

ctrlX - CORE

- **NodeRed – Part 02**

- Communicate XM with NodeRed in ctrlX Core

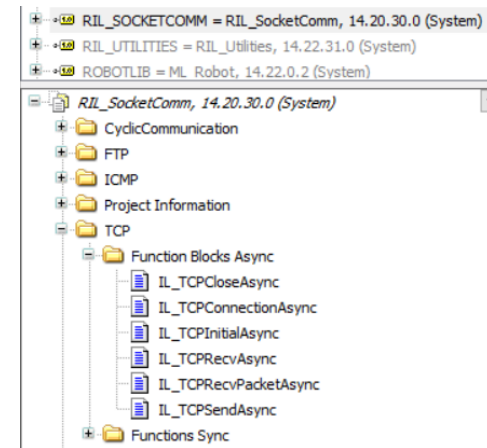
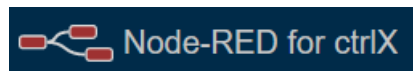
Jordi Laboria (DCET/SLF4-ES)

rexroth
A Bosch Company



GOALS:

- Data Injection Example (Visualization and Formatting)
- Data reading in NodeRed (ctrlX Core)
- Send Data from the NodeRed of the ctrlX Core and read them in the XM



Data Injection

Example

(Visualization and Formatting)

- Before we start manipulating the data coming from the XM, let's see how the management of the data sent and received by communications works. These data, in our case, will be received in buffer format, which means that we will see them all grouped and then we will have to manipulate them to separate them.

- In order to understand the operation and before activating the communications we will see how we generate a Buffer to be able to simulate the data that we will receive and that of course we can send to the XM from or to the NodeRed

The "inject" function will allow us to generate the test buffer

We will modify the name by entering the properties of the function

Select the buffer option

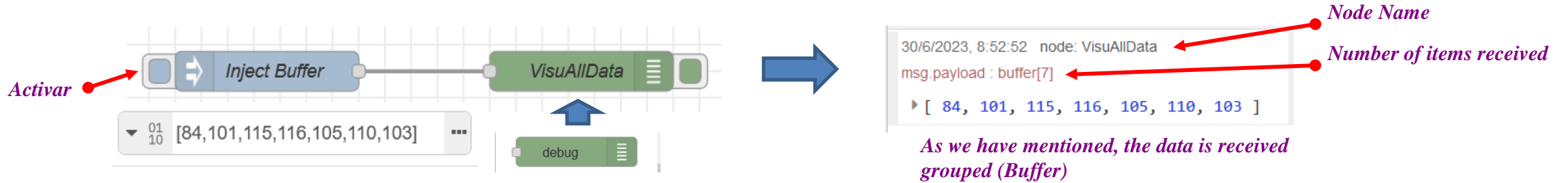
And we create the 7 elements that will form the buffer as shown in the image



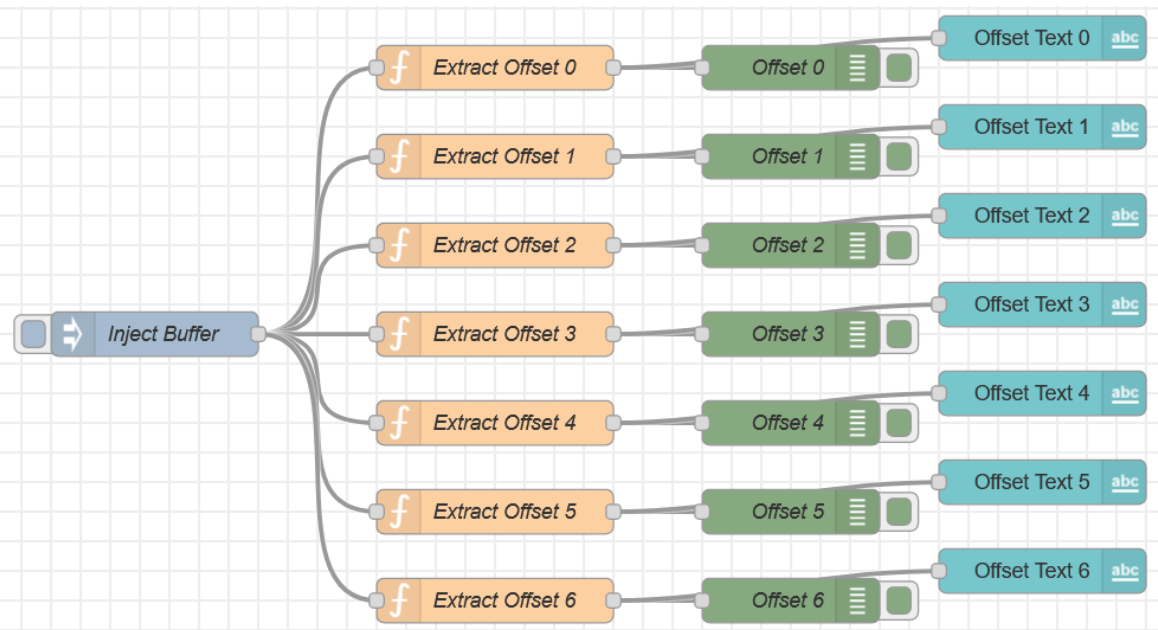
Do not forget that the modifications made must be activated with "Deploy"

Node-RED for ctrlX Deploy  

- If we then add a "debug" function and activate the "Inject Buffer" we will see the result we receive



- To see how the system works and "separate" the 7 elements of the Buffer received, we will use the following example



Debug

```

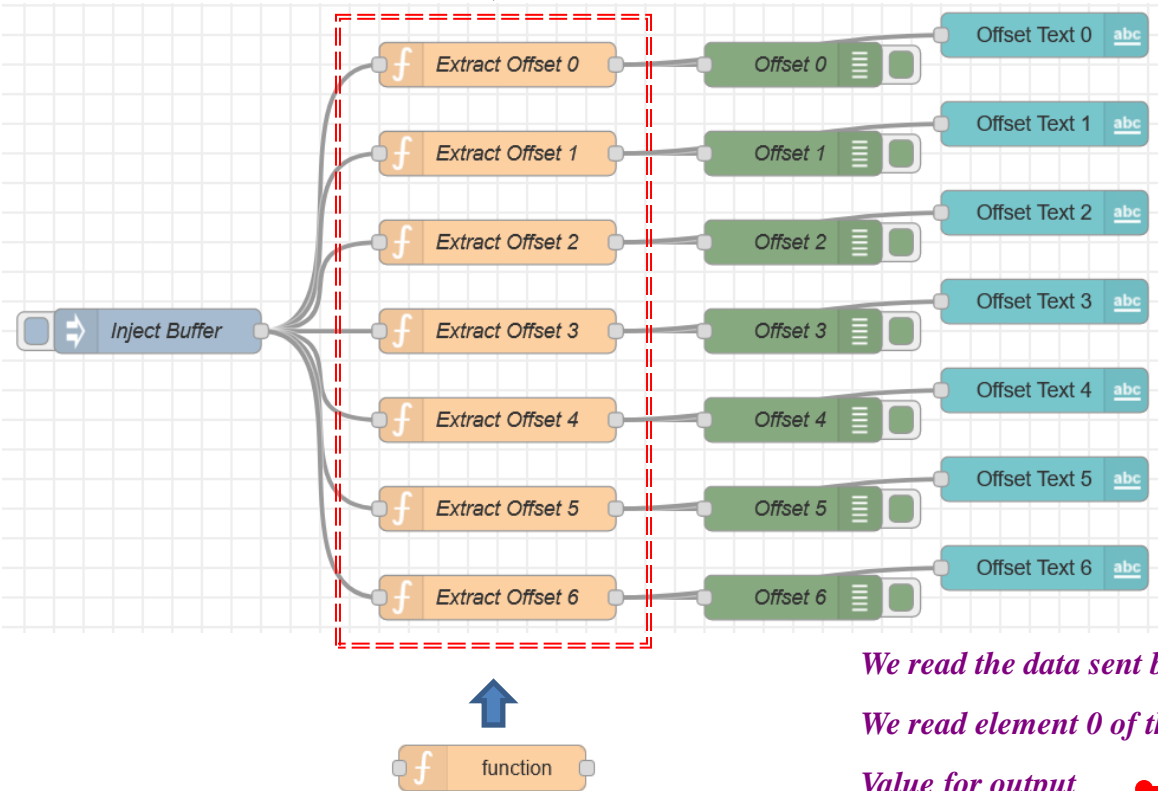
30/6/2023, 9:06:02 node: Offset 0
msg.payload : number
84
30/6/2023, 9:06:02 node: Offset 1
msg.payload : number
101
30/6/2023, 9:06:02 node: Offset 2
msg.payload : number
115
30/6/2023, 9:06:02 node: Offset 3
msg.payload : number
116
30/6/2023, 9:06:02 node: Offset 4
msg.payload : number
105
30/6/2023, 9:06:02 node: Offset 5
msg.payload : number
110
30/6/2023, 9:06:02 node: Offset 6
msg.payload : number
103
    
```

Visualization on the DashBoard

Offset Text 0	84
Offset Text 1	101
Offset Text 2	115
Offset Text 3	116
Offset Text 4	105
Offset Text 5	110
Offset Text 6	103

- The "inject Buffer" we have already seen how it should be configured, therefore, the next step is to configure the extraction functions of the different elements of the buffer received

Functions for extracting data from the buffer



To extract the data from the buffer individually we will do it as follows

Edit function node

Delete Cancel Done

Properties

Name Extract Offset 0

Setup On Start On Message On Stop

```

1 var ReadBuffer = msg.payload;
2 var value = ReadBuffer.readUInt8(0);
3 msg.payload = value;
4 return msg;

```

We read the data sent by the previous instruction → `var ReadBuffer = msg.payload;`
 We read element 0 of the Buffer → `var value = ReadBuffer.readUInt8(0);`
 Value for output → `msg.payload = value;`
 Message to the next statement → `return msg;`

- The extraction of the buffer can be done in different ways depending on the data received or sent. In the case of the example we are using the byte option to visualize and know how to extract this data so that the process is understood.

Debug

```
30/6/2023, 9:06:02 node: Offset 0  
msg.payload : number  
84  
30/6/2023, 9:06:02 node: Offset 1  
msg.payload : number  
101  
30/6/2023, 9:06:02 node: Offset 2  
msg.payload : number  
115  
30/6/2023, 9:06:02 node: Offset 3  
msg.payload : number  
116  
30/6/2023, 9:06:02 node: Offset 4  
msg.payload : number  
105  
30/6/2023, 9:06:02 node: Offset 5  
msg.payload : number  
110  
30/6/2023, 9:06:02 node: Offset 6  
msg.payload : number  
103
```

Code

```
var ReadBuffer = msg.payload;  
ar value = ReadBuffer.readUInt8(0);  
sg.payload = value;  
return msg;  
  
var ReadBuffer = msg.payload;  
var value = ReadBuffer.readUInt8(2);  
msg.payload = value;  
return msg;  
  
var ReadBuffer = msg.payload;  
var value = ReadBuffer.readUInt8(4);  
msg.payload = value;  
return msg;  
  
var ReadBuffer = msg.payload;  
var value = ReadBuffer.readUInt8(6);  
msg.payload = value;  
return msg;
```

DashBoard

Offset Text 0	84
Offset Text 1	101
Offset Text 2	115
Offset Text 3	116
Offset Text 4	105
Offset Text 5	110
Offset Text 6	103

```
30/6/2023, 8:52:52 node: VisuAllData  
msg.payload : buffer [7]  
▶ [ 84, 101, 115, 116, 105, 110, 103 ]
```

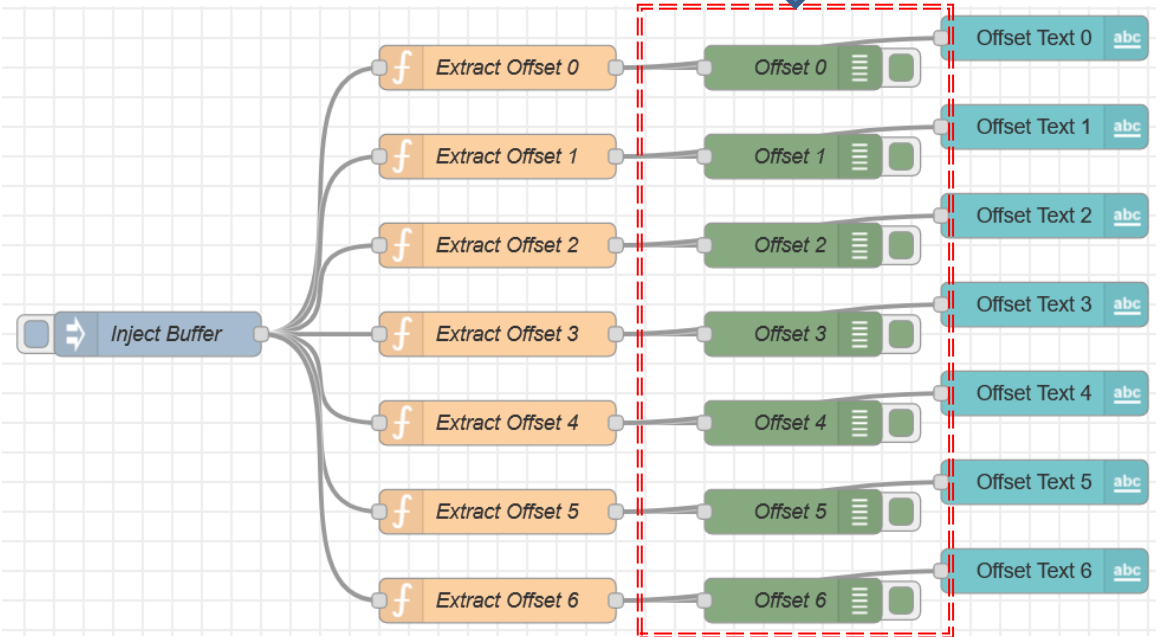
Offset 0 1 2 3 4 5 6



The data received or sent in NodeRed, are always "manipulated" in byte format, as in the example, however, in many cases, these data will be integer values or even in real format. The way to treat this data we will see later

- The next group of instructions are the ones we will use to visualize in the debug. It is always important to have the help of this element to be able to see what we are receiving or if the data is of the type we want to use.

Functions for data visualization in the "debug"

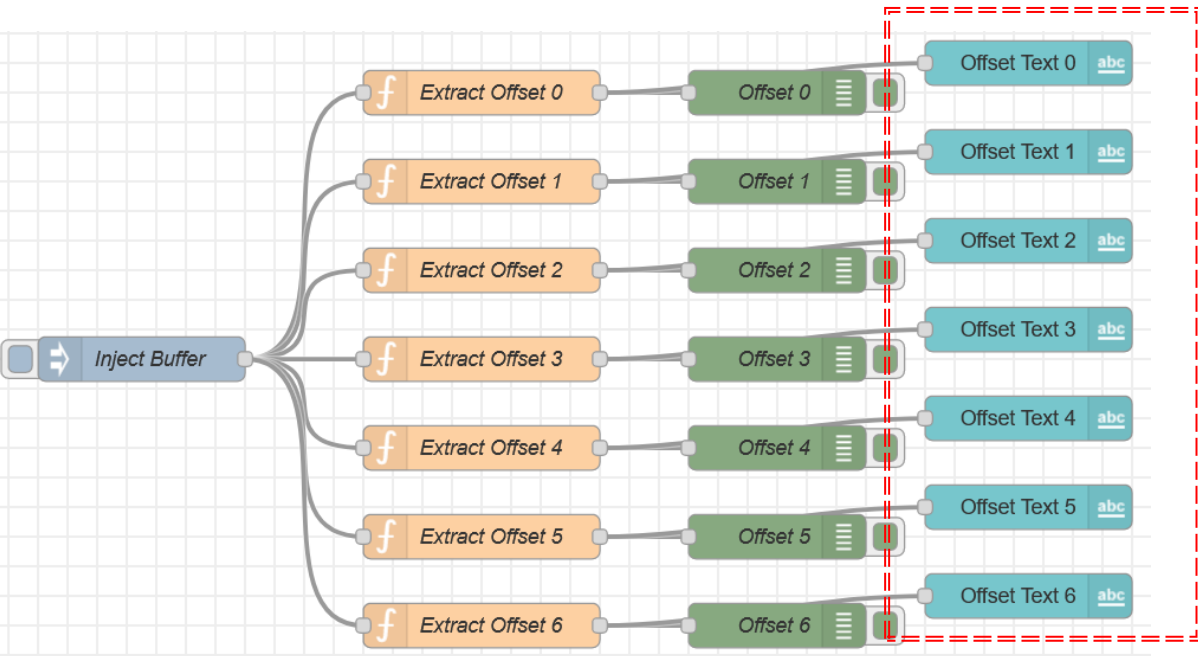


In the debug functions and given that the message received is of the type msg.payload the only thing that we will have to modify in each of them is the name

The screenshot shows the 'Edit debug node' configuration panel. It has buttons for 'Delete', 'Cancel', and 'Done'. Below these are 'Properties' and 'To' sections. The 'Output' dropdown is set to 'msg. payload' and the 'Name' field is 'Offset 0'. The 'To' section has 'debug window' checked, 'system console' unchecked, and 'node status (32 characters)' unchecked. Red dashed boxes highlight the 'Output' and 'Name' fields, with blue arrows pointing to them from the text above.

- The last function of this part of the example is the visualization in the "DashBoard"

Functions for data visualization in the "DashBoard"



Visualization on the DashBoard

Offset Text 0	84
Offset Text 1	101
Offset Text 2	115
Offset Text 3	116
Offset Text 4	105
Offset Text 5	110
Offset Text 6	103

In the "text" functions and since the message received is of the type `msg.payload` the only thing that we must modify in each of them is the name



Edit text node

Delete Cancel **Done**

Properties

Group: [ctrlX Core] PLC_Data

Size: auto

Label: Offset Text 0

Value format: {{msg.payload}}

Layout: label value | label value | label value

Class: Optional CSS class name(s) for widget

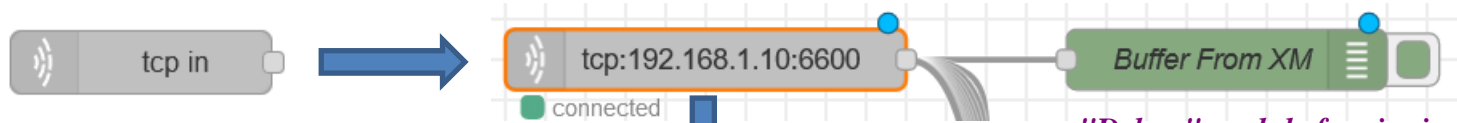
Name:

Enabled

Reading data in the Node Red (ctrlX Core)

- At this point we will see how to establish communication with the XM and send and receive data using the NodeRed of the ctrlX Core

To establish communication with the XM we will use the function "tcp in"



"Debug" module for viewing the received buffer

! IP address of the computer with which we want to communicate

! Choose the "Stream of" and "Buffer" option

Edit tcp in node

Delete Cancel Done

⚙ Properties

Type Connect to port 6600

at host 192.168.1.10

Enable secure (SSL/TLS) connection

Output stream of Buffer payload(s)

```

30/6/2023, 13:01:18 node: TimeComWithXM
192.168.1.10 : msg.payload : number
0.328

30/6/2023, 13:01:18 node: Buffer From XM
192.168.1.10 : msg.payload : buffer[20]
▶ [ 71, 15, 0, 0, 0, 0, 0, 0, 0, 0 ... ]

30/6/2023, 13:01:19 node: TimeComWithXM
192.168.1.10 : msg.payload : number
0.341

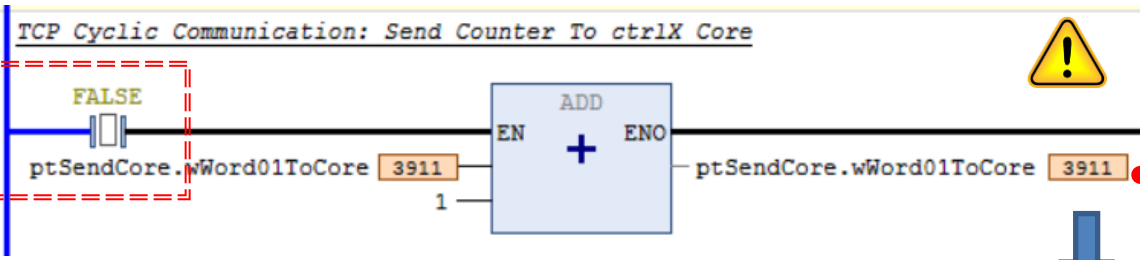
30/6/2023, 13:01:19 node: Buffer From XM
192.168.1.10 : msg.payload : buffer[20]
▶ [ 71, 15, 0, 0, 0, 0, 0, 0, 0, 0 ... ]

30/6/2023, 13:01:19 node: TimeComWithXM
192.168.1.10 : msg.payload : number
0.745

30/6/2023, 13:01:19 node: Buffer From XM
192.168.1.10 : msg.payload : buffer[20]
▶ [ 71, 15, 0, 0, 0, 0, 0, 0, 0, 0 ... ]
    
```

The value sent by the first word of the communications from the XM arrives at the NodeRed as we have commented in buffer mode therefore in two bytes. We have placed the contact FALSE to be able to visualize the value arrived in a fixed way and to be able to see how it arrives to us.

! As you can see the value comes to us with the order of the buffer changed



3911	DEC	15	71
	BIN	0000 1111	0100 0111

- Although we have not yet commented from the XM the structure sent is as follows. Therefore what is being sent is a total of 10 words

```

TYPE DUT_SendToCore :
STRUCT
  wWord01ToCore: WORD;
  wWord02ToCore: WORD;
  wWord03ToCore: WORD;
  wWord04ToCore: WORD;
  wWord05ToCore: WORD;
  wWord06ToCore: WORD;
  wWord07ToCore: WORD;
  wWord08ToCore: WORD;
  wWord09ToCore: WORD;
  wWord10ToCore: WORD;
END_STRUCT
END_TYPE
    
```

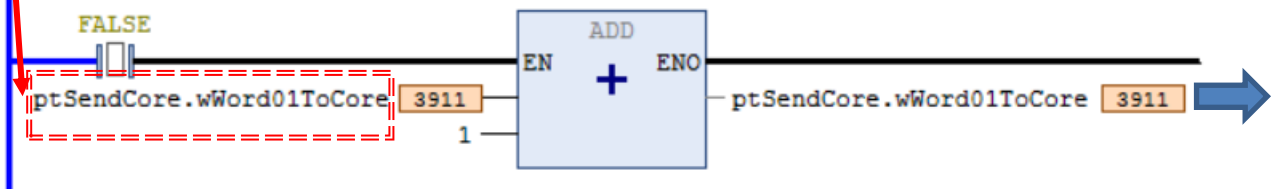
! However, and without any type of treatment, the data received in the NodeRed appear in a Buffer of 20 Bytes (10 Words)

```

3/7/2023, 8:29:16 node: Buffer From XM
192.168.1.10 : msg.payload : buffer[20]
    
```

! Hexadecimal value viewer

TCP Cyclic Communication: Send Counter To ctrlX Core



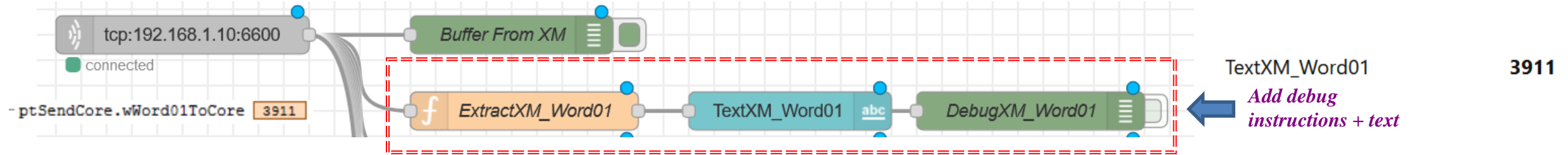
3911

	DEC	15	71
	BIN	0000 1111	0100 0111

- Therefore this part comes to us with a value that we can not use directly and we must clear the value using a function as we have already seen previously



- To extract the first Word and correctly visualize the result we will use the function with the following instructions:



OK

NOK

Reading the buffer with .readInt16LE

```
var ReadBuffer = msg.payload;
var value = ReadBuffer.readInt16LE(0);
msg.payload = value;
return msg;
```

Reading the buffer with .readInt16BE

```
1 var ReadBuffer = msg.payload;
2 var value = ReadBuffer.readInt16BE(0);
3 msg.payload = value;
4 return msg;
```

The result is correct and the value we are sending from the XM appears

```
3/7/2023, 8:49:35 node: DebugXM_Word01
192.168.1.10 : msg.payload : number
3911
```

3911	DEC	15	71
	BIN	0000 1111	0100 0111

The result is NOT correct and the value appears with the order of the bytes changed

```
3/7/2023, 8:51:13 node: DebugXM_Word01
192.168.1.10 : msg.payload : number
18191
```

18191	DEC	71	15
	BIN	0100 0111	0000 1111

It is important to check the correct reception of the values received

- In this list we have the different methods for the conversion of the values (Read, Write, To String)

- `buf.readInt64BE([offset])`
- `buf.readInt64LE([offset])`
- `buf.readUInt64BE([offset])`
- `buf.readUInt64LE([offset])`
- `buf.readDoubleBE([offset])`
- `buf.readDoubleLE([offset])`
- `buf.readFloatBE([offset])`
- `buf.readFloatLE([offset])`
- `buf.readInt8([offset])`
- `buf.readInt16BE([offset])`
- `buf.readInt16LE([offset])`
- `buf.readInt32BE([offset])`
- `buf.readInt32LE([offset])`
- `buf.readIntBE(offset, byteLength)`
- `buf.readIntLE(offset, byteLength)`
- `buf.readUInt8([offset])`
- `buf.readUInt16BE([offset])`
- `buf.readUInt16LE([offset])`
- `buf.readUInt32BE([offset])`
- `buf.readUInt32LE([offset])`
- `buf.readUIntBE(offset, byteLength)`
- `buf.readUIntLE(offset, byteLength)`
- `buf.write(string[, offset[, length]][, encoding])`
- `buf.writeBigInt64BE(value[, offset])`
- `buf.writeBigInt64LE(value[, offset])`
- `buf.writeBigUInt64BE(value[, offset])`
- `buf.writeBigUInt64LE(value[, offset])`
- `buf.writeDoubleBE(value[, offset])`
- `buf.writeDoubleLE(value[, offset])`
- `buf.writeFloatBE(value[, offset])`
- `buf.writeFloatLE(value[, offset])`
- `buf.writeInt8(value[, offset])`
- `buf.writeInt16BE(value[, offset])`
- `buf.writeInt16LE(value[, offset])`
- `buf.writeInt32BE(value[, offset])`
- `buf.writeInt32LE(value[, offset])`
- `buf.writeIntBE(value, offset, byteLength)`
- `buf.writeIntLE(value, offset, byteLength)`
- `buf.writeUInt8(value[, offset])`
- `buf.writeUInt16BE(value[, offset])`
- `buf.writeUInt16LE(value[, offset])`
- `buf.writeUInt32BE(value[, offset])`
- `buf.writeUInt32LE(value[, offset])`
- `buf.writeUIntBE(value, offset, byteLength)`
- `buf.writeUIntLE(value, offset, byteLength)`
- `buf.toString([encoding[, start[, end]])`



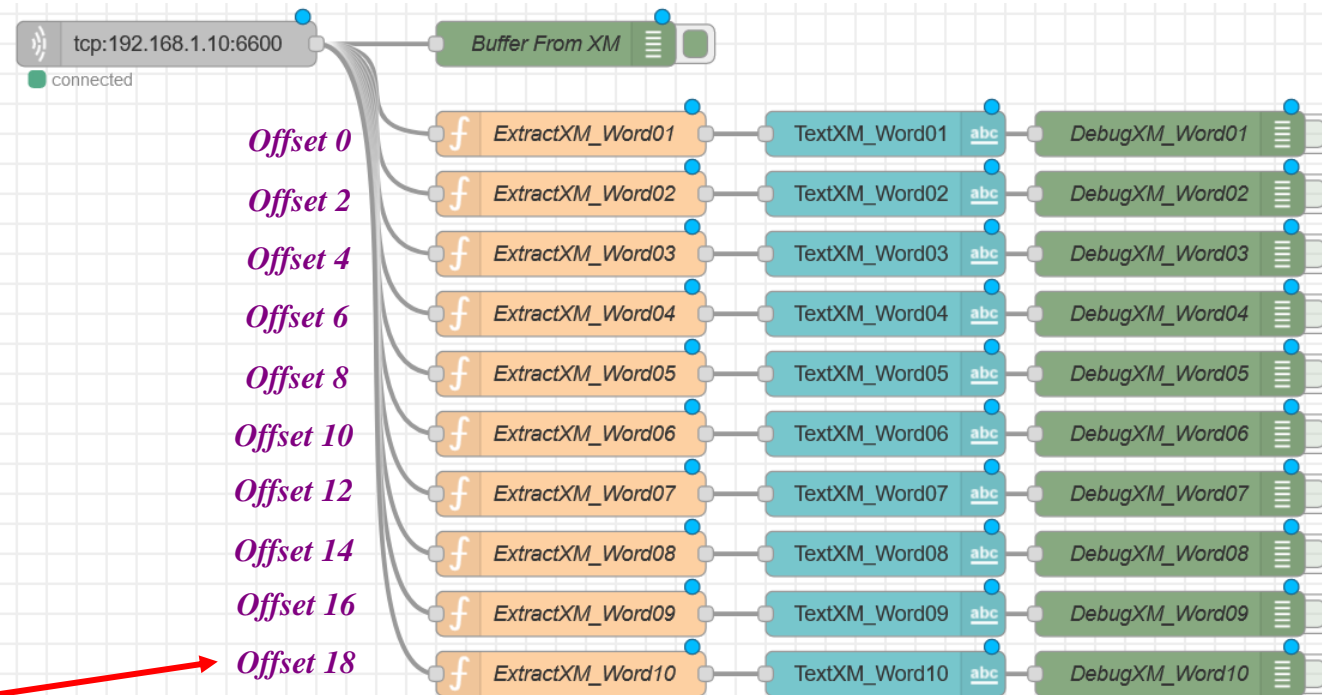
Methods used in the previous example

- Obviously, if we want to receive all the data we must include in the frame, the structures of functions of each of the elements

ptSendCore	DUT_SendToCore	
wWord01ToCore	WORD	3911
wWord02ToCore	WORD	2523
wWord03ToCore	WORD	3450
wWord04ToCore	WORD	4734
wWord05ToCore	WORD	5212
wWord06ToCore	WORD	6453
wWord07ToCore	WORD	7865
wWord08ToCore	WORD	8234
wWord09ToCore	WORD	9123
wWord10ToCore	WORD	10234

Values sent by XM

NodeRed with the relevant processing of the data received



Values received in the Dashboard

TextXM_Word01	3911
TextXM_Word02	2523
TextXM_Word03	3450
TextXM_Word04	4734
TextXM_Word05	5212
TextXM_Word06	6453
TextXM_Word07	7865
TextXM_Word08	8234
TextXM_Word09	9123
TextXM_Word10	10234

In the function module we can modify the number of output points if necessary



```
var ReadBuffer = msg.payload;
var value = ReadBuffer.readUInt16LE(0);
msg.payload = value;
return msg;
```

Offset number corresponding to the buffer

Sending data from the Node Red (ctrlX Core) to the XM

- The next step is to send data to the XM

We insert a function of type "inject" that we will use to generate a data buffer

Edit inject node

Delete Cancel Done

Properties

Name: Inject Buffer

msg.payload = [30,1,115,1,116,0,10,0,110,0,103,0,104,0,105,0,106,0,107,0,108,0,109,0,110,0,111,0,112,0,113,0,114,0,115,0,116,0,117,0,118,0,119,0,120,0,121,0,122,0,123,0,124,0,125,0,126,0,127,0,128,0,129,0,130,0,131,0,132,0,133,0,134,0,135,0,136,0,137,0,138,0,139,0,140,0,141,0,142,0,143,0,144,0,145,0,146,0,147,0,148,0,149,0,150,0,151,0,152,0,153,0,154,0,155,0,156,0,157,0,158,0,159,0,160,0,161,0,162,0,163,0,164,0,165,0,166,0,167,0,168,0,169,0,170,0,171,0,172,0,173,0,174,0,175,0,176,0,177,0,178,0,179,0,180,0,181,0,182,0,183,0,184,0,185,0,186,0,187,0,188,0,189,0,190,0,191,0,192,0,193,0,194,0,195,0,196,0,197,0,198,0,199,0,200,0,201,0,202,0,203,0,204,0,205,0,206,0,207,0,208,0,209,0,210,0,211,0,212,0,213,0,214,0,215,0,216,0,217,0,218,0,219,0,220,0,221,0,222,0,223,0,224,0,225,0,226,0,227,0,228,0,229,0,230,0,231,0,232,0,233,0,234,0,235,0,236,0,237,0,238,0,239,0,240,0,241,0,242,0,243,0,244,0,245,0,246,0,247,0,248,0,249,0,250,0,251,0,252,0,253,0,254,0,255]

Buffer editor

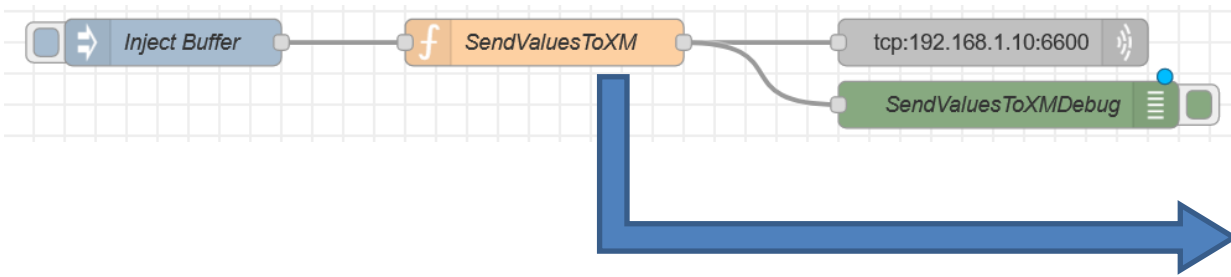
Handle as JSON array

1	[30,1,115,1,116,0,10,0,110,0,103,0,104,0,105,0,106,0,107,0,108,0,109,0,110,0,111,0,112,0,113,0,114,0,115,0,116,0,117,0,118,0,119,0,120,0,121,0,122,0,123,0,124,0,125,0,126,0,127,0,128,0,129,0,130,0,131,0,132,0,133,0,134,0,135,0,136,0,137,0,138,0,139,0,140,0,141,0,142,0,143,0,144,0,145,0,146,0,147,0,148,0,149,0,150,0,151,0,152,0,153,0,154,0,155,0,156,0,157,0,158,0,159,0,160,0,161,0,162,0,163,0,164,0,165,0,166,0,167,0,168,0,169,0,170,0,171,0,172,0,173,0,174,0,175,0,176,0,177,0,178,0,179,0,180,0,181,0,182,0,183,0,184,0,185,0,186,0,187,0,188,0,189,0,190,0,191,0,192,0,193,0,194,0,195,0,196,0,197,0,198,0,199,0,200,0,201,0,202,0,203,0,204,0,205,0,206,0,207,0,208,0,209,0,210,0,211,0,212,0,213,0,214,0,215,0,216,0,217,0,218,0,219,0,220,0,221,0,222,0,223,0,224,0,225,0,226,0,227,0,228,0,229,0,230,0,231,0,232,0,233,0,234,0,235,0,236,0,237,0,238,0,239,0,240,0,241,0,242,0,243,0,244,0,245,0,246,0,247,0,248,0,249,0,250,0,251,0,252,0,253,0,254,0,255]
---	---

1 1E 01 73 01 74 00 0A 00 6E 00 67 00 68 00 69 00
2 6A 00 6B 01

Remember that the correct order of sending is reversed and that therefore we must "prepare" it for sending to the XM

- To convert the generated buffer data to the correct values in its visualization in the XM we must implement a new function between the buffer and the data sent to perform the conversion.



```
1  var test1 = msg.payload;
2  var valueWord01 = test1.readUInt16LE(0);
3  var valueWord02 = test1.readUInt16LE(2);
4  var valueWord03 = test1.readUInt16LE(4);
5  var valueWord04 = test1.readUInt16LE(6);
6  var valueWord05 = test1.readUInt16LE(8);
7  var valueWord06 = test1.readUInt16LE(10);
8  var valueWord07 = test1.readUInt16LE(12);
9  var valueWord08 = test1.readUInt16LE(14);
10 var valueWord09 = test1.readUInt16LE(16);
11 var valueWord10 = test1.readUInt16LE(18);
12 // Write Values
13 var testing = msg.payload;
14 testing.writeUInt16LE(valueWord01,0);
15 testing.writeUInt16LE(valueWord02, 2);
16 testing.writeUInt16LE(valueWord03, 4);
17 testing.writeUInt16LE(valueWord04, 6);
18 testing.writeUInt16LE(valueWord05, 8);
19 testing.writeUInt16LE(valueWord06, 10);
20 testing.writeUInt16LE(valueWord07, 12);
21 testing.writeUInt16LE(valueWord08, 14);
22 testing.writeUInt16LE(valueWord09, 16);
23 testing.writeUInt16LE(valueWord10, 18);
24 msg.payload = testing;
25 return msg;
26
```

Reading the "injected" values"

The auxiliary values are extracted, in LE format (following the example of the data received), arranging them on the auxiliary variables, once the value has been extracted from the offset that the Word that we want to send must generate.

Reading of the "injected" values that we will modify for writing

The offset values read earlier are now written to the temporary buffer "testing" and this variable is the one that is sent to the XM

Variable that is sent to the XM

- The treated values are sent to the XM as follows

JSON with the data sent from the "injector"

[30,1,115,1,116,0,10,0,110,0,103,0,104,0,105,0,106,0,107,1]

[30,1,

286	DEC	1	30
	BIN	0000 0001	0001 1110

Data received at XM

ptReceiveCore		DUT_ReceiveFromCore	
◆	wWord01FromCore	WORD	286
◆	wWord02FromCore	WORD	371
◆	wWord03FromCore	WORD	116
◆	wWord04FromCore	WORD	10
◆	wWord05FromCore	WORD	110
◆	wWord06FromCore	WORD	103
◆	wWord07FromCore	WORD	104
◆	wWord08FromCore	WORD	105
◆	wWord09FromCore	WORD	106
◆	wWord10FromCore	WORD	363

Offset 0

```
14 testing.writeUInt16LE(valueWord01, 0);
```

115,1,

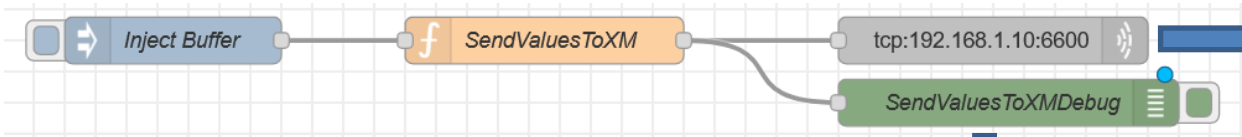
371	DEC	1	115
	BIN	0000 0001	01110011

ptReceiveCore		DUT_ReceiveFromCore	
◆	wWord01FromCore	WORD	286
◆	wWord02FromCore	WORD	371
◆	wWord03FromCore	WORD	116
◆	wWord04FromCore	WORD	10
◆	wWord05FromCore	WORD	110
◆	wWord06FromCore	WORD	103
◆	wWord07FromCore	WORD	104
◆	wWord08FromCore	WORD	105
◆	wWord09FromCore	WORD	106
◆	wWord10FromCore	WORD	363

Offset 2

```
15 testing.writeUInt16LE(valueWord02, 2);
```

- The next two blocks allow us to send the data to the XM on the one hand and visualize them in the "Debug" on the other.



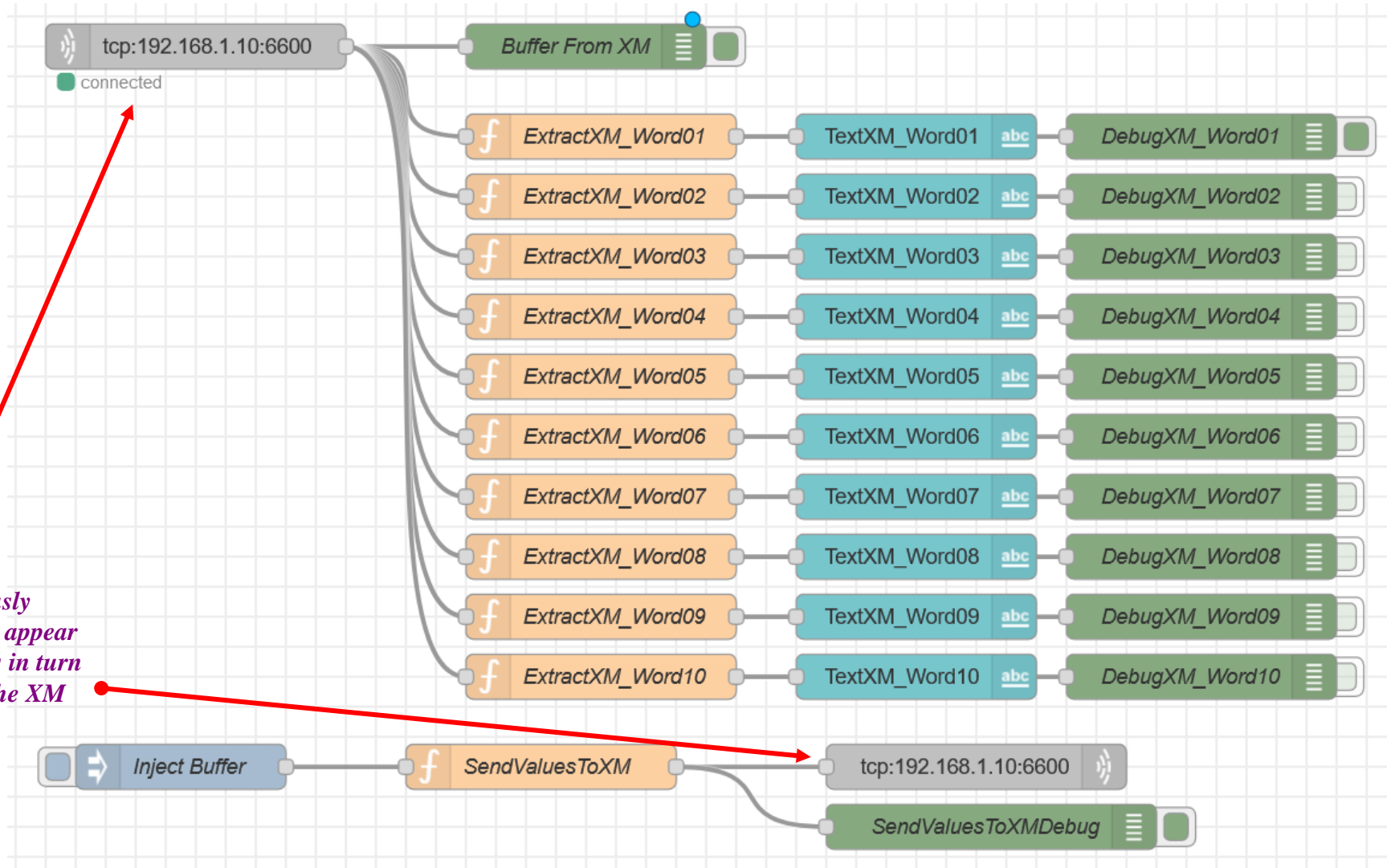
Data "Injected" into the XM

The screenshot shows the configuration for the 'tcp:192.168.1.10:6600' node. The 'Type' dropdown menu is set to 'Reply to TCP' and is highlighted with a red dashed box. Other options include 'Delete', 'Cancel', 'Done', and 'Properties'.

Data to Visualizer "Debug"

The screenshot shows the configuration for the 'SendValuesToXMDebug' node. The 'Output' dropdown menu is set to 'msg. payload' and is highlighted with a red dashed box. The 'To' section has 'debug window' checked, while 'system console' and 'node status (32 characters)' are unchecked. The 'Name' field is set to 'SendValuesToXMDebug'.

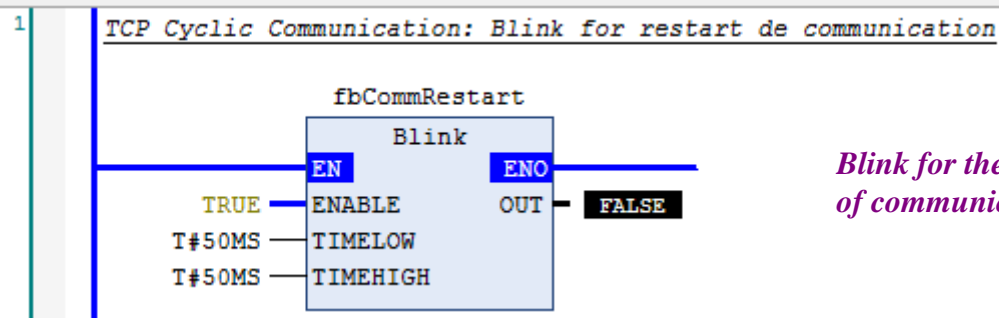
- The sending of data is activated from the element with which we establish the initial connection



The connection previously opened and that should appear as "connected" enables in turn the sending of data to the XM

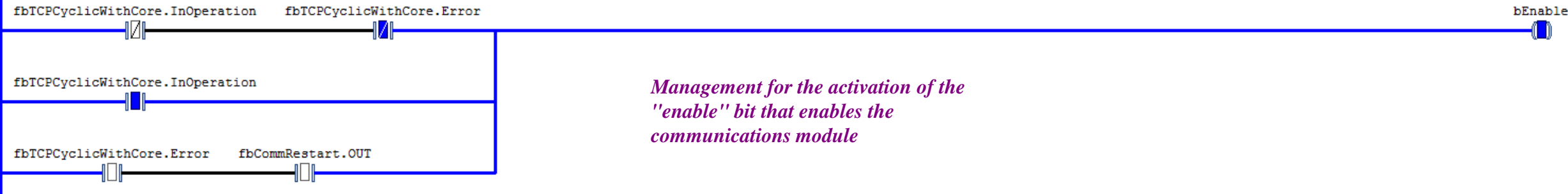
Program at XM

- For the example program located in the XM we will use practically the same structure used in the example of cyclic communications. Only in this case the "mode" used will be of the Cyclic Event Server type



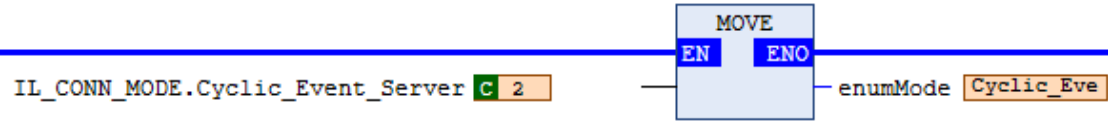
Blink for the "restart" of communications

2 | TCP Cyclic Communication: Check to Start Enable Communication



Management for the activation of the "enable" bit that enables the communications module

3 TCP Cyclic Communication: Select Communication Type (in ctrlX Core select "Cyclic Client")

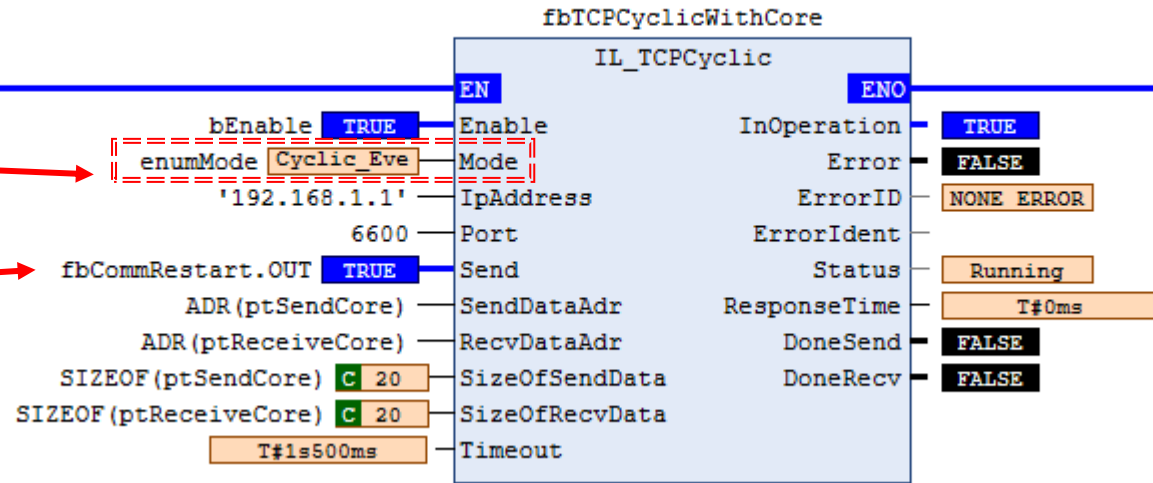


Selection of the communication mode. In the example "Cyclic_Event_Server"

4 TCP Cyclic Communication: Activate comunicacion with NodeRed of ctrlX Core (IP 192.168.1.1 Port 6600)

Mode

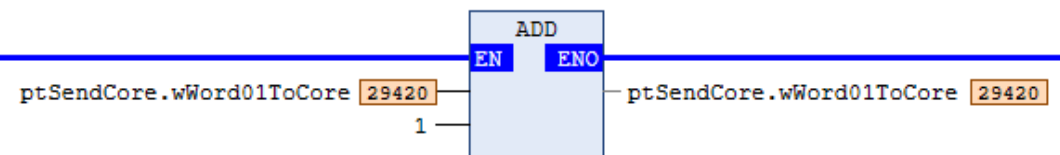
Impulse-controlled sending mode. Required in "Cyclic Event Server" mode



Using this example we can restore the connection in any of the cases.

Value for test sent to ctrlX Core

5 TCP Cyclic Communication: Send Counter To ctrlX Core



The screenshot shows a software interface with a table of variables and their values. The table has columns for Expression, Type, Value, and Prepared value. Red arrows point from the 'Value' column to a list of text elements on the right, which are labeled TextXM_Word01 through TextXM_Word10. The values in the table are: TRUE, 23191, 23, 12, 13, 14, 543, 16, 17, 18, 19.

Expression	Type	Value	Prepared value
bEnable	BOOL	TRUE	
ptSendCore	DUT_SendTo...		
wWord01ToCore	WORD	23191	
wWord02ToCore	WORD	23	
wWord03ToCore	WORD	12	
wWord04ToCore	WORD	13	
wWord05ToCore	WORD	14	
wWord06ToCore	WORD	543	
wWord07ToCore	WORD	16	
wWord08ToCore	WORD	17	
wWord09ToCore	WORD	18	
wWord10ToCore	WORD	19	
ptReceiveCore	DUT_Receive...		
enumMode	IL_CONN_MODE	Cyclic_Even...	
fbCommRestart	Blink		
fbTCPCyclicWithCore	IL_TCPCyclic		

TextXM_Word01 23193

TextXM_Word02 23

TextXM_Word03 12

TextXM_Word04 13

TextXM_Word05 14

TextXM_Word06 543

TextXM_Word07 16

TextXM_Word08 17

TextXM_Word09 18

TextXM_Word10 19

Thank you for your attention

