

SIEMENS
Ingenuity for life



Discrete event simulation as a tool for virtual commissioning

Using a discrete event simulation model as a base for generation and verification of rudimentary PLC logic

Master's thesis in Production Engineering

Marcus Andersson
Atle Zvantesson

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

MASTER'S THESIS 2018:EENX-30

Discrete event simulation as a tool for virtual commissioning

Using a discrete event simulation model as a base for generation and verification of rudimentary PLC logic

Marcus Andersson
Atle Zvantesson



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of System and Control
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Discrete event simulation as a tool for virtual commissioning
Using a discrete event simulation model as a base for generation and verification of
rudimentary PLC logic
Marcus Andersson
Atle Zvantesson

© Marcus Andersson & Atle Zvantesson, 2018.

Supervisors:
Henrik Carlsson, Volvo Cars
Maria Ludvigsson, ÅF

Examiner:
Petter Falkman, Department of Electrical Engineering

Master's Thesis 2018:EENX-30
Department of Electrical Engineering
Division of System and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Picture showing parts of an electric monorail system modelled in Plant Simulation.

Typeset in L^AT_EX
Gothenburg, Sweden 2018

Discrete event simulation as a tool for virtual commissioning
Using a discrete event simulation model as a base for generation and verification of
rudimentary PLC logic
Marcus Andersson
Atle Zvantesson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Virtual commissioning is growing more popular in industry today, as it allows companies to develop their production quicker and at a lower price since changes can be tested and evaluated without stopping the real-world system. Virtual commissioning is used for many different applications and projects. The purpose of this thesis is to evaluate if the discrete event simulation (DES) program Plant Simulation can be used as tool for virtual commissioning of Programmable Logic Controller (PLC) code.

This will be tested by modelling an electric monorail system, with a line at Volvo Cars Torslanda plant used as a reference. This digital twin will be used as a base for the design of the monorail control system. The model will at the same time be connected and controlled by a virtual PLC to make sure that the logic works as intended. Testing of the PLC code against the Plant Simulation model will be done continuously as new functionalities are added. In the end the whole model should be controlled by the PLC rather than the internal logic.

The next step is to investigate if and how the PLC code can be automatically generated based on data from the Plant Simulation model. This is achieved by creating a library in the PLC of all relevant functions that is needed to control the monorail. This library is then used to create function block instances for every rail in the Plant Simulation model.

What follows then is a summation of the method results, as well as a list of improvements that would allow Plant Simulation to get a more widespread use as a virtual commission tool.

Keywords: Virtual commissioning, Discrete event simulation, PLC code auto generation, Plant Simulation, TIA Portal Openness

Acknowledgements

We would like to thank everyone who supported us and our work during this project. However, we would also like to thank the following people a little extra (in no particular order):

- **Henrik Carlsson** at Volvo Cars for pointing us in the right direction and helping us when we got stuck.
- **Maria Ludvigsson** at ÅF for being a great mentor who answered every question we asked and who kept us pushing forward
- **Pär Ström** at ÅF for helping us to find a topic that both we and our stakeholders found exciting
- **Johan Nordling** at Siemens for helping us with the licenses for the TIA-portal and PLCSIM Advanced
- **Hans Sjöberg** at Chalmers University for helping us with all licensing troubles encountered in the start of the project
- **Eduard Kapoun** at Summ Systems for helping us understand the connection between Plant Simulation and PLCSIM Advanced

Finally we would also like to say thanks to **Petter Falkman** for his role as the examiner for this project.

Marcus Andersson and Atle Zvantesson, Gothenburg, 2018

Contents

List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Background	1
1.2 Scope	2
1.3 Aim	2
1.4 Research questions	2
2 Theory	3
2.1 Virtual commissioning	3
2.2 Discrete Event Simulation	3
2.2.1 Technomatix Plant Simulation	4
2.3 PLC	5
2.3.1 PLC Programming languages	5
2.3.2 STEP 7 TIA Portal	5
2.3.3 PLCSIM Advanced	5
2.3.4 TIA Portal Openness	7
2.4 Open platform communication	7
2.5 Electric Monorail System	7
2.5.1 Volvo Cars EMS	7
2.6 Previous research	9
3 Methods	10
3.1 Connecting Plant Simulation and PLCSIM Advanced	10
3.1.1 Creating the connection	10
3.1.2 Sending data	11
3.1.3 Building the test model	12
3.2 Coding with a standard	13
3.2.1 Plant Simulation standard	14
3.2.2 PLC standard	14
3.2.3 PLC hierarchy	16
3.3 Modelling the EMS	16
3.4 Auto-generation of PLC code	18

4	Results	20
4.1	Hardware differences between PLC:s	20
4.2	Lack of detail in Plant Simulation	20
4.2.1	Measurement problems in EMS	20
4.3	Slowdowns when running Plant Simulation with a virtual PLC	21
4.4	Connection between Plant Simulation and TIA Portal	21
4.5	Creating the PLC code	21
4.6	Lack of documentation	22
5	Conclusion	23
5.1	Improvement potential in the software	23
5.2	Limitations of the connection between Plant Simulation and PLC	24
5.3	The extent to which PLC code can be designed based on a Plant Simulation model	24
5.4	Future research topics	25
5.5	Closing statements	25
	Bibliography	26
A	Connecting PLCSIM Advanced to the TIA Portal	I
A.1	Setting up the Virtual PLC	I
A.1.1	Using the Virtual Ethernet Adapter	II
A.2	Configuring TIA for simulation	II
A.3	Connecting and downloading to the virtual PLC	II
A.4	Converting 300 code to 1500	III
B	Connecting Plant Simulation to a virtual PLC	VI
B.1	Setting up the connection	VI
B.1.1	Sending data to the PLC	VIII
B.1.2	Receiving data from the PLC	IX
B.2	Connecting to a remote virtual PLC	X
B.3	Understanding the data exchange interval	XI
B.3.1	Notes regarding the data exchange	XII
C	Control logic of the test model	XIII
D	SIMTALK code for exporting data	XVI
E	XML code for safety zone 320	XX
E.1	Results from imported code	XXII
F	PLC Standard	XXXII
F.1	Behavior of the Function Blocks in GlobalLib	XXXII

List of Figures

2.1	The connection between PLCSIM Advanced Runtime Instance on one PC to the STEP 7 TIA Portal on another PC[7].)	6
2.2	Overview of the EMS line. The assembly station is located in the lower left corner. Notice the XC90 track in black and the V90 track in green.	8
3.1	The dialog window of the PLCSIM_Advanced object in Plant Simulation	11
3.2	A table showing all IN-signals to the PLC as well as their aliases and connected model attributes	12
3.3	3D view of the model used to test the connection with PLCSIM Advanced. The Connection is set up with the PLCSIM object on the left of the line	13
3.4	A Plant Simulation Toolbox with nine different types of rail	14
3.5	The function block library in the PLC	15
3.6	An example of a rail control Function Block, showing the blocks inputs to the left of the block, and outputs on the right side of the block.	15
3.7	A graphical representation of the differences between two instances of the same rail type.	16
3.8	A graphical representation of the hierarchy of the PLC program. The main block can not communicate directly with the function blocks. However, blocks on the same level depend on information from each other.	17
3.9	2D view of a part of the Pant Simulation Model. The coloured backgrounds represents different safety zones	17
3.10	The PLC template for an ACC network with generic signals and variables at the connections instead of instance specific ones.	18
3.11	A text-file a list of all rails in safety zone 340, and a list of all XML templates used for autogenerating a new PLC code.	19
4.1	An error prompt in Plant Simulation. There are no available documentation on why this error occurs or how to fix the problem.	22
A.1	The PLCSIM Advanced Interface for Online Access and TCP/IP communication.	I

A.2	The Start tab in PLCSIM Advanced showing the an example of instance name and the correct IP address and Sub-net mask.	I
A.3	The PLCSIM Advanced Interface.	II
A.4	Showing the Siemens PLCSIM Virtual switch protocol installed on the physical Ethernet adapter Inter(R) Ethernet Connection and the Siemens PLCSIM Virtual Ethernet Adapter in the Network Connections list in the Windows Control Panel.	III
A.5	Showing where to find the project properties menu.	IV
A.6	Showing where to check Support simulating during block compilation in the project properties menu.	IV
A.7	Showing how to search for devices in the TIA Portal.	V
A.8	Showing the Extended donwload to device window in the TIA Portal.	V
B.1	The Manage Class Library screen. Activated objects have a ticked checkbox	VI
B.2	The toolbar in Plant Simulation with the PLCSIM_Advanced object selected	VII
B.3	Figure showing how to connect the PLCSIM_Advanced object to the virtual PLC	VII
B.4	A table showing all IN-signals to the PLC as well as their aliases and connected model attributes	VIII
B.5	Two OUT-signals connected to a method and a global variable respectively	IX
B.6	Figure showing how to input the address and port number when connecting to a remote virtual PLC	X
B.7	Figure showing the data exchange interval fields in the PLCSIM_Advanced object and the item groups	XI
C.1	The PLC code used for controlling the test model	XIV
C.2	The list of IN-signals and their connections in Plant Simulation . . .	XV
C.3	The list of OUT-signals and their connections in Plant Simulation . .	XV
E.1	The resulting PLC code, in ladder logic, for safety zone 320	XXXI

List of Tables

3.1	The four categories of signals that can be imported to Plant Simulation	10
3.2	Benefits and drawbacks of using different standards	14
3.3	The function of the rail control Function Blocks I/O signals	16

1

Introduction

Virtual commissioning is becoming a more and more popular tool for companies that want to expand or change their production. Proper use of Virtual commissioning reduces both the cost and time needed to implement large-scale changes. Changes to existing lines and factories can be evaluated without stopping production and layouts for new factories can be designed and tested in the virtual world before the physical factory has been constructed.

1.1 Background

At the plant in Volvo Cars Torslanda there is an Electrical Monorail System (EMS) used in production with PLC control. As well as functional PLC code there also exists an DES model used for production flow simulation. However the current DES model is independant of the PLC as can hence not be used for Virtual commissioning of the PLC.

Today there exists a multitude of programs for virtual commissioning of production cells and shorter lines, such as Process Simulate and Robot Studio, but no tool for larger scopes. Volvo Cars, together with ÅF and Siemens, wants to evaluate whether or not the discrete event simulation program Plant Simulation can be used as such a tool. The general idea is to use the Plant Simulation model as a base for the development of the PLC control logic for the real system. The code should not create the whole program at once, but should instead be scalable enough for continuous development during the design of the production.

If successful, it means that the general design of the control program can be created in the early design-stages of a project, thus shortening the development time and lessening the workload of the PLC programmers. It could also introduce new ideas to the line-builders if the PLC engineers are involved in parts of the decision making, *i.e.* designing the production system and the control logic together. This plays in to a broader push in industry of having cross functional teams and involving more functions earlier in new projects.

If the connection between Plant Simulation and the PLC is successful it would also be useful to find out to which extent the PLC code can be auto-generated. The code should then preferably be created based on information from the Plant Simulation model. The best-case scenario would be if the PLC code were continuously updated in conjunction with changes in the Plant Simulation model.

1.2 Scope

The primary focus of the model will be the monorail itself. Modelling switches, junctions and stations will have a lower priority. Information such as process times for machines, the total number of hangers and breakdown data will be largely ignored, as the aim of the project is to evaluate the connection between Plant Simulation and the PLC rather than production flow analysis. Furthermore, the real EMS line contains a service station that will not be modelled, as breakdowns of the hangers will be ignored.

The PLC code will control an EMS line according to a standard but not feature any alarms or HMI:s.

1.3 Aim

The primary goal of this project is to evaluate if and how a discrete event simulation model can be used for designing or creating PLC code. A secondary aim of auto generation of the PLC code was added later a supplement to the main question.

1.4 Research questions

This project can be summarised by a research question, that will help focus the work to specific areas:

How could a DES program be used by engineers as a tool for virtual commissioning of PLC?

The question can further be split into three sub-questions:

RQ1 What limitations are there in the connection between Plant Simulation and PLC?

RQ2 To which extent could such a code be programmed, *i.e.* which abstraction level is appropriate?

RQ3 What is needed in the future for Plant Simulation to get widespread use for this type of application?

These questions will be handled throughout the thesis and further investigated in section 5.

2

Theory

2.1 Virtual commissioning

Virtual commissioning is a method of simulation models to test interfaces between software and hardware to identify and address design flaws and errors. As virtual commissioning can be performed without the usage of the physical system or hardware, the commissioning can be performed earlier on in the project before the physical system is completed. The fact no physical resources except a PC are necessary for virtual commissioning means that several different physical layouts and systems can also be tested without the additional expenses.

Proper use of virtual commissioning allows companies to design and evaluate changes quicker, while still having the production up and running. Virtual commissioning is becoming a more and more popular tool for companies that want to expand or change their production.

2.2 Discrete Event Simulation

Discrete event simulation (DES) is a tool that allows its users to quickly simulate and analyse many different scenarios in a short span of time. As the name suggests, events in such a system occur at discrete points in time, when they trigger a change in the system. No changes can occur in the system without an event triggering it, so the program only needs to model the events and not the time between events [1]. Such a system takes less processing power to simulate than a continuous system, as non-valuable time is ignored.

A DES model is made up of several structural components including, but not limited to: *entities*, *attributes*, *list and queues*, *events* and *Activities* [1, 2].

- In a DES model, entities are the only objects that cause any changes in the system. Entities represents all objects that requires explicit representation in a system, such as machines, products, workers and transports [1].
- Attributes are properties that describe entities. The *name* of an entity is a common attribute found in most entities, while an attribute like *colour* or *size* is more often found in products.
- Lists and queues are used to order entities according to certain attributes, such as creation time or type. Queues can also be used in buffers in to simulate ordering systems, such as first-in-first-out.

- An event is simply a change in a system. If no event is happening the system doesn't change.
- An activity on the other hand is made up of two events that mark the start and end of an operation that changes the state of an entity, such as a product being processed in a machine.

2.2.1 Technomatix Plant Simulation

Plant Simulation works in a similar way to many other DES-programs in that entities move around a system, creating new events every time a change of state happens at any place in the simulation. The moving entities in Plant Simulation generally belongs to the class *Manufacturing Units*, commonly referred to as *MU:s*. MU:s in a system usually represent products, parts, containers or transporters.

MU:s can move around the system in a few different ways, most commonly by lines, tracks or transporters. Line and track entities can represent a multitude of different conveyor types, while transporters can represent vehicles such as AGV:s and forklifts. Lines and tracks can be connected to a single machine or track to simulate one-piece flows, or to several different objects in order to create more complex systems. Furthermore, sensors can be attached at any point of a line or track. These sensors can then be connected to attributes in the passing MU or methods that will be executed when the sensor is triggered.

Plant Simulation follows an object-oriented logic, similar to other programming languages such as Java and C. This hierarchy allows users to address objects by following its file-path [3]. Furthermore, each entity possesses several attributes that describe the state of said entity. Most attributes are shared between different entity types, while others are specific to certain classes. An example of common attributes shared by all entities is names and classes. Some attributes are more specific to certain entities or classes. A machine, for example, also contains attributes that describe set-up and process times, while a MU has instead has attributes that describe its destination and colour.

One of the most useful objects in Plant Simulation is Methods. Methods are written in Siemens' own coding language SIMTALK and are used to control the simulation. A SIMTALK method could for example be used to determine the destination of an MU depending on a certain attribute, or change the speed of a conveyor.

In Plant Simulation there are several ways of triggering a SIMTALK method: a method can, for example, be used as an entrance- or exit control in a processing entity. A method can also be called from another method, which sometimes requires parameters, similar to subroutines in other coding languages.

A Plant Simulation model can be created in both 2D and 3D. Generally, a 3D environment requires much more processing power to simulate compared to a 2D one. However, a 3D environment gives users and observers a better understanding of the relationship between the model and the reality. Plant simulation can seamlessly switch from a 2D to a 3D model and vice versa, so it is possible to build the system in 2D and then adjust the positions and geometry of machines and conveyors in the 3D space to create a more accurate model of the real system.

2.3 PLC

Programmable Logic Controller (PLC) are often used in industry to control manufacturing equipment and robots in the factories. The PLC were introduced to replace relay and timer systems [4]. However the PLC is more flexible than the old relay-systems and can be reprogrammed to a different behaviour and signals response by just uploading a new code, rather than changing the connections of cables as in the old relay system. A PLC consists of multiple parts, a Central Processing Unit (CPU), interface for communication, an I/O, a programming unit, power-supply and memory unit [5].

The CPU contains one or more microprocessors that reads inputs signals and computes the correct output using the program stored in the memory unit. The communication interface allows the PLC to communicate in a network with other hardware. In such a network the PLC can collect data, handle synchronisation, verify other hardware and connections. The programming unit is used to write programs that can be transferred to the memory unit. The programming interface can be a small display with buttons for input, a terminal with a keyboard, or more commonly today, a PC connected to the PLC from which the code is downloaded to the PLC.

2.3.1 PLC Programming languages

PLC can be coded in 5 different languages following the IEC61131-3 standard [6]. Structured text and instruction list are text based languages, and while they are more easily auto-generated they are less graphical and harder to follow by operators without text based coding experience. Ladder diagrams, Functional Block Diagrams and Sequential Function Charts are all graphical languages which makes them easier to follow in real time. Ladder coding follows the structure of an old relay diagram and is and as such easier to use by operators with previous experience in relay design.

2.3.2 STEP 7 TIA Portal

STEP 7 TIA Portal is a software engineering tool for SIMATIC PLC:s. It's both a programming interface for PLC:s and a system and diagnostics tool for both real and virtual PLC controllers.

2.3.3 PLCSIM Advanced

PLCSIM Advanced is a simulation software that is capable of emulating the behaviour of a SIMATIC 1500 series PLC. The software has a direct interface between STEP 7 TIA Portal, Plant Simulation and also Process Simulate which allows it to control simulated objects as if they were controlled by a real 1500 PLC.

PLCSIM Advanced has two settings, `Local` and `Virtual Ethernet TCP/IP`. `Local` creates a soft PLC on the computer that is only accessible on the host computer. By using the `virtual Ethernet TCP/IP` mode the soft PLC is accessible on any network the host PC is connected to. `Virtual Ethernet TCP/IP` allows the PLC simulation to be run on a different computer than than the TIA portal and or Plant Simulation and hence in theory should allow increased performance (less computing time) since the simulations and interfaces are divided up between several hardwares. When running Plant Simulation on one computer and PLCSIM Advanced on another distributed communication via TCP/IP is used. The Simulated PLC runtime instance connects to PLCSIM Virtual Ethernet Adapter through PLCSIM Virtual switch, that in turn connects to the PCs Ethernet Adapter. Plant Simulation or STEP 7 on the client PC connects to the Host PCs Ethernet adapter via normal Ethernet TCP/IP communication, which in turn relays the connection to the PLC as shown in figure 2.1.

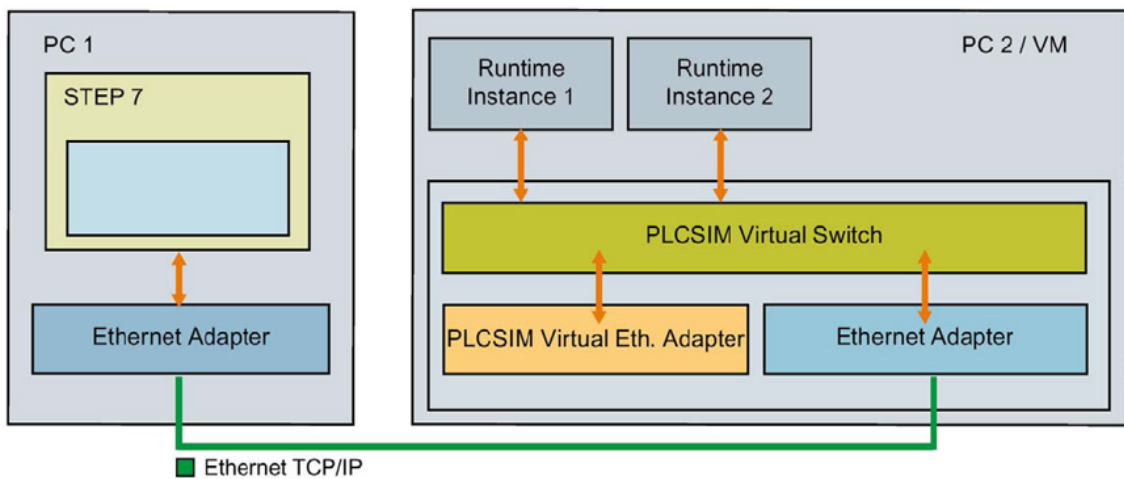


Figure 2.1: The connection between PLCSIM Advanced Runtime Instance on one PC to the STEP 7 TIA Portal on another PC[7].)

The version used in the project, PLCSIM Advanced V2.0 has the capability to emulate 2 different families of PLC CPU:s, the SIMATIC ET 200 SP series and the SIMATIC 1500 series. However the version used at Volvo for the specific EMS used in this project is a 300 series, which means that the current code can not be directly run with PLCSIM Advanced without major changes of the code, as some functions aren't cross-compatible between the two series.

Virtual time

The virtual CPU have two clocks, a virtual and a real clock. The virtual clock controls cycle times and time measurements in the PLC, while the real clock is used for communication with STEP 7 and the PCs Operating system. Virtual time scaling means the virtual clock is speed up to a 100 times faster than real time. This does not change the execution speed of the program, but change how frequently the main program is executed. If the PC is incapable of executing main before the next scan cycle, the virtual controller goes to `STOP` mode.

Freeze state of the virtual controller means that the virtual clock stops, the execution of the program stops but the virtual controller is still accessible from the TIA Portal. In the freeze mode the CPU is still in `RUN` mode, just waiting for a synchronisation point from other simulation partners.

2.3.4 TIA Portal Openness

TIAPortalOpennessDemo is an open source application for windows that connects to the STEP 7 TIA Portal via the TIA Openness protocol. TIAPortalOpennessDemo has a user interface that allows the user to manually import blocks PLC code from an XML file into the project in the TIA Portal, or export PLC code blocks to XML files [8].

2.4 Open platform communication

OPC is an industry standard for communicating between software applications and industrial control hardware. This allows production systems modeled in DES applications such as Tecnomatix Plant Simulation to be controlled by a PLC controller. By emulating the control and communication behavior of the physical system the simulation model can be used for virtual commissioning of the PLC code.

The OPC interface between the PLC and simulation model can affect the behavior and accuracy of the simulation due to Free-wheeling and synchronization issues [9]. However, a previous master thesis by Engström and Liao [10] has shown that a model built in the DES software Siemens Plant Simulation software can be controlled by a PLC through an OPC protocol without any negative effects on accuracy, given that the simulation is run in real-time. PLCSIM Advanced has a direct interface with Plant Simulation that allows faster-than-real-time simulation without synchronization issues.

2.5 Electric Monorail System

An electric monorail system (EMS) is transport system with rail going independently controlled vehicles, also referred to as hangers. Switch points are used to allow the vehicles to change lines. The track is either mounted hanging from the ceiling or a steel structure leading to an unobstructed factory floor [11]. EMS are commonly used in automotive industry to carry parts such as chassis and doors between assembly stations.

2.5.1 Volvo Cars EMS

The monorail system at Volvo is used for transporting the car sides from the body side line to the framing line where it is joined to the floor of the car. The layout of the line can be seen in figure 2.2. The hangers picks up a right and a left side for either a XC90 (black track) or a V90 (green track) and then drops them off at an assembly station.

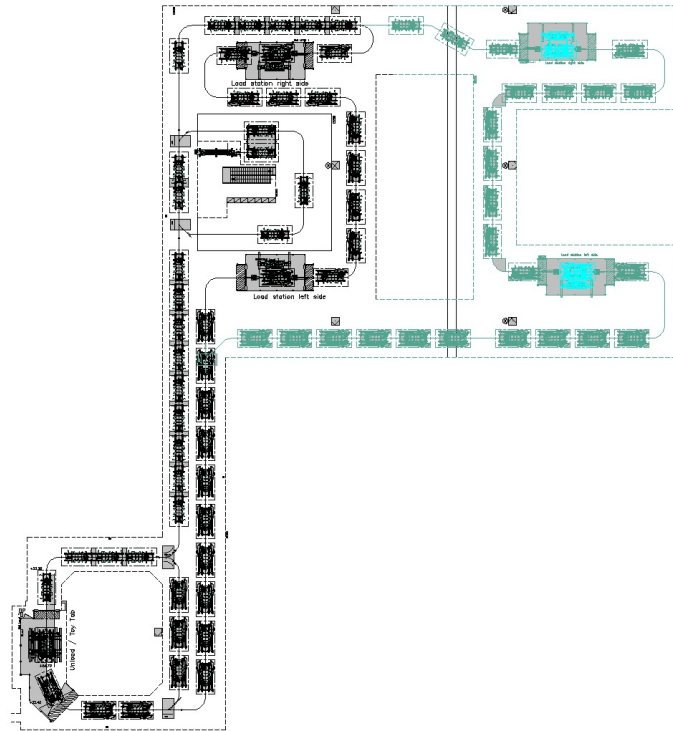


Figure 2.2: Overview of the EMS line. The assembly station is located in the lower left corner. Notice the XC90 track in black and the V90 track in green.

The monorail system and the hangers are controlled by an PLC using three control rails. These three rails are the power, presence and command rail. When the power rail is enabled by the PLC it powers on the hangers and allows them to travel. The presence rail detects any hangers on the rail and signal the presence of a hanger to the PLC. The command rail gets a speed command from the PLC that any hangers on the rail will follow.

The monorail is separated into several distinct "cuts", both figuratively and literally. Each cut has a specific function following the outlines in Volvo Cars internal standard for electric monorail systems [12]. For example: an accumulator rail (or ACC-rail) can have several hangers on it and thus works as a buffer, while a transport zone (TZ-rail) only allows one hanger at a time and is used in corners and other places where there is a risk of collision. A group of cuts are further grouped into different "safety-zones", each with their own HMI:s and safety PLC:s. This ensures that parts of the line can still function even if there is a problem in a zone.

2.6 Previous research

The idea of having a digital twin of a line or factory that evolves as the project progresses has been discussed for at least a few years, with [13] describing how such a system could work.

As virtual commissioning is becoming more and more relevant more research into what can be digitalised has been performed.

A few projects have been performed in the area virtual commissioning of PLC by, mostly using specialised simulation softwares such as Process Simulate (by Siemens PLM), Robot Studio (by ABB Robotics) or even CATIA/DELMIA (by Dassault Systèmes) [14, 15, 16].

There has also been some studies on what should be the focus in the future [17]. They conclude that virtual commissioning is an important aspect in the future of production development, but that it still has some obstacles to overcome, such as deciding which accuracy level to strive for.

The idea of creating a Simulation model based on existing PLC code has been examined for a few years [18, 19] but those models were never meant to be used for anything other than to evaluate the PLC program.

However, the topic of this report will focus on the opposite, namely creating PLC code based on a simulation model. Some research on automatic code generation has already been done, with [20, 21, 22, 23, 24] focusing on creating PLC code in several of the different languages described in section 2.3.1, although none of them have examined how DES programs could be used in such applications.

3

Methods

3.1 Connecting Plant Simulation and PLCSIM Advanced

Connecting Plant Simulation to a PLC is something that has been examined before, with a master thesis by Viktor Engström and Zhizhong Liao investigating how a connection via an OPC server works, as well as which limitations there are[10] . However, from Plant Simulation 14.0 and onwards users have the possibility of connecting directly to PLCSIM Advanced. What follows here is a brief explanation of the connection between Plant Simulation and PLCSIM_Advanced. A more comprehensive guide on the connection can be found in appendix B.

3.1.1 Creating the connection

In order to control the Plant Simulation model a connection to the virtual PLC must be made. This is achieved by inserting a *PLCSIM_Advanced* object into the model, whose dialog window can be seen in figure 3.1. The connection is created by first inputting the name of the virtual PLC into the text box marked "name" and then checking the "Active" box. A green dot will appear on the PLCSIM_Advanced object when the connection is active. Pressing the button "Import Items..." when the connection is active creates a table containing all signals and tags in the PLC program. The signals are grouped into four categories which can be seen in table 3.1 [25].

The signals in the item lists can be connected to attributes and methods in Plant Simulation by adding their path to the "Simulation Model Attribute" field at the respective row.

Table 3.1: The four categories of signals that can be imported to Plant Simulation

I	Input signals to the PLC	Write access in Plant Simulation
O	Output signals from the PLC	Read access in Plant Simulation
M	Marker Variables in the virtual PLC	Read/write access in Plant Simulation
DB	Data in PLC data blocks	Read/write access in Plant Simulation

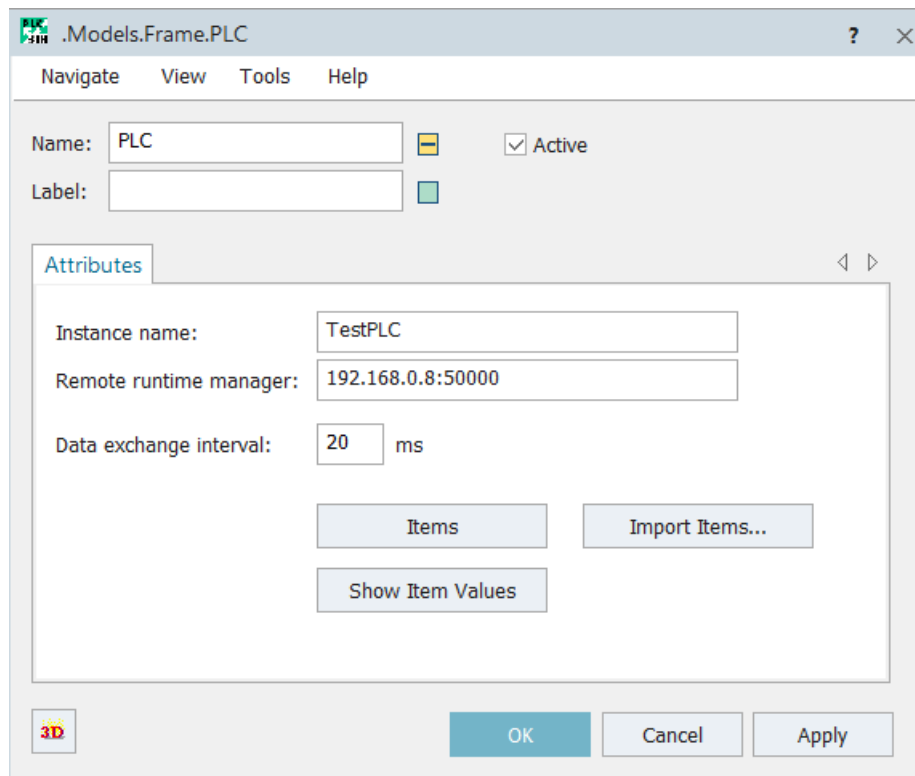


Figure 3.1: The dialog window of the PLCSIM_Advanced object in Plant Simulation

3.1.2 Sending data

Data is exchanged between Plant Simulation and the PLC at set intervals. This interval can be adjusted by changing the "data exchange interval" in the PLCSIM_Advanced object (see figure 3.1). A low number means that the data is exchanged more often, while a higher number exchanges data less frequently.

It is also possible to have a slower update rate on certain signal groups by changing the interval field in the item list. If an interval in the item list is higher than the data exchange rate then that group will update less frequently than the other signals. If the interval value is lower than the data exchange value (including interval = 0) however, the group should instead use the data exchange interval. Unfortunately, this last feature doesn't seem to work as intended in Plant Simulation 14.0. The interval for the item groups will always override the data exchange interval, making it redundant.

The PLC can receive the signals from the Plant Simulation model in different ways. A signal can for example be connected to an attribute in the simulation. In figure 3.2 the IN-signal "PlantSimToggleButton" is directly connected to the value of the checkbox (the button labelled "Start/Stop Conveyor"). When the value of the checkbox changes, the IN-signal changes with it.

OUT-signals from the PLC can be connected to the Plant Simulation model in the same way as IN-signals can. They can be directly linked to variables or attributes via the Simulation Model Attribute field in the item list.

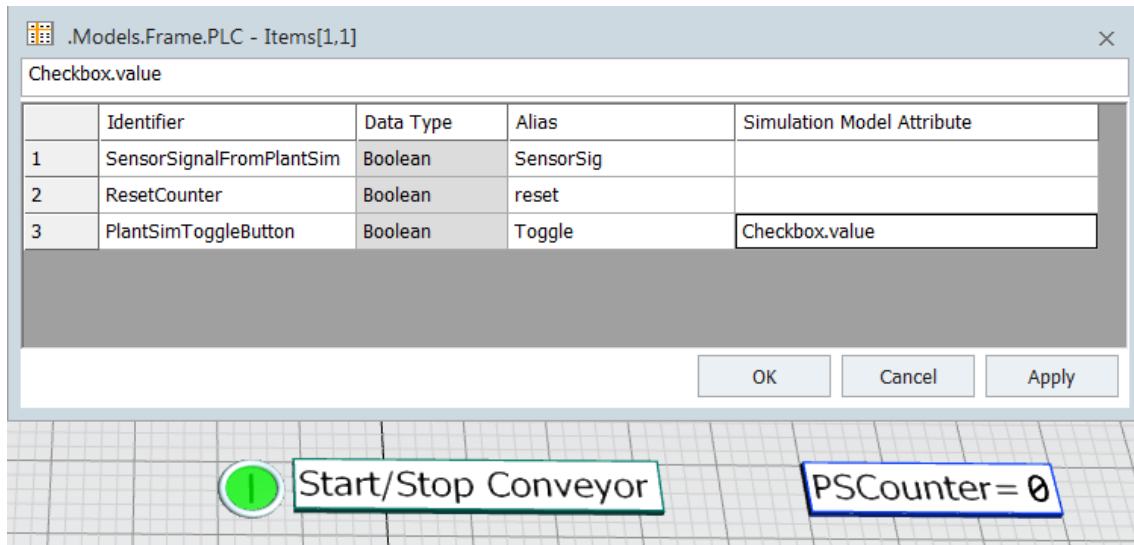


Figure 3.2: A table showing all IN-signals to the PLC as well as their aliases and connected model attributes

A difference between IN and OUT signals is that OUT signals can be connected directly to a SIMTALK method. When such an OUT-signal changes value the method is triggered with the signal value as a parameter.

Notes regarding the data exchange

One very important thing to note is that the data exchange is **not** treated as an event in Plant Simulation! This holds true for both signals sent to and from the PLC. If the exchange should happen at a time when no events or activities are executing in the simulation no data will be transferred. However, it would seem like the exchange is "put on-hold" until the next event or activity happens in the model.

3.1.3 Building the test model

In order to test and understand how the connection between Plant Simulation and PLCSIM_Advanced works a simple test model was created. The model can be seen in figure 3.3, and the PLC code for the control system can be seen in appendix C. The model contains a source connected to a buffer, which is connected to a short line (with a sensor) that in turn is connected to the drain. It also contains a PLCSIM_Advanced object that allows for direct communication with a virtual PLC.

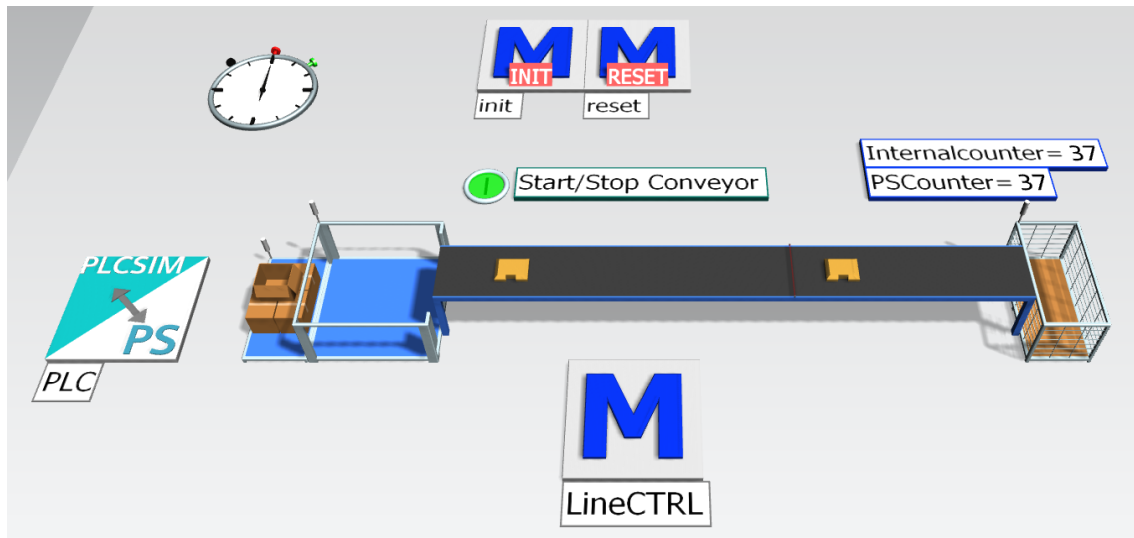


Figure 3.3: 3D view of the model used to test the connection with PLCSIM Advanced. The Connection is set up with the PLCSIM object on the left of the line

The logic of the model is rather simple: The button labelled "Start/Stop Conveyor" is connected to the tag "PlantSimToggleButton" in the PLC. When the value of the button changes the connected PLC tag changes with it, which in turn changes the value of the tag "ConveyorCTRL". This tag is connected to the attribute `Line.Stopped`, which decides if the line is moving or not. The sensor on the line sends a signal to the PLC each time a MU passes. This signal triggers a counter that then sums up how many MU:s that have passed the sensor. The sensor also triggers an internal counter that is used to compare the PLC logic to the internal logic.

3.2 Coding with a standard

Before starting the modelling of the EMS line it was decided to use a shared standard between Plant Simulation and the PLC, for a few key reasons:

- Easier to compare data in Plant Simulation and the PLC
- A Standard makes the system much more scaleable
- Easier communication between programmer and modeller

The next question was weather to use Volvo's standard or to create a custom standard. Both ways have their own benefits and drawbacks, as seen in figure 3.2

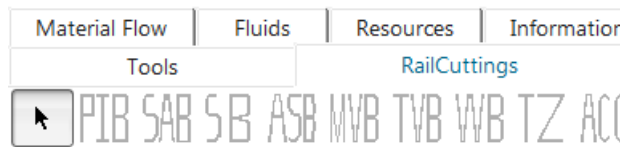
In the end it was decided to use a custom standard as it allowed for a greater degree of flexibility. The standard was to be based on Volvo's standard for EMS lines [12] with objects and signals sharing the same name in both Plant Simulation and the PLC.

Table 3.2: Benefits and drawbacks of using different standards

	Pros	Cons
Volvo's standard	Accurately represents the real system, pre-built library	Very complex, lots of superfluous functions (alarms, HMI:s)
Custom standard	Tailor made for our needs, Still based on Volvo standard, easy to adapt	Unusable outside of the project, extra work is needed to create the library

3.2.1 Plant Simulation standard

The Plant simulation model contains a user-created toolbox, figure 3.4, that contains an object for each different type of rail described by the Volvo Cars standard for EMS lines [12]. Each of these objects contain methods and attributes that are used when communicating with the PLC. Having these different types also allows for easier changes to specific types of rails, as every rail of a certain type present in the model are instances of the "blueprint" in the toolbox. Updating the "blueprint" means that all rails of that class will immediately inherit the same changes.

**Figure 3.4:** A Plant Simulation Toolbox with nine different types of rail

The rails follow a naming standard of "T_Zone_Type+instance", with the T signifying that the object is a type of track. Based on this standard it can easily be understood that the rail with the name T_420_ACC1 is then the first ACC rail in safety zone 420. The postscript `_direction` is added when a zone contains more than one in- or outgoing track. Postscript `_1` means that the track travels straight after a switch or into a junction, while the postscript `_9` is used for rails that enters or leaves from the side. So a rail with the name T_330_ASB3_3 is the third ASB rail in safety zone 330 and it turns after a switch.

3.2.2 PLC standard

The PLC contains a similar library as Plant Simulation with a function block (FB) for each type of rail, as seen in figure 3.5, which can be dragged and dropped into the PLC code. All rail function blocks follows the same I/O structure with the same type of input and output signals, shown in figure 3.6.

The meaning and function of the I/O is listed in table 3.3:

The unique behaviour of each function block listed in the library is explained further in appendix F.

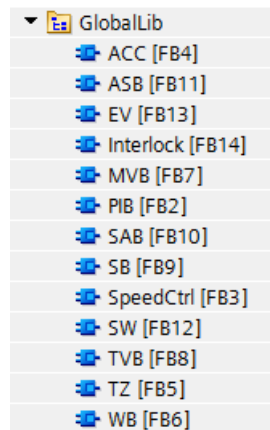


Figure 3.5: The function block library in the PLC

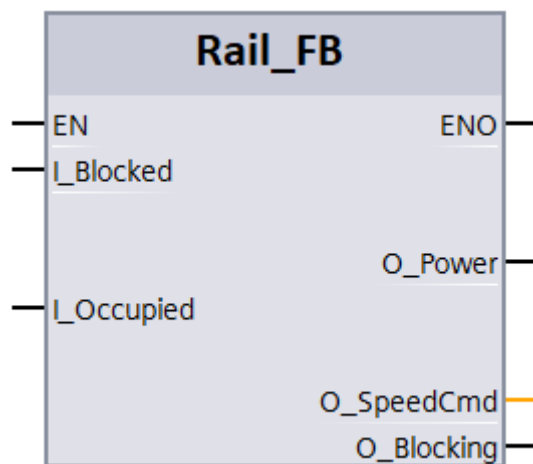


Figure 3.6: An example of a rail control Function Block, showing the blocks inputs to the left of the block, and outputs on the right side of the block.

When a new rail object is added to the Plant Simulation model, an instance of the corresponding type to the Plant Simulation object should be inserted to a new network in the PLC. The simulation object T_330_ASB3_3 is then controlled by an ASB function block, connected to the data block DB_330_ASB3_3.

The naming of the blocks follow the same standard as the names in Plant Simulation, with the exception of the T prefix. This makes it easy to see which block is connected to which track, as block 330_TZ1 in the PLC has a sibling T_330_TZ1 in the Plant Simulation model. The PLC I/O also follows the same standard meaning that the signals for block 330_TZ2 all have "330_TZ2" as a prefix. This in turn means that the only difference between the network 330_TZ2 and 330_TZ3 is the prefix of the signals and the data block as seen in figure 3.7.

Table 3.3: The function of the rail control Function Blocks I/O signals

I/O	Function
I_Blocked	Blocking signal from the upstream rail. If high, stops all hangers on the current rail to avoid collisions.
I_Occupied	Input from the presence rail to the PLC that a hanger is currently somewhere on the rail cut.
O_Power	Enables power on the rail, allowing the hangers to travel.
O_SpeedCmd	Sets a speed command in the form of an integer to all hangers present on the rail.
O_Blocking	Sends a blocking signal to the downstream rail if the current rail is occupied.

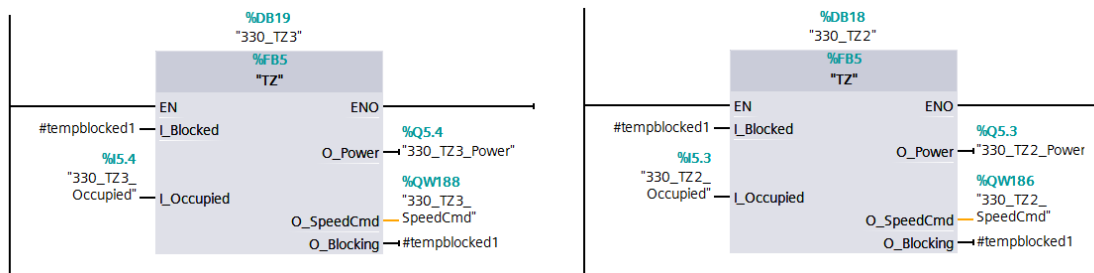


Figure 3.7: A graphical representation of the differences between two instances of the same rail type.

3.2.3 PLC hierarchy

The PLC program is built in layers in order to make it as scalable as possible. The topmost layer is the "Main" block which controls the system by communicating with the second layer. This layer is composed of the different safety zones explained in section 2.5.1, with each zone represented by a separate block. These blocks, in turn, contain instances of the blocks representing the different types of rails found in the zone. This hierarchy is graphically explained in figure 3.8. Signals from the PLC are only set in the lowest layer of the hierarchy.

3.3 Modelling the EMS

The modelling of the line in Plant Simulation started by creating the layout of the EMS using regular line object. The measurements of the EMS line were taken from the CAD files of the site.

The line objects making up the EMS were then systematically replaced with the appropriate pieces from the user-created toolbox (figure 3.4) with the names following the standard set in section 3.2.1. The sequence of rails were given by a chart over the busses connected to the EMS, which did not contain any measurements. Therefore the total length of the EMS remains correct, while the individual rail segments are inaccurate. This problem is further explained in section 4.2.1.

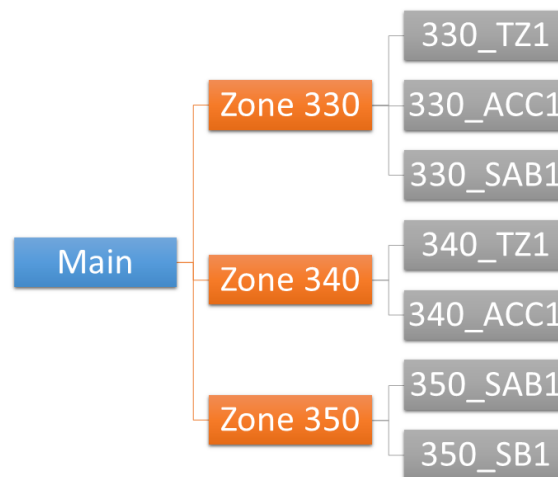


Figure 3.8: A graphical representation of the hierarchy of the PLC program. The main block can not communicate directly with the function blocks. However, blocks on the same level depend on information from each other.

After two safety zones (320 and 330) were done in both the model and the PLC code the model was test-run with the PLC. In order to make sure that model and the PLC had the same signals an Excel spreadsheet was created that contained the correct signal names for both applications. The spreadsheet could then be imported into the TIA Portal or the item lists in the PLCSIM_Advanced object.

After modelling and testing another safety zone it was decided to investigate if the PLC code could in some way be automatically generated based on the Plant Simulation model, as described in section 3.4.

The rest of the model was constructed as before, while the PLC code were auto-generated zone by zone as they were completed. A part of the completed model can be seen in figure 3.9.

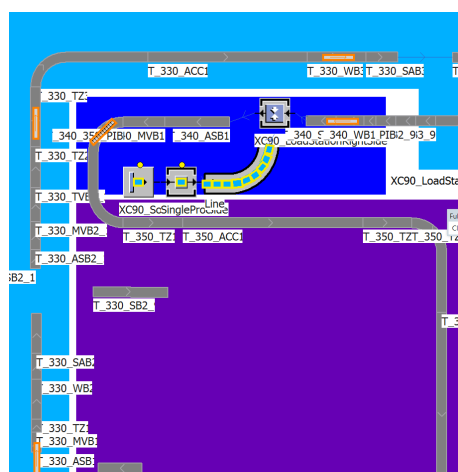


Figure 3.9: 2D view of a part of the Pant Simulation Model. The coloured backgrounds represents different safety zones

3.4 Auto-generation of PLC code

As the PLC ladder code for every safety zone follows the same structure there is a lot of repetitive code, and a lot of it can be copy-pasted. However if copy-pasting is used, the order of the rails still differs from zone to zone, and needs to be rearranged in the new code. Also, I/O signals still has to be rewritten and new data blocks for every new rail function block needs to be created. This is an error prone process, as it is easy to misspell an I/O variable, especially as they only differs to each other with an integer. It was found that using a script to create the PLC connections would be a quicker and safer process. However such a script is lacking in the TIA Portal, PLC code was exported to XML and edited using a C# Program.

In order to automatically generate usable PLC code four steps had to be taken:

1. Create XML templates for the PLC blocks

A template for every type of rail, along with a matching data block, was first created by hand in the TIA Portal, an example can be seen in figure 3.10. These are identical to functional code but have their names and variables changed to generic ones that can be found and replaced by a macro. The templates was then exported to XML format using the TIAPortalOpennessDemo application introduced in section 2.3.4. The XML code can be seen in appendix E.

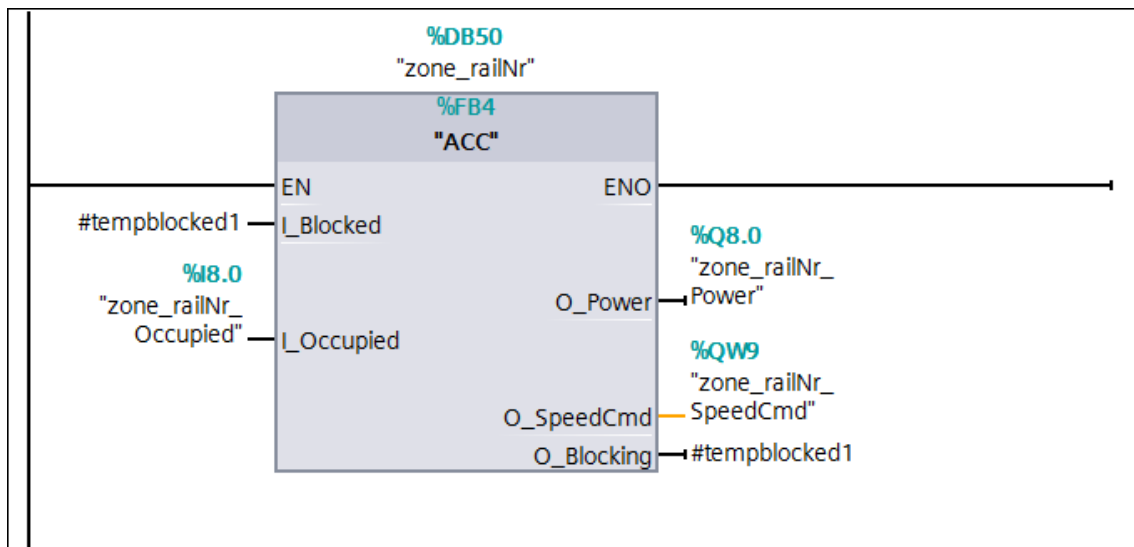


Figure 3.10: The PLC template for an ACC network with generic signals and variables at the connections instead of instance specific ones.

2. Create relevant data-set in Plant Simulation

Since the code is supposed to be generated based on the Plant Simulation model, relevant data had to be extracted. The code shown in appendix D was used to create a text file for each safety zone containing the name of all rails in reverse sequential order (since that is how the PLC program is built). Since the PLC code and Plant Simulation uses the same naming standard, this is all information that is needed to map all variables and I/O for the rail control logic.

3. Create a program to edit the PLC templates

A C# program referred to as PLCBuilder was written that creates PLC or updates PLC code for a safety zone with rail data from the zone exported directly from the DES model in Plant Simulation.

4. Autogenerate PLC code with PLCBuilder

The first step to autogenerate PLC code for a safety zone is to first check if there already exists PLC code for the zone in the TIA Portal. If it does, the code should be exported to XML format using TIAPortalOpennessDemo to the directory of PLCBuilder. Next step is to update the rail list input file to PLCBuilder. Then when PLCBuilder is executed, it reads the rail list, and checks if the rail already exists in the XML file for the existing PLC code. If not, it appends the XML template of the listed rail type as seen figure 3.11, and renames the generic variables according to the name in the list. An XML file for a Data block corresponding to the Function Block generated is also created. If a rail object exists in the old

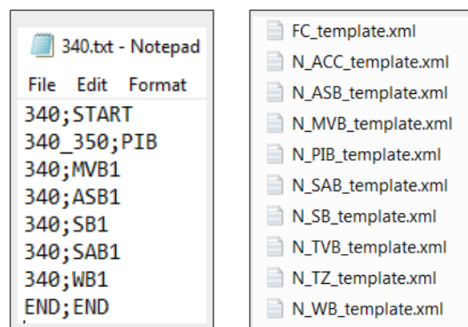


Figure 3.11: A text-file a list of all rails in safety zone 340, and a list of all XML templates used for autogenerating a new PLC code.

PLC code but not in the list, it will delete the entire XML node for the rail cut. When PLCBuilder has gone through the entire list and checked that it matches with the XML file, the program is finished. The new autogenerated XML file can then be imported into the PLC program and converted to functional ladder code through TIAPortalOpennessDemo and then downloaded to the virtual PLC through the TIA Portal. Examples of XML code and the result of the import can be found in appendix E.

4

Results

Some problems were found stemming both from limitations in the software and from the early state of some the functions being used. The software needs to mature before the work method proposed by this project can prove beneficial for industry. However, the connection between Plant Simulation and PLCSIM Advanced still works despite these problems and it was possible to control the whole line.

4.1 Hardware differences between PLC:s

A problem can occur when using Plant Simulation as a virtual commissioning tool for existing production lines, or when using pre-existing PLC-libraries. As explained in section 2.3.3, PLCSIM Advanced can only emulate PLC:s with 1500- and 200SP series CPU:s. If the library exists on an older hardware, *e.g.* a 300 series CPU, there is a risk that parts of the library will be unusable, as functions might have been dropped or changed.

4.2 Lack of detail in Plant Simulation

The types of objects that can be modelled in Plant Simulation can be rather limited, especially compared to more specialised virtual commissioning tools such as Process Simulate. The problem was also observed by [10] who claimed that not all objects found in a production cell could be accurately represented. However, this should prove to be a minimal problem as long as the Plant Simulation model focuses on systems at line-level or higher.

4.2.1 Measurement problems in EMS

A problem more specific to the Plant Simulation model created during the project is that the measurements of the rails are assumed. A blueprint of the EMS line were provided by Volvo, which gave the overall lengths of the line. However, no blueprint could be found giving the lengths of individual rail cuts. This means that the total length of a piece of track is known, but not where the physical cuts are located. This could prove troublesome as the model is no longer an accurate representation of the real system, especially when times are important. But as this model is made to test the connection between Plant Simulation and the PLC it did not affect the result in any significant way.

4.3 Slowdowns when running Plant Simulation with a virtual PLC

One apparent problem were found while running the EMS line with a virtual PLC: more signals resulted in a slower simulation speed. More signals means that the simulation model has to handle more data during cycles since Plant Simulation exchanges data with the PLC at set intervals. The phenomenon is explained in greater detail in section B.3.

The problems with the slowdowns can temporarily be solved by either increasing the time-scaling factor up the virtual PLC or by changing the data exchange frequency so that Plant Simulation updates less frequently. Of the two options the first one is preferable, even though none of them are optimal.

Speeding up the PLC means that the data exchange is completed quicker. This also means that the model can be simulated faster than real-time. However, if the PLC is scaled faster or slower then the model it means that the PLC doesn't work in the same way as it would in a physical environment which invalidates the idea behind virtual commissioning.

Reducing the update frequency of the signals poses the same risk of misrepresenting the real system as time-scaling the PLC. Moreover there is a risk that important signals are either missed or sent to the PLC too late. This can pose a problem when sending pulses or alarm signals. There also exist a problem wherein the update interval of the item groups always override the data exchange interval of the PLCSIM_Advanced object. This problem is further explained in appendix B.3.

4.4 Connection between Plant Simulation and TIA Portal

There are currently no way of connecting or transferring data between Plant Simulation and the TIA Portal. The methods used in this project has either been to transfer the signal list via Excel or by using TIAPortalOpennessDemo and the C# program presented in section 3.4.

4.5 Creating the PLC code

It was quickly discovered during the modelling of the EMS line that the creation of the PLC code often led to errors, mostly in the form of spelling mistakes or miscouplings in the PLC code. Some of the problems stemmed from bad communication between the modeller and the programmer, while some problems stemmed from the homogeneous nature of the standard, *e.g.* misreading and working on block T_330_SB2_9 instead of T_330_SB1_9. In this case copy-pasting code sections are one way of reducing the amount of repetitive coding but this further increases the chance of mislabelling variables signals in the code.

Another obvious problem is that the time needed for writing the code increases as the model grows and the function blocks becomes more complex. This also leads to a greater chance of errors or forgotten signals. The solution to these problems is to auto generate the PLC code based on the previously made code and information from the Plant Simulation model. The details is explained earlier in section 3.4.

4.6 Lack of documentation

Finally, one of the problems found is the absence of documentation regarding the communication between Plant Simulation to PLCSIM Advanced. Much of the information on *how* the connection works are either wrong, (such as how a signal groups should choose the *higher* of the group interval or the exchange interval while they in reality always chooses the group specific) or missing (such as the data exchange not being classified as an event in Plant Simulation). The lack of documentation also means that it's hard to fix problems such as the error in figure 4.1, as it doesn't explain where or why the problem occurs.

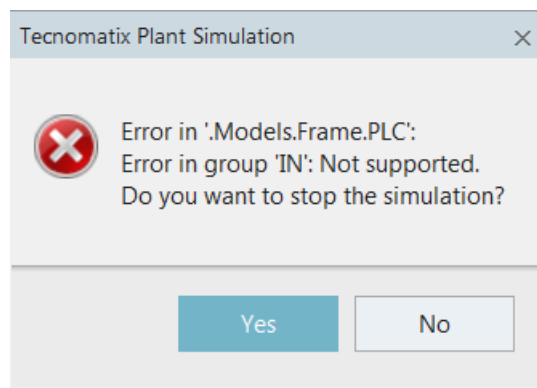


Figure 4.1: An error prompt in Plant Simulation. There are no available documentation on why this error occurs or how to fix the problem.

5

Conclusion

All in all it can safely be said that Plant Simulation can be used as a tool for virtual commissioning of PLC code, but the software's needs to be updated and engineered for the new tasks first. What follows is a summary of the parts that needs to be improved and what kind of knowledge that is needed for this type of work. The research questions introduced in section 1.4 will be discussed in the following sections. The questions RQ1 through RQ3 will be discussed in sections 5.2 through 5.4 respectively.

5.1 Improvement potential in the software

The project has shown that Plant simulation can be used as a tool for virtual commissioning, at least in small to medium sized projects. There are however a few aspects that have to be improved or further investigated before this can be adapted into the engineers toolbox:

- The problem with the data exchange interval explained in sections 4.3 and B.3 has to be fixed.
- Pressing the "import item" button should update the existing item groups rather than creating new groups.
 - The problem with creating new groups is that the simulation model attribute doesn't carry over between from the old groups with the same name, so they will have to be added again.
- It should be possible to share, or at least connect, libraries in Plant Simulation and the TIA Portal.
 - When adding an object into the Plant Simulation model it should be as easy as possible to add the respective function block to the PLC.
- It should be possible to create and connect variables and I/Os to function blocks using macros within the TIA Portal.
 - The current method of using TIAPortalOpennessDemo also requires understanding of both XML and another programming environment (such as C#) to create an application that auto generates new PLC code.

5.2 Limitations of the connection between Plant Simulation and PLC

In the current state the modeller and/or programmer must possess certain skills in order to do utilise Plant Simulation in the context of PLC design:

- Knowledge about the system
 - The discrete event model should be modelled as accurately as possible to the real system, which means that the modeller needs to have access to blueprints and measurements
- Knowledge about Discrete event simulation
 - The modeller must understand how to create a functioning DES model, as well as how to emulate objects in the real system not available in the DES software
- Knowledge about PLC programming
 - The Programmer needs knowledge about PLC programming to write the code that runs the simulation model. Even if an auto-generated approach is chosen such as the one in this project, an understanding of PLC programming is necessary to design the structure and the templates to be used by the PLC code generator.
- Knowledge about other programming
 - Writing the I/O list is too time consuming in both Plant Simulation and the PLC for larger projects and writing a macro that auto-generates the I/O list becomes almost necessary. Writing an application that generates PLC code in XML format is both requires understanding of XML code and a programming environment able to edit XML files, such as C#.

5.3 The extent to which PLC code can be designed based on a Plant Simulation model

- It is possible to generate a functional draft of PLC code for an EMS line to be used for further development by using the method explained in this project.
 - similarly, higher level of industrial robot control should easily be programmed using the same method, with only consideration to start and end positions and zone booking *e.g.* no kinematics or collision detection.
- The level of detail of Plant Simulation is insufficient to model detailed station behaviour and hence the method is only suitable for higher level of control of production systems.

5.4 Future research topics

One of the major drawbacks of using the method proposed in this report is that the Plant Simulation user has to do double work. There is currently no possible way of translating SIMTALK into ladder or vice versa. This means that two different models will be built, one for each way of control. A proposal for future research would be to either create a macro that can switch a model from using any of the two languages. Another similar topic would be to evaluate whereat or not a Plant Simulation model can be coded using ladder logic.

Another drawback of controlling a DES model with a PLC is that you can't simulate very quickly. The maximum possible speed achieved with PLCSIM Advanced is a time scaling factor of 100. This relatively slow reading speed makes such a model unfit for "regular" simulation uses such as capacity analysis, as simulating a multiple runs of full years of productions is not uncommon to achieve a higher precision. Even shorter experiments would take an unreasonable long time to complete. Future research could focus on creating virtual PLC:s that can achieve much higher reading speeds.

Another future application would be to connect Plant Simulation directly to the TIA Portal. The two programs should share a library, so that when an object is inserted into Plant Simulation a corresponding block is inserted into a network in the TIA Portal.

5.5 Closing statements

The project has shown that Plant Simulation can be used for PLC design in the future, but it still requires a lot of polish. Furthermore, Plant Simulation should focus more on testing the flow control of larger systems with a low-complexity control system while real-time simulation software such as Process Simulate are better suited for smaller or more signal-heavy applications.

Bibliography

- [1] George S. Fishman. *Discrete-Event Simulation : Modeling, Programming, and Analysis*. 2001, p. 537. ISBN: 9781475735529. DOI: 10.1111/1467-9884.00369{_}9. URL: <https://link.springer.com/content/pdf/10.1007%2F978-1-4757-3552-9.pdf>.
- [2] Ricki G. Ingalls. “Introduction to simulation”. In: *Proceedings of the 2008 Winter Simulation Conference*. Ed. by S. J. Mason et al. 2008, pp. 17–26. ISBN: 9780874216561. DOI: 10.1007/s13398-014-0173-7.2. arXiv: 9809069v1 [gr-qc]. URL: <https://www.informs-sim.org/wsc08papers/005.pdf%20http://portal.acm.org/citation.cfm?id=256563.256571>.
- [3] Steffen Bangsow. “Basics”. In: *Tecnomatix Plant Simulation*. Cham: Springer International Publishing, 2016. Chap. 1, pp. 1–15. DOI: 10.1007/978-3-319-19503-2{_}1. URL: http://link.springer.com/10.1007/978-3-319-19503-2_1.
- [4] Ephrem Ryan Alphonsus and Mohammad Omar Abdullah. *A review on the applications of programmable logic controllers (PLCs)*. 2016. DOI: 10.1016/j.rser.2016.01.025. URL: https://ac.els-cdn.com/S1364032116000551/1-s2.0-S1364032116000551-main.pdf?_tid=746c97bd-f699-4fdb-9371-bcca2552ef1f&acdnat=1526627328_6acc9fdce366942564d73cf26dcf0bd7.
- [5] William Bolton and William Bolton. “Chapter 1 – Programmable Logic Controllers”. In: *Programmable Logic Controllers*. 2015, pp. 1–22. ISBN: 9780128029299. DOI: 10.1016/B978-0-12-802929-9.00001-7. URL: https://ac.els-cdn.com/B9780128029299000017/3-s2.0-B9780128029299000017-main.pdf?_tid=0cd4a36e-02d9-4d6c-a4b5-b24e5033327e&acdnat=1526627192_13d095ba4ac834cfe56ea2b9e7218f52.
- [6] PLCOpen. *Introduction into IEC 61131-3 Programming Languages*. URL: http://www.plcopen.org/pages/tc1_standards/iec_61131_3/.
- [7] Siemens AG. *S7-PLCSIM Advanced Function Manual*. Nürnberg, 2017.
- [8] Siemens. *TIA Portal Openness : Introduction and Demo Application*. 2016. URL: <https://support.industry.siemens.com/cs/document/108716692/tia-portal-openness%3A-introduction-and-demo-application?dti=0&lc=en-WW>.
- [9] Henrik Carlsson et al. “Methods for reliable simulation-based PLC code verification”. In: *IEEE Transactions on Industrial Informatics* 8.2 (2012), pp. 267–278. ISSN: 15513203. DOI: 10.1109/TII.2011.2182653.
- [10] Viktor Engström and Zhizhong Liao. “PLC Integrated Discrete Event Simulation for Production Systems”. PhD thesis. Chalmers University of Technol-

- ogy, 2017, p. 87. URL: <http://publications.lib.chalmers.se/records/fulltext/251696/251696.pdf>.
- [11] EISENMANN Anlagenbau GmbH & Co. KG. *This is EISENMANN*. 2010. URL: http://www.eisenmann.com/dam/jcr:97d1f633-20a5-4054-bf34-3a466adc6cc6/EHB_2010_en.pdf.
- [12] Volvo Cars Gent. *Standard for Electro Monorail Systems*. 2001.
- [13] Mathias Oppelt and Leon Urbas. “Integrated virtual commissioning an essential activity in the automation engineering process: From virtual commissioning to simulation supported engineering”. In: *IECON Proceedings (Industrial Electronics Conference)*. IEEE, Oct. 2014, pp. 2564–2570. ISBN: 9781479940325. DOI: 10.1109/IECON.2014.7048867. URL: <http://ieeexplore.ieee.org/document/7048867/>.
- [14] Jesper Halmsjö and Jonas Fält. “Emulation of a production cell Developing a Virtual Commissioning model in a concurrent environment”. PhD thesis. Chalmers University of Technology, 2016, p. 92. URL: <http://publications.lib.chalmers.se/records/fulltext/241210/241210.pdf>.
- [15] Sara Winther. “Virtual commissioning of production process Final report”. PhD thesis. Chalmers University of Technology, 2017, p. 44. URL: <http://publications.lib.chalmers.se/records/fulltext/250446/250446.pdf>.
- [16] Luis Villagómez Guerrero, Virgilio Vásquez López, and Julián Echeverry Mejía. “Virtual Commissioning with Process Simulation (Tecnomatix)”. In: *Computer-Aided Design and Applications* (2014). ISSN: 16864360. DOI: 10.1080/16864360.2014.914400.
- [17] Chi G. Lee and Sang C. Park. “Survey on the virtual commissioning of manufacturing systems”. In: *Journal of Computational Design and Engineering* 1.3 (2014), pp. 213–222. ISSN: 22884300. DOI: 10.7315/JCDE.2014.021. URL: <http://linkinghub.elsevier.com/retrieve/pii/S2288430014500292>.
- [18] Hyeong Tae Park et al. “Plant model generation for PLC simulation”. In: *International Journal of Production Research* 48.5 (Mar. 2010), pp. 1517–1529. ISSN: 00207543. DOI: 10.1080/00207540802577961. URL: <http://www.tandfonline.com/doi/abs/10.1080/00207540802577961>.
- [19] Sang C. Park, Minsuk Ko, and Minho Chang. “A reverse engineering approach to generate a virtual plant model for PLC simulation”. In: *The International Journal of Advanced Manufacturing Technology* 69.9-12 (Dec. 2013), pp. 2459–2469. ISSN: 0268-3768. DOI: 10.1007/s00170-013-5209-1. URL: <http://link.springer.com/10.1007/s00170-013-5209-1>.
- [20] Petter Falkman, Erik Helander, and Mikael Andersson. “Automatic generation: A way of ensuring PLC and HMI standards”. In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*. 2011. ISBN: 9781457700187. DOI: 10.1109/ETFA.2011.6059201.
- [21] Martin Dahl et al. “Integrated Virtual Preparation and Commissioning: supporting formal methods during automation systems development”. In: *IFAC-PapersOnLine* 49.12 (2016), pp. 1939–1944. ISSN: 24058963. DOI: 10.1016/j.ifacol.2016.07.914. URL: www.sciencedirect.com.
- [22] Mulman Budha et al. “Generation of PLC ladder diagram using modular structure”. In: *2008 International Conference on Computational Intelligence for*

- Modelling Control and Automation, CIMCA 2008*. 2008. ISBN: 9780769535142. DOI: 10.1109/CIMCA.2008.125.
- [23] Sebastian Süß, Anton Strahilov, and Christian Diedrich. “Behaviour simulation for Virtual Commissioning using co-simulation”. In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*. 2015. ISBN: 9781467379298. DOI: 10.1109/ETFA.2015.7301427.
- [24] Dániel Darvas, Enrique Blanco Viñuela, and István Majzik. “PLC Code Generation Based on a Formal Specification Language”. In: (2016).
- [25] Georg Piepenbrock. “Virtual Commissioning in Plant Simulation utilizing the new SIMATIC S7-PLCSIM Advanced Interface and OPC UA”. In: *2017 Plant Simulation Worldwide User Conference*. Stuttgart, Germany, 2017, p. 13.

A

Connecting PLCSIM Advanced to the TIA Portal

A.1 Setting up the Virtual PLC

For running PLCSIM Advanced Instance and the PlantSimulation model on different Host and client PCs you must first select **PLCSIM Virtual Eth. Adapter** in the Online Access settings. **TCP/IP Communication** should be set to **Local Area Connection** as in figure A.1.

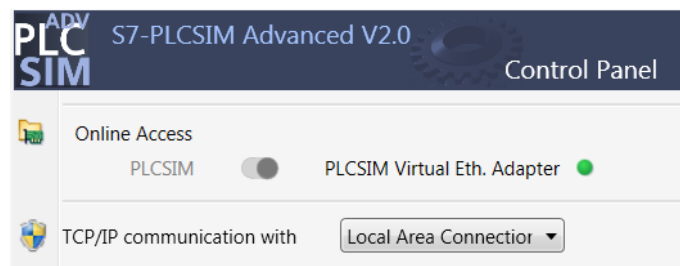


Figure A.1: The PLCSIM Advanced Interface for Online Access and TCP/IP communication.

In the Start PLC tab a unique instance name should be assigned for each virtual PLC instance. If an existing Instance name is chosen, the virtual PLC will load the old PLC:s files from the Virtual SIMATIC Memory Card. The IP address should preferably be set to IP address 192.168.1.1 as the virtual PLC instance will automatically be reset to the IP address 192.168.0.1 when the TIA portal connects to the virtual controller. The sub-net mask should be set to 255.255.255.0. as shown in figure A.2.

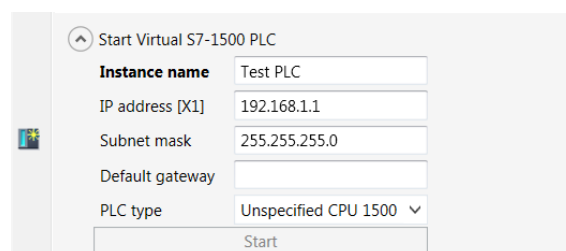


Figure A.2: The Start tab in PLCSIM Advanced showing the an example of instance name and the correct IP address and Sub-net mask.

A.1.1 Using the Virtual Ethernet Adapter

The Runtime manager port is used to identify the PLC from the client and should be set to port 50000 and enabled with the tick box next to it as in figureA.3.

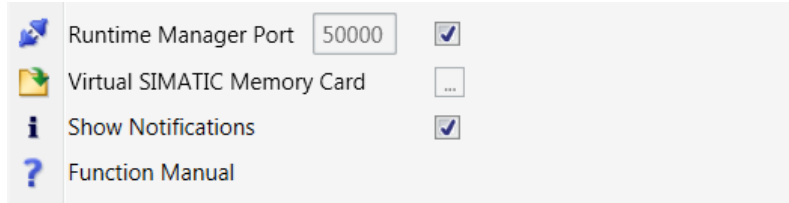


Figure A.3: The PLCSIM Advanced Interface.

Checking the runtime manager port starts the Siemens PLCSIM Virtual Ethernet Adapter and installs Siemens PLCSIM Virtual switch program to the the Ethernet adapter of the host PC. The Virtual Switch enables data exchange between the physical and virtual Ethernet adapters. After the installation, the a new Local Area Connection should show up for the Virtual Ethernet Adapter in the Network Connections folder in the Windows Control Panel, and the Virtual Switch should show up under the used items list in the physical Ethernet adapters properties as in figure A.4.

A.2 Configuring TIA for simulation

Check Support simulation during block compilation must be enabled in the Protection tab in the projects properties menu as in figure A.5 and A.6. This is compiles the PLC code in a format that is able to be downloaded and run on the virtual PLC.

A.3 Connecting and downloading to the virtual PLC

First scan for devices according to A.7 or pressing download to device. Then in **Extended download to device window** follow the following steps:

1. Select Siemens PLCSIM Virtual Ethernet Adapter in PG/PC Interface if it's not automatically selected.
2. connection to interface/subnet: try all interfaces
3. Press **Start Search**
4. select the device at address 192.168.1.1 with the same name as the projects PLC, not the one with the .profinet interface extension to the name in the list of target device
5. press **Load** in the extended download to device window.
6. Press yes in a command prompt asking the user to assign a additional IP Address.

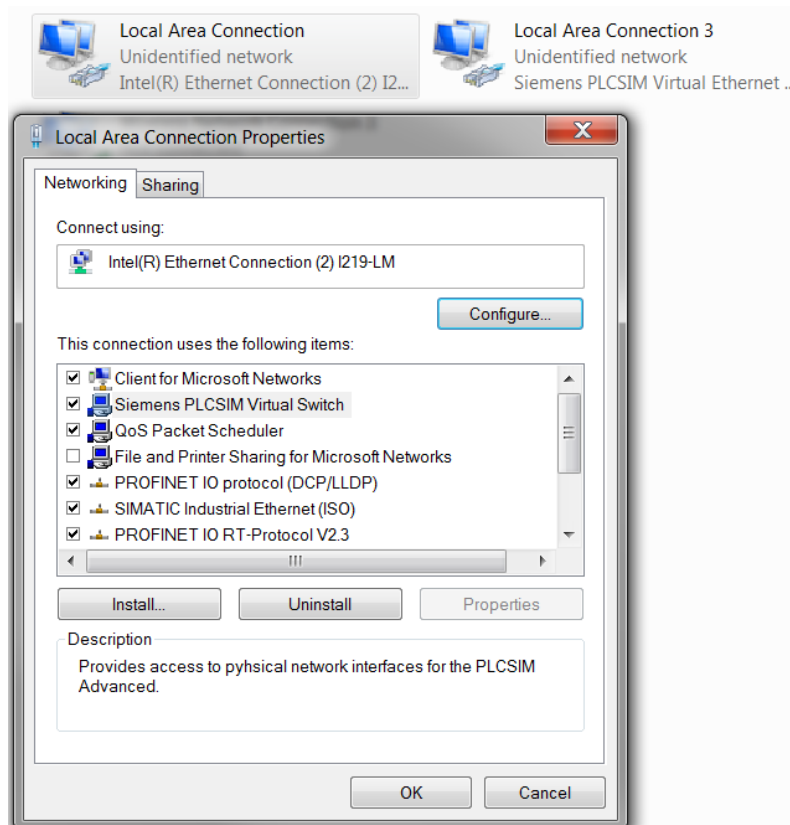


Figure A.4: Showing the Siemens PLCSIM Virtual switch protocol installed on the physical Ethernet adapter Inter(R) Ethernet Connection and the Siemens PLCSIM Virtual Ethernet Adapter in the Network Connections list in the Windows Control Panel.

7. An IP Address on the same subnet as the device is added.
8. Press extended download to device again.
9. Select and load to the device at address 192.168.1.1.
10. the Download Preview window opens.
11. Press Load in the Download Preview window
12. Press finish when the PLC is fully loaded and the program is successfully loaded to the virtual PLC.

A.4 Converting 300 code to 1500

issue with converting the existing library to the virtual PLC due to different variants most of the code can be converted but standard function blocks in one PLC demand different inputs and outputs than the same FB in the other PLC several function blocks from the old library was unusable after they had all been imported and had to be imported again, after which point they worked.

A. Connecting PLCSIM Advanced to the TIA Portal

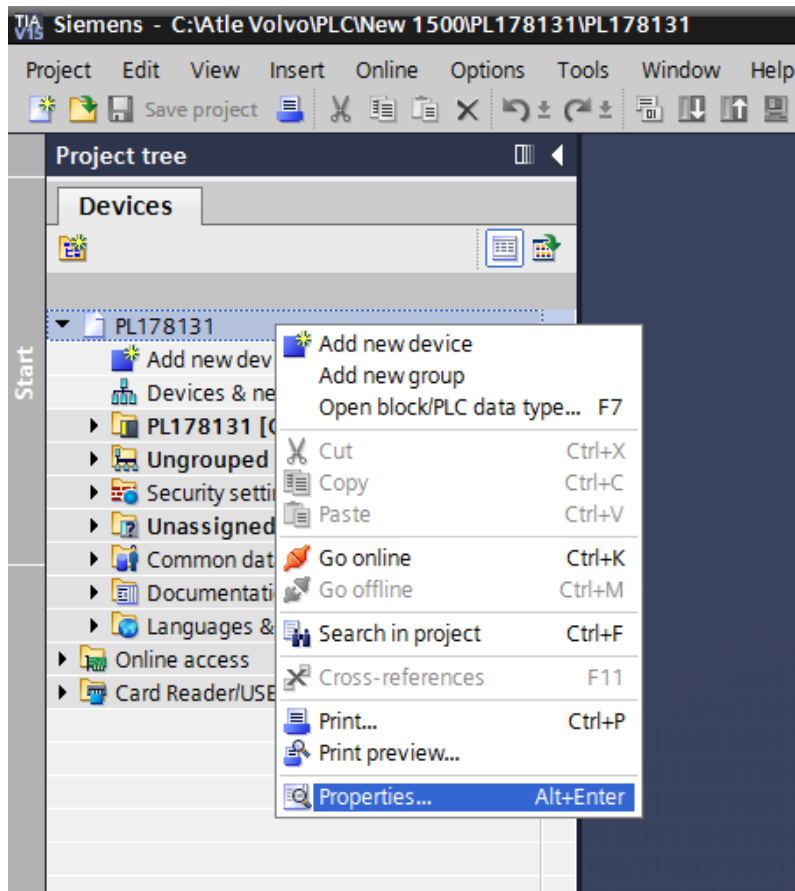


Figure A.5: Showing where to find the project properties menu.

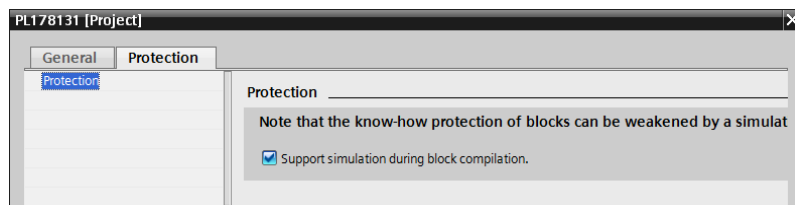


Figure A.6: Showing where to check Support simulating during block compilation in the project properties menu.

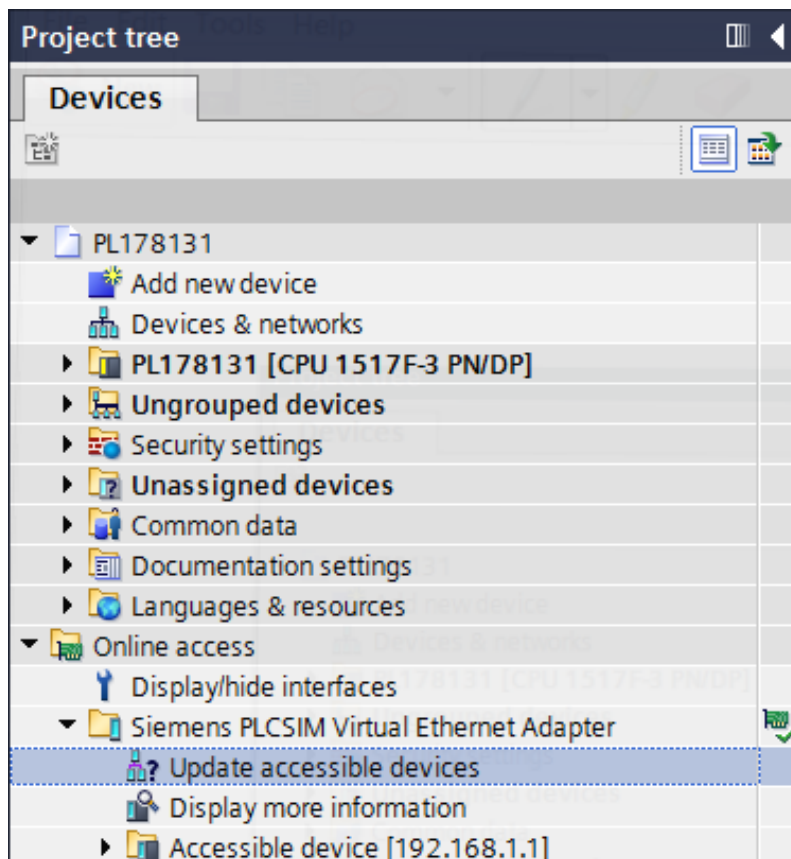


Figure A.7: Showing how to search for devices in the TIA Portal.

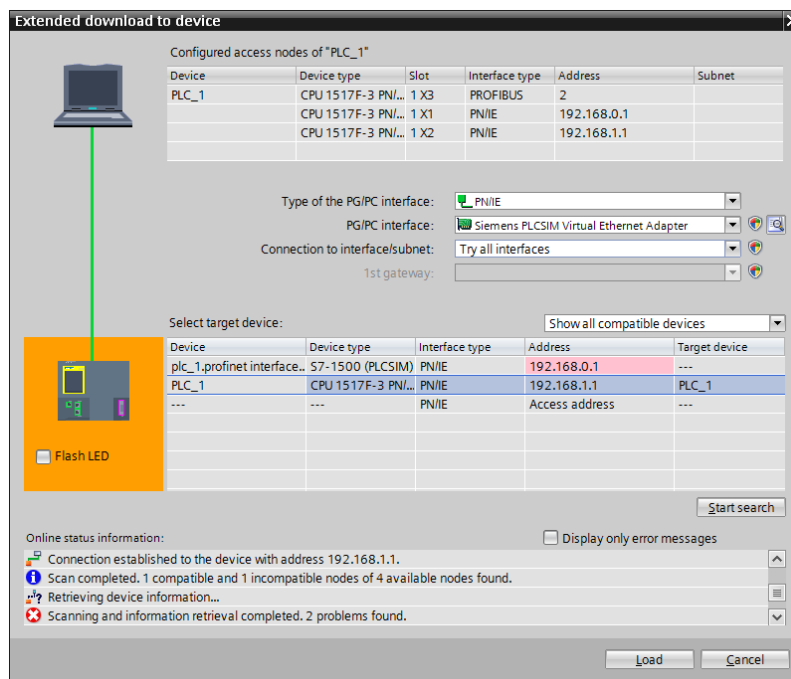


Figure A.8: Showing the Extended download to device window in the TIA Portal.

B

Connecting Plant Simulation to a virtual PLC

B.1 Setting up the connection

When connecting a virtual PLC to Plant Simulation some preparations have to be made. First of all you must make sure that the PLCSIM_Advanced package is included in the model. This is achieved by pressing the button named "Manage Class Library" under the "Home" ribbon. In the "Basic Objects" tab, scroll down to the line with PLCSIM_Advanced and tick the checkbox, as shown in figure B.1.

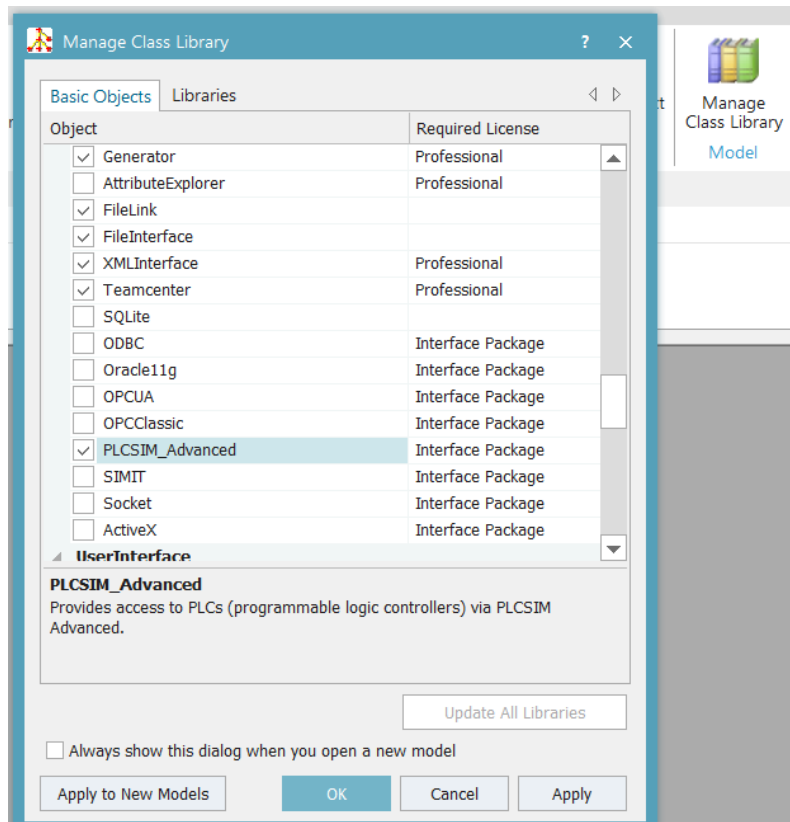


Figure B.1: The Manage Class Library screen. Activated objects have a ticked checkbox

After clicking the OK button the object should appear in the toolbox under the "Information Flow" tab, like in figure B.2.

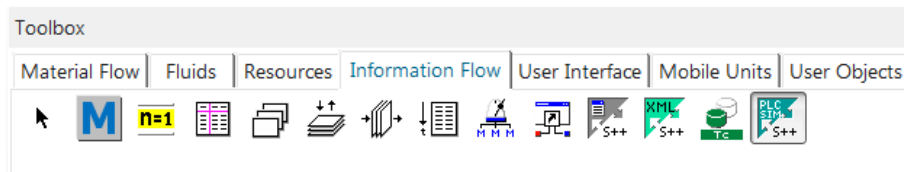


Figure B.2: The toolbar in Plant Simulation with the PLCSIM_Advanced object selected

The PLCSIM_Advanced object is then inserted into the model in the same way as any other entity. The connection to a local virtual PLC is made by opening the PLCSIM_Advanced block and inserting the name of the virtual PLC into the field named "Instance Name" and then ticking the box named "Active". A green dot on the PLCSIM_Advanced object shows that the connection is active. Figure B.3 shows how the connection is made. Refer to section B.2 for a guide on how to connect Plant Simulation to a virtual PLC on a remote desktop.

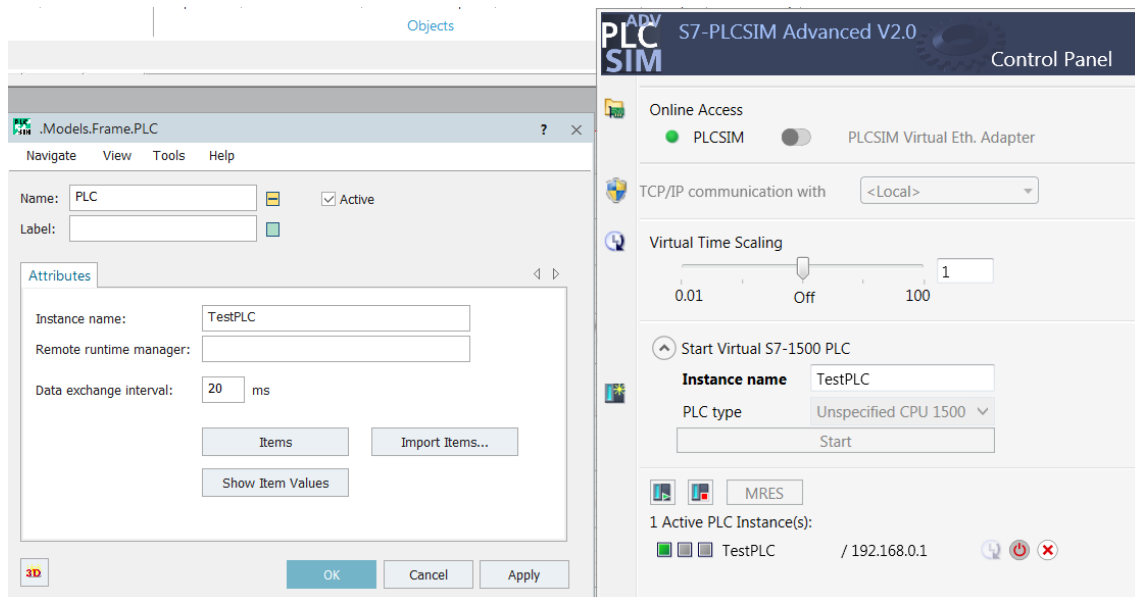
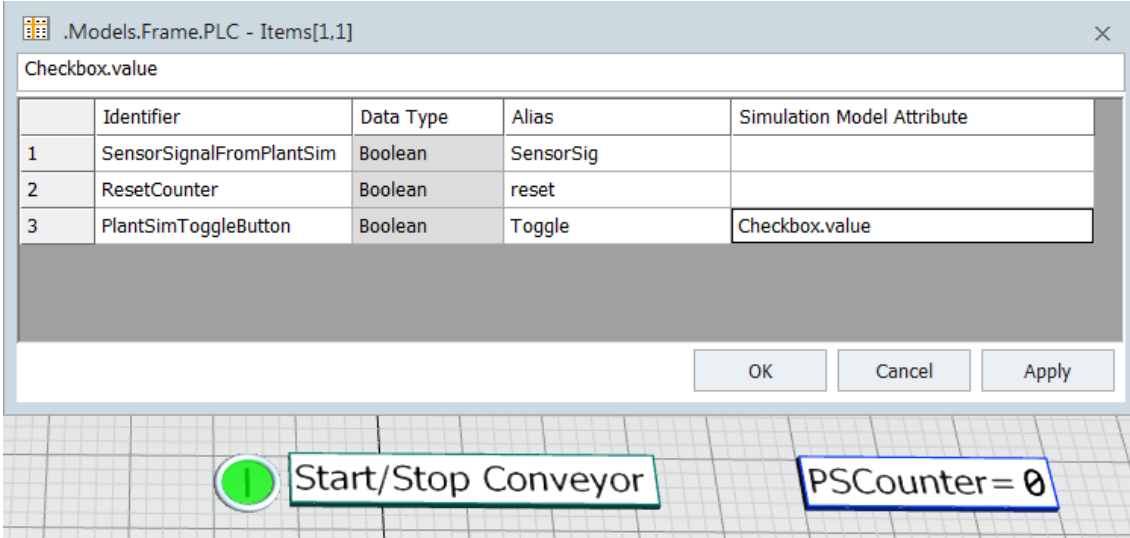


Figure B.3: Figure showing how to connect the PLCSIM_Advanced object to the virtual PLC

Once a connection is established Plant Simulation can begin to process signals to and from the PLC. However, in order for the PLC to be able to affect the simulation the signals must be connected to attributes in the model. This can be achieved in two different ways. Pressing the "Import Items..." button in the PLCSIM_Advanced object simply imports all of the tags found in the PLC, grouped into four different categories (see table 3.1).

Signals can also be added manually by first clicking on the "Items" button and then selecting (or creating) a signal group. Furthermore, you can also import signals from spreadsheets created in external programs such as Excel.

Double clicking on any of the four categories brings up a list of all the signals of that type. The "Identifier" column shows the name of the signal in the PLC, while the "Alias" column shows the path to the signal. In figure B.4 the path to the PLC tag "SensorSignalFromPlantSim" is `.Models.Frame.PLC.SensorSig`. The Simulation Model Attribute field can be filled in to connect the signal directly to a specific attribute, variable or SIMTALK method in Plant Simulation.



	Identifier	Data Type	Alias	Simulation Model Attribute
1	SensorSignalFromPlantSim	Boolean	SensorSig	
2	ResetCounter	Boolean	reset	
3	PlantSimToggleButton	Boolean	Toggle	Checkbox.value

Figure B.4: A table showing all IN-signals to the PLC as well as their aliases and connected model attributes

B.1.1 Sending data to the PLC

The PLC can receive signals from the Plant Simulation model in different ways. A signal can for example be connected to an attribute in the simulation. In figure B.4 the IN-signal "PlantSimToggleButton" is directly connected to the value of the checkbox (the button labelled "Start/Stop Conveyor"). When the value of the checkbox changes, the IN-signals changes with it.

An IN-signal can also be set directly from a SIMTALK method as if it was a regular variable in Plant Simulation. This also means that the signal can only change value when the method is triggered (unless it is also connected to a model attribute). The code in listing B.1 shows a way to send a pulse to the PLC when a sensor is triggered. A method has to be used for this since it's currently not possible to connect an IN signal directly to a sensor. This is because sensors are not defined objects in Plant Simulation. Sensors are instead treated as internal methods in the objects on which they are located, and methods cannot be connected to PLC IN-signals.

```

1 //Plant Simulation code for sending a pulse
2 --Start method when sensor is triggered
3 param SensorID: integer, Front: boolean
4 --Send TRUE
5 PLC.sensorSig := True
6 --Wait slightly longer than update frequency

```

```

7 wait 0.05
8 --Send FALSE
9 PLC.sensorSig := False

```

Listing B.1: Example of how a pulse can be sent

If a sensor is supposed to send a high signal while triggered, as opposed to a pulse, it can be solved by setting the sensor to trigger on both back and front while using a method similar to that in listing B.2:

```

1 //Method to hold signal value
2 --Parameter "Front" is true when MU enters sensor range and false when MU
  exits sensor range
3 param SensorID: integer, Front: boolean
4 --Set the signal to the PLC to the value of parameter "Front"
5 PLC.sensorSig := Front

```

Listing B.2: Example of how a signal can be high while a MU passes

B.1.2 Receiving data from the PLC

OUT-signals from the PLC can also be connected to the Plant Simulation model in the same way as IN-signals can. They can be directly linked to variables or attributes via the Simulation Model Attribute field in the item list.

A difference between IN and OUT signals is that OUT signals can be connected directly to a SIMTALK method, as seen in figure B.5. When such an OUT-signal changes value the method is triggered with the signal value as a parameter (see listing B.3). The signal must be declared as a parameter in the method, but the parameter does not have to be used.

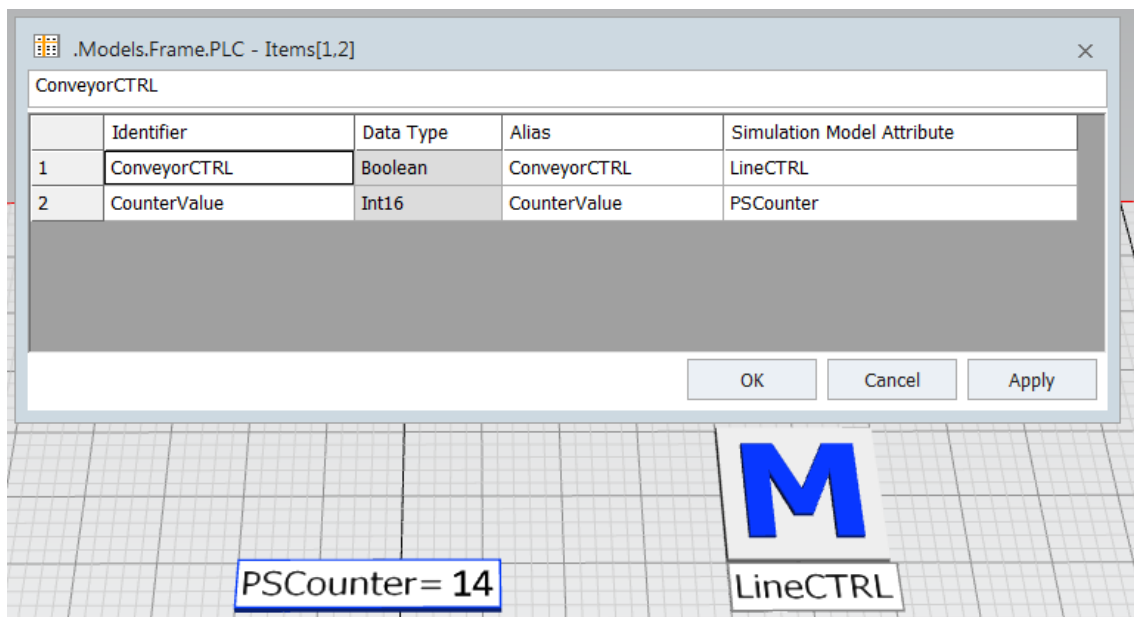


Figure B.5: Two OUT-signals connected to a method and a global variable respectively

```
1 //Method LineCTRL
2 --Recieve PLC-signal as parameter
3 param LineActive : boolean
4 --Set the attribute "stopped" of the objcet "line" to the same value as
  the incoming signal
5 line.stopped := LineActive
```

Listing B.3: The PLC-signal is sent to the method LineCTRL as the parameter LineActive. The attribute stopped in the object line is then set to the same value as the incoming PLC-signal

B.2 Connecting to a remote virtual PLC

Using a virtual PLC on a remote desktop works much in the same way as using it on a local one, although a few extra steps have to be taken to make the connection. First of all the two computers must be connected to the same network, either via LAN or Ethernet cable. Secondly, the virtual PLC must be using the PLCSIM Virtual Ethernet Adapter (refer to appendix A, section A.1.1 for how to do this).

Lastly you must add the address to the **computer that is running the PLC**, as well as the port it connects through, in the field labelled "Remote runtime manager". When that is done the connection is made by checking the "active" checkbox and pressing the "Apply" or "OK" buttons in the dialog window.

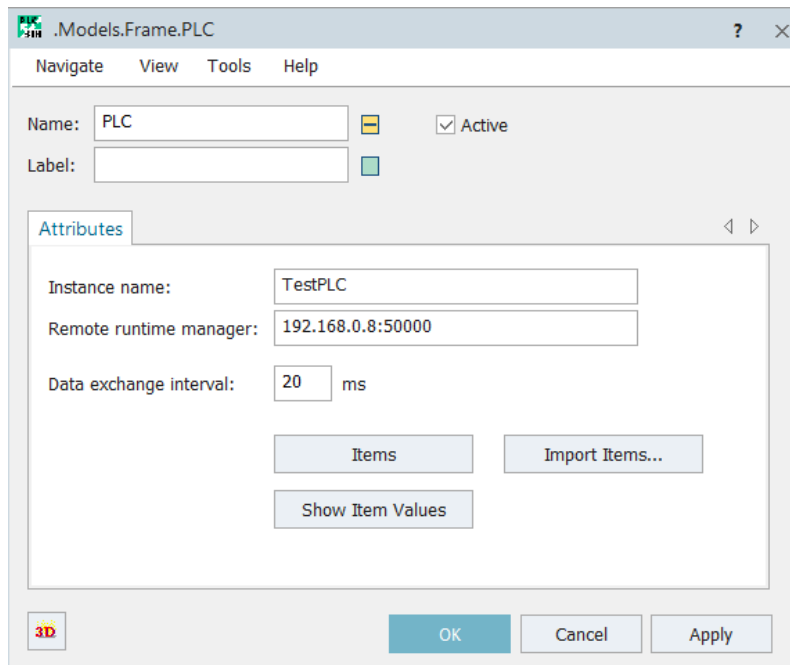


Figure B.6: Figure showing how to input the address and port number when connecting to a remote virtual PLC

B.3 Understanding the data exchange interval

In large models with many signals between Plant Simulation and the PLC performance can become an issue. The problem stems from the fact that Plant Simulation updates the signal lists at set intervals. However, the update rate (in milliseconds) can be changed in the "Data exchange interval" field in the PLCSIM_Advanced object (see figure B.7). A low number means that the data is exchanged more often, while a higher number exchanges data less frequently.

A slight exception occurs if the data exchange interval is set to 0; In that case the PLC is set to single step mode and data is exchanged after each cycle in the PLC. Furthermore the PLC is set in freeze-mode during the data exchange to make sure that all signals are transferred correctly. The freeze state is a specific mode in the virtual PLC where the virtual time is stopped, no timers are running and the program is not executed [7]. This is however NOT the same as stopping the PLC.

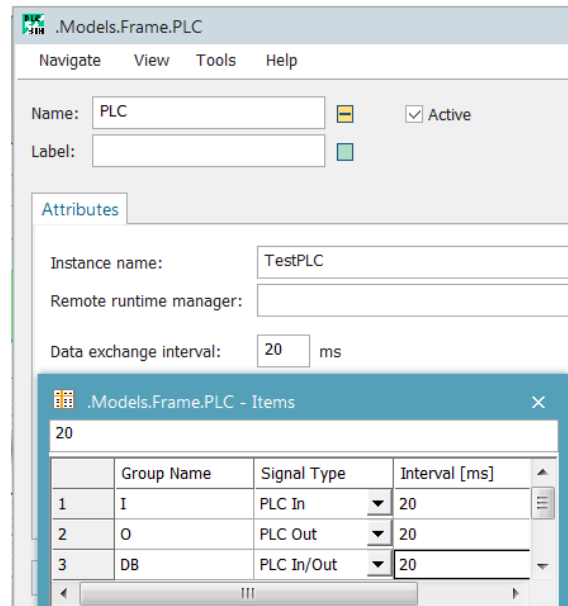


Figure B.7: Figure showing the data exchange interval fields in the PLCSIM_Advanced object and the item groups

It is also possible to have a slower update rate on certain signal groups by changing the interval field in the item list (see figure B.7). If an interval in the item list is higher than the data exchange rate then that group will update less frequently than the other signals. If the interval value is lower than the data exchange value (including interval = 0) however, the group should instead use the data exchange interval. Unfortunately, this last feature doesn't seem to work as intended in Plant Simulation 14.0. The interval for the item groups will always override the data exchange interval, making it redundant. Setting the data exchange interval to 2 seconds and a group interval to 20 milliseconds means that the group will exchange data every 20 milliseconds.

B.3.1 Notes regarding the data exchange

One very important thing to note is that the data exchange is **not** treated as an event in Plant Simulation! This holds true for both signals sent to and from the PLC. If the exchange should happen at a time when no events or activities are executing in the simulation no data will be transferred. However, it would seem like the exchange is "put on-hold" until the next event or activity happens in the model.

C

Control logic of the test model

The following chapter further explains the control logic of the test model. Figure C.1 shows the four networks making up the PLC code, figure C.2 shows how the PLC-IN signals are connected in Plant Simulation while C.3 shows the PLC-OUT signals.

C. Control logic of the test model

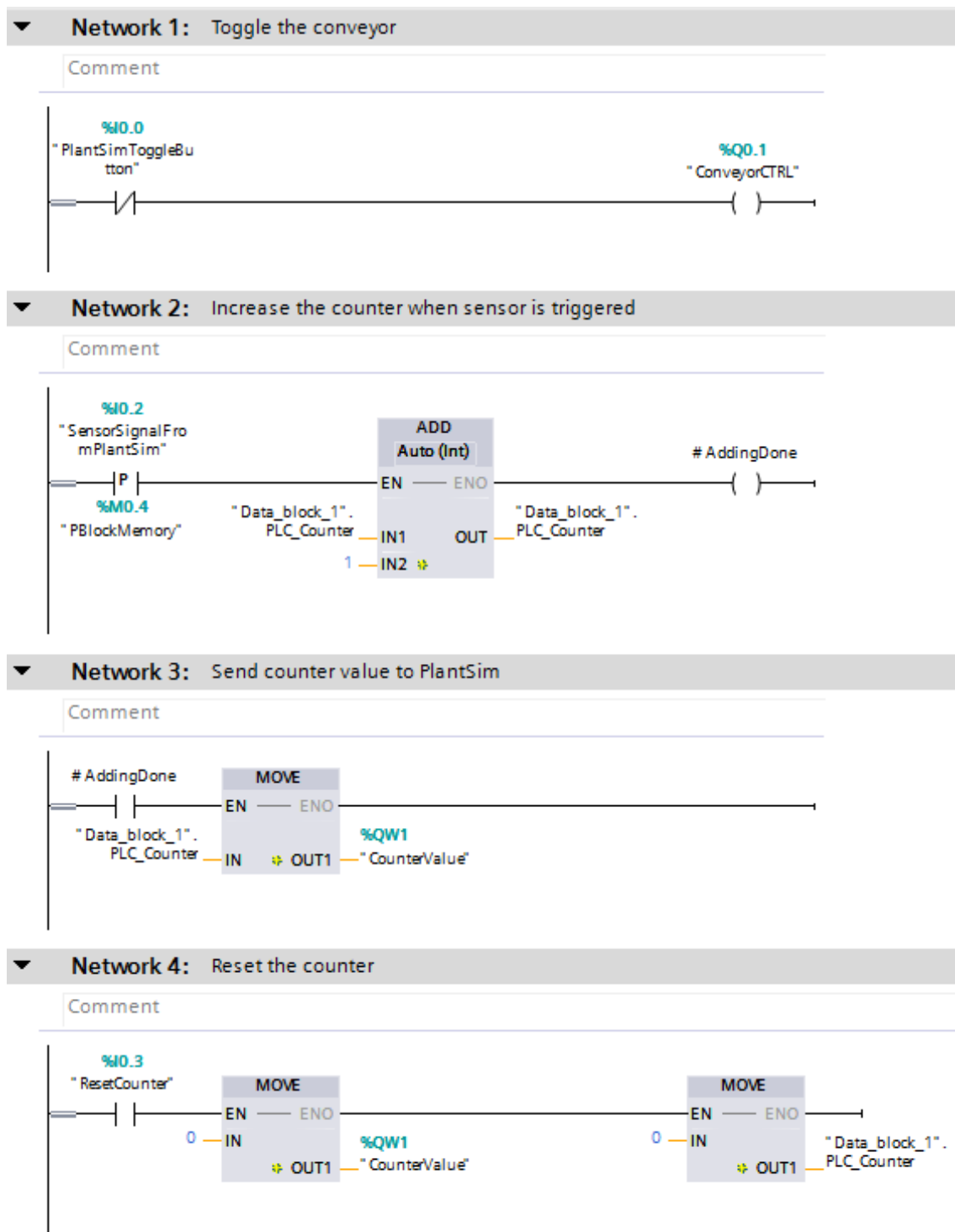


Figure C.1: The PLC code used for controlling the test model

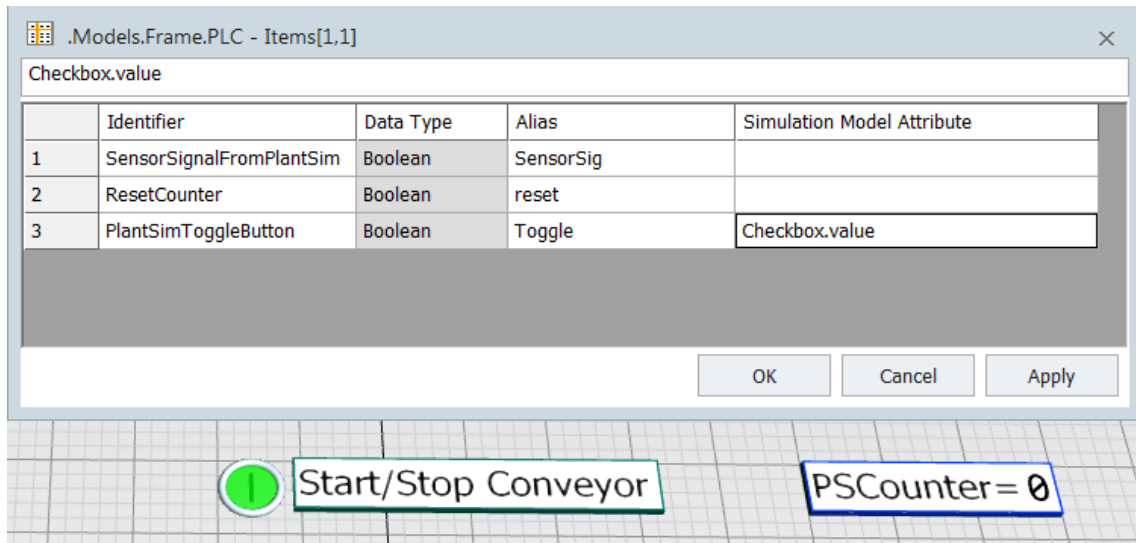


Figure C.2: The list of IN-signals and their connections in Plant Simulation

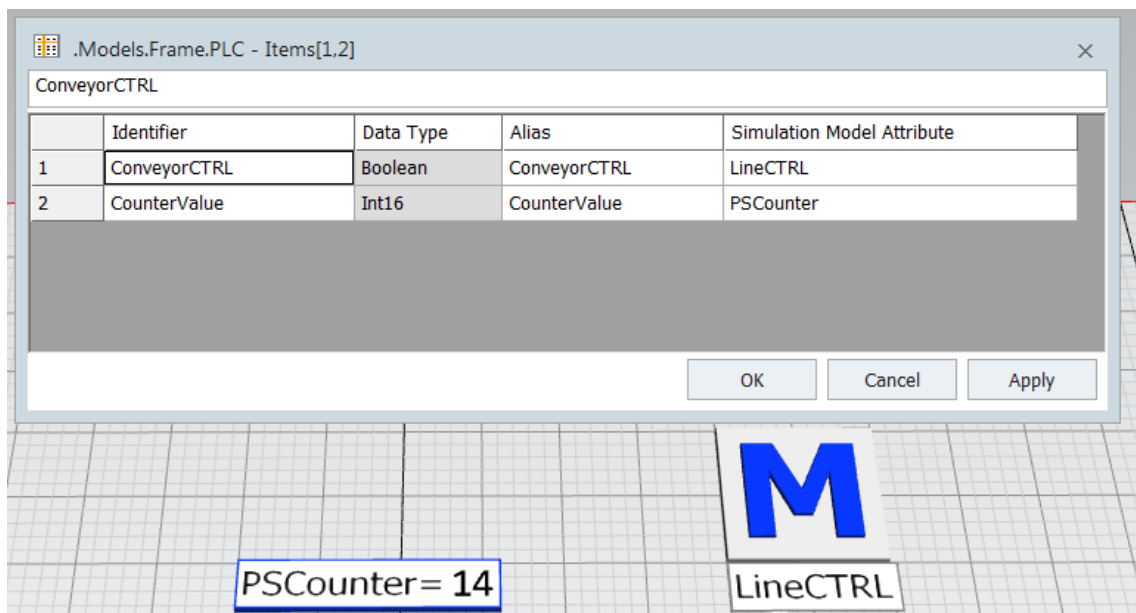


Figure C.3: The list of OUT-signals and their connections in Plant Simulation

D

SIMTALK code for exporting data

Plant Simulation has many ways of sharing and exporting data that is interesting and relevant to development projects. However, there is no way of exporting data such as layouts or connections to external programs. A SIMTALK code was therefore written so that information relevant to the PLC code generation could be exported. The code in listing D.1 is used to create a table, in reverse sequential order, of all rails in a safety zone. The table can then be exported as a text file, XML file or Excel spreadsheet.

```
1  -- .Models.Frame.XMLUpdater
2  /*****
3  This method is used to create a list of all tracks in a
4  safety zone in reverse sequential order.
5  Tracks that goes straight (postscript "_1") are added first
6  and deviating tracks (postscript "_9") are added afterwards.
7  The list is used in the C# programme to autogenerate PLC code
8  for the specific safety zone.
9  *****/
10 param Zone : string -> table[string,string]
11
12 var FramePath      : object := .models.frame
13 var ThisObject     : object
14 var MemoryObject   : object
15 var FirstPIB       : object
16 var LastPIB        : object
17 var FirstPIB2      : object
18 var LastPIB2       : object
19
20 var IsTrack        : string := "T_"
21 var IsSwitch       : string := "SW"
22 var IsStraight     : string := "_1"
23 var IsDeviant      : string := "_9"
24 var IsPIB          : string := "PIB"
25
26 var BlockType      : string
27 var FirstPIBName   : string
28 var LastPIBName    : string
29 var NameToEdit     : string
30 var PrevZone       : string
31 var SuccArray      : string[]
```

```

32 var PIBZone          : string
33
34 var i                : integer:= 1
35 var j                : integer:= 1
36
37 var FirstPIBBool: boolean:= FALSE
38 var LastPIBBool  : boolean:= FALSE
39 -----
40
41 //This is the table that is returned to the calling method
42 result.create
43 //A new zone always starts after a PIB
44 FirstPIBName := Zone+"_"+IsPIB
45
46 //Loop to find the first PIB
47 //If more than one is found the zone has two incoming tracks
48 repeat
49     //Find the first PIB
50     if pos(FirstPIBName,FramePath.node(i).name) > 0 AND FirstPIBBool =
FALSE
51         FirstPIBBool := TRUE
52         FirstPIB := FramePath.node(i)
53         //Look for a second first PIB
54     elseif pos(FirstPIBName,FramePath.node(i).name) > 0 AND
FirstPIBBool = TRUE
55         FirstPIB2 := FramePath.node(i)
56     end
57     i := i + 1
58 until i = FramePath.numNodes
59
60 //Loop to find the last PIB
61 //If more than one is found the zone has two outgoing tracks
62 repeat
63     //Find the last PIB
64     if pos(Zone,FramePath.node(j).name) > 0 AND pos(IsPIB,FramePath.
node(j).name) > 0 AND pos(FirstPIBName,FramePath.node(j).name) = 0 AND
LastPIBBool = FALSE
65         LastPIB := FramePath.node(j)
66         LastPIBBool := TRUE
67         //Look for a second last PIB
68     elseif pos(Zone,FramePath.node(j).name) > 0 AND pos(IsPIB,
FramePath.node(j).name) > 0 AND pos(FirstPIBName,FramePath.node(j).
name) = 0 AND LastPIBBool = TRUE
69         LastPIB2 := FramePath.node(j)
70     end
71     j := j + 1
72 until j = FramePath.numNodes
73
74 //Add all straight tracks

```

D. SIMTALK code for exporting data

```
75 ThisObject := FirstPIB.succ(1)
76 repeat
77     if pos(IsTrack,ThisObject.name) > 0 AND pos(IsSwitch,ThisObject.
name) = 0 -- Only add Tracks
78         NameToEdit := ThisObject.name
79         BlockType := omit(NameToEdit,1,6) -- Remove the string "
T_XXX_"
80         //Add the track zone and type to the top of the list
81         result.InsertRow(1)
82         result.WriteRow(1,1,Zone)
83         result.WriteRow(2,1,BlockType)
84     elseif pos(IsSwitch,ThisObject.name) > 1 AND ThisObject.numSucc >
1--If switch, look for more successors
85         MemoryObject := ThisObject.succ(2) --This WILL be
overwritten i a safetyzone has more than one switch (by design)
86     end
87     ThisObject := ThisObject.Succ(1) --Go to next successor
88 until ThisObject = LastPIB OR ThisObject = LastPIB2
89
90 --Add last PIB to result
91 SuccArray := splitString(ThisObject.succ(1).name,"_")
92 PIBZone := Zone+"_"+SuccArray[2]
93 result.InsertRow(1)
94 result.WriteRow(1,1,PIBZone)
95 result.WriteRow(2,1,IsPIB)
96
97 //If there are two incoming tracks, add the deviates
98 if FirstPIB2 /= void
99     ThisObject := FirstPIB2.succ(1)
100    repeat
101        NameToEdit := ThisObject.name
102        BlockType := omit(NameToEdit,1,6) -- Remove the string "
T_XXX_"
103        //Add the track zone and type to the top of the list
104        result.InsertRow(1)
105        result.WriteRow(1,1,Zone)
106        result.WriteRow(2,1,BlockType)
107        ThisObject := ThisObject.Succ(1)
108    until ThisObject.numPred > 1 -- When the next object has more than
one predecessor all deviates are added
109 end
110
111 //If there are two outgoing tracks, add the deviates
112 if MemoryObject /= void
113     ThisObject := MemoryObject
114     repeat
115         NameToEdit := ThisObject.name
116         BlockType := omit(NameToEdit,1,6) -- Remove the string "
T_XXX_"
```

```
117         //Add the track zone and type to the top of the list
118         result.InsertRow(1)
119         result.WriteRow(1,1,Zone)
120         result.WriteRow(2,1,BlockType)
121         ThisObject := ThisObject.Succ(1)
122         until ThisObject = LastPIB OR ThisObject = LastPIB2 -- When the
next object is a PIB, stop
123         //Find the name of the last PIB
124         SuccArray := splitString(ThisObject.succ(1).name,"_")
125         PIBZone := Zone+"_"+SuccArray[2]
126         //Add the PIB to the top of the list
127         result.InsertRow(1)
128         result.WriteRow(1,1,PIBZone)
129         result.WriteRow(2,1,IsPIB)
130     end
131
132 //Define the start of the safety zone
133 result.InsertRow(1)
134 result.WriteRow(1,1,Zone)
135 result.WriteRow(2,1,"START")
136
137 //Define the end of the safety zone
138 result.AppendRow("END","END")
139
140 return result
```

Listing D.1: The SIMTALK code used for creating the tables that are imported into PLCBuilder

E

XML code for safety zone 320

TIAPortalOpennessDemo connects to the STEP 7 TIA Portal and allow the import and export of ladder code as XML. C# Program PLCBuilder mentioned in 3.4 reads in an existing XML file if an update of a current code is desired, if a new auto generated code the program instead reads in an XML template as in listing E.1.

The program appends a rail XML template at row 58 (under the text INSERT_TEMPLATE_BELOW) for each rail cut listed in the list from Plant Simulation, and generic variables in the template are updated with the data from Plant Simulation. The XML file is then saved in the TIAPortalOpenness directory.

The XML code is imported into the PLC program in the TIA Portal via the TIAPortalOpennessDemo application. The application transforms the XML code into ladder logic which can then be loaded into a PLC.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <Document>
3   <Engineering version="V15" />
4   <SW.Blocks.FC ID="0">
5     <AttributeList>
6       <AutoNumber>true</AutoNumber>
7       <CodeModifiedDate ReadOnly="true">2018-04-23T07:50:22.1699085Z</
CodeModifiedDate>
8       <CompileDate ReadOnly="true">2018-04-23T07:50:45.9334297Z</
CompileDate>
9       <CreationDate ReadOnly="true">2018-04-23T07:44:16.4882299Z</
CreationDate>
10      <HandleErrorsWithinBlock ReadOnly="true">>false</
HandleErrorsWithinBlock>
11      <HeaderAuthor />
12      <HeaderFamily />
13      <HeaderName />
14      <HeaderVersion>0.1</HeaderVersion>
15      <Interface><Sections xmlns="http://www.siemens.com/automation/
Openness/SW/Interface/v3">
16        <Section Name="Input" />
17        <Section Name="Output" />
18        <Section Name="InOut" />
19        <Section Name="Temp">
20          <Member Name="tempblocked1" Datatype="Bool" />
21        </Section>
22        <Section Name="Constant" />
```

```

23 <Section Name="Return">
24   <Member Name="Ret_Val" Datatype="Void" Accessibility="Public" />
25 </Section>
26 </Sections></Interface>
27   <InterfaceModifiedDate ReadOnly="true">2018-04-23T07:44:16.4882299Z<
  /InterfaceModifiedDate>
28   <IsConsistent ReadOnly="true">>true</IsConsistent>
29   <IsIECCheckEnabled>>false</IsIECCheckEnabled>
30   <IsKnowHowProtected ReadOnly="true">>false</IsKnowHowProtected>
31   <IsWriteProtected ReadOnly="true">>false</IsWriteProtected>
32   <LibraryConformanceStatus ReadOnly="true">Error: The block contains
  calls of single instances.
33 Warning: The block contains direct access to inputs, outputs, bit memories
  , indirect access to the registry or indirect access to external
  memory areas that are unknown at the time of compilation.
34 </LibraryConformanceStatus>
35   <MemoryLayout>Optimized</MemoryLayout>
36   <ModifiedDate ReadOnly="true">2018-04-23T07:50:22.1699085Z</
  ModifiedDate>
37   <Name>T_ACC</Name>
38   <Number>8</Number>
39   <ParameterModified ReadOnly="true">2018-04-23T07:44:16.4882299Z</
  ParameterModified>
40   <PLCSimAdvancedSupport ReadOnly="true">>true</PLCSimAdvancedSupport>
41   <ProgrammingLanguage>LAD</ProgrammingLanguage>
42   <SetENOAutomatically>False</SetENOAutomatically>
43   <StructureModified ReadOnly="true">2018-04-23T07:44:16.4882299Z</
  StructureModified>
44   <UDABlockProperties />
45   <UDAEnableTagReadback>>false</UDAEnableTagReadback>
46 </AttributeList>
47 <ObjectList>
48   <MultilingualText ID="1" CompositionName="Comment">
49     <ObjectList>
50       <MultilingualTextItem ID="2" CompositionName="Items">
51         <AttributeList>
52           <Culture>en-US</Culture>
53           <Text />
54         </AttributeList>
55       </MultilingualTextItem>
56     </ObjectList>
57   </MultilingualText>
58 INSERT_TEMPLATE_BELOW
59 INSERT_TEMPLATE_ABOVE
60   <MultilingualText ID="8" CompositionName="Title">
61     <ObjectList>
62       <MultilingualTextItem ID="9" CompositionName="Items">
63         <AttributeList>
64           <Culture>en-US</Culture>

```

```

65     <Text />
66     </AttributeList>
67     </MultilingualTextItem>
68     </ObjectList>
69     </MultilingualText>
70     </ObjectList>
71 </SW.Blocks.FC>
72 </Document>

```

Listing E.1: The XML template for a safety zone to which rail templates are either manually or using PLCBuilder.

E.1 Results from imported code

Importing the code found in listing E.2 produces the PLC code seen in figure E.1.

```

1 <Document>
2 <Engineering version="V15" />
3 <SW.Blocks.FC ID="0">
4 <AttributeList>
5 <AutoNumber>>true</AutoNumber>
6 <CodeModifiedDate ReadOnly="true">2018-04-13T12:51:43.0092879Z</
CodeModifiedDate>
7 <CompileDate ReadOnly="true">2018-04-18T10:18:55.5744709Z</
CompileDate>
8 <CreationDate ReadOnly="true">2018-04-05T09:54:47.8320375Z</
CreationDate>
9 <HandleErrorsWithinBlock ReadOnly="true">>false</
HandleErrorsWithinBlock>
10 <HeaderAuthor />
11 <HeaderFamily />
12 <HeaderName />
13 <HeaderVersion>0.1</HeaderVersion>
14 <Interface><Sections xmlns="http://www.siemens.com/automation/
Openness/SW/Interface/v3">
15 <Section Name="Input">
16 <Member Name="I_Blocked" Datatype="Bool" Accessibility="Public" />
17 </Section>
18 <Section Name="Output">
19 <Member Name="O_Occupied" Datatype="Bool" Accessibility="Public" />
20 </Section>
21 <Section Name="InOut" />
22 <Section Name="Temp" />
23 <Section Name="Constant" />
24 <Section Name="Return">
25 <Member Name="Ret_Val" Datatype="Void" Accessibility="Public" />
26 </Section>
27 </Sections></Interface>

```

```

28     <InterfaceModifiedDate ReadOnly="true">2018-04-10T07:28:52.155191Z</
InterfaceModifiedDate>
29     <IsConsistent ReadOnly="true">true</IsConsistent>
30     <IsIECCheckEnabled>false</IsIECCheckEnabled>
31     <IsKnowHowProtected ReadOnly="true">false</IsKnowHowProtected>
32     <IsWriteProtected ReadOnly="true">false</IsWriteProtected>
33     <LibraryConformanceStatus ReadOnly="true">Error: The block contains
calls of single instances.
34 Warning: The object contains access to global data blocks.
35 Warning: The block contains direct access to inputs, outputs, bit memories
, indirect access to the registry or indirect access to external
memory areas that are unknown at the time of compilation.
36 </LibraryConformanceStatus>
37     <MemoryLayout>Optimized</MemoryLayout>
38     <ModifiedDate ReadOnly="true">2018-04-13T12:51:43.0092879Z</
ModifiedDate>
39     <Name>320</Name>
40     <Number>1</Number>
41     <ParameterModified ReadOnly="true">2018-04-10T07:28:52.155191Z</
ParameterModified>
42     <PLCSimAdvancedSupport ReadOnly="true">true</PLCSimAdvancedSupport>
43     <ProgrammingLanguage>LAD</ProgrammingLanguage>
44     <SetENOAutomatically>False</SetENOAutomatically>
45     <StructureModified ReadOnly="true">2018-04-10T07:28:52.155191Z</
StructureModified>
46     <UDABlockProperties />
47     <UDAEnableTagReadback>false</UDAEnableTagReadback>
48 </AttributeList>
49 <ObjectList>
50     <MultilingualText ID="1" CompositionName="Comment">
51         <ObjectList>
52             <MultilingualTextItem ID="2" CompositionName="Items">
53                 <AttributeList>
54                     <Culture>en-US</Culture>
55                     <Text />
56                 </AttributeList>
57             </MultilingualTextItem>
58         </ObjectList>
59     </MultilingualText>
60     <SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
61         <AttributeList>
62             <NetworkSource><FlgNet xmlns="http://www.siemens.com/automation/
Openness/SW/NetworkSource/FlgNet/v2">
63 <Parts>
64     <Access Scope="GlobalVariable" UIId="21">
65     <Symbol>
66     <Component Name="320_ACC1_Occupied" />
67     <Address Area="Input" Type="Bool" BitOffset="29" Informative="true
" />

```



```
68     </Symbol>
69 </Access>
70 <Access Scope="GlobalVariable" UId="22">
71     <Symbol>
72         <Component Name="320/330_PIB" />
73         <Component Name="0_Blocking" />
74         <Address Area="None" Type="Bool" BlockNumber="2" BitOffset="88"
Informative="true" />
75     </Symbol>
76 </Access>
77 <Access Scope="GlobalVariable" UId="23">
78     <Symbol>
79         <Component Name="320_ACC1_Power" />
80         <Address Area="Output" Type="Bool" BitOffset="29" Informative="
true" />
81     </Symbol>
82 </Access>
83 <Access Scope="GlobalVariable" UId="24">
84     <Symbol>
85         <Component Name="320_ACC1_SpeedCmd" />
86         <Address Area="Output" Type="Int" BitOffset="1264" Informative="
true" />
87     </Symbol>
88 </Access>
89 <Access Scope="LocalVariable" UId="25">
90     <Symbol>
91         <Component Name="0_Occupied" />
92     </Symbol>
93 </Access>
94 <Part Name="Contact" UId="26" />
95 <Call UId="27">
96     <CallInfo Name="ACC" BlockType="FB">
97         <IntegerAttribute Name="BlockNumber" Informative="true">4</
IntegerAttribute>
98         <DateAttribute Name="ParameterModifiedTS" Informative="true">
2018-04-13T12:34:05</DateAttribute>
99         <Instance Scope="GlobalVariable" UId="28">
100             <Component Name="320_ACC1" />
101             <Address Area="DB" Type="ACC" BlockNumber="3" BitOffset="0"
Informative="true" />
102         </Instance>
103         <Parameter Name="I_Blocked" Section="Input" Type="Bool">
104             <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
105         </Parameter>
106         <Parameter Name="I_Occupied" Section="Input" Type="Bool">
107             <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
108         </Parameter>
```

```
109     <Parameter Name="O_Power" Section="Output" Type="Bool">
110         <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
111     </Parameter>
112     <Parameter Name="O_SpeedCmd" Section="Output" Type="Int">
113         <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
114     </Parameter>
115     <Parameter Name="O_Blocking" Section="Output" Type="Bool">
116         <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
117     </Parameter>
118 </CallInfo>
119 </Call>
120 </Parts>
121 <Wires>
122     <Wire UId="29">
123         <Powerrail />
124         <NameCon UId="27" Name="en" />
125         <NameCon UId="26" Name="in" />
126     </Wire>
127     <Wire UId="30">
128         <IdentCon UId="21" />
129         <NameCon UId="26" Name="operand" />
130     </Wire>
131     <Wire UId="31">
132         <NameCon UId="26" Name="out" />
133         <NameCon UId="27" Name="I_Occupied" />
134     </Wire>
135     <Wire UId="32">
136         <IdentCon UId="22" />
137         <NameCon UId="27" Name="I_Blocked" />
138     </Wire>
139     <Wire UId="33">
140         <NameCon UId="27" Name="O_Power" />
141         <IdentCon UId="23" />
142     </Wire>
143     <Wire UId="34">
144         <NameCon UId="27" Name="O_SpeedCmd" />
145         <IdentCon UId="24" />
146     </Wire>
147     <Wire UId="35">
148         <NameCon UId="27" Name="O_Blocking" />
149         <IdentCon UId="25" />
150     </Wire>
151 </Wires>
152 </FlgNet></NetworkSource>
153     <ProgrammingLanguage>LAD</ProgrammingLanguage>
154 </AttributeList>
```

```
155     <ObjectList>
156       <MultilingualText ID="4" CompositionName="Comment">
157         <ObjectList>
158           <MultilingualTextItem ID="5" CompositionName="Items">
159             <AttributeList>
160               <Culture>en-US</Culture>
161               <Text />
162             </AttributeList>
163           </MultilingualTextItem>
164         </ObjectList>
165       </MultilingualText>
166       <MultilingualText ID="6" CompositionName="Title">
167         <ObjectList>
168           <MultilingualTextItem ID="7" CompositionName="Items">
169             <AttributeList>
170               <Culture>en-US</Culture>
171               <Text>ACC1</Text>
172             </AttributeList>
173           </MultilingualTextItem>
174         </ObjectList>
175       </MultilingualText>
176     </ObjectList>
177   </SW.Blocks.CompileUnit>
178   <SW.Blocks.CompileUnit ID="8" CompositionName="CompileUnits">
179     <AttributeList>
180       <NetworkSource><FlgNet xmlns="http://www.siemens.com/automation/
181       Openness/SW/NetworkSource/FlgNet/v2">
182     <Parts>
183       <Access Scope="GlobalVariable" UId="21">
184         <Symbol>
185           <Component Name="320_330_PIB_Occupied" />
186           <Address Area="Input" Type="Bool" BitOffset="30" Informative="true
187           " />
188         </Symbol>
189       </Access>
190       <Access Scope="GlobalVariable" UId="22">
191         <Symbol>
192           <Component Name="TRUE" />
193           <Address Area="Memory" Type="Bool" BitOffset="0" Informative="true
194           " />
195         </Symbol>
196       </Access>
197       <Access Scope="GlobalVariable" UId="23">
198         <Symbol>
199           <Component Name="TRUE" />
200           <Address Area="Memory" Type="Bool" BitOffset="0" Informative="true
201           " />
202         </Symbol>
203       </Access>
```

```

200 <Access Scope="LocalVariable" UId="24">
201   <Symbol>
202     <Component Name="I_Blocked" />
203   </Symbol>
204 </Access>
205 <Access Scope="GlobalVariable" UId="25">
206   <Symbol>
207     <Component Name="320_330_PIB_Power" />
208     <Address Area="Output" Type="Bool" BitOffset="30" Informative="
209 true" />
210   </Symbol>
211 </Access>
212 <Access Scope="GlobalVariable" UId="26">
213   <Symbol>
214     <Component Name="320_330_PIB_SpeedCmd" />
215     <Address Area="Output" Type="Int" BitOffset="1280" Informative="
216 true" />
217   </Symbol>
218 </Access>
219 <Part Name="Contact" UId="27" />
220 <Call UId="28">
221   <CallInfo Name="PIB" BlockType="FB">
222     <IntegerAttribute Name="BlockNumber" Informative="true">2</
223 IntegerAttribute>
224     <DateAttribute Name="ParameterModifiedTS" Informative="true">
225 2018-04-13T12:34:36</DateAttribute>
226     <Instance Scope="GlobalVariable" UId="29">
227       <Component Name="320/330_PIB" />
228       <Address Area="DB" Type="PIB" BlockNumber="2" BitOffset="0"
229 Informative="true" />
230     </Instance>
231     <Parameter Name="I_PowerPre" Section="Input" Type="Bool">
232       <StringAttribute Name="InterfaceFlags" Informative="true">
233 S7_Visible</StringAttribute>
234     </Parameter>
235     <Parameter Name="I_PowerSucc" Section="Input" Type="Bool">
236       <StringAttribute Name="InterfaceFlags" Informative="true">
237 S7_Visible</StringAttribute>
238     </Parameter>
239     <Parameter Name="I_Blocked" Section="Input" Type="Bool">
240       <StringAttribute Name="InterfaceFlags" Informative="true">
241 S7_Visible</StringAttribute>
242     </Parameter>
243     <Parameter Name="I_Occupied" Section="Input" Type="Bool">
244       <StringAttribute Name="InterfaceFlags" Informative="true">
245 S7_Visible</StringAttribute>
246     </Parameter>
247     <Parameter Name="O_Power" Section="Output" Type="Bool">

```

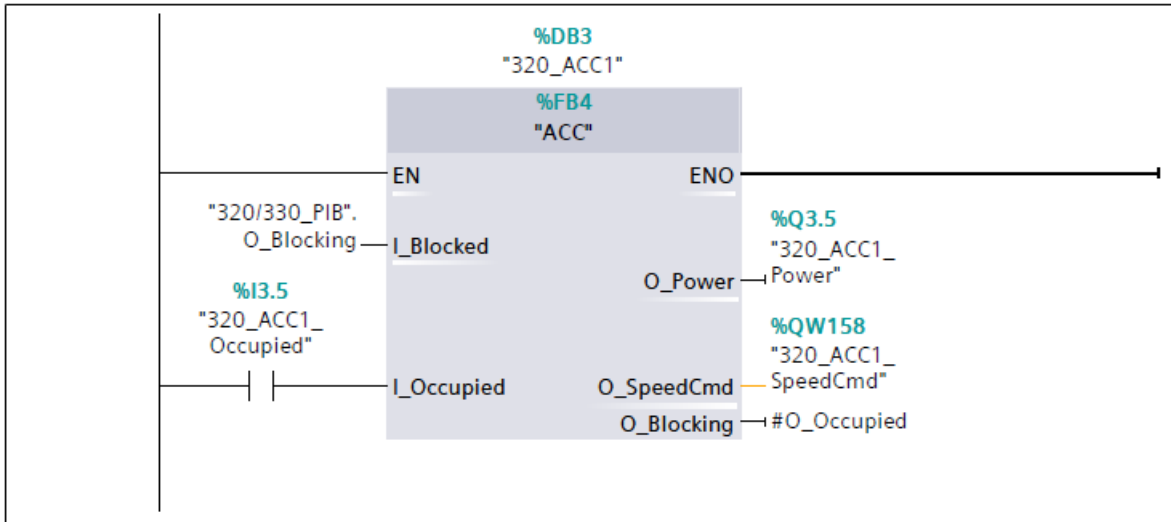
```
239     <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
240   </Parameter>
241   <Parameter Name="O_SpeedCmd" Section="Output" Type="Int">
242     <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
243   </Parameter>
244   <Parameter Name="O_Blocking" Section="Output" Type="Bool">
245     <StringAttribute Name="InterfaceFlags" Informative="true">
S7_Visible</StringAttribute>
246   </Parameter>
247 </CallInfo>
248 </Call>
249 </Parts>
250 <Wires>
251   <Wire UIId="31">
252     <Powerrail />
253     <NameCon UIId="28" Name="en" />
254     <NameCon UIId="27" Name="in" />
255   </Wire>
256   <Wire UIId="32">
257     <IdentCon UIId="21" />
258     <NameCon UIId="27" Name="operand" />
259   </Wire>
260   <Wire UIId="33">
261     <NameCon UIId="27" Name="out" />
262     <NameCon UIId="28" Name="I_Occupied" />
263   </Wire>
264   <Wire UIId="34">
265     <IdentCon UIId="22" />
266     <NameCon UIId="28" Name="I_PowerPre" />
267   </Wire>
268   <Wire UIId="35">
269     <IdentCon UIId="23" />
270     <NameCon UIId="28" Name="I_PowerSucc" />
271   </Wire>
272   <Wire UIId="36">
273     <IdentCon UIId="24" />
274     <NameCon UIId="28" Name="I_Blocked" />
275   </Wire>
276   <Wire UIId="37">
277     <NameCon UIId="28" Name="O_Power" />
278     <IdentCon UIId="25" />
279   </Wire>
280   <Wire UIId="38">
281     <NameCon UIId="28" Name="O_SpeedCmd" />
282     <IdentCon UIId="26" />
283   </Wire>
284   <Wire UIId="39">
```

```
285     <NameCon UId="28" Name="O_Blocking" />
286     <OpenCon UId="30" />
287     </Wire>
288 </Wires>
289 </FlgNet></NetworkSource>
290     <ProgrammingLanguage>LAD</ProgrammingLanguage>
291 </AttributeList>
292 <ObjectList>
293     <MultilingualText ID="9" CompositionName="Comment">
294         <ObjectList>
295             <MultilingualTextItem ID="A" CompositionName="Items">
296                 <AttributeList>
297                     <Culture>en-US</Culture>
298                     <Text />
299                 </AttributeList>
300             </MultilingualTextItem>
301         </ObjectList>
302     </MultilingualText>
303     <MultilingualText ID="B" CompositionName="Title">
304         <ObjectList>
305             <MultilingualTextItem ID="C" CompositionName="Items">
306                 <AttributeList>
307                     <Culture>en-US</Culture>
308                     <Text>command for PIB310/320</Text>
309                 </AttributeList>
310             </MultilingualTextItem>
311         </ObjectList>
312     </MultilingualText>
313 </ObjectList>
314 </SW.Blocks.CompileUnit>
315 <SW.Blocks.CompileUnit ID="D" CompositionName="CompileUnits">
316     <AttributeList>
317         <NetworkSource />
318         <ProgrammingLanguage>LAD</ProgrammingLanguage>
319     </AttributeList>
320     <ObjectList>
321         <MultilingualText ID="E" CompositionName="Comment">
322             <ObjectList>
323                 <MultilingualTextItem ID="F" CompositionName="Items">
324                     <AttributeList>
325                         <Culture>en-US</Culture>
326                         <Text />
327                     </AttributeList>
328                 </MultilingualTextItem>
329             </ObjectList>
330         </MultilingualText>
331         <MultilingualText ID="10" CompositionName="Title">
332             <ObjectList>
333                 <MultilingualTextItem ID="11" CompositionName="Items">
```

```
334         <AttributeList>
335             <Culture>en-US</Culture>
336             <Text />
337         </AttributeList>
338     </MultilingualTextItem>
339 </ObjectList>
340 </MultilingualText>
341 </ObjectList>
342 </SW.Blocks.CompileUnit>
343 <MultilingualText ID="12" CompositionName="Title">
344     <ObjectList>
345         <MultilingualTextItem ID="13" CompositionName="Items">
346             <AttributeList>
347                 <Culture>en-US</Culture>
348                 <Text />
349             </AttributeList>
350         </MultilingualTextItem>
351     </ObjectList>
352 </MultilingualText>
353 </ObjectList>
354 </SW.Blocks.FC>
355 </Document>
```

Listing E.2: The XML code that is to be imported to the TIA portal

Network 1: ACC1



Network 2: command for PIB310/320

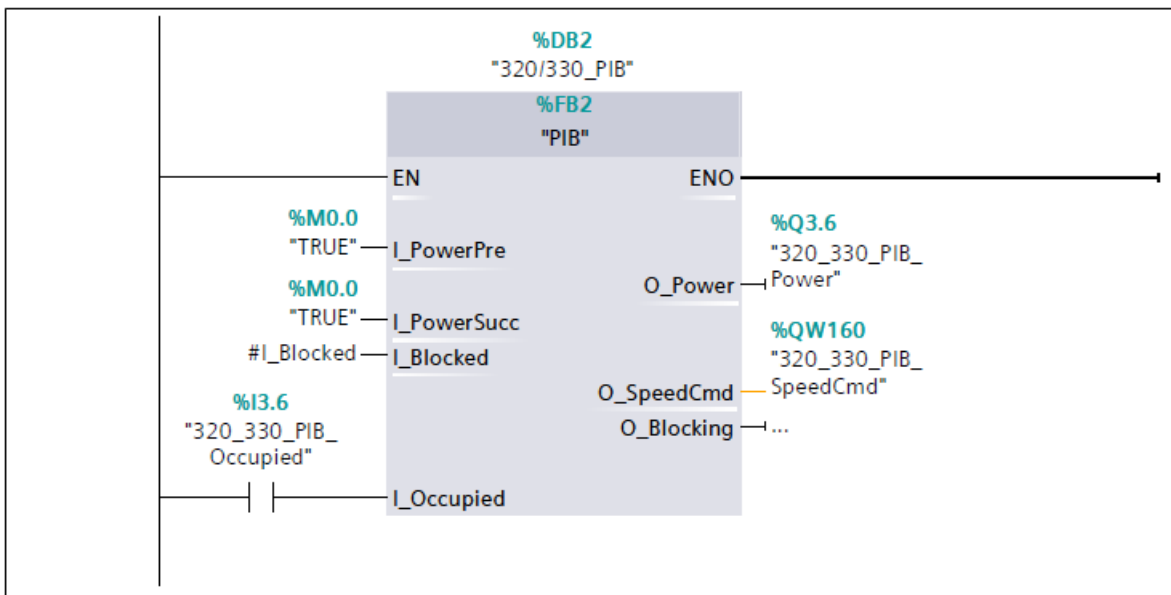


Figure E.1: The resulting PLC code, in ladder logic, for safety zone 320

F

PLC Standard

The PLC library is made up of several different function blocks that represent the different functions laid out in the Volvo Cars standard for EMS lines [12]. The list in section F.1 describes the different types of blocks.

F.1 Behavior of the Function Blocks in GlobalLib

The PLC library is based on the Volvo EMS standard and the difference between the control logics are explained in the list below:

- **ACC** is the function block (FB) for Accumulation Blocks. Multiple hangers are allowed on these rails. The ACC block sets a constant speed command to the rail as long as the zone is active.
 - When the ACC is not blocked the FB sets SpeedCmd to 40 meter-s/minute. When blocked the FB sets SpeedCmd to 0 meter/minute
- **ASB** is the FB for After Segment Blocks, while a hanger is present on this rail the lift or switch cant move without causing damage.
 - When occupied and not blocked the FB sets SpeedCmd to 6 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute
- **EV** is the FB controlling the clamping of elevators
 - When a hanger is present in the SB for the elevator the hanger is clamped after which point another PLC moves the elvator to the right position.

- **Interlock** is an internal FB inside the EV and SW blocks that stops the movement of movable rails if there is an interlocking problem with a hanger, *e.g.* a hanger has entered the area but hasn't traveled through to the TVB yet.
- **MVB** is the function block for Move permission Blocks. It is used to indicate that a switch or lift can move without interlocking problems with a hanger.
 - When occupied and not blocked the FB sets SpeedCmd to 30 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute
- **PIB** is the FB for Power Isolation Block which only has power enabled if the safety zone before and after the block both have power enabled.
 - When occupied and not blocked the FB sets SpeedCmd to 30 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute.
- **SAB** is the FB for Safety Blocks that cuts the power before a switch or a rail.
 - When occupied and not blocked the FB sets SpeedCmd to 6 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute
- **SB** is the FB Segment Blocks, the moving rail in a switch or elevator.
 - When occupied and not blocked the FB sets SpeedCmd to 6 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute
- **SpeedCtrl** is an internal FB in every rail FB that sets the SpeedCmd signal of the rail depending on multiple inputs.
- **SW** is the FB controlling switches.
- **TVB** is the FB for Transport permission Block, indicating that a PLC has moved past a point of interest, such as the end of an MVB.
 - When occupied and not blocked the FB sets SpeedCmd to 30 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to meter/minute

- **TZ** is the FB for Transport zones, used where only a single carrier are allowed in curves or short straight lines.
 - When occupied and not blocked the FB sets SpeedCmd to 6 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute
- **WB** is the FB for Waitng Blocks, which controls the flow before switches and lifts or where flow decisions are taken.
 - When occupied and not blocked the FB sets SpeedCmd to 6 meter-s/minute. When occupied it sets blocking of previous rail. When blocked or not occupied the FB sets SpeedCmd to 0 meter/minute