*	Määrittää toiminnon numeron. Lukuarvo välillä 0…31.

Ulostulomuuttujat

nIsFunctionActiveOut	*BOOL*	Veturin toiminnon tila.

Selitys

Yksinkertaistettu versio ohjelmalohkosta FB_SetLocoFunction, kun halutaan vaihtaa veturin toiminnon tilaa.

3.2 Lisätarvikkeiden toiminnot

FB_SetAccessory

Vaihda veturin toiminnon tila päinvastaiseksi.

Yhteisten lisäksi kolme sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nAccessoryAddressIn	*DWORD*	Määrittää kohdelaitteen osoitteen.
nAccessorySettingIn	*BYTE*	Määrittää asetuksen numeron. Lukuarvo välillä 0…31.
nAccessoryIntensityIn	*BYTE*	Määrittää toiminnon intensiteetin. Lukuarvo välillä 0…31.

Selitys

Lisätarvikkeiden asetukset ja niiden intensiteetin arvot löytyvät usein laitteen käyttöoppaasta. Lisätarvikkeita ovat esimerkiksi vaihteet ja opasteet. Asetuksen numero voi olla esimerkiksi tietyn kohdelaitteen tietty lamppu, jolle asetetaan tietty intensiteetti. Kumpikin arvo voidaan asettaa välillä 0…31.

FB_SetTurnout

Aseta vaihteen asento.

Yhteisten lisäksi kaksi sisääntulomuuttujaa ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

nTurnoutAddressIn	*DWORD*	Määrittää kohdelaitteen osoitteen.
bSetStraightIn	*BOOL*	Määrittää vaihteen asennon.

Selitys

Yksinkertaistettu versio ohjelmalohkosta FB_SetAccessory. Kun vaihteen asento halutaan kääntää suoralle vaihteelle, bSetStraightIn asetetaan arvoon TOSI. Se asetetaan arvoon EPÄTOSI, kun vaihde halutaan kääntää poikkeavalle raiteelle.

3.3 Muut toiminnot

FB_SetTrackPower

Kytke raiteiden virta päälle tai pois päältä.
Yhteisten lisäksi yksi sisääntulomuuttuja ja ei yhtään ulostulomuuttujaa.

Sisääntulomuuttujat

bPowerIn	*BOOL*	Määrittää raiteiden virran tilan.

Selitys

Raiteelle voidaan kytkeä virta muuttujan bPowerIn arvolla TOSI ja virta voidaan sammuttaa arvolla EPÄTOSI. Alustaa virrat kytkettäessä myös rataprotokollat.

FB_VersionPinger

Kysy ratayksikön käyttöliittymän versiota ja selvitä vastauksen vasteaika.
Yhteisten lisäksi ei yhtään sisääntulomuuttujaa ja kaksi ulostulomuuttujaa.

Ulostulomuuttujat

nPingMsOut	*UINT*	Ratayksikön vasteaika millisekunteina.
nVersionOut	*WORD*	Ratayksikön käyttöliittymän versio.

Selitys

Ratayksikkö palauttaa nykyisen käyttöliittymänsä version. Kun ohjelmalohko suoritetaan, se mittaa samalla ajan suorittamisesta vastauksen saamiseen. Tämä vasteaika palautetaan millisekunteina.

3.4 Uusien ohjelmalohkojen luominen ja mukautetut komennot

Ohjelmalohkoissa hyödynnetään kahta lähetysfunktiota:

- FB_CanSendUnVerified, jolla voidaan lähettää Märklin-komentoja ilman, että niihin odotetaan vastausta tai muutenkaan seurataan onnistumista. Ratayksikkö saattaa kuitenkin vahvistaa komennon tästä huolimatta, sitä ei vain rekisteröidä.
- FB_CanSend, jolla voidaan lähettää Märklin-komentoja vahvistetusti eli siten, että ratayksiköltä odotetaan kuittaus komennon vastaanottamisesta. Komento todetaan onnistuneeksi vain, jos vahvistus saapuu määritetyn odotusajan kuluessa. Tämä on suositellumpi funktio, ellei ole erityisesti tarpeen lähettää komentoa ilman vahvistusta.

Uusi ohjelmalohko luodaan määrittämällä Märklinin dokumentaation avulla komennon numero, pituuskoodi ja tarvittavat datatavut. Ne voidaan asettaa joko valmiiksi, jos ne eivät muutu, tai käyttäjän syöttämänä, kuten veturin osoite. Sen jälkeen kutsutaan sopivaa lähetysfunktiota näillä parametreilla ja odotetaan vastausta. Vastaus voidaan määrittää joko suoraan ulostulomuuttujin avulla tai tiettyjen ehtojen perusteella vastaanotettua viestiä analysoimalla. FB_CanSend palauttaa aina koko vastauksena saadun viestin.

FB_CanSendUnVerified

Lähetä Märklin-komento ilman vastaanottokuittausta.
Neljä sisääntulomuuttujaa ja yksi ulostulomuuttuja.

Sisääntulomuuttujat

nCommandIn	*BYTE*	Suoritettava Märklin-komento.
nResponseBitIn	*BOOL*	Vastausbitin tila.
nLengthIn	*UINT*	Märklin-komennon pituuskoodi.
arrDataIn	*CANMESSAGE*	Lähetettävät datatavut.

Ulostulomuuttujat

F_CanSendUnVerified	*BOOL*	Palauttaa arvon TOSI, jos komento lisättiin jonoon onnistuneesti.

FB_CanSend

Lähetä Märklin-komento ilman vastaanottokuittausta.
Viisi sisääntulomuuttujaa ja seitsemän ulostulomuuttujaa.

Sisääntulomuuttujat

IN	*BOOL*	Kun TOSI, ohjelmalohko suoritetaan ja sen tulokset säilytetään, kunnes taas EPÄTOSI.
nCommandIn	*BYTE*	Suoritettava Märklin-komento.
nLengthIn	*UINT*	Märklin-komennon pituuskoodi.
arrDataIn	*CANMESSAGE*	Lähetettävät datatavut.
nTimeoutIn	*TIME*	Odotusaika millisekunteina, jonka verran odotetaan kuittausta viestiin.

Ulostulomuuttujat

RECEIVED	*BOOL*	Palauttaa arvon TOSI, jos komento lisättiin jonoon onnistuneesti.
BUSY	*BOOL*	Palauttaa arvon TOSI, jos lähetysfunktio on käynnissä, muutoin arvon EPÄTOSI.
ERROR	*BOOL*	Palauttaa arvon TOSI, jos viestiä ei voitu lisätä lähetysjonoon tai siihen ei saatu vastausta odotusajan sisällä.
nCommandOut	*BYTE*	Ratayksiköltä vastaanotetun viestin Märklin-komento.
nHashOut	*WORD*	Ratayksiköltä vastaanotetun viestin hajautusarvo.
nLengthOut	*UINT*	Ratayksiköltä vastaanotetun viestin pituuskoodi eli datatavujen määrä.
arrDataOut	*CANMESSAGE*	Ratayksiköltä vastaanotetun viestin datatavut.

Lähetysfunktioiden selitykset

Kumpaankin funktioon tuodaan sisään lähetettävä Märklin-komento, pituuskoodi sekä lähetettävät datatavut. FB_CanSendUnVerified-funktioon syötetään lisäksi vastausbitti eli tieto siitä, onko viesti pyyntö vai vastaus aiempaan pyyntöön. Tätä parametria ei ole vahvistetulla lähetysfunktiolla FB_CanSend, koska kaikki sen lähettämät viestit ovat pyyntöjä, joihin odotetaan vastausta. Vahvistettu funktio tarvitsee myös odotusajan. Se palauttaa tiedon onnistumisesta ja vastausviestin.

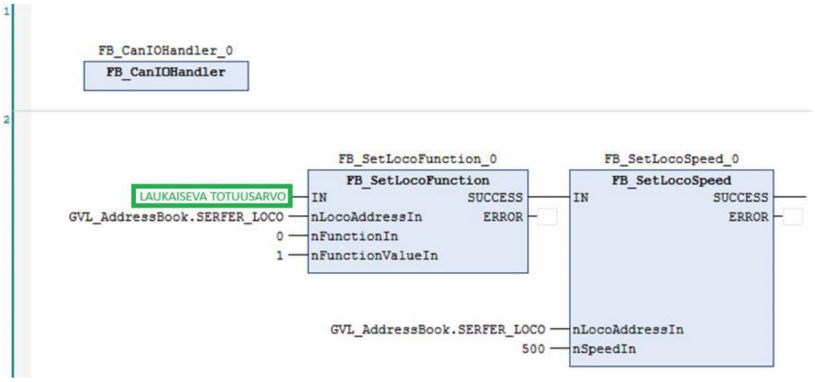
3.5 Sisäiset funktiot

Rajapintaohjelmassa on kansion ”Interface Functions (Internal)” sisällä rajapinnan sisäiseen toimintaan tarkoitettuja funktioita. Niitä ei ole tarkoitus kutsua ohjelmakirjastoa käytettäessä, vaan ne on tarkoitettu rajapintaohjelman käyttöön. Kaikki niiden mahdollistamat toiminnot voidaan suorittaa myös käytettäviksi tarkoitetuilla ohjelmalohkoilla, jotka on esitetty edellisissä aliluvuissa.

Sisäisten funktioiden globaalit muuttujat ovat listassa GVL_CanRxTx. Kuten itse funktioita, myöskään niitä ei ole tarkoitettu käyttäjän käytettäväksi. Vääränlainen käyttö voi pahimmassa tapauksessa johtaa ohjelman virheelliseen toimintaan tai onnettomuuksiin raiteilla, jos virheviestejä ei saada välitettyä oikein.

4 KÄYTTÖESIMERKKI

Käyttöesimerkkinä esitetään kuvassa 14 ohjelma, jossa ohjataan veturin ajovalot päälle ja asetetaan nopeusohjeeksi arvo 500. Veturin osoitteen määrittämisessä käytetään apuna osoitekirjaa, josta on valittu osoitemuuttuja SERFER_LOCO. Ohjelman alussa on FB_CanIOHandler, joka tulee sisällyttää ohjelmakirjastoa käytettäessä ohjelmakiertoon rajapintaohjelman toimimiseksi.



KUVA 14. Ohjelmakirjaston käyttöesimerkki.

Ajovalot kytketään päälle ohjelmalohkolla FB_SetLocoFunction. Ajovalojen toiminnon numero on 0 ja koska ne kytketään päälle, arvoparametri on 1. Nopeus asetetaan ohjelmalohkolla FB_SetLocoSpeed, johon syötetään vain nopeusaskel 500. Ohjelmalohkot suoritetaan peräkkäin, eli edellisen lohkon onnistumisen vahvistava ulostulomuuttuja laukaisee seuraavan ohjelmalohkon.

Ensimmäinen ohjelmalohko voidaan laukaista halutulla totuusarvomuotoisella herätteellä. ERROR-ulostulomuuttujiin voitaisiin liittää toiminnallisuus, kuten hätäseis tai muu reaktio kriittisen viestin lähettämisen epäonnistumiselle.