ctrlX - CORE

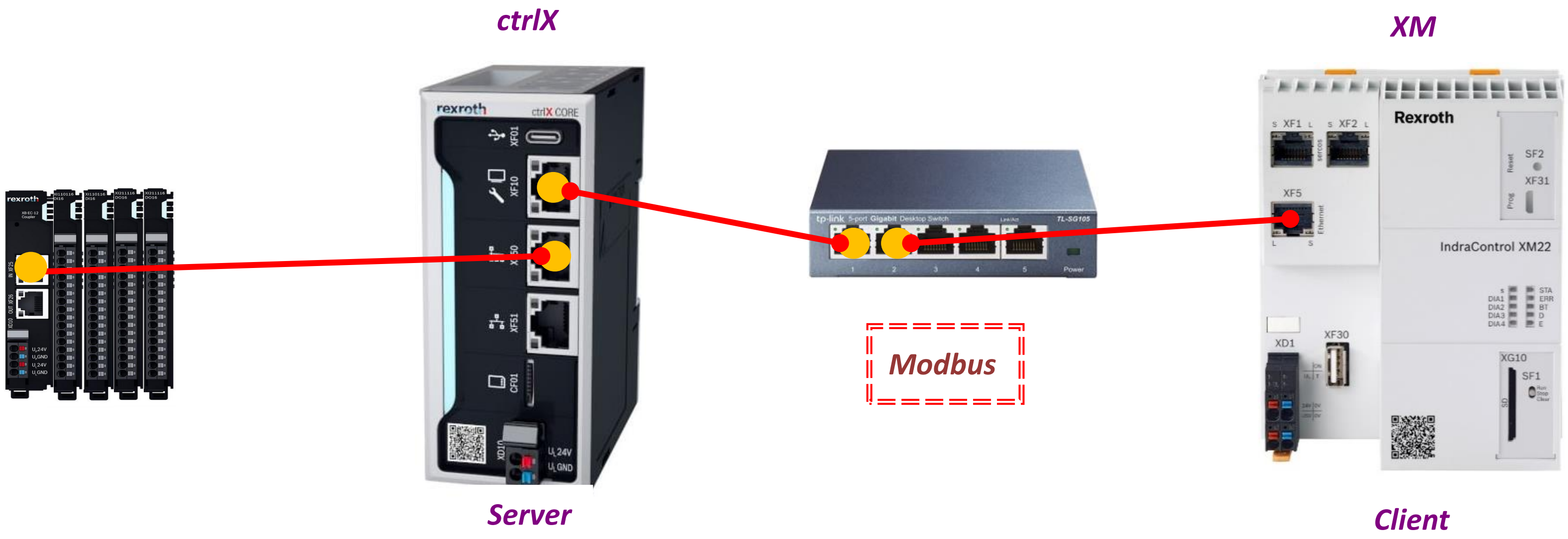
- ## ctrlX I/O

- Configuration of the ctrlX I/O
- Annex: Communication ctrlX < == > XM
- ctrlX sample program
- XM sample program

Jordi Laboria (DCET/SLF4-ES)

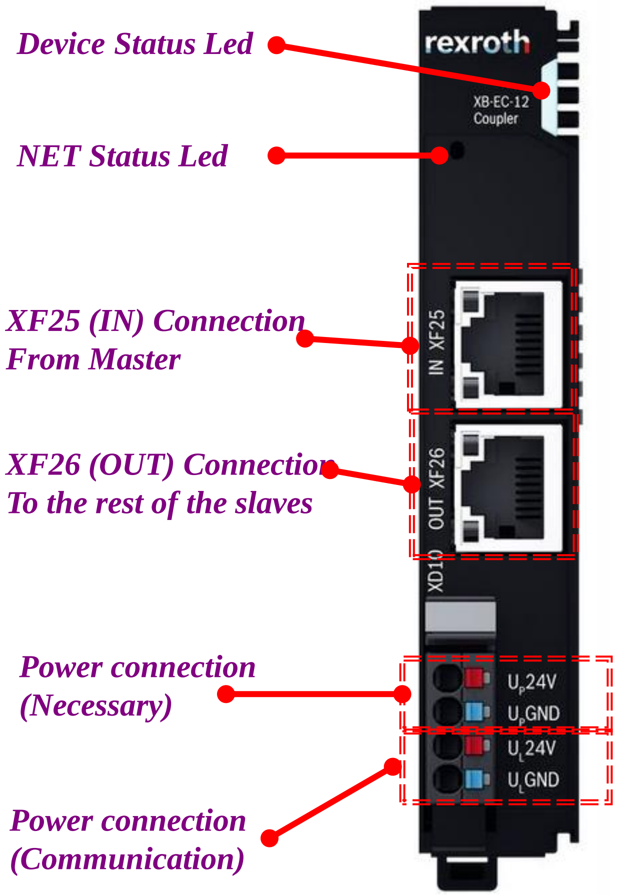
GOALS:

- Configuration of the I/O Modules (Digital Inputs / Outputs)
- Modbus communication for input control and output activation from the XM



Configuration of the ctrlX I/O

Connection of the EtherCat Bus Coupler



Both connections are required

The NET status LED is specified by the ETG (EtherCAT Technology Group) and indicates the EtherCAT bus state at the bus coupler.

The operating state is displayed in green GN:

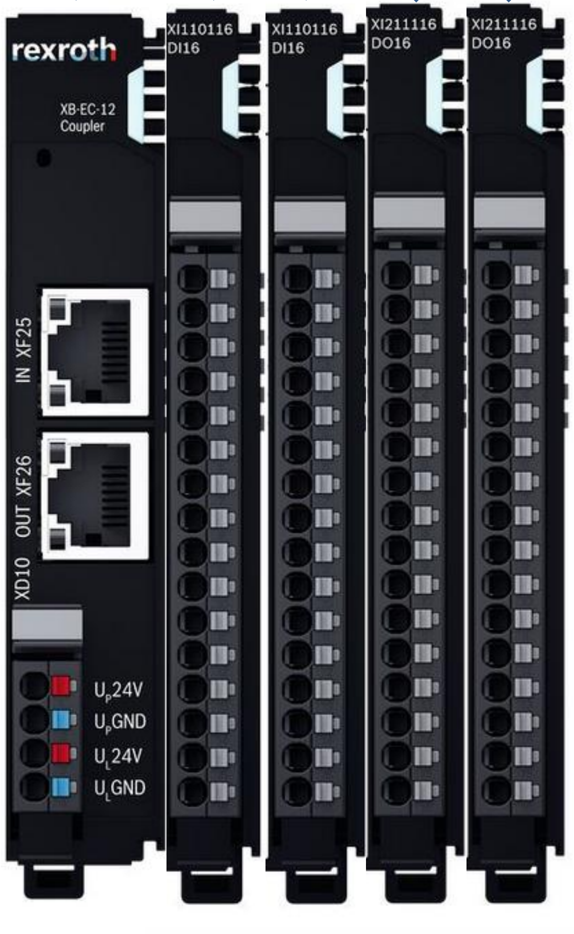
LED color green	Description
Off	Status INIT
Flickers	Status BOOT
Flashes	Status PRE-OP
Single flash	Status SAFE-OP
Lit	Status OP

The error state is displayed in red RD:

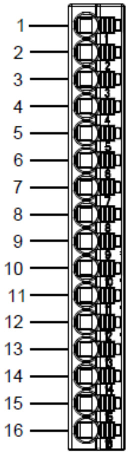
LED color red	Description
Off	No error
Flickers	Boot error
Flashes	Invalid configuration
Single flash	Local error (e.g. synchronization)
Double flash	Watchdog error
Lit	Communication error

In the example we will use the following configuration

Bus Coupler XB-EC-12 | *Inputs XI110116* | *Outputs XI211116*

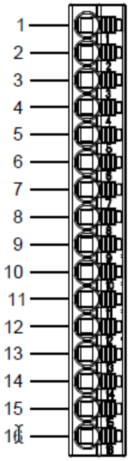


Inputs XI110116



Clamping point	Assignment	Color
1	1 - DI CH.1	Grey
2	2 - DI CH.2	Grey
3	3 - DI CH.3	Grey
4	4 - DI CH.4	Grey
5	5 - DI CH.5	Grey
6	6 - DI CH.6	Grey
7	7 - DI CH.7	Grey
8	8 - DI CH.8	Grey
9	9 - DI CH.9	Grey
10	10 - DI CH.10	Grey
11	11 - DI CH.11	Grey
12	12 - DI CH.12	Grey
13	13 - DI CH.13	Grey
14	14 - DI CH.14	Grey
15	15 - DI CH.15	Grey
16	16 - DI CH.16	Grey

Outputs XI211116

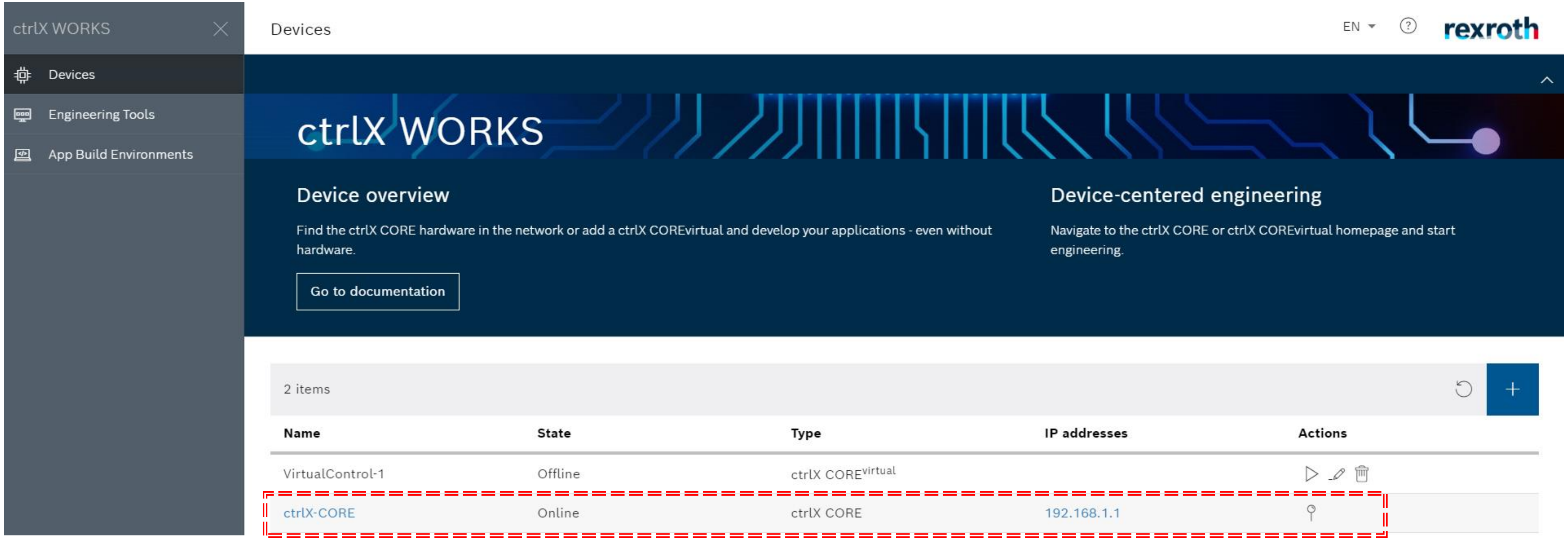


Clamping point	Assignment	Color	Max. current
1	1 - DO CH.1	Grey	0.5 A
2	2 - DO CH.2	Grey	0.5 A
3	3 - DO CH.3	Grey	0.5 A
4	4 - DO CH.4	Grey	0.5 A
5	5 - DO CH.5	Grey	0.5 A
6	6 - DO CH.6	Grey	0.5 A
7	7 - DO CH.7	Grey	0.5 A
8	8 - DO CH.8	Grey	0.5 A
9	9 - DO CH.9	Grey	0.5 A
10	10 - DO CH.10	Grey	0.5 A
11	11 - DO CH.11	Grey	0.5 A
12	12 - DO CH.12	Grey	0.5 A
13	13 - DO CH.13	Grey	0.5 A
14	14 - DO CH.14	Grey	0.5 A
15	15 - DO CH.15	Grey	0.5 A
16	16 - DO CH.16	Grey	0.5 A

The "Device State" signals are identical in all the devices

Device state	LED flashing pattern
Booting	BU BU BU BU BU – – – – –
Initialization	BU BU BU BU BU BU BU BU BU BU
It is currently configured. Module not yet ready.	GN GN GN GN GN – – – – –
Process data transmission, outputs inactive.	GN GN GN GN GN GN GN GN GN –
Module in "Run" state	GN GN GN GN GN GN GN GN GN GN
Error and warning states	
Logic or peripheral voltage error	RD RD RD RD RD RD RD RD RD RD
Communication or configura-tion error	RD RD RD RD RD – – – – –

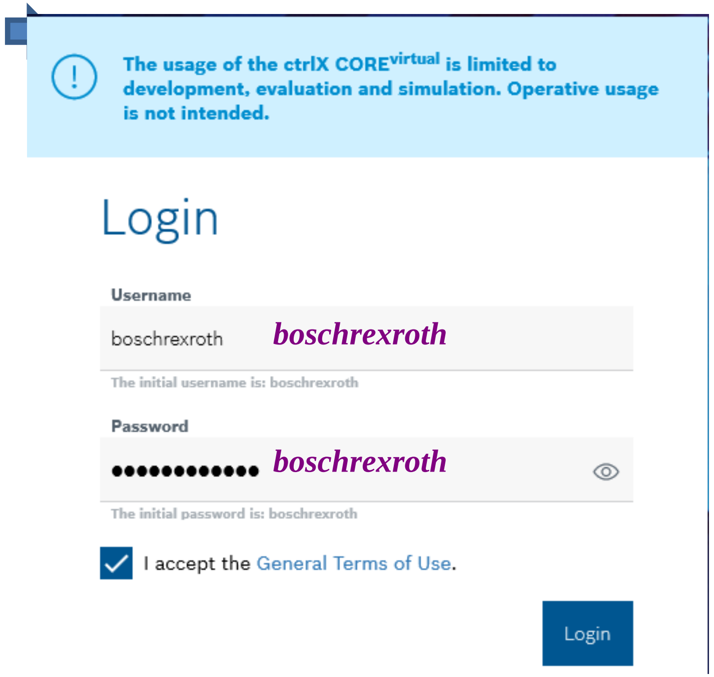
The next step will be to proceed to enter the configuration from the Software.
The first screen that will open will show us the equipment that we have generated, virtual or real systems if we are connected to any of the equipment, in the image a ctrlX-Core appears in "State" Online, which is the one we are using for tests



If we connect, clicking on the "IP addresses" will open another menu in which we will be asked for the Password for the connection, which will open the connection from the part of the browser that we have preset in the system.

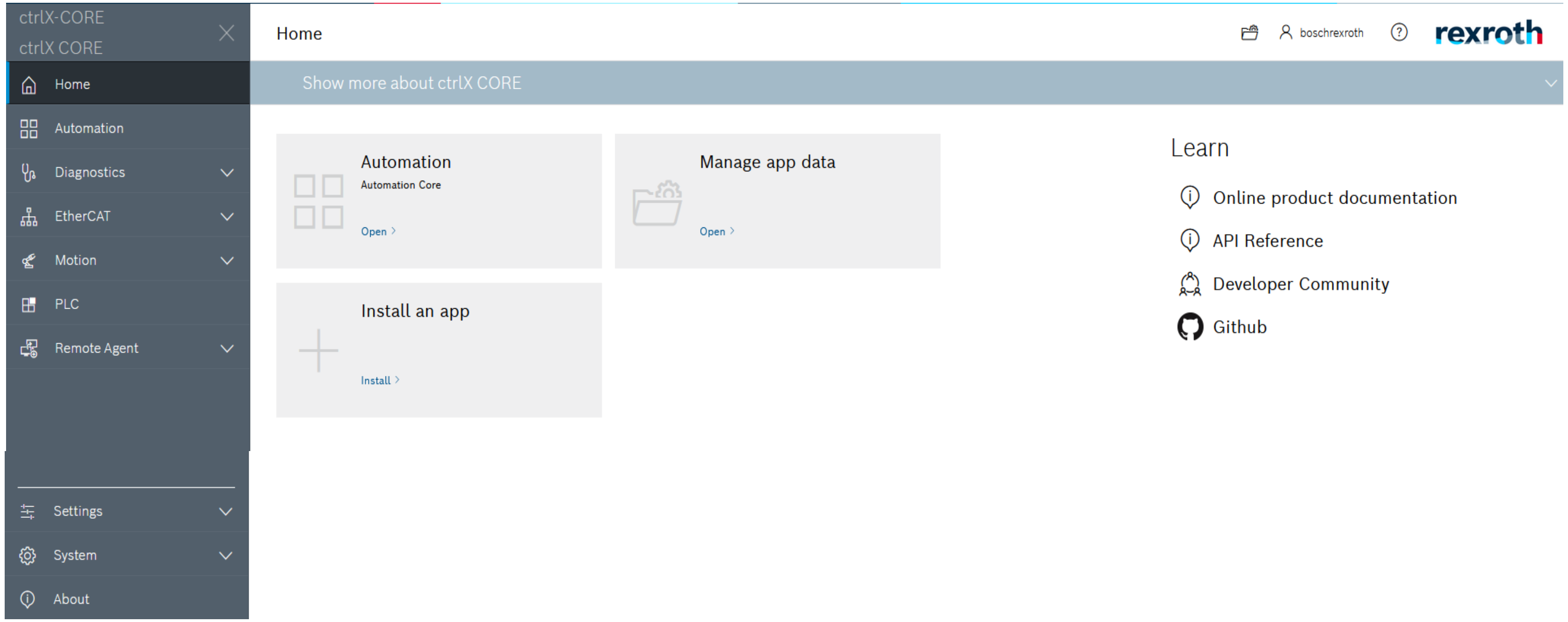
ctrlX- *Configuration of the ctrlX I/O – “UserName” and “Password” by default*

By default the first connection can be made using the default “UserName” and “Password”



The screen may vary depending on the software version used

After this first connection, the system will ask us to modify the "Password", which must include a minimum of 12 digits, including numbers and capital letters, the "Username" does not need to be changed.

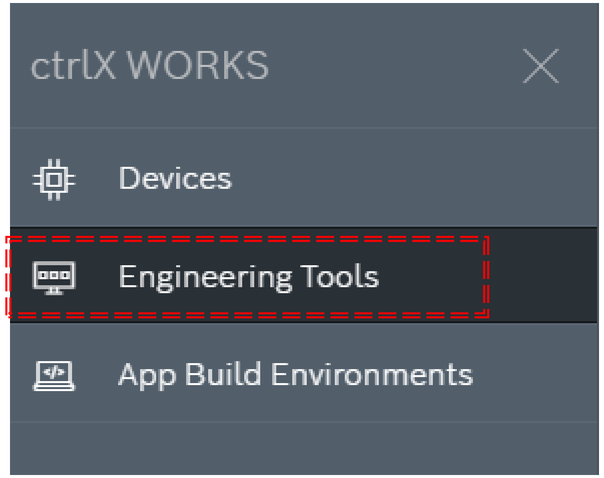


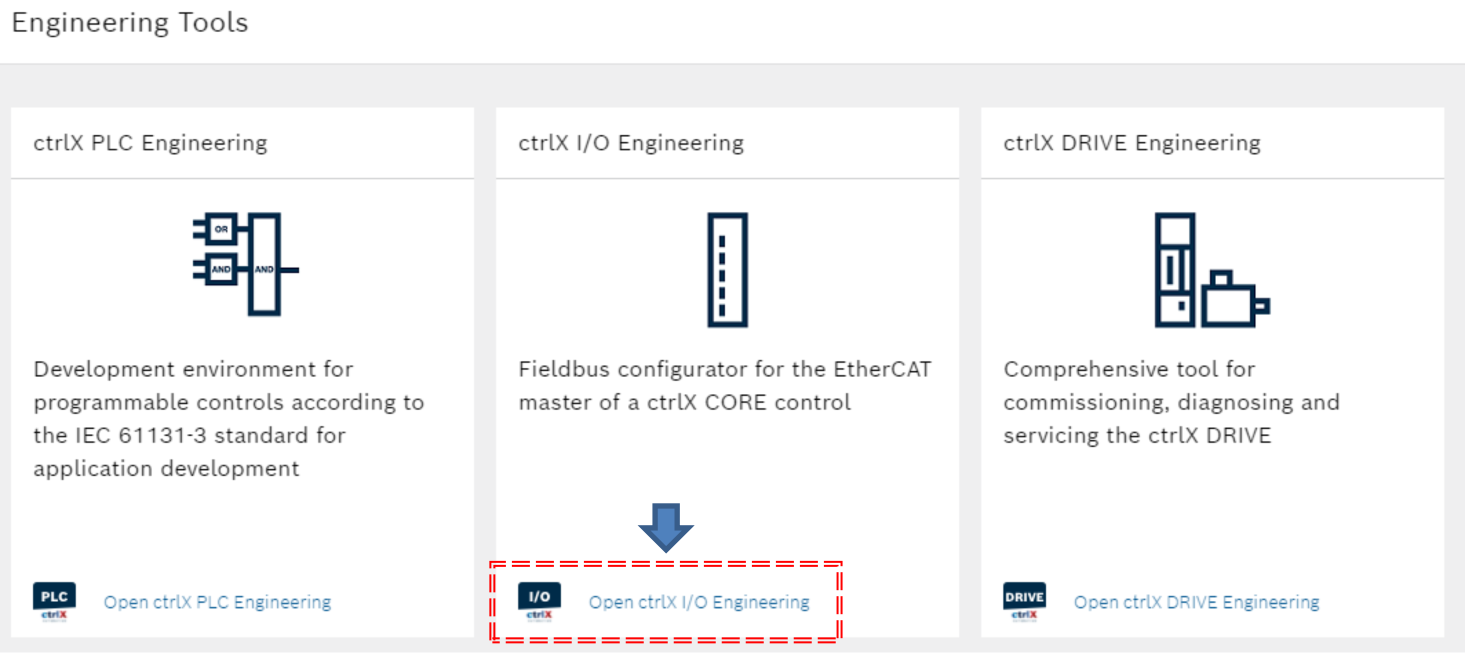
This displays the generic menu, from which we can access all the available elements.

rexroth
A Bosch Company

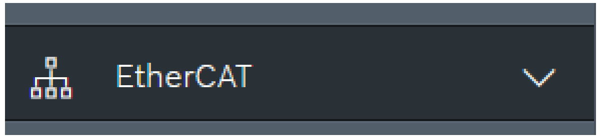
ctrlX - *Configuration of the ctrlX I/O – Access to “Open ctrlX I/O Enginnering”*

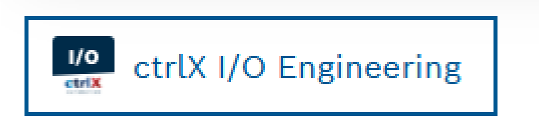
We go back one step and from the Software we will choose the “Engineering Tools” option and within this we will activate the “Open ctrlX I/O Engineering” option





We can also access from the browser, accessing the “EtherCAT” menu

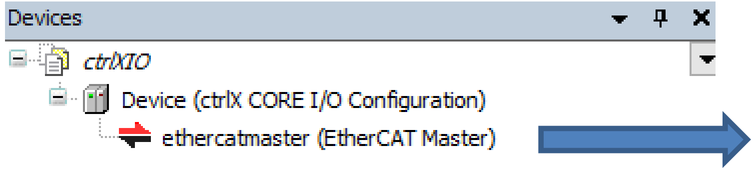


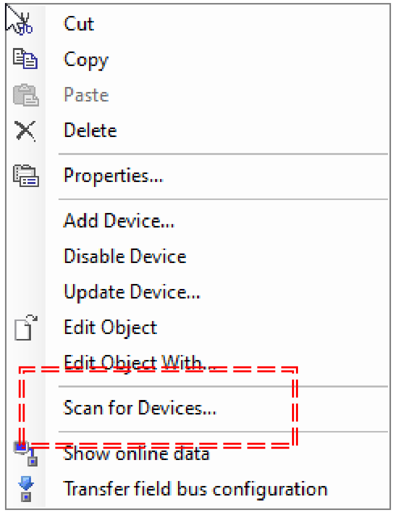


rexroth
A Bosch Company

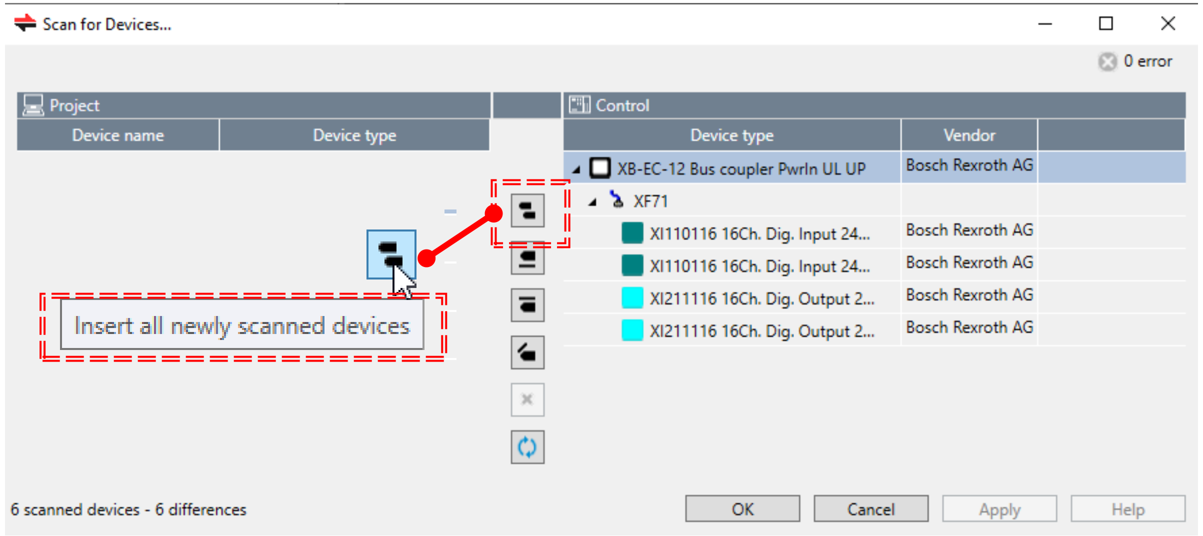
Once in the system we must, if not, incorporate the "EtherCat Master"

Then we must scan the equipment

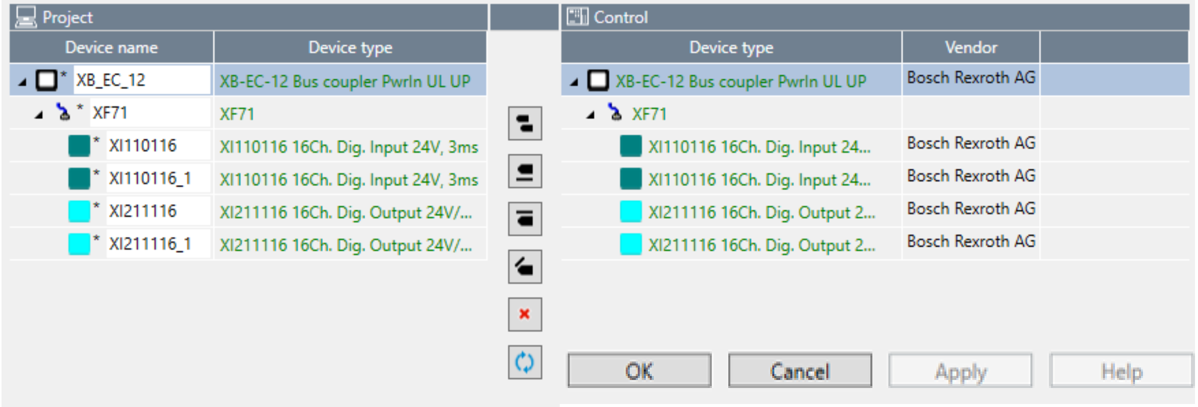




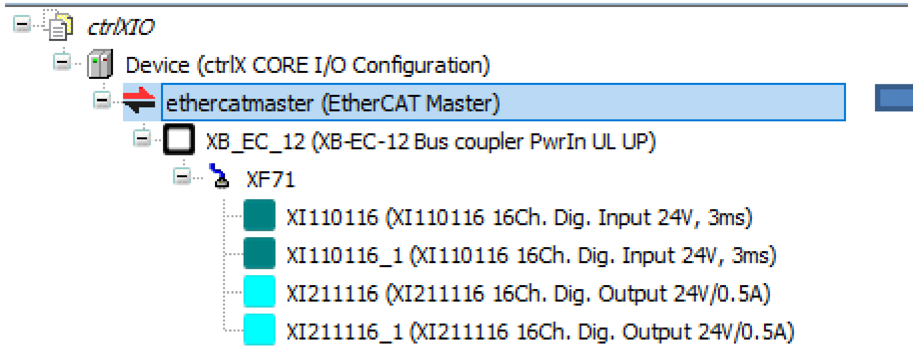
The "scanned" equipment appears on the right and we proceed to insert the new elements



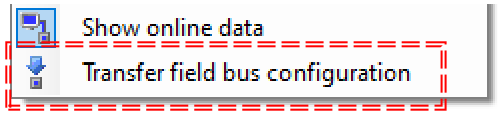
After this, the found equipment also appears on the left and we can proceed to "Apply" or directly "OK"

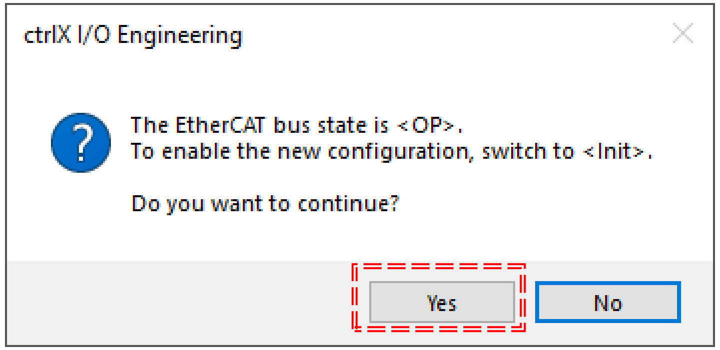


Now all the new modules appear under the "EtherCat Master"

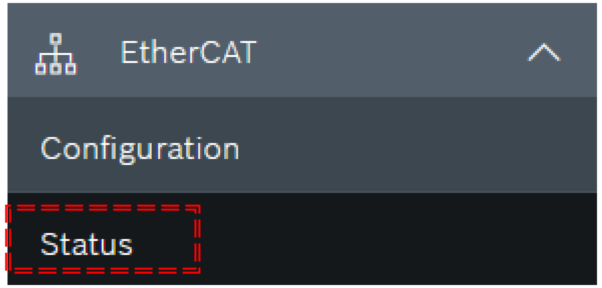


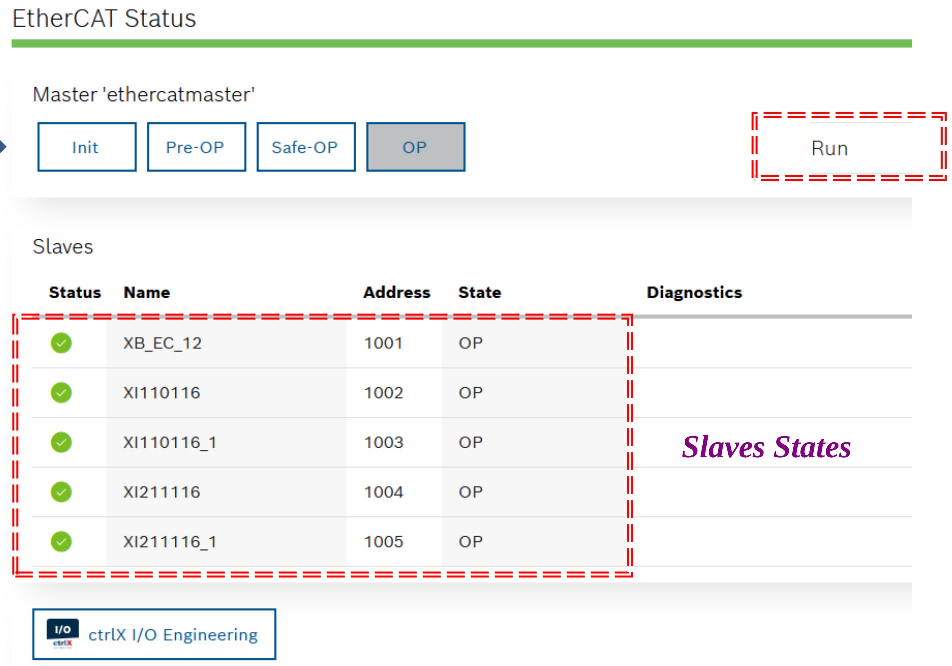
Then we proceed to make the transfer, with the new configuration





From the browser we can see how the EtherCat network communication is. We can even modify its status using the operating mode buttons.



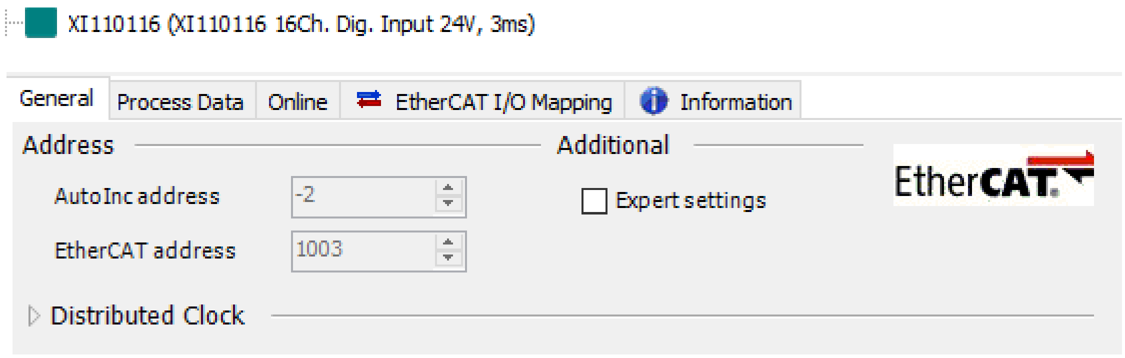


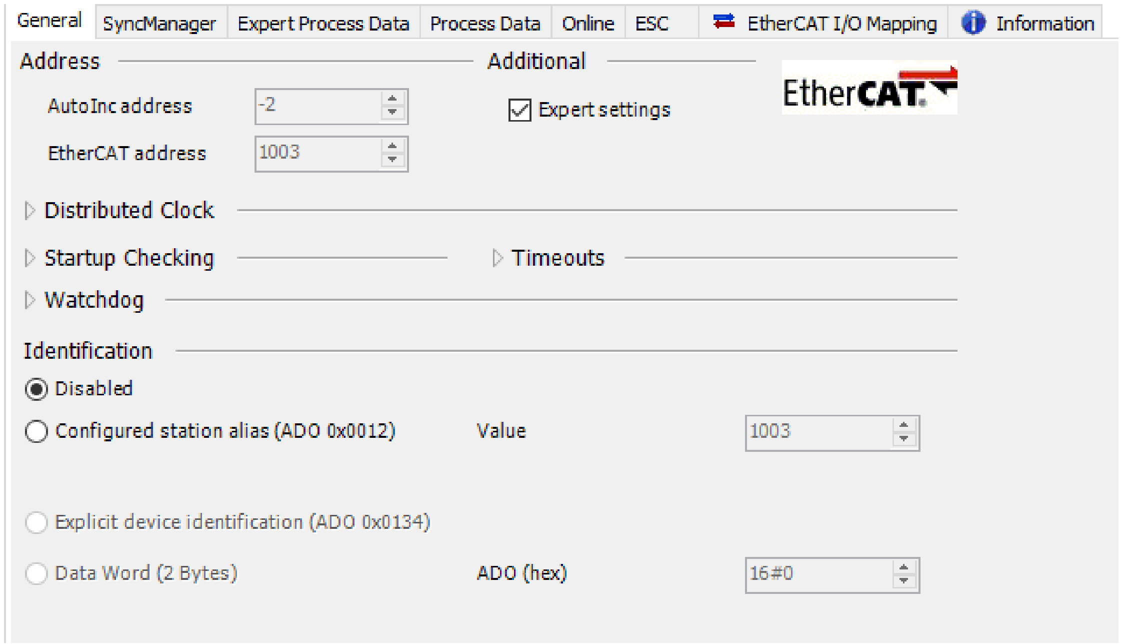
EtherCat Bus in Run

Slaves States

rexroth
A Bosch Company

All modules have associated menus, in general, activating the "Expert Settings" more menus are displayed. In the example we see the first input module.





If we enter the "EtherCat I/O Mapping" menu we can see the assigned I/O areas

From my point of view, the I/O areas should be in the same range to avoid confusion, regardless of the elements of each I/O unit. We will modify the initial areas of each group, to respect the order both in the Inputs and in the Outputs.

The modification will be made from this section, modifying the first of the bytes, assuming the system, the modification of the rest automatically.

EtherCAT I/O Mapping

Input Mapping

Variable	Mapping	Channel	Address	Type	Unit	Description
		Value	%IX10.0	BIT		Value
		Value	%IX10.1	BIT		Value
		Value	%IX10.2	BIT		Value
		Value	%IX10.3	BIT		Value
		Value	%IX10.4	BIT		Value
		Value	%IX10.5	BIT		Value
		Value	%IX10.6	BIT		Value
		Value	%IX10.7	BIT		Value
		Value	%IX11.0	BIT		Value
		Value	%IX11.1	BIT		Value
		Value	%IX11.2	BIT		Value
		Value	%IX11.3	BIT		Value
		Value	%IX11.4	BIT		Value
		Value	%IX11.5	BIT		Value
		Value	%IX11.6	BIT		Value
		Value	%IX11.7	BIT		Value

Variable	Mapping	Channel	Address	Type	Unit	Description
		Value	%IX12.0	BIT		Value
		Value	%IX12.1	BIT		Value
		Value	%IX12.2	BIT		Value
		Value	%IX12.3	BIT		Value
		Value	%IX12.4	BIT		Value
		Value	%IX12.5	BIT		Value
		Value	%IX12.6	BIT		Value
		Value	%IX12.7	BIT		Value
		Value	%IX13.0	BIT		Value
		Value	%IX13.1	BIT		Value
		Value	%IX13.2	BIT		Value
		Value	%IX13.3	BIT		Value
		Value	%IX13.4	BIT		Value
		Value	%IX13.5	BIT		Value
		Value	%IX13.6	BIT		Value
		Value	%IX13.7	BIT		Value

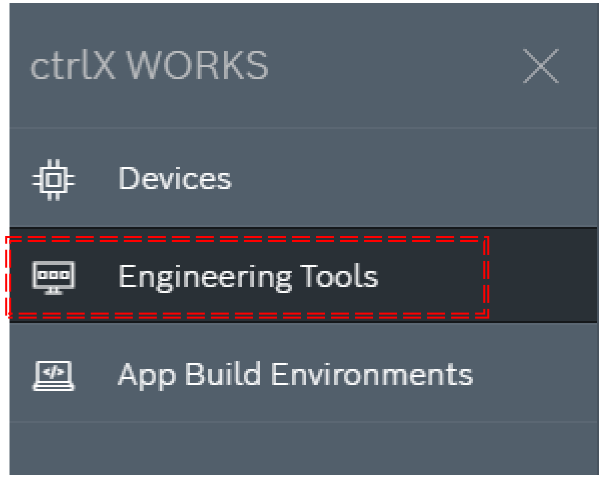
Output Assignment

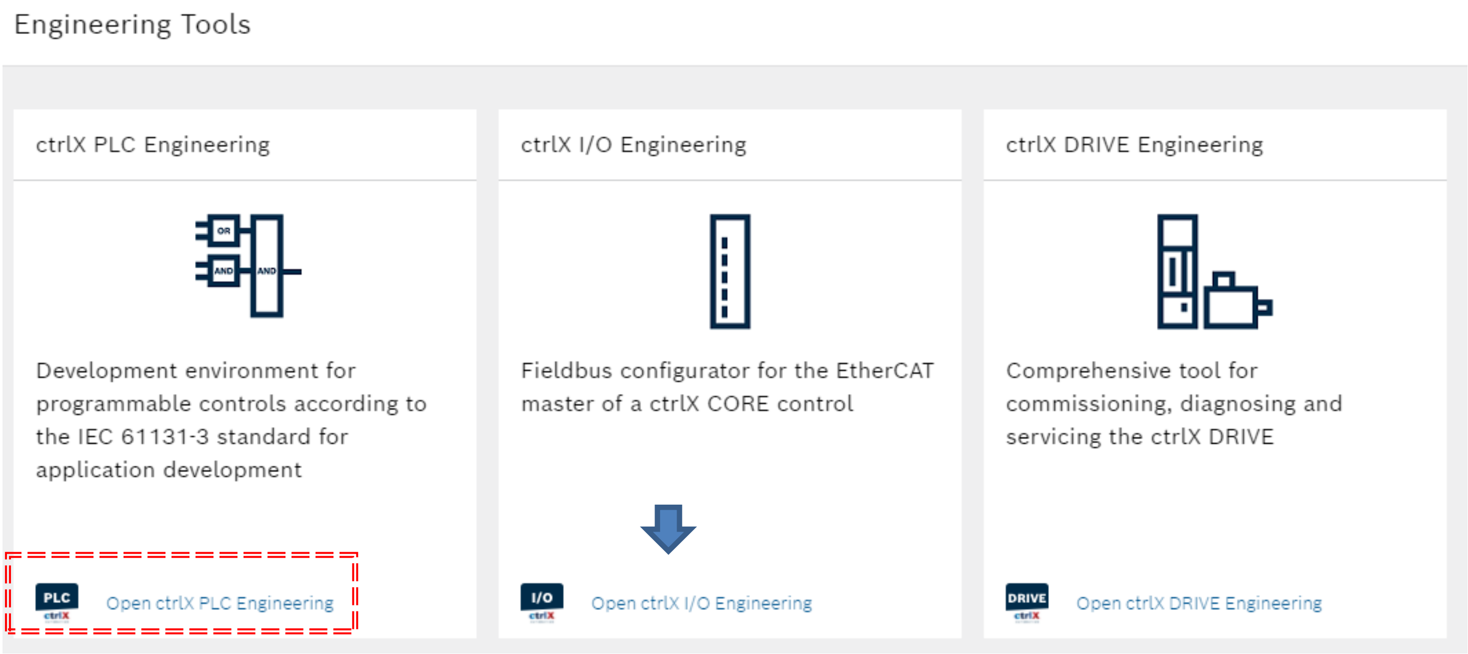
Variable	Mapping	Channel	Address	Type	Unit	Description
		Value	%QX10.0	BIT		Value
		Value	%QX10.1	BIT		Value
		Value	%QX10.2	BIT		Value
		Value	%QX10.3	BIT		Value
		Value	%QX10.4	BIT		Value
		Value	%QX10.5	BIT		Value
		Value	%QX10.6	BIT		Value
		Value	%QX10.7	BIT		Value
		Value	%QX11.0	BIT		Value
		Value	%QX11.1	BIT		Value
		Value	%QX11.2	BIT		Value
		Value	%QX11.3	BIT		Value
		Value	%QX11.4	BIT		Value
		Value	%QX11.5	BIT		Value
		Value	%QX11.6	BIT		Value
		Value	%QX11.7	BIT		Value

Variable	Mapping	Channel	Address	Type	Unit	Description
		Value	%QX12.0	BIT		Value
		Value	%QX12.1	BIT		Value
		Value	%QX12.2	BIT		Value
		Value	%QX12.3	BIT		Value
		Value	%QX12.4	BIT		Value
		Value	%QX12.5	BIT		Value
		Value	%QX12.6	BIT		Value
		Value	%QX12.7	BIT		Value
		Value	%QX13.0	BIT		Value
		Value	%QX13.1	BIT		Value
		Value	%QX13.2	BIT		Value
		Value	%QX13.3	BIT		Value
		Value	%QX13.4	BIT		Value
		Value	%QX13.5	BIT		Value
		Value	%QX13.6	BIT		Value
		Value	%QX13.7	BIT		Value

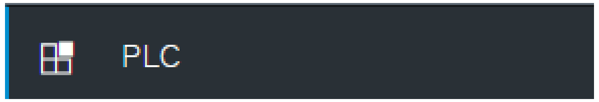
Online Connection ctrlX

Assuming that we already have the configuration of the modules prepared, we are going to incorporate this in the part of the PLC program (Codeys)





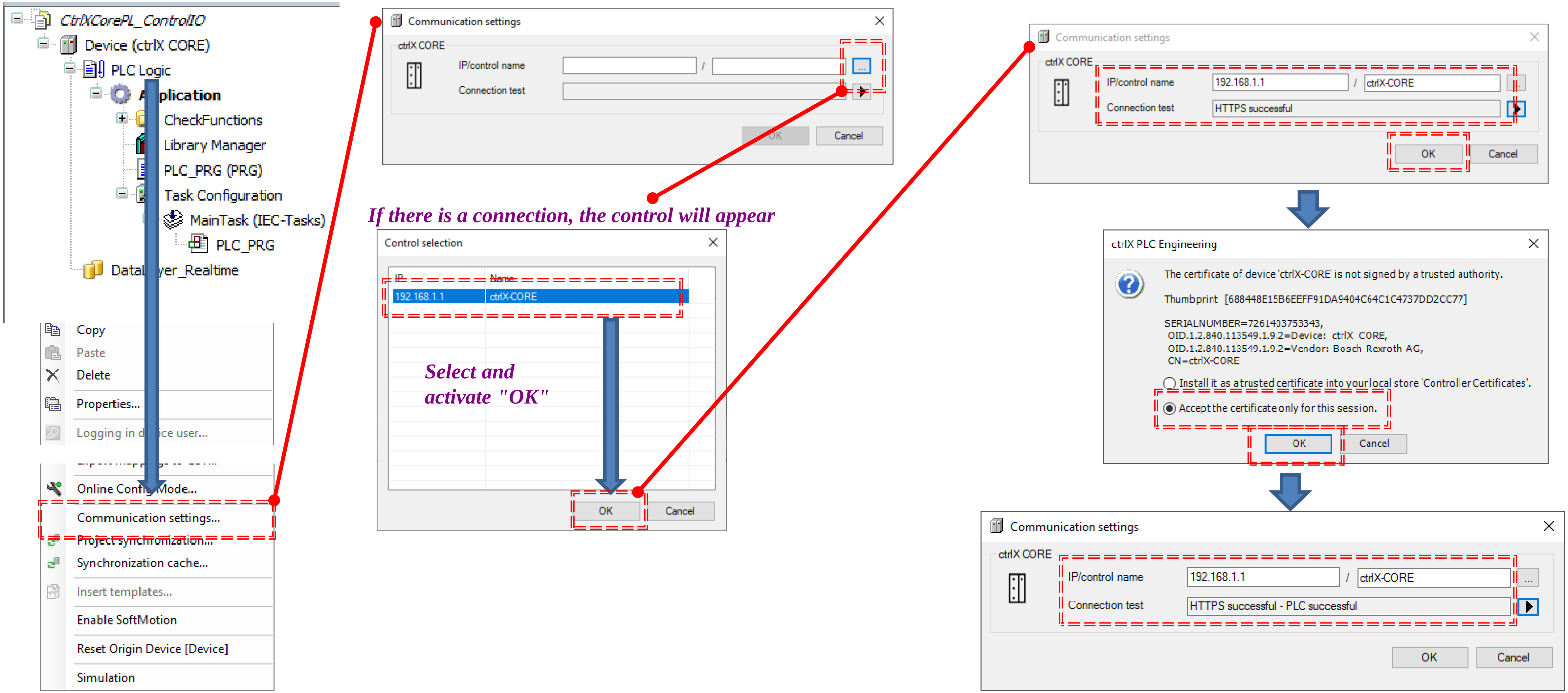
We can also access from the browser, accessing the "PLC" menu



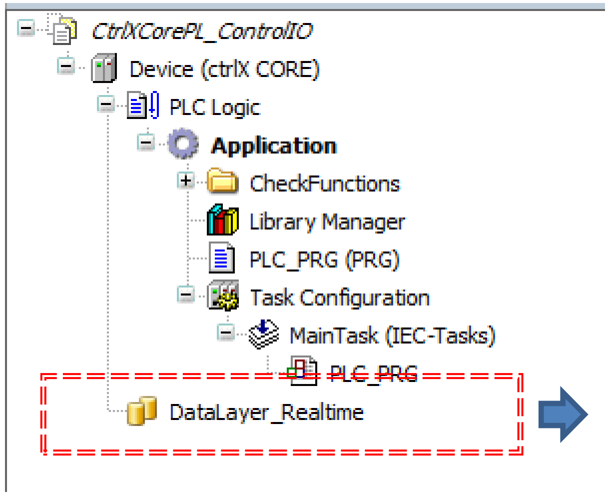
rexroth
A Bosch Company

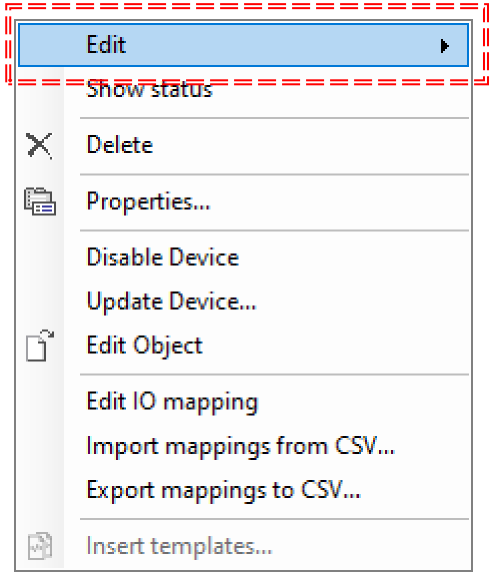
ctrlX - Configuration of the ctrlX I/O – Online Connection

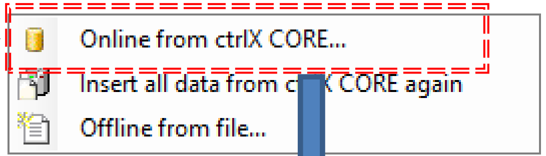
Once the PLC application is open, we will proceed to verify the connection with the equipment.



With the established communication we will go on to incorporate the I/O unit to the "DataLayer_Realtime"

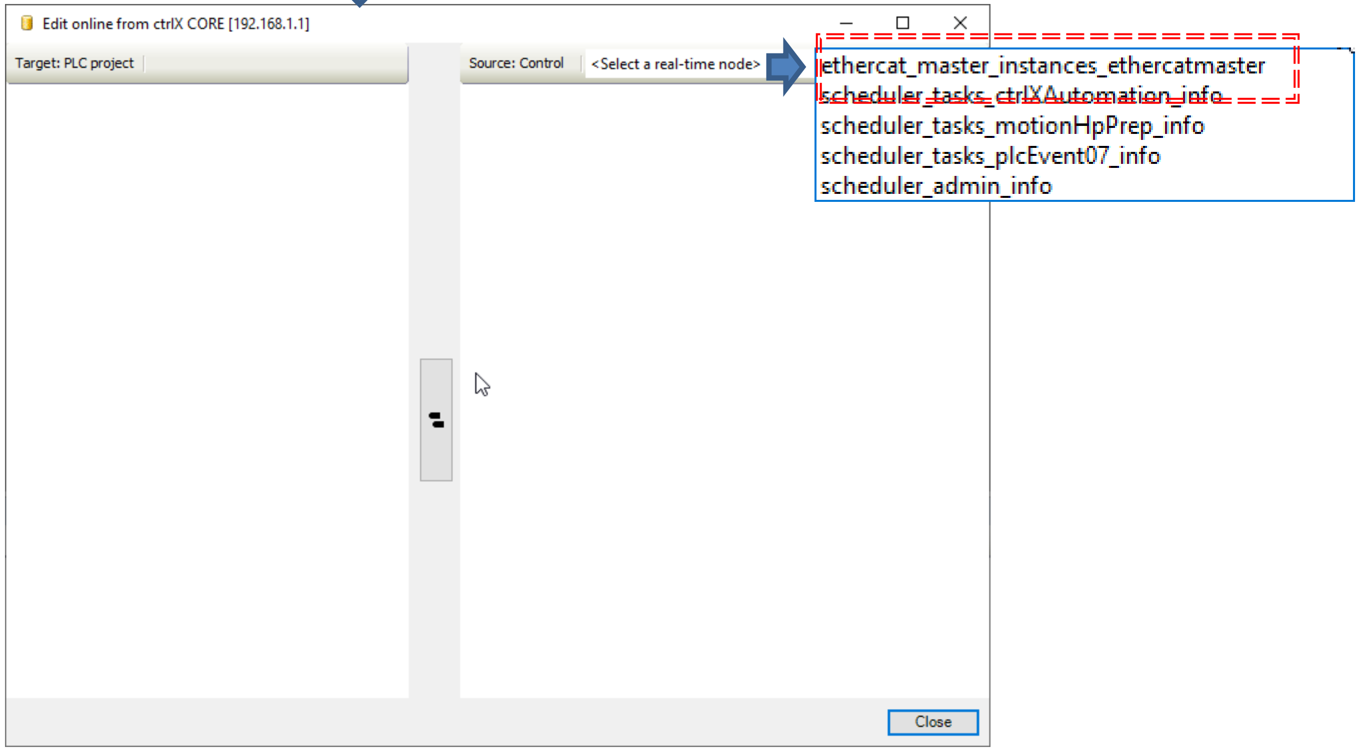






The part of I/O configured previously should be "imported" to the "Data Layer Realtime"

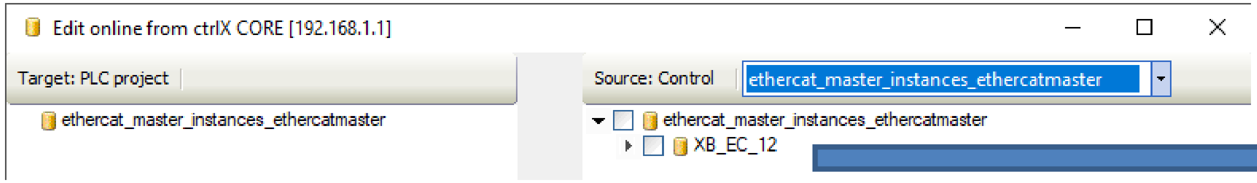
We select the "RealTime Node"



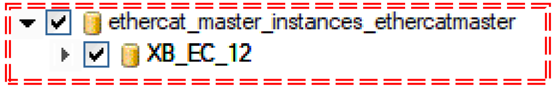
rexroth
A Bosch Company

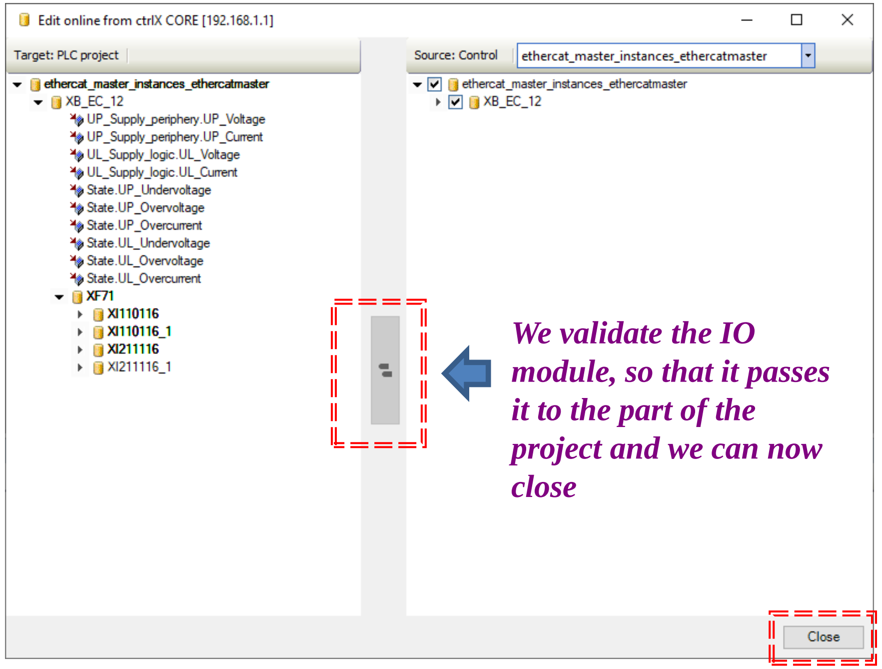
ctrlX - *Configuration of the ctrlX I/O – Online Connection*

When inserting the "Real Time Node" the module that we had previously sent from the I/O configuration already appears



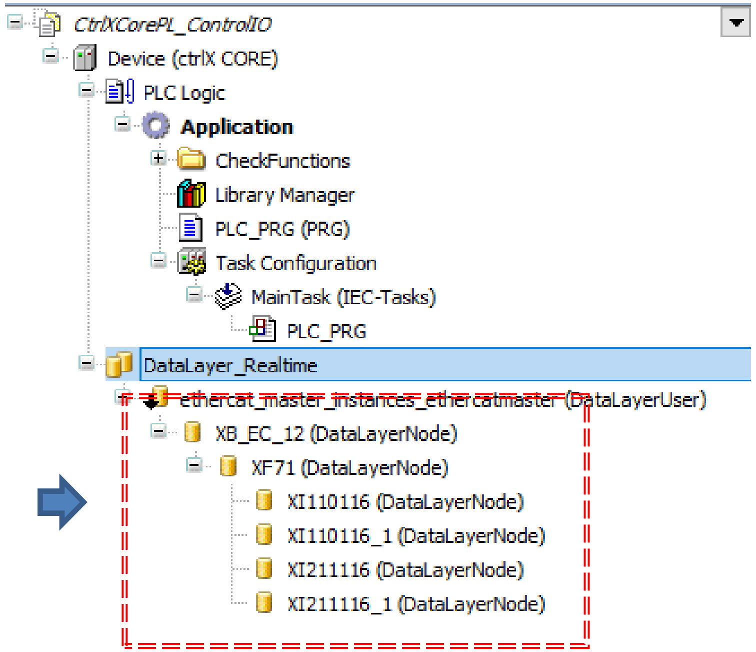
We select the IO header and the whole set is activated





We validate the IO module, so that it passes it to the part of the project and we can now close

Bus Coupler and IO modules incorporated into the "Data Layer Real Time"



rexroth
A Bosch Company

Before sending the configuration, I recommend reviewing the I/O areas, since many times an unwanted "scrolling" of areas occurs at the moment in which the modules "appear" in the "DataLayer_RealTime"

DataLayerNode I/O Mapping

Find

Configuring the inputs in the "Data Layer Real Time"

Variable	Mapping	Channel	Address	Type
		Channel_1.Value	%IX8.6	BIT
		Channel_2.Value	%IX8.7	BIT
		Channel_3.Value	%IX9.0	BIT
		Channel_4.Value	%IX9.1	BIT
		Channel_5.Value	%IX9.2	BIT
		Channel_6.Value	%IX9.3	BIT
		Channel_7.Value	%IX9.4	BIT
		Channel_8.Value	%IX9.5	BIT
		Channel_9.Value	%IX9.6	BIT
		Channel_10.Value	%IX9.7	BIT
		Channel_11.Value	%IX10.0	BIT
		Channel_12.Value	%IX10.1	BIT
		Channel_13.Value	%IX10.2	BIT
		Channel_14.Value	%IX10.3	BIT
		Channel_15.Value	%IX10.4	BIT
		Channel_16.Value	%IX10.5	BIT

Configuration of the previously defined inpu the ctrlX I/O Engineering

ctrlX I/O Engineering

Variable	Mapping	Channel	Address	Type	Unit	Description
		Value	%IX10.0	BIT		Value
		Value	%IX10.1	BIT		Value
		Value	%IX10.2	BIT		Value
		Value	%IX10.3	BIT		Value
		Value	%IX10.4	BIT		Value
		Value	%IX10.5	BIT		Value
		Value	%IX10.6	BIT		Value
		Value	%IX10.7	BIT		Value
		Value	%IX11.0	BIT		Value
		Value	%IX11.1	BIT		Value
		Value	%IX11.2	BIT		Value
		Value	%IX11.3	BIT		Value
		Value	%IX11.4	BIT		Value
		Value	%IX11.5	BIT		Value
		Value	%IX11.6	BIT		Value
		Value	%IX11.7	BIT		Value

Be careful with the areas assigned when passing the PLC part since they are modified by the system itself, I recommend verifying them and modifying them again according to what was previously configured

This also happens because the headend, the Coupler bus, has some input areas that are used as information and that end in this case in the %IX8.5 bit. In this case, when recovering the information downloaded from the existing configuration, the system continues from the free areas and therefore adjusts the values

rexroth
A Bosch Company

Here we can see the Inputs area that is using the header. Therefore, this structure must be kept in mind when structuring the areas of the various I/O units that we have in the machine.

Occupation of the Inputs of the headers of the I/O units

XB_EC_12 (DataLayerNode)

DataLayerNode I/O Mapping | DataLayerNode Parameters | Status | Information

Find | Filter Show all | Add FB for IO Channel... | Go to Instance

Variable	Mapping	Channel	Address	Type	Default Value	Current Value
		UP_Supply_periphery.UP_Voltage	%IW0	UINT		24141
		UP_Supply_periphery.UP_Current	%IW2	UINT		91
		UL_Supply_logic.UL_Voltage	%IW4	UINT		24172
		UL_Supply_logic.UL_Current	%IW6	UINT		162
		State.UP_Undervoltage	%IX8.0	BIT		FALSE
		State.UP_Overvoltage	%IX8.1	BIT		FALSE
		State.UP_Overcurrent	%IX8.2	BIT		FALSE
		State.UL_Undervoltage	%IX8.3	BIT		FALSE
		State.UL_Overvoltage	%IX8.4	BIT		FALSE
		State.UL_Overcurrent	%IX8.5	BIT		FALSE

Also in the "DataLayerNode I/O Mapping" of each of the modules, at least the "Enabled 2" option should be activated so that the I/O are updated automatically

DataLayerNode I/O Mapping

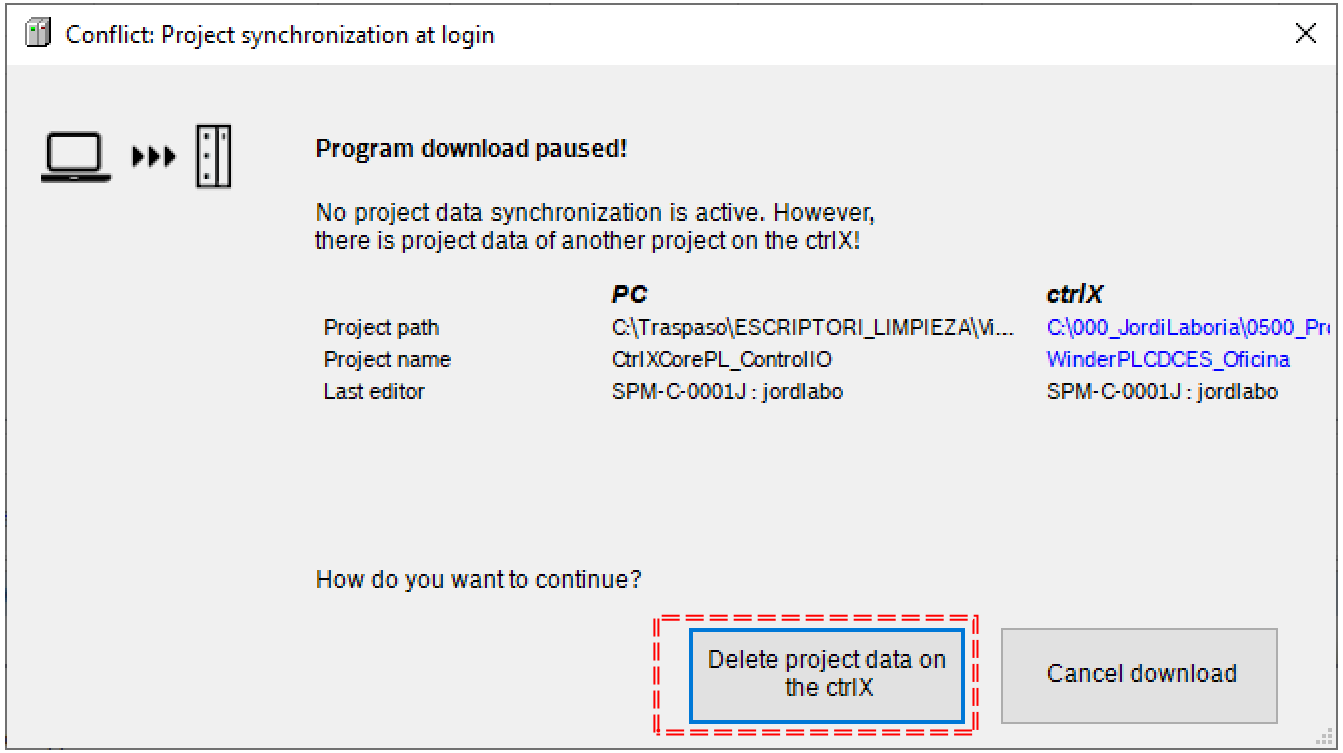
Find

Use parent device setting
Enabled 1 (use bus cycle task if not used in any task)
Enabled 2 (always in bus cycle task)

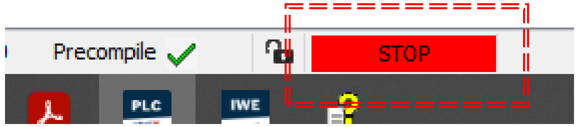
Codesys, if you do not have any of the "Enabled" options activated, does not refresh the I/O areas, unless they are used in some part of the program, which obviously leads to errors if this issue is unknown, since it will surely we will think that the input or output cards are faulty

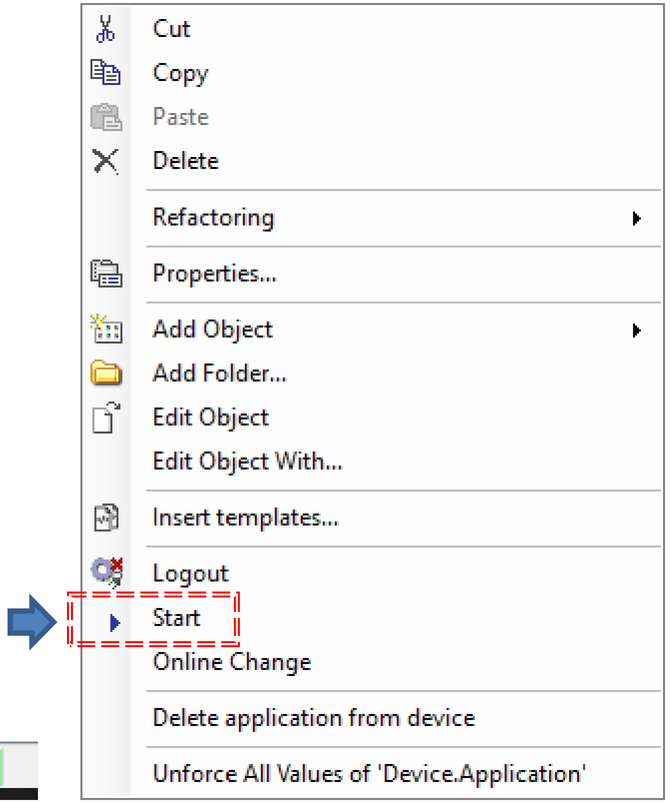
Now we can connect with the ctrlX to transfer the program or at least the configuration created up to that moment

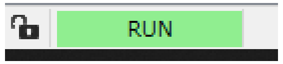
If the program we are sending is another program, ctrlX already had one loaded, this screen will appear, we apply "Delete Project Data On The CtrlX" and the program will be loaded on the computer



By default it will be in the "Stop" state, so we will proceed to start it up



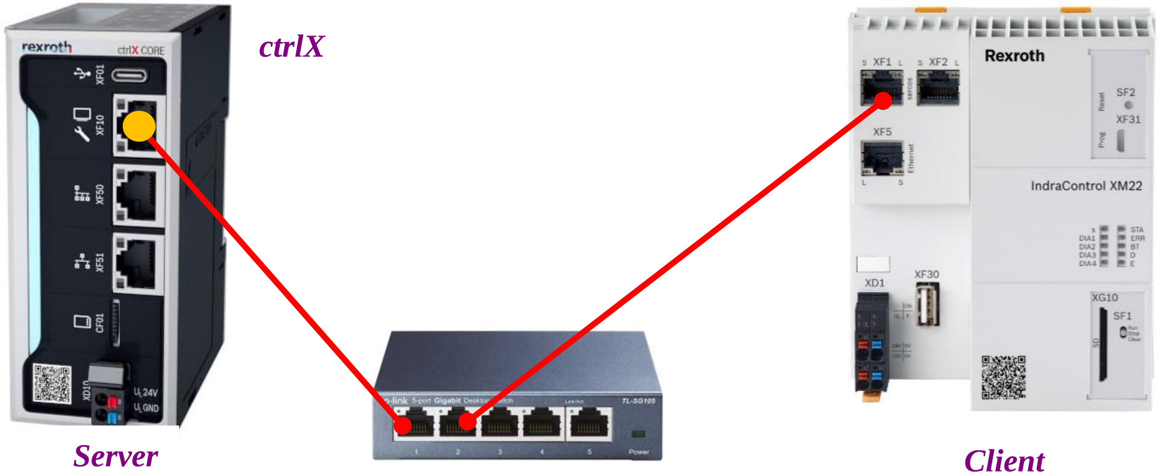




Annex Modbus communication

Modbus communication between the ctrlX and an XM is detailed in another manual. However, now the objective is to pass the state of the inputs to the XM and activate the outputs from the XM. In itself, the program will not differ much from what has already been mentioned in the previous manual, however, some aspects must be taken into account.

The system will use the Server - Client model



The Modbus operating modules that we are going to use allow the passage of up to four types of areas.

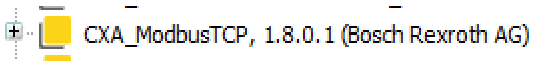
I / O Area
65536 Bit Coil
65536 Discrete Input
65536 Word Input Register
65536 Word Holding Register

Although this is relatively true, the reality is that the assignment of areas in both teams differs slightly in the part called Coil.
Therefore, care must be taken when transferring data.

I / O Area	ctrlX
65536 Bit Coil	Byte
65536 Discrete Input	Byte
65536 Word Input Register	Word
65536 Word Holding Register	Word

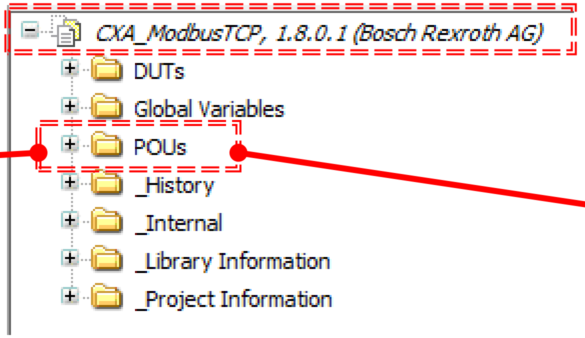
I / O Area	XM
65536 Bit Coil	Bool
65536 Discrete Input	Bool
65536 Word Input Register	Word
65536 Word Holding Register	Word

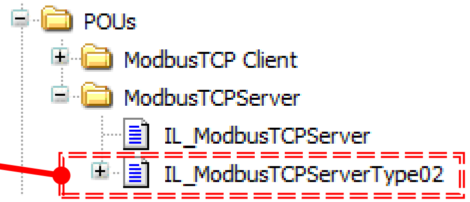
For this we are going to use the following libraries that must be installed on both computers



ctrlX

SERVER

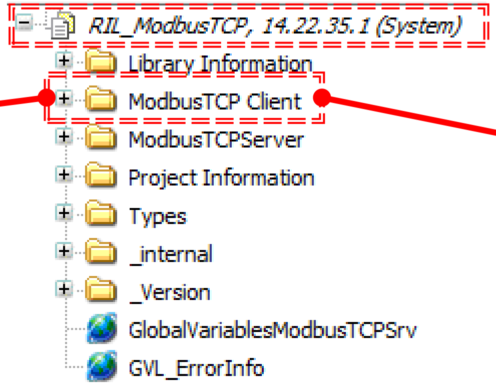


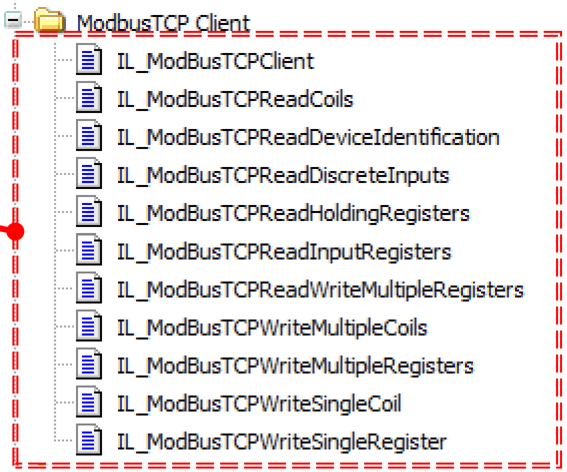


n the case of the example, we are going to use ctrlX as a server and to send the data, we will use the "…Type2" module, which allows us to send (and receive in two areas) the four commented areas.

XM

CLIENT





The XM as a client will use several of the modules since we will collect the information sent based on the group of data received.

The data sent or received, as we have said, can be of four different types:

COIL DATA:

Coil Data	
Opciones :	Lectura / Escritura
Estructura :	Byte (ctrlX) / Bool (XM)
Definir tamaño del array con :	SIZEOF()
Representación :	1 Array / 8 bits
Opcional :	Uso de estructuras

ctrlX
```
arCoilData: ARRAY [0..7] OF BYTE;
```
XM
```
arCoilData: ARRAY [0..7] OF BOOL;
```

INPUT DATA:

Input Data	
Opciones :	Lectura
Estructura :	Byte (ctrlX) / Bool (XM)
Definir tamaño del array con :	SIZEOF()
Representación :	1 Array / 8 bits
Opcional :	Uso de estructuras

ctrlX
```
arInputData: ARRAY [0..7] OF BYTE;
```
XM
```
arInputData: ARRAY [0..7] OF BOOL;
```

HOLDING REGISTER DATA:

HoldingRegisterData	
Opciones :	Lectura / Escritura
Estructura :	Word
Definir tamaño del array con :	SIZEOF()
Representación :	Word (16 bits)
Opcional :	Uso de estructuras

```
stHoldingRegisterData:dutRegisterData;
```

The HoldingRegisterData, since they can be written, we will use them to activate the outputs from the XM

REGISTER DATA:

These types of data allow the sending of structures and in them we can have different types of elements, Word, Int, Real, etc.

```
TYPE dutRegisterData :
STRUCT
  arWord:   ARRAY[0..999]OF WORD;     // 0 - 999
  arInt:    ARRAY[0..999]OF INT;      // 1000 - 1999
  arUint:   ARRAY[0..999]OF UINT;     // 2000 - 2999
  arDint:   ARRAY[0..499]OF DINT;     // 3000 - 3999
  arUdint:  ARRAY[0..499]OF UDINT;    // 4000 - 4999
  arReal:   ARRAY[0..499]OF REAL;     // 5000 - 5999
  arString: ARRAY[0..99]OF STRING(19);// 6000 - 6999

END_STRUCT
END_TYPE
```

RegisterData	
Opciones :	Lectura
Estructura :	Word
Definir tamaño del array con :	SIZEOF()
Representación :	Word (16 bits)
Opcional :	Uso de estructuras

```
stRegisterData:dutRegisterData;
```

The RegisterData as they are read only will be used to capture the image of the inputs of the IO modules

rexroth
A Bosch Company

Example of data passing at the HoldingRegisterData level, which also works for the RegisterData

ctrlX

```
fbModbusComToXM( // IL_ModbusTCPServerType02
    Enable:=bEnable ,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> ,
    ErrorIdent=> ,
    Port:= ,
    CoilData:=ADR(arCoilData) ,
    SizeOfCoilData:=SIZEOF(arCoilData) ,
    InputData:=ADR(arInputData) ,
    SizeOfInputData:=SIZEOF(arInputData) ,
    HoldingRegisterData:=ADR(stHoldingRegisterData) ,
    SizeOfHoldingRegisterData:=SIZEOF(stHoldingRegisterData) ,
    InputRegisterData:=ADR(stRegisterData) ,
    SizeOfInputRegisterData:=SIZEOF(stRegisterData) ,
    Stats=> );
```

Send / Receive HoldingRegisterData Area

XM

fbModbusCLient	IL_ModBusTCPClient
fbModBusReadCoils	IL_ModBusTCPReadCoils
fbModBusReadDiscreteInputs	IL_ModBusTCPReadDiscreteInputs
fbModBusReadHoldingRegisters_Word	IL_ModBusTCPReadHoldingRegisters
fbModBusReadHoldingRegisters_int	IL_ModBusTCPReadHoldingRegisters
fbModBusReadHoldingRegisters_Uint	IL_ModBusTCPReadHoldingRegisters
fbModBusReadHoldingRegisters_Dint	IL_ModBusTCPReadHoldingRegisters
fbModBusReadHoldingRegisters_Udint	IL_ModBusTCPReadHoldingRegisters
fbModBusReadHoldingRegisters_Real	IL_ModBusTCPReadHoldingRegisters
fbModBusReadHoldingRegisters_String	IL_ModBusTCPReadHoldingRegisters
fbModBusWriteMultipleCoils	IL_ModBusTCPWriteMultipleCoils
fbModbusWriteMultipleHoldingRegisters	IL_ModBusTCPWriteMultipleRegisters

The data must be extracted in individual groups, if we want to access the different parts of the generated structure, for this reason the same type of module is used, but each one for a different group.

ctrlX

stHoldingRegisterData	dutRegisterData
arWord	ARRAY [0..999] OF WORD
arInt	ARRAY [0..999] OF INT
arUint	ARRAY [0..999] OF UINT
arDint	ARRAY [0..499] OF DINT
arUdint	ARRAY [0..499] OF UDINT
arReal	ARRAY [0..499] OF REAL
arString	ARRAY [0..99] OF STRING(19)

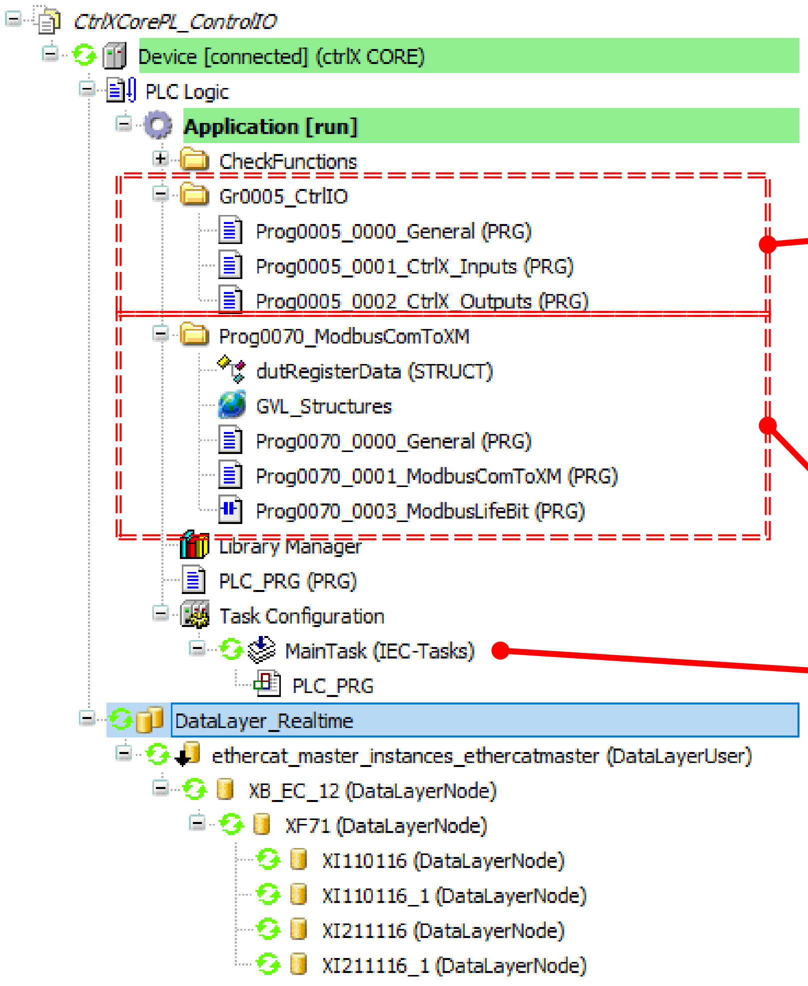
XM

rexroth
A Bosch Company

Program used in the ctrlX

ctrlX - Control program used in the ctrlX

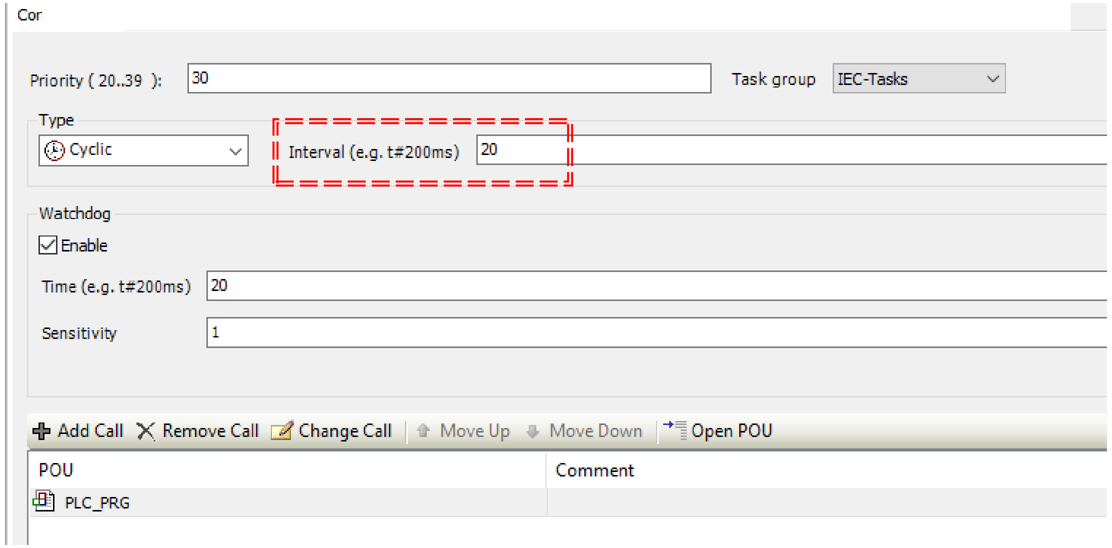
The control program used in the ctrlX is generated as follows



Control of the IO.
In the modules, the data from the Inputs is received and sent to the registers used by the Modbus and the data from the Modbus is also received and the outputs are written.

General control of the Modbus communications server.

The interval time has been modified since it had a value of 200ms.



rexroth
A Bosch Company

The way to "send" the input data and "receive" the outputs from the Modbus can be done as complicated as we want, but in my opinion the easiest thing is to do it as it appears in the image

Prog0005_0001_CtrlX_Inputs (PRG)

Let's not forget that at the input level the headers are using almost 8 bytes of inputs.

```
// Assignment of the Input Areas to the Modbus Registers sent to the XM
// Warning with the input areas of the XB_EC_12 headers as they have some input values
// The step of the inputs must be taken into account depending on the module used, if it is 8 inputs or 16 inputs
stRegisterData.arWord[0]:=%IW0;
stRegisterData.arWord[1]:=%IW2;
stRegisterData.arWord[2]:=%IW4;
stRegisterData.arWord[3]:=%IW6;
stRegisterData.arWord[4]:=%IW8;
stRegisterData.arWord[5]:=%IW10;// Inputs Module 1
stRegisterData.arWord[6]:=%IW12;// Inputs Module 2
stRegisterData.arWord[7]:=%IW14;
stRegisterData.arWord[8]:=%IW16;
stRegisterData.arWord[9]:=%IW18;
stRegisterData.arWord[10]:=%IW20;
stRegisterData.arWord[11]:=%IW22;
// Registers up to 999 (0 to 999) can be added
```

Let us remember that the input modules, in the example, were addressed on the input areas Bytes 10/11 and Bytes 12/13, this presupposes that if we want to match these values to a "known" Modbus Register, we can use the logic , just by dividing the initial value of that area by 2, which in the first case will give us a 5 and in the second case a 6. The assignment is relatively easy.

Prog0005_0002_CtrlX_Outputs (PRG)

```
// Assignment of the Outputs Areas to the Modbus Holding Registers Received to the XM
// The step of the Outputs must be taken into account depending on the module used, if it is 8 Outputs or 16 Outputs
%QW0:=stHoldingRegisterData.arWord[0];
%QW2:=stHoldingRegisterData.arWord[1];
%QW4:=stHoldingRegisterData.arWord[2];
%QW6:=stHoldingRegisterData.arWord[3];
%QW8:=stHoldingRegisterData.arWord[4];
%QW10:=stHoldingRegisterData.arWord[5]; // Module 1
%QW12:=stHoldingRegisterData.arWord[6]; // Module 2
%QW14:=stHoldingRegisterData.arWord[7];
%QW16:=stHoldingRegisterData.arWord[8];
%QW18:=stHoldingRegisterData.arWord[8];
%QW20:=stHoldingRegisterData.arWord[10];
%QW22:=stHoldingRegisterData.arWord[11];
```

The same case indicated in the inputs serves us for the assignment of the outputs.

The life bit control should be placed on the outputs to prevent them from maintaining the "ON" state and not generating operating problems.

rexroth
A Bosch Company

ctrlX - Control program used in the ctrlX

The relationship of the bytes with the words of the Register is visible in this image

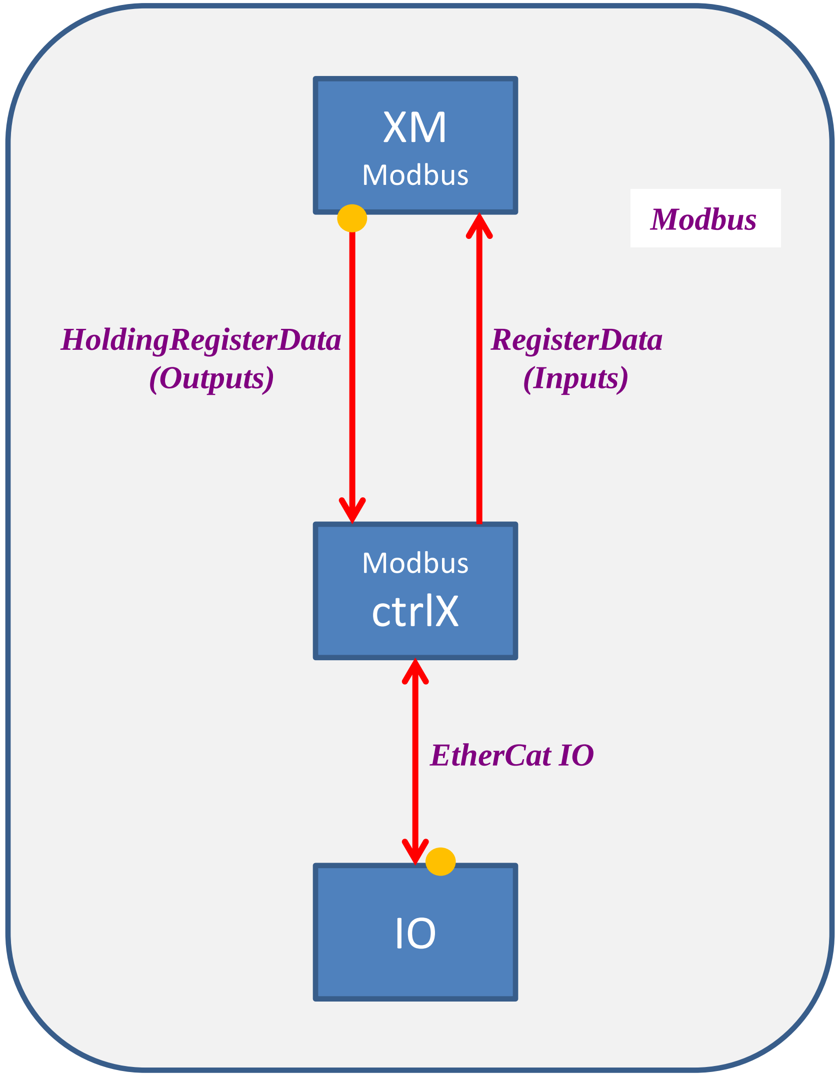
Registers	Byte	Byte
IW0	0	1
IW1	2	3
IW2	4	5
IW3	6	7
IW4	8	9
IW5	10	11
IW6	12	13
IW7	14	15
IW8	16	17
IW9	18	19
IW10	20	21
IW11	22	23
IW12	24	25
IW13	26	27
IW14	28	29
IW15	30	31

HOLDING REGISTER DATA:

HoldingRegisterData	
Opciones :	Lectura / Escritura
Estructura :	Word
Definir tamaño del array con :	SIZEOF()
Representación :	Word (16 bits)
Opcional :	Uso de estructuras

REGISTER DATA:

RegisterData	
Opciones :	Lectura
Estructura :	Word
Definir tamaño del array con :	SIZEOF()
Representación :	Word (16 bits)
Opcional :	Uso de estructuras



rexroth
A Bosch Company

The ctrlX is used as a system server, leaving the program structure as follows

- Prog0070_ModbusComToXM
  - dutRegisterData (STRUCT)
  - GVL_Structures
  - Prog0070_0000_General (PRG)
  - Prog0070_0001_ModbusComToXM (PRG)
  - Prog0070_0003_ModbusLifeBit (PRG)

```
TYPE dutRegisterData :
STRUCT
    /// 0 - 999
    arWord: ARRAY[0..999] OF WORD;
    /// 1000 - 1999
    arInt: ARRAY[0..999] OF INT;
    /// 2000 - 2999
    arUint: ARRAY[0..999] OF UINT;
    /// 3000 - 3999
    arDint: ARRAY[0..499] OF DINT;
    /// 4000 - 4999
    arUdint: ARRAY[0..499] OF UDINT;
    /// 5000 - 5999
    arReal: ARRAY[0..499] OF REAL;
    /// 6000 - 6999
    arString: ARRAY[0..99] OF STRING(19);
END_STRUCT
END_TYPE
```

Array structures that can be used for system control.

The arrays have a maximum value defined, which cannot be sent completely, but in a fragmented way. Although this will be subject to the number of variables that we have to use

Assignment of the two main communication areas that we are going to use for data passing

```
//{attribute 'qualified_only'}
VAR_GLOBAL
    stRegisterData          : dutRegisterData;  // Send To XM (Inputs Image)
    stHoldingRegisterData   : dutRegisterData;  // Receive From XM (To Outputs)
    bLifeBitError           :BOOL;              // Comunication with ctrlX Modbus Error
END_VAR
```

Input values, sent to XM

Values received from the XM and that will activate the outputs

The Prog0070_0000_General POU calls the two Modbus communication control modules on the client side

```
Prog0070_0001_ModbusComToXM();
Prog0070_0003_ModbusLifeBit();
```

- Prog0070_ModbusComToXM
  - dutRegisterData (STRUCT)
  - GVL_Structures
  - Prog0070_0000_General (PRG)
  - Prog0070_0001_ModbusComToXM (PRG)
  - Prog0070_0003_ModbusLifeBit (PRG)

Variables

```
PROGRAM Prog0070_0001_ModbusComToXM
VAR
    fbModbusComToXM: IL_ModbusTCPServerType02;
    arCoilData: ARRAY[0..7] OF BYTE;
    arInputData: ARRAY[0..7] OF BYTE;
    bEnable: BOOL;
    bInOperation: BOOL;
    bError: BOOL;
END_VAR
```

From Prog0070_0001_ModbusComToXM the control module for Modbus communication used as system client is activated

Code

```
fbModbusComToXM( // IL_ModbusTCPServerType02
    Enable:=bEnable ,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> ,
    ErrorIdent=> ,
    Port:= ,
    CoilData:=ADR(arCoilData) ,
    SizeOfCoilData:=SIZEOF(arCoilData) ,
    InputData:=ADR(arInputData) ,
    SizeOfInputData:=SIZEOF(arInputData) ,
    HoldingRegisterData:=ADR(stHoldingRegisterData) ,
    SizeOfHoldingRegisterData:=SIZEOF(stHoldingRegisterData) ,
    InputRegisterData:=ADR(stRegisterData) ,
    SizeOfInputRegisterData:=SIZEOF(stRegisterData) ,
    Stats=> );
```

The module is activated with "Enable" and should indicate that everything is Ok, if it enters "InOperation"

Data of type CoilData

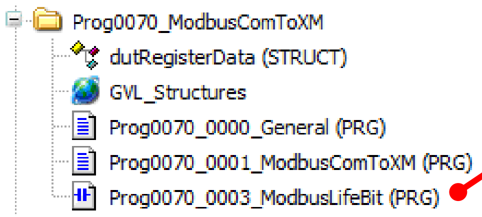
Data of type InputData

Data of type HoldingRegisterData

Data of type RegisterData

The last two areas are the ones that we are going to use for the exchange of I/O signals.

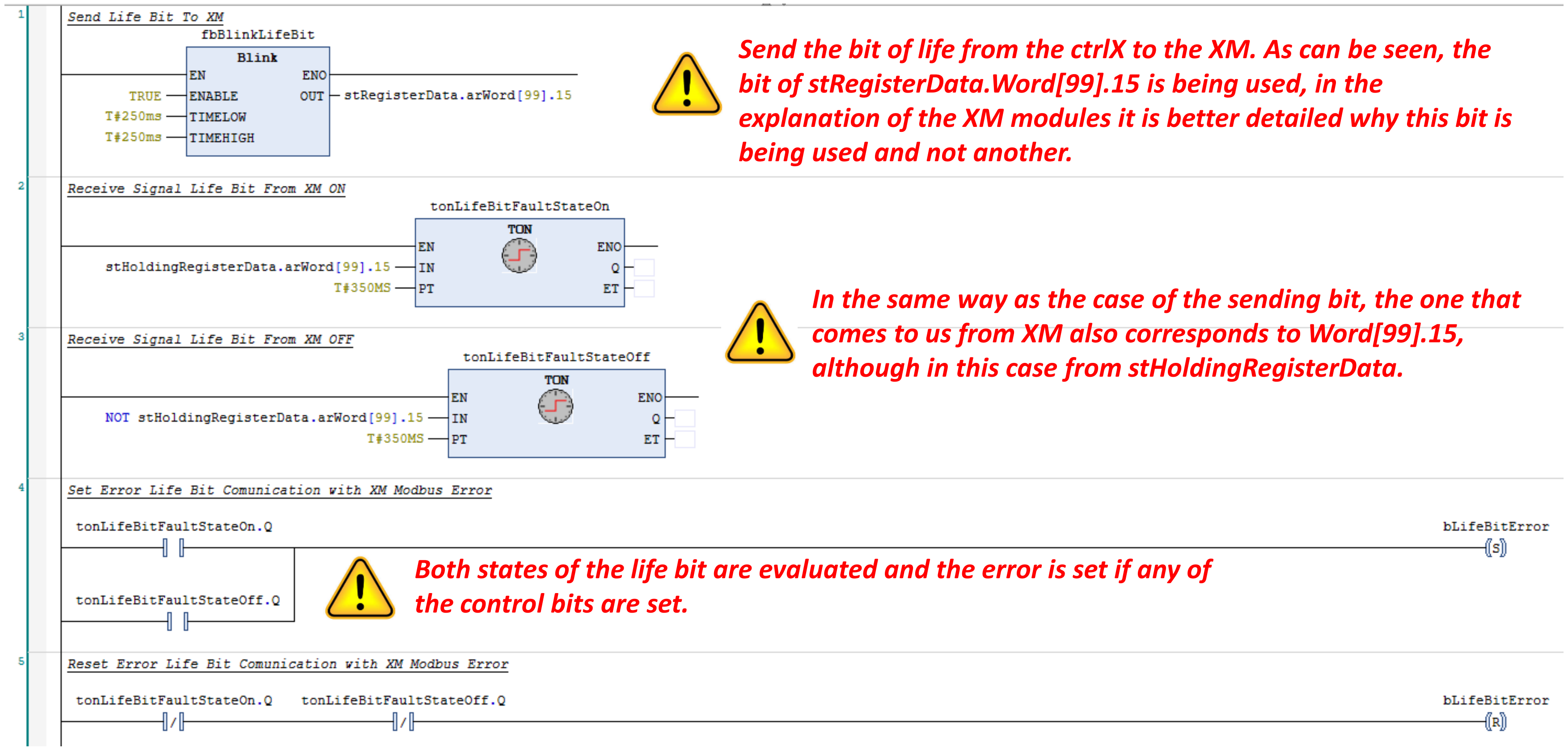
The last module located in the ctrlX manages the control of the life bit that is sent to the XM and the error control in the communications coming from the XM



Variables

```
PROGRAM Prog0070_0003_ModbusLifeBit
VAR
    fbBlinkLifeBit              : Blink;
    tonLifeBitFaultStateOn      : TON;
    tonLifeBitFaultStateOff     : TON;
END_VAR
```

Code



Send the bit of life from the ctrlX to the XM. As can be seen, the bit of stRegisterData.Word[99].15 is being used, in the explanation of the XM modules it is better detailed why this bit is being used and not another.

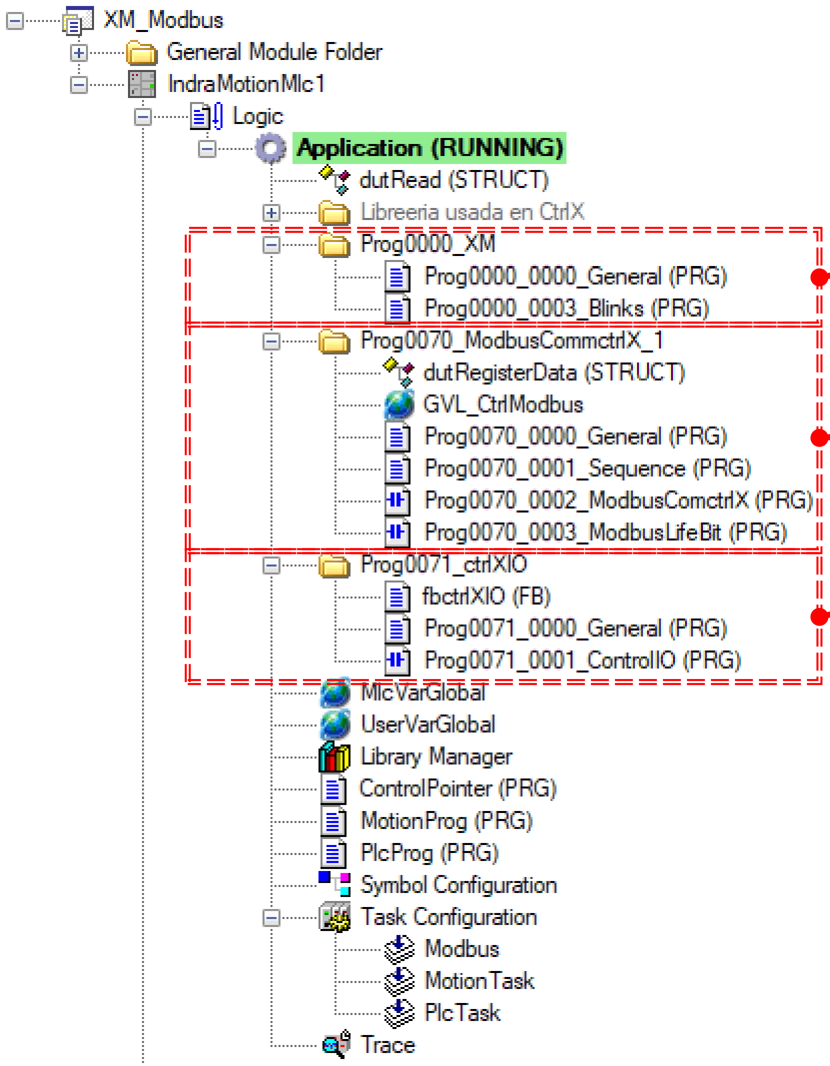
In the same way as the case of the sending bit, the one that comes to us from XM also corresponds to Word[99].15, although in this case from stHoldingRegisterData.

Both states of the life bit are evaluated and the error is set if any of the control bits are set.

Program used in the XM

ctrlX - Control program used in the XM

In the XM part and only for Modbus communications control we will use the following structure:



The rest of the program will be conditioned to the machine's own functionalities and each of them will have a different structure.

XM control. In the example there is only one Blink module that I don't know how to use.

Modbus communication control. It includes the control sequence and the activation modules, for the RegisterData and the HoldingRegisterData, as well as the Bit of Life control module.

Modules for receiving and sending IO states

rexroth
A Bosch Company

The structure of the Modbus communications area used in the XM is as follows:

- Prog0070_ModbusCommctrlX_1
  - dutRegisterData (STRUCT)
  - GVL_CtrlModbus
  - Prog0070_0000_General (PRG)
  - Prog0070_0001_Sequence (PRG)
  - Prog0070_0002_ModbusComctrlX (PRG)
  - Prog0070_0003_ModbusLifeBit (PRG)

```
TYPE dutRegisterData :
STRUCT
  arWord:   ARRAY[0..999]OF WORD;
  arInt:    ARRAY[0..999]OF INT;
  arUint:   ARRAY[0..999]OF UINT;
  arDint:   ARRAY[0..499]OF DINT;
  arUdint:  ARRAY[0..499]OF UDINT;
  arReal:   ARRAY[0..499]OF REAL;
  arString: ARRAY[0..99]OF STRING(19);
END_STRUCT
END_TYPE
```

Array structures that can be used for system control. This structure is identical to the one used in the ctrlX part.

Modules used for communication

Values received for the inputs from the ctrlX

Values sent from the XM to the ctrlX and that will activate the outputs

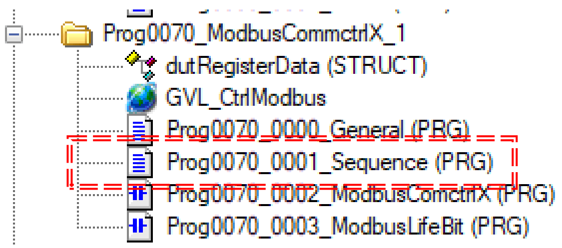
```
VAR_GLOBAL
// Control Sequence
    iSecComModbus        :INT;    // Modbus Communication Sequence
    wStepsModbus         :WORD;   // Word for Control Sequence Steps
    wStepsModbusMask     :WORD;   // Word Mask for Reset values Sequence Steps
    iErrorStep           :INT;    // Step Number Error
    strErrorMessage      :STRING; // String wiht error message step
    bSequenceStart       :BOOL;   // Sequence Enabled
    bLifeBitError        :BOOL;   // Comunication with ctrlX Modbus Error
// Modules for Modbus Communication Control
    fbModbusCLient: IL_ModBusTCPClient;
    fbModBusReadRegisters_Word:IL_ModBusTCPReadInputRegisters;
    fbModbusWriteMHR_Word: IL_ModBusTCPWriteMultipleRegisters;
// Values For I/O Image
    stRegisterData           :dutRegisterData; // Values From ctrlX
    stHoldingRegisterData    :dutRegisterData; // Values To ctrlX
END_VAR

VAR_GLOBAL CONSTANT
// Var Constant for Sequence Modbus Steps
Step00:INT:=0;
Step01:INT:=1;
Step02:INT:=2;
Step03:INT:=3;
END_VAR
```


rexroth
A Bosch Company

ctrlX - *Control program used in the XM*

The sequence control module manages the steps for the activation of the modules and in case of an error in the communications it is restarted to be able to start again automatically.



The sequence in this case consists of four steps

Prog0070_0001_Sequence

Sequence Control

Step 0: Initialization
Step 1: Activate Communication
Step 2: Reading RegisterData (Inputs)
Step 3: Write HoldingRegisterData (Outputs)

Prog0070_0002_ModbusComctrlX

Execution Modules

Activate Communication Modbus

Reading RegisterData (Inputs)

Write HoldingRegisterData (Outputs)

rexroth
A Bosch Company

View of the sequence control, steps 0 and 1, as well as the communications activation module

Prog0070_0001_Sequence

Sequence Control

```
CASE iSecComModbus OF
0: // Initialization Communication Module
wStepsModbus:=wStepsModbusMask;
bSequenceStart:=FALSE;
iSecComModbus:=1;
1:  // Initialization. Enable Communication Module
wStepsModbus:=wStepsModbusMask;
IF NOT fbModbusCLient.InOperation AND NOT fbModbusCLient.Error THEN
    wStepsModbus.Step01:=TRUE;  // Enable
END_IF

IF fbModbusCLient.Enable AND fbModbusCLient.Error THEN
    wStepsModbus.Step01:=FALSE;  // Enable
    iErrorStep:=1;
    strErrorMessage:='Communication Not Start';
END_IF

IF fbModbusCLient.Enable AND fbModbusCLient.InOperation THEN
    bSequenceStart:=TRUE;
    wStepsModbus:=wStepsModbusMask;
    iSecComModbus:=2;
    iErrorStep:=0;
    strErrorMessage:='';
END IF
```

Step 0, variable initialization

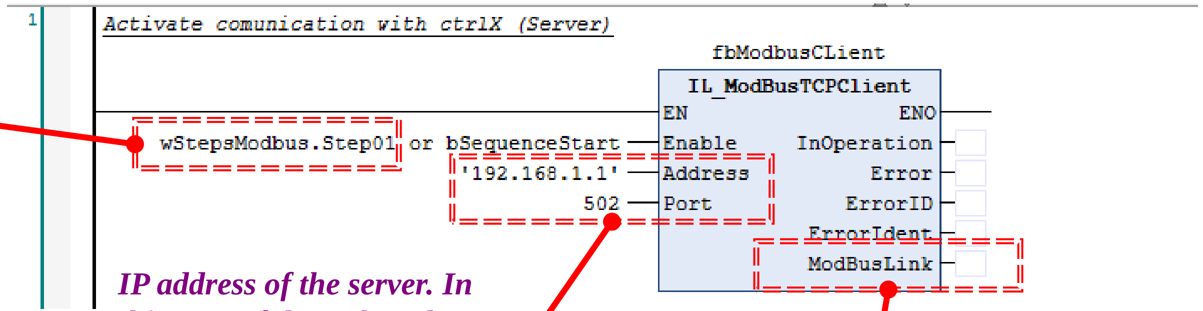
Step 1: Activate Communication

In case of error, the pass bit is deactivated and the system reconnection is attempted.

If the system establishes Modbus communication, the active sequence bit is activated and the jump to the next step is executed.

Prog0070_0002_ModbusComctrlX

Execution Modules



IP address of the server. In this case of the ctrlX. The default port is 502

Modbus Link address that we will use to control the read/write modules

View of Step 2, Reading the registers with the image of the inputs:

Step 2: Activate reading of the "RegisterData"

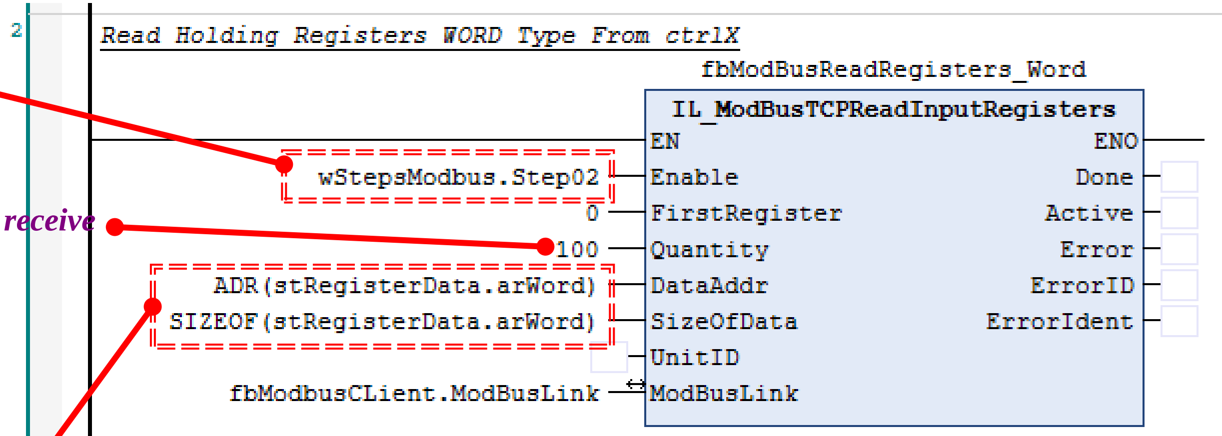
```
2: // Read Register Word
wStepsModbus:=wStepsModbusMask;
wStepsModbus.Step02:=TRUE;
// Module error check
IF fbModBusReadRegisters_Word.Enable AND fbModBusReadRegisters_Word.Error THEN
    iErrorStep:=2;// Modificar salto al paso 5 Antes 2
    strErrorMessage:='Module Read Register Word Fault';
END_IF

// If the module runs correctly, jump to the next step
IF fbModBusReadRegisters_Word.Enable AND fbModBusReadRegisters_Word.Done THEN
    wStepsModbus:=wStepsModbusMask;
    bSequenceStart:=TRUE;
    iSecComModbus:=3;
    iErrorStep:=0;
    strErrorMessage:='';
END_IF
// Control of general communication for restart in case of failure
IF fbModbusCLient.Error THEN
    iSecComModbus:=0;
END_IF
```

Step 2: Error control in the read block

Chequeo de errores para inicialización del sistema

Number of items to send / receive



The stRegisterData receive information on the status of the inputs from the ctrlX.

rexroth
A Bosch Company

View Step 3, writing the registers for the activation of the outputs:

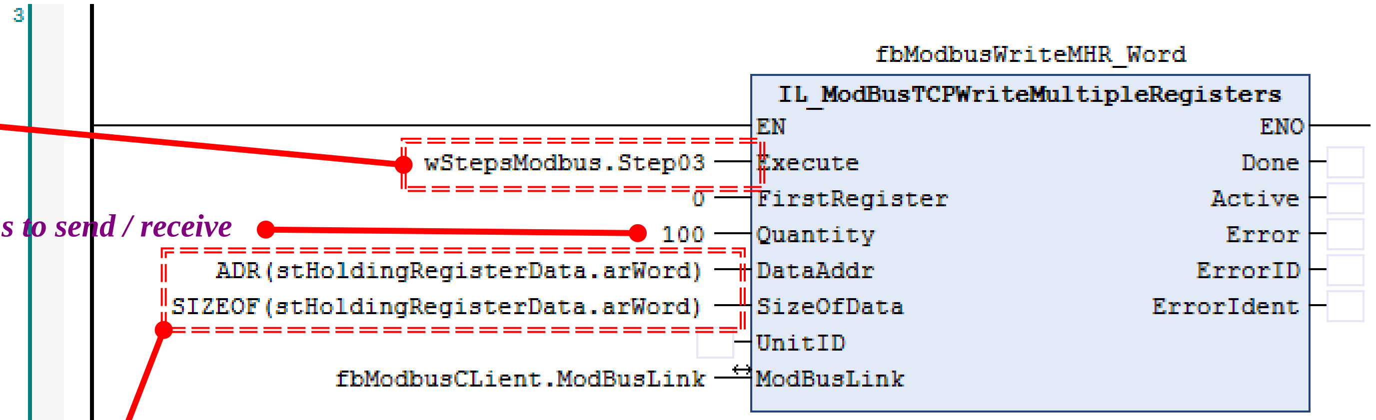
Step 3: Activate Writing of the "HoldingRegisterData"

```
3: // Read Holding Registers in Word Type
wStepsModbus:=wStepsModbusMask;
wStepsModbus.Step03:=TRUE;
// Module error check
IF fbModbusWriteMHR_Word.Execute AND fbModbusWriteMHR_Word.Error THEN
    iErrorStep:=3;
    strErrorMessage:='Module Write HoldingRegisters in Word Type Fault';
END_IF
// If the module runs correctly, jump to the next step
IF fbModbusWriteMHR_Word.Execute AND fbModbusWriteMHR_Word.Done THEN
    wStepsModbus:=wStepsModbusMask;
    bSequenceStart:=TRUE;
    iSecComModbus:=2; // modificar salto al paso 2 antes 6
    iErrorStep:=0;
    strErrorMessage:='';
END_IF
// Control of general communication for restart in case of failure
IF fbModbusCLient.Error THEN
    iSecComModbus:=0;
END_IF
END_CASE
```

Step 3: Error handling in the write block

Error check for system initialization

Number of items to send / receive



The HoldingstRegisterData send the activation data of the outputs to the ctrlX

rexroth
A Bosch Company

The last of the Modbus control POUs is the control life bit. As in the ctrlX part, here we also activate a bit to send to the XM and manage the control of possible errors in the communication coming from ctrlX

Prog0070_ModbusCommctrlX_1
- dutRegisterData (STRUCT)
- GVL_CtrlModbus
- Prog0070_0000_General (PRG)
- Prog0070_0001_Sequence (PRG)
- Prog0070_0002_ModbusComctrlX (PRG)
- Prog0070_0003_ModbusLifeBit (PRG)

Variables

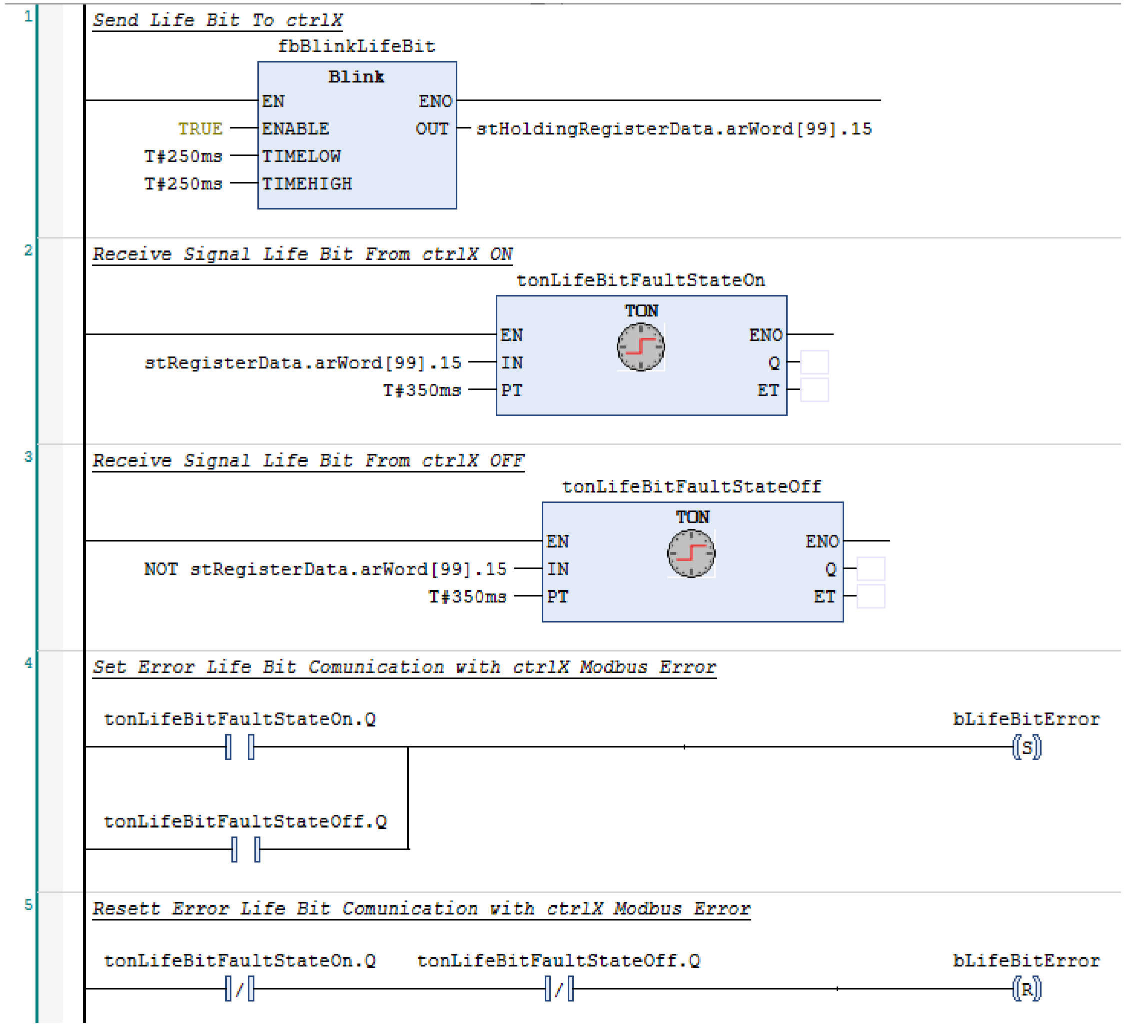
```
PROGRAM Prog0070_0003_ModbusLifeBit
VAR
    fbBlinkLifeBit: Blink;
    tonLifeBitFaultStateOn: TON;
    tonLifeBitFaultStateOff: TON;
END_VAR
```

Send the life bit from XM to ctrlX. As can be seen, the bit of stHoldingRegisterData.Word[99].15 is being used, in the explanation of the XM modules it is better detailed why this bit is being used and not another.

In the same way as the case of the sent bit, the one that comes to us from XM also corresponds to Word[99].15, although in this case from stRegisterData.

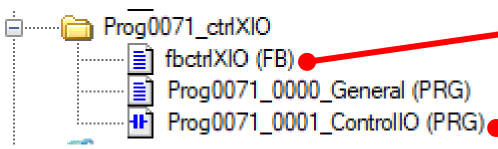
Both states of the life bit are evaluated and the error is set if any of the control bits are set.

Código



rexroth
A Bosch Company

Then we have the modules that are responsible for managing the input and output data



Fb for I/O control, which will help us to instantiate the various input and output registers

I/O control module

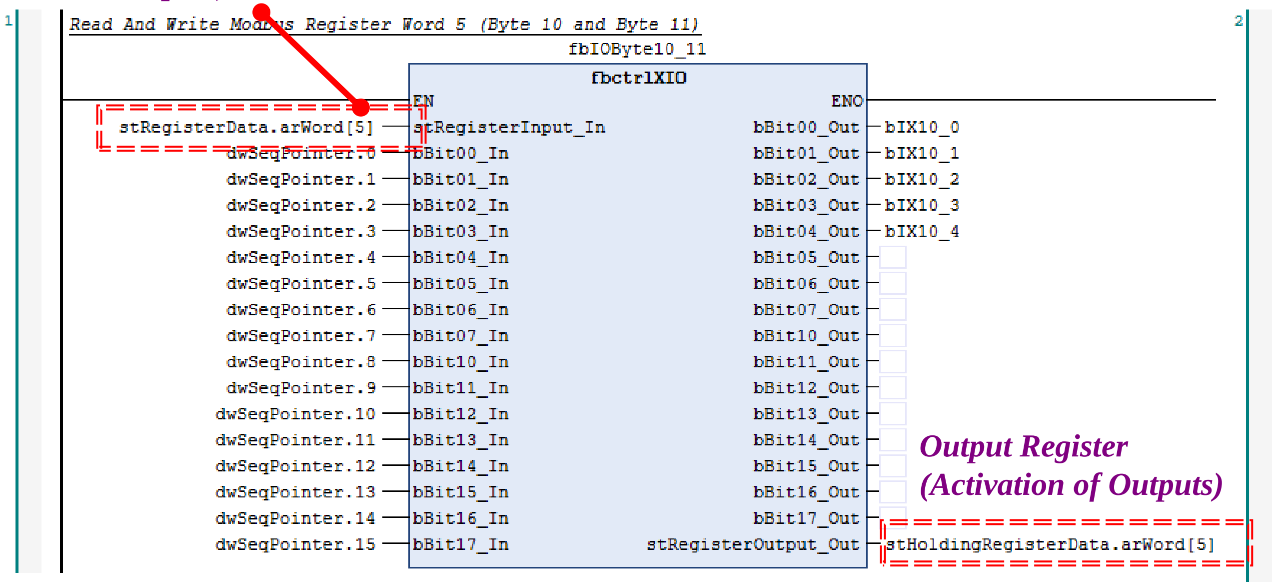
```
PROGRAM Prog0071_0001_ControlIO
VAR
    fbIOByte10_11: fbctrlXIO;
    fbIOByte12_13: fbctrlXIO;
    bIX10_0: BOOL; // Entrada %IX10.0
    bIX10_1: BOOL; // Entrada %IX10.1
    bIX10_2: BOOL; // Entrada %IX10.2
    bIX10_3: BOOL; // Entrada %IX10.3
    bIX10_4: BOOL; // Entrada %IX10.4
END_VAR
```

Instanciados para los registros En el ejemplo dos módulos

Ejemplo de bits utilizados para las entradas

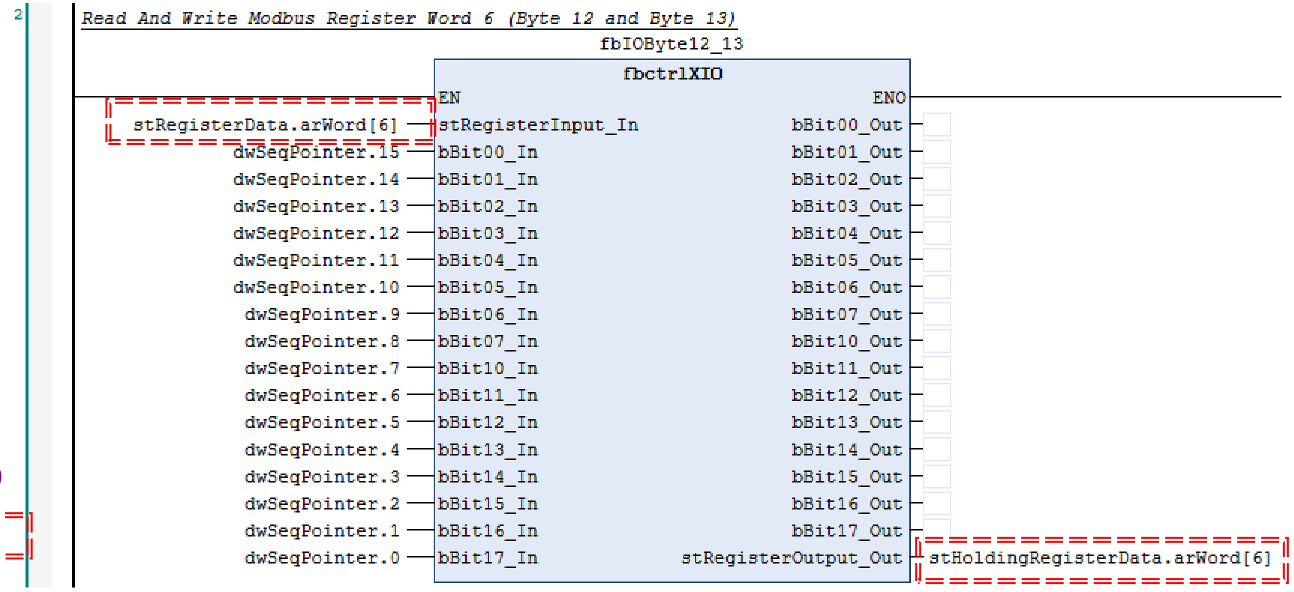
Input register (Image of the inputs)

Word Number 5 (Bytes 10 y 11)



Output Register (Activation of Outputs)

Word Number 6 (Bytes 12 y 13)

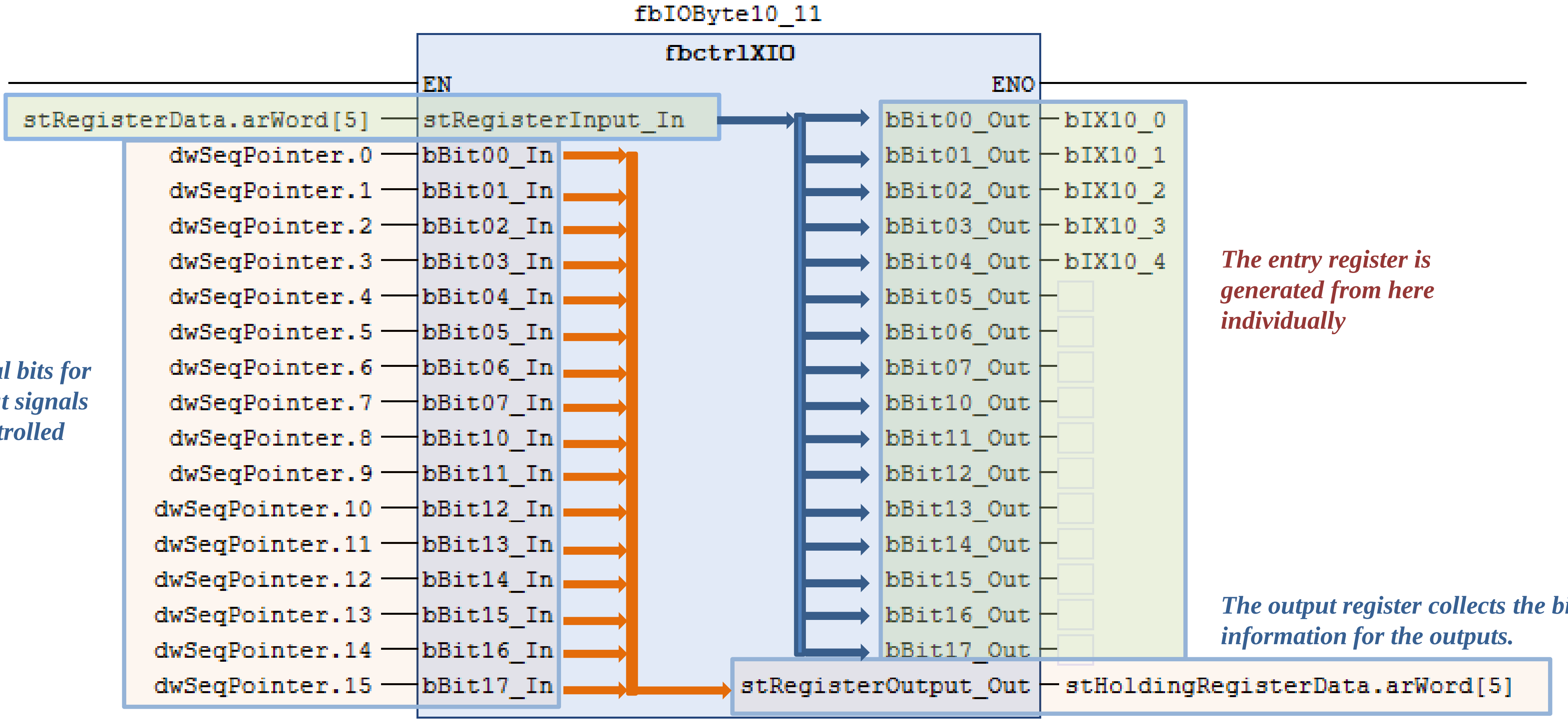


rexroth
A Bosch Company

The instantiated modules control the read and write registers one by one, in order to facilitate the assignment of the bits that must be the image of the inputs and of the bits for the outputs used in the project.

The input register internally generates the bits individually

Individual bits for the output signals to be controlled



The entry register is generated from here individually

The output register collects the bit information for the outputs.

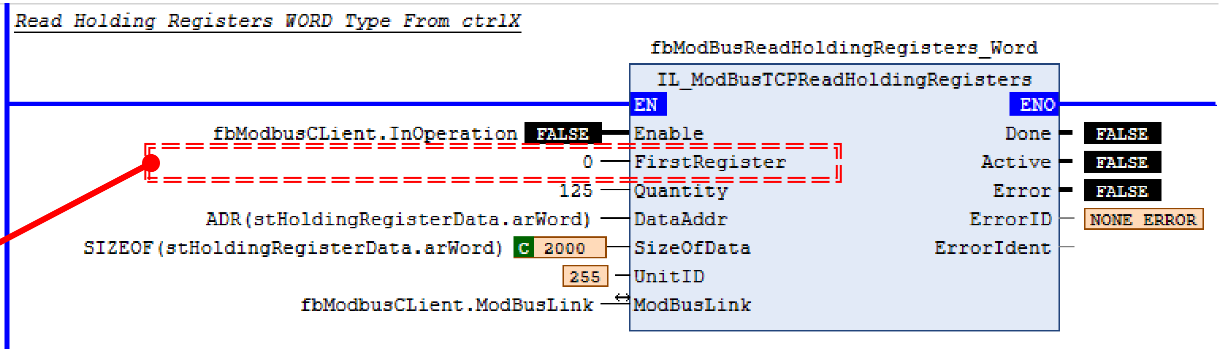
rexroth
A Bosch Company

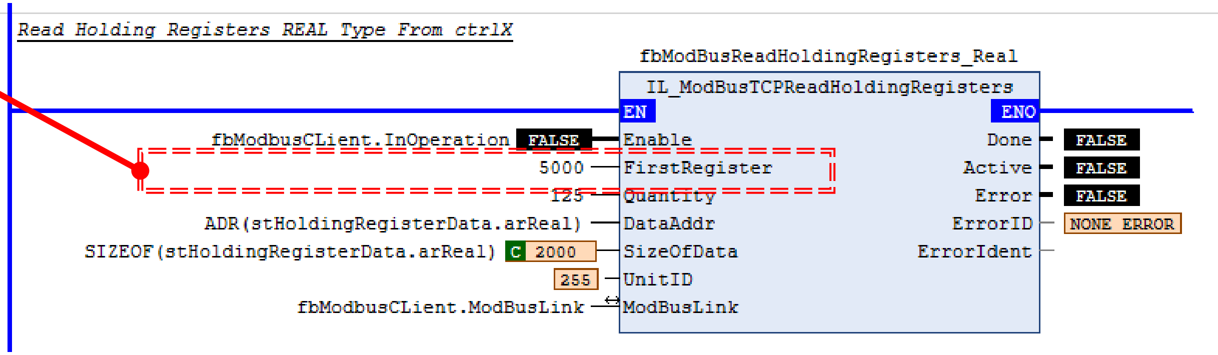
The communication structure of the areas used in the Registerdata and HoldingRegisters are in Word format, so we must calculate at what point each read / write sector is initialized, especially if we want to send or receive more than 125 elements.

The first 4 areas of the structure are in Word format, so the real occupation corresponds 1 to 1. The following areas are in double word format, so the assigned value should be multiplied * 2, in this way we can adjust the values and know in which area we can act.

From the example and with the structure shown, each area has a starting point

stHoldingRegisterData	dutRegisterData	
arWord	ARRAY [0..999] OF WORD	*0*
arInt	ARRAY [0..999] OF INT	*1000*
arUint	ARRAY [0..999] OF UINT	*2000*
arDint	ARRAY [0..499] OF DINT	*3000*
arUdint	ARRAY [0..499] OF UDINT	*4000*
arReal	ARRAY [0..499] OF REAL	*5000*
arString	ARRAY [0..99] OF STRING(19)	*6000*





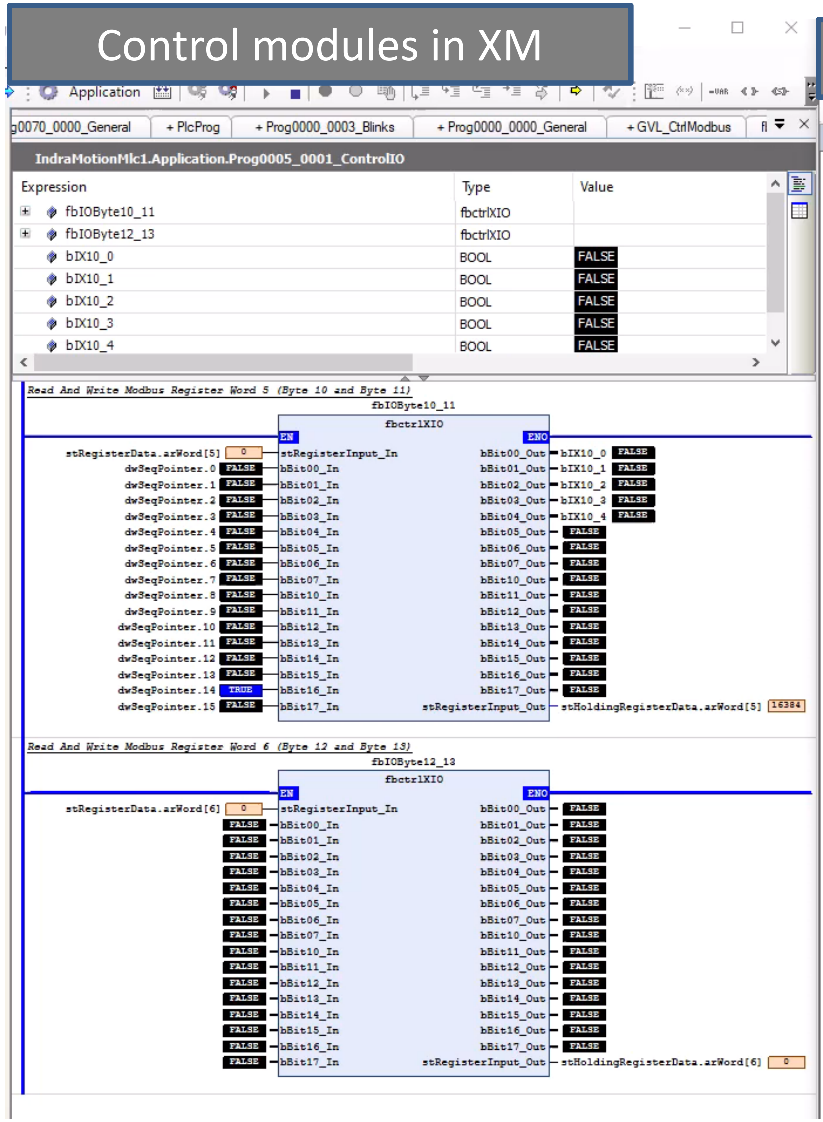
The modules only allow sending 125** elements each time through the "Quantity" parameter, if we have an area of 1000 words, for example, we must take into account that access should be made in eight blocks, modifying in this case the value assigned in "FirstRegister"

	Start	*End*
1	0	124
2	125	249
3	250	374
4	375	499
5	500	624
6	625	749
7	750	874
8	875	999

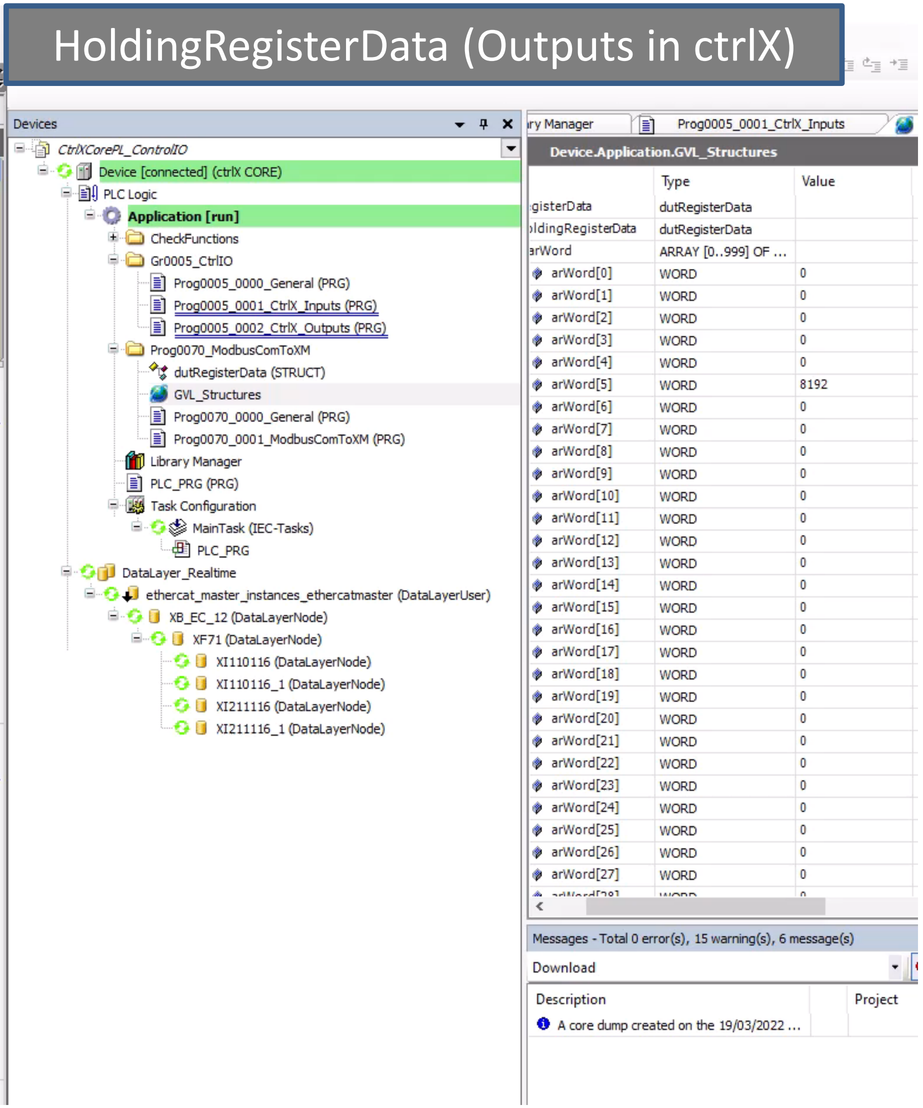
*****In some cases it has been observed that only a maximum of 100 elements could be sent, so in the communication example used, we are using a 100 and that is why we use Word[99].15 as the life bit***

rexroth
A Bosch Company

Control modules in XM



HoldingRegisterData (Outputs in ctrlX)



rexroth
A Bosch Company

Notes:

- ***It is recommended to use a sequence of steps to carry out a controlled sending of the data.***
- ***If this is not done, errors may occur.***
- ***The communication must be associated with a standard Task and never with a Sercos task.***
- ***The sequence used in the XM part manages the control of the reads and the error in the initial communication.***
- ***This programming is done my way and it is obvious that the final goal can be achieved in many ways.***
- ***This is just a small example of Modbus communication and some program changes may be required to get the best results.***
- ***As we have some areas for reading and writing and others for reading only, we can use them separately and in this way manage the data sent and the data received.***

Thanks for your attention

rexroth
A Bosch Company