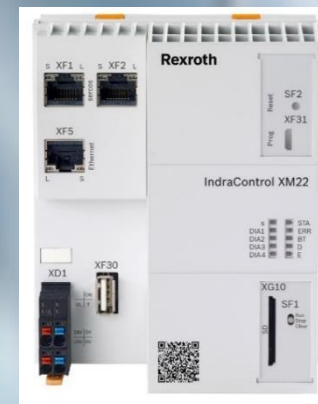
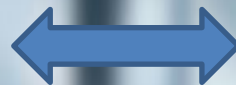


ctrlX - CORE

- *Communication with XM*

- *Cyclic TCP Communication (with IL_TCPCyclic Module)*
- *Communication Using Network Variables*
- *Communications using Cyclic UDP (With IL_UDPCyclic Module)*

Jordi Laboria (DCET/SLF4-ES)

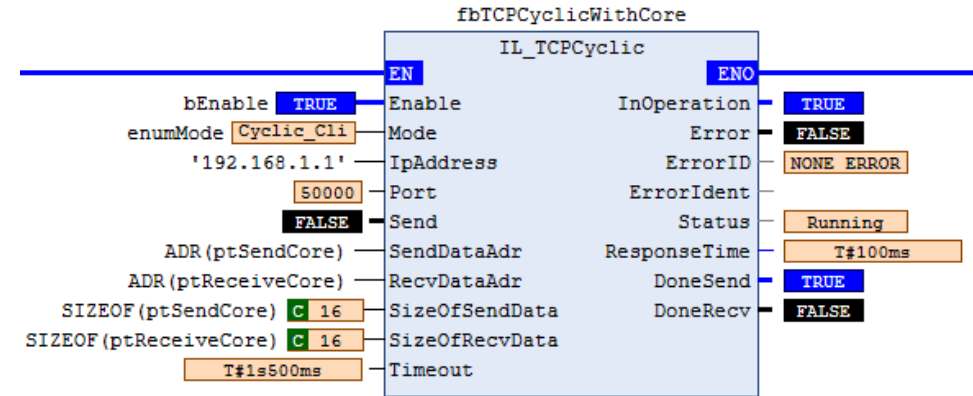
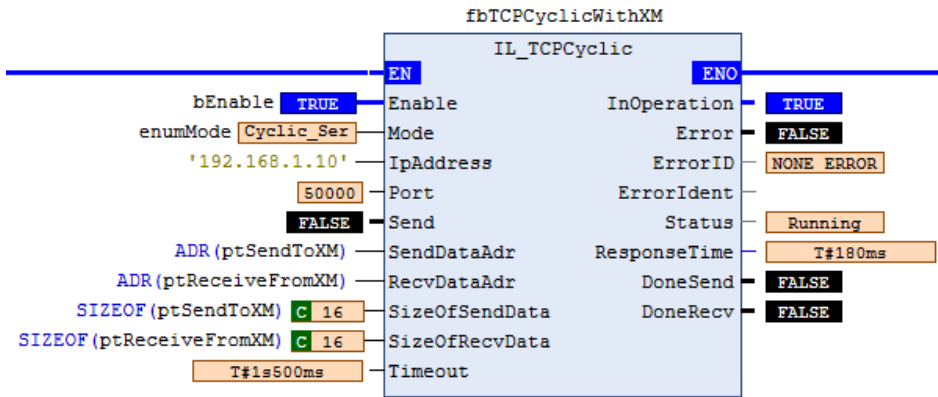


Goals:

- Establish communication between ctrlX Core and XM using the IL_TCPCyclic module
- Establish communication between ctrlX Core and XM with the use of network variables
- Establish communication between ctrlX Core and XM with the use of the IL_UDPCyclic module



Communication With IL_TCP Cyclic



- The procedure for managing sending and receiving data is extremely easy.
 - The first step, logically is the location and insertion of the libraries that contain the module that we are going to use
 - In ctrlX Core or ctrlXDrive with Core, the library will be CXA_SOCKETCOMM
 - In XM the library will be RIL_SOCKETCOMM

ctrlX Core

XM

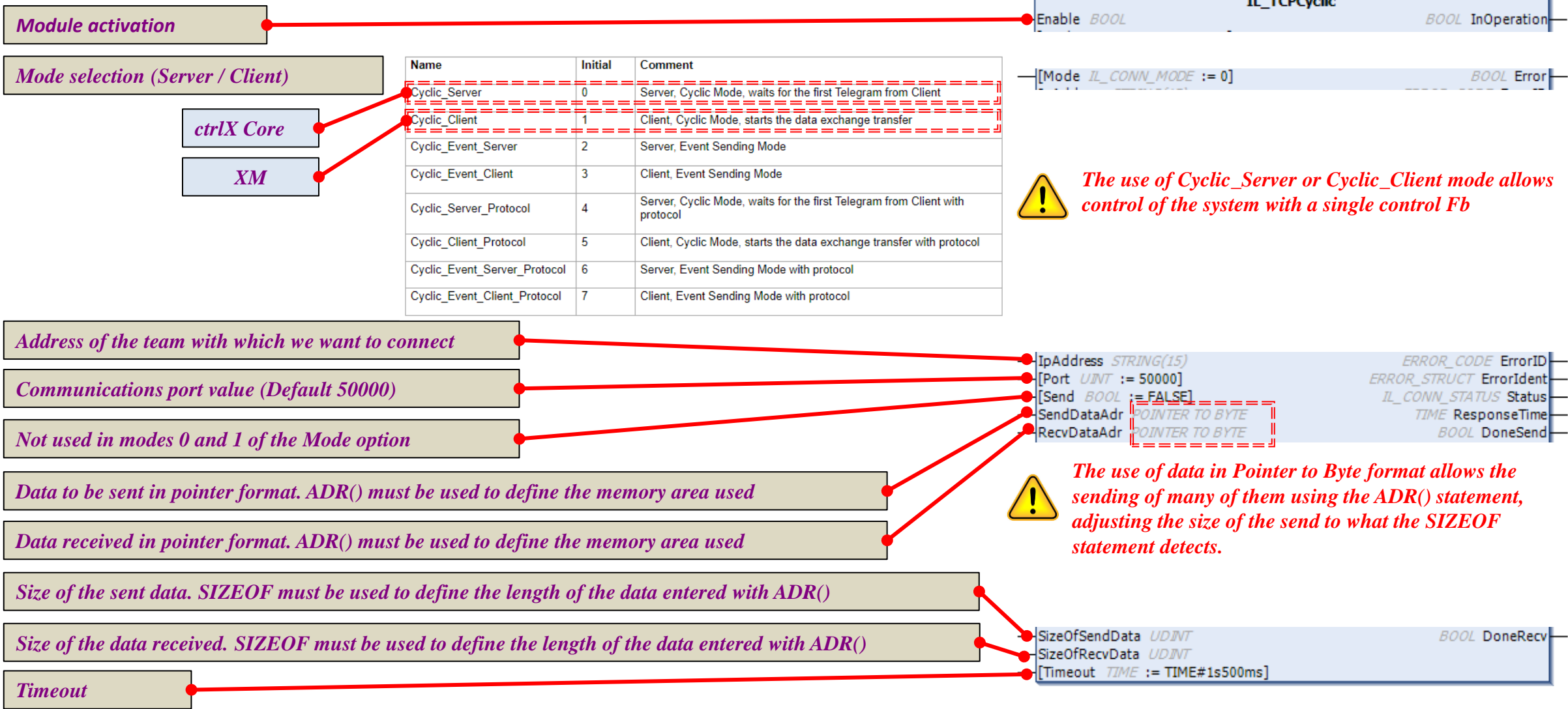
IL_TCPcyclic

IL_TCPcyclic

In both cases and regardless of whether it will be "server" or "client" the library will be the same: IL_TCPcyclic

ctrlX - Example communication TPC Cyclic (Description Inputs module)

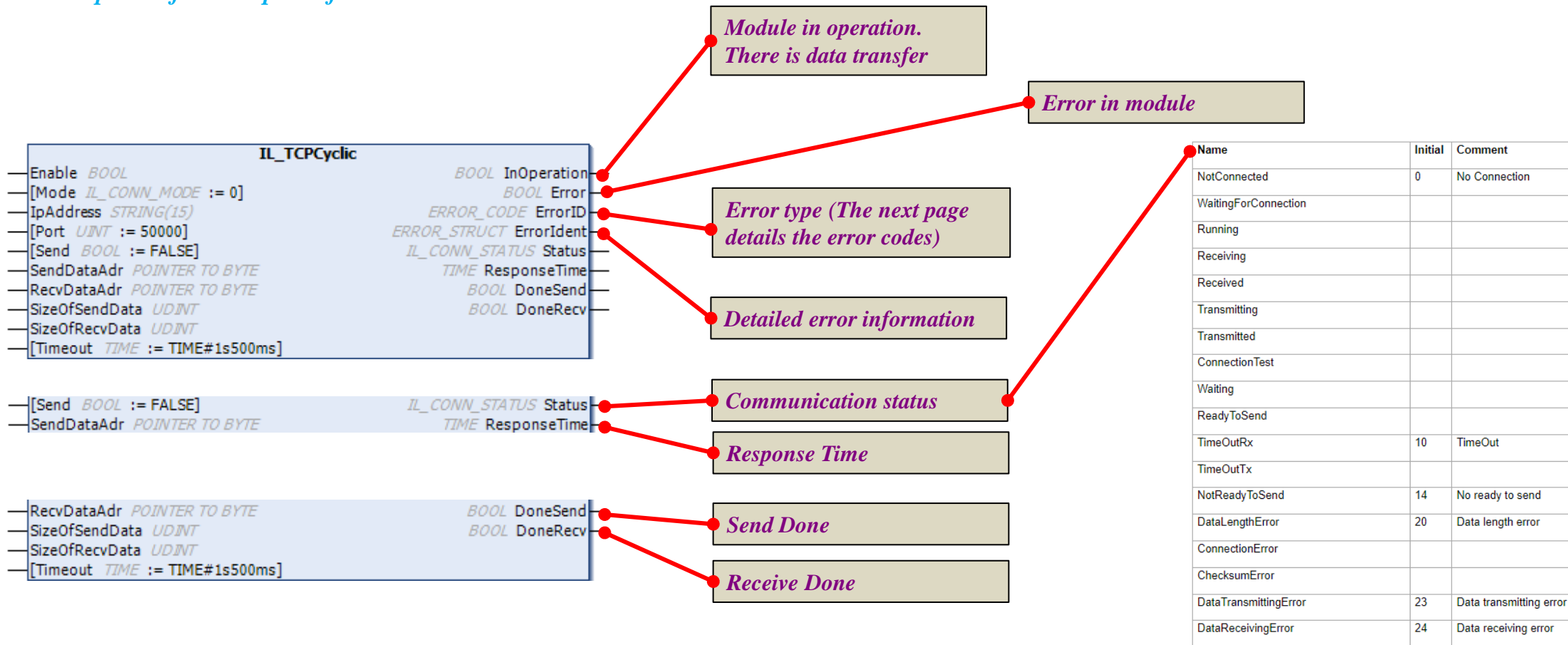
- Description of the inputs of the communications module:



! The use of Cyclic_Server or Cyclic_Client mode allows control of the system with a single control Fb

! The use of data in Pointer to Byte format allows the sending of many of them using the ADR() statement, adjusting the size of the send to what the SIZEOF statement detects.

- Description of the outputs of the communications module:



The states of the module allow us to find out how the system is, however, if we are slightly cautious, the program will rearm itself, either in case of disconnection of the communication cable or loss of power of one of the elements (ctrlX Core or XM), so that we should always be connected if the conditions are correct.

- Module error codes:

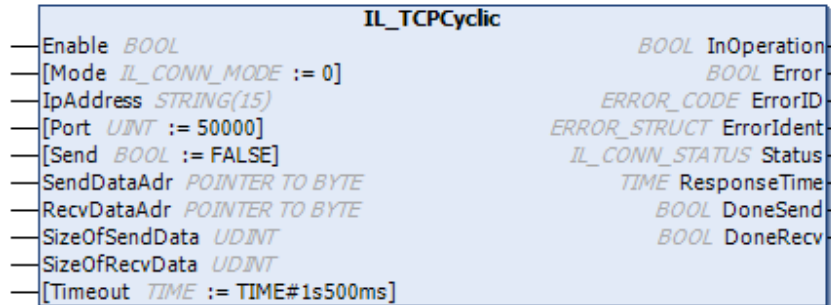
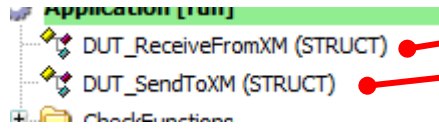


Table: ETHERNET_TABLE

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#0040	16#000D	Couldn't create socket.
OTHER_ERROR	16#0041	16#000E	Couldn't set I/O mode.
OTHER_ERROR	16#0042	16#000F	Couldn't bind socket.
OTHER_ERROR	16#0043	16#0010	Couldn't places socket to listening for incoming connection.
OTHER_ERROR	16#0044	16#0011	Couldn't accept incoming connection attempt on socket.
OTHER_ERROR	16#0045	16#0012	Couldn't connect to socket.
INPUT_INVALID_ERROR	16#0202	16#FEFE/16#0032	Invalid socket handle.
INPUT_INVALID_ERROR	16#0204	16#FEF8/16#0016	Invalid port number.
INPUT_INVALID_ERROR	16#0208	16#FEFC/16#0028	Invalid destination address.
COMMUNICATION_ERROR	16#0207	16#0000	Port/Address already in use.
COMMUNICATION_ERROR	16#0209	16#FED7/16#003D	Connection refused.
COMMUNICATION_ERROR	16#020A	16#FEC0/16#003C	TimeOut occured (Default = 1500ms).
COMMUNICATION_ERROR	16#020B	16#0000	Host not found.
COMMUNICATION_ERROR	16#020C	16#FED8/16#0041	No route to host.
COMMUNICATION_ERROR	16#020E	16#0000	Connection closed.
COMMUNICATION_ERROR	16#020F	16#0000	Software caused connection abort.
COMMUNICATION_ERROR	16#0210	16#0000	Data packet oversized. SendDataAdr is not greater than 0.
COMMUNICATION_ERROR	16#0210	16#0001	SizeOfSendData is not greater than 0.
COMMUNICATION_ERROR	16#0210	16#0002	DataLength limit of SendDataAdr reached, max. 1400 BYTE.
COMMUNICATION_ERROR	16#0210	16#0003	Data packet oversized. SendDataAdr is not greater than 0.
COMMUNICATION_ERROR	16#0210	16#0004	SizeOfRecvData is not greater than 0.
COMMUNICATION_ERROR	16#0210	16#0005	DataLength limit of RecvDataAdr reached, max. 1400 BYTE.
COMMUNICATION_ERROR	16#0210	16#0006	SizeOfRecvData is smaller than size of received Data.
COMMUNICATION_ERROR	16#0211	16#0000	Connection reset by peer.
COMMUNICATION_ERROR	16#0212	16#0000	Incompatible protocol.
COMMUNICATION_ERROR	16#1001	16#0000	Data Transmitting Error. SendData not equal to RecvData
COMMUNICATION_ERROR	16#1002	16#0000	Wrong Mode selected

- Example of program in ctrlX Core: (Structures and variables)

Structures used in the example



ctrlX Core

```
1 TYPE DUT_SendToXM :  
2 STRUCT  
3 wWord01ToXm: WORD;  
4 wWord02ToXm: WORD;  
5 wWord03ToXm: WORD;  
6 wWord04ToXm: WORD;  
7 wWord05ToXm: WORD;  
8 wWord06ToXm: WORD;  
9 wWord07ToXm: WORD;  
10 wWord08ToXm: WORD;  
11 wWord09ToXm: WORD;  
12 wWord10ToXm: WORD;  
13 END_STRUCT  
14 END_TYPE
```

```
1 TYPE DUT_ReceiveFromXM :  
2 STRUCT  
3 wWord01FromXm: WORD;  
4 wWord02FromXm: WORD;  
5 wWord03FromXm: WORD;  
6 wWord04FromXm: WORD;  
7 wWord05FromXm: WORD;  
8 wWord06FromXm: WORD;  
9 wWord07FromXm: WORD;  
10 wWord08FromXm: WORD;  
11 wWord09FromXm: WORD;  
12 wWord10FromXm: WORD;  
13 END_STRUCT  
14 END_TYPE
```

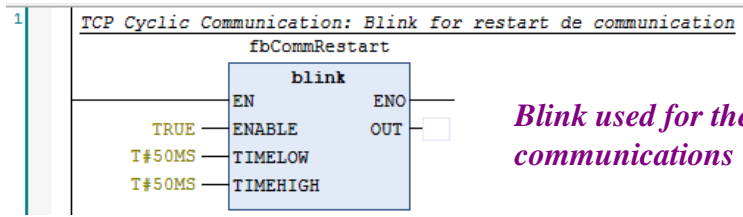
```
2 VAR  
3 // VARIABLES  
4 // Communication Enable  
5 bEnable :BOOL;  
6 // Structures For Receive and Send To XM  
7 ptSendToXM : DUT_SendToXM;  
8 ptReceiveFromXM : DUT_ReceiveFromXM;  
9 // Enumerator for Communication Define Type  
10 enumMode : IL_CONN_MODE;  
11 // Blinking Signal For Communication Start  
12 fbCommRestart : Blink;  
13 // Module for Communication Control  
14 fbTCPCyclicWithXM : IL_TCPCyclic;  
15 END_VAR  
16
```

Variables used in the example

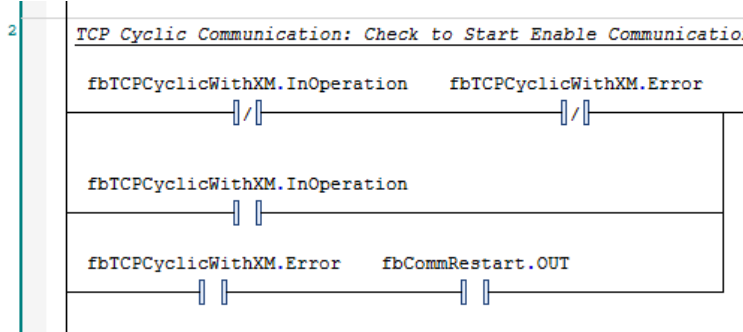
ctrlX - Example communication TPC Cyclic (Example program in ctrlX Core)

- Example of program in the ctrlX Core: (Code)

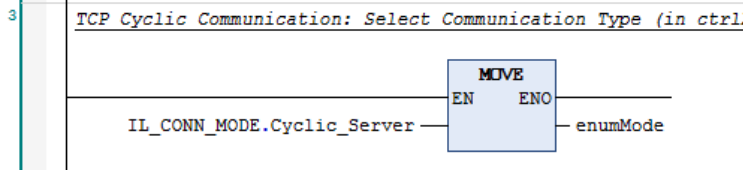
ctrlX Core



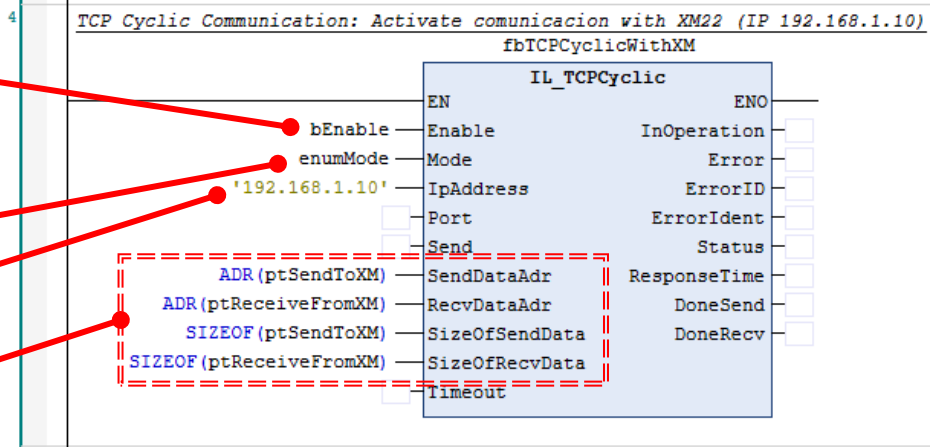
Blink used for the "restart" of communications



Bit for module activation



Selecting the type of communication used

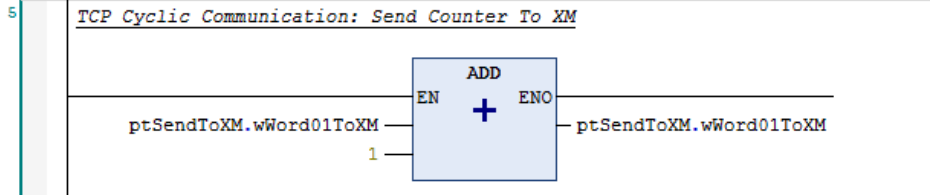


XM IP address

Variable structure

ctrlX Core

Name	Initial	Comment
Cyclic_Server	0	Server, Cyclic Mode, waits for the first Telegram from Client
Cyclic_Client	1	Client, Cyclic Mode, starts the data exchange transfer



Counter for communications test

- Example of program in the XM: (Structures and variables)

Structures used in the example

- DUT_ReceiveFromCore (STRUCT)
- DUT_SendToCore (STRUCT)

XM

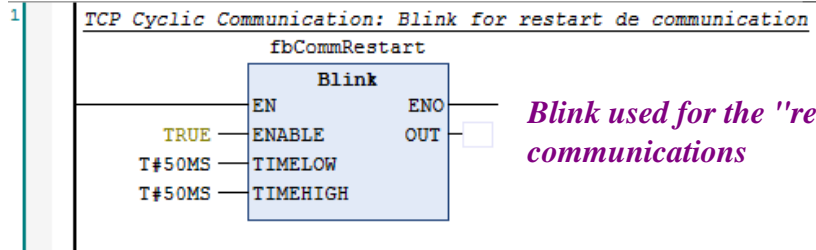
```
1 TYPE DUT_SendToCore :  
2 STRUCT  
3   wWord01ToCore: WORD;  
4   wWord02ToCore: WORD;  
5   wWord03ToCore: WORD;  
6   wWord04ToCore: WORD;  
7   wWord05ToCore: WORD;  
8   wWord06ToCore: WORD;  
9   wWord07ToCore: WORD;  
10  wWord08ToCore: WORD;  
11  wWord09ToCore: WORD;  
12  wWord10ToCore: WORD;  
13 END_STRUCT  
14 END_TYPE  
15
```

```
1 TYPE DUT_ReceiveFromCore :  
2 STRUCT  
3   wWord01FromCore: WORD;  
4   wWord02FromCore: WORD;  
5   wWord03FromCore: WORD;  
6   wWord04FromCore: WORD;  
7   wWord05FromCore: WORD;  
8   wWord06FromCore: WORD;  
9   wWord07FromCore: WORD;  
10  wWord08FromCore: WORD;  
11  wWord09FromCore: WORD;  
12  wWord10FromCore: WORD;  
13 END_STRUCT  
14 END_TYPE  
15
```

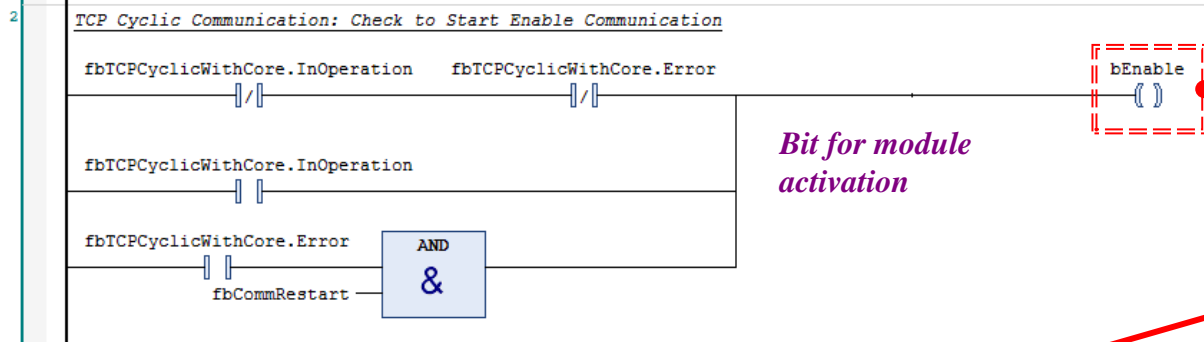
```
1 PROGRAM POU_CommTCPCyclic  
2 VAR  
3 // VARIABLES  
4 // Communication Enable  
5   bEnable          : BOOL;  
6 // Structures For Receive And Send Data To ctrlx Core  
7   ptSendCore       : DUT_SendToCore;  
8   ptReceiveCore    : DUT_ReceiveFromCore;  
9 // Enumerator For Communication Define Type  
10  enumMode          : IL_CONN_MODE;  
11 // Blinking Signal For Communication Start  
12  fbCommRestart     : Blink;  
13 // Module for Communication Control  
14  fbTCPCyclicWithCore : IL_TCPCyclic;  
15 END_VAR  
16
```

Variables used in the example

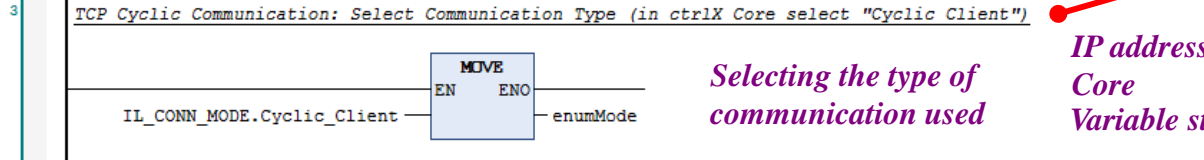
- Example of program in the XM: (Code)



Blink used for the "restart" of communications



Bit for module activation

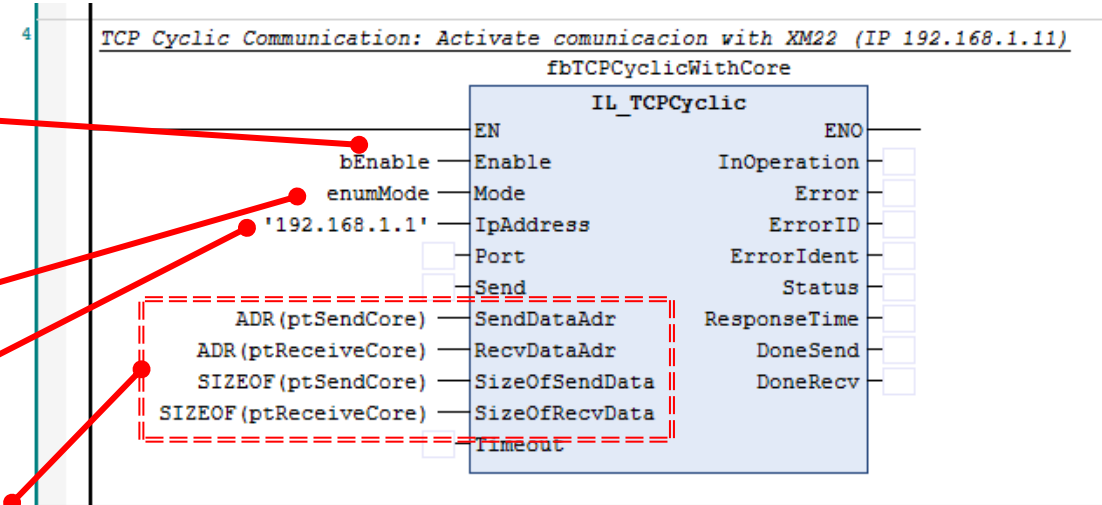


Selecting the type of communication used

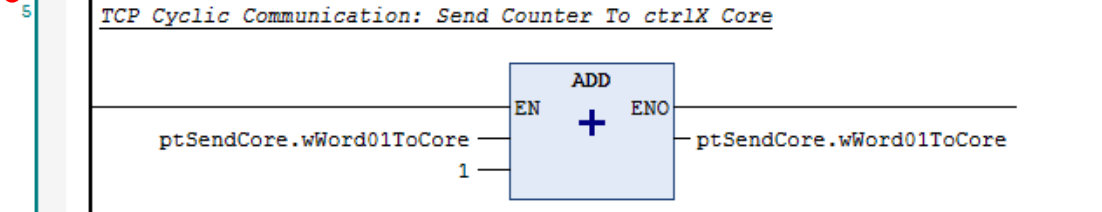
IP address of the ctrlX Core Variable structure

Name	Initial	Comment
Cyclic_Server	0	Server, Cyclic Mode, waits for the first Telegram from Client
Cyclic_Client	1	Client, Cyclic Mode, starts the data exchange transfer

XM



Counter for communications test



ctrlX - Example communication TPC Cyclic (Video Example)

The screenshot displays the SIMATIC Manager interface with the following components:

- Library Manager:** Shows the loaded library `Device.Application.POU_CommTCPcyclic` with functions like `IL_PingAsync`, `bExecute`, `bEnable`, `ptSendToXM`, `ptReceiveFromXM`, `enumMode`, `BLINK_0`, and `fbTCPcyclicWithXM`.
- Watch 1:**

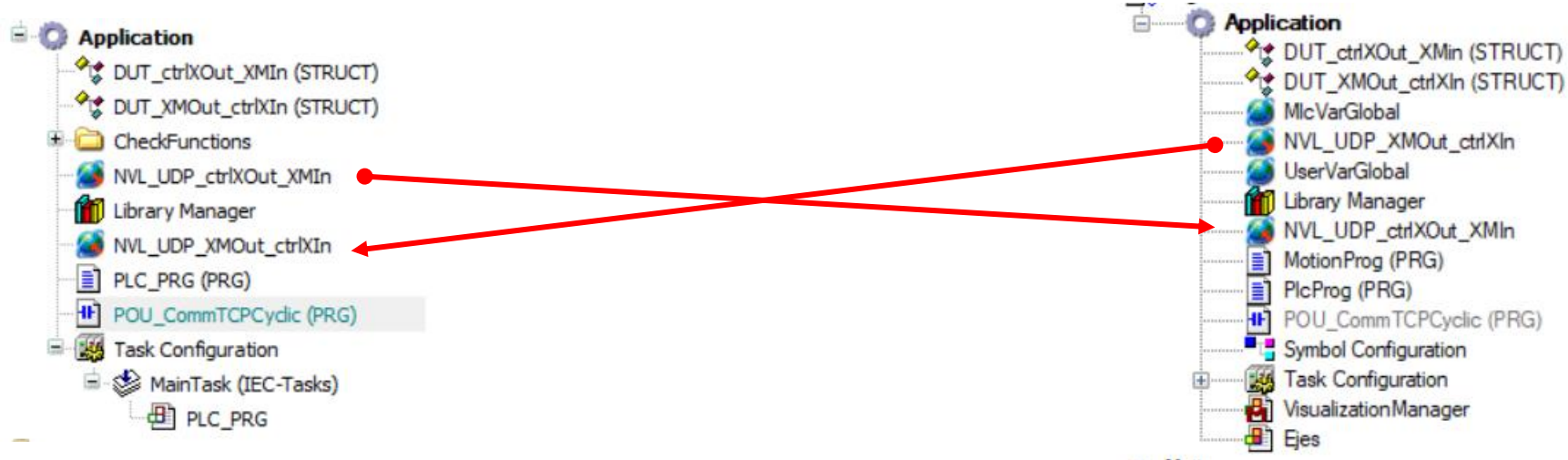
Expression	Appli...	Type	Value
<code>POU_CommTCPcyclic.ptReceiveFromXM.wWord01FromXm</code>	Devic...	WORD	13440
<code>POU_CommTCPcyclic.ptSendToXM.wWord01ToXm</code>	Devic...	WORD	62252
- Monitoring 1:**

Expression	Application	Type	Value
<code>POU_CommTCPcyclic.ptReceiveCore.wWord01FromCore</code>	IndraMotionMlc1.Ap...	WORD	62252
<code>POU_CommTCPcyclic.ptSendCore.wWord01ToCore</code>	IndraMotionMlc1.Ap...	WORD	13442
- Logic Diagrams:**
 - fbTCPcyclicWithXM:** A function block for communication with the XM. It includes a `MOVE` block to set `enumMode` to `Cyclic_Ser` (0) and an `fbTCPcyclicWithCore` block. The latter has inputs for `bEnable` (TRUE), `enumMode` (Cyclic_Ser), `IPAddress` ('192.168.1.10'), `Port` (50000), `Send` (FALSE), `ADR` (ptSendToXM), `SIZEOF` (16), and `Timeout` (T#1s500ms). Its outputs include `InOperation` (TRUE), `Error` (FALSE), `ErrorID` (NONE ERROR), `Status` (Running), `ResponseTime` (T#180ms), `DoneSend` (FALSE), and `DoneRecv` (TRUE).
 - fbTCPcyclicWithCore:** A function block for communication with the ctrlX Core. It has similar inputs and outputs, with `enumMode` set to `Cyclic_Cli` (1) and `ResponseTime` (T#100ms).
 - ADD Blocks:** Two `ADD` blocks are used to increment counters. One increments `ptSendToXM.wWord01ToXM` (62252) by 1, and the other increments `ptSendCore.wWord01ToCore` (13442) by 1.

ctrlX Core

XM

Communication with Network Variables



- CodeSys also allows communication via list of specific variables for sending and receiving over the network.

ctrlX Core

XM

Variable tables for communication

Variable tables for communication

- Cut
- Copy
- Paste
- Delete
- Refactoring
- Properties...
- Add Object**
- Add Folder...
- Edit Object
- Edit Object With...
- Insert templates...
- Login
- Delete application from device
- Alarm Configuration...
- Application...
- Axis Group...
- Cam table...
- CNC program...
- CNC settings...
- Communication Manager...
- Data Sources Manager...
- DUT...
- External File...
- Global Variable List...
- Global Variable List (tasklocal)...
- Image Pool...
- Interface...
- Network Variable List (Receiver)...**
- Network Variable List (Sender)...**
- Persistent Variables...
- POU...
- Recipe Manager...
- Symbol Configuration...
- Text List...
- Trace...
- Trend Recording Manager...
- Unit Conversion...
- Visualization...
- Visualization Manager...

Add

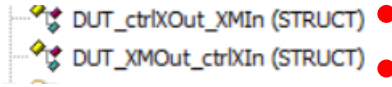
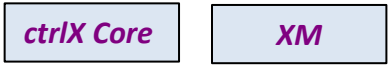
- Browse Cross References
- Browse Call Tree
- Rename 'Application'
- Data import...
- Login
- Delete application from device
- Export
- Import...
- Online compare
- Compare...
- Paste (Ctrl+V)
- Delete (Del)
- Rename (F2)
- Find element...
- ECAD resource identifier
- Print preview...
- Print... (Ctrl+P)
- New view
- Properties...

Variable tables for communication

- Add...
- Data types...
- Folder...
- GVL for logic exchange...
- Global Network Variable List (receiver)...**
- Global Network Variable List (sender)...
- Global Persistent Variable List...
- Global Variable List...
- Image pool...
- Interface...
- POU...
- Recipe manager...
- Text list...
- Trace...
- Visualization...
- POU templates

ctrlX - Example communication with Network Variables

- We will start by defining the table of variables in the ctrlX Core and also in the XM




The number of variables used is the same as that used in the previous example, however the names have been modified since now we will have some files that we must access to establish communication

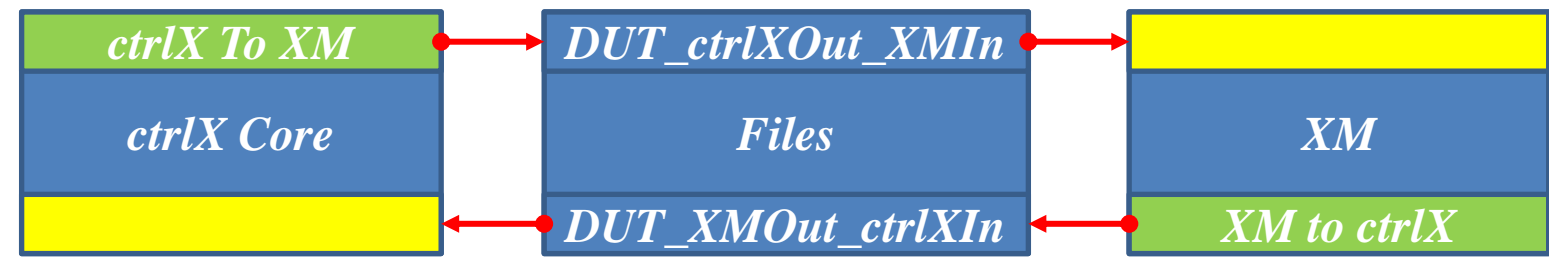
```

1  TYPE DUT_XMOut_ctrlXIn :
2  STRUCT
3  wW01XMOut_ctrlXIn: WORD;
4  wW02XMOut_ctrlXIn: WORD;
5  wW03XMOut_ctrlXIn: WORD;
6  wW04XMOut_ctrlXIn: WORD;
7  wW05XMOut_ctrlXIn: WORD;
8  wW06XMOut_ctrlXIn: WORD;
9  wW07XMOut_ctrlXIn: WORD;
10 wW08XMOut_ctrlXIn: WORD;
11 wW09XMOut_ctrlXIn: WORD;
12 wW10XMOut_ctrlXIn: WORD;
13 END_STRUCT
14 END_TYPE
    
```

```

1  TYPE DUT_ctrlXOut_XMIn :
2  STRUCT
3  wW01ctrlXOut_XMIn: WORD;
4  wW02ctrlXOut_XMIn: WORD;
5  wW03ctrlXOut_XMIn: WORD;
6  wW04ctrlXOut_XMIn: WORD;
7  wW05ctrlXOut_XMIn: WORD;
8  wW06ctrlXOut_XMIn: WORD;
9  wW07ctrlXOut_XMIn: WORD;
10 wW08ctrlXOut_XMIn: WORD;
11 wW09ctrlXOut_XMIn: WORD;
12 wW10ctrlXOut_XMIn: WORD;
13 END_STRUCT
14 END_TYPE
    
```


 *These structures should be identical in both devices*



- Description of options for variables of type "Sender"

Network type	UDP
Task	Task of the current application that controls the variables to be sent. IndraLogic always sends the variables at the end of a task cycle.
Variable list ID	To identify the network variable list. Has to be unambiguous.
Pack variables	<input checked="" type="checkbox"/> : IndraLogic bundles the variables for sending in packets (telegrams); the size depends on the network type. <input type="checkbox"/> : IndraLogic generates one package per variable.
Transmit checksum	<input checked="" type="checkbox"/> : A checksum is provided for each variable packet. The receiver checks the checksum to make sure that the variable definitions match from the sender and receiver. A packet with nonmatching checksums is not accepted.
Confirmation	<input checked="" type="checkbox"/> : IndraLogic sends an acknowledgment message for each received data package. If the sender does not receive a confirmation before it sends again, then an error is written to the diagnostic structure. Note: From version 3.5.7.0 of the NetVarUdp library, no reception channel is assigned anymore if no confirmed transfer is selected. Thus, network variables can also be exchanged between two controls on one hardware device.
Cyclic transmission, interval	IndraLogic sends the variables within the defined interval. Example of time definition: "T#70ms".
Transfer upon change, minimum gap	<input checked="" type="checkbox"/> : IndraLogic sends the variables only if their values have changed. You can use "minimum gap" to define the least amount of time between two transmissions.
Transmit on event, variable	<input checked="" type="checkbox"/> : IndraLogic sends the variables as soon as the defined variable yields TRUE.
Settings	Log-specific settings; possible entries depend on the network library: Port: Number of the port that IndraLogic uses for data exchange with other network units. The Default value is "1202". You can change the current value in the Value field at any time: Select the field, press the <spacebar>, enter the value. Attention: The other nodes in the network must define the same port! If more than one UDP connection is defined in the project, then the port numbers in all configurations are adapted to this value. Broadcast Addr.: The preset value is "255. 255. 255. 255" means that data is exchanged with all network devices. Change the current value in the Value field: Select field, press <spacebar>, enter the address or the address range of a subnetwork. Example: "197 . 200. 100 . 255" in case of communication with all nodes having IP addresses in the range from "197 . 200. 100 . x".

Add Network Variable List (Sender) ✕

 Create a global variable list to send via a network
(Use object properties to edit settings)

Name:

Network type: Settings...

Task:

Listidentifier:

Pack variables

Transmit checksum

Acknowledgement

Cyclictransmission

Transmit on change

Transmit on event

Interval:

Minimum gap:

Variable:



The "ListIdentifier" must be different in each of the transmission lists.

- We will first create a table of variables of type Network Variable List for sending data to the XM

ctrlX Core

Add Network Variable List (Sender)

Create a global variable list to send via a network
(Use object properties to edit settings)

Name: NVL_UDP_ctrlXOut_XMIn

Network type: UDP

Task: MainTask

Listidentifier: 1

Pack variables

Transmit checksum

Acknowledgement

Cyclictransmission Interval: T#50ms

Transmit on change Minimum gap: T#20ms

Transmit on event Variable:

Add Cancel

The type to select is UDP

And we can select with "Settings" the port and the network IP address where the information will be sent.

Parameter	Value	Default value
Port	1202	1202
Broadcast Adr.	255.255.255.255	255.255.255.255

OK Cancel



"Broadcast Adr." must contain the value of the IP of the equipment with which we want to communicate. The "Port" should be different for each communication module



The "ListIdentifier" must be different in each of the transmission elements

We will create the variable for communication using the previously generated structure

```
1 //({attribute 'qualified_only'})
2 VAR_GLOBAL
3     UDP_ctrlXOut_XMIn: DUT_ctrlXOut_XMIn;
4 END_VAR
```

- And we will repeat the same in the XM creating the list of network variables to which we will add the structure that we are going to use for communication

XM


```
1 TYPE DUT_XMOut_ctrlXIn :  
2 STRUCT  
3   wW01XMOut_ctrlXIn: WORD;  
4   wW02XMOut_ctrlXIn: WORD;  
5   wW03XMOut_ctrlXIn: WORD;  
6   wW04XMOut_ctrlXIn: WORD;  
7   wW05XMOut_ctrlXIn: BOOL;  
8   wW06XMOut_ctrlXIn: BOOL;  
9   wW07XMOut_ctrlXIn: BOOL;  
10  wW08XMOut_ctrlXIn: BOOL;  
11  wW09XMOut_ctrlXIn: BOOL;  
12  wW10XMOut_ctrlXIn: WORD;  
13 END_STRUCT  
14 END_TYPE
```

Add Network Variable List (Sender) ✕

Create a global variable list to send via a network
(Use object properties to edit settings)

Name:

Network type: UDP Settings...

Task: MotionTask

List identifier:

Pack variables
 Transmit checksum
 Acknowledgement

Cyclic transmission Interval:
 Transmit on change Minimum gap:
 Transmit on event Variable:

Finish Cancel

"Broadcast Adr." must contain the value of the IP of the equipment with which we want to communicate. The "Port" should be different for each communication module

The "ListIdentifier" must be different in each of the transmission elements

Keep in mind that if the communication files are not generated, we will not be able to create the network variables for receiving data

In this case we will also create the variable for communication using the previously generated structure.

```
1 //({attribute 'qualified_only'})  
2 VAR_GLOBAL  
3   UDP_XMOut_ctrlXIn: DUT_XMOut_ctrlXIn;  
4 END_VAR  
5
```

- In both cases, ctrlX and XM the generation of the file to include in the other PLC must be done in the same way

ctrlX Core

NVL_UDP_ctrlXOut_XMIn

Use as Plc Realtime data

- Cut
- Copy
- Paste
- Delete
- Browse
- Refactoring
- Properties...

Properties - NVL_UDP_ctrlXOut_XMIn [Device: PLC Logic: Application]

Common | **Link To File** | Build | Access Control | Network Variables

File name: []

Import before compile
 Export before compile

OK Cancel Apply

File name: UDP_ctrlXOut_XMIn
 Save as type: GVL Export files (*.gvl)

Save Cancel

File name: C:\Traspaso\ESCRIPTORI_LIMPIEZA\VideosProgramas\XM_ctrlX_Ti

OK

UDP_ctrlXOut_XMIn.gvl

```

1 <GVL>
2 <Declarations><![CDATA[(attribute 'qualified_only')
3 VAR_GLOBAL
4   UDP_ctrlXOut_XMIn: DUT_ctrlXOut_XMIn;
5 END_VAR]]></Declarations>
6 <NetVarSettings Protocol="UDP">
7   <ListIdentifier>1</ListIdentifier>
8   <Pack>True</Pack>
9   <Checksum>False</Checksum>
10  <Acknowledge>False</Acknowledge>
11  <CyclicTransmission>True</CyclicTransmission>
12  <TransmissionOnChange>False</TransmissionOnChange>
13  <TransmissionOnEvent>False</TransmissionOnEvent>
14  <Interval>T#50ms</Interval>
15  <MinGap>T#20ms</MinGap>
16  <EventVariable>
17  </EventVariable>
18  <ProtocolSettings>
19    <ProtocolSetting Name="Broadcast Adr." Value="192.168.1.10" />
20    <ProtocolSetting Name="Port" Value="1202" />
21  </ProtocolSettings>
22 </NetVarSettings>
23 </GVL>
    
```

Variable used (in the example)

List Identifier Number

Port and IP Number

File generated from the communications structure with extension *. GVL

- Creation of the file in the XM

The process involves the following steps:

- Right-clicking on the XM file `NVL_UDP_XMOut_ctrlXIn` and selecting `Properties...`.
- In the `Properties - NVL_UDP_XMOut_ctrlXIn` dialog, selecting the `Link To File` tab.
- Clicking the `...` button in the `Filename:` field to open a file selection dialog.
- Clicking `Save` in the file selection dialog to save the file as `UDP_XMOut_ctrlXIn.gvl`.
- Clicking `OK` in the `Properties` dialog.

The resulting GVL file content is as follows:

```
1 <GVL>
2 <Declarations><![CDATA[/(attribute='qualified-only')
3 VAR_GLOBAL
4 UDP_XMOut_ctrlXIN: DUT_XMOut_ctrlXIN;
5 END_VAR]]></Declarations>
6 <NetVarSettings Protocol="UDP">
7   <ListIdentifier>2</ListIdentifier>
8   <Pack>True</Pack>
9   <Checksum>False</Checksum>
10  <Acknowledge>False</Acknowledge>
11  <CyclicTransmission>True</CyclicTransmission>
12  <TransmissionOnChange>False</TransmissionOnChange>
13  <TransmissionOnEvent>False</TransmissionOnEvent>
14  <Interval>T#50ms</Interval>
15  <MinGap>T#20ms</MinGap>
16  <EventVariable>
17  </EventVariable>
18  <ProtocolSettings>
19    <ProtocolSetting Name="Broadcast Adr." Value="192.168.1.1" />
20    <ProtocolSetting Name="Port" Value="1203" />
21  </ProtocolSettings>
22 </NetVarSettings>
23 </GVL>
```

Annotations in the GVL code:

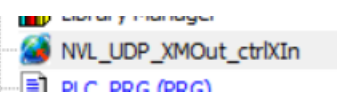
- `Variable used (in the example)` points to the `UDP_XMOut_ctrlXIN` declaration.
- `List Identifier Number` points to the `2` in `<ListIdentifier>2</ListIdentifier>`.
- `Port and IP Number` points to the `1203` and `192.168.1.1` values in the `<ProtocolSettings>` block.

UDP_XMOut_ctrlXIn.gvl 19/06/2023 10:32 GVL File

File generated from the communications structure with extension *. GVL

- Now we return to the ctrlX Core and incorporate the list of network variables

Variable tables Generated from the file



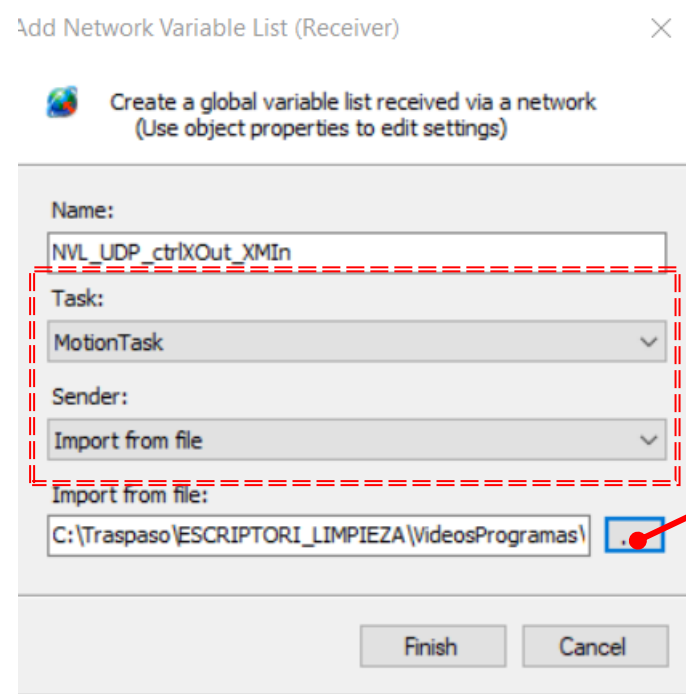
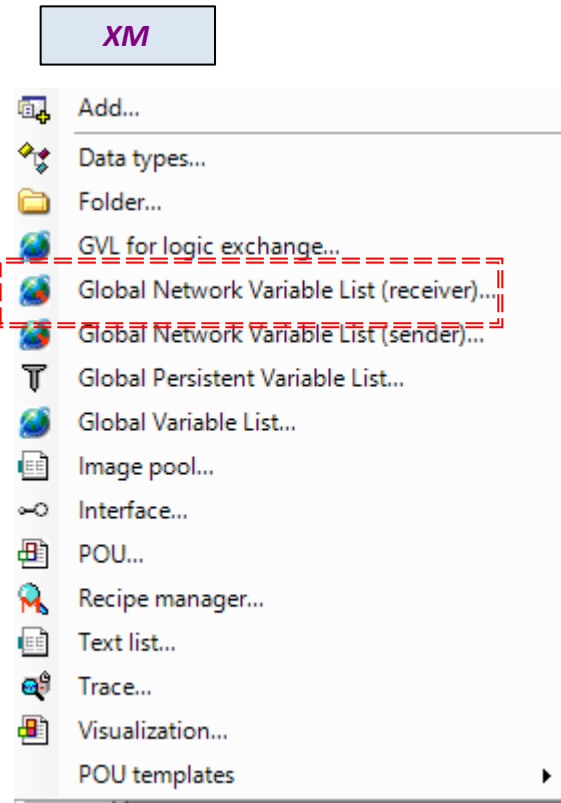
```

1 //This gobal variable list is received via the network.
2 //Sender: Imported from file 'C:\Traspaso\ESCRIPTORI_LIMPIEZA\VideosProgramas\XM_ctrlX_TCPCommunication\GVLFiles\UDP_XMOut_ctrlXIn.gvl'
3 //Protocol: UDP
4
5 //(attribute 'qualified_only')
6 VAR_GLOBAL
7     UDP_XMOut_ctrlXIn: DUT_XMOut_ctrlXIn;
8 END VAR
    
```



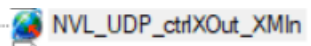
The data this table of variables cannot be modified from here.

- The first step, in any case, will be the incorporation of the desired kinematics to the project



UDP_ctrlXOut_XMIn.gvl

Variable tables Generated from the file



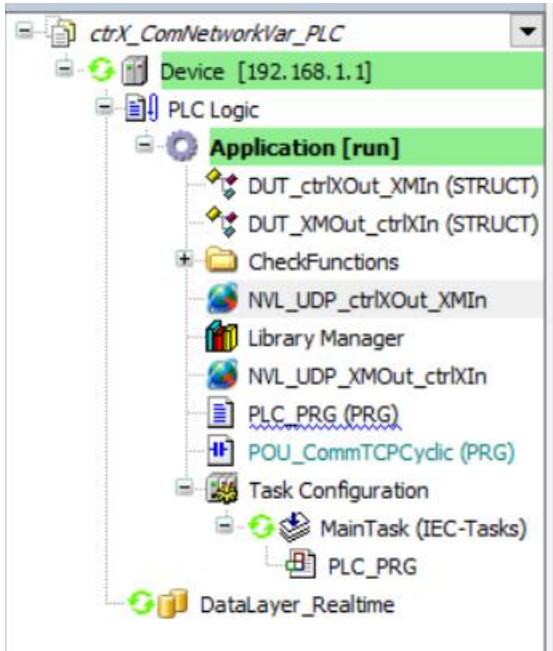
```
1 //This gobal variable list is received via the network.
2 //Sender: Imported from file 'C:\Traspaso\ESCRIPTORI_LIMPIEZA\VideosProgramas\XM_ctrlX_TCPCommunication\GVLFiles\UDP_ctrlXOut_XMIn.gvl'
3 //Protocol: UDP
4
5 //{attribute 'qualified_only'}
6 VAR_GLOBAL
7     UDP_ctrlXOut_XMIn: DUT_ctrlXOut_XMIn;
8 END_VAR
```



The data this table of variables cannot be modified from here.

- The system generates the group of communication variables automatically

ctrlX Core



Expression	type	Value
NetVarTxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
NetVarRxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
TxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_TX_UDP	
RxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_RX_UDP	
NetVarManager_UDP_MainTask_0	NETVARMANAGER_UDP_FB	
NetVarManager_UDP_MainTask_1	NETVARMANAGER_UDP_FB	
UDP_ctrlXOut_XMIn	DUT_ctrlXOut_XMIn	
wW01ctrlXOut_XMIn	WORD	345
wW02ctrlXOut_XMIn	WORD	152
wW03ctrlXOut_XMIn	WORD	789
wW04ctrlXOut_XMIn	WORD	45
wW05ctrlXOut_XMIn	WORD	546
wW06ctrlXOut_XMIn	WORD	121
wW07ctrlXOut_XMIn	WORD	0
wW08ctrlXOut_XMIn	WORD	0
wW09ctrlXOut_XMIn	WORD	0
wW10ctrlXOut_XMIn	WORD	0

 Variables Group Automatically generated

Variables used for communication

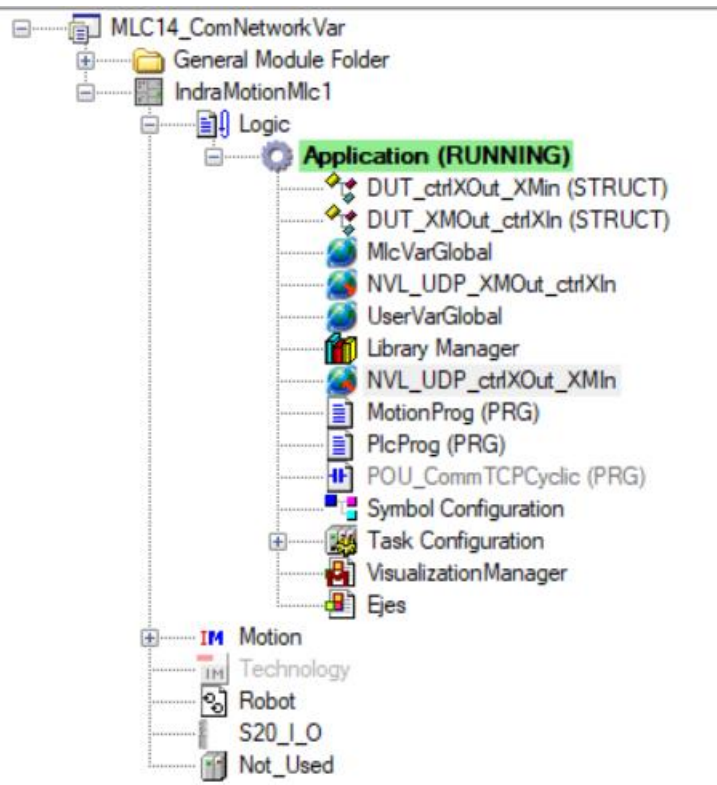
 Variables Group Automatically generated

Variables used for communication

Expression	type	value
NetVarTxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
NetVarRxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
TxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_TX_UDP	
RxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_RX_UDP	
NetVarManager_UDP_MainTask_0	NETVARMANAGER_UDP_FB	
NetVarManager_UDP_MainTask_1	NETVARMANAGER_UDP_FB	
UDP_XMOut_ctrlXIn	DUT_XMOut_ctrlXIn	
wW01XMOut_ctrlXIn	WORD	1311
wW02XMOut_ctrlXIn	WORD	2323
wW03XMOut_ctrlXIn	WORD	0
wW04XMOut_ctrlXIn	WORD	0
wW05XMOut_ctrlXIn	WORD	0
wW06XMOut_ctrlXIn	WORD	0
wW07XMOut_ctrlXIn	WORD	0
wW08XMOut_ctrlXIn	WORD	0
wW09XMOut_ctrlXIn	WORD	0
wW10XMOut_ctrlXIn	WORD	0

- In the XM it works the same way


XM



Expression	Type	Value
NetVarTxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
NetVarRxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
TxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_TX_UDP	
RxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_RX_UDP	
NetVarManager_UDP_MotionTask_0	NETVARMANAGER_UDP_FB	
NetVarManager_UDP_MotionTask_1	NETVARMANAGER_UDP_FB	
UDP_XMOut_ctrXIn	DUT_XMOut_ctrXIn	
wW01XMOut_ctrXIn	WORD	1311
wW02XMOut_ctrXIn	WORD	2323
wW03XMOut_ctrXIn	WORD	0
wW04XMOut_ctrXIn	WORD	0
wW05XMOut_ctrXIn	WORD	0
wW06XMOut_ctrXIn	WORD	0
wW07XMOut_ctrXIn	WORD	0
wW08XMOut_ctrXIn	WORD	0
wW09XMOut_ctrXIn	WORD	0
wW10XMOut_ctrXIn	WORD	0

 **Variables Group Automatically generated**

Variables used for communication

 **Variables Group Automatically generated**

Variables used for communication

Expression	Type	Value
NetVarTxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
NetVarRxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIAGSTRUCT	
TxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_TX_UDP	
RxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_RX_UDP	
NetVarManager_UDP_MotionTask_0	NETVARMANAGER_UDP_FB	
NetVarManager_UDP_MotionTask_1	NETVARMANAGER_UDP_FB	
UDP_ctrXOut_XMIn	DUT_ctrXOut_XMIn	
wW01ctrXOut_XMIn	WORD	345
wW02ctrXOut_XMIn	WORD	152
wW03ctrXOut_XMIn	WORD	789
wW04ctrXOut_XMIn	WORD	45
wW05ctrXOut_XMIn	WORD	546
wW06ctrXOut_XMIn	WORD	121
wW07ctrXOut_XMIn	WORD	0
wW08ctrXOut_XMIn	WORD	0
wW09ctrXOut_XMIn	WORD	0
wW10ctrXOut_XMIn	WORD	0

- Additional notes

- Different "ListIdentifier" for each of the sending groups

ctrlX Core	List identifier: <input type="text" value="1"/>	XM	List identifier: <input type="text" value="2"/>
-------------------	---	-----------	---

- Different "Port", if errors appear in the compilation process

- "Broadcast Adr." with the IP address of the equipment to be communicated

Parameter	Value	Default value
Port	1202	1202
Broadcast Adr.	192.168.1.10	255.255.255.255

OK
Cancel

Parameter	Value	Default value
Port	1203	1202
Broadcast Adr.	192.168.1.1	255.255.255.255

OK
Cancel



The port and IP address of the computer to be communicated is important because in some cases the system did not communicate correctly

ctrlX - Example communication with Network Variables

XM

*MLC14Safety_NetworkVar_2023V00.xiwp - IndraWorks Engineering

File Edit View Project NVL_UDP_XMOut_ctrlXIn Build Debug Diagnostics Tools Window Help

192.168.1.10 mode P0 P2 BB

Project Explorer

- MLC14_ComNetworkVar
 - General Module Folder
 - IndraMotionMlc1
 - Logic
 - Application (RUNNING)
 - DUT_ctrlXOut_XMin (STRUCT)
 - DUT_XMOut_ctrlXIn (STRUCT)
 - MlcVarGlobal
 - NVL_UDP_XMOut_ctrlXIn
 - UserVarGlobal
 - Library Manager
 - NVL_UDP_ctrlXOut_XMin
 - MotionProg (PRG)
 - PlcProg (PRG)
 - POU_CommTCPcyclic (PRG)
 - Symbol Configuration
 - Task Configuration
 - VisualizationManager
 - Ejes

IndraMotionMlc1.Application.NVL_UDP_XMOut_ctrlXIn

Expression	Type	Value
NetVarTxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIASTRUCT	
NetVarRxDiag_UDP	ARRAY [0..0] OF NETVARUDPDIASTRUCT	
TxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_TX_UDP	
RxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_RX_UDP	
NetVarManager_UDP_MotionTask_0	NETVARMANAGER_UDP_FB	
NetVarManager_UDP_MotionTask_1	NETVARMANAGER_UDP_FB	
UDP_XMOut_ctrlXIn	DUT_XMOut_ctrlXIn	
wW01XMOut_ctrlXIn	WORD	1311
wW02XMOut_ctrlXIn	WORD	2323
wW03XMOut_ctrlXIn	WORD	0
wW04XMOut_ctrlXIn	WORD	0
wW05XMOut_ctrlXIn	WORD	0
wW06XMOut_ctrlXIn	WORD	0
wW07XMOut_ctrlXIn	WORD	0

ctrlX Core

C:\Traspaso\ESCRIPTORI_LIMPIEZA\VideosProgramas\XM_ctrlX_TCPCommunication\ctrlX\PLC_ComNetworkVar\ctrlX_ComNetworkVar_PLC.project* - ctrlX PLC Engineering

File Edit View Project Build Online Debug Tools Window Help

Devices

- ctrlX_ComNetworkVar_PLC
 - Device [192.168.1.1]
 - PLC Logic
 - Application [run]
 - DUT_ctrlXOut_XMin (STRUCT)
 - DUT_XMOut_ctrlXIn (STRUCT)
 - CheckFunctions
 - NVL_UDP_ctrlXOut_XMin
 - Library Manager
 - NVL_UDP_XMOut_ctrlXIn
 - PLC_PRG (PRG)

Device.Application.NVL_UDP_XMOut_ctrlXIn

Expression	Type	Value	Prepared value
NetVarManager_UDP_MainTask_1	NETVARMANAGER_UDP_FB		
UDP_XMOut_ctrlXIn	DUT_XMOut_ctrlXIn		
wW01XMOut_ctrlXIn	WORD	1311	1311
wW02XMOut_ctrlXIn	WORD	2323	2323
wW03XMOut_ctrlXIn	WORD	0	
wW04XMOut_ctrlXIn	WORD	0	
wW05XMOut_ctrlXIn	WORD	0	

ctrlX - Example communication with Network Variables

XM

The screenshot shows the IndraWorks Engineering interface. The Project Explorer on the left displays a tree structure for 'MLC14_Safety_NetworkVar_2023V00.xiwp'. The main window shows the 'IndraMotionMlc1.Application.NVL_UDP_ctrlXOut_XMIn' table with the following data:

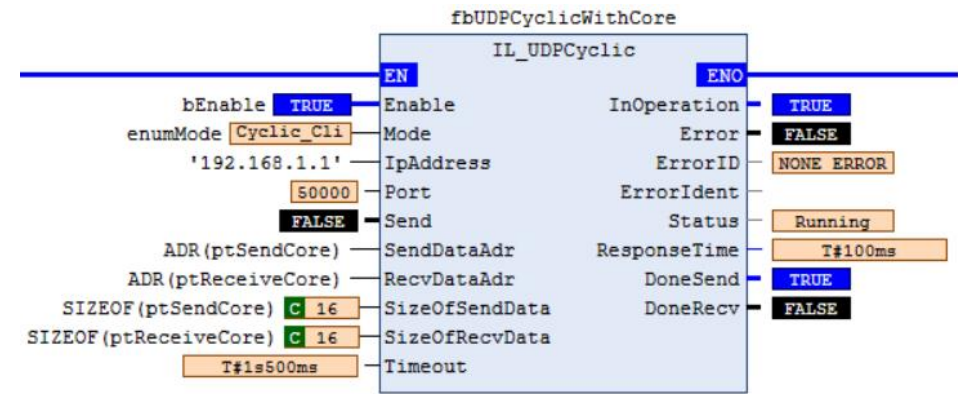
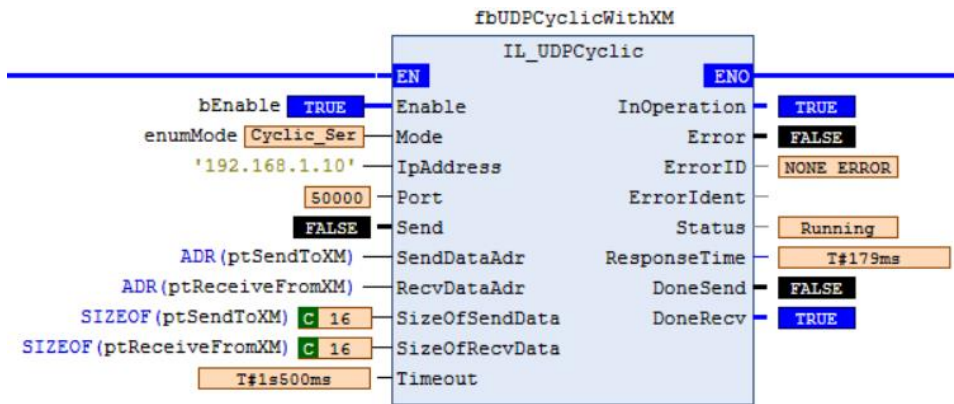
Expression	Type	Value
RxPDOs_UDP	ARRAY [0..0] OF NETVARPDO_RX_UDP	
NetVarManager_UDP_MotionTask_0	NETVARMANAGER_UDP_FB	
NetVarManager_UDP_MotionTask_1	NETVARMANAGER_UDP_FB	
UDP_ctrlXOut_XMIn	DUT_ctrlXOut_XMIn	
wW01ctrlXOut_XMIn	WORD	345
wW02ctrlXOut_XMIn	WORD	152
wW03ctrlXOut_XMIn	WORD	789
wW04ctrlXOut_XMIn	WORD	45
wW05ctrlXOut_XMIn	WORD	546
wW06ctrlXOut_XMIn	WORD	121
wW07ctrlXOut_XMIn	WORD	0
wW08ctrlXOut_XMIn	WORD	0

ctrlX Core

The screenshot shows the ctrlX Core interface. The 'Device.Application.NVL_UDP_ctrlXOut_XMIn' table displays the same data as the IndraWorks screenshot. Red arrows point from the values in the 'Value' column to a list of values on the right side of the image:

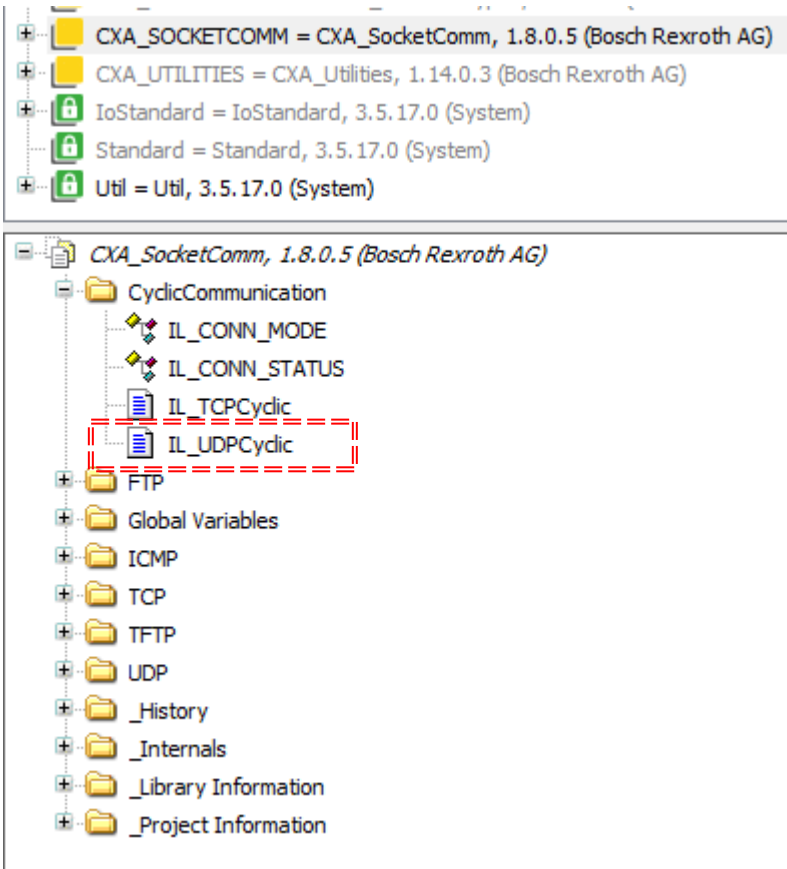
- 345
- 152
- 789
- 45
- 546
- 121

Communication with IL_UDPCyclic

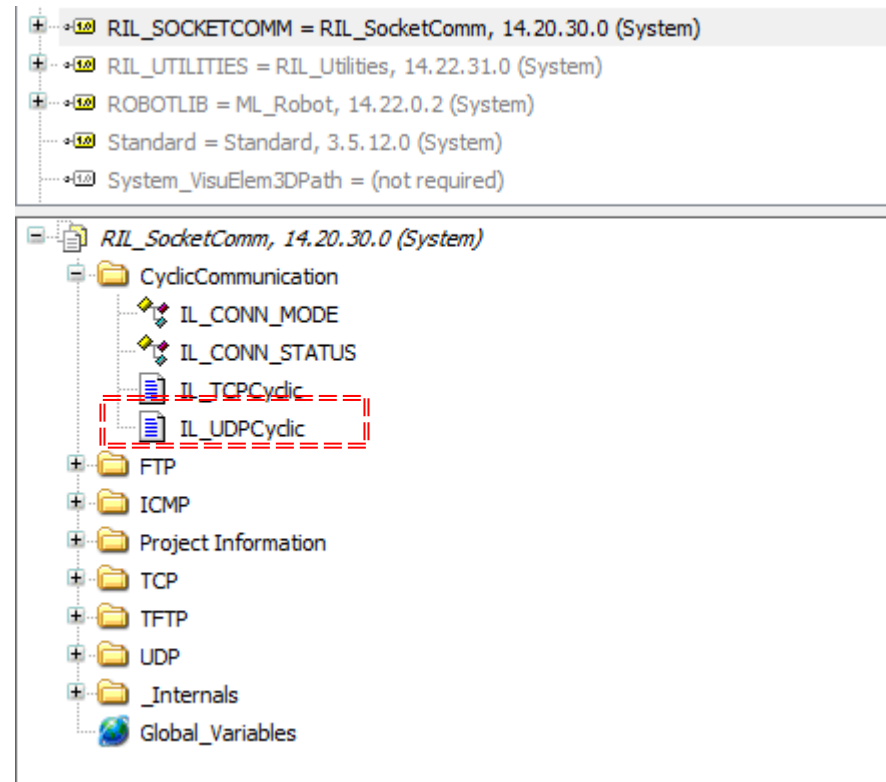


- The control module for UDP communication is the same as we have previously used for TCP communication
 - In ctrlX Core or ctrlXDrive with Core, the library will be CXA_SOCKETCOMM
 - In XM the library will be RIL_SOCKETCOMM

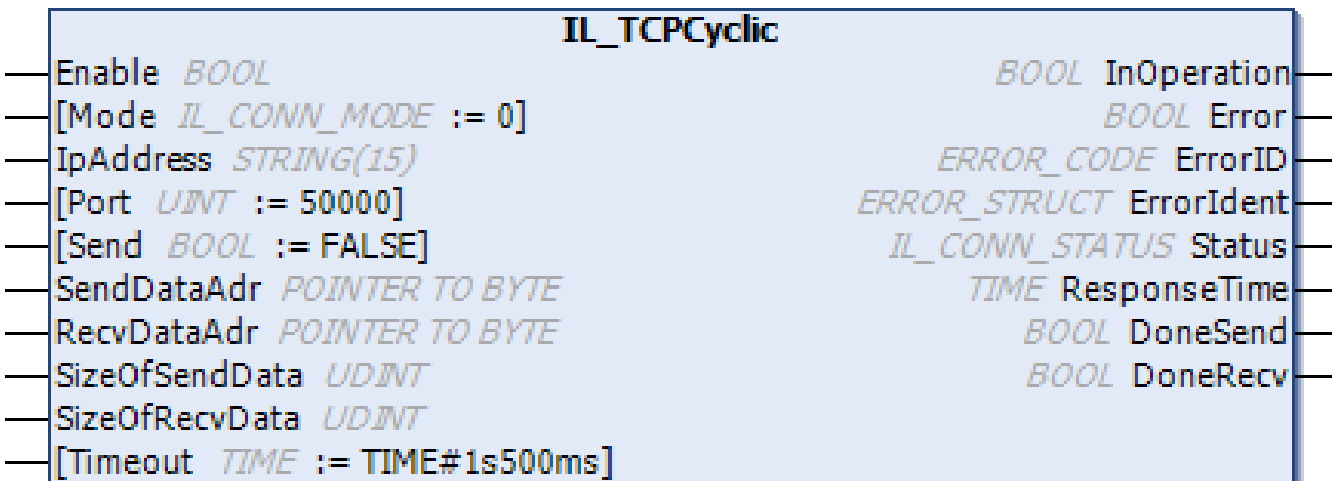
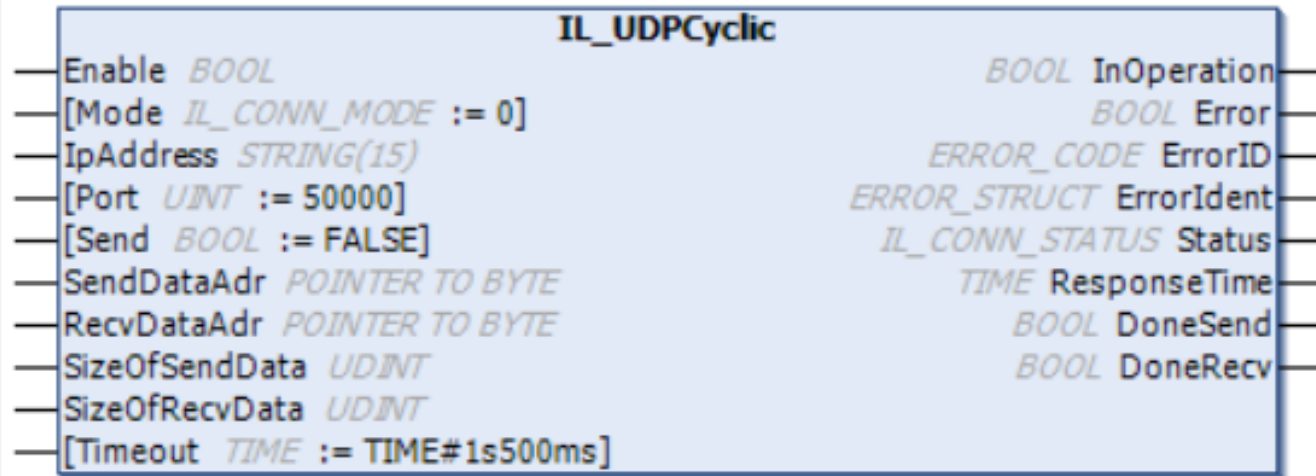
ctrlX Core



XM



- As can be seen in the view of both modules, these are identical:



The example used for the control of communication in TCP mode, serves us practically if changes to be used with UDP, modifying that if the instances of the modules used to not generate problems

- Modification in the ctrlX Core, of the name for the control block using IL_UDPCyclic

ctrlX Core

The image displays the SIMATIC Manager interface for the `POU_CommUDPCyclic` program. On the left, the LAD editor shows the program code with a red dashed box highlighting the line `fbUDPCyclicWithXM: IL_UDPCyclic;`. A red arrow points from this line to the right, where a detailed parameter configuration for the `fbUDPCyclicWithXM` block is shown. The configuration is organized into two columns: `EN` (Enable) and `ENO` (InOperation). The `IL_UDPCyclic` block name is also highlighted with a red dashed box.

Parameter	Value	EN	ENO
bEnable	TRUE	Enable	InOperation
enumMode	Cyclic_Ser	Mode	Error
IPAddress	'192.168.1.10'	IpAddress	ErrorID
Port	50000	Port	ErrorIdent
Send	FALSE	Send	Status
SendDataAdr	ADR (ptSendToXM)	SendDataAdr	ResponseTime
RecvDataAdr	ADR (ptReceiveFromXM)	RecvDataAdr	DoneSend
SizeOfSendData	SIZEOF (ptSendToXM) C 16	SizeOfSendData	DoneRecv
SizeOfRecvData	SIZEOF (ptReceiveFromXM) C 16	SizeOfRecvData	DoneRecv
Timeout	T#1s500ms	Timeout	

- Modification in the XM, of the name for the control block using IL_UDPCyclic

XM

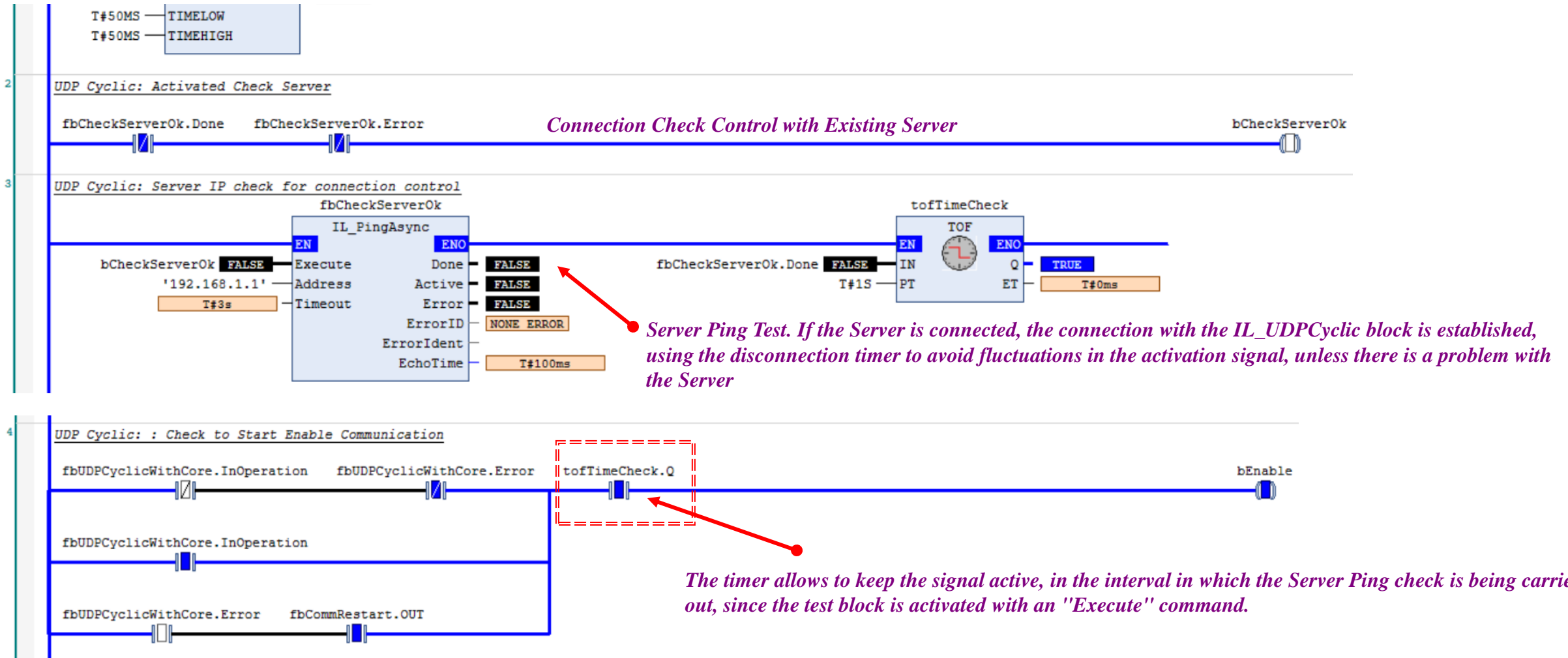
- POU_CommUDPCyclic (PRG)
- Symbol Configuration

```
1 PROGRAM POU_CommUDPCyclic
2 VAR
3 // VARIABLES
4 // Communication Enable
5   bEnable          : BOOL;
6 // Structures For Receive And Send Data To ctrlx Core
7   ptSendCore       : DUT_SendToCore;
8   ptReceiveCore    : DUT_ReceiveFromCore;
9 // Enumerator For Communication Define Type
10  enumMode          : IL_CONN_MODE;
11 // Blinking Signal For Communication Start
12  fbCommRestart     : Blink;
13 // Module for Communication Control
14  fbUDPCyclicWithCore : IL_UDPCyclic;
15
16 // Server IP check for connection control (UDP Connection)
17  fbCheckServerOk : IL_PingAsync;
18  bCheckServerOk  : BOOL;
19  tofTimeCheck    : TOF;
20 END_VAR
21
```

Parameter	Value	Terminal	Value	
bEnable	TRUE	EN	InOperation	TRUE
enumMode	Cyclic_Cli	Mode	Error	FALSE
IPAddress	'192.168.1.1'	IpAddress	ErrorID	NONE ERROR
Port	50000	Port	ErrorIdent	NONE ERROR
Send	FALSE	Send	Status	Running
SendDataAdr	ADR(ptSendCore)	SendDataAdr	ResponseTime	T#100ms
RecvDataAdr	ADR(ptReceiveCore)	RecvDataAdr	DoneSend	TRUE
SizeOfSendData	C 16	SizeOfSendData	DoneRecv	FALSE
SizeOfRecvData	C 16	SizeOfRecvData		
Timeout	T#1s500ms	Timeout		

! New variables, which must be added for block control using IL_UDPCyclic

- To guarantee the operation of the system in the XM part (Client) we must make a small modification in the boot control of the communications block



- Additional notes

- Each of the systems has its peculiarities*
- UDP is usually faster in terms of communication speed*
- TCP is more reliable as data is delivered correctly*
- UDP can deliver incomplete data*
- Using Network Variables (UDP) you can enable controls such as the transmission of the "Checksum" or the "Acknowledgement" of the data sent.*
- If we are going to use any of these systems, we must be clear about the one that best suits what we want to do.*

Thank you for your attention

