PES: How to integrate Momentum I/O platform in PES

PES 4.1

Rodrigo Elgueta – Marco Siena

Rev 1.0
March 2016

Table of Contents

1 INTRODUCTION AND OBJECTIVE

The purpose of this document is to show how to integrate Modicon Momentum legacy I/O platform into PES system and architecture.

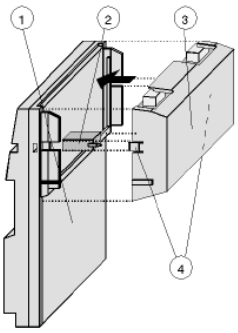
This document focuses especially on brown-field plants, where existing Momentum platforms may require integration in PES.

This document is based on features avalaible in PES 4.1.

2 MODICON MOMENTUM COMMUNICATION TECHNOLOGIES

2.1 Momentum Hardware

Modicon Momentum I/O legacy offer is based on the following main components:
- Processor* or Communication Bus Adapter
- Option adapters*
- I/O unit (also named I/O base)



1 I/O unit
2 Connecting plug (ATI interface)
3 Bus adapter (with 1 or 2 bus plugs depending on the bus type)
4 Spring clips

**Not considered in this KB*

This document will focus only on Communication Bus Adapters. The solution based on Processor, (171CCx references) using Modbus or Modbus TCP ports is solved like a normal integration of a device with relevant protocols. In the case of Serial Option Adapter (172JNN21032 reference) that use Modbus port, the same solution is applied. The Option Adapters for Modbus Plus networks (172PNNxx references) is not covered in this document.

The list of Communication Bus Adapters available in Momentum legacy offer is shown below:

For this Network...	Order this Adapter...	and this Manual...
ControlNet	170 LNT 810 00	870 USE 007
DeviceNet	170 LNT 710 00	870 USE 104
Ethernet	170 ENT 110 01	870 USE 114
FIPI/O	170 FNT 110 00	870 USE 005
InterBus	170 INT 110 00 170 INT 110 01 170 INT 120 00	870 USE 009
Modbus Plus (IEC data format)	170 PNT 110 20 (Single Port) 170 PNT 160 20 (Dual Port)	870 USE 103
Modbus Plus (984 data format)	170 NEF 110 21 (Single Port) 170 NEF 160 21 (Dual Port)	870 USE 111
Profibus-DP	170 DNT 110 00	870 USE 004

Not feasible in PES architectures

Feasible in PES architectures

Note: Some systems using the reference 170ENT11002 instead of 170ENT11001/0, the first one has a 100 Base T port and the second ones have a 10 Base T port. Both are compatible with all IO Bases.

Whenever a Momentum system has to be integrated in a PES architecture, it is recommended to use Ethernet Modbus TCP communication (170 ENT 100 01) or Profibus-DP communication (170 DNT 110 00).

The list of available I/O bases is shown below.

There is no particular restriction about the usage of the below described I/O bases.

Please consider the aspects related to the lifecycle while planning your modernization (especially concerning “Specials” type I/O bases).

Analog

The following analog I/O bases are available.

Part Number	Channels	Type	Details
170 AAI 030 00	8	Input	Broken wire detection
170 AAI 140 00	16	Input	Single-ended
170 AAI 520 40	4	Input	RTD/Thermocouple/mV
170 AAO 120 00	4	Output	0...20 mA
170 AAO 921 00	4	Output	4...20 mA

Combination

The following I/O bases support a combination of analog and discrete I/O.

Part Number	Channels	Type	Details
170 AMM 090 00	4 analog in 2 analog out 4 discrete in 2 discrete out	Input/Output	24 VDC
170 ANR 120 90 Unipolar	6 analog in 4 analog out 8 discrete in 8 discrete out	Input/Output	24 VDC
170 ANR 120 91 Bipolar	6 analog in 4 analog out 8 discrete in 8 discrete out	Input/Output	24 VDC

Discrete

The following discrete I/O bases are available.

Part Number	Points	Type	Details
170 ADI 340 00	16	Input	24 VDC
170 ADI 350 00	32	Input	24 VDC
170 ADI 540 50	16	Input	120 VAC
170 ADI 740 50	16	Input	230 VAC
170 ADM 350 10	16 in 16 out	Input Output	24 VDC, True High
170 ADM 350 11	16 in 16 out	Input Output	24 VDC, True High Fast Inputs
170 ADM 350 15	16 in 16 out	Input Output	24 VDC, True Low
170 ADM 370 10	16 in 8 out	Input Output	24 VDC @ 2 A
170 ADM 390 10	16 in 12 out	Input Output	24 VDC
170 ADM 390 30	10 in 8 relay out	Input Output	24 VDC
170 ADM 690 51	10 in 8 out	Input Output	120 VAC
170 ADO 340 00	16	Output	24 VDC
170 ADO 350 00	32	Output	24 VDC
170 ADO 530 50	8	Output	115 VAC @ 2A
170 ADO 540 50	16	Output	120 VAC
170 ADO 730 50	8	Output	230 VAC @ 2A
170 ADO 740 50	16	Output	230 VAC
170 ARM 370 30	10 in 8 out	Input Output	120 VAC Powered 24 VDC in

NOTE: The 170 ADM 690 50 has been replaced by the 170 ADM 690 51.

Specials

The following specialty I/O bases are available.

Part Number	Points	Type	Details
170 AEC 920 00	2	Counter	24 VDC
170 ANM 050 10		Seriplex	
170 ADM 540 80	6 in/3 out	Modbus	120 VAC

Each I/O base has a specific amount of data to exchange with its master. The amount and meaning of these words is defined in **Modicon Momentum I/O Base User Guide** (doc 870 USE 002 00) document.

The following is a brief view of these data areas.

Reference	Reg. To Read	Reg. To Write	Reference	Reg. To Read	Reg. To Write
170AAI03000	8	2	170ADM54080	6	3
170AAI14000	16	4	170ADM69050	1	1
170AAI52040	4	4	170ADM69051	1	1
170AAO12000	0	5	170ADM85010	1	1
170AAO92100	0	5	170ADO34000	0	1
170ADI34000	1	0	170ADO35000	0	2
170ADI35000	2	0	170ADO53050	0	1
170ADI54050	1	0	170ADO54050	0	1
170ADI74050	1	0	170ADO73050	0	1
170ADM35010	1	1	170ADO74050	0	1
170ADM35011	1	1	170ADO83030	0	1
170ADM35015	1	1	170AMM09000	5	5
170ADM37010	1	1	170ANR12090	12	12
170ADM39010	1	1	170ANR12091	12	12
170ADM39030	1	1	170ARM37030	1	1

For example, this is the mapping required for 170 AAI 030 00 I/O base

I/O Map

The I/O base must be mapped as eight contiguous input words and two contiguous output words, as follows:

Word	Input Data	Output Data
1	Value, input channel 1	Parameters for input channels 1 ... 4
2	Value, input channel 2	Parameters for input channels 5 ... 8
3	Value, input channel 3	Not used
4	Value, input channel 4	Not used
5	Value, input channel 5	Not used
6	Value, input channel 6	Not used
7	Value, input channel 7	Not used
8	Value, input channel 8	Not used

The data amount and format is the same for each communication bus type.

2.2 Preferred implementations

Based on the considerations explained in the previous chapter, the preferred implementations are described below:

For **Modbus TCP/IP** communication, it is recommended to use Modbus TCP implicit messaging (IO Scanning service). Indeed the Momentum I/O platform will be part of the DIO network. The integration of each IO Base into the platform needs an IP address previously defined. If the IP is not previously assigned, a BOOTP address server must be used in the configuration of the system, in alternative the IP has to be associated manually starting from default IP address (available only in 170ENT11001 only).

The following procedure to assign IP Address to a Momentum Base is in **Momentum 170ENT11001/170ENT11000 Ethernet Communications Adapter User Guide** (870 USE 114 00)

Supported on the Momentum 170ENT11001 ONLY
The Momentum 170ENT11001in an "out of the box" condition will accept an IP address from two sources:

- BOOTP server
- IP address derived from its MAC address

The MAC address is the IEEE Global Address and is found on the Momentum 170ENT11001's Global Address label.
To use the default IP wait for the timeout period to expire and the device be available at its default IP.
The default IP derives from the last four octets of the MAC address by doing a hex to decimal conversion of these octets.
Example:
One Momentum 170ENT11001 when made at the factory was assigned the unique IEEE Global Address of 000054102D11.
000054102D11 hex maps to 84.16.45.17, which is the default IP address of that Momentum 170ENT1101. This IP address maybe a duplicate of an IP address on your network. Therefore, ensure that the IP address is not a duplicate before attaching the Momentum 170ENT11001.

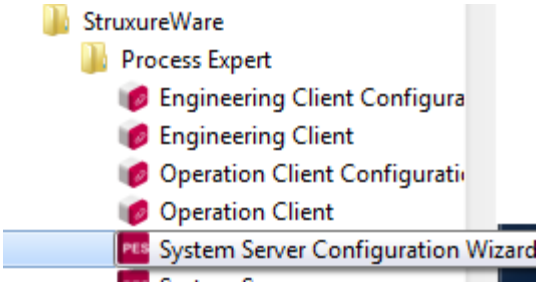
- For **Profibus DP** communication, it is recommended to use PRM (Profibus Remote Master). PRM must be scanned from a compatible IO Scanner master (for example, Quantum **140NOE771*1**, M340 **BMXNOE01*0.2** or M580 **BMENOC03*1**)

3 CASE STUDY 1: PROFIBUS COMMUNICATIONS

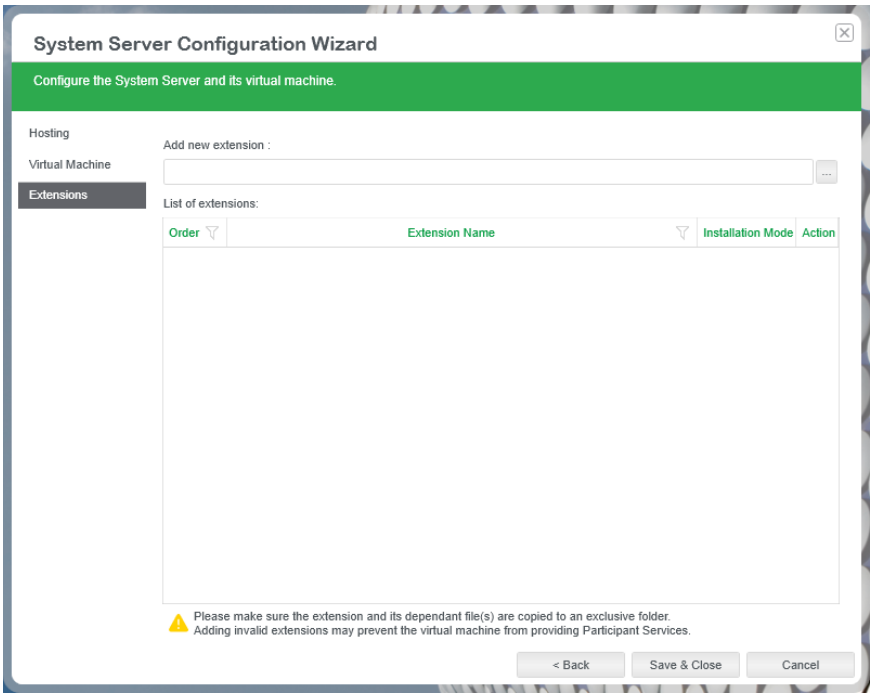
3.1 Installation of Momentum GSD into the HW catalog

In order to properly define the configuration of our Momentum I/O system in PRM configuration, it is necessary to install the GSD file (***ASA_1752.GSD***) into the PES VM.

In PES 4.1 the right place to do it is the **System Server Configuration Wizard**.
From version 4.1, PES is able to propagate the installation of additional features to the VM (such as GSD files, DTMs and so on) to the VM of the engineering clients of the system.

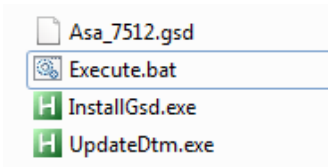


The third tab of the configuration wizard is named “Extensions”

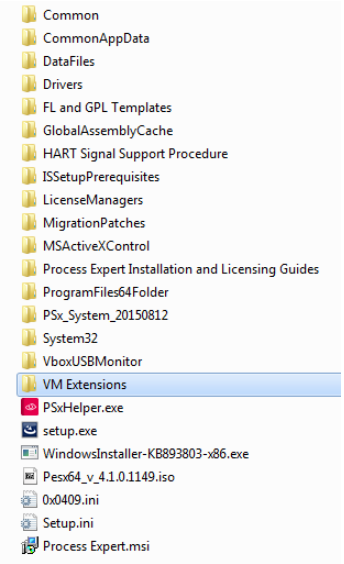


From here it is possible to point to a folder containing the following files:

1- **GSD to install** (in this case *ASA_7512.GSD*)
2- **Execute.bat** file containing actions to perform
3- **InstallGsd.exe** file
4- **UpdateDTM.exe** file



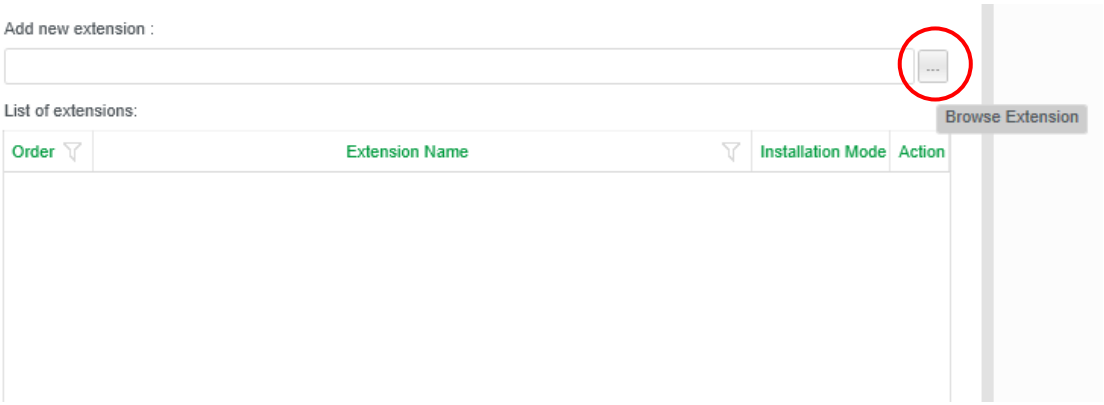
It is possible to prepare this folder using the resources available under %PES setup folder%**\VM Extensions\InstallGSD** folder.



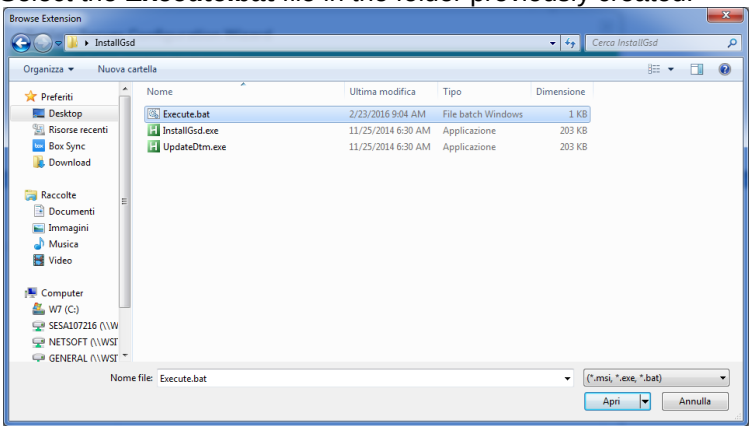
The **Execute.bat** file by default contains these instructions. There is no need to modify it.

```
@echo off
echo Installing GSD...
InstallGsd.exe "%~dp0"
echo Installing DTM
echo Updating DTM catalog...
UpdateDtm.exe
echo Done
```

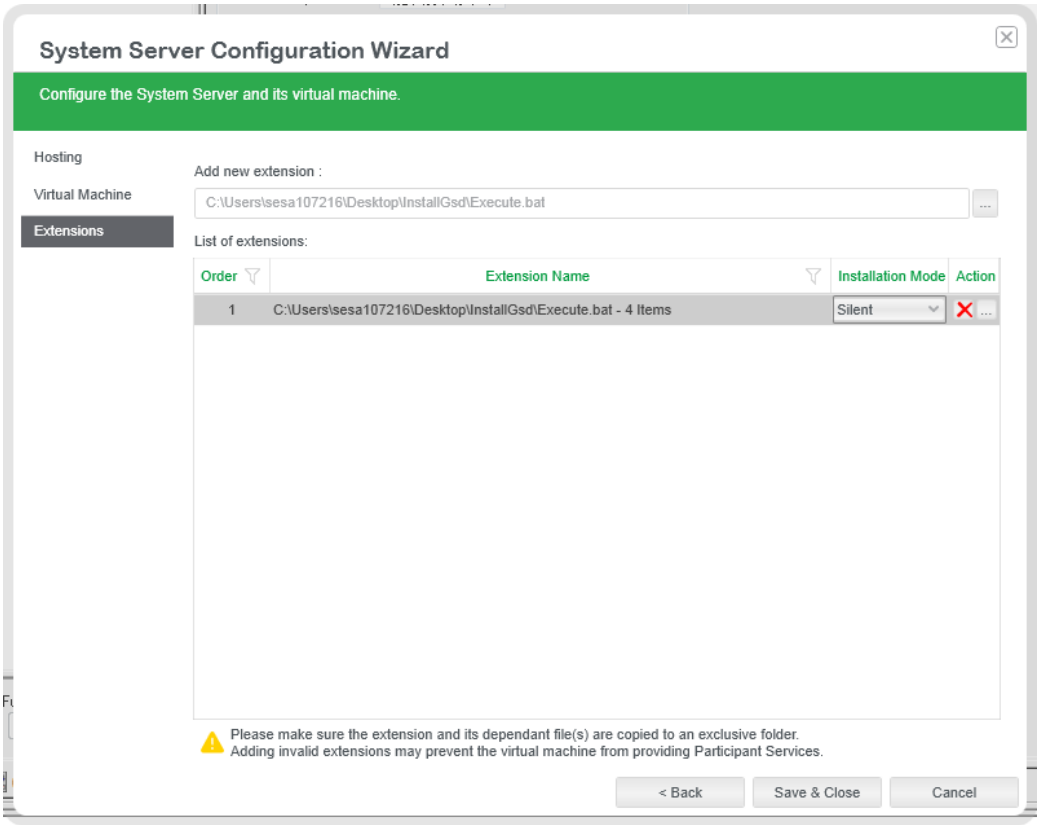
Select now the Browse button under “Add new extension”



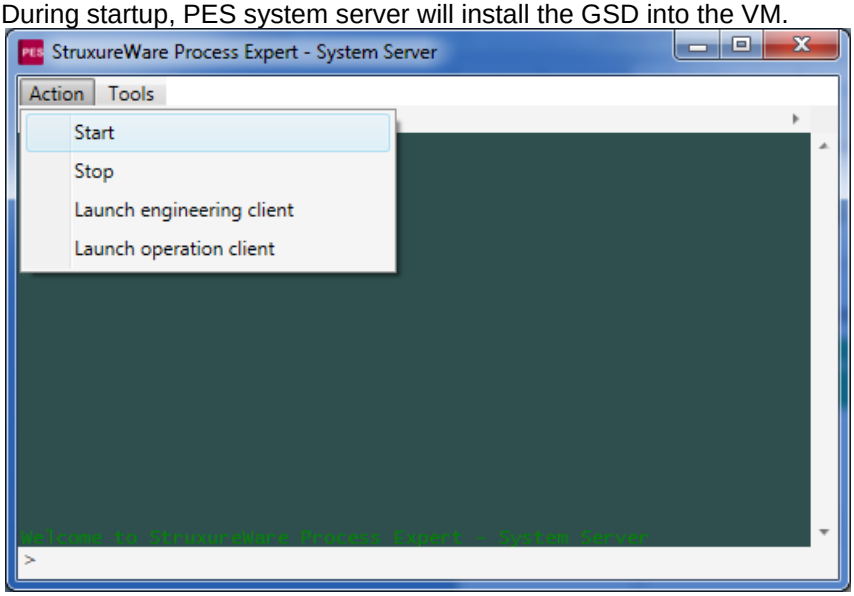
Select the **Execute.bat** file in the folder previously created.



Validate clicking "**Save and Close**".



Next step is to start the system server.
During startup, PES system server will install the GSD into the VM.




```
2016-03-08 15:22:01.6201 INFO  [T  5] Pes.Vm.4.1#0 on localhost Need to generate because the following files are modified after previous Vm generation.
2016/03/08 03:13:41.092 -- C:\Users\sesa107216\AppData\Local\Schneider Electric\Process Expert 4.1\Vm\Pes.Vm.4.1.UserExtensions\UserExtensions.xml
2016/03/08 03:13:41.028 -- C:\Users\sesa107216\AppData\Local\Schneider Electric\Process Expert 4.1\Vm\Pes.Vm.4.1.UserExtensions\InstallGsd_1\Asa_7512.gsd
2016/03/08 03:13:41.029 -- C:\Users\sesa107216\AppData\Local\Schneider Electric\Process Expert 4.1\Vm\Pes.Vm.4.1.UserExtensions\InstallGsd_1\Execute.bat
2016/03/08 03:13:41.030 -- C:\Users\sesa107216\AppData\Local\Schneider Electric\Process Expert 4.1\Vm\Pes.Vm.4.1.UserExtensions\InstallGsd_1\InstallGsd.exe
2016/03/08 03:13:41.031 -- C:\Users\sesa107216\AppData\Local\Schneider Electric\Process Expert 4.1\Vm\Pes.Vm.4.1.UserExtensions\InstallGsd_1\UpdateDtm.exe

2016-03-08 15:25:14.6201 INFO  [T  5] Extensions on Pes.Vm.4.1#0 on localhost:
Results of application of extensions

c:\Pes\Vm\Extensions\ActiveX>regsvr32 /s FM20.dll
Extension ActiveX is applied

c:\Pes\Vm\Extensions\DisableLicenseCheck>reg delete "HKLM\SOFTWARE\Schneider Electric\Common\License" /v "PSxProcessName" /f
Extension DisableLicenseCheck is applied

c:\Pes\Vm\Extensions\FixedCustomString>xcopy /V /Y /R AttributesManagement.XML  "c:\Program Files\Schneider Electric\Unity Pro"
C:AttributesManagement.XML
1 File(s) copied
Extension FixedCustomString is applied

c:\Pes\Vm\Extensions\FixedFullScreen>copy unityxl.xpdf "C:\ProgramData\Schneider Electric\Unity Pro\Xpdf\unityxl.xpdf"
        1 file(s) copied.
Extension FixedFullScreen is applied

c:\Pes\Vm\Extensions\FixedIp>netsh int ipv4 set address "Local Area Connection" static 192.168.101.15 255.255.255.0 192.168.101.2

Extension FixedIp is applied

c:\Pes\Vm\Extensions\FixedUnityDump>regedit /s UnityDump.reg
Extension FixedUnityDump is applied
Results of application of user extensions.
Installing GSD...
Installing DTM
Updating DTM catalog...
Done
Extension InstallGsd_1 is applied

c:\Pes\Vm\Extensions\UpdateDtmCatalog>ren UpdateDtm.txt UpdateDtm.exe

c:\Pes\Vm\Extensions\UpdateDtmCatalog>call UpdateDtm.exe

c:\Pes\Vm\Extensions\UpdateDtmCatalog>ren UpdateDtm.exe UpdateDtm.txt
Extension UpdateDtmCatalog is applied

2016-03-08 15:27:16.8086 INFO  [T  4] From [127.0.0.1] 2016-03-08 14:27:16.8070 INFO  Vm [Pes.Vm.4.1#0 on localhost] Create ToolConnector for VJC.7.4
2016-03-08 15:27:17.4511 INFO  [T  4] From [127.0.0.1] 2016-03-08 14:27:17.4479 INFO  Vm [Pes.Vm.4.1#0 on localhost] Preparation stage for VJC.7.4
2016-03-08 15:27:31.9666 INFO  [T  4] From [127.0.0.1] 2016-03-08 14:27:31.9588 INFO  Vm [Pes.Vm.4.1#0 on localhost] Create ToolConnector for Advantys.8.0
2016-03-08 15:27:31.9666 INFO  [T  4] From [127.0.0.1] 2016-03-08 14:27:31.9588 INFO  Vm [Pes.Vm.4.1#0 on localhost] Preparation stage for Advantys.8.0
2016-03-08 15:27:31.9666 INFO  [T  4] From [127.0.0.1] 2016-03-08 14:27:31.9588 INFO  Vm [Pes.Vm.4.1#0 on localhost] Create ToolConnector for Unity.10.0
2016-03-08 15:27:31.9716 INFO  [T  4] From [127.0.0.1] 2016-03-08 14:27:31.9688 INFO  Vm [Pes.Vm.4.1#0 on localhost] Preparation stage for Unity.10.0
2016-03-08 15:28:32.6152 INFO  [T 16] From [127.0.0.1] 2016-03-08 14:28:32.5986 INFO  Vm [Pes.Vm.4.1#0 on localhost] Create ToolConnector for Unity.10.0
2016-03-08 15:28:32.6177 INFO  [T 16] From [127.0.0.1] 2016-03-08 14:28:32.5986 INFO  Vm [Pes.Vm.4.1#0 on localhost] Preparation stage for Unity.10.0
2016-03-08 15:30:02.7097 INFO  [T  5] Vm Pes.Vm.4.1#0 on localhost is started
2016-03-08 15:30:02.7097 INFO  [T  5] Agent services initialized
2016-03-08 15:30:02.7097 INFO  [T  5] Processing server startup tasks...
2016-03-08 15:30:02.7317 INFO  [T 34] Session [Id:23a628fb-b4cc-45eb-9877-3dab2253f49c System:Global User:PES Server (System)] Initializing Identifier Cache.
2016-03-08 15:30:07.2011 INFO  [T 40] Session [Id:23a628fb-b4cc-45eb-9877-3dab2253f49c System:Global User:PES Server (System)] Initializing Identifier Cache.
2016-03-08 15:30:10.1412 INFO  [T 40] Session [Id:23a628fb-b4cc-45eb-9877-3dab2253f49c System:Global User:PES Server (System)] Starting purge of content repository
2016-03-08 15:30:32.1241 INFO  [T 40] Session [Id:23a628fb-b4cc-45eb-9877-3dab2253f49c System:Global User:PES Server (System)] Purge of content repository done
2016-03-08 15:30:32.6872 INFO  [T 40] Session [Id:23a628fb-b4cc-45eb-9877-3dab2253f49c System:Global User:PES Server (System)] Loading template binaries...
2016-03-08 15:30:41.8545 INFO  [T  5] Server is ready (V4.1.0.1149)
```

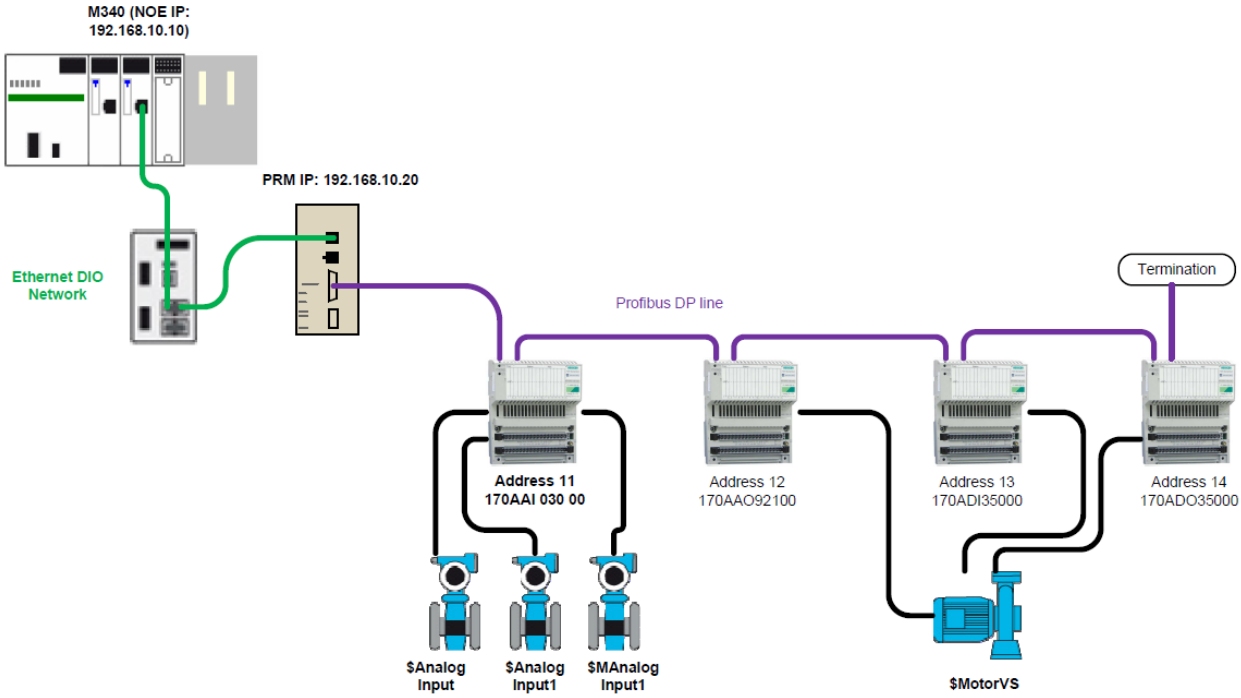
GSD installation + Unity DTM catalog update

Server is ready

3.2 Creation of Topology

In this example, we will consider the following use case:

- M340 PAC with BMXNOE0100.2
- PRM
- Profibus Momentum I/O base 1: 170AAI 030 00 (8 AI) – address 11
- Profibus Momentum I/O base 2: 170AAO92100 (4 AO) – address 12
- Profibus Momentum I/O base 3: 170ADI35000 (32 DI) – address 13
- Profibus Momentum I/O base 4: 170ADO35000 (32 DI) – address 14

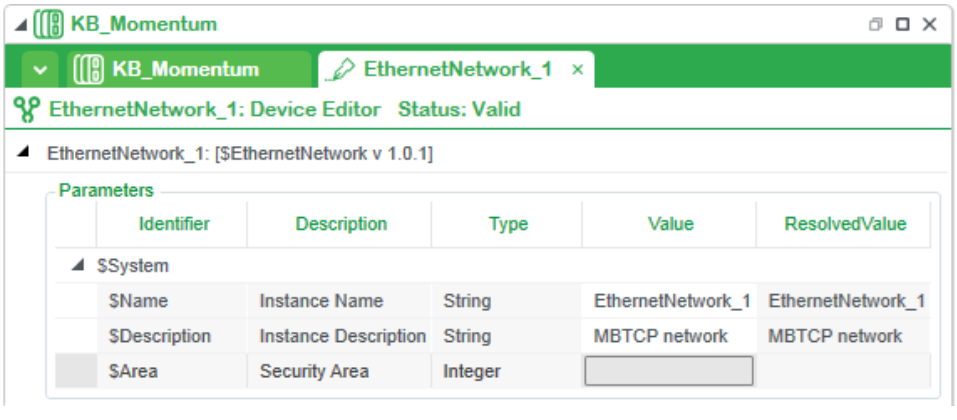


In this example, Analog Input island **170AAI03000** will manage 3 Analog Input GPL objects: one **$AnalogInput**, one **$AnalogInput1**, one **$MAnalogInput1**.
For this island, it will be demonstrated how to design a template for managing the I/O base features and for linking it to process objects in Application Manager, achieving a full object oriented approach.

Other I/O bases will be instead managed with $GenericDevice and refinement and they will serve a GPL **$MotorVS** object with hardwired command (Running and Fail **DI on 170ADI35000**, Run and Direction **DO on 170ADO35000**, Speed Setpoint **AO on 170AAO92100**).

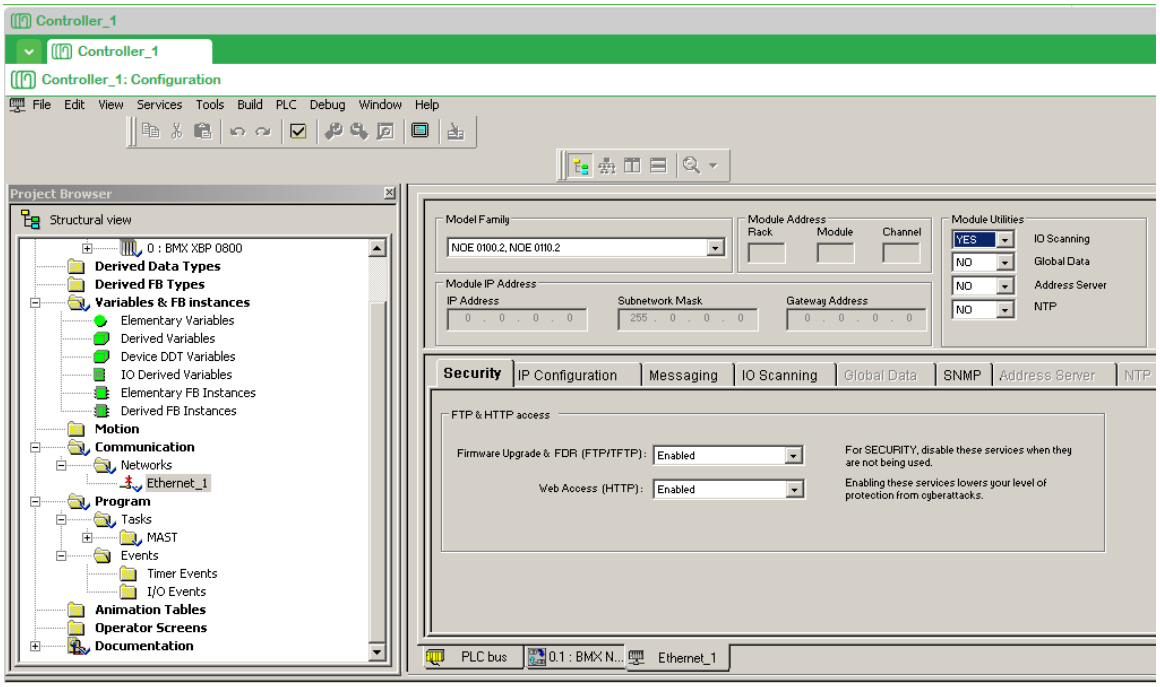
3.2.1 Ethernet network

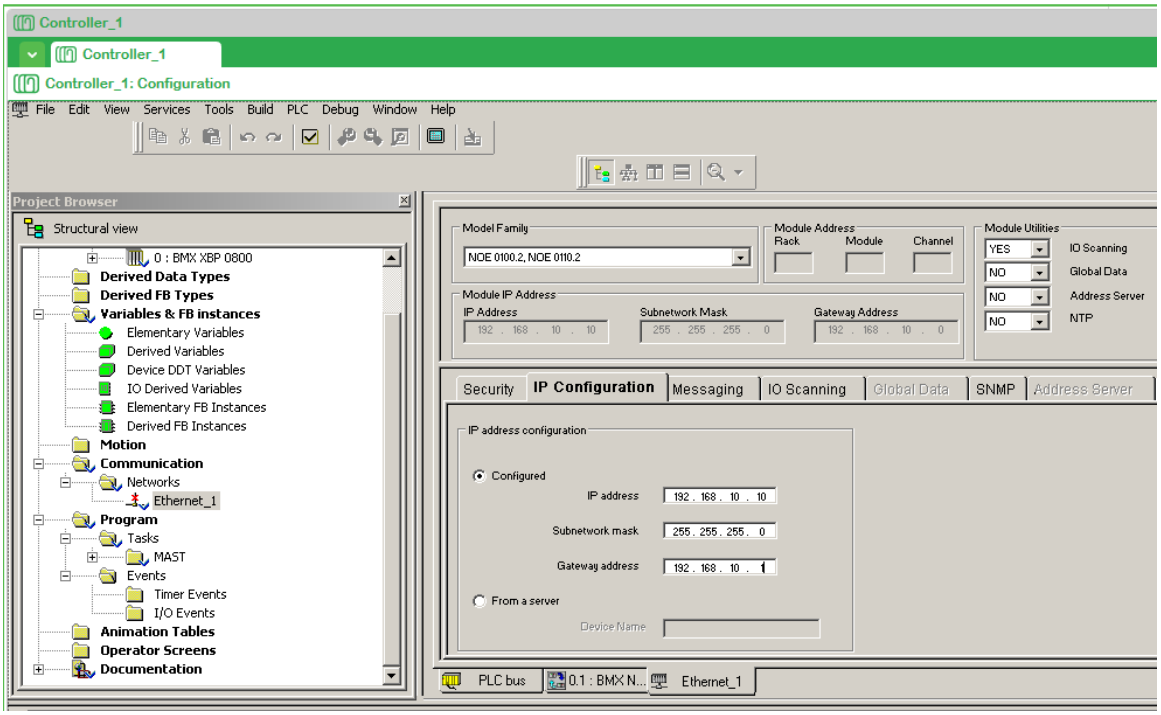
Create an Ethernet network.

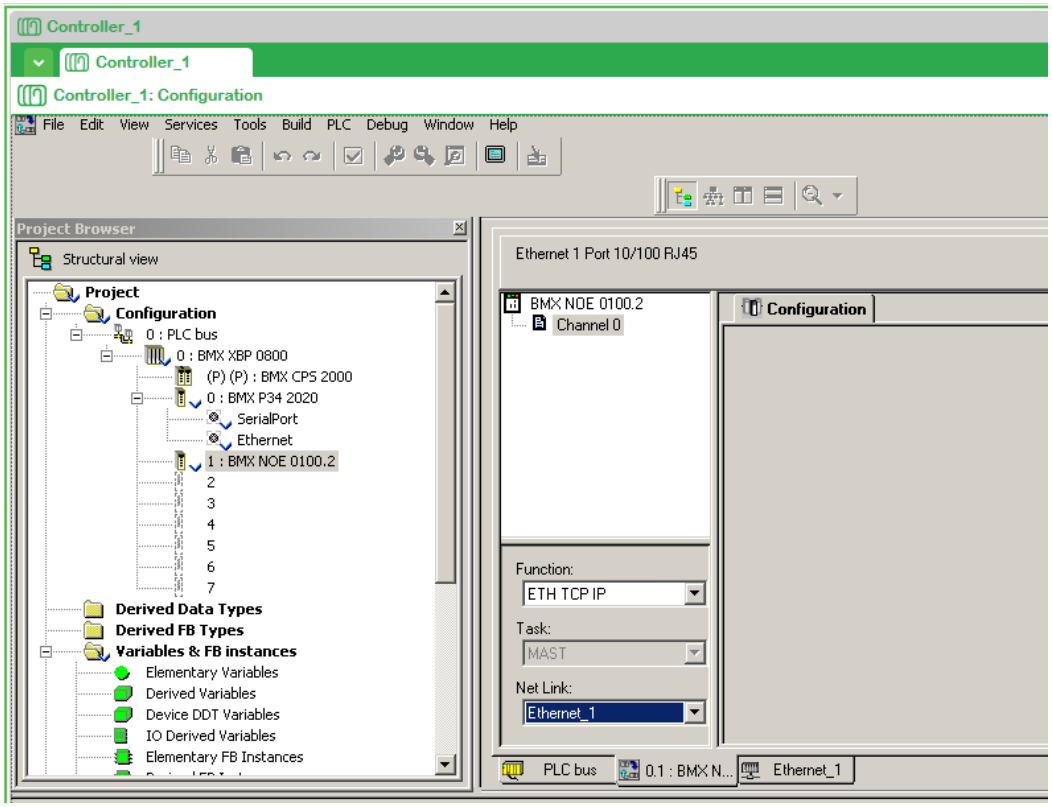


3.2.2 Controller configuration

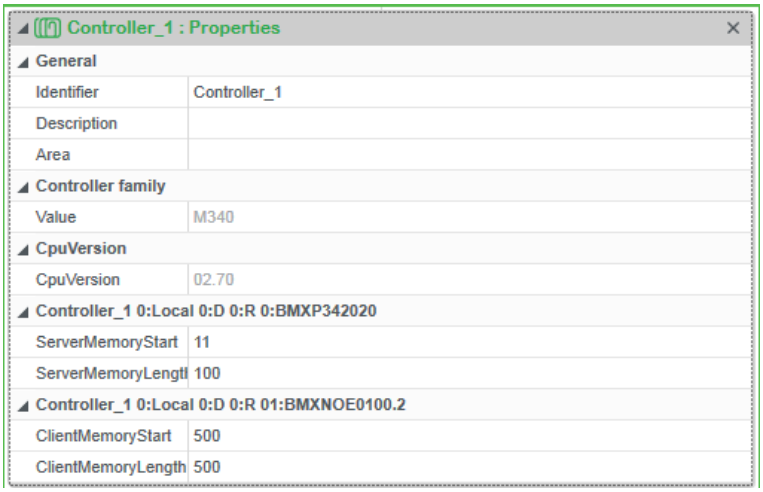
Create a new M340 Controller with BMXNOE0100.2 card. Remember to enable IO Scanner service.



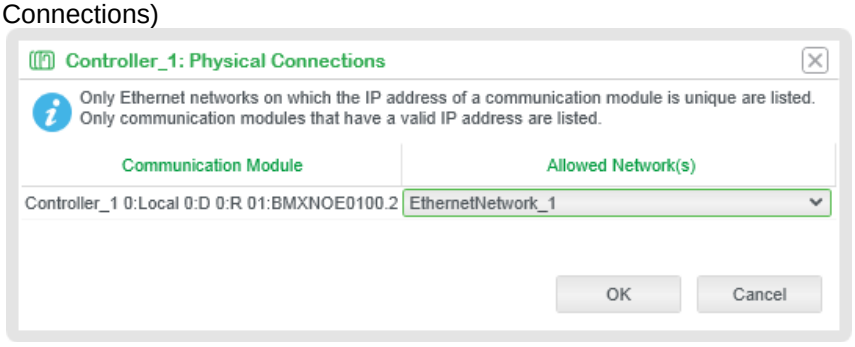




Remember also to configure the proper reservation of located area for Modbus TCP Client communication (right click on Controller_1 ➔ properties).

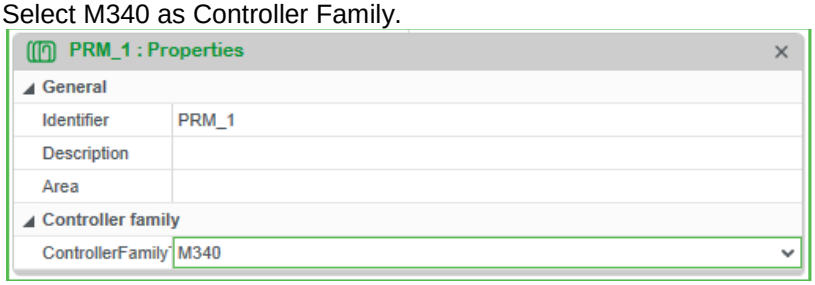


Connect the M340 NOE to Ethernet Network previously created (right click on Controllers_1 ➔ Physical Connections)

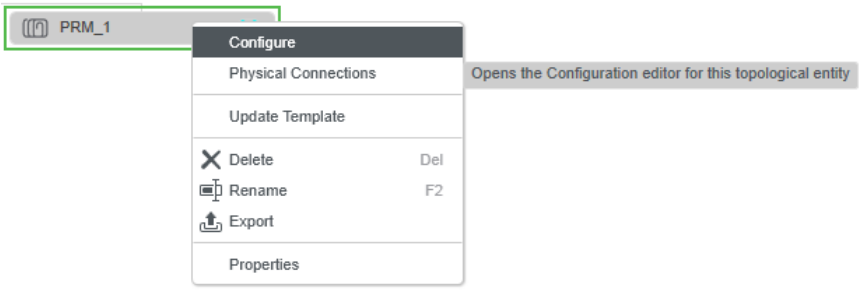


3.2.3 PRM Configuration

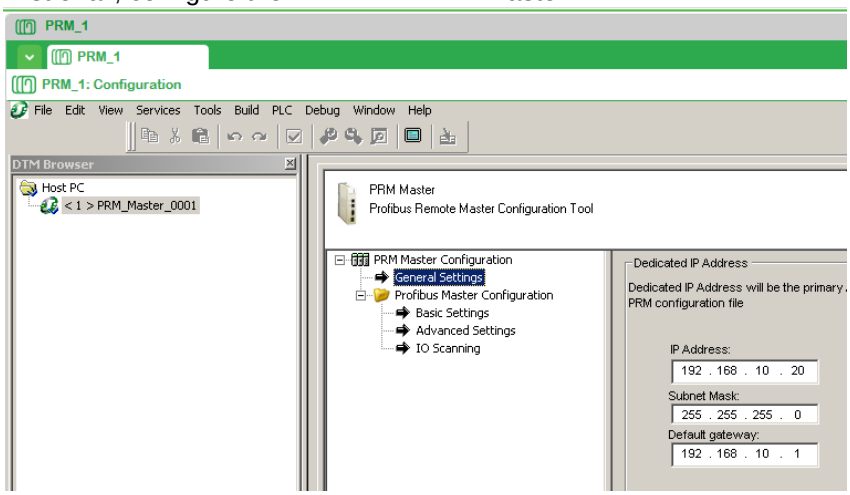
Create a new PRM. Before to configure it, right click and select Properties.
Select M340 as Controller Family.



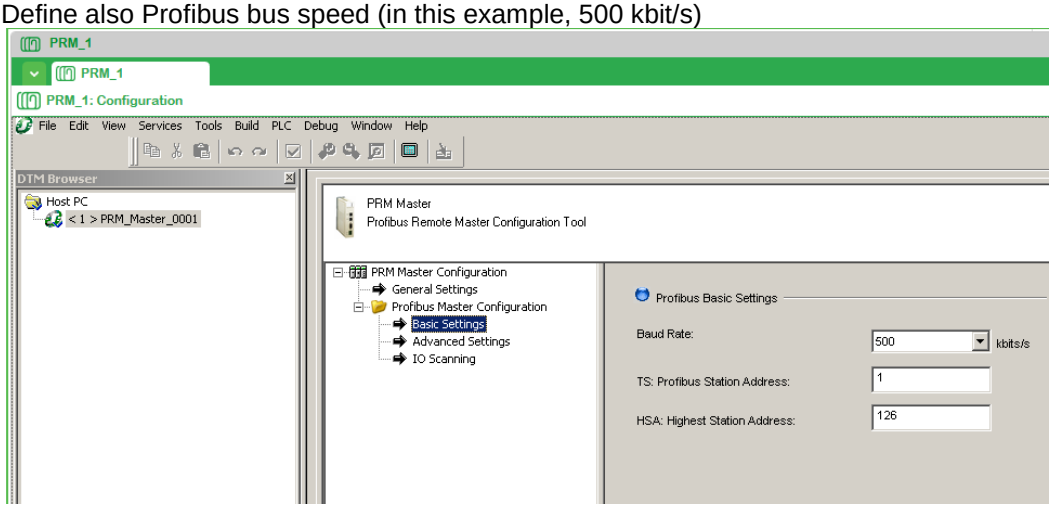
Now it is possible to launch PRM “Configure”



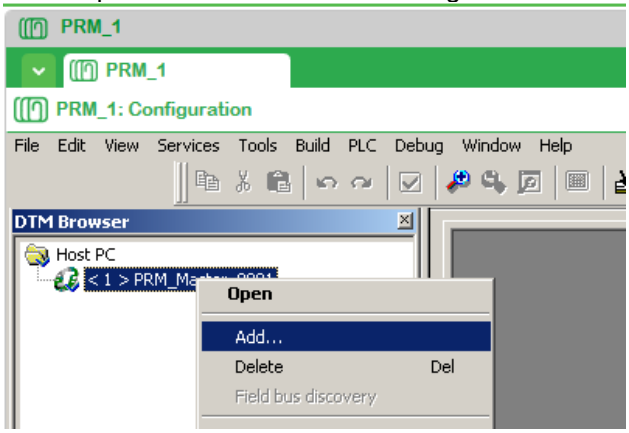
First of all, configure the PRM IP in PRM Master DTM



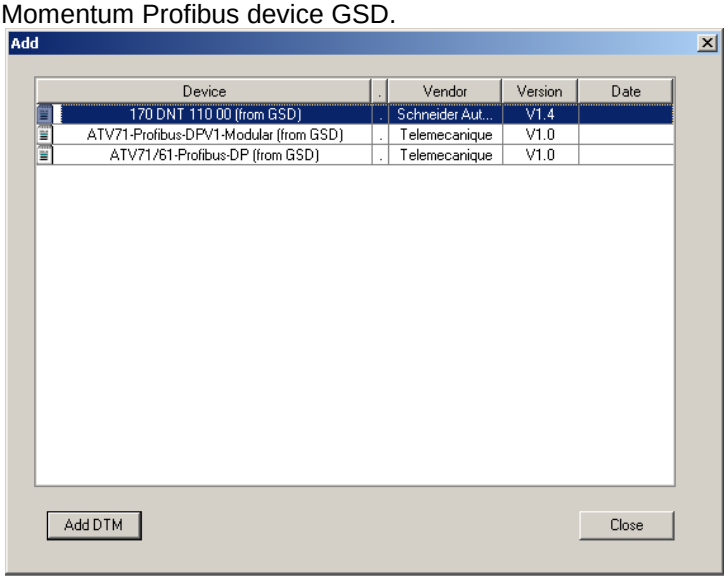
Define also Profibus bus speed (in this example, 500 kbit/s)



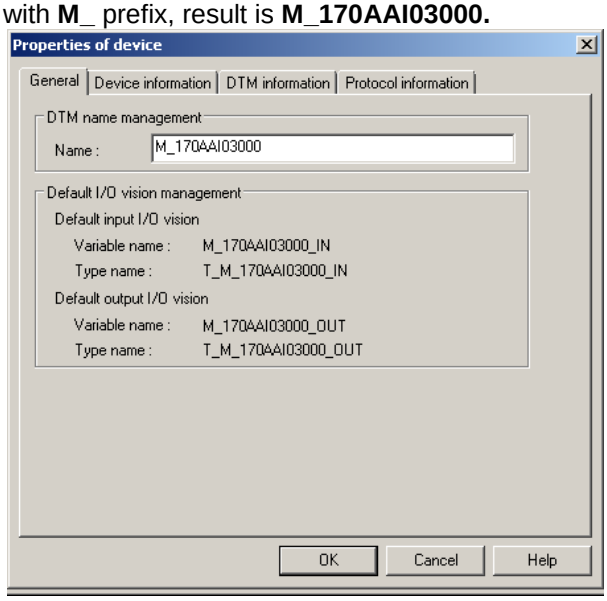
Next step is to start to add devices. Right click on PRM Master DTM and select “Add”



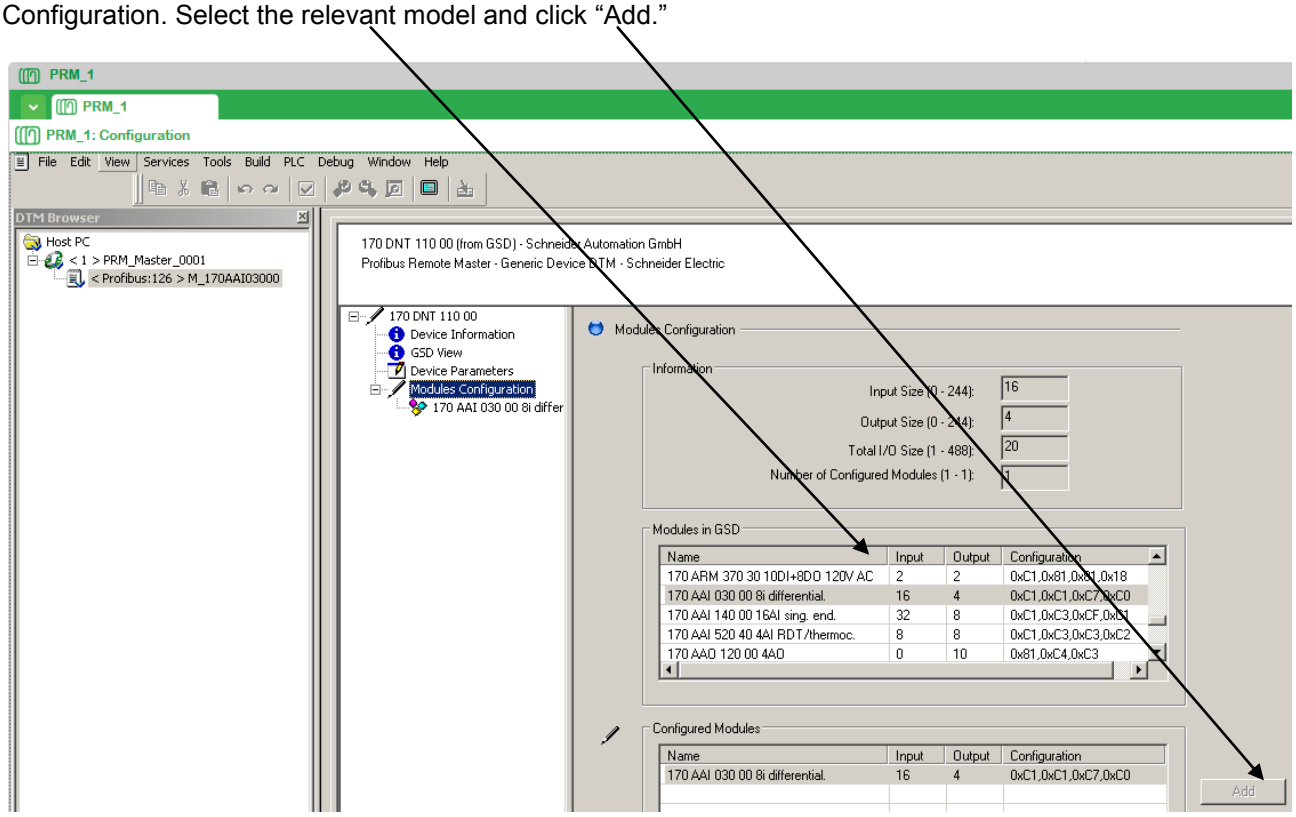
The 170 DNT 110 00 Profibus communication adapter is now available. This GSD contains all the I/O bases of the Momentum Platform, which can be selected in Profibus device configuration stage. Select the Momentum Profibus device GSD.



Define a name for the device. In this case first Momentum node is called with the same name of the I/O base with **M_** prefix, result is **M_170AAI03000.**

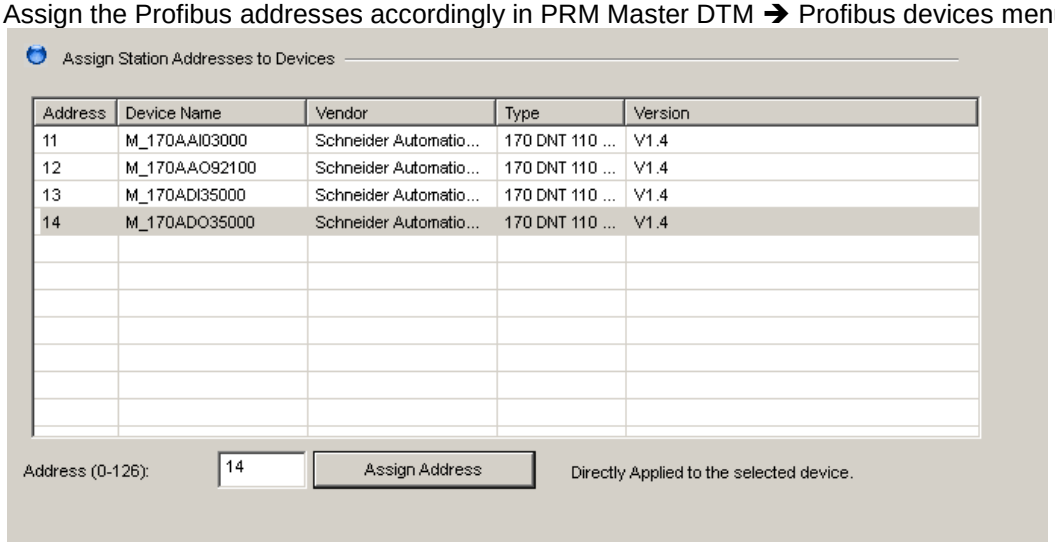


After the new device has been added, it is necessary to specify the associated I/O base in “Modules” Configuration. Select the relevant model and click “Add.”

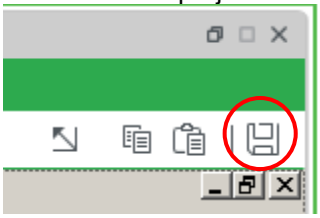


Repeat the last two steps for all I/O bases listed in architecture description.

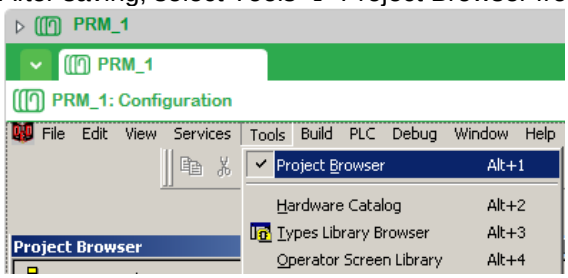
Assign the Profibus addresses accordingly in PRM Master DTM ➔ Profibus devices menu



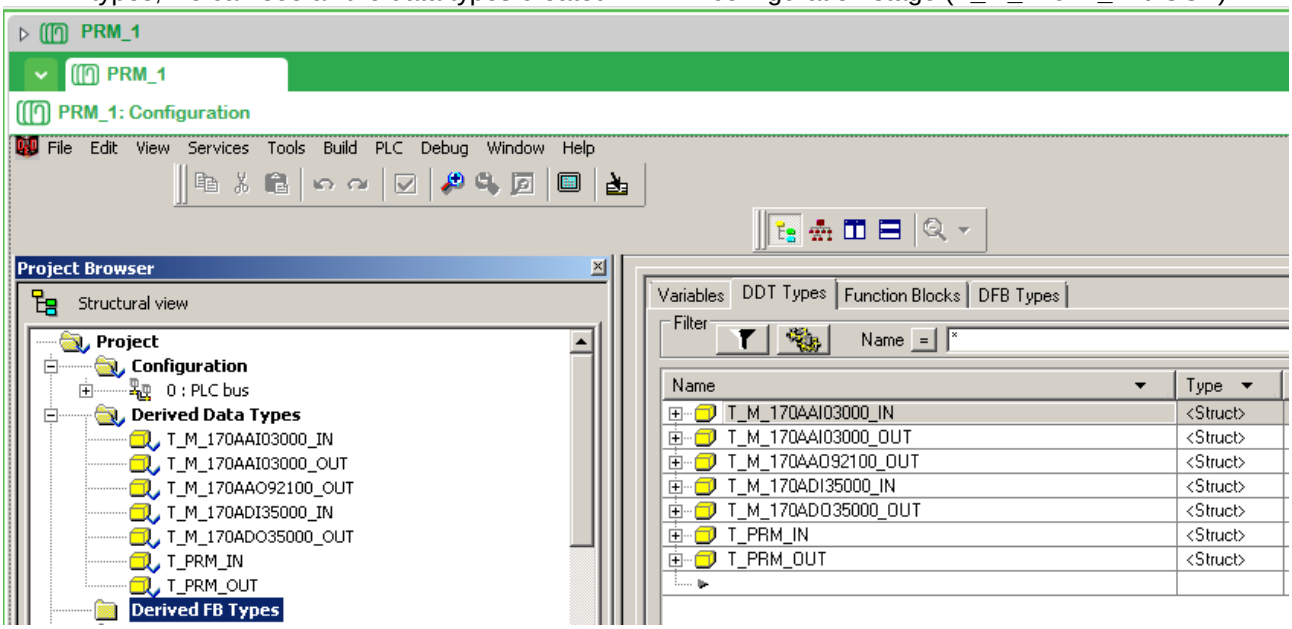
Save the PRM project.



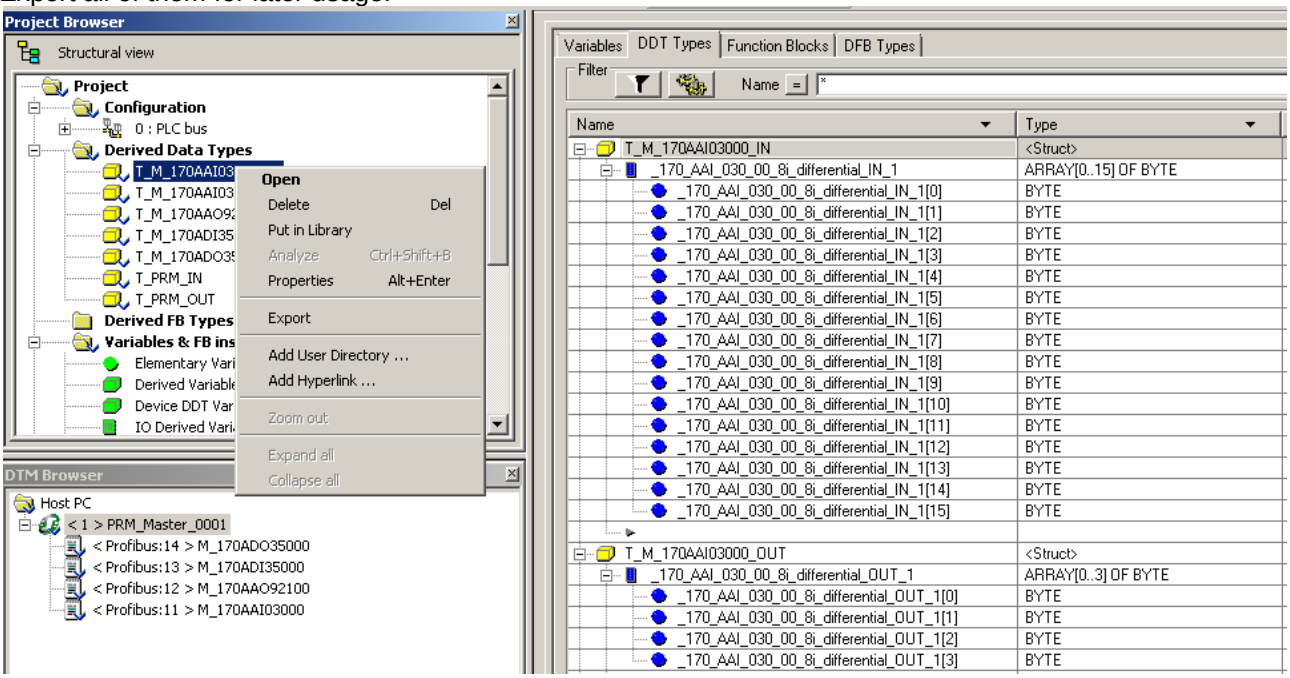
After saving, select Tools ➔ Project Browser from Unity participant.



In DDT types, we can see all the data types created in PRM configuration stage (**T_M_170***_IN / OUT**).



Export all of them for later usage.



The advantage of these DDT is that they are automatically created and they already include the amount of data to exchange with the I/O base.

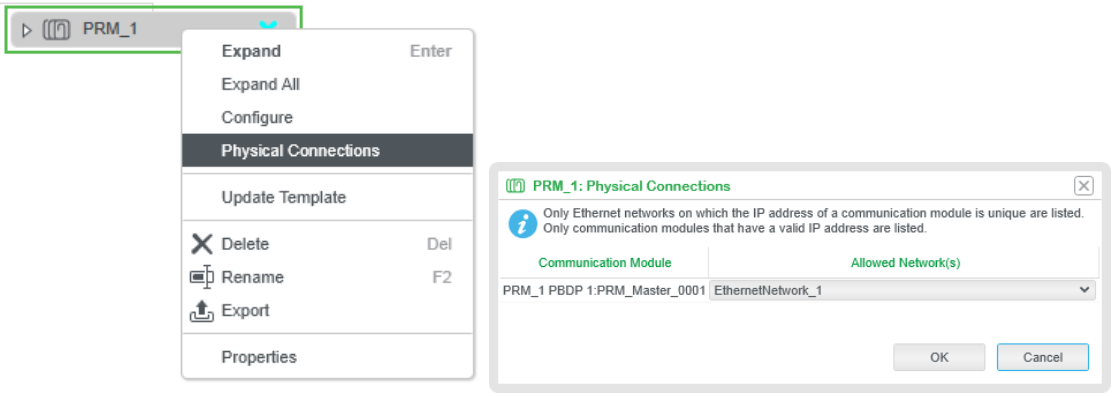
For **170AAI03000** for example we have 8 words IN and 2 words out (see following user manual excerpt).

I/O Map

The I/O base must be mapped as eight contiguous input words and two contiguous output words, as follows:

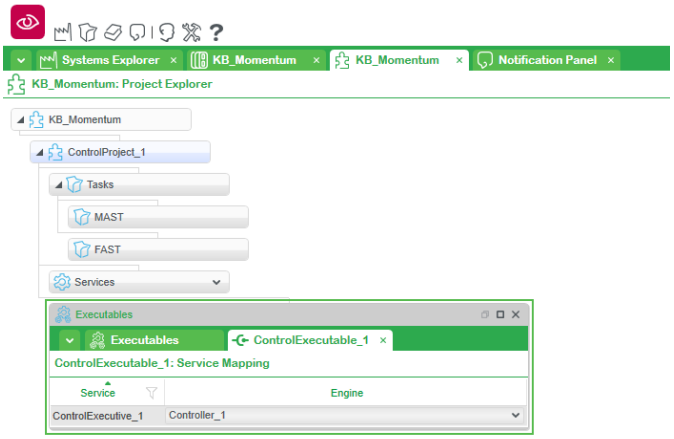
Word	Input Data	Output Data
1	Value, input channel 1	Parameters for input channels 1 ... 4
2	Value, input channel 2	Parameters for input channels 5 ... 8
3	Value, input channel 3	Not used
4	Value, input channel 4	Not used
5	Value, input channel 5	Not used
6	Value, input channel 6	Not used
7	Value, input channel 7	Not used
8	Value, input channel 8	Not used

Close the PRM configuration editor, connect the PRM to the previously created Ethernet network.



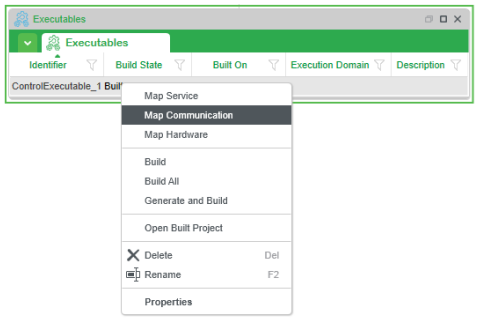
3.3 Creation of Project

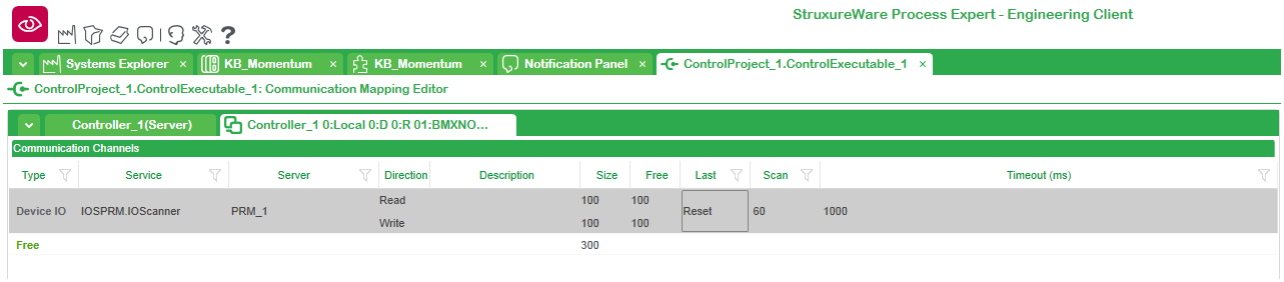
In Project editor, create a new M340 control project.



Map the **ControlExecutive_1** service to **Controller_1**

Add the PRM communication to the NOE Card in Map Communications stage.





3.4 Creation of template for 170AAI03000

In this chapter, it is described the object oriented philosophy of the integration of Momentum I/O bases.

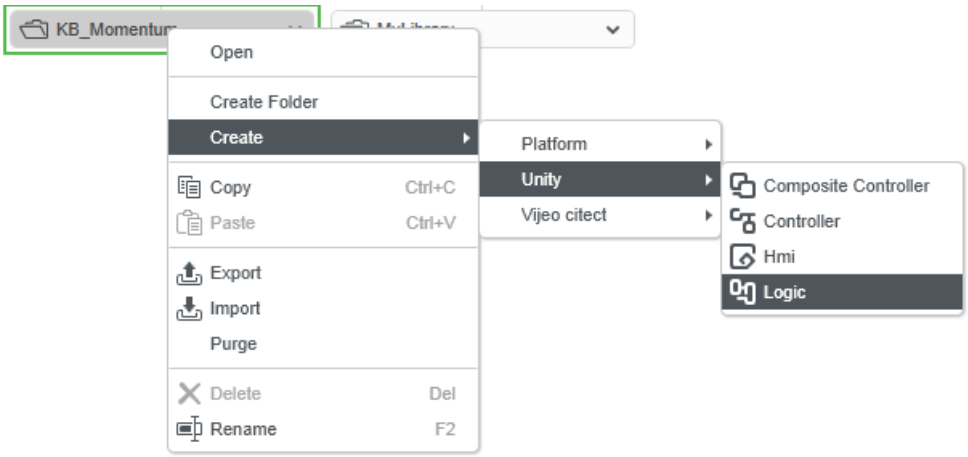
Based on the DDT exported from PRM configuration and the specific features of an I/O base, we can create new templates for each I/O base type.
This approach has the purpose of facilitating the engineers to assign the I/O signals of Momentum I/O bases to other objects in Application Manager, to increase productivity and improve software quality.

In this example we will show the object oriented approach for Momentum Analog Input base 170AAI03000.
For other I/O bases, we will show a different approach (using $GenericDevice template and refinement).

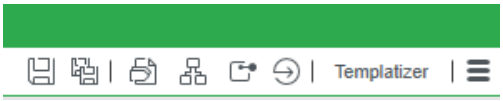
3.4.1 Creation of UL facet – AAI03000_UL

It is possible to create a new Unity Logic facet that contains the following items:

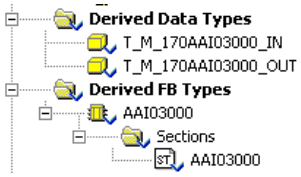
- DDT IN and DDT OUT previously exported
- A DFB instance that manages the input data (raw value and channel status) and output data (in this example, the channel configuration)
- Elementary variables associated to the object functionality (diagnostics, links to other objects).



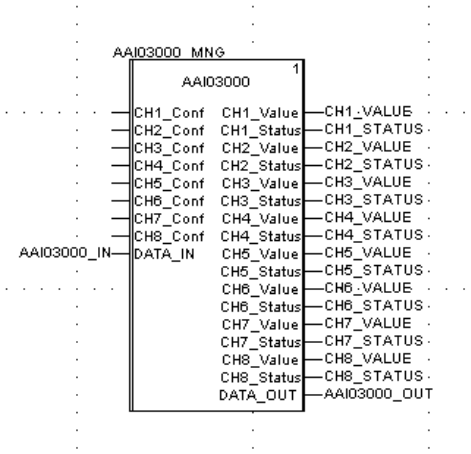
Thanks to the **Templatizer** we can open a Unity Pro participant session and develop our template logic.



We can import the DDT IN and DDT OUT types previously exported and create a new DFB type



After doing this, it is necessary to create an FBD section with an instance of the new DFB type.



Inputs CH1...CH8_Conf
The idea of this DFB is to have the 8 channels configuration in input (as parameters configured in PES AM) according to the following paradigm (excerpt from *Momentum I/O Base User Guide*).

Codes for Analog Input Parameters

Use the following codes to set the parameters for each analog input channel:

Code (binary)	Code (hex)	Parameter
0000	0	Reserved value (see note below)
0010	2	+/-5V and +/-20mA input range
0011	3	+/-10V input range
0100	4	Channel inactive
1010	A	1 ... 5V and 4 ... 20 mA input range

NOTE: The 0000 reserved value is more a control than a parameter. It forces the I/O base into a default condition where it continues to receive field inputs according to the previous channel parameters.

Input DATA_IN
DFB will manage the 8 channel raw values and status according to **DATA_IN** structure (read from Profibus input area). This input has to be connected to a DDT variable (**T_M_170AAI03000_IN** type), that will be mapped to the %MW read area after build.

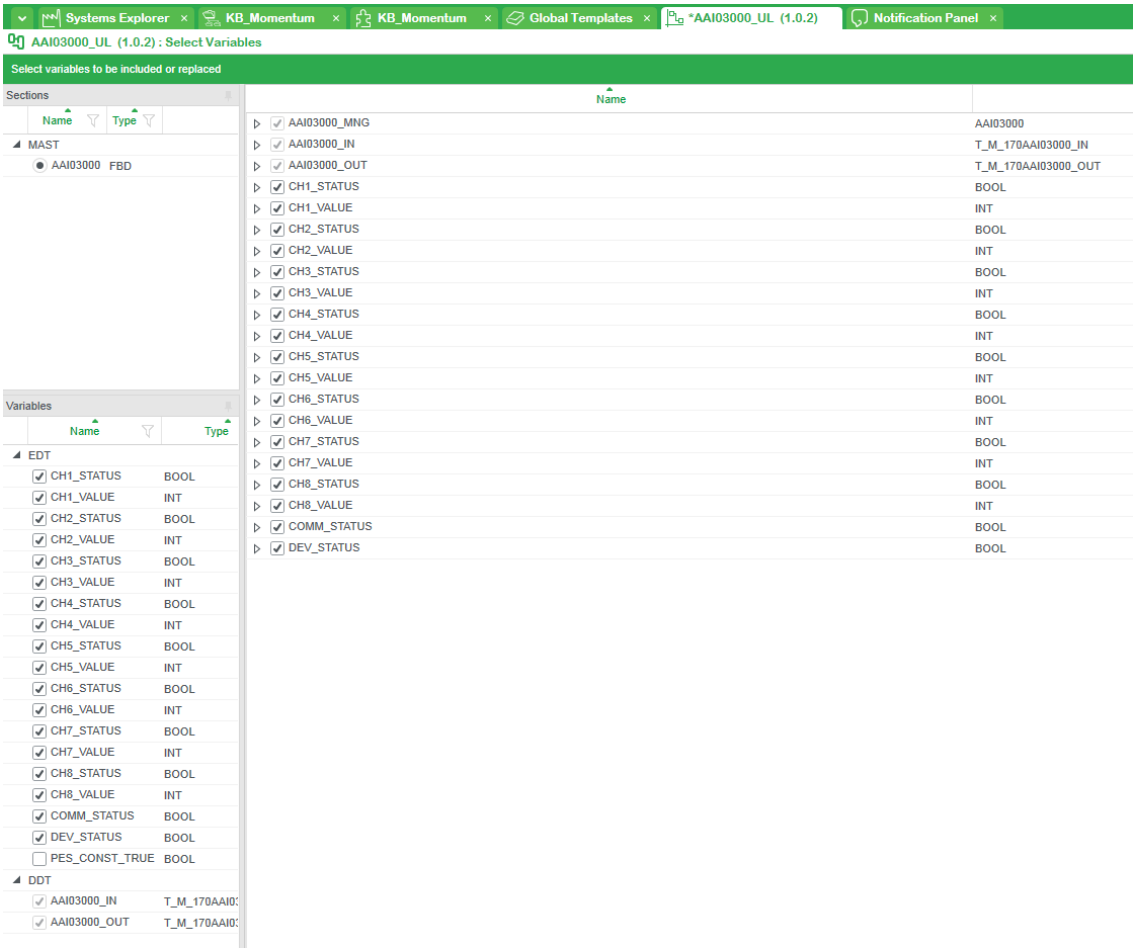
Outputs CH1…CH8_Value and CH1...CH8_Status
These outputs are meant to be connected to other objects (for example, **$AnalogInput** objects). Each output is linked to an elementary variable, which will be used for defining connection to other objects.

Output DATA_OUT
This structure contains the configuration that will be written to Profibus output area. In this case the structure contains the configuration of each analog channel. This output is connected to a DDT variable (**T_M_170AAI03000_OUT** type), that will be mapped to the %MW write area after build.

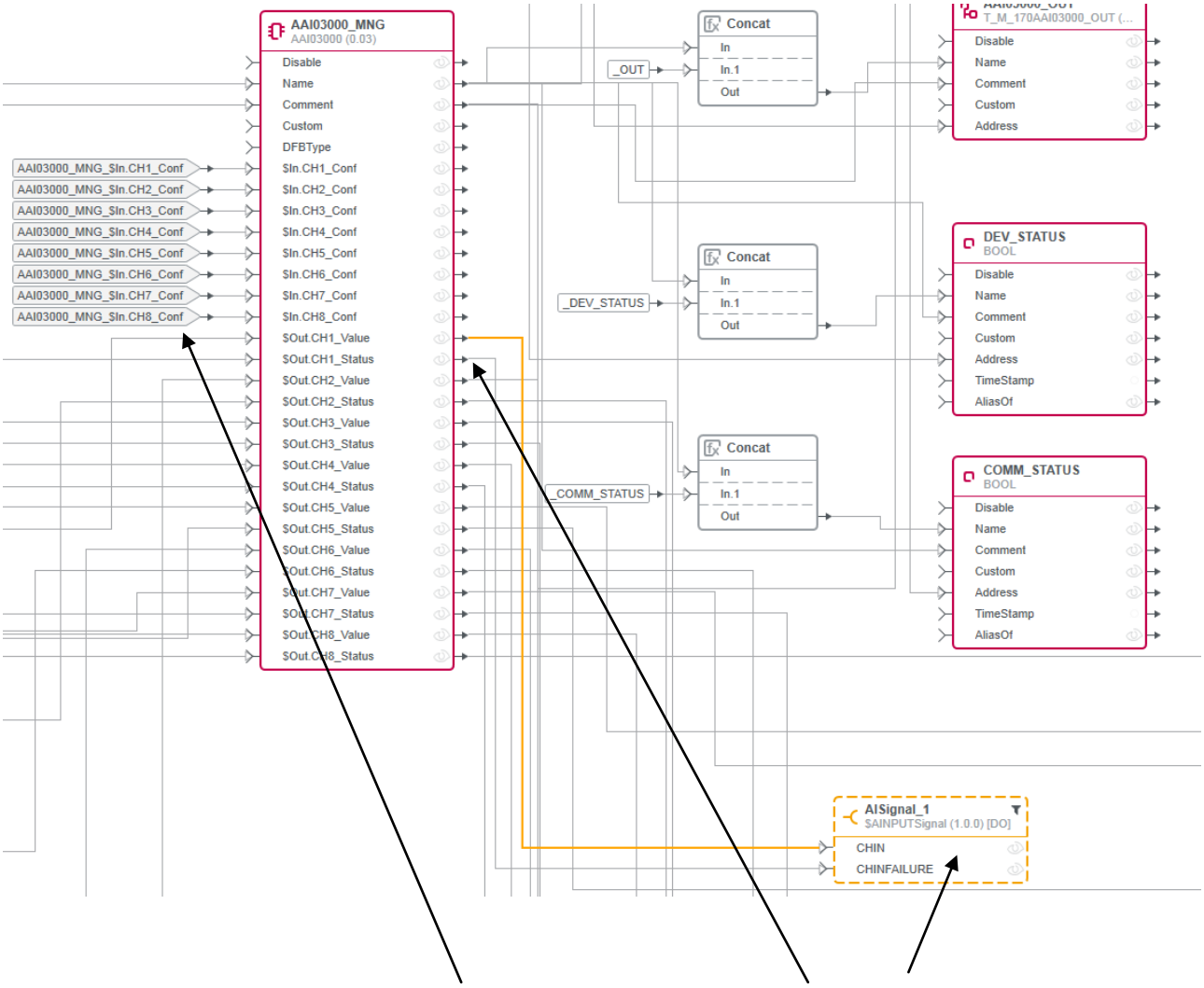
DFB code is shown below.

```
(*type conversion*)
Channel1_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[0],DATA_IN._1
70_AAI_030_00_8i_differential_IN_1[1]));
Channel2_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[2],DATA_IN._1
70_AAI_030_00_8i_differential_IN_1[3]));
Channel3_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[4],DATA_IN._1
70_AAI_030_00_8i_differential_IN_1[5]));
Channel4_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[6],DATA_IN._1
70_AAI_030_00_8i_differential_IN_1[7]));
Channel5_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[8],DATA_IN._1
70_AAI_030_00_8i_differential_IN_1[9]));
Channel6_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[10],DATA_IN._
170_AAI_030_00_8i_differential_IN_1[11]));
Channel7_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[12],DATA_IN._
170_AAI_030_00_8i_differential_IN_1[13]));
Channel8_rawVal:=word_to_int(byte_as_word(DATA_IN._170_AAI_030_00_8i_differential_IN_1[14],DATA_IN._
170_AAI_030_00_8i_differential_IN_1[15]));

(*channel failure when bit 15 is true - broken wire*)
CH1_STATUS:=Channel1_rawVal=32767 or Channel1_rawVal=-32768;
CH2_STATUS:=Channel2_rawVal=32767 or Channel2_rawVal=-32768;
CH3_STATUS:=Channel3_rawVal=32767 or Channel3_rawVal=-32768;
CH4_STATUS:=Channel4_rawVal=32767 or Channel4_rawVal=-32768;
CH5_STATUS:=Channel5_rawVal=32767 or Channel5_rawVal=-32768;
CH6_STATUS:=Channel6_rawVal=32767 or Channel6_rawVal=-32768;
CH7_STATUS:=Channel7_rawVal=32767 or Channel7_rawVal=-32768;
CH8_STATUS:=Channel8_rawVal=32767 or Channel8_rawVal=-32768;

(*outputs to momentum - channel configuration*)
(*each byte contains configuration of two channels. Second channel is then shifted and added*)
DATA_OUT._170_AAI_030_00_8i_differential_OUT_1[0]:=int_to_byte(CH1_CONF+16*CH2_CONF);
DATA_OUT._170_AAI_030_00_8i_differential_OUT_1[1]:=int_to_byte(CH3_CONF+16*CH4_CONF);
DATA_OUT._170_AAI_030_00_8i_differential_OUT_1[2]:=int_to_byte(CH5_CONF+16*CH6_CONF);
DATA_OUT._170_AAI_030_00_8i_differential_OUT_1[3]:=int_to_byte(CH7_CONF+16*CH8_CONF);

(*value outputs*)
CH1_VALUE:=Channel1_rawVal;
CH2_VALUE:=Channel2_rawVal;
CH3_VALUE:=Channel3_rawVal;
CH4_VALUE:=Channel4_rawVal;
CH5_VALUE:=Channel5_rawVal;
CH6_VALUE:=Channel6_rawVal;
CH7_VALUE:=Channel7_rawVal;
CH8_VALUE:=Channel8_rawVal;
```

After creation of the DFB and its associated elements, select FBD section, all DFBs, DDTs and EDTs variables to be imported in Global Templates workspace.

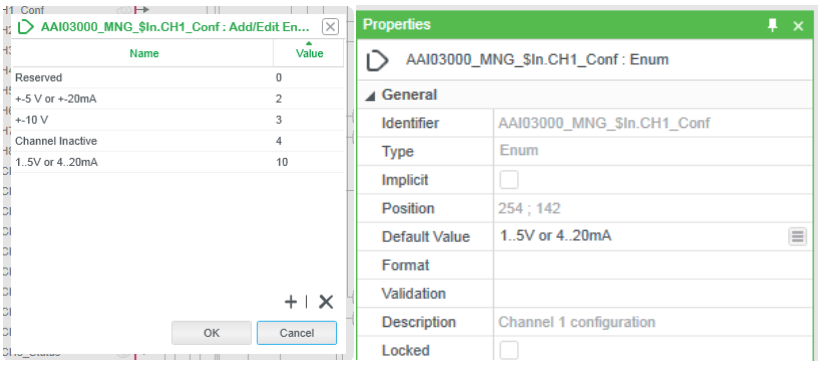


The following images show most important points of the implementation done in the global templates workspace, where we have now represented DFB interface, as well as DDT and EDT variables.



Configuration of Channels by mean of **Enums**

Each signal (value + status) is linked to an $**AInputSignal** DO interface.



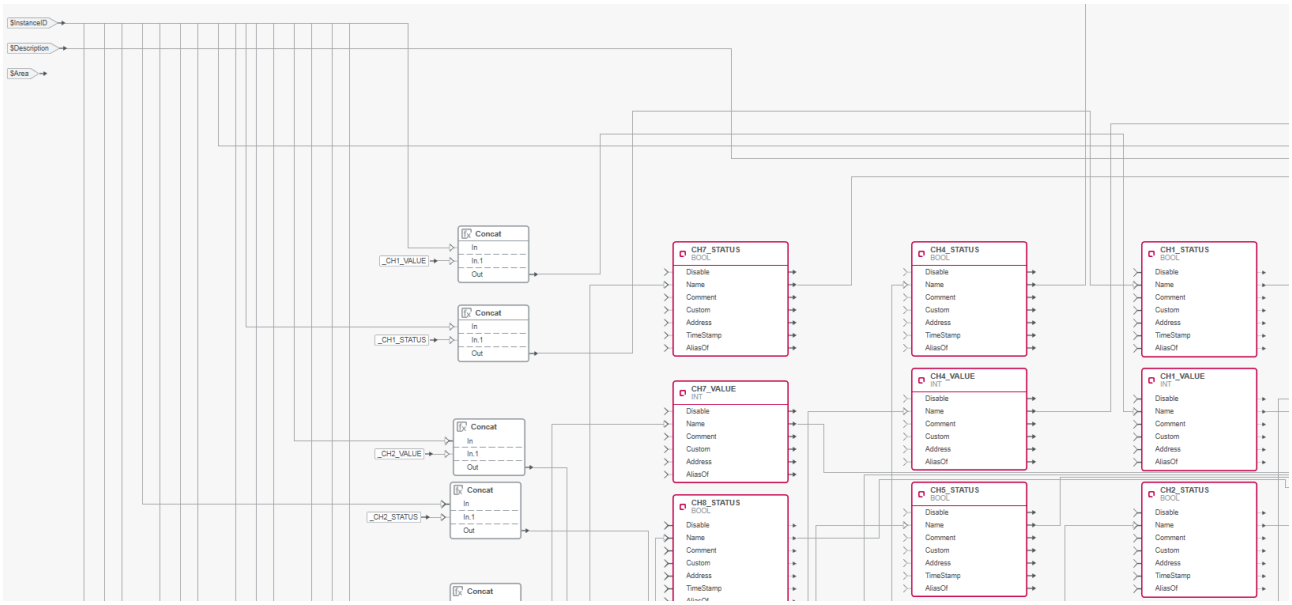
Codes for Analog Input Parameters

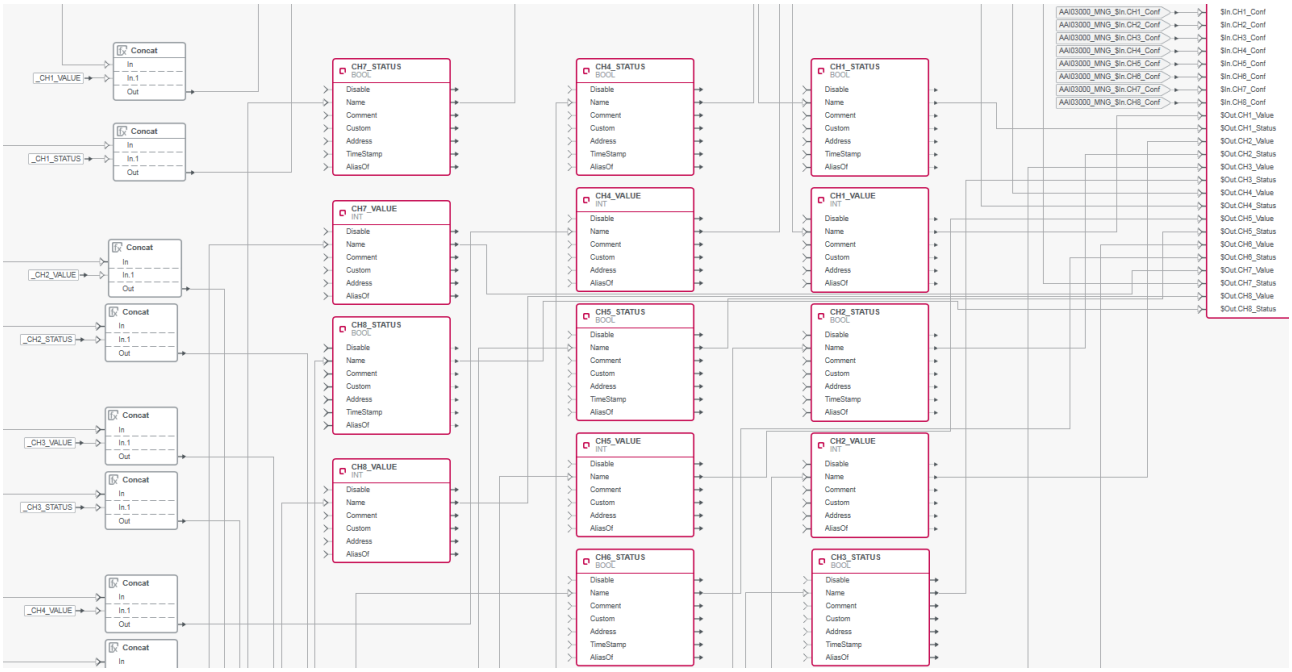
Use the following codes to set the parameters for each analog input channel:

Code (binary)	Code (hex)	Parameter
0000	0	Reserved value (see note below)
0010	2	+/-5V and +/-20mA input range
0011	3	+/-10V input range
0100	4	Channel inactive
1010	A	1 ... 5V and 4 ... 20 mA input range

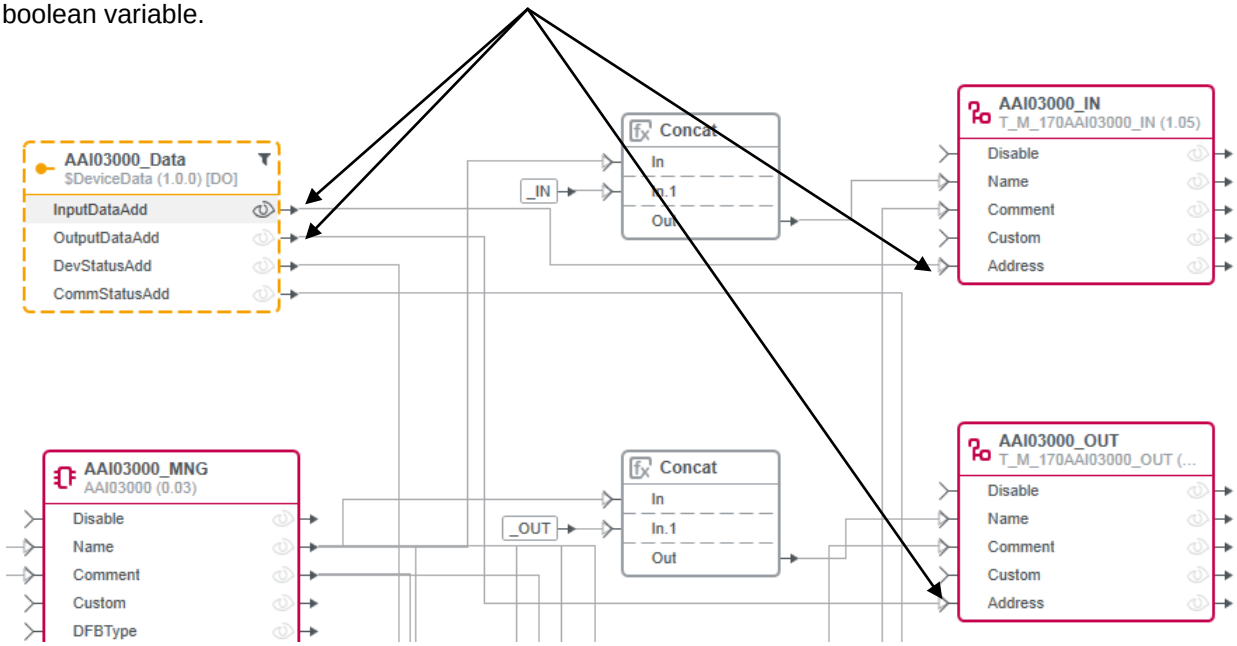
NOTE: The 0000 reserved value is more a control than a parameter. It forces the I/O base into a default condition where it continues to receive field inputs according to the previous channel parameters.

Channel status and value variables are referenced and linked to the DFB object





$**DeviceData** DO interface is connected to IN and OUT DDT structures. This will allow the proper hardware mapping (assignment of %MW to address field). It is also possible to map the Communication status to a boolean variable.

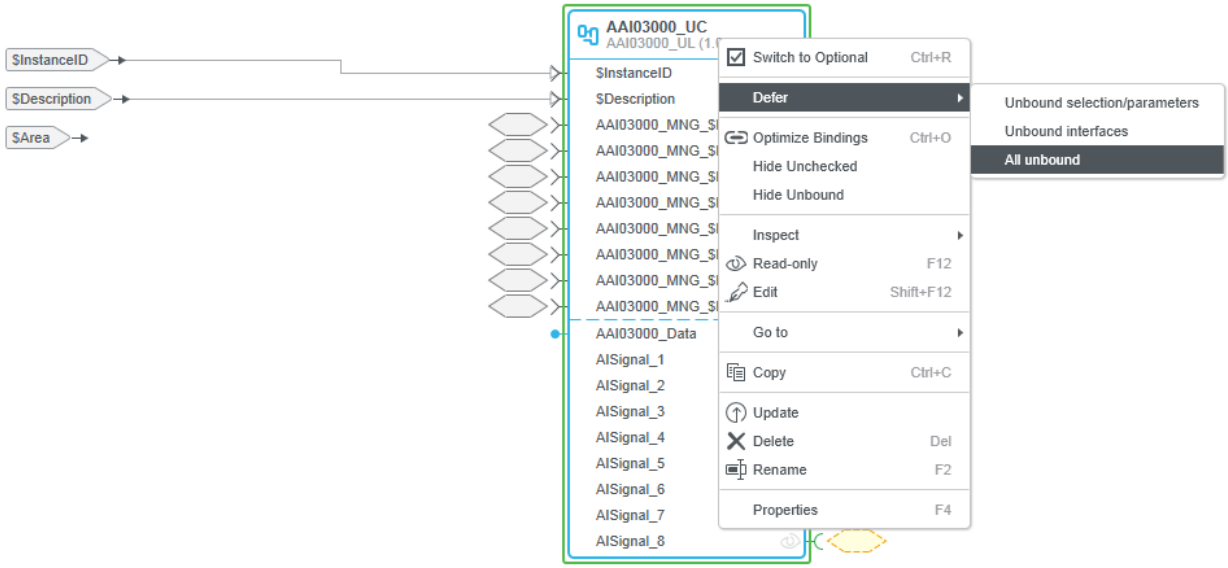


Save the template.

3.4.2 Creation of composite template – AAI03000

It is now possible to create a composite template containing the UL facet described in previous chapter.

Thanks to the Defer mechanism we can propagate parameters and interfaces.



This composite will have these main features:

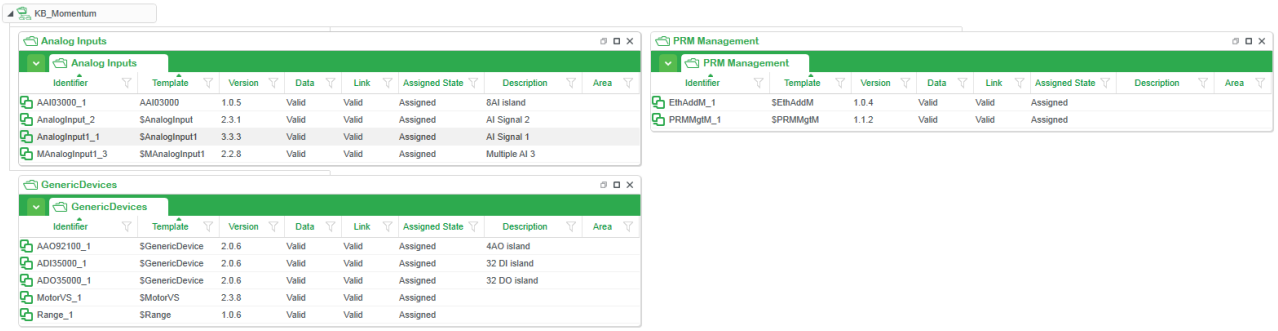
- Interface for data mapping named **AAI03000_Data**
- Configurations of each AI Channel published in Application Manager
- 8 interfaces for each analog input signal (**AISignal_*)**

3.5 Creation of application manager object

In the Application Manager we can now create instances for managing PRM and Momentum I/O bases.

The purpose of this chapter is also to show the benefits of object oriented approach (templatization) compared to usage of $**GenericDevice** instances and consequent refinement.

This is the overall structure of Application Manager of the example.

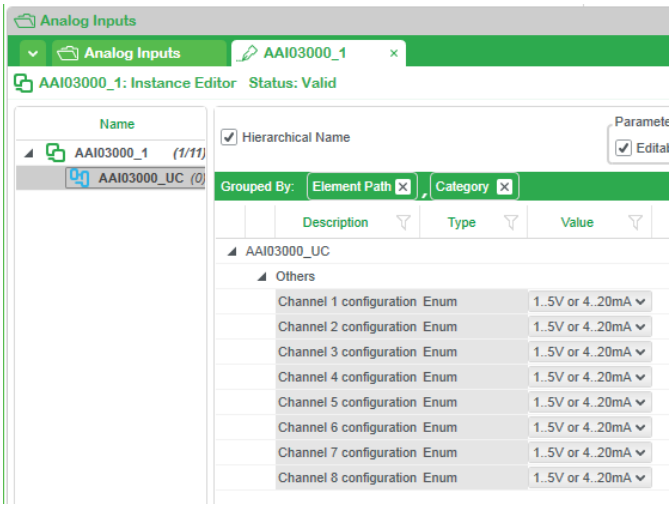


3.5.1 Creation of AAI03000 – object oriented approach

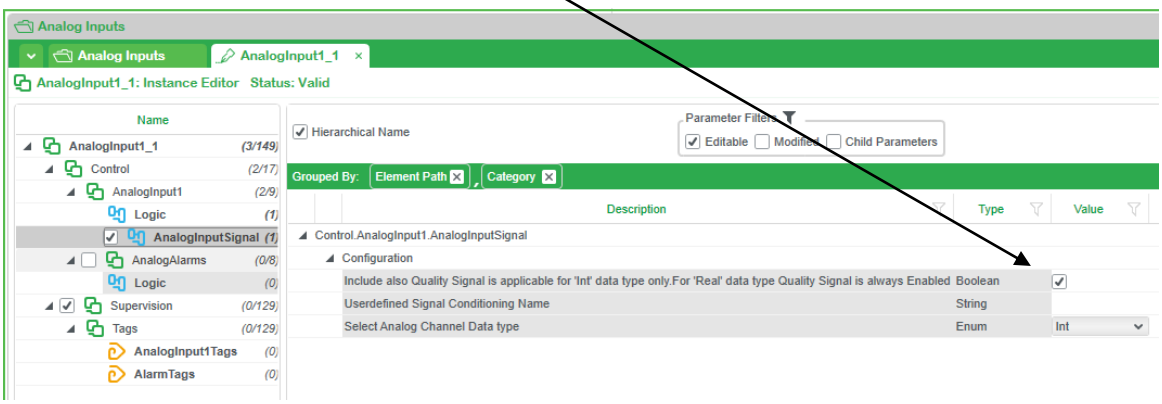
In this example following instances will be created in “AnalogInput” folder:

- 1 **AAI03000** instance (for management of 170AAI03000)
- 1 $**AnalogInput** instance (to be connected to AAI03000 instance, signal 1)
- 1 $**AnalogInput1** instance (to be connected to AAI03000 instance, signal 2)
- 1 **$MAnalogInput1** instance, 4 AI channels enabled (to be connected to AAI03000 instance, signal 3-4-5-6)

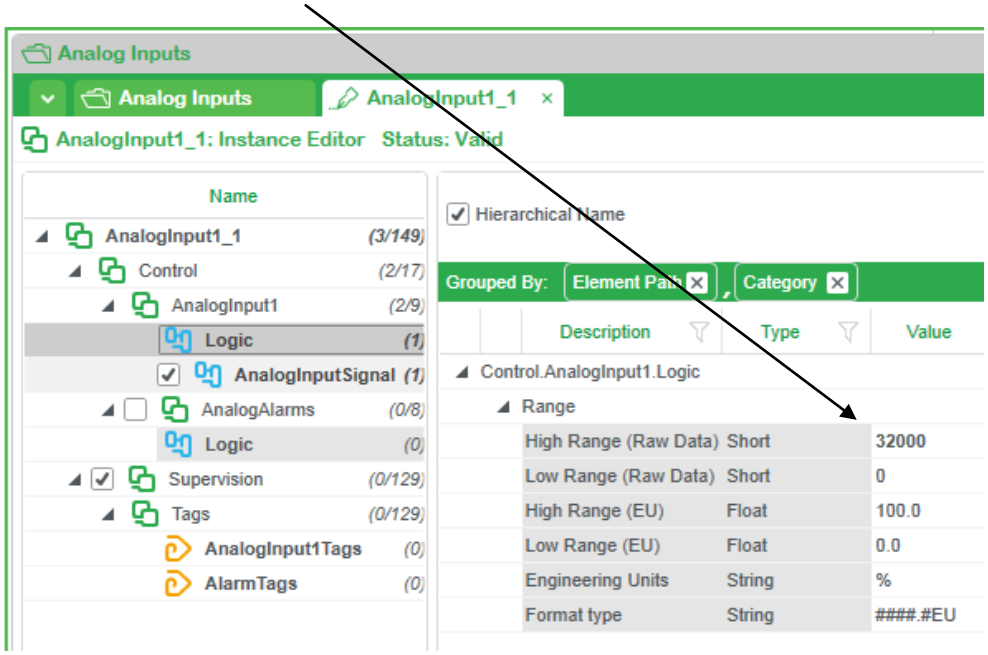
Here we can see the configuration of each block.



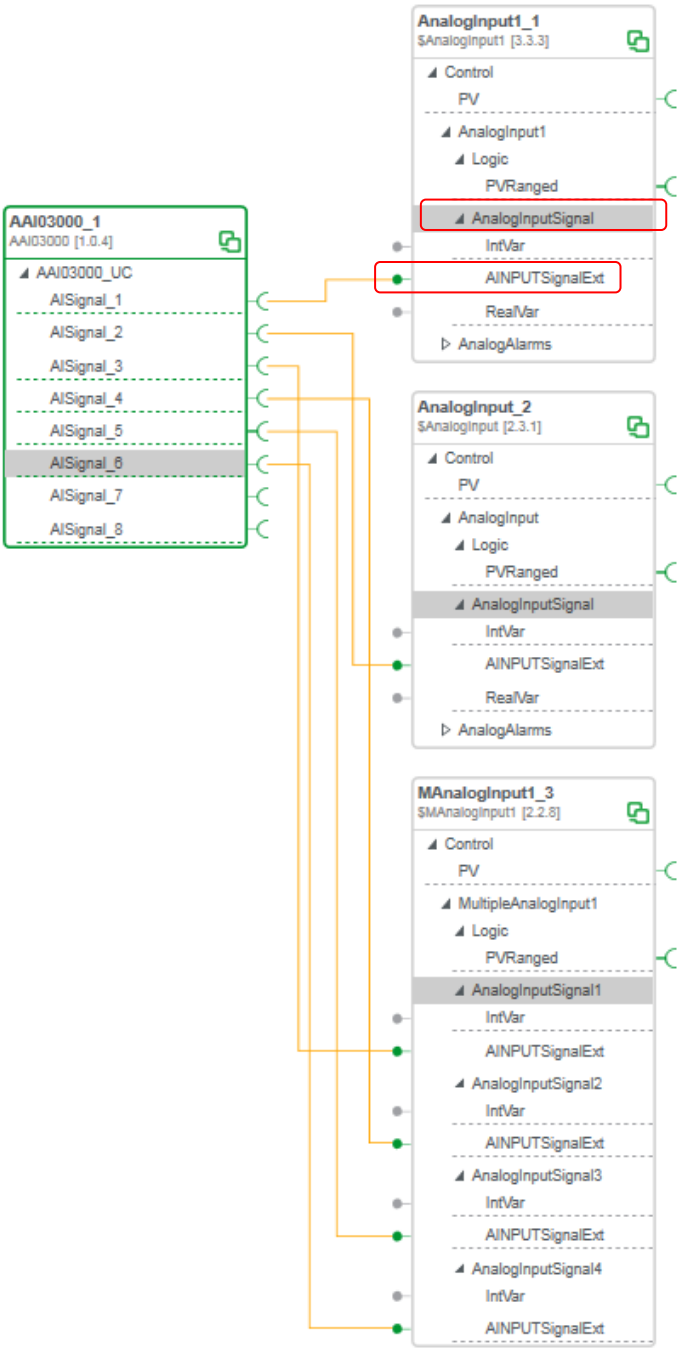
Remember also to enable the Signal Quality for AnalogInputSignal facet. This will allow the connection of the interfaces.



Raw value is always 0-32000 on this island.



After instance creation, it is possible to create links between AAI03000 Momentum object and AnalogInput objects (on **AnalogInputSignal** facet – **AINPUTSignalExt** interface)



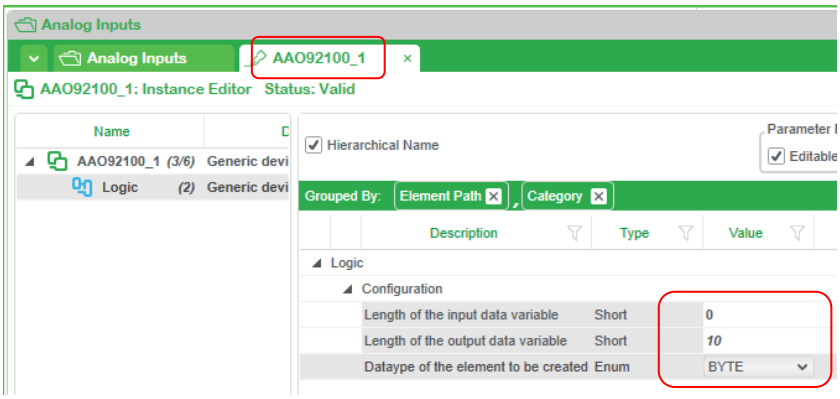
3.5.2 Creation of other momentum I/O bases – Generic Device + refinement approach

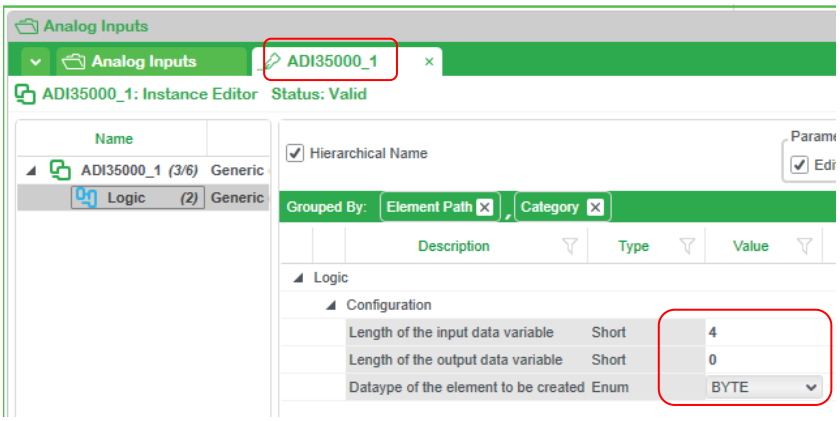
For all other bases, in this example we will use $**GenericDevice** objects.

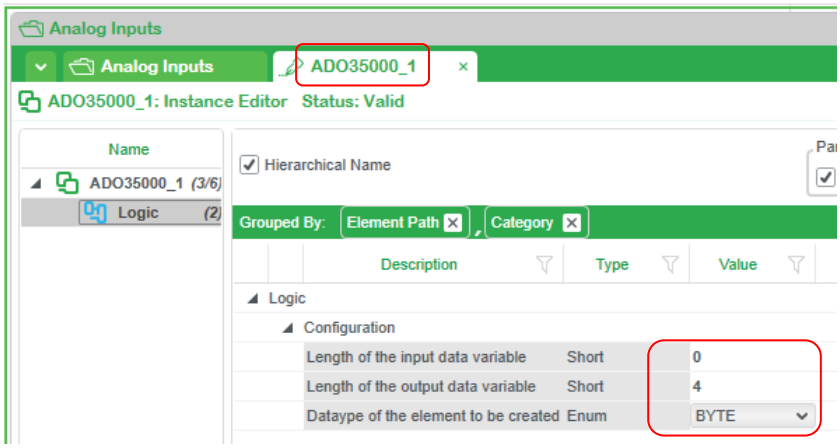
Following instances are created in “GenericDevice” folder:

- 1 **$GenericDevice** instance, 10 bytes output (for management of **170AAO92100**)
- 1 **$GenericDevice** instance, 4 bytes input (for management of **170ADI35000**)
- 1 **$GenericDevice** instance, 4 bytes output (for management of **170ADO35000**)

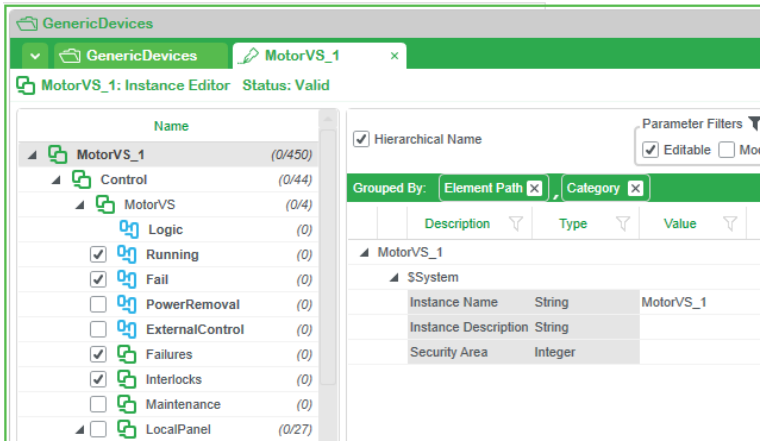
Byte length to assign is correspondent with amount of bytes requested in PRM topology configuration.

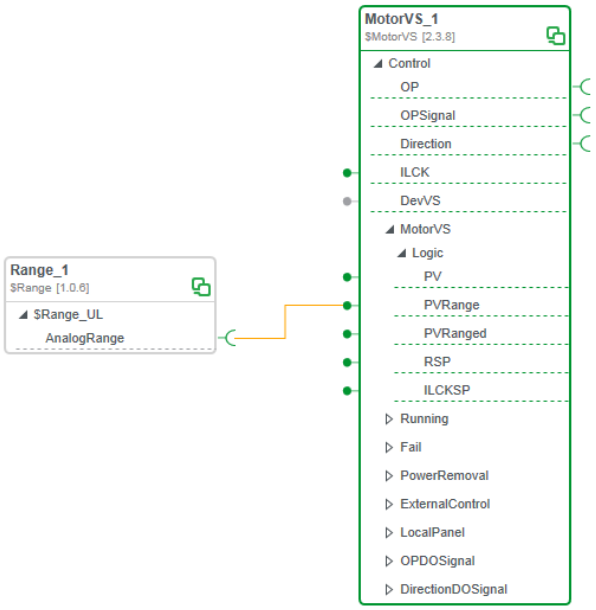






In this example it is created also an instance of **$MotorVS** (and relevant $**Range** for PV) in order to show later on refinement operations.



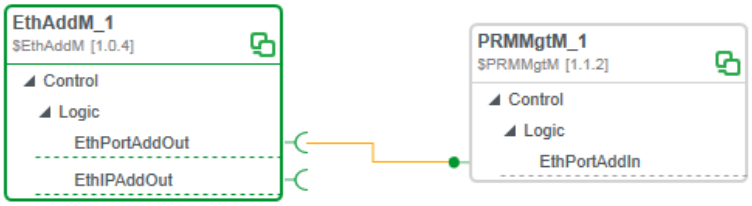


3.5.3 Creation of Profibus PRM objects

For PRM management, following objects will be created

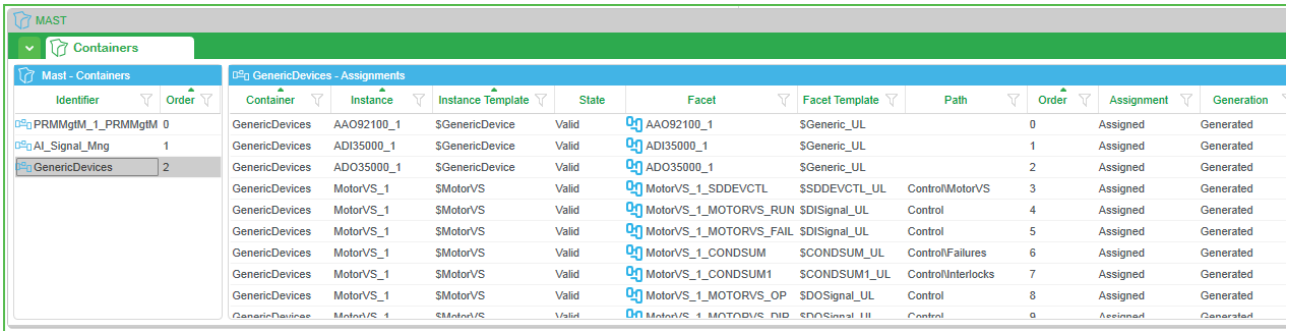
- $**PRMMgt** instance
- $**EthAddIn** instance

These object need to be linked as follows.



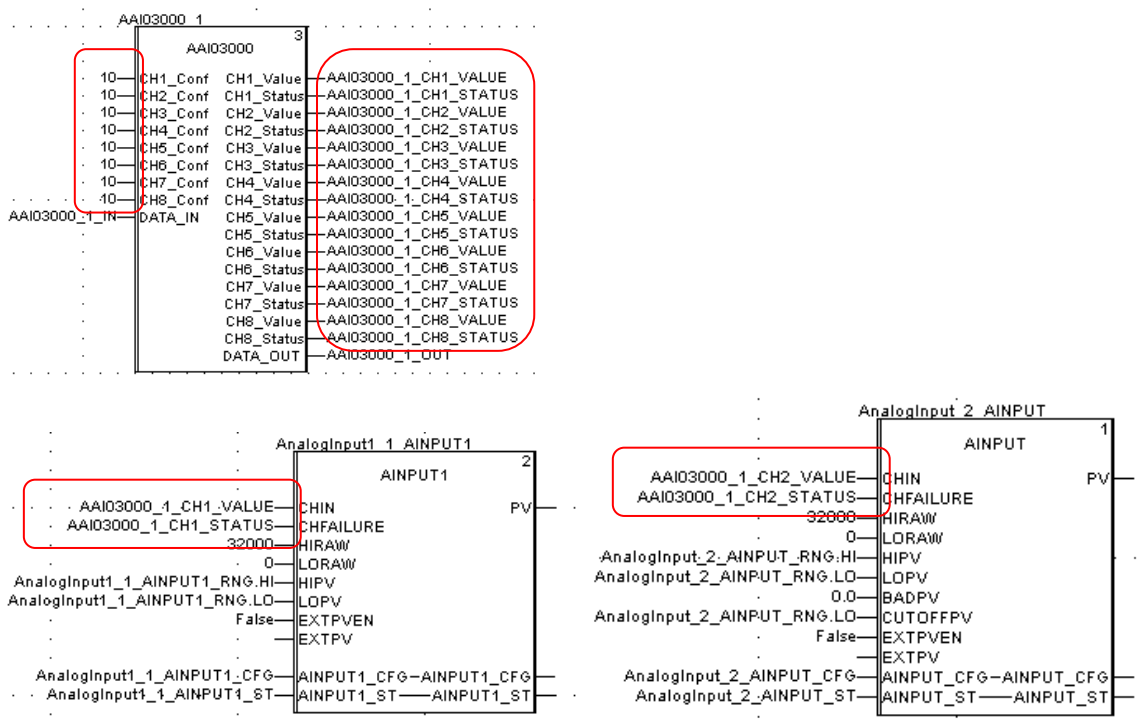
3.6 Project Manager - assignment and generation

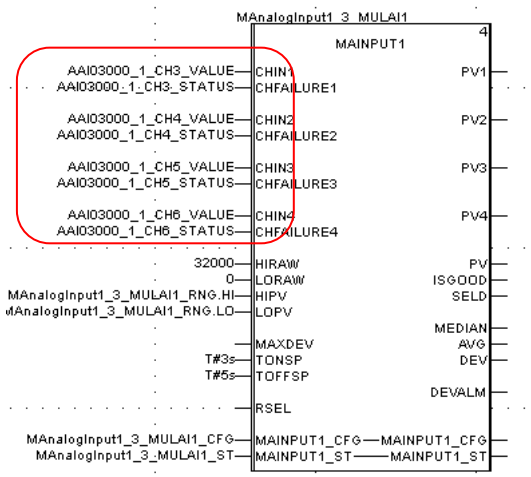
Assign and generate instances.



3.6.1 Generated code for AAI03000

Here the result of the generation for AAI03000 Momentum base. Configuration parameters and links have been properly created by PES, so the code has been completely managed by it without need of refinement.





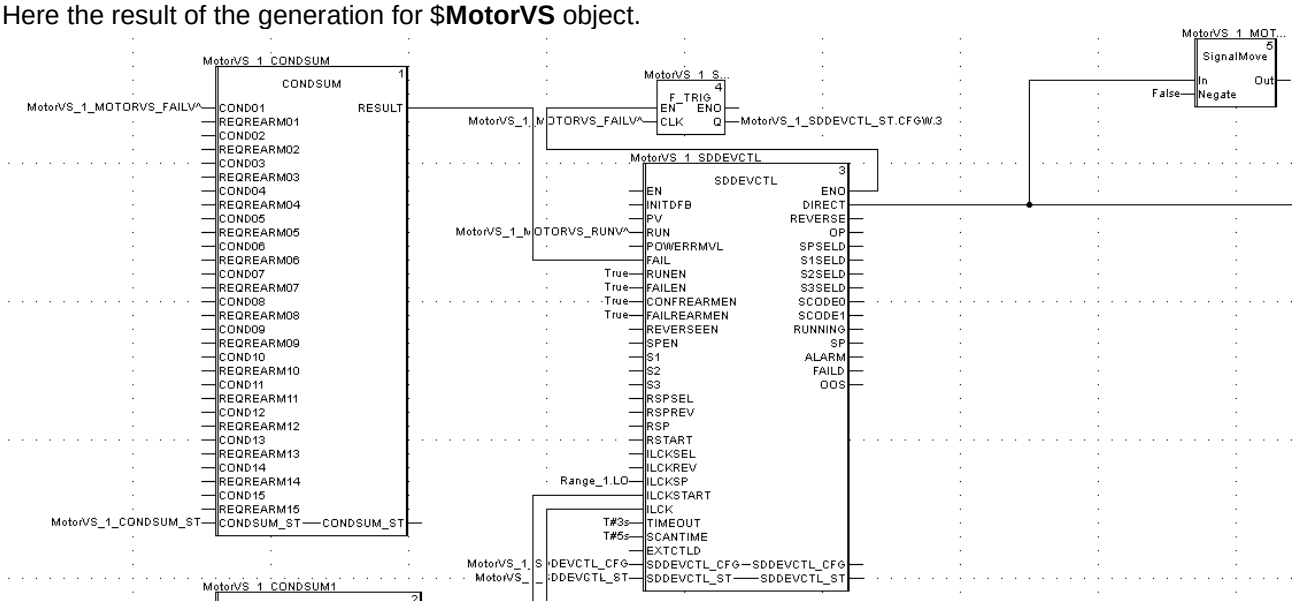
3.6.2 Generated code for Generic Devices

Here the result of the generation for generic devices.

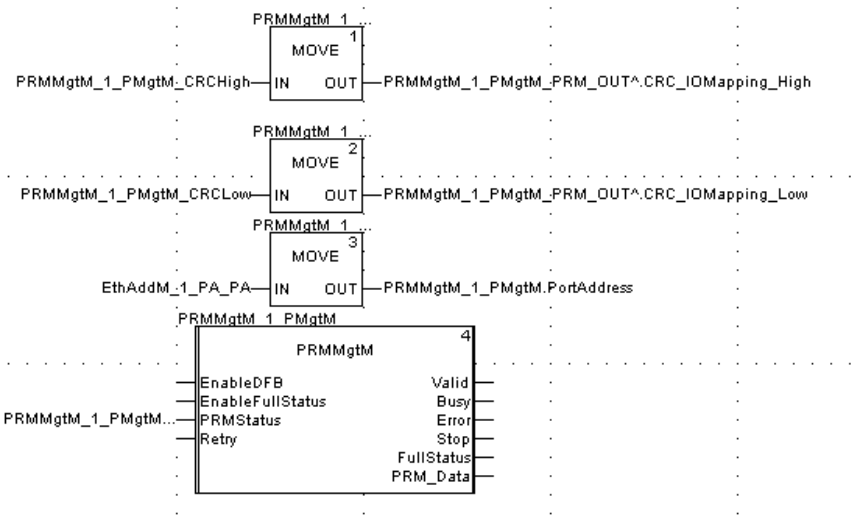
AA092100_1_IOScannerSts	BOOL
AA092100_1_OutputData	ARRAY[0..9] OF BYTE
AI35000_1_InputData	ARRAY[0..3] OF BYTE
AI35000_1_IOScannerSts	BOOL
AO35000_1_IOScannerSts	BOOL
AO35000_1_OutputData	ARRAY[0..3] OF BYTE

PES generates only data blocks related to these devices.
So the management of the information read and written from Profibus DP bus has to be managed by mean of Unity Pro refinement.

Here the result of the generation for $**MotorVS** object.



3.6.3 Generated code for PRM management



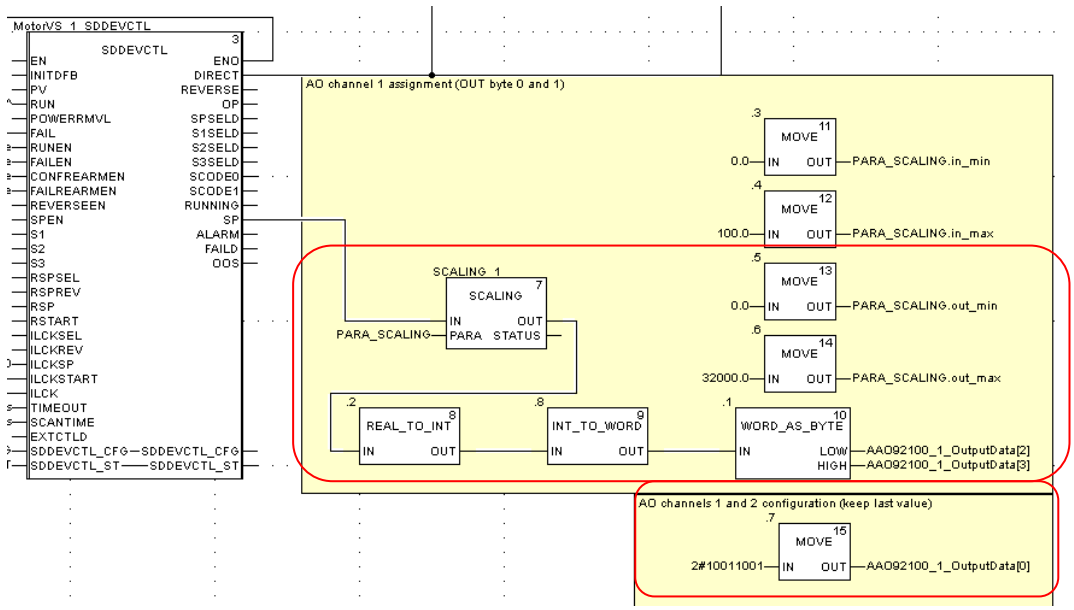
3.7 Project Manager – refinement – Generic Device approach

In this example it is shown how to manage the $MotorVS input/outputs belonging to Momentum islands in refinement mode.

Since this example uses PES 4.1 with Hardware Abstraction Layer 2.0, it is possible to reference the Run feedback (**RUNV**), the Failure feedback (**FAILV**) to **ADI35000** island input data and to reference Direction command (**DIRV**) and Run command (**OPV**) to **ADO35000** island output data.

MotorVS_1_MOTORVS_DIRV	REF_TO ANY_BOOL	REF(ADO35000_1_OutputData[0].0)
MotorVS_1_MOTORVS_FAILV	REF_TO ANY_BOOL	REF(ADI35000_1_InputData[0].1)
MotorVS_1_MOTORVS_OPV	REF_TO ANY_BOOL	REF(ADO35000_1_OutputData[0].1)
MotorVS_1_MOTORVS_RUNV	REF_TO ANY_BOOL	REF(ADI35000_1_InputData[0].0)

Speed setpoint **SP** (which doesn't belong to HAL) can be assigned to DDT OUT.
SP is scaled from 0-100% real to 0-32000 on WORD format, and then copied to the two relevant bytes on DDT out structure of **AAO92100**.
It is also necessary to define the fallback value of the Analog Output channel of the Momentum AO island AAO92100 on first output word.



Here the relevant excerpt from the manual.

Key

This section focuses on output word 1, as highlighted in the table below:

Word	Output Data
1 = LSW	**Parameters for Output channels 1 ... 4**
2	Value, output channel 1
3	Value, output channel 2
4	Value, output channel 3
5= MSW	Value, output channel 4

Illustration

Parameters are set by entering a four-bit code in output word 1, as follows:

Output Word 1 (Register 4x, parameter word)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
for output channel 4				for output channel 3				for output channel 2				for output channel 1			

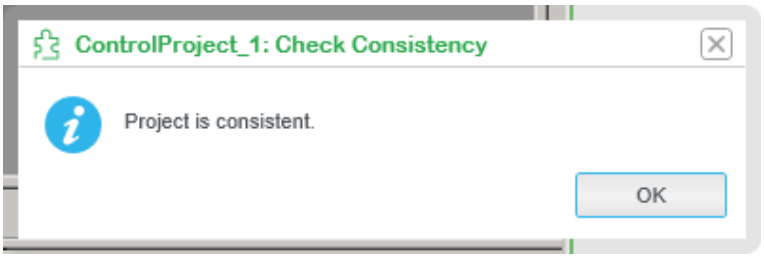
Parameter Codes

The value entered in this word defines the behaviour of the I/O module in case of loss of communication. Each 4-bit nibble in output word 1 must be configured with one of the following binary codes to define the channel parameters. Parameters must be set for all four channels before the module can be commissioned.

In each case, the x may be a 0 or a 1:

Code	Output Parameter)	Function
0000	Reserved value	Forces the I/O base into a default condition where it continues to receive field inputs according to the previous received channel parameters.
00x1	Output to Zero	Sends a value to the base that causes it to apply zero at the field output.
01x1	Full Range	Sends a value to the base that causes it to apply full scale (+10 V or + 20 mA) at the field output.
10x1	Output Last Value	Sends a value to the base that causes it to apply the last received value at the field output.

These refinement operations do not break facets consistency.



3.8 Project Manager – hardware mapping

In this chapter it is shown how to perform the hardware mapping, based on the Topology and Application manager configurations done before.

The following objects need mapping:

- EthAddM_1 (Ethernet port address M340), **EMPortCHM** HW interface
- PRMMgtM_1 (PRM management M340), **PRMCTRLData** and **PRMEthernetAddress** HW interfaces
- AAI03000_1 (template for management of AAI03000 island), **AAI03000_DATA** HW interface
- AAO92100_1 (generic device for management of AAO92100 island), **GenericData** HW interface
- ADI35000_1 (generic device for management of ADI35000 island), **GenericData** HW interface
- ADO35000_1 (generic device for management of ADO35000 island), **GenericData** HW interface

Available for mapping

Instance	Template	HW Mapping IF	HW Mapping Type	Instance	Project Facet	Facet Type	HW Interface	HW Mapping Type
Controller_1 0:Local 0:D 0:R 0:BMXP342020	$M340CPUMBEth	MBControllerSlave.ModbusAddress	$ModbusAddress/HO					
Controller_1 0:Local 0:D 0:R 0:BMXP342020	$M340CPUMBEth	MBTCPEth.EMPortChM	$MEthernetPortCH/HO					
Controller_1 0:Local 0:D 0:R 01:BMXNOE0100.2	$MNOEIOSClient	MBTCPEth.EMPortChM	$MEthernetPortCH/HO	EthAddM_1	EthAddM_1_PA	$EthAddM	EMPortCHM	$MEthernetPortCH/CO
PRM_1 PBDP 1:PRM_Master_0001	$HWPRM	.PRMCTRLData	$PRMCTRLData/HO	PRMMgtM_1	PRMMgtM_1_PMgtM	$PRMMgtM	PRMCTRLData	$PRMCTRLData/DO
PRM_1 PBDP 1:PRM_Master_0001	$HWPRM	.PRMMapping	$PRMMapping/HO	PRMMgtM_1	PRMMgtM_1_PMgtM	$PRMMgtM	PRMEthernetAddress	$PRMMapping/CO
PRM_1 PBDP 11:M_170AAI03000	$GenericPBSlave	.ProfibusDPAddress	$ProfibusDPAddress/HO					
PRM_1 PBDP 11:M_170AAI03000	$GenericPBSlave	DevData.DevData	$DeviceData/HO	AAI03000_1	AAI03000_1	AAI03000	AAI03000_Data	$DeviceData/DO
PRM_1 PBDP 11:M_170AAI03000	$GenericPBSlave	DevInputData.InputData	$DataAdd/HO					
PRM_1 PBDP 11:M_170AAI03000	$GenericPBSlave	DevOutputData.OutputData	$DataAdd/HO					
PRM_1 PBDP 12:M_170AAO92100	$GenericPBSlave	.ProfibusDPAddress	$ProfibusDPAddress/HO					
PRM_1 PBDP 12:M_170AAO92100	$GenericPBSlave	DevData.DevData	$DeviceData/HO	AAO92100_1	AAO92100_1	$GenericDevice	GenericData	$DeviceData/DO
PRM_1 PBDP 12:M_170AAO92100	$GenericPBSlave	DevInputData.InputData	$DataAdd/HO					
PRM_1 PBDP 12:M_170AAO92100	$GenericPBSlave	DevOutputData.OutputData	$DataAdd/HO					
PRM_1 PBDP 13:M_170ADI35000	$GenericPBSlave	.ProfibusDPAddress	$ProfibusDPAddress/HO					
PRM_1 PBDP 13:M_170ADI35000	$GenericPBSlave	DevData.DevData	$DeviceData/HO	ADI35000_1	ADI35000_1	$GenericDevice	GenericData	$DeviceData/DO
PRM_1 PBDP 13:M_170ADI35000	$GenericPBSlave	DevInputData.InputData	$DataAdd/HO					
PRM_1 PBDP 13:M_170ADI35000	$GenericPBSlave	DevOutputData.OutputData	$DataAdd/HO					
PRM_1 PBDP 14:M_170ADO35000	$GenericPBSlave	.ProfibusDPAddress	$ProfibusDPAddress/HO					
PRM_1 PBDP 14:M_170ADO35000	$GenericPBSlave	DevData.DevData	$DeviceData/HO	ADO35000_1	ADO35000_1	$GenericDevice	GenericData	$DeviceData/DO
PRM_1 PBDP 14:M_170ADO35000	$GenericPBSlave	DevInputData.InputData	$DataAdd/HO					
PRM_1 PBDP 14:M_170ADO35000	$GenericPBSlave	DevOutputData.OutputData	$DataAdd/HO					

MotorVS signal facets do not need hardware mapping.
CRAIPAddress HW interface of EthAddM_1 object doesn't need hardware mapping as well in this case.
So the following HW facets will remain unassigned.

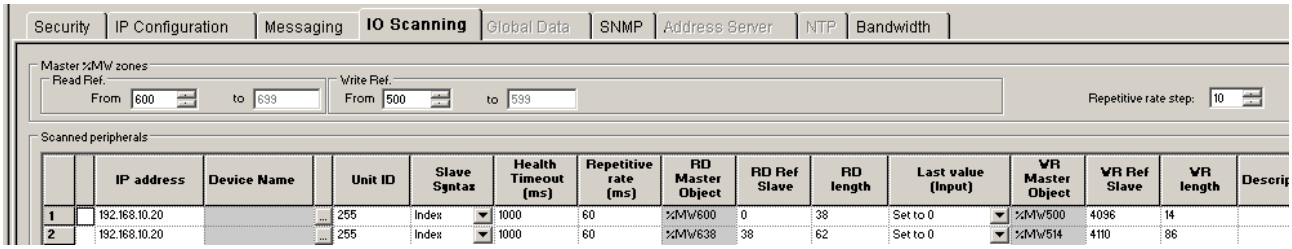
Available for mapping

Instance	Project Facet	Facet Type	HW Interface	HW Mapping Type
EthAddM_1	EthAddM_1_PA	$EthAddM	CRAIPAddress	$EthernetAddress/CO
MotorVS_1	MotorVS_1_MOTORVS_RUN	$MotorVS	DIChannel	$DIChannel/SO
MotorVS_1	MotorVS_1_MOTORVS_FAIL	$MotorVS	DIChannel	$DIChannel/SO
MotorVS_1	MotorVS_1_MOTORVS_OP	$MotorVS	DOChannel	$DOChannel/SO
MotorVS_1	MotorVS_1_MOTORVS_DIR	$MotorVS	DOChannel	$DOChannel/SO

3.9 Project Manager – result of build

In this chapter it is shown the result of Build operation.

PRM IO Scanner line is shown below.



PRM management object %MW assignment is shown below (in this case it uses HAL 2.0 functionality REF_TO*)

PRMMgtM_1_PMgtM_PRM_IN	REF_TO T_PRM_IN		REF(%MW600)
PRMMgtM_1_PMgtM_PRM_OUT	REF_TO T_PRM_OUT		REF(%MW500)

AAI03000 IN and OUT DDT %MW assignment is shown below.

Name	Type	Address
AAI03000_1_IN	T_M_170AAI03000_IN	%MW627
AAI03000_1_OUT	T_M_170AAI03000_OUT	%MW503

AAO92100, **ADI35000** and **AD035000** generic data arrays %MW assignment is shown below

Name	Type	Address
AAO92100_1_OutputData	ARRAY[0..9] OF BYTE	%MW505
ADI35000_1_InputData	ARRAY[0..3] OF BYTE	%MW635
ADO35000_1_OutputData	ARRAY[0..3] OF BYTE	%MW511

These assignment are consistent with PRM configuration.

Refinement modifications for MotorVS management are also kept.

MotorVS_1_MOTORVS_DIRV	REF_TO ANY_BOOL		REF(ADO35000_1_OutputData[0].0)
MotorVS_1_MOTORVS_FAILV	REF_TO ANY_BOOL		REF(ADI35000_1_InputData[0].1)
MotorVS_1_MOTORVS_OPV	REF_TO ANY_BOOL		REF(ADO35000_1_OutputData[0].1)
MotorVS_1_MOTORVS_RUNV	REF_TO ANY_BOOL		REF(ADI35000_1_InputData[0].0)

4 CASE STUDY 2: MODBUS TCP/IP COMMUNICATIONS

In this case the integration of the device is made using the resources that you have in the normal installation of the system, you do not need to install an extension and the process to define the characteristics of each IO Base must be made one by one instead to use a wizard like the Profibus case detailed previously.

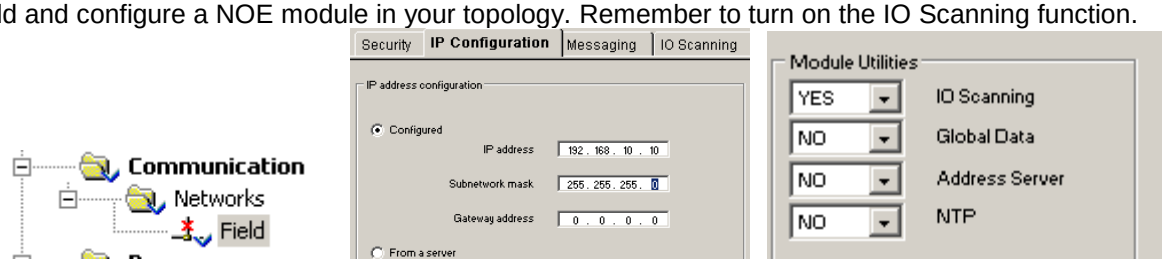
For this example we use the same configuration but changed the communication to Modbus TCP. The addresses are the following:

- M340 PAC with BMXNOE0100.2 IP Address: 192.168.10.10
- Ethernet Momentum I/O base 1: 170AAI 030 00 (8 AI) – IP Address 192.168.10.11
- Ethernet Momentum I/O base 2: 170AAO92100 (4 AO) – IP Address 192.168.10.12
- Ethernet Momentum I/O base 3: 170ADI35000 (32 DI) – IP Address 192.168.10.13
- Ethernet Momentum I/O base 4: 170ADO35000 (32 DI) – IP Address 192.168.10.14

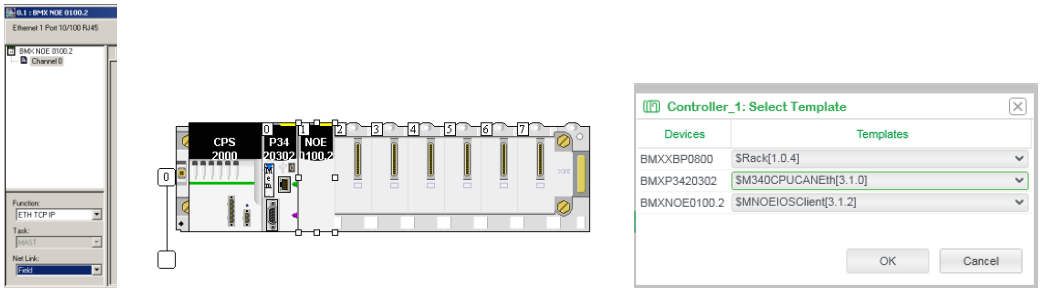
4.1 Controller configuration

The steps to define each base are the following:

Add and configure a NOE module in your topology. Remember to turn on the IO Scanning function.

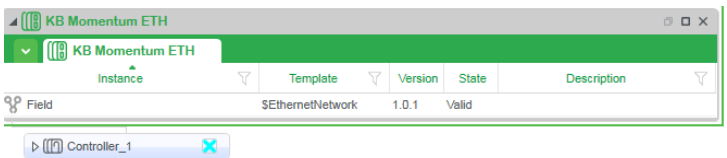


Also remember to link this configuration to the module. Select the Template and version to model each element. Save and close.



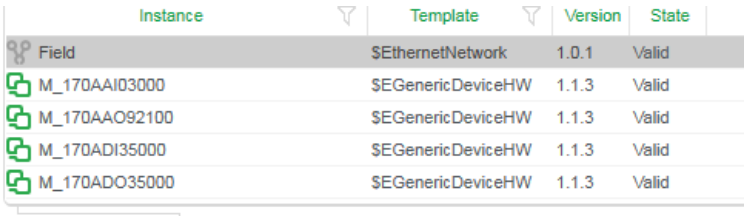
4.2 Creation of Bases in Topological Manager using $EGenericDeviceHW

Add a new Ethernet network and named Field.

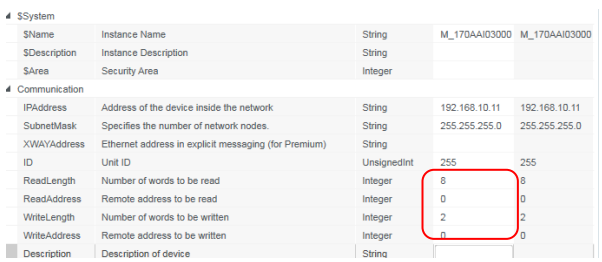


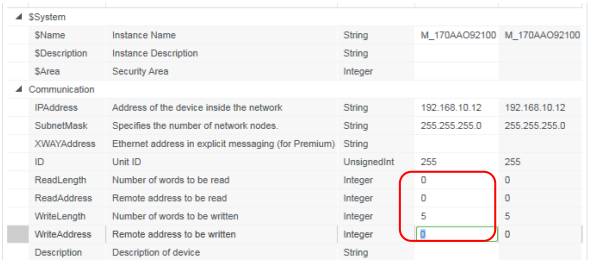
And now add a Modbus TCP Generic Device from the Foundation Library.

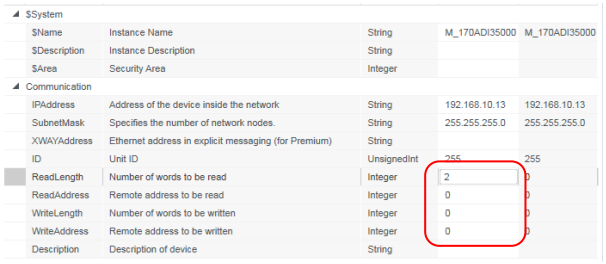
Repeat for each base in the example. The result must look like as

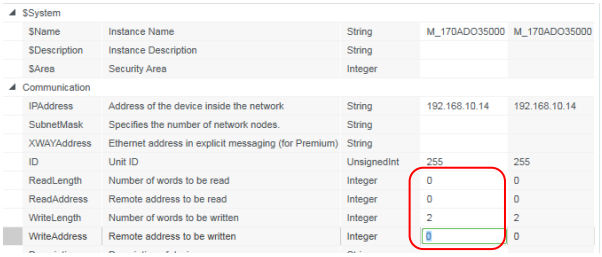


Fill the data required for the template. From the list given in section 2 check the amount of data required for each IO Base. Also define the IP as showed before.

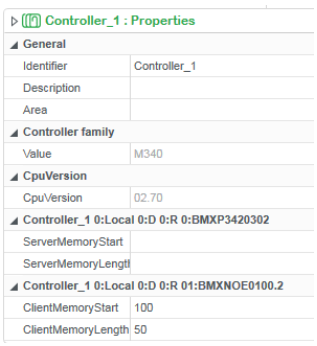


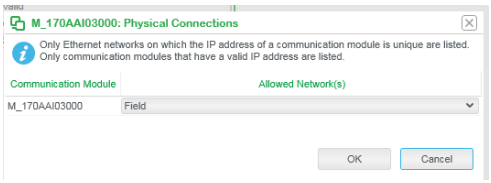


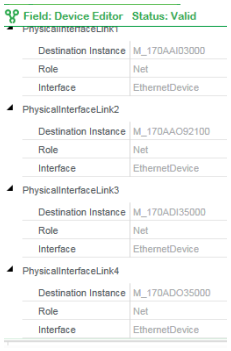




Don´t forget to add an area in the memory of the controller to manage data of the I/O bases. Configure the Physical Connection as the same in all elements.







4.3 Creation of Project and Map Services

Create a new Control Project, Generate and create a new Executable. Map the Services to the controller and Map Communications. This screen must appears:

ControlProject_1.ControlExecutable_1: Communication Mapping Editor

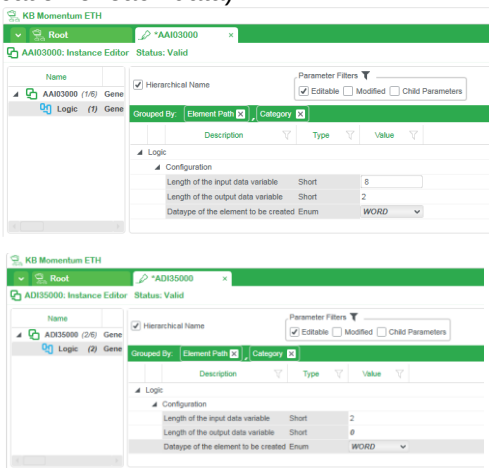
Controller_1(Server) | Controller_1 0:Local 0:D 0:R 01:BMXNO...

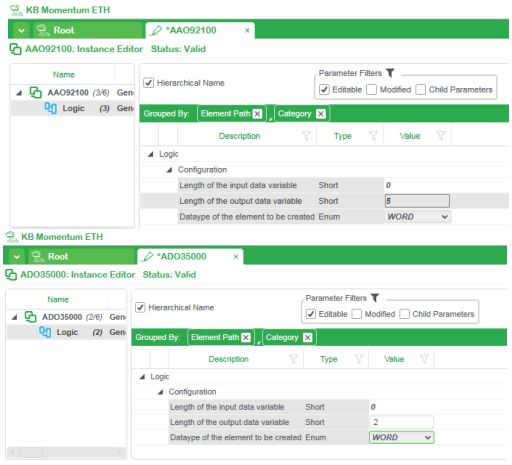
Communication Channels

Type	Service	Server	Direction	Description	Size	Free	Last	Scan	Timeout...
Device IO	IOSDev.IOScanner	M_170AAI03000	Read		8	8	Hold	60	1000
			Write		2	2			
Device IO	IOSDev.IOScanner	M_170AAO92100	Read		0	0	Hold	60	1000
			Write		5	5			
Device IO	IOSDev.IOScanner	M_170ADI35000	Read		2	2	Hold	60	1000
			Write		0	0			
Device IO	IOSDev.IOScanner	M_170ADO35000	Read		0	0	Hold	60	1000
			Write		2	2			
Free					31				

4.4 Creation of Bases in Application Manager using $GenericDevice

Using the same approach showed in the Profibus example, we use a $GenericDevice template for each IO Base. In this version we use WORD data type instead of BYTE (BYTE type is mandatory in Profibus case for the identification of each data)





As usual now we have to assign facets and generate

ControlProject_1: Assignment Editor

Instances

Instance	Template	Version	Assigned State	Description
KB Momentum ETH				
AAI03000	$GenericDevice	2.0.6	Assigned	
AAO92100	$GenericDevice	2.0.6	Assigned	
ADI35000	$GenericDevice	2.0.6	Assigned	
ADO35000	$GenericDevice	2.0.6	Assigned	

ADO35000 - Facets

Click on this icon '>>' on instances pane to load facets here.

ControlProject_1 - Containers

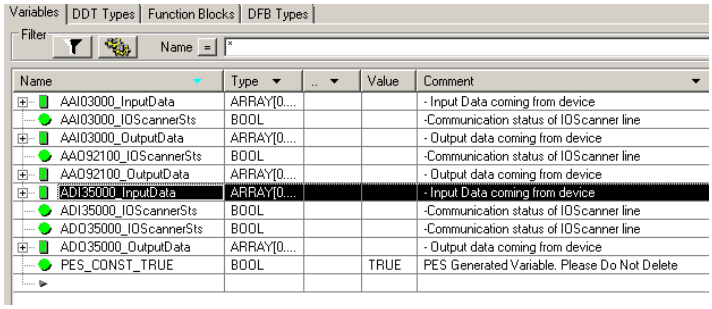
Path	Container
.\	ControlProject_1
.\ControlProject_1\Tasks\Mast	FBDSection_1

FBDSection_1 - Assignments — Assign Candidate Facets Automatically

Container	Instance	Instance Template	State	Facet	Facet Template	Path	Order	Assignment	Generation
FBDSection_1	AAI03000	$GenericDevice	Valid	AAI03000	$Generic_UL		0	Assigned	Non Generated
FBDSection_1	AAO92100	$GenericDevice	Valid	AAO92100	$Generic_UL		1	Assigned	Non Generated
FBDSection_1	ADI35000	$GenericDevice	Valid	ADI35000	$Generic_UL		2	Assigned	Non Generated
FBDSection_1	ADO35000	$GenericDevice	Valid	ADO35000	$Generic_UL		3	Assigned	Non Generated

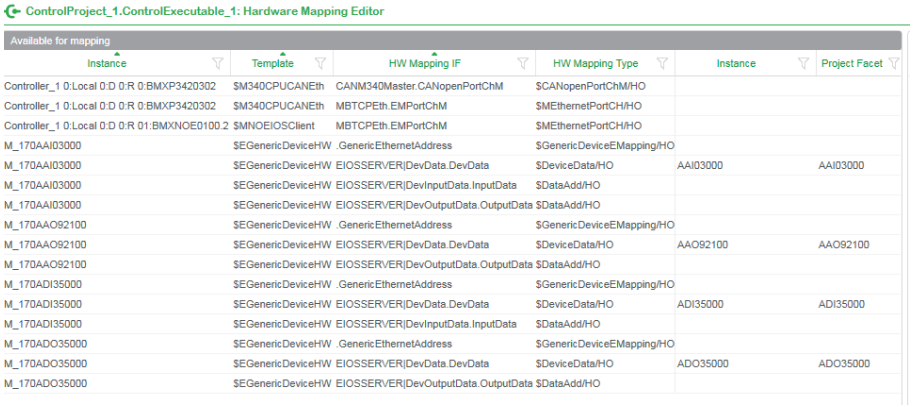
4.5 Data types associated to Bases.

As previously mentioned, when you refine the project data blocks associated to each Base will appear. The amount of registers defined is the same that the one configured in the previous steps. No DFBs is generated in the program section.

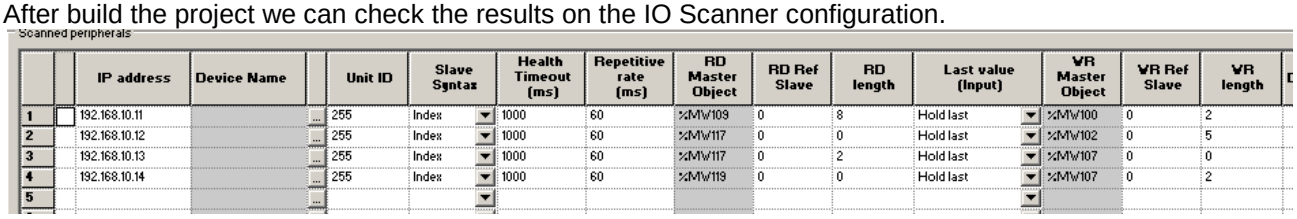


4.6 Mapping the hardware

Now we have to map the hardware associated to GenericDevice instances.



After build the project we can check the results on the IO Scanner configuration.



Finally the data areas associated to the variables previously generated is shown in this image.