Peer 2 Peer communications

PES 4.1

Written by Hui-Alvin Wen & Marco Siena

Rev 1.0
May 2016

Table of Contents

1 INTRODUCTION AND OBJECTIVE

The purpose of this document is to summarize the Peer2Peer communications features available in PES 4.1 and to provide typical examples that can be found in real projects.

Peer2Peer communications means data exchange between controllers, which is a common requirement in DCS systems.

1.1 Prerequisites

In order to have a good understanding of this document it is necessary to have knowledge of how to create and configure PES systems using the StruXureWare Process Expert Engineering Client software.

It is required also to have a good understanding of the HAL 2.0 (Hardware Abstraction Layer, based on new REF_TO types of Unity Pro) mechanism introduced in PES 4.1.

1.2 Reference documentation

The Peer2Peer communication feature is described in detail in *Process Expert User Guide* – (*EIO0000001114 01/2016)* starting from page 366.

Some further corrections to this manual are shown in the appendix (chapter 9).

2 PEER 2 PEER IMPLEMENTATION IN PES

2.1 Communication technology

Peer2Peer communication is implemented in PES system using Modbus TCP implicit messaging (IO Scanner) as the communication channel.

It is possible to define a Peer2Peer communication channel between a controller communication card with Modbus TCP/IP IO scanner capability and a controller that acts as a Modbus TCP/IP server.

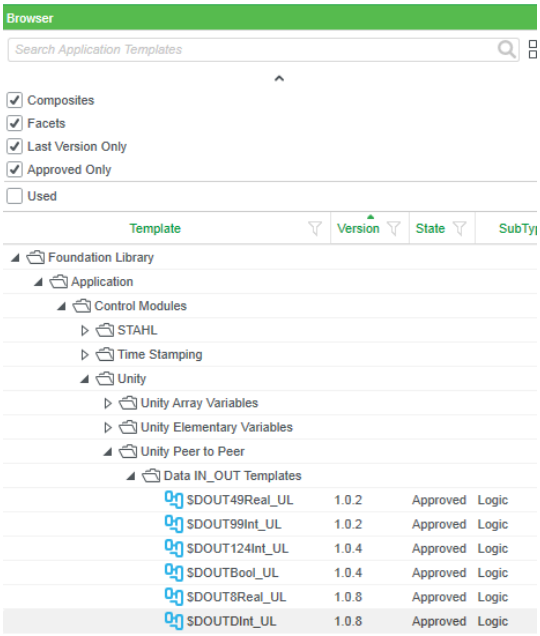
On top of this communication channel, using PES it's possible to add a certain amount of "network variables".
A network variable is a Unity variable (elementary or derived) defined in a certain control project (having the role of the "owner") whose value has to be transferred to another control project (having the role of the "consumer").

2.2 PES library features

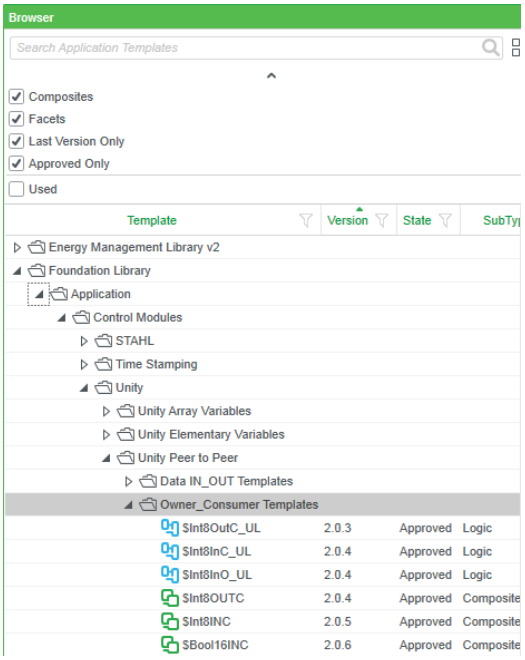
PES engineering environment provides also a set of features allowing to easily defining the creation and the exchange of network variables, depending on the scenario.

In the PES foundation library, there are two different sets of objects:

1- *Foundation Library ➔ Application ➔ Control Modules ➔ Unity ➔ Unity Peer to Peer ➔* ***Data IN_OUT templates***



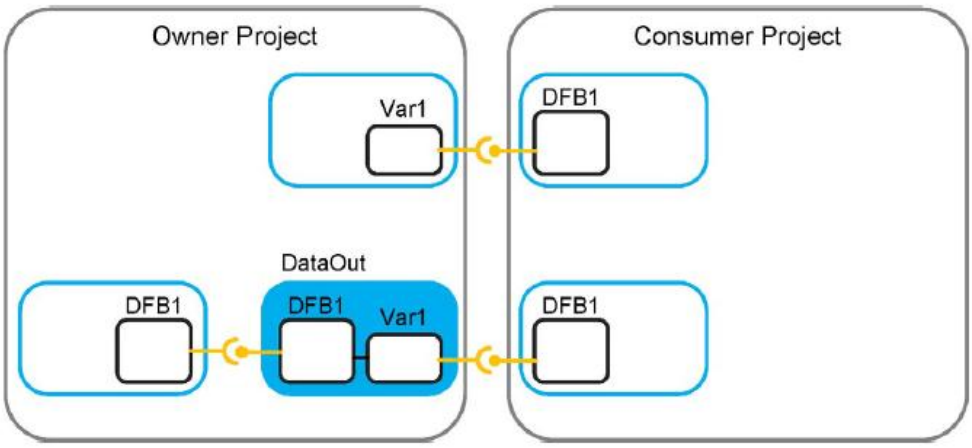
2- *FoundationLibrary ➔ Application ➔ Control Modules ➔ Unity ➔ Unity Peer to Peer ➔* ***Owner_Consumer Templates***



2.3 Peer2Peer scenario 1: scattered data

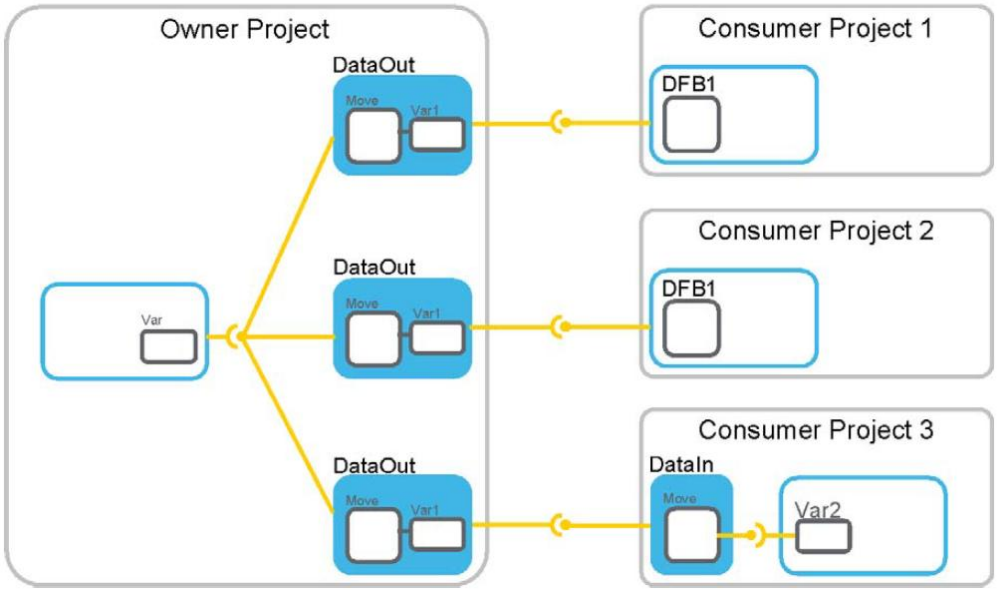
This scenario typically answers to the need of exchanging variables that are involved in the controller logic.

In the following drawing it is represented a Consumer Project with some DFB instances that needs some variables that are managed by a consumer project.
A typical application could be a BOOL variable on Owner Project that has to connected to an input pin of a DFB in the consumer project (for example, `ILCK` or `FAIL` pins of a GPL `DEVCTL` block).
This variable can be either an elementary variable or the output of a DFB that is assigned and executed on Owner project.



In this case, on a single Peer2Peer communication channel (Owner Project ⇔ Consumer Project), multiple network variables can be exchanged.

A single information that resides on the Owner Project can be transferred also to multiple Consumer Projects, as shown in the following drawing.



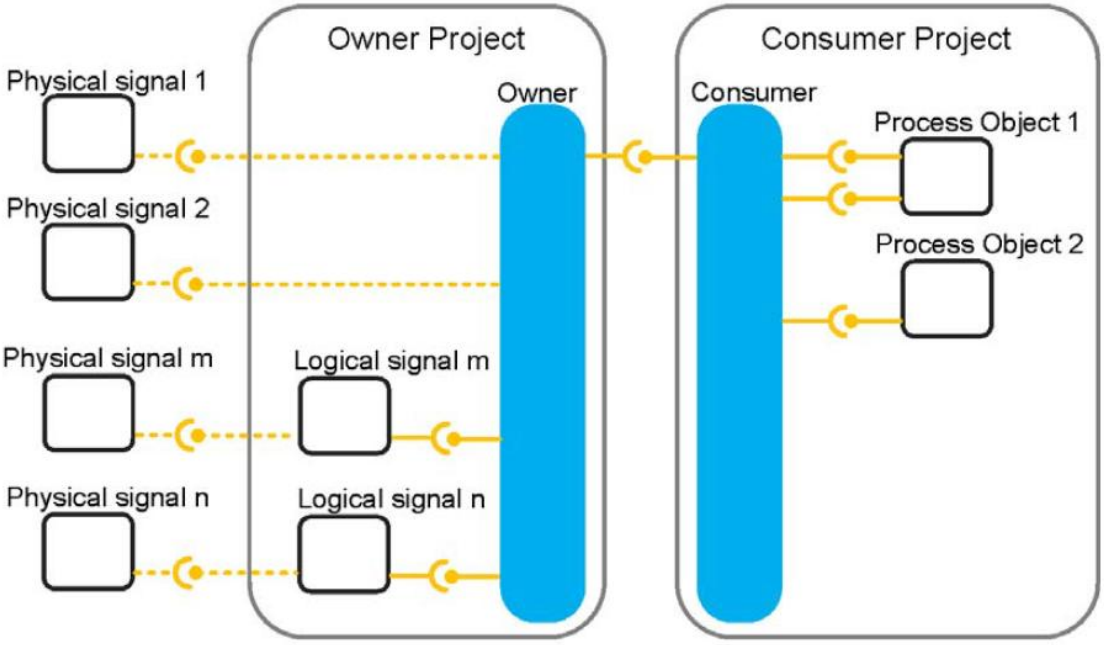
In this case it is necessary to define 3 Peer2Peer communication channels:
Owner Project ⇔ Consumer Project 1
Owner Project ⇔ Consumer Project 2
Owner Project ⇔ Consumer Project 3

For the implementation of this scenario, it is possible to use the ***DATA IN_OUT templates*** family objects.

2.4 Peer2Peer scenario 2: hardware signals exchange

The second scenario is applicable when it is necessary to the exchange variables related to Physical signals.

In this case, the Owner Project has some signals connected to its physical I/O system (can be local or remote I/Os) that are needed from a process object which is assigned and executed on the Consumer Project.



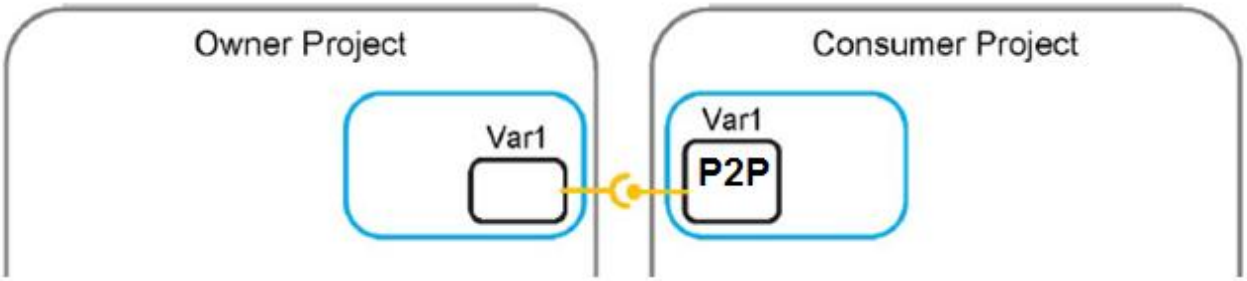
For the implementation of this scenario, it is possible to use the ***OWNER_CONSUMER templates*** objects.

These objects allow to define blocks of I/Os, that can be easily assigned in the hardware mapping stage to an I/O module. This mechanism is very useful when one or more controllers are used as a Distributed I/O: a typical case is the usage of the M340 PRA (Peripheral Remote Adapter) low end CPU.

2.5 Peer2Peer scenario 3: simple variable exchange

This scenario answers to the need of exchanging variables not related to objects or DFBs in a simple way.

This mechanism is achieved by mean of refinement in control projects.
It is possible to define a variable as network variable on a control project by entering “P2P” string in the “Custom” attribute of Unity Pro. Assigning the “P2P” attribute means that the relevant control project will act as Consumer, indeed a variable with the same name has to exist on the Owner project.



3 REFERENCE PLATFORM CONFIGURATION

3.1 Objective of the platform

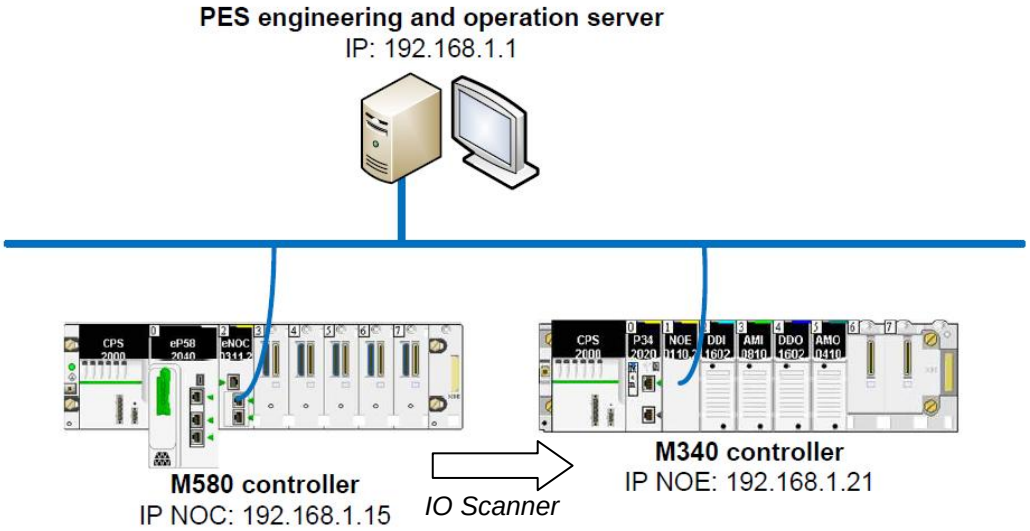
The objective of this platform is to test all the above mentioned scenarios in a simple real architecture.

This architecture is made by the following elements:

1- M580 CPU with NOC card
2- M340 CPU with NOE, 16DI, 8AI, 16DO, 4AO cards.
3- PES engineering and operation server

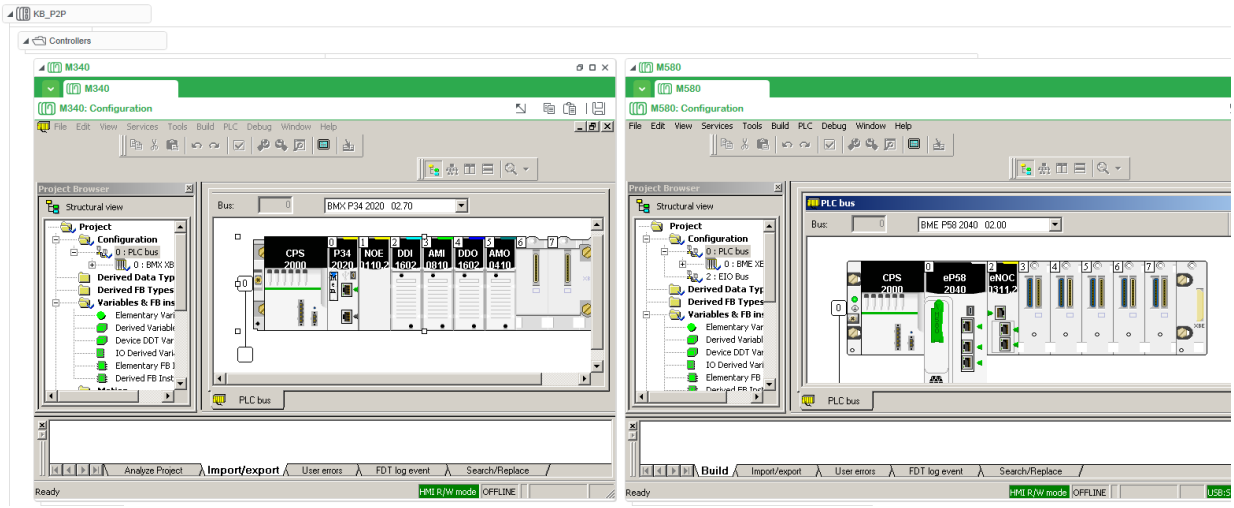
All these components are connected to a control network.

The M580 NOC will be configured as Modbus TCP I/O scanner client, M340 NOE will be configured as Modbus TCP server.

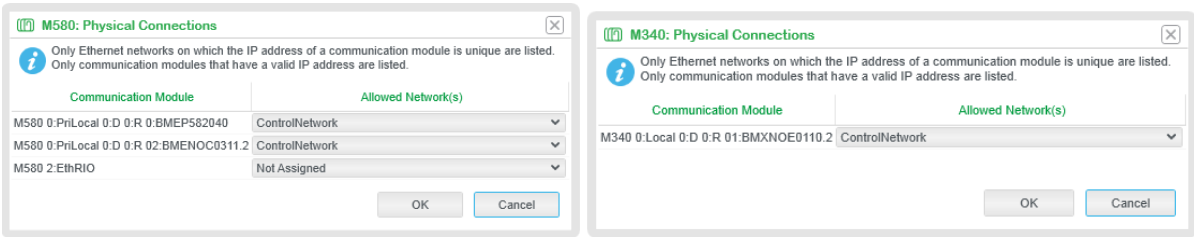


3.2 Configuration of the platform – Topology Manager

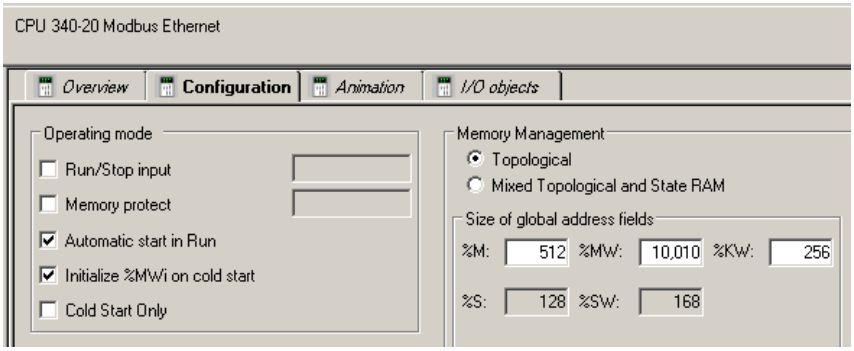
In this stage it is necessary to configure the M580 controller, the M340 controller, the Station Node and the Control Network as usual.



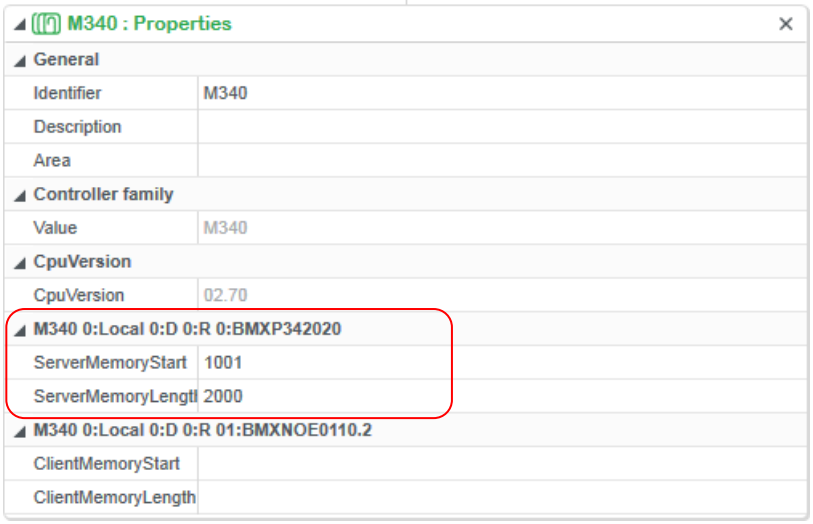
Both controllers' communication cards have to be connected to the Control Network.



Since the communication is implemented with Modbus TCP implicit messaging, it is important to remind that a certain amount of %MW has to be reserved for this communication on the server side. So the recommendation is to increase the default %MW area of the M340 controller, for example to 10k words.



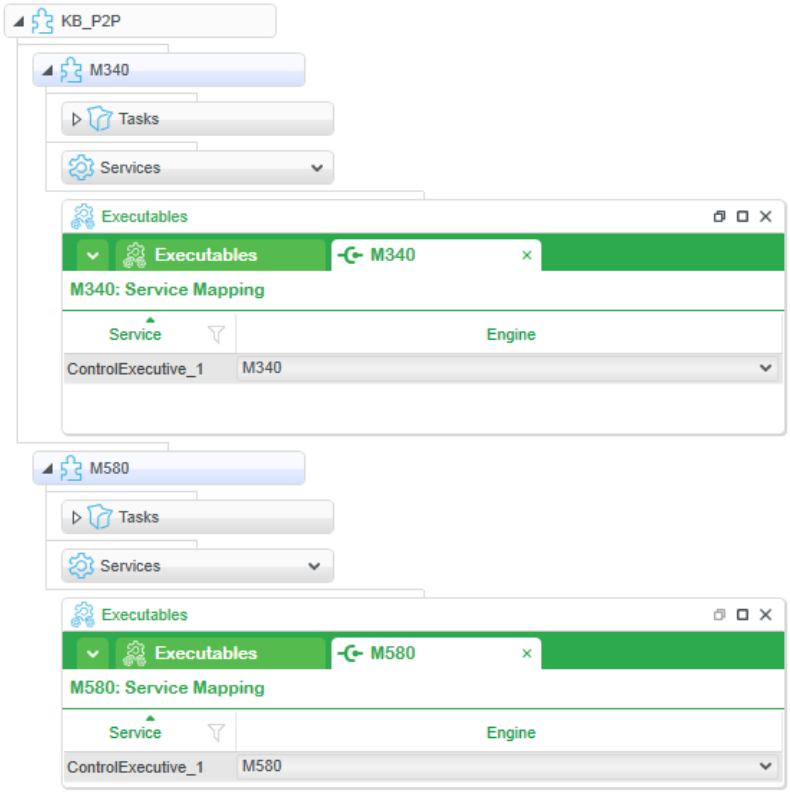
On the M340 controller properties, it is also necessary to define the area that will be used to expose information on the Modbus server. In this case, 2k words are reserver starting from %MW1001 (*ServerMemoryStart* parameter has to be odd).



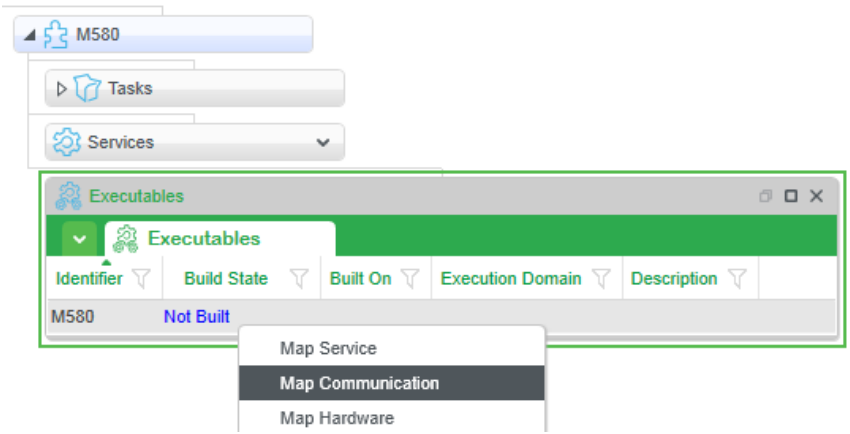
In this case the Modbus TCP IO Scanner client is an M580 which doesn't require any further configuration. This because the M580 uses Device DDT functionality instead of %MW in order to map information related to Peer2Peer communications.
In case the IO Scanner client were a M340 NOE or a Quantum NOE, it would have been necessary to configure the Client area in the same Property editor (*ClientMemoryStart* and *ClientMemoryLength* parameters).

3.3 Configuration of communication channel – Project Manager

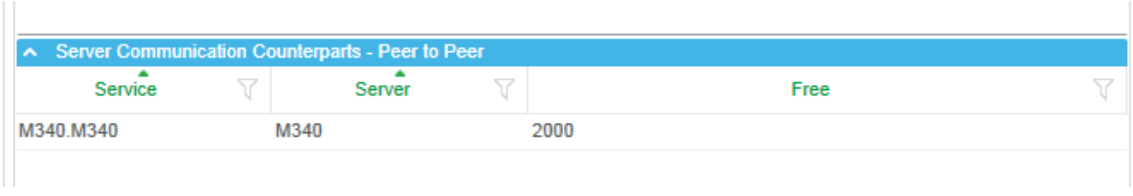
Create and map the two control projects as usual.



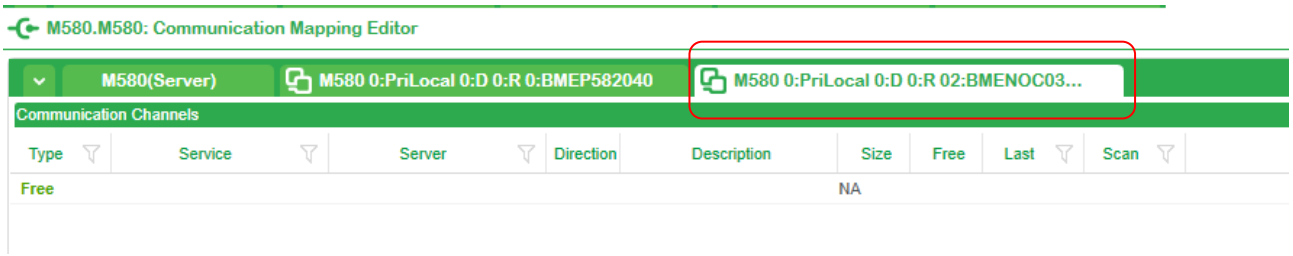
Now it is possible to define our communication channel, which will host all the network variables to exchange. Right click on M580 controller and select “Map Communications”.



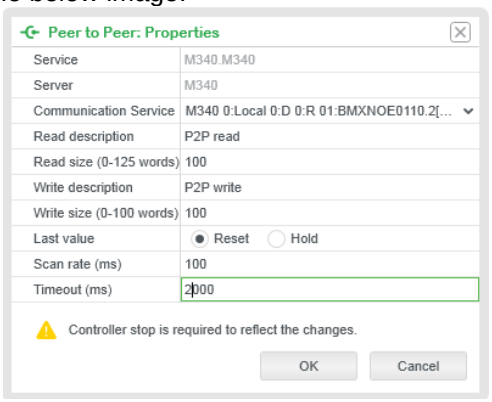
On the bottom right part of the screen, all available Modbus TCP server counterparts will appear. In this case, we can find our M340, that has 2k words available for the communication.



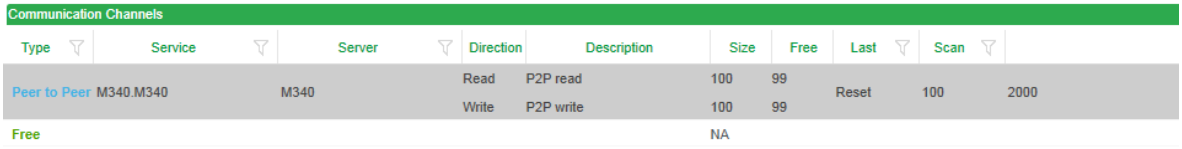
Select now the NOC module tab and drag&drop the M340 server counterpart on it.



Fill the properties as shown in the below image.



By doing this, this communication channel will host up to 100 words to be read from M340 and up to 100 words to be written to M340.

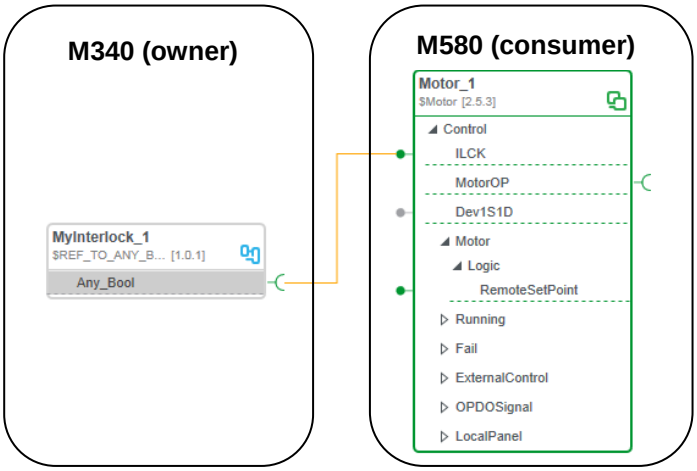


4 CASE STUDY 1: SCATTERED DATA

4.1 Definition of the case study

The objective of this case study is to show how to define network variables in a scattered data scenario without the need of any special template.

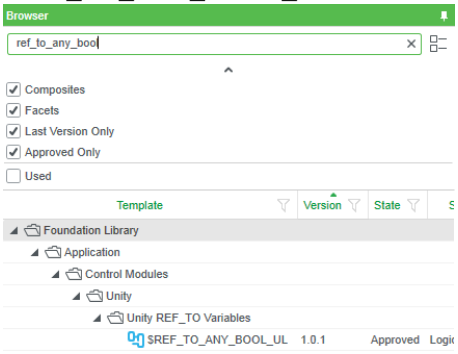
In this case, a simple Boolean variable on the M340 controller will determine the interlock of a motor assigned to the M580 controller.
For example, this variable could be set to 1 under certain conditions on the M340 code in order to achieve a temporary stop of the motor managed by the M580.



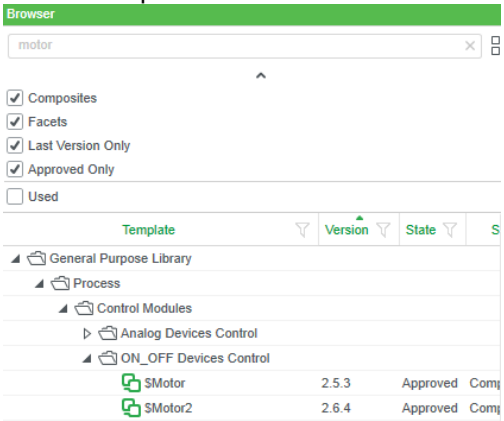
4.2 Creation of instances – Application Manager

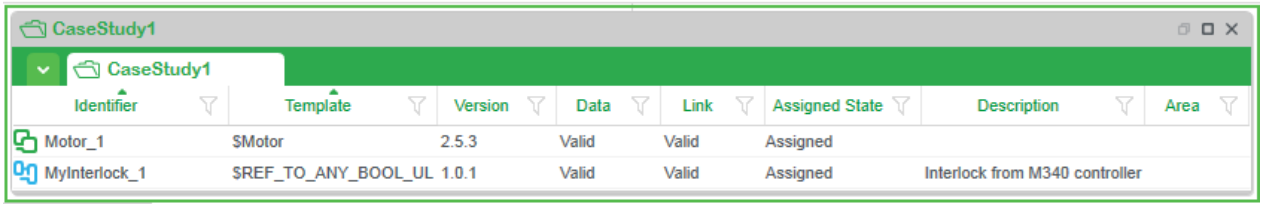
In this case study we need to define two instances:

- **$REF_TO_ANY_BOOL_UL** facet

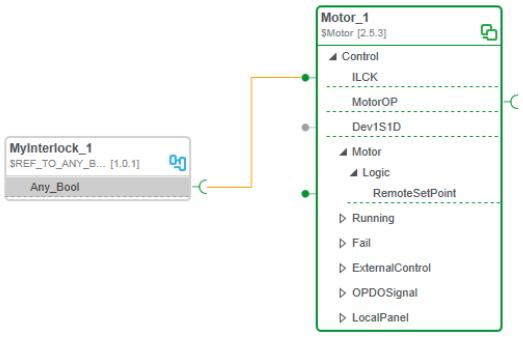


- **$Motor** template





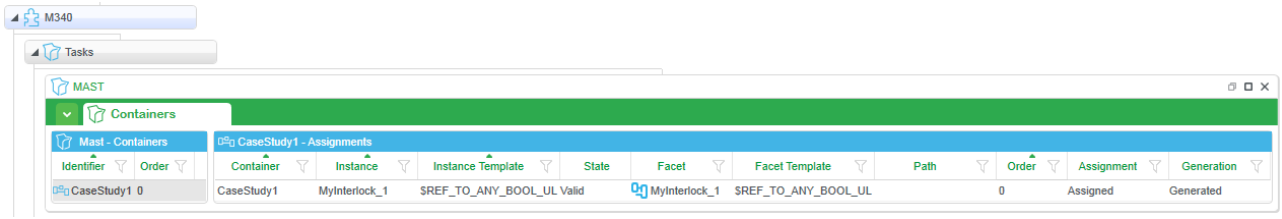
After the definition of these instances it is possible to directly link them as shown in the case study presentation.



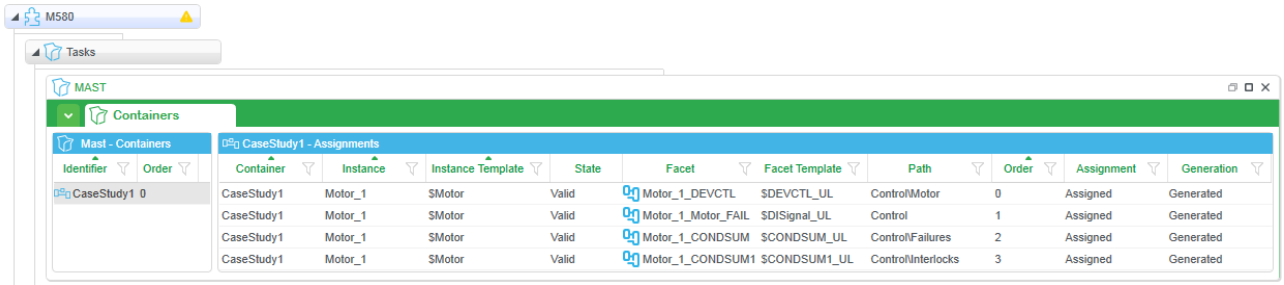
4.3 Generation and mapping of network variables – Project Manager

Assign and generate the two instances to the relevant control projects:

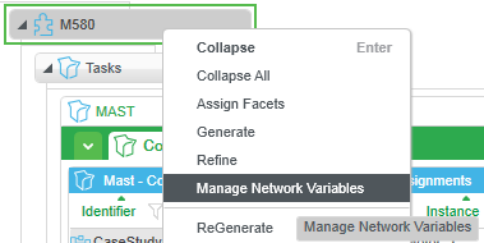
- **$REF_TO_ANY_BOOL_UL** instance on M340



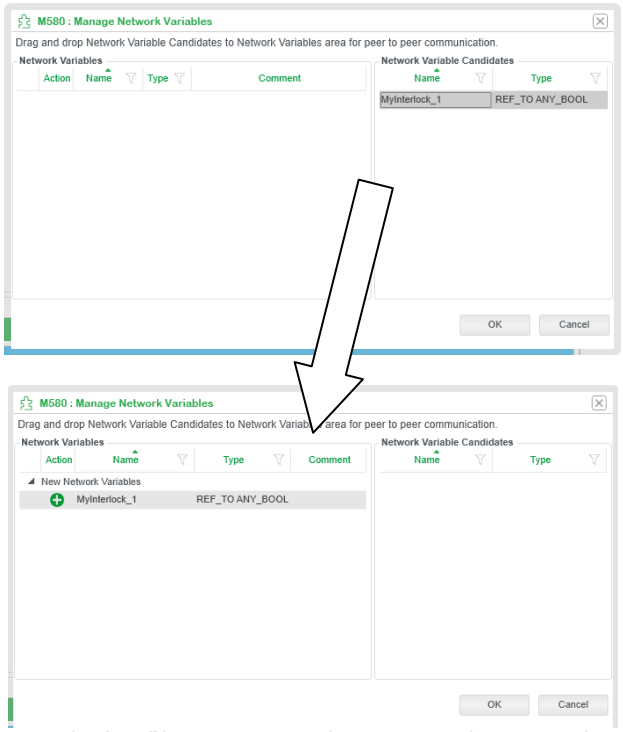
- **$Motor** instance on M580



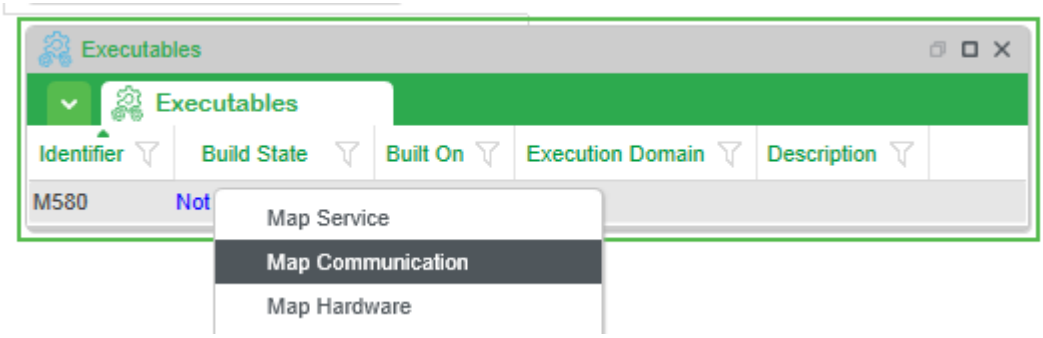
After this step, right click on M580 (consumer project) and select “Manage Network Variables”.



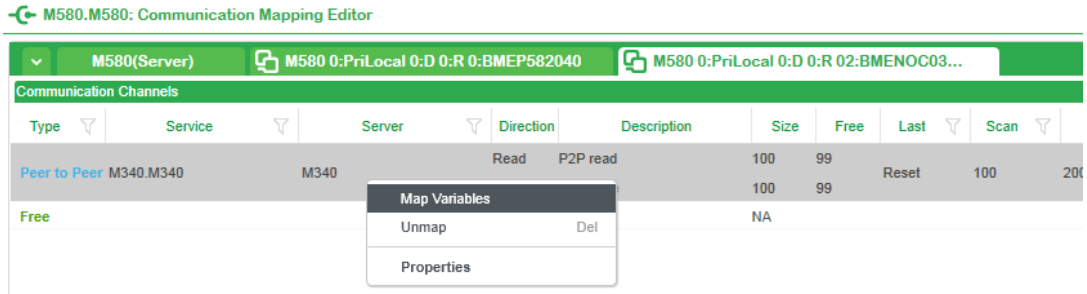
"*MyInterlock_1*" variable will appear in the Candidates box. Drag&drop it to the left part of the popup.



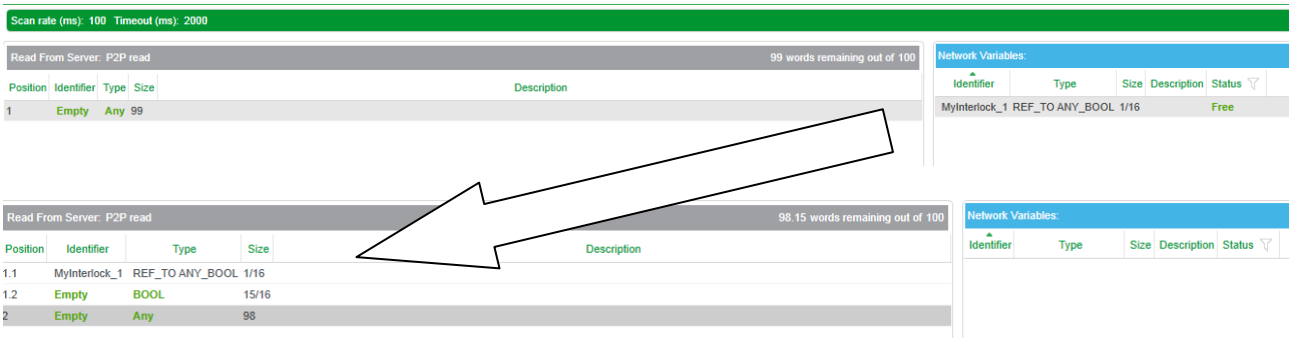
The network variable for "*MyInterlock_1"* is now created. Next step is to map it on the Peer2Peer communication channel.
To do this, right click on M580 project and select "Map Communication"



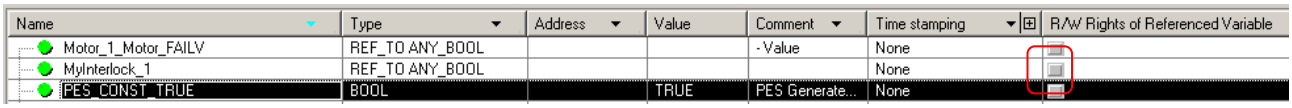
Go to the NOC tab, right click on the Peer 2 Peer channel previously created and select "Map Variables"



MyInterlock_1 will appear in the Network Variables list. Drag&drop it to the left part of the window, which is the stack of the variables included in the communication channel.

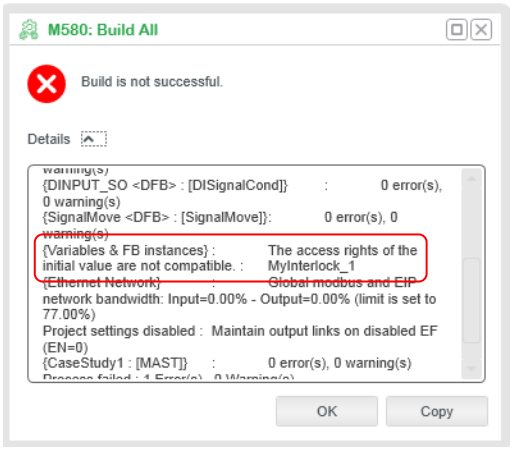


Before to build, refine the control project of M580 and untick the **R/W Rights of Referenced Variable** attribute (see following excerpt from PES User Guide which states the reason)



NOTE: The software does not set the **R/W Rights of Referenced Variable** attribute for REF_TO type network variables that it creates. If it needs to be enabled, you need to refine the consumer project and set the attribute manually *(see Process Expert, Control Participant Services, User Guide)* in the variable properties window of the Control Participant.

If not unchecked, this property will lead to a Build error.



Build both control projects.

4.4 Results

After the build on M340 (owner), *MyInterlock_1* variable is referenced to a %MW address that will be scanned by the NOC.

MyInterlock_1	REF_TO ANY_BOOL		REF(%MW1002.0)

After the build on M580 (consumer), *MyInterlock_1* variable is referenced to the proper item in the Device DDT, which is automatically created from the DTM manager.

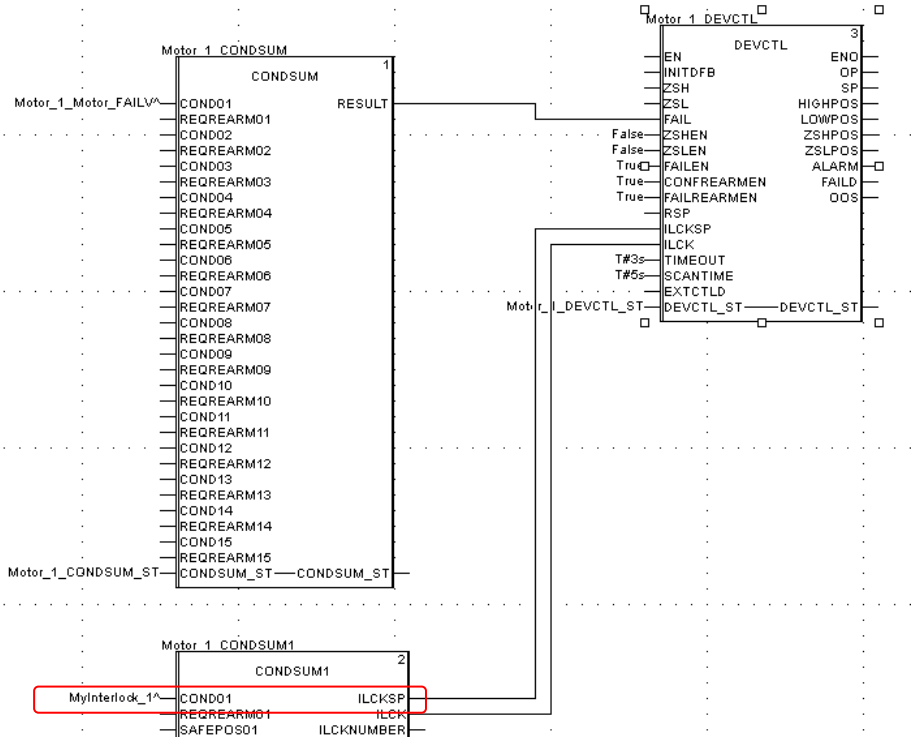
MyInterlock_1	REF_TO ANY_BOOL		REF(M3401_BMXNOE011022.Inputs.Free[0].0)

< 192.168.1.15 > BMENOC0311_2
< Modbus:192.168.1.21 > M3401_BMXNOE011022

Connection Bit	Unit ID	Health Time Out(ms)	Repetitive Rate(ms)	RD Address	RD Length
1	255	2000	100	1002	99

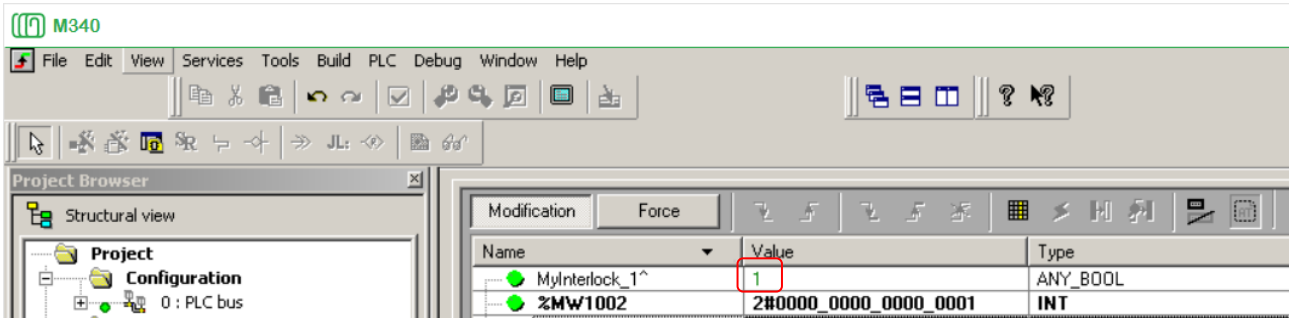
M3401_BMXNOE011022	T_M3401_BMXNOE011022
Freshness	BOOL
Freshness_1	BOOL
Inputs	T_M3401_BMXNOE011022_IN
Free	ARRAY[0..98] OF INT
Outputs	T_M3401_BMXNOE011022_OUT
Free	ARRAY[0..98] OF INT

MyInterlock_1 is then linked to the proper Interlock pin on the Motor section.

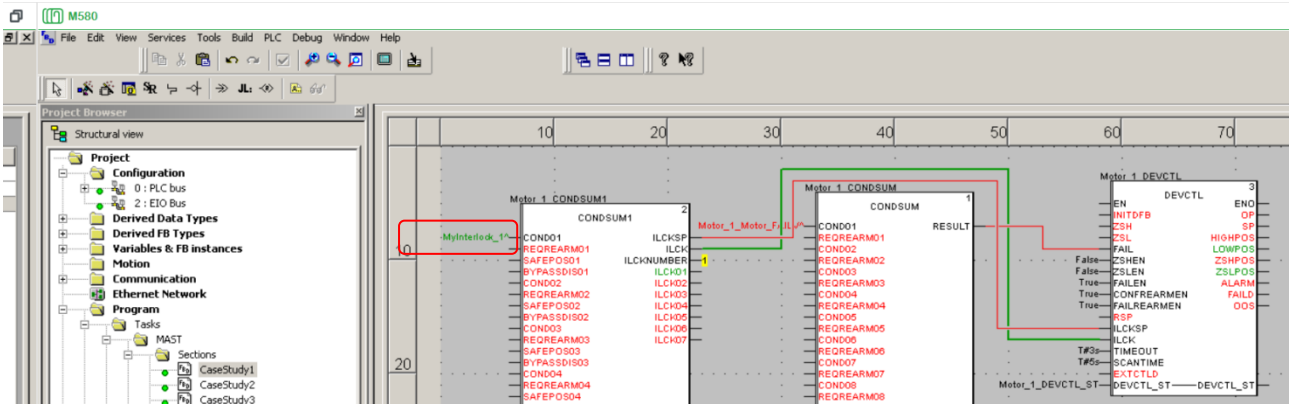


4.5 Testing with Hardware

Running in **M340** (Owner)



Running in **M580** (Consumer)

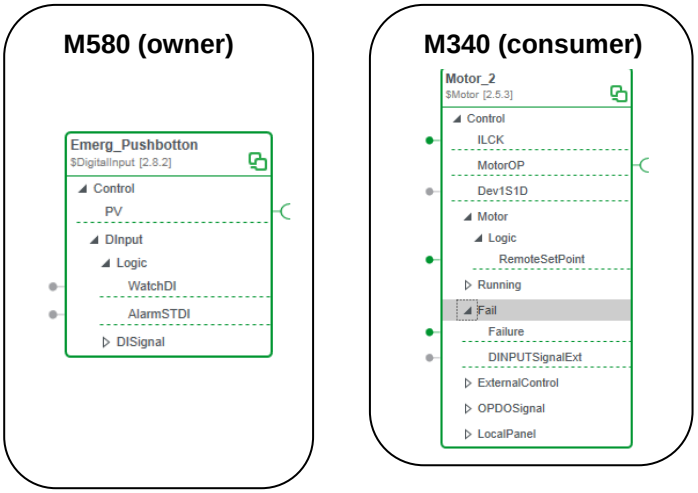


5 CASE STUDY 2: SCATTERED DATA WITH DATA IN_OUT FACETS

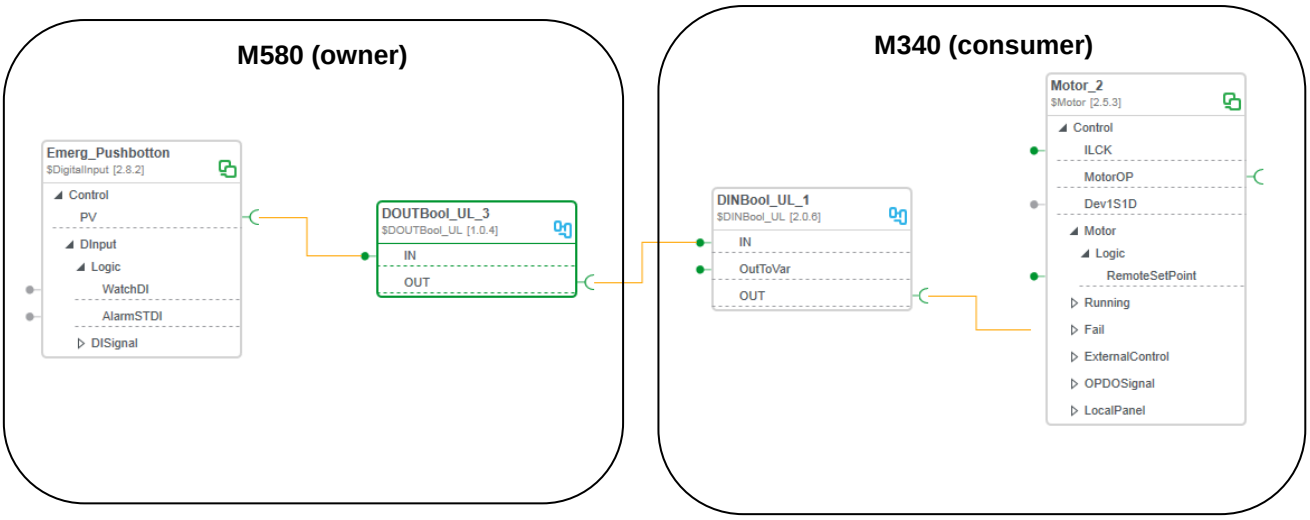
5.1 Definition of the case study

The objective of this case study is to show the interactions between to objects placed on different controllers.
When it is necessary to connect composite template instances belonging to different controllers, it is necessary to use the **DATA IN_OUT** templates.

For example, in this case the PV of a *$DigitalInput* block on M580 (which acts as owner in this case) should determine the Failure of a *$Motor* template on the M340 (which acts as consumer).

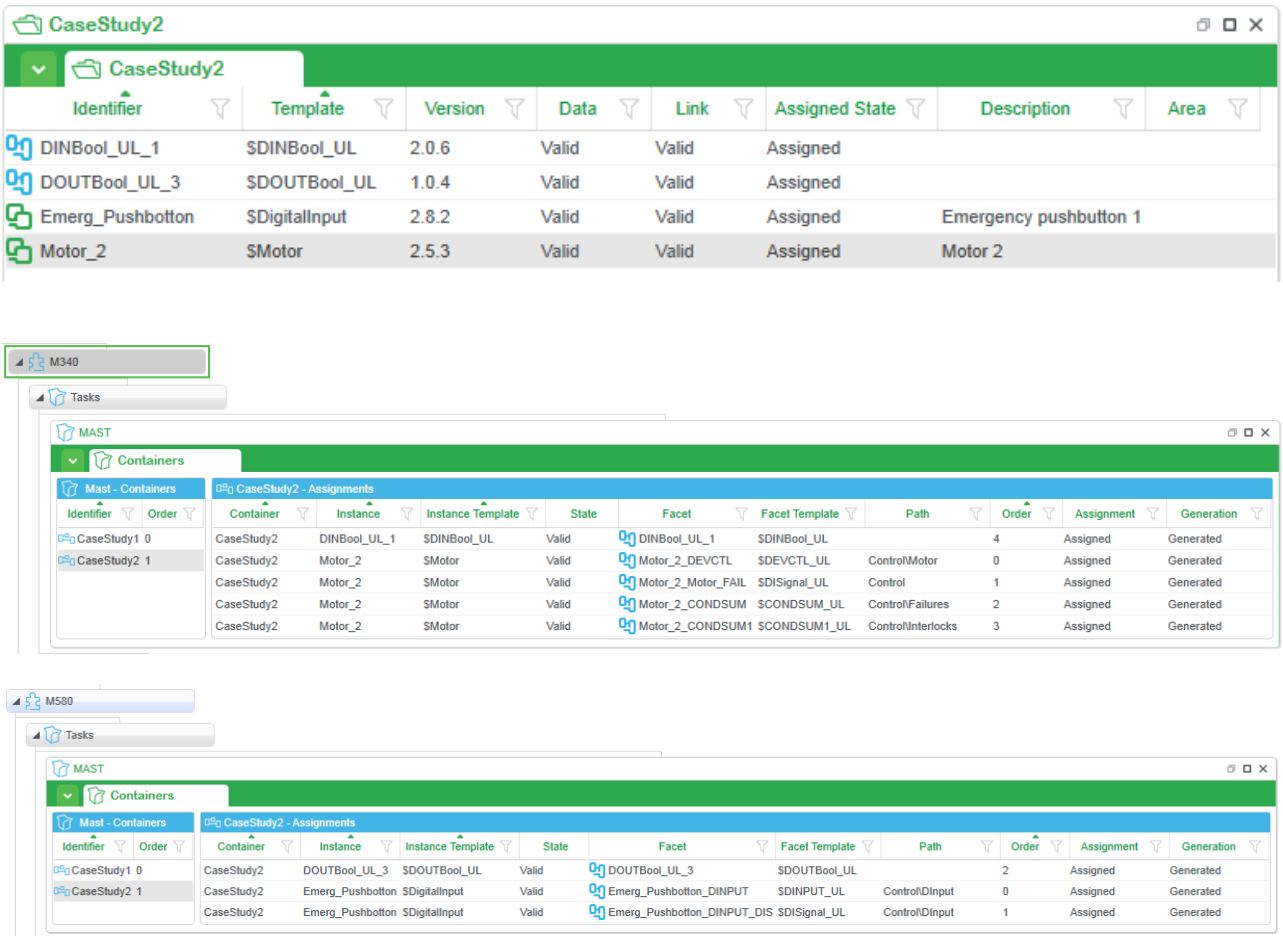


In this case it is possible to create instances of *$DOutBool_UL* and *$DINBool_UL* facets instances that links the objects between them.



5.2 Creation of instances – Application Manager

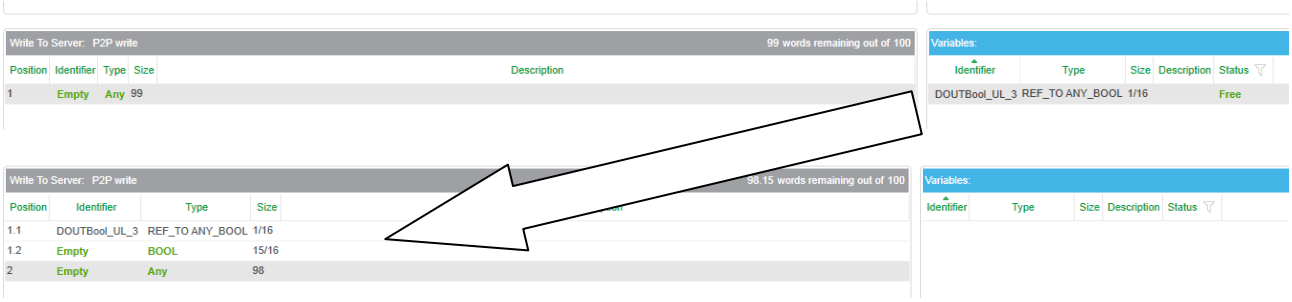
Create the instances, link and assign them to the relevant control projects.



Generate the control project

5.3 Mapping network variables – Project Manager

After the generation, it is possible to map the network variable on the M580 “Map communications” editor, no need to enter in “Manage Network Variables” editor (this thanks to the usage of DATA IN_OUT template).
In this case the *DOUT_Bool_UL_3* variable has to be written in the M340 and it will appear in the IOScanner Write section.

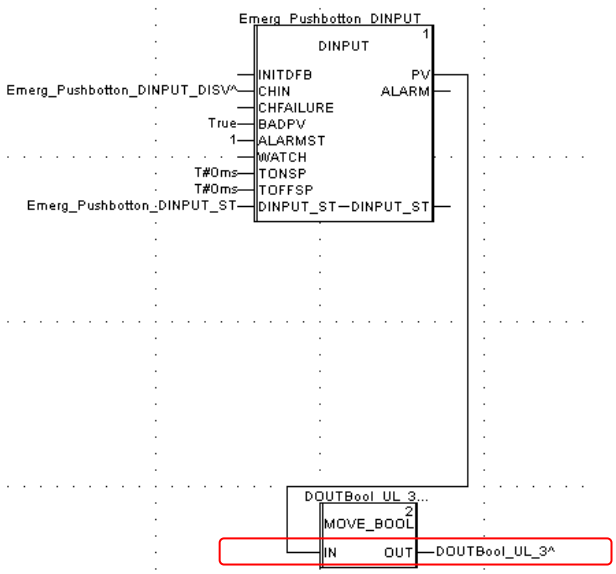


5.4 Results

After the build on M580 (owner), the *DOUTBool_UL_3* is mapped on the Device DDT output area to be written on M340.

DOUTBool_UL_3	REF_TO ANY_BOOL		REF(M3401_BMXNOE011022.Outputs.Free[0].0)

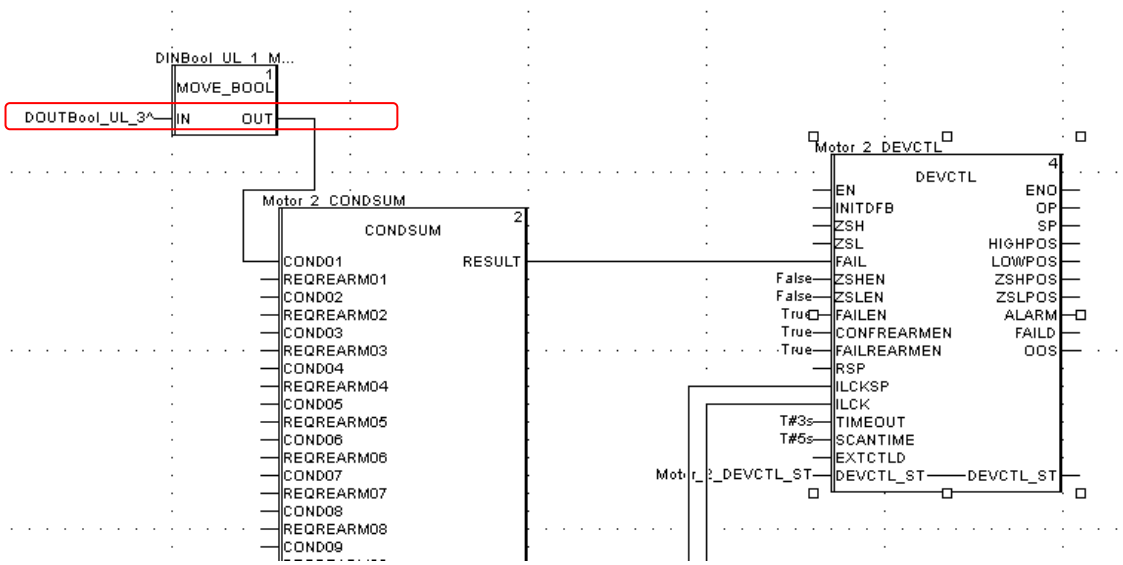
This variable will transfer the value of the PV of the Emergency Pushbutton DINPUT block.



After the build on M340 (consumer), *DOUTBool_UL_3* variable is referenced to a %MW address that will be written by the M580 NOC.

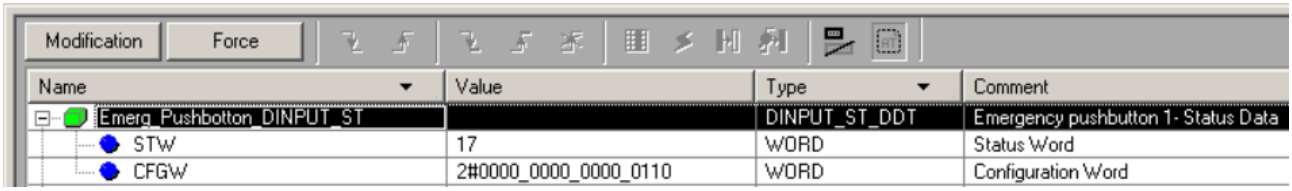
DOUTBool_UL_3	REF_TO ANY_BOOL		REF(%MW1102.0)

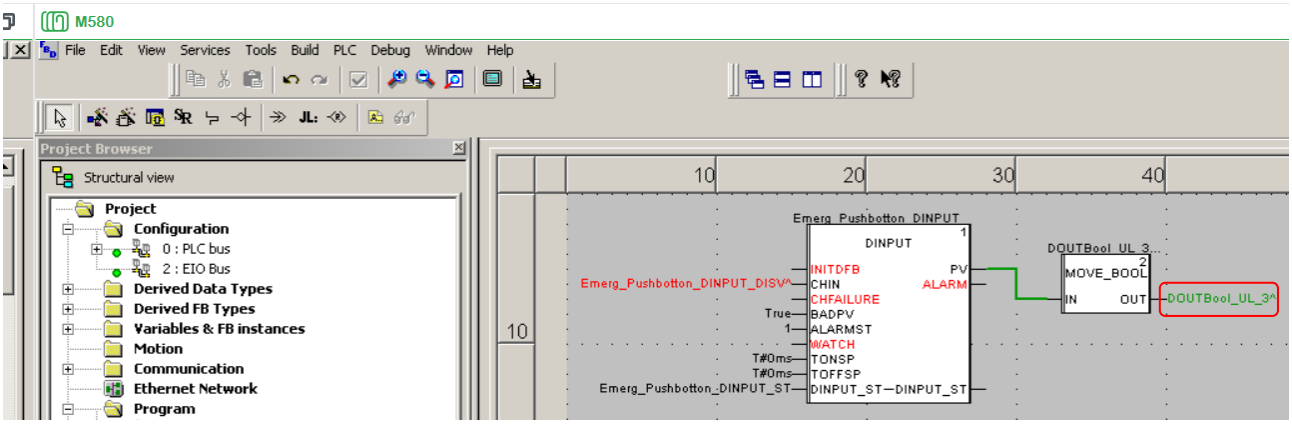
The *DOUTBool_UL_3* variable is then connected to the Failures block associated to the motor



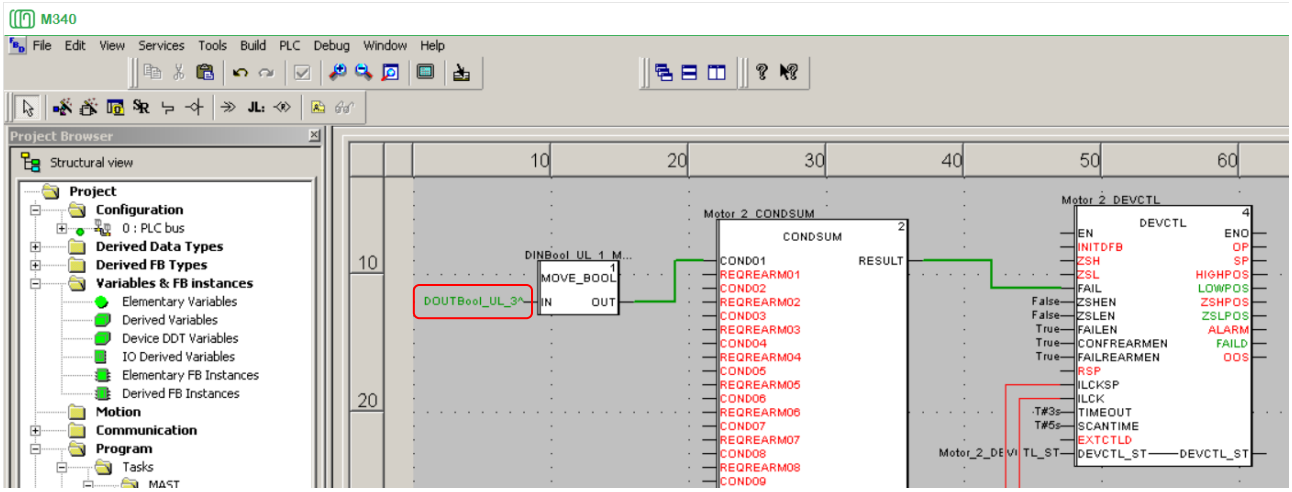
5.5 Testing with Hardware

Running in **M580** (Owner)





Running in **M340** (Consumer)



6 CASE STUDY 3: OWNER / CONSUMER TEMPLATES

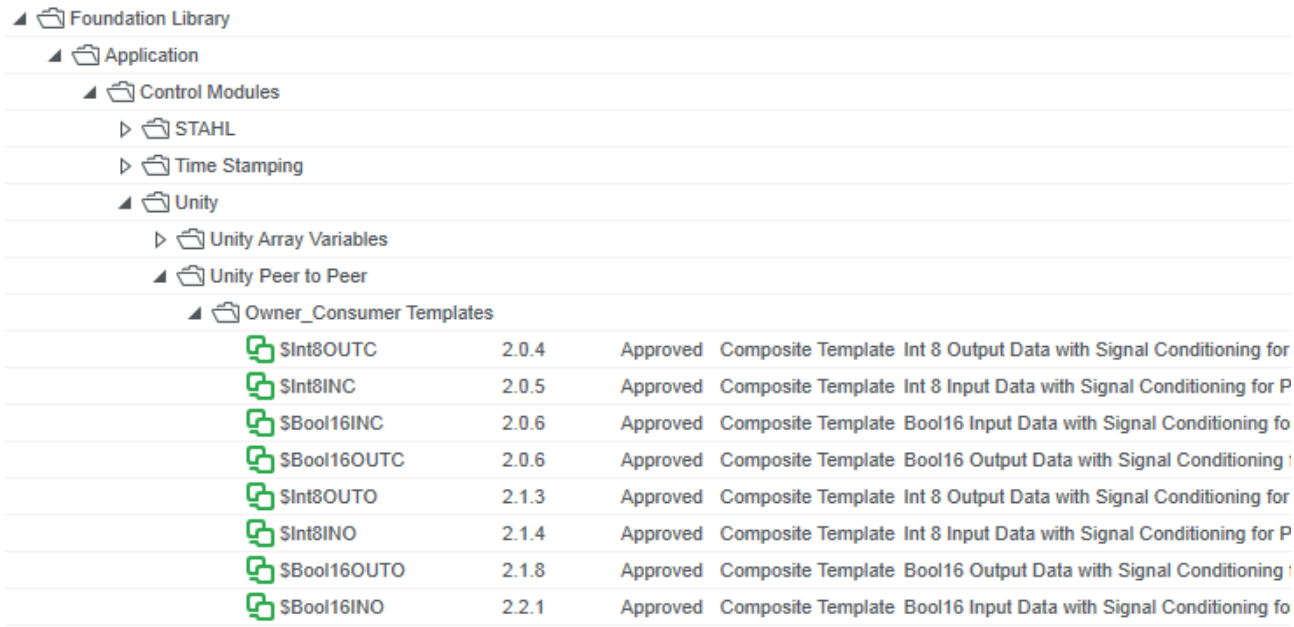
6.1 Definition of the case study

This case study considers the exchange of physical I/O data between controllers.

The need behind this case study is to use a controller (or some I/O modules belonging to a controller) as an intelligent distributed I/O system.
The typical application is a central consumer project (typically M580 or Quantum) managing control objects having field I/Os distributed on an M340 I/O system with BMXPRA0100 (or BMXCPU) processor.

In this example M340 is acting as the owner: it means that it has some physical I/Os connected to its I/O system that are required from control objects belonging to another controller, which is the M580 having the role of the consumer.
More in detail, the M340 has 4 analog input signals that need to be transferred to the M580.

The right templates to use in this case study are the **Owner_Consumer** Templates, available under the Foundation Library.



These templates include a set of Hardware Mapping facets and an interface that can be connected to the Peer2Peer partner.

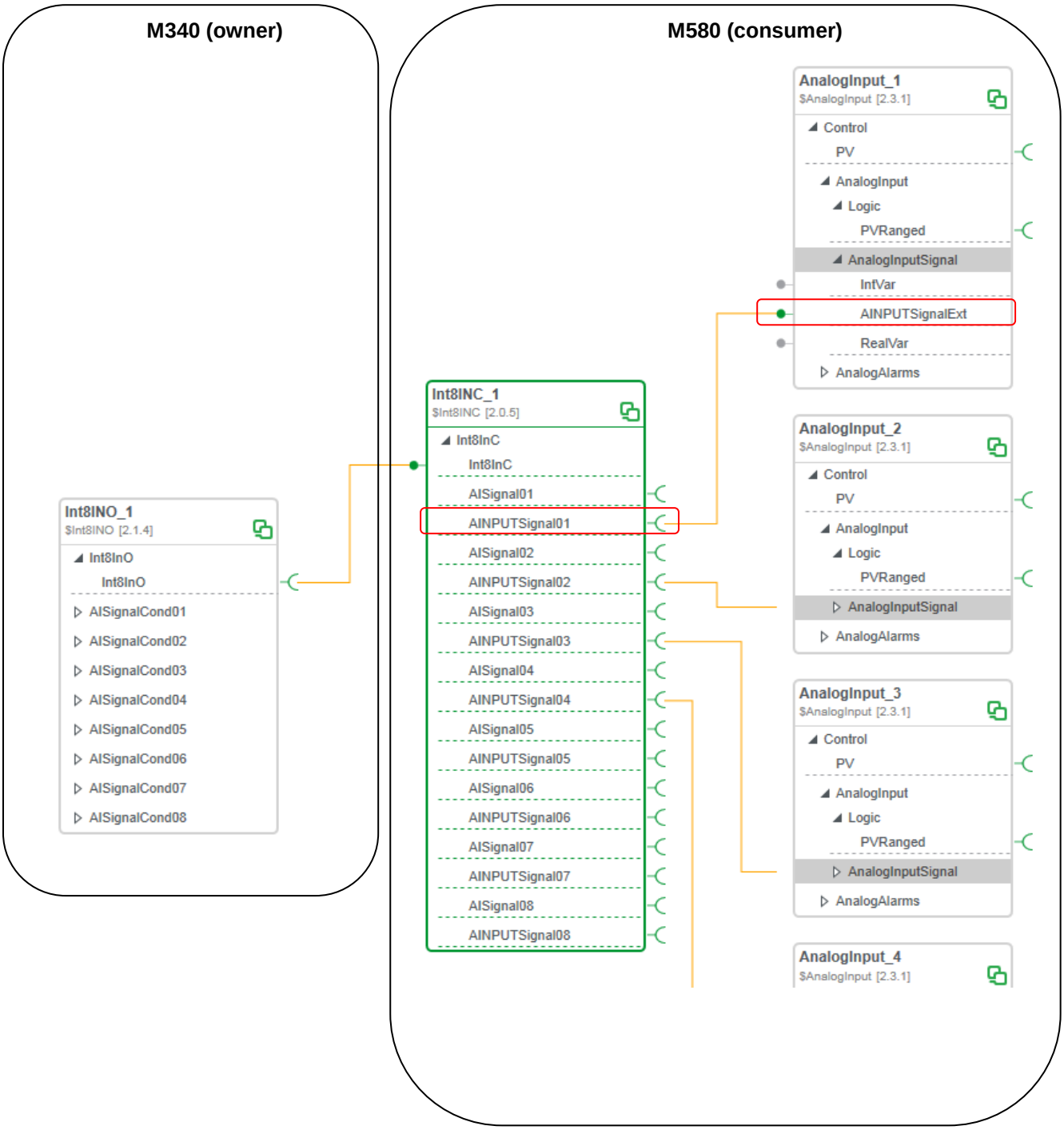
When analog data (input or output) has to be exchanged, use **$Int*** templates.
When digital data (input or output) has to be exchanged, use **$Bool*** templates.

The control project hosting the connection of the physical signal is the Owner, so **$*O** has to be used
The control project hosting the control object using the signal value is the Consumer, so **$*C** has to be used.

To complete the choice of the right template, it is necessary to choose whether it is **$*IN*** or **$*OUT***.
The choice must be made with the perspective of the signal Owner.

Based on the above considerations, in this example the M340 hosts an *$Int8INO* object, the M580 hosts an *$Int8NC* object. These to object are mutually connected.

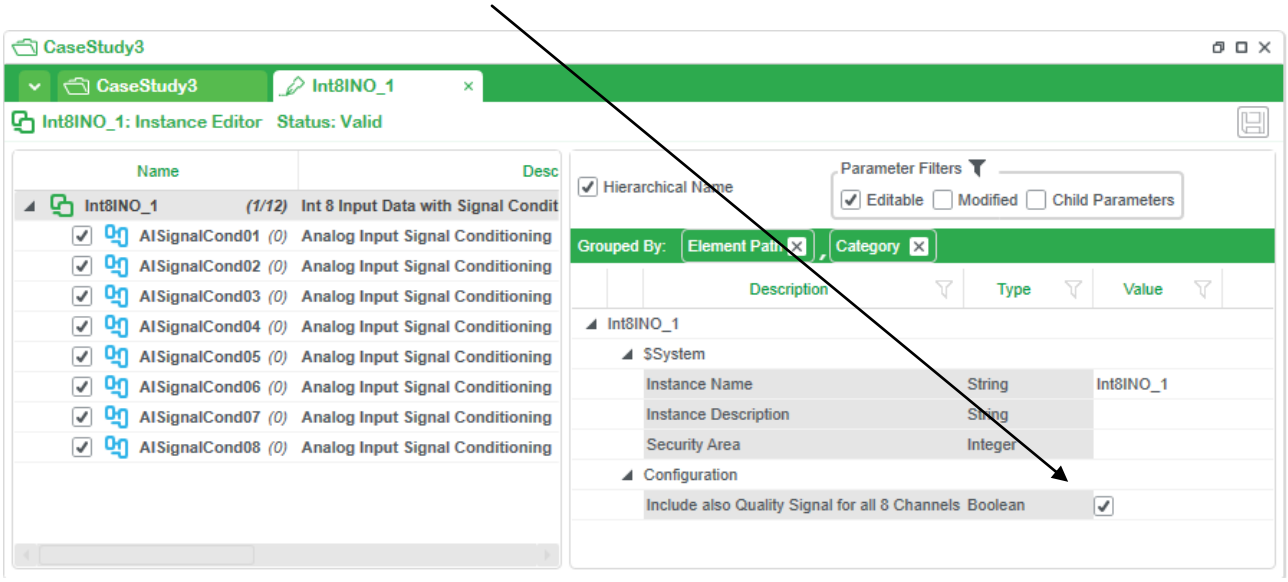
The *$Int8NC* object has then to be connected to the relevant control objects, for example *$AnalogInput* instances. Since the Signal Conditiong has been enabled, the right interface to use is ***AINPUTSignalX*** **←→ AINPUTSignalExt**.
In case the signal conditioning is not enabled, the interface to use would be **AISignalX ←→ IntVar**.



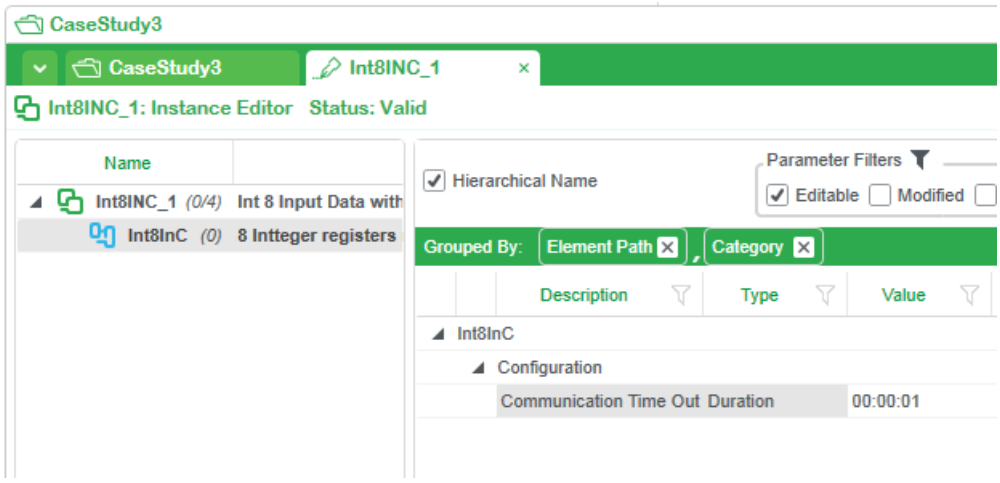
6.2 Creation of instances – Application Manager

Create instances in the Application Manager for the above mentioned objects.

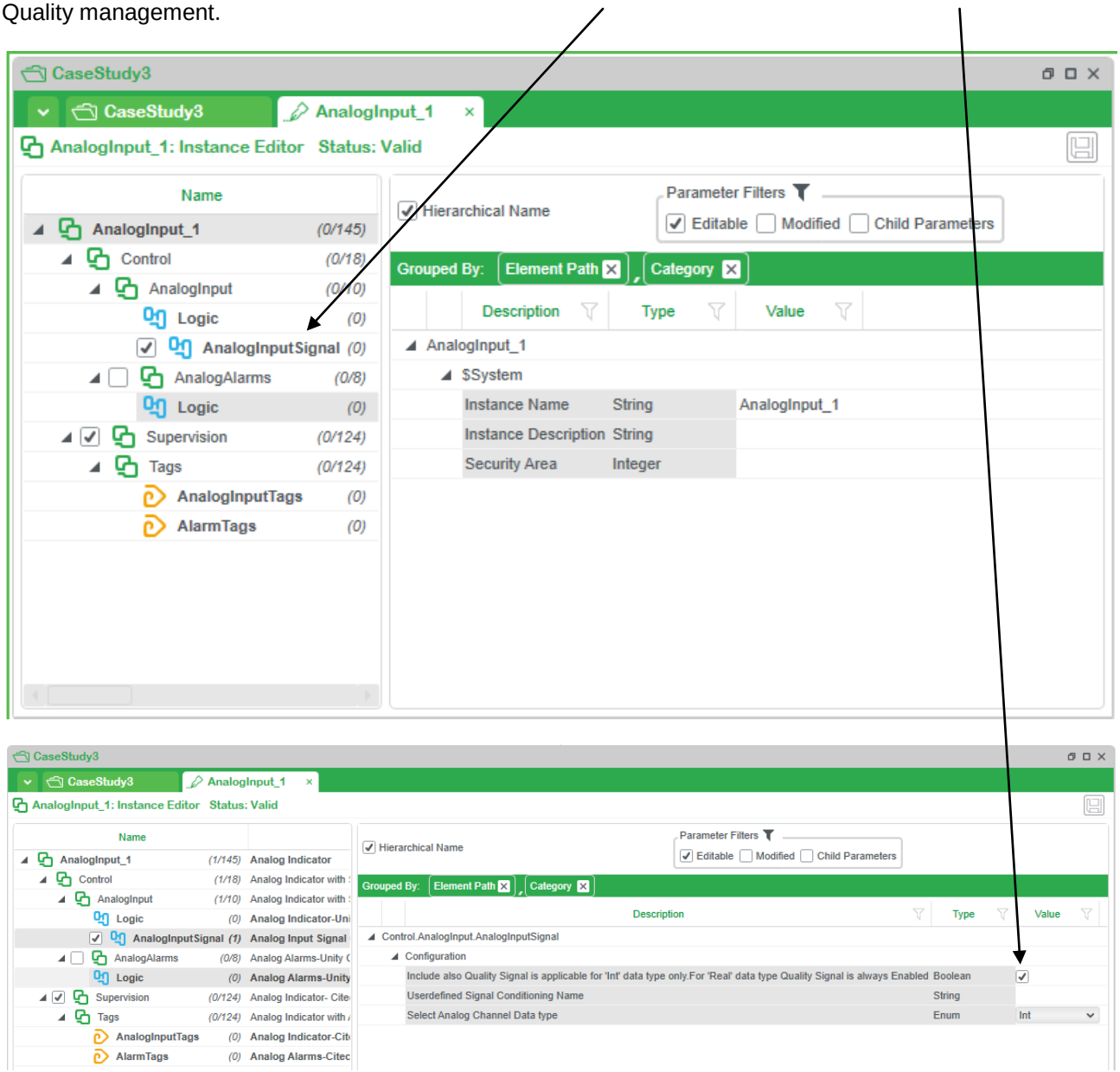
The $*Int8INO* object contains up to 8 physical Analog Input signals that can be mapped on the owner control project.
It is recommended to enable the Signal Quality management.



The $*Int8INC* template has the configuration of the *Communication Time Out*: this is not the IO Scanner timeout, but a timeout managed at application level. The IN_OUT templates include the management of the quality of the signal, which allows defining the bad value / fallback value in case of no data refresh for a certain period.



For $*AnalogInput* template, it is recommended to mark the *AnalogInputSignal* hardware facet with Signal Quality management.



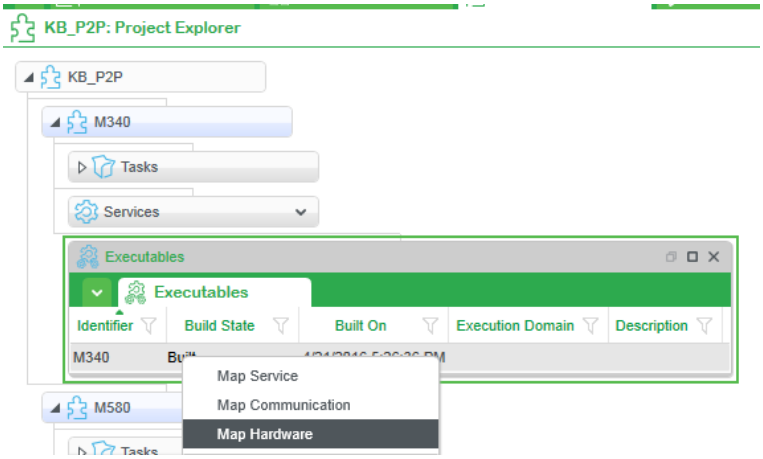
This will allow the proper connection coming from $*Int8INC* object.

Create links between instances as shown in the previous chapter.

Assign facets to new sections on relevant controllers and generate.

6.3 Mapping hardware – Project Manager

Next step is hardware mapping that has to be done on the Owner project (M340)

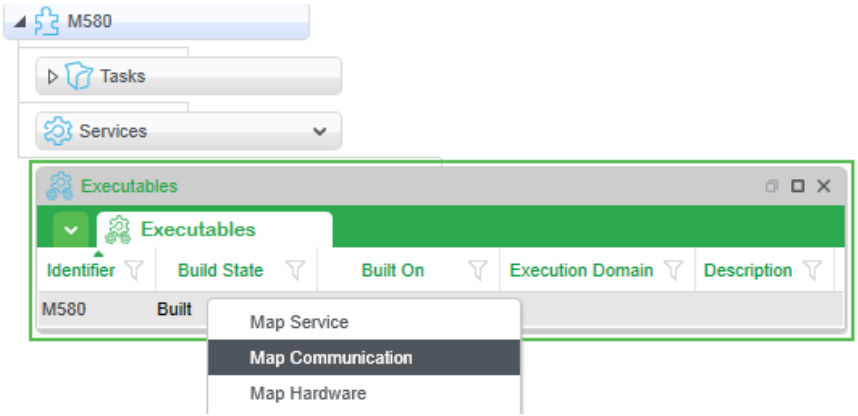


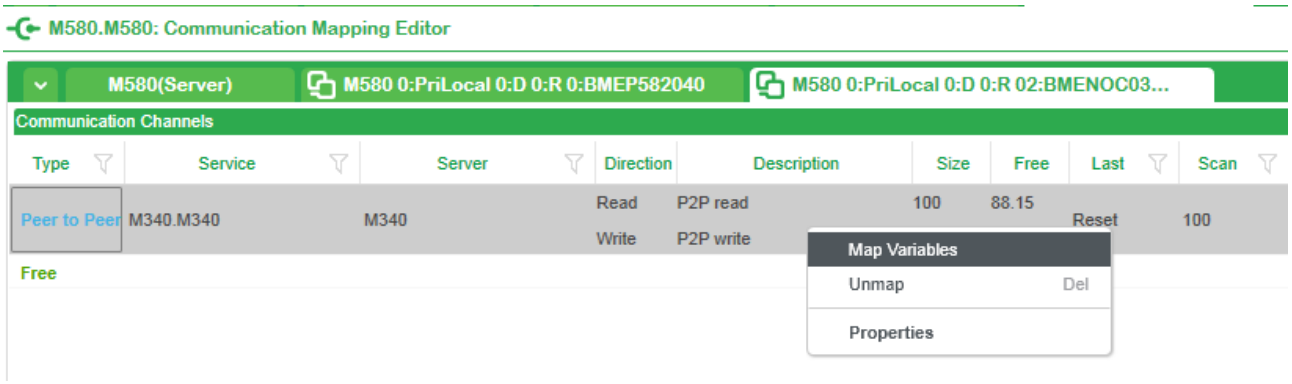
8 Analog input channels can be mapped to the BMXAMI0810 module.

M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_01.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_01	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_02.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_02	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_03.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_03	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_04.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_04	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_05.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_05	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_06.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_06	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_07.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_07	$Int8INO	AIChannel	$AIChannel/SO
M340 0:Local 0:D 0:R 03:BMXAMI0810	$MPAInput8	MPAInputCh_08.MPAInput	$AIChannel/HO	Int8INO_1	Int8INO_1_08	$Int8INO	AIChannel	$AIChannel/SO

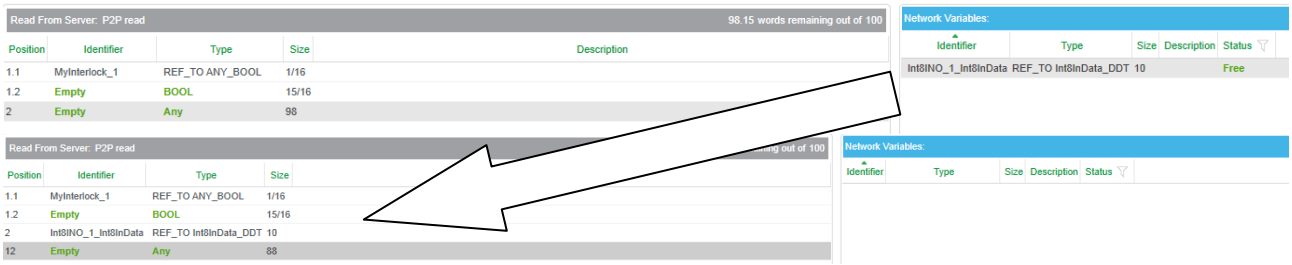
6.4 Mapping network variables – Project Manager

The mapping of the network variable has to be done on the control project which acts as IO Scanner client, in this case the M580 (NOC module).





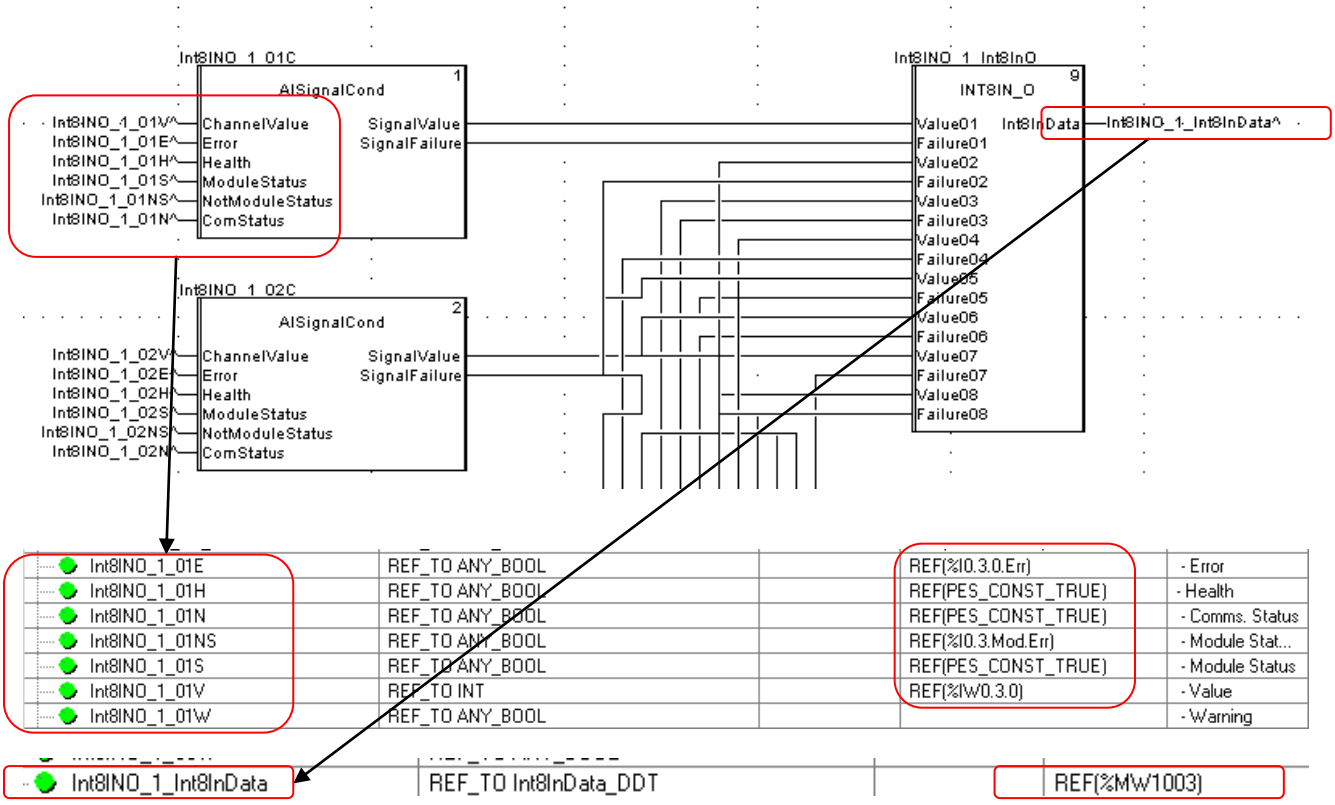
The data has to be mapped on a Read area having type ANY.
In this example we have also a BOOL area due to the implementation of the Case study 1.



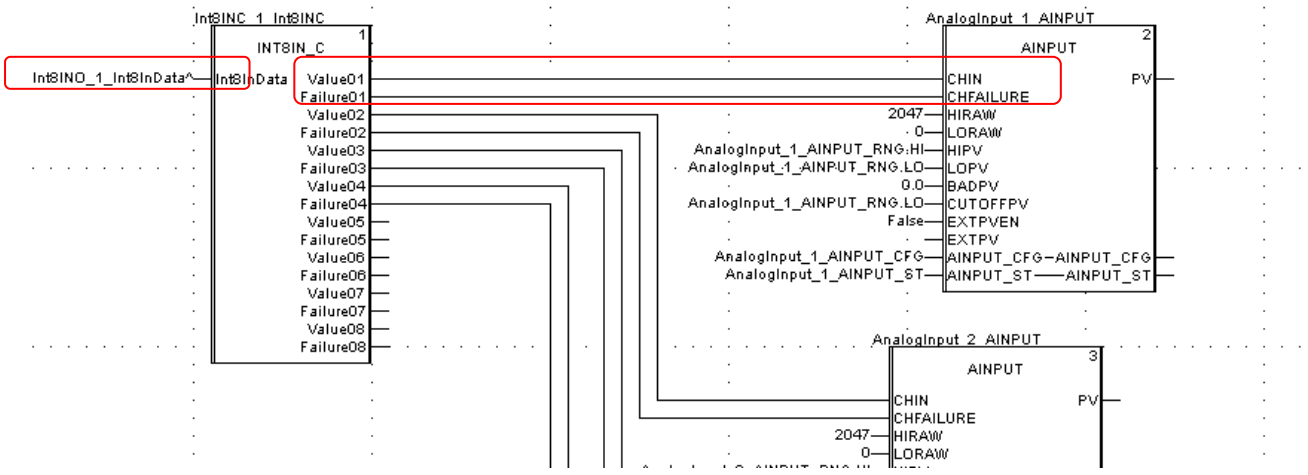
6.5 Results

Build both control projects.

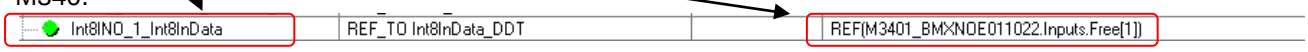
On **M340**, the *Int8INO* DFB is connected to the 8 Analog input channels with signal conditioning (mapped to the relevant %IW and diagnostic data of BMXAMI0810) and to the *InData* variable (mapped on a %MW scanned by M580).



On the **M580**, each *AINPUT* DFB is connected (signal + failure status) to the *Int8INC* DFB.

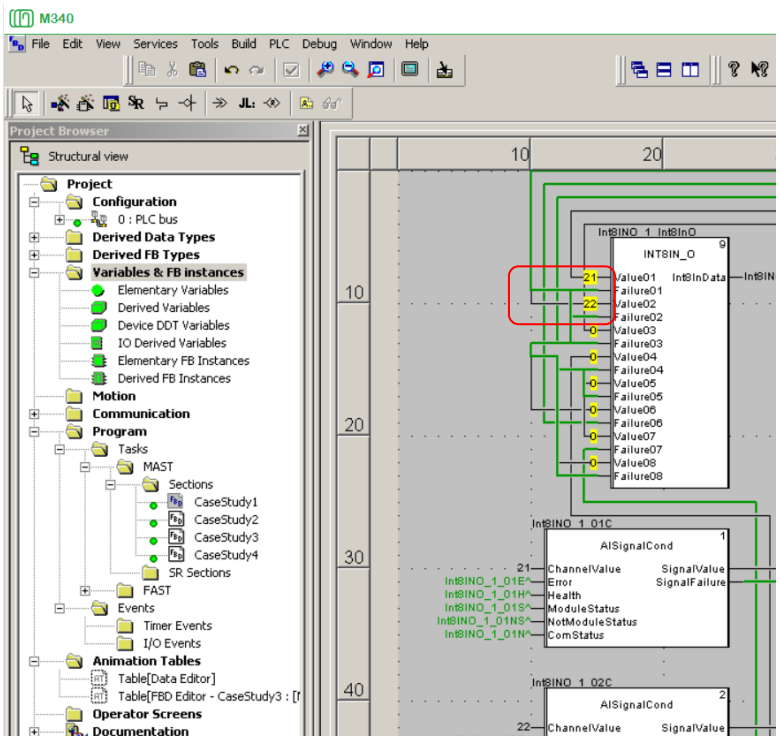


Int8INC *InData* is mapped to the Device DDT structure in charge of receiving IO Scanner data read from M340.

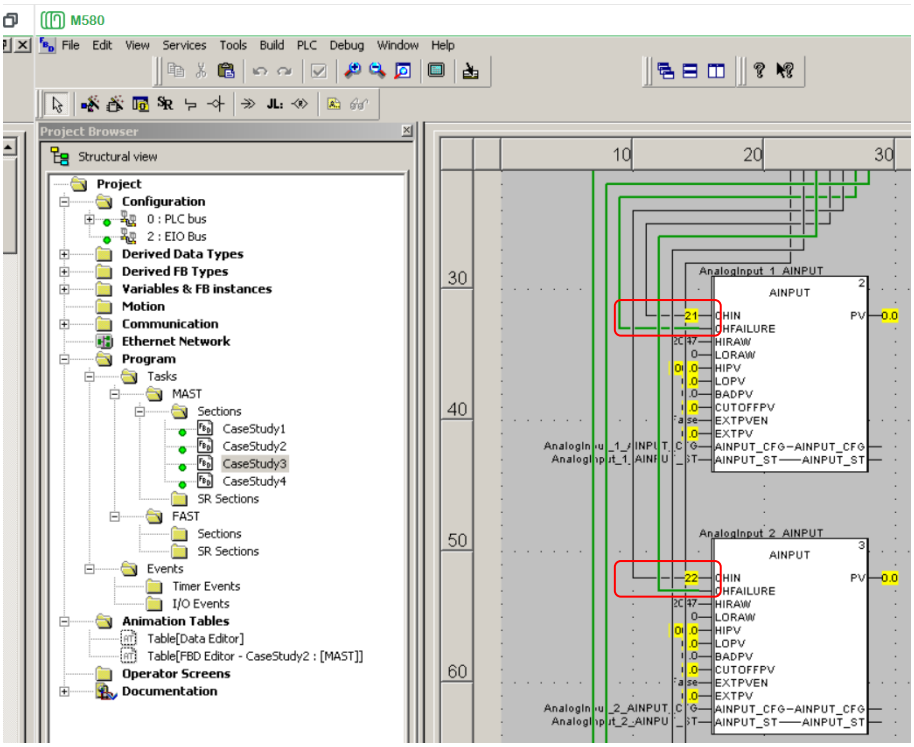


6.6 Testing with Hardware

Running in **M340** (Owner)



Running in **M580** (Consumer)



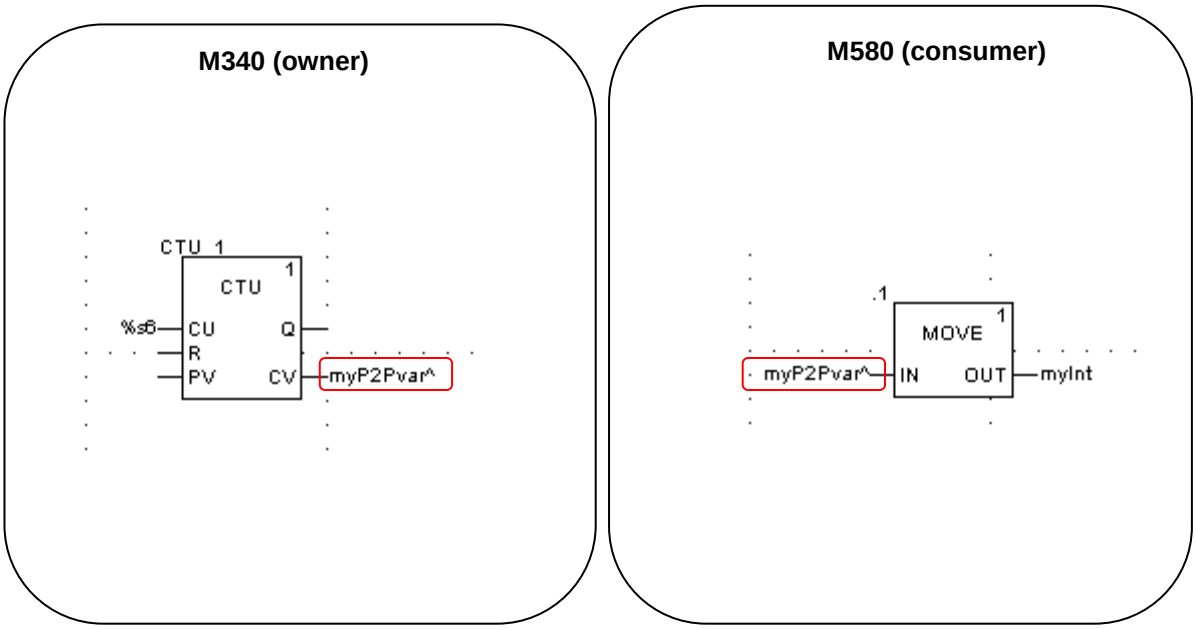
7 CASE STUDY 4: USAGE OF P2P CUSTOM ATTRIBUTE IN REFINEMENT

7.1 Definition of the case study

This case study can be analyzed and realized entirely in the project manager using refinement.

The aim of this case study is to show how to transfer control variables typically not belonging to objects between two control projects.
For example, a simple integer variable (CV of an Up Counter) is transferred from the owner to the consumer.

The variable to be exchanged in this case needs to be *REF_TO INT* (see the appendix for Network Variables definition rules).



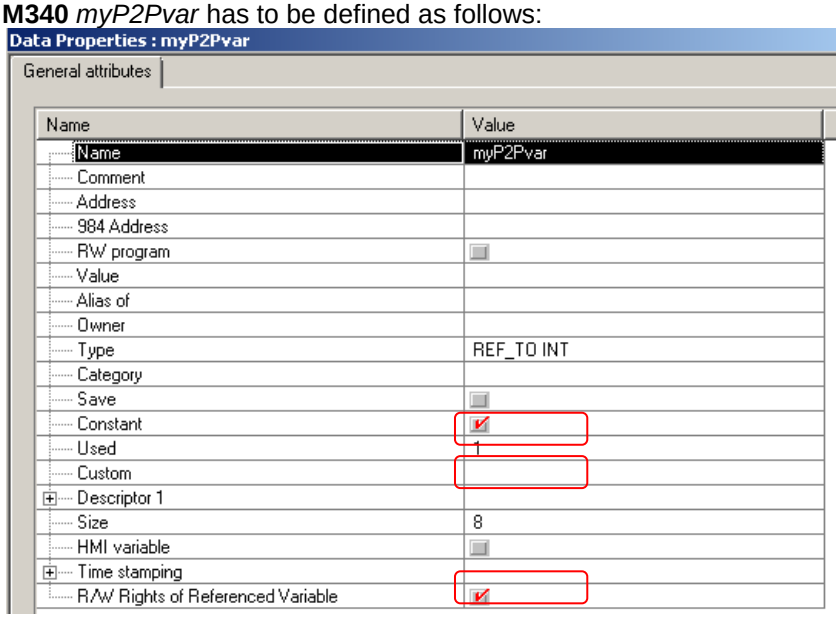
7.2 Creation of variables

It is possible to define two control variables with the same name on both control projects, and adjust the following properties:
Note that the attributes in yellow are not properly updated yet in the current Process Expert User Guide for PES 4.1.

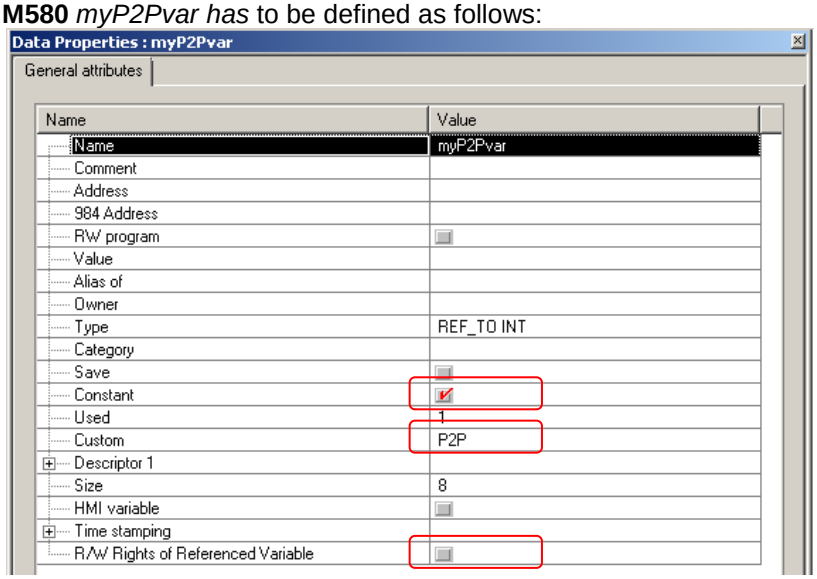
Usage	Variable attribute	Owner project/Server	Consumer project/Client
Reading data from the server	Designation of the variable in the context of Process Expert	Variable	Network variable
	Name	Any valid name	Same name
	Type	Reference data type (REF_TO) only	Same type
	Custom	Blank	P2P
	Constant	True (selected)	True (selected)
	R/W Rights of Referenced Variable	True (selected)	False (cleared)
Writing data to the server	Designation of the variable in the context of Process Expert	Network variable	Variable
	Name	Any valid name	Same name
	Type	Reference data type (REF_TO) only	Same type
	Custom	P2P	Blank
	Constant	True (selected)	True (selected)
	R/W Rights of Referenced Variable	True (selected)	True (selected)

According to this table, it is necessary to define the variables in refinement mode on M340 (owner) and on M580 (consumer)

M340 *myP2Pvar* has to be defined as follows:

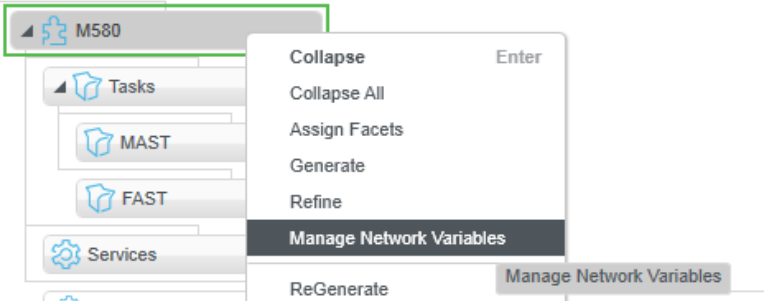


M580 *myP2Pvar has* to be defined as follows:

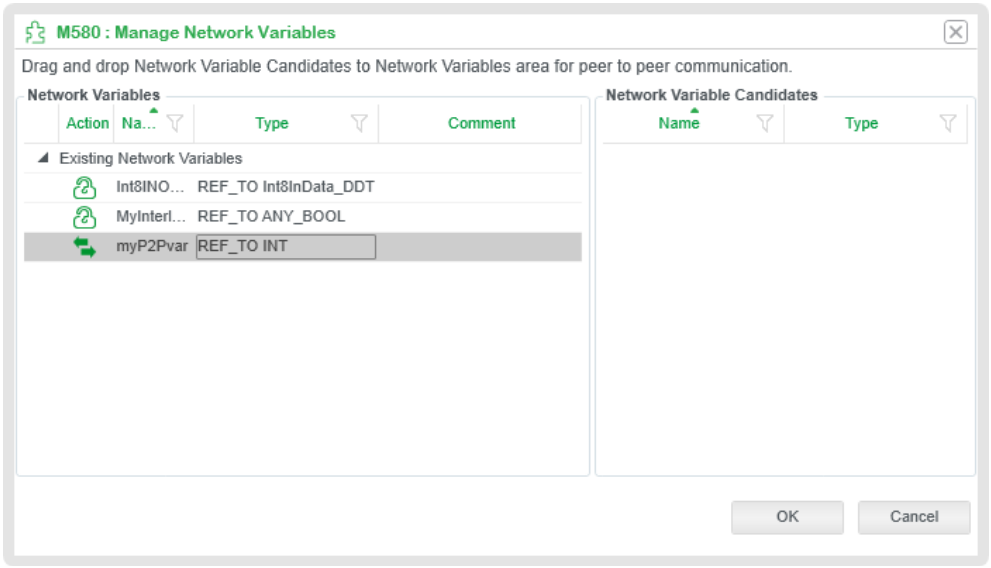


After creating the variables, create an FBD section on both controllers and add the code shown in chapter 7.1

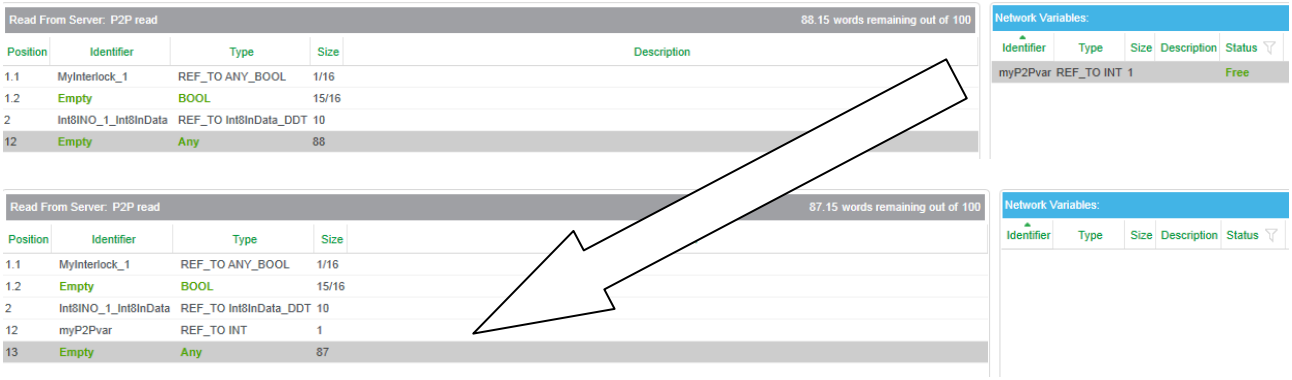
Once the refinement actions are completed on both projects, right click on Consumer project (M580) and select “Manage Network Variables”



myP2Pvar will properly appear in Existing Network Variables.



Add the *myP2Pvar* network variable to the communication channel (Map communications ➔ Map variables on consumer M580 project)



Build both control projects.

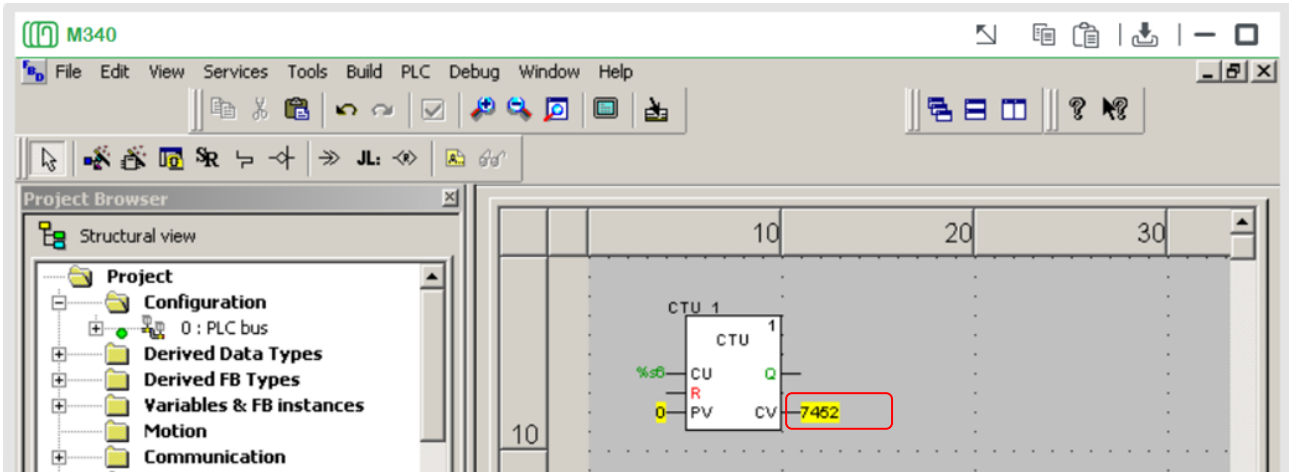
7.3 Result

Here the result after build on M340 (consumer). *myP2Pvar* is mapped on %MW address scanned by consumer (M580)

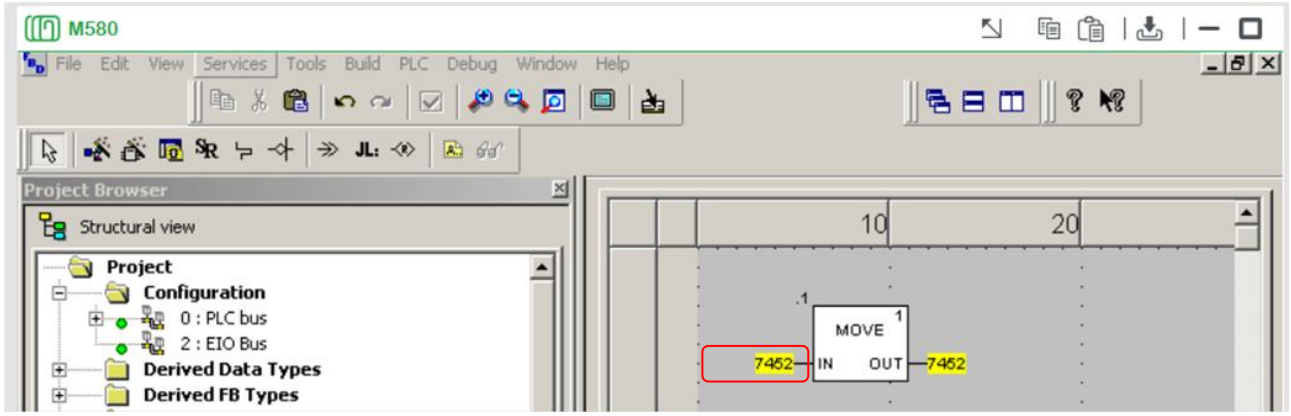
Here the result after build on M580 (consumer). *myP2Pvar* is referenced to the Device DDT in charge of receiving data from IO Scanner.

7.4 Testing with Hardware

Running in **M340** (Owner)



Running in **M580** (Consumer)



8 CONCLUSIONS

8.1 Summary of choices

Peer2Peer implementations in PES allow fulfilling multiple needs and use cases.

Here a summary of the use cases described in this article.

Case study	Description	Engineering environment	PES library features	Typical use case	PROs	CONs
1	Scattered data: facet to composite	Application manager + Project manager (refinement)	-	Development of application using objects approach	No need of templates, easy to define in AM	Need of manual refinement
2	Scattered data: composite to composite	Application manager	*DATA IN_OUT* facets	Development of application using objects approach	Good structuring for the application, automated creation of code thanks to generate.	Potential increase on the object count, generate needed
3	Data blocks: physical I/O exchange	Application manager	*Owner_Consumer* templates	Flexible distribution of I/O and control objects, usage of M340 / PRA as DIO platform	Well structured for physical I/O exchange and assignment, automated creation of code thanks to generate.	Links assignment in Edit Links editor can be time consuming, generate needed
4	Refinement mode	Project manager (refinement)	-	Development of application using refinement approach	Easy and fast development, no need of generate	Can be heavy and with lower code quality if lots of variables need to be exchanged

9 APPENDIX: IMPORTANT THINGS TO REMIND

9.1 Supported data types depending on controller type

The feasibility of the mapping of certain variables on Peer2Peer communication channels depends on the type of the controller used as Modbus TCP IO Scanner client.
There are some limitations only on M580 platform, which supports only REF_TO data types.
Here the excerpt from PES User Guide related to this aspect.
Note: attributes in yellow are not properly updated yet on PES user guide for version 4.1

The table describes required variable attribute values in the Control Participant when the topological entity acting as client communicates by using a Quantum CPU module or an NOE communication module (with Quantum or M340 platform).

Usage	Variable attribute	Owner project/Server	Consumer project/Client
Reading data from the server	Designation of the variable in the context of Process Expert	Variable	Network variable
	Name	Any valid name	Same name
	Type	Any, including the Reference data type (REF_TO). Some restrictions apply.	Same type
	Custom	Blank	P2P
	Constant(1)	True (selected)	True (selected)
	R/W Rights of Referenced Variable(1)	True (selected)	False (cleared)
Writing data to the server	Designation of the variable in the context of Process Expert	Network variable	Variable
	Name	Any valid name	Same name
	Type	Any, including the Reference data type (REF_TO). Some restrictions apply.	Same type
	Custom	P2P	Blank
	Constant(1)	True (selected)	True (selected)
	R/W Rights of Referenced Variable(1)	True (selected)	True (selected)
(1)	For variables of the Reference data type (REF_TO) only		

The table describes required variable attribute values in the Control Participant when the topological entity acting as client communicates by using an M580 CPU module with or without NOC communication module, or a Quantum controller with NOC communication module.

Usage	Variable attribute	Owner project/Server	Consumer project/Client
Reading data from the server	Designation of the variable in the context of Process Expert	Variable	Network variable
	Name	Any valid name	Same name
	Type	Reference data type (REF_TO) only	Same type
	Custom	Blank	P2P
	Constant	True (selected)	True (selected)
	R/W Rights of Referenced Variable	True (selected)	False (cleared)
Writing data to the server	Designation of the variable in the context of Process Expert	Network variable	Variable
	Name	Any valid name	Same name
	Type	Reference data type (REF_TO) only	Same type
	Custom	P2P	Blank
	Constant	True (selected)	True (selected)
	R/W Rights of Referenced Variable	True (selected)	True (selected)

9.2 Controller firmware versions

The HAL 2.0 requires the installation of the following firmware versions:

M340 ➔ 2.70 minimum
M580 ➔ 2.10 minimum
Quantum ➔ 3.30 minimum

The following firwmare versions for communication cards have been used:

BMX NOE0110 ➔ 6.30
BME NOC0311 ➔ 2.04