Application note			
Subject	V430-F setup for EIP communication in Sysmac studio with NX/NJ PLC	**Date**	26-May-2021
Version	1.2	**Author**	Eldad Ben Shalom OCR FAE EMEA eldad.ben.shalom@omron.com

Purpose

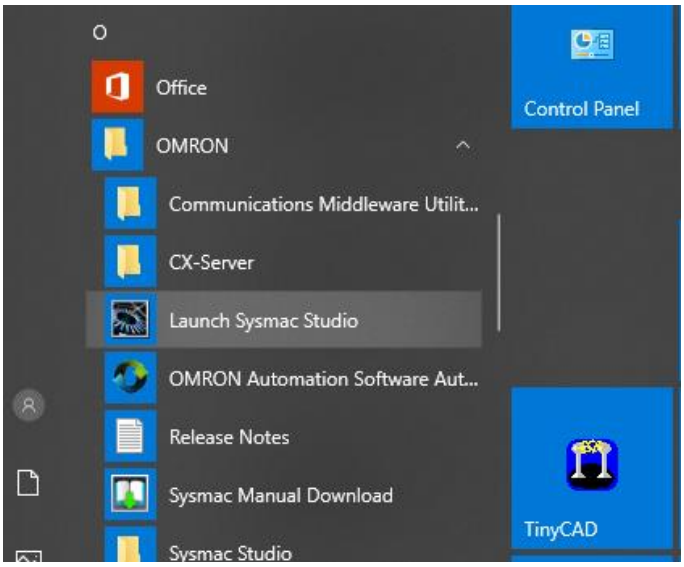
The purpose of this document is to describe the setup configurations required for adding and using V430 code reader in Sysmac PLC project with OMRON NX/NJ PLC. The intention is to target beginners with Sysmac and with V430. Therefore, extra details are explained with a walkthrough the steps to bring up a working setup. In this document there are practical examples of read / write data between the PLC and reader, sending match data and flow control implementation.

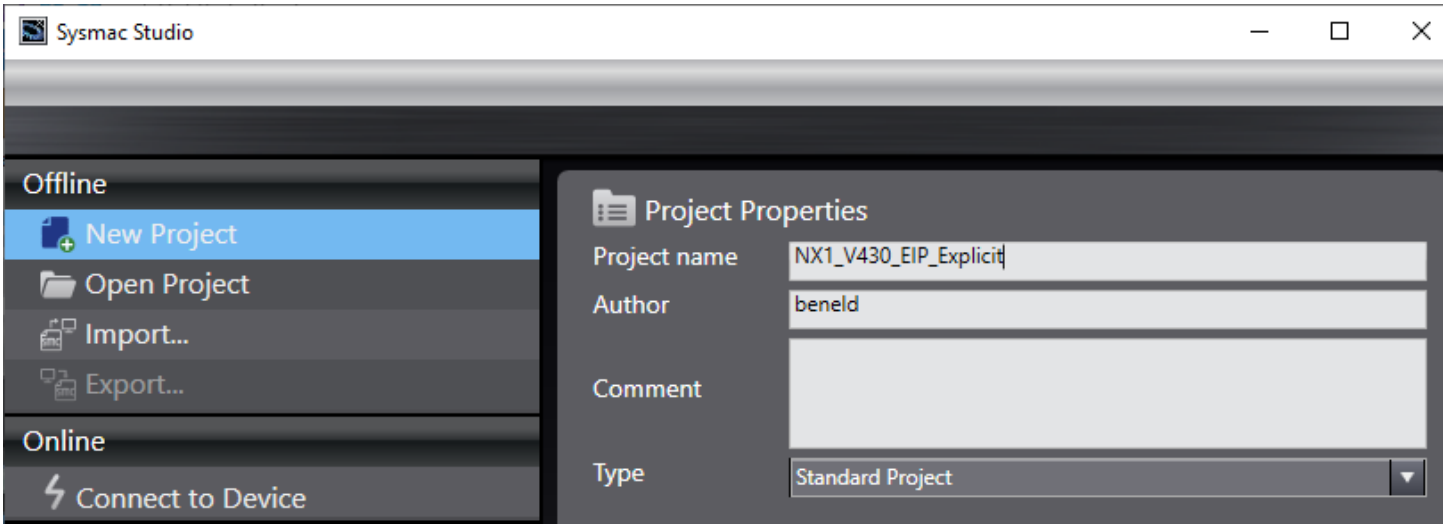
Required setup/HW

This tutorial document uses Omron PLC model NX102-9020 and V430-F code reader.
Software: Sysmac studio

Create new project

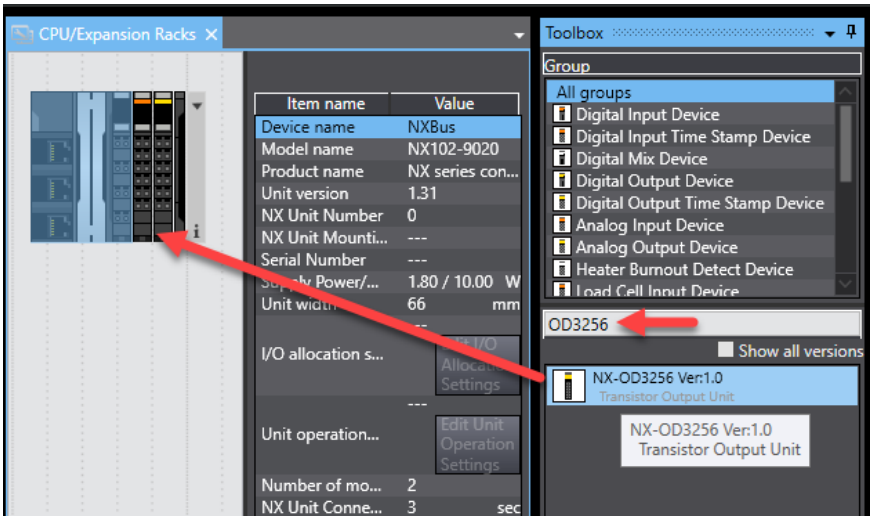
1. Start-up Sysmac software and create a new project, providing a new name to the project.





Adding the PLC controller to the project

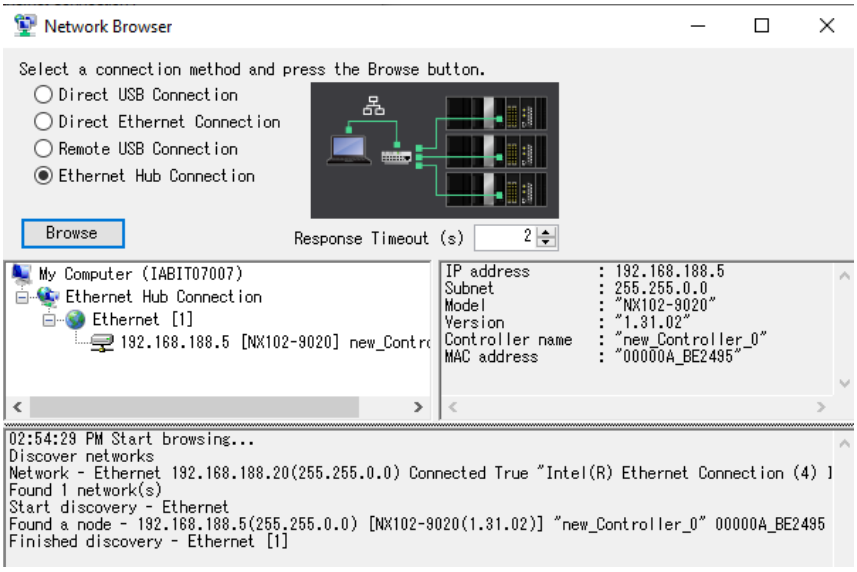
Add the PLC and the external modules like power supply and I/O extensions into the project, by finding them first in the toolbox search on the right side. Then drag and drop into the PLC rack on the left.



Verify correct network settings

Next stage is to configure the network settings on PLC, PC and reader to be on the same network. For example, in our project the network settings are below:

PLC:	192.168.188.5,	255.255.0.0
PC:	192.168.188.20,	255.255.0.0
V430 :	192.168.188.2,	255.255.0.0

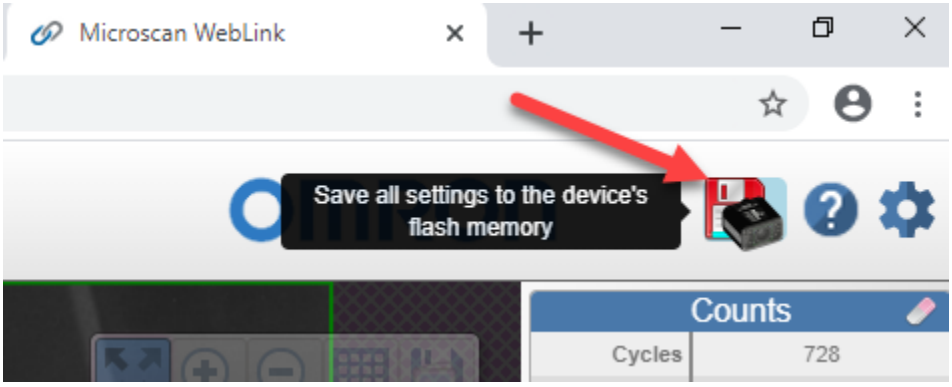


Omron Europe B.V.
Wegalaan 67-69
2132 JD Hoofddorp

Reader configuration

On the reader side, we need to enable EIP communication, then reboot.
Enable EIP from Weblink (verify that byte swapping is disabled).

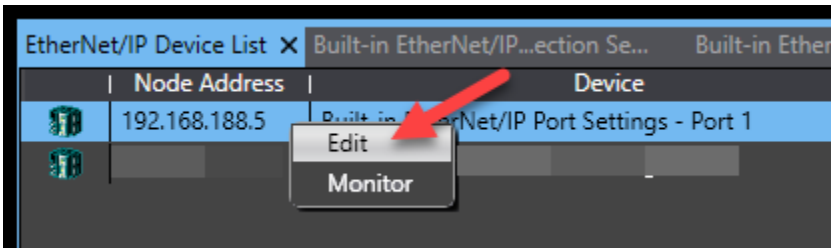
After enabling EIP, save settings to flash and reboot the device:



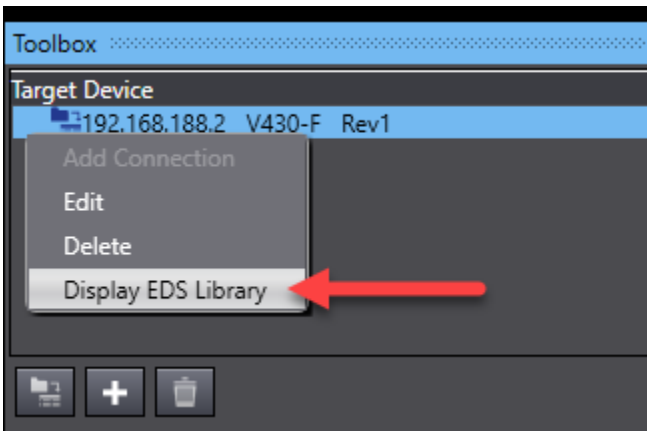
Install EDS file

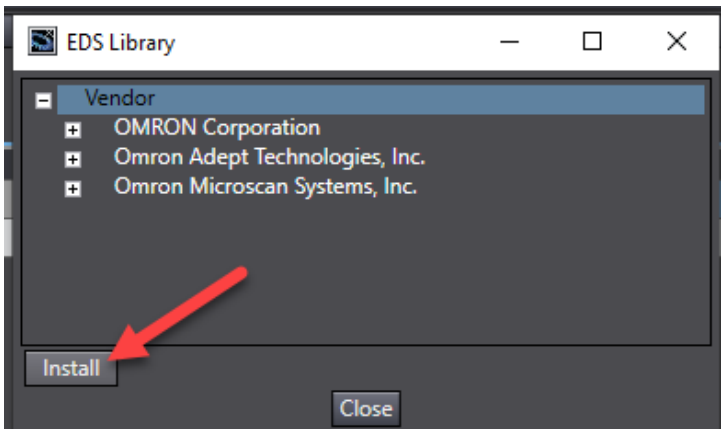
In Sysmac menu, select **Tools->EIP connection settings**, In the toolbox **right click->display EDS library**. If V430 EDS is installed already, it will appear there. If not, press install button and browse to the EDS file saved locally on the PC. The EDS file can be downloaded from Omron GTKS website. Verify that the EDS file used corresponds to the firmware version on the reader. See more info in the documentation.
(EDS file used for this tutorial: V430(32-9000097-01.eds**)**

In EIP Device list window, right click on the PLC, then Edit:

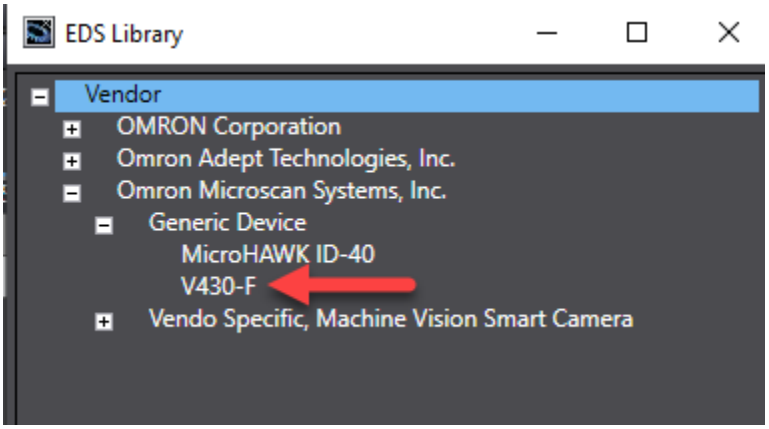


Then in the toolbox on the right side, right click in the window, then ‘Display EDS Library’.
Then click ‘Install’ button and browse to the EDS file on the PC.



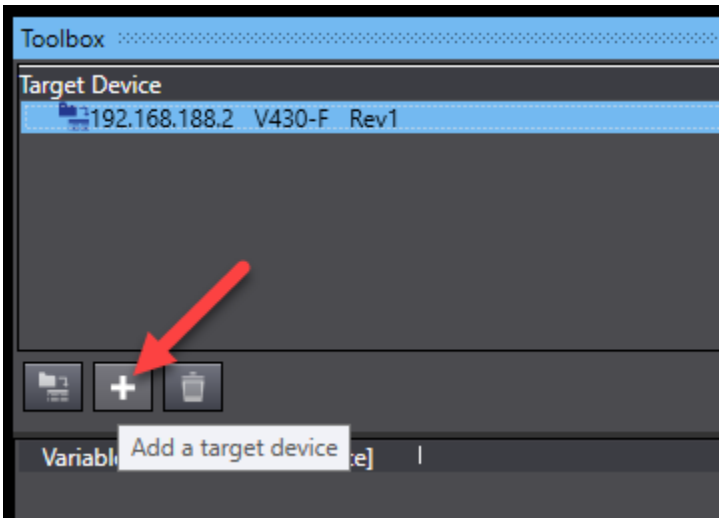


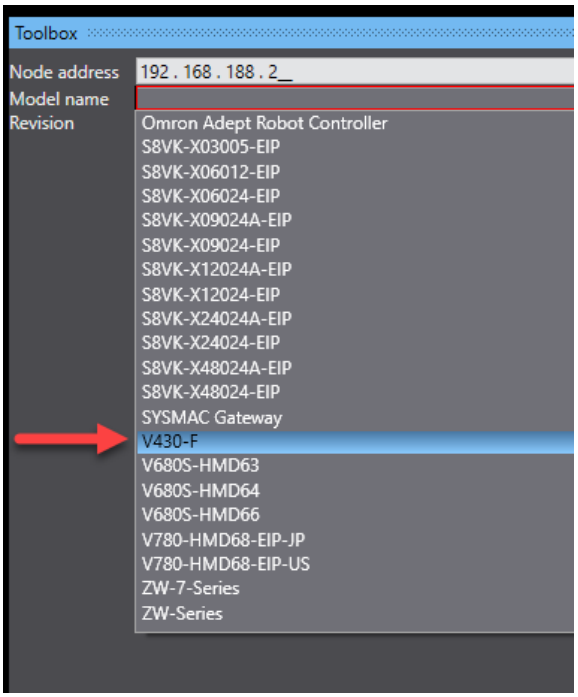
After a successful installation of the EDS file, the V430 appears in the installed library:



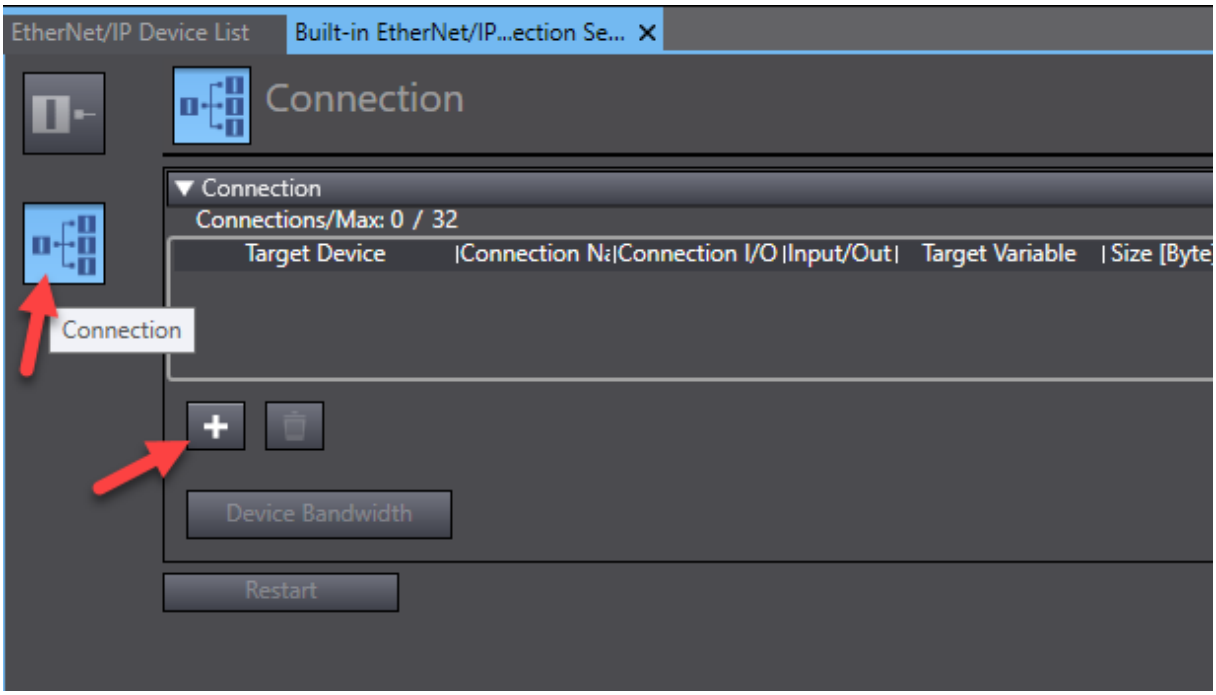
Create new EIP connection for the reader

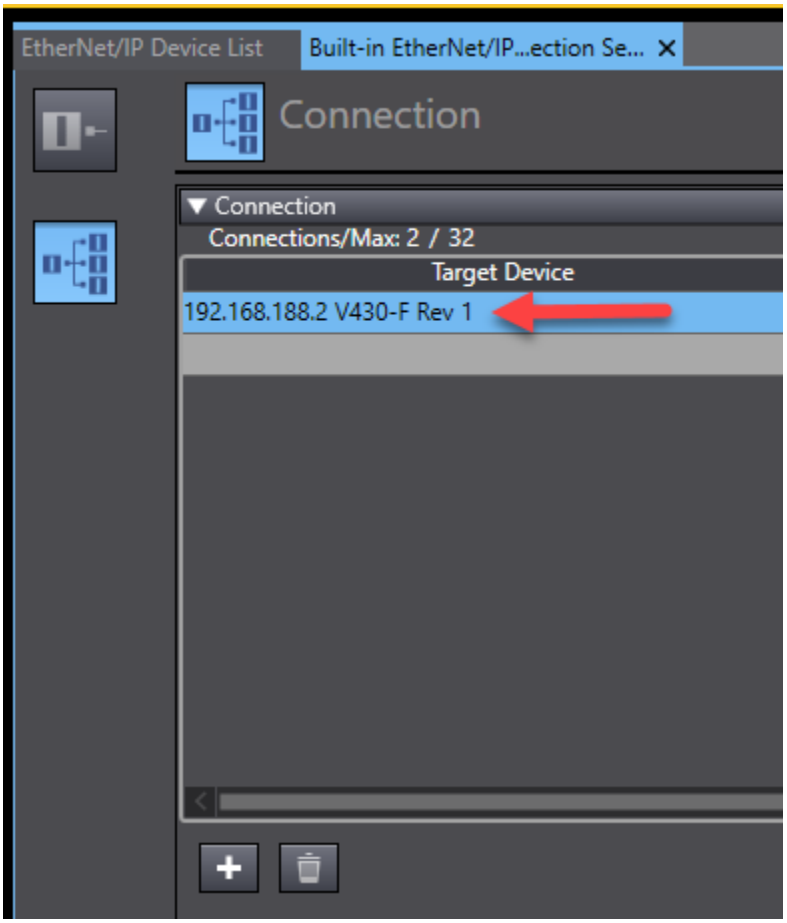
After EDS file is installed, in toolbox on the right side, Click '+' button to add a device, give IP settings and select V430 from the list. Then V430 will be available for selection as a new EIP connection.





Open window Built-in EIP port settings in connection view, and add new connection:





Sysmac library with data type definitions

What is sysmac data types

Data types are pre-defined data structures which are relevant for a specific EIP I/O device. When connected over EIP, the PLC can read and write data according to specific structure which must be defined in the PLC project. Once the data type lib is added properly to the project, the PLC programmer can then use those pre-defined data types in the programming.
It is not mandatory to use pre-defined data types for communication with V430 from the PLC. It provides the PLC designer with tools to create efficient and readable code especially in cases of many readers connected to a single PLC. It also saves time for the programmer.

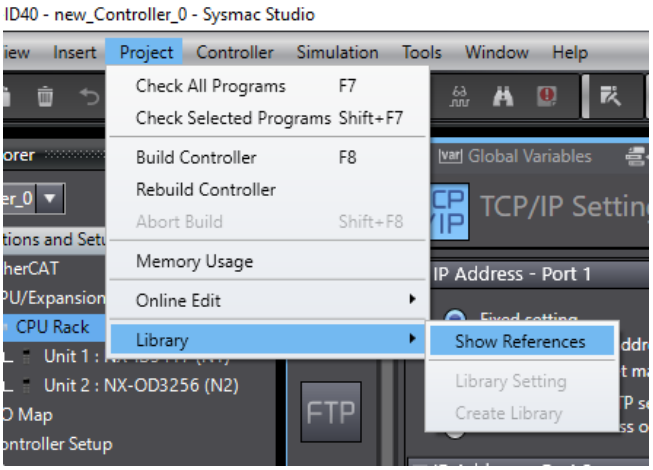
Selecting the correct data type library

OMRON provides for V430 6 different lib files for 6 different combinations of Input/Output assembly pairs (please see Input/Output assembly overview in the following chapter). After deciding which combination will be used in the project, the user can select the corresponding library file (.slr) and use it in the project:

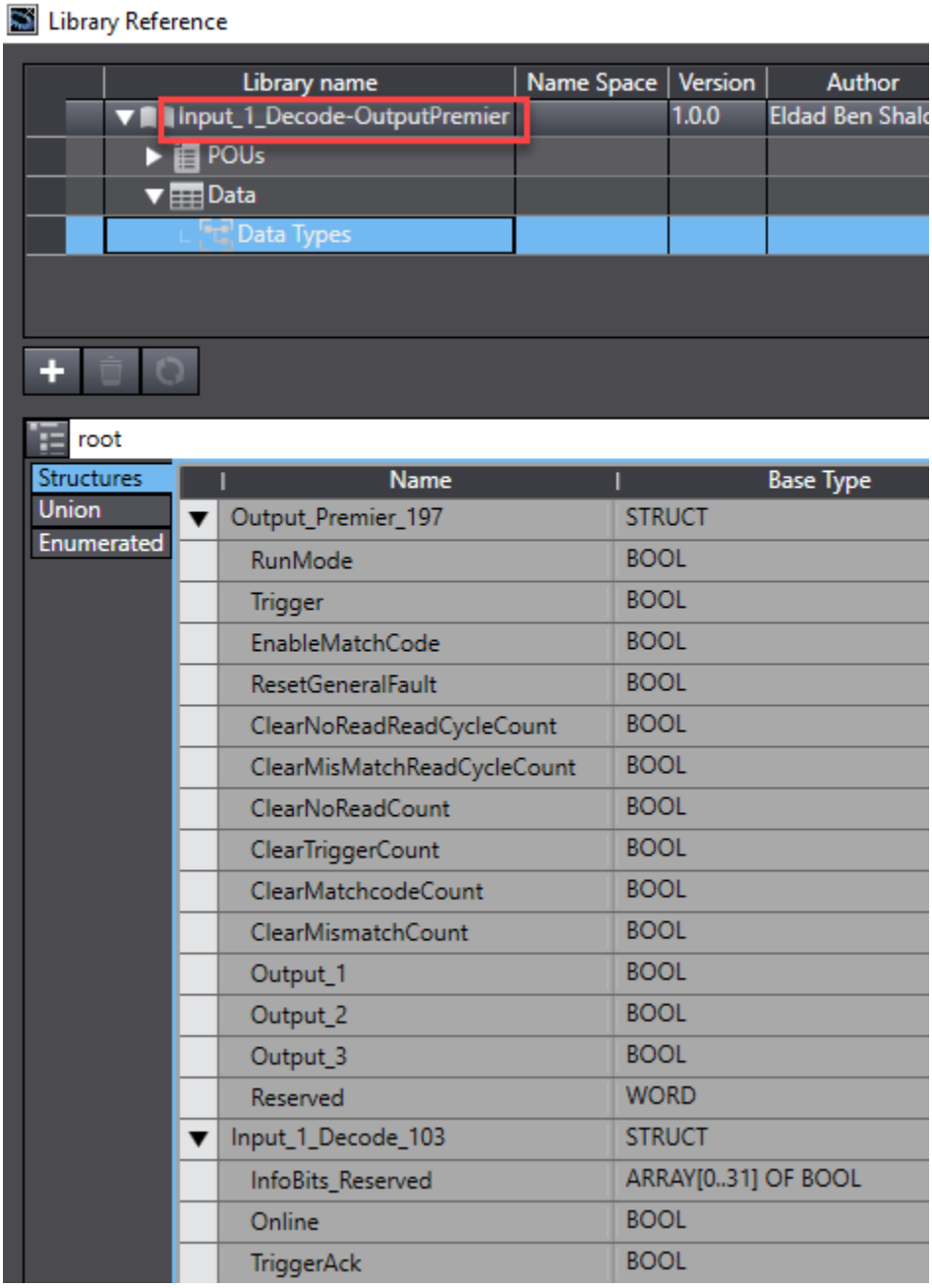
- BigInputAssembly-OutputLegacy.slr
- Input_1_Decode-OutputPremier.slr
- Input_4_Decode-OutputPremier.slr
- Input_MXL-OutputPremier.slr
- Input_N_Decode-OutputPremier.slr
- SmallInputAssembly-OutputLegacy.slr

Adding library file to the project

To use OMRON lib for V430 data types, follow the steps below:
Import the required library which includes data structures for V430 memory mapping. To install the library file, In the menu select **Project->Library->Show References**, then add the library by clicking on the **+** button, browsing to the provided .slr file in the PC.



Then we have the required data types and structures pre-defined in the project for the V430. When we look inside the library, we can see the data structure in detail, as defined in the V430 communications manual:



Input / Output assembly

In this stage, before we add connection variables, we need to select the desired input / output assemblies to work with. There are several options to choose from. Only one module can be selected from input, and one form output. See table below for all the valid pairs of input/output assemblies. *Note that selecting a wrong combination of input/output assemblies might lead to malfunction in the communication.

Possible combinations	Input Assembly name	ID Number	Output Assembly name	ID Number
1	Small Input	100	Output legacy	198
2	Big Input	101	Output legacy	198
3	MXL/SLC	102	Output premier	197
4	Input 1 decode	103	Output premier	197
5	Input 4 decode	104	Output premier	197
6	Input N decode	105	Output premier	197

Please refer to the documentation for full details. Here we will give example of using Big Input Assembly (#101) and Legacy output assembly (#198).

Using Big Input Assembly

When more info is required to be read from the reader, like real time reader status for example, which is required for proper communication handshake between reader and PLC, we will need to use the Big input assembly (Assembly 101, which has 176 byte in total):

SHORT DESCRIPTION	SIZE (BYTES)
USER-DEFINED TAG ECHO	4
COMMAND ECHO	4
OUPUT CONTROL ECHO	4
EXTERNAL INPUT STATUS	4
EXTERNAL OUTPUT STATUS	4
DEVICE STATUS	4
READ CYCLE SEQUENCE COUNTER	4
TRIGGER COUNT	4
DECODE/MATCH COUNT	4
MISMATCH COUNT	4
NOREAD COUNT	4
DECODE DATA LENGTH	4
DECODE DATA STRING	128

For more info about the input assembly, please refer to the documentation of ID-40 reader. In the following pages we bring some examples of reading useful input data from the reader. Let's look first at the documentation to see what info we can get from the device:

4.10.5.1 External Input Status Bit Field

BIT	PIN NAME
0	Trigger
1	New Master
2-31	Reserved for future use

0 = No current sensed on input

1 = Current sensed on input

4.10.6 External Output Status

The current status of the physical output pins on the unit

BIT	PIN NAME
0	Output 1
1	Output 2
2	Output 3
3-31	Reserved for future use

0 = Output contact is open

1 = Output contact is closed

4.10.7 Device Status

Provides the current status of the unit. Below is the bit field table that defines each bit and the relationship to the unit's status

BIT	PIN NAME
0	Reserved
1	New Master Requested
2-7	Reserved for future use
8	Scanning Disabled
9-15	Reserved for future use
16	In read cycle
17	Actively Scanning

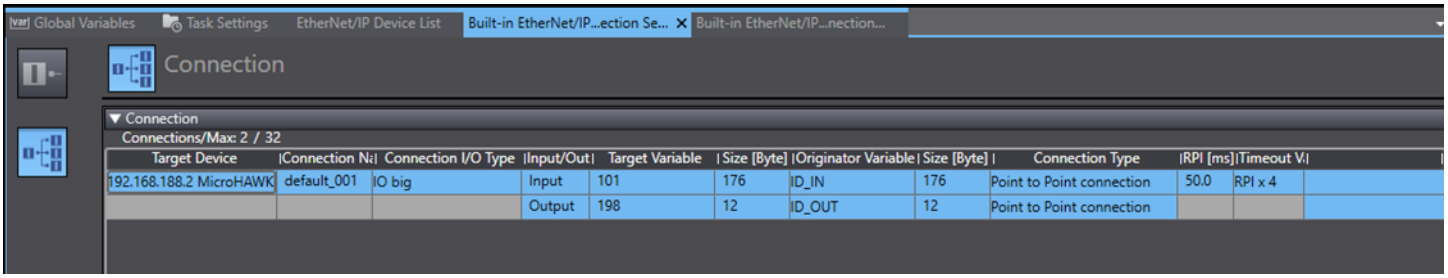
Counters

Several counters are available, which can be very useful in run time. Triggers counter, Read Cycle counter, No Reads counter and more…

Decode data
Just like in the Small Input Assembly, here we also have the decode string with the length info of the string in the preceding 4 bytes.

Required setup in Sysmac

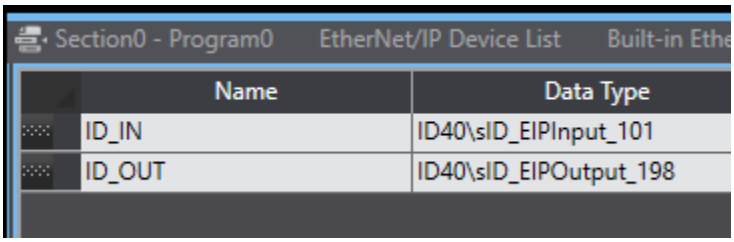
To use the Big Input structure to get data from the reader, we need to define the correct input variable in Sysmac (step by step instructions are described earlier in this document). After the setup, verify that the displayed structure size in bytes is correct, according to the documentation. Otherwise the project will not be downloaded to the PLC properly.



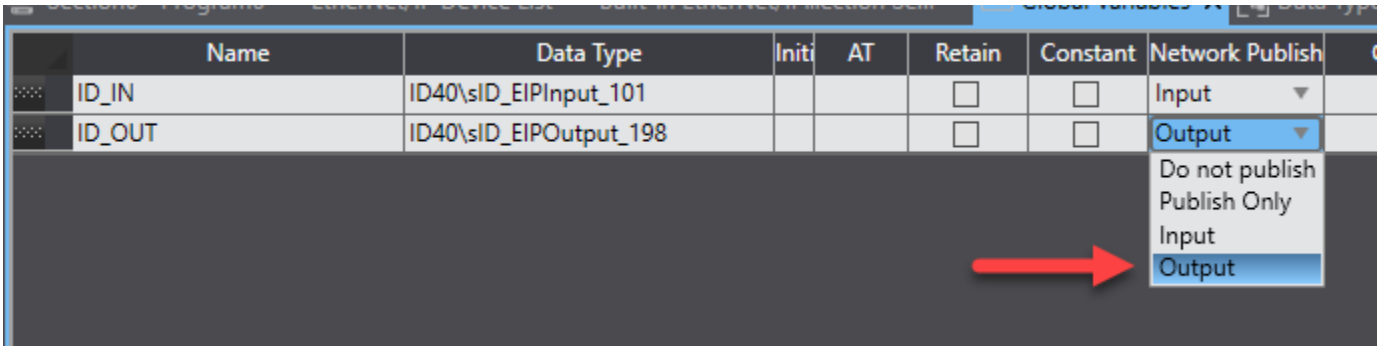
*In this tutorial we use a library file which already has the required pre-defined data types, which makes the setup easier.

Create global variables

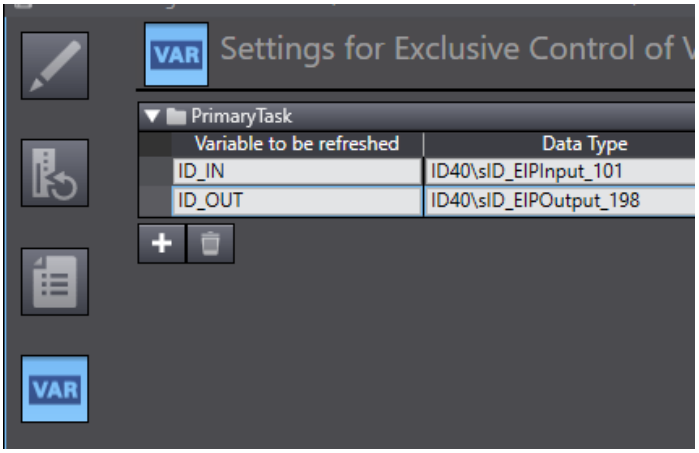
In the Sysmac explorer on the left side, under **global variables** create 2 new variables, one for input and one for output:



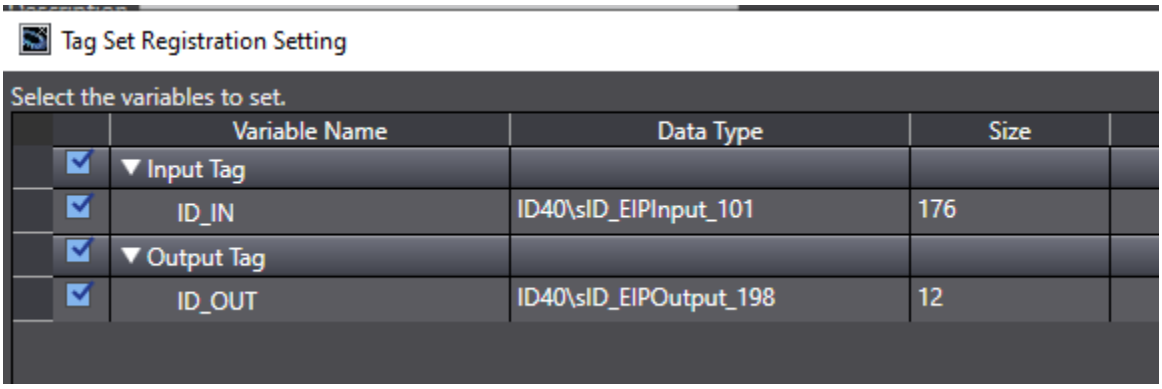
Now make them published, so we can register them later successfully:



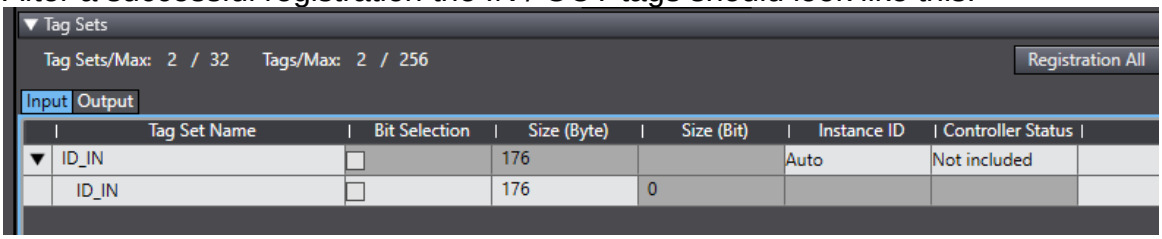
Under task settings -> VAR tab, add the two variables we have just created, using '+' button. Select them from the options:

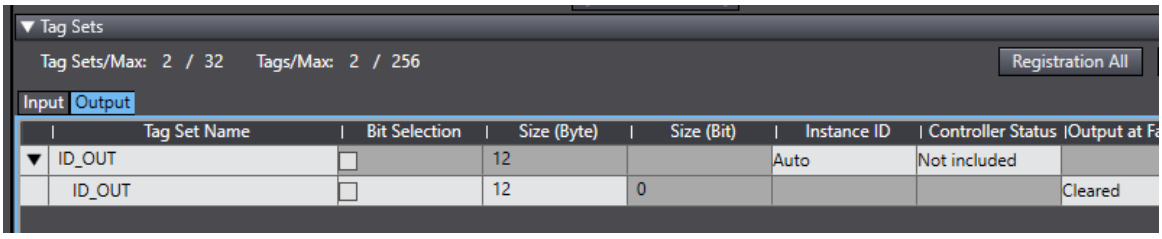


Go to built-in port settings and click the button **register all**, to register all tags:
(To reach this window, first select **Tools->EIP settings**, then **right mouse** on the PLC and click **Edit.** Then click on the button **10)**

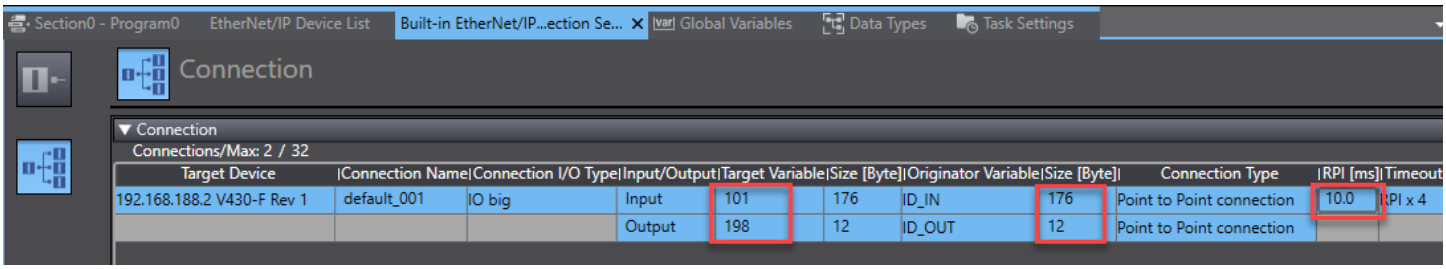


After a successful registration the IN / OUT tags should look like this:





Then in connection tab add the target variable and originator to have the following final settings:

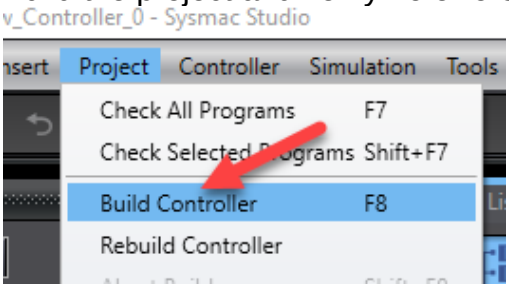


***Note that shortest EIP refresh rate (RPI) should be 10ms faster than that can overload the reader.**

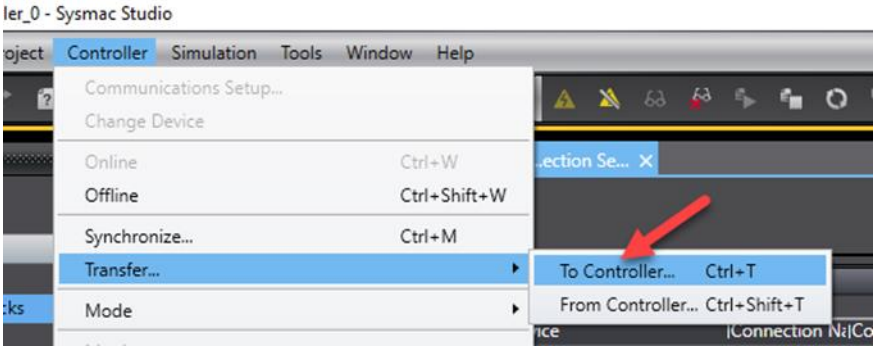
The required settings and configurations for the PLC project are completed. Now the project is ready to compile and download to the PLC, then we can test operations like read / write data between the PLC and the reader.

Transfer the project to the PLC

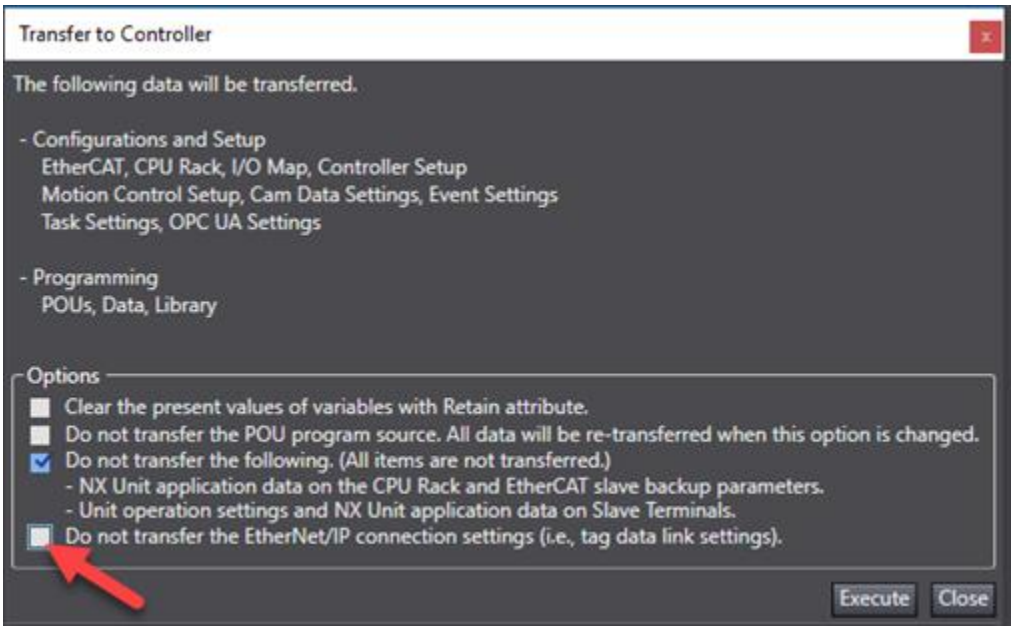
Build the project and verify no errors:



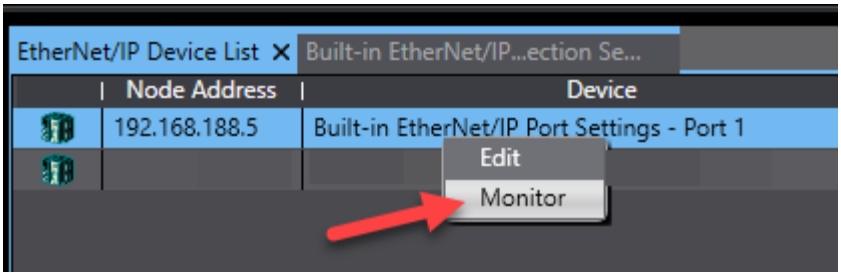
Go to online mode and download the project to PLC. Then verify no errors on PLC in run mode.



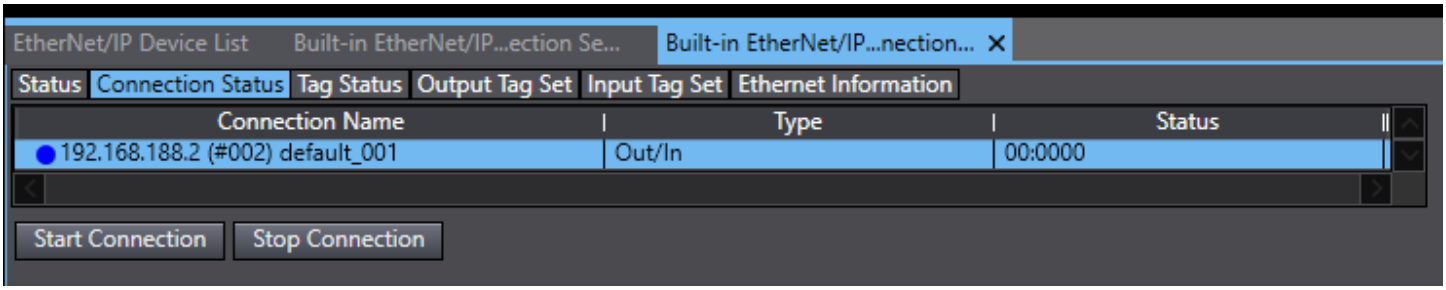
*Note that you need to enable the transfer of Network settings, as we changed those for creating the EIP connection properly. (It is disabled by default). See screenshot below.

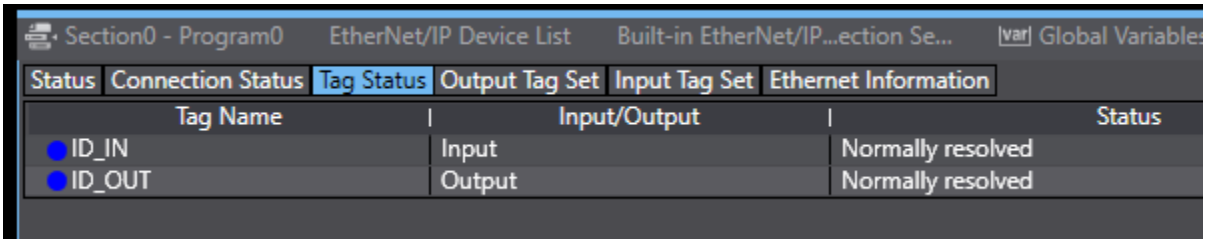


When PLC is set back to run mode, the EIP communication is started. Verify the connection status by monitoring the PLC status: right click on the PLC, then select 'monitor':



See below example for a typical monitor window of a correct connection status and tag status:



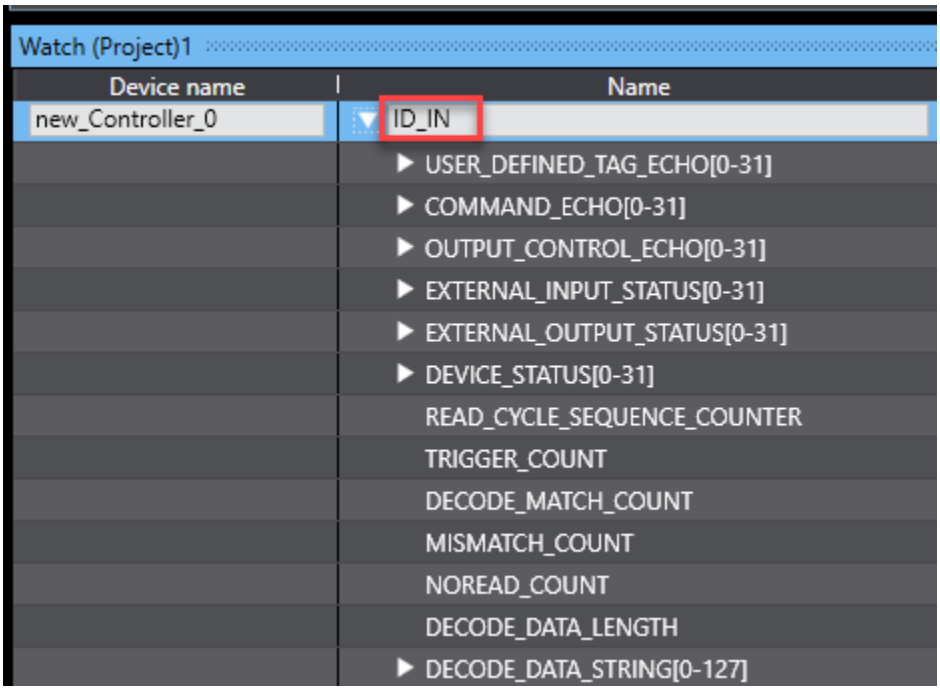


Reading data from reader to PLC

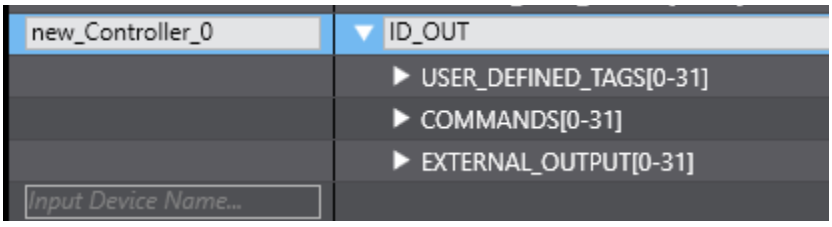
To see the incoming data from the reader to the PLC use watch window, adding the relevant data structures (ID_IN and ID_OUT).

Note that in this tutorial we are using Big Input assembly which has extended amount of input data, which gives useful options for reading useful status data from the reader.

The incoming data from the reader to the PLC can be found under ID_IN structure:

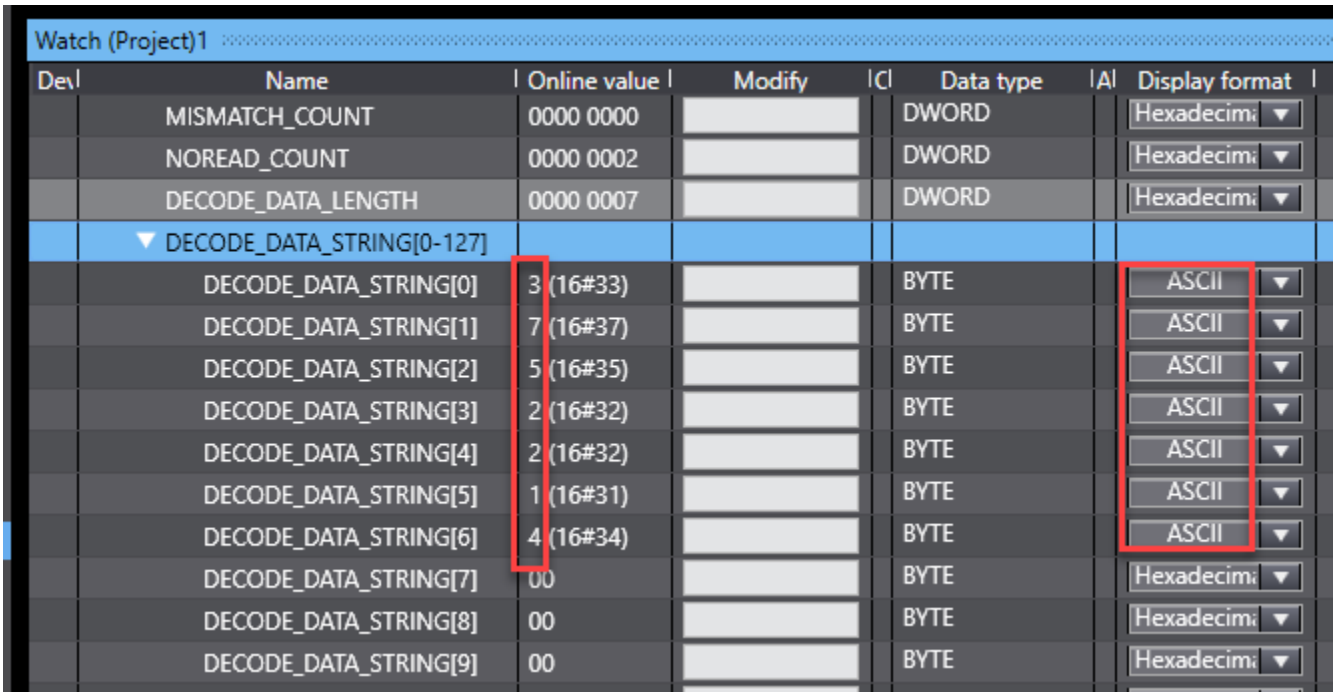


The outgoing data from PLC to reader, can be found under ID_OUT structure:



Example 1 – read decoded text

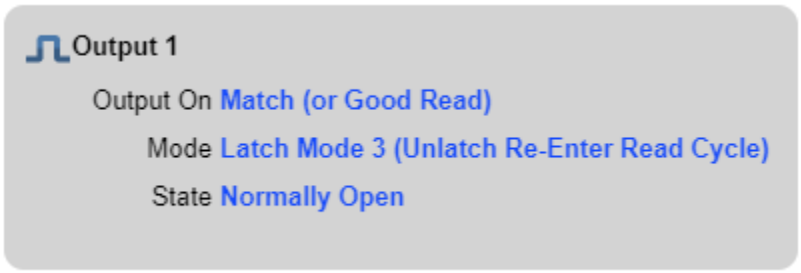
To find the decoded text after a good read or no_read, we need to view the decode data string array as shown below. So verify that there is a symbol in front of the reader, trigger once and view the results.
For better readability we change the display format column to show ASCII format:



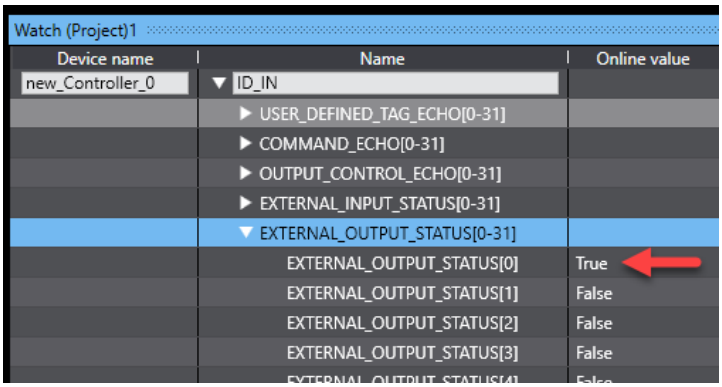
Just for a cross-check, view in parallel the good read symbol in Weblink:

Example 2 - Get status of reader's digital outputs

We trigger a Good Read to activate output 1, after configuring it to Latch on a good read in Weblink. After the next trigger, verify that the relevant output is activated:

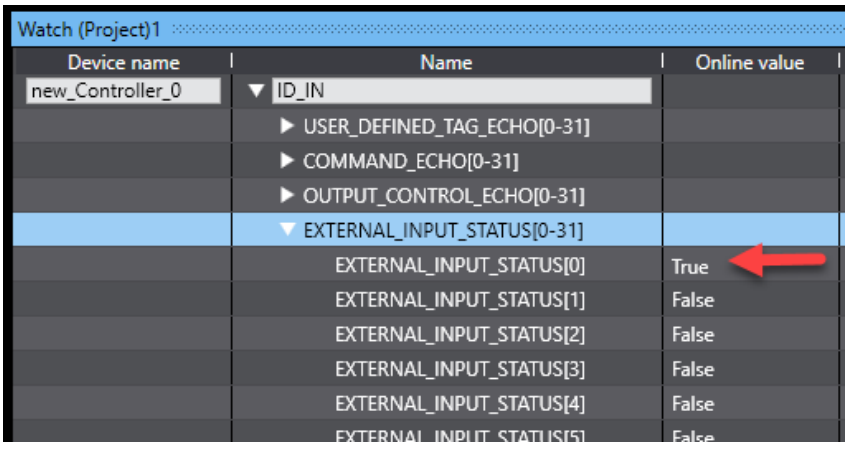


In the watch window, looking at bit 0 of external output status memory location, we will find that this bit is TRUE:



Example 3 - Get status of external trigger input

Each time we trigger the reader from external source (for example Trigger sensor), we can monitor the status of that input signal in bit 0 of External Input Status register:



Example 4 – Read the current value of the available counters

A good monitoring method of the system's performance is to monitor different counters and compare the values. Let's look at the counter values in our setup. In the watch window we extend the view of structure ID_IN to view all counters listed. For better readability we configure the display format column to show number in decimal.

In the screenshot below we can see the current values of the counters in our setup. From the numbers we can learn the following:

1. The reader received 4 triggers in total
2. The reader entered 4 read cycles (it means that there are no missed triggers / overruns)
3. Decode counter shows 3 good reads
4. No_Read counter shows 1 No_Read event

▶ DEVICE_STATUS[0-31]					
READ_CYCLE_SEQUENCE_COUNTER	4			DWORD	Decimal
TRIGGER_COUNT	4			DWORD	Decimal
DECODE_MATCH_COUNT	3			DWORD	Decimal
MISMATCH_COUNT	0			DWORD	Decimal
NOREAD_COUNT	1			DWORD	Decimal
DECODE_DATA_LENGTH	6			DWORD	Decimal

Sending data from PLC to reader

For sending data from PLC to reader, in this example we use the **Legacy output** assembly:

SHORT DESCRIPTION	PLC DATA TYPE	SIZE (BYTES)
USER DEFINED TAGS	DINT	4
COMMANDS	DINT	4
EXTERNAL OUTPUT	DINT	4

Commands:

Using the Command BIT fields we can control the reader's status / operation. For example, to trigger the device from the PLC which will be described later in this document as example.
The possible BIT Commands are listed below:

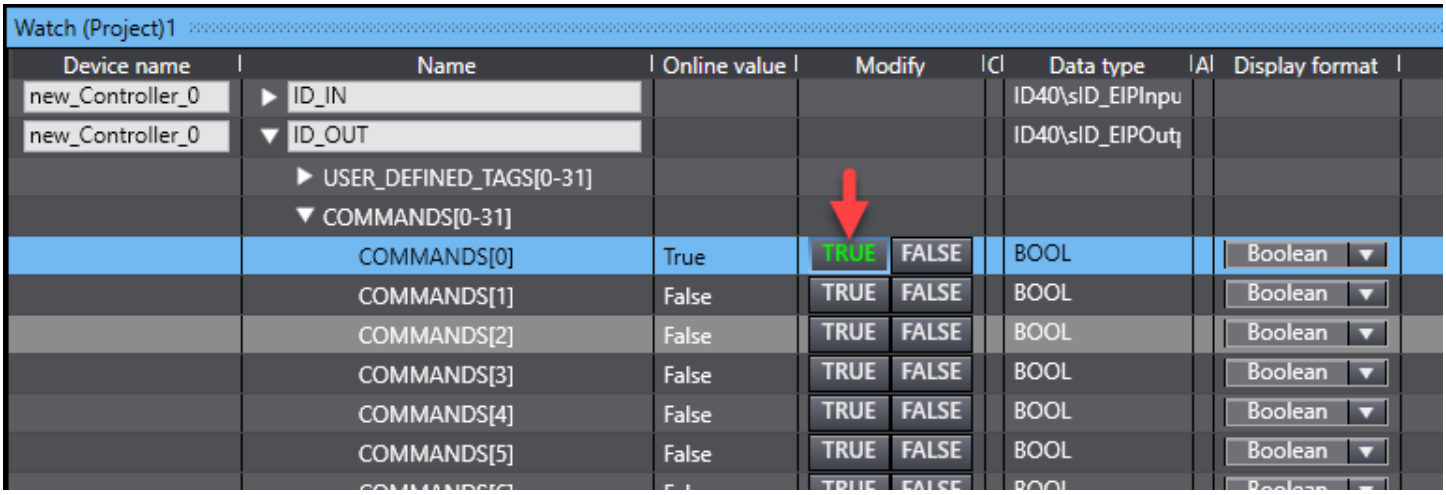
BIT FIELD	COMMAND
0	Trigger
1	New Master
2	Buffer Overflow
3-7	Reserved
8	Disable Scanning
9-15	Reserved
16	Clear Read Cycle Report and Counters
17	Unlatch Outputs
18-31	Reserved

=>Trigger (bit 0) – triggers the reader once
=>Disable scanning (bit 8) – puts the reader to online / offline mode
=>New Master (bit 1) – saves a new master on the next good read
=>Clear (bit 16) – Clears the results data from the memory

For detailed description about the functionality below, please refer to the documentation. See below several examples for sending control commands to the reader:

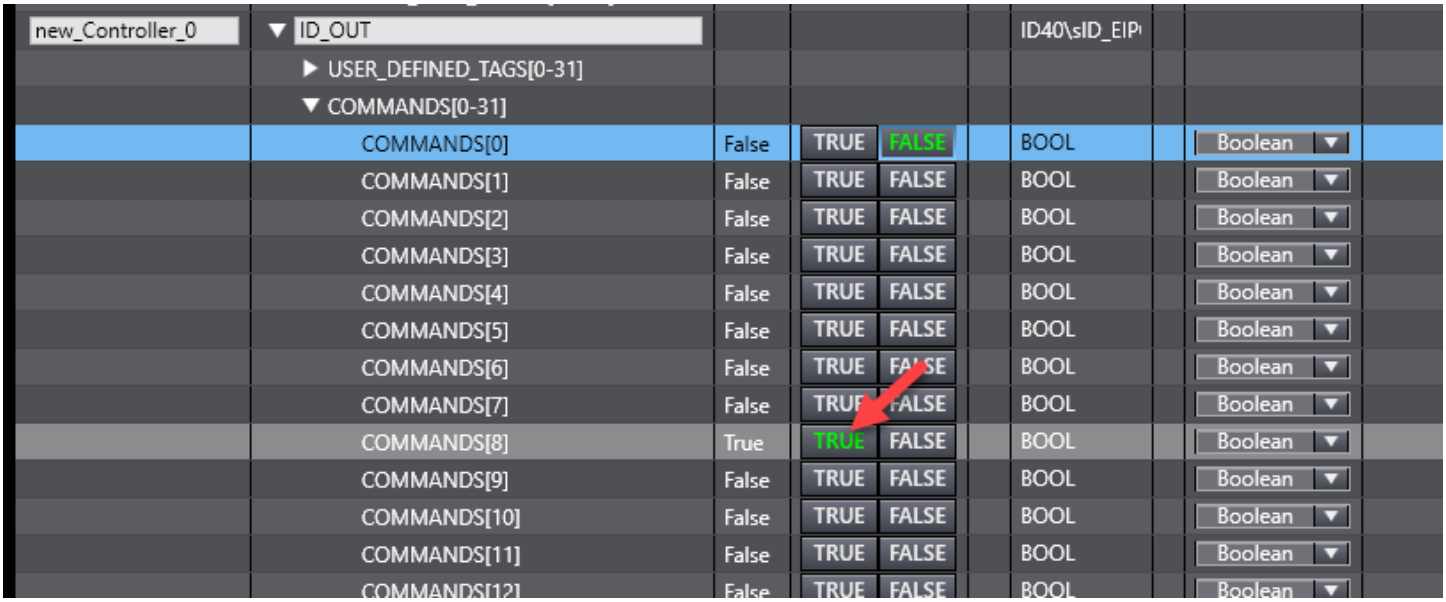
Example 1 - Trigger the reader from the PLC over EIP

When looking at the data structure of the command bit fields, we see that bit 0 is the trigger bit. To trigger the device we need to change this bit from FALSE to TRUE (then back to false):
In the watch window, under EIP_OUT -> Commands change bit 0 to TRUE and watch the reader to verify that it was triggered:



Example 2 - Disable scanning (put reader to offline) over EIP

We can put the reader to offline / online state using bit 8 of the of the Command BITs register. By default this bit is FALSE and reader online (scanning enabled -> reader reacts to triggers). When the bit is TRUE, it means that the reader is put to offline and scanning is disabled. Triggers are ignored.



After forcing this bit to TRUE in watch window, try to trigger the reader and verify that it is not reacting to triggers.

Sending match data to the reader - Explicit messages

The Matchcode feature is commonly used. This feature enables the reader to compare the reading result to a known string (Matchdata) on every trigger and send out a Pass/Fail result. (Please refer to the manual for more details).

The preferred way to send a new Match data to the reader is by K command over EIP. It is possible only by using explicit messages. We will give first a quick overview below, then describe a real example on how to implement this in the PLC program.

Explicit messages are useful also for other purposes. It enables sending K-commands to the reader. By sending K commands we can change the reader settings on the fly, as well as read various status data from the reader.

Explicit messages – Overview

Explicit messaging is sending / receiving data between PLC and reader over TCP/IP communication. Therefore, it is meant for data that is not time critical like the Implicit messaging I/O data. Implicit messaging (which is the cyclic data which goes back and forth between reader and PLC) is sent over UDP packets.

Using Explicit messaging the user can send and receive data which is not pre-defined in size and timing like the implicit I/O messaging. For example, the user can send and receive any K commands to and from the reader and request any data from the reader, as well as change reader configuration in run time.

To send and receive K Commands to the V430 reader, the assembly below can be used:

Serial Command Assembly (Instance Decimal: 69 Hex: 0x45) IN/OUT = PLC ⇔ ID-40 reader

This assembly is accessible only through explicit messaging, to allow the programmer to use the implicit Input/Output assemblies in parallel with this explicit assembly. In the same project it will be possible to use the Input/Output assemblies for read/write data to/from the reader, as well as use this explicit command assembly to send and receive K commands.

See relevant info below, taken from the reader device documentation:
(Service code and class are in HEX format).

4.18.1 Assembly Information

To use this assembly please use the following information in the MSG Instruction

Service Code	**45**
Class	68
Instance	1
Attribute	1

4.18.2 Serial Command Assembly Table

Description of the Serial Command Assembly. This mimics a STRING in the PLC program of 256 bytes long.

SHORT DESCRIPTION	PLC DATA TYPE	SIZE (BYTES)
COMMAND LENGTH	DINT	4
COMMAND STRING	SINT[256]	256

Total: 260 bytes

The following table shows which service type is supported by each input / output assembly:

ASSEMBLY NAME	GET (0X0E)	SET (0X10)	GET/SET (0X45)
INPUT SMALL	Yes	No	No
INPUT BIG	Yes	No	No
INPUT MXL/SLC	Yes	No	No
INPUT 1 DECODE	Yes	No	No
INPUT 4 DECODE	Yes	No	No
INPUT N DECODE	Yes	No	No
OUTPUT (LEGACY)	Yes	Yes	No
OUTPUT	Yes	Yes	No
SERIAL COMMANDS	Yes	Yes	Yes

Example 3 – sending matchdata K command using explicit messages

This example was created using PLC NX102-9020 and V430 reader. But the method described below is valid also for NJ PLCs.

Assembly info

We will use the following Assembly info, as taken from the documentation:

Service Code = 0x45 (= 69d)
Class = 0x68 (= 104d)
Instance = 1
Attribute = 1

Send message structure

Structure of the message to send out from the PLC to the reader:

Command length	Command
4 bytes (LSB first)	N bytes (including the brackets <>)

For example, the K command **<K231?>** (requesting matchcode info) has a length of 7 characters. Therefore, the first 4 bytes (command length) will be: **07 00 00 00** (LSB first). Then comes the 7 bytes of the K command itself. The total length of this message will be 11 bytes. 4 bytes of length info and 7 bytes of the command itself.

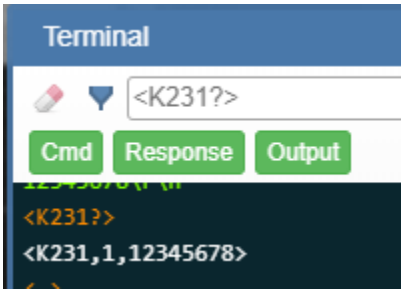
Receive message structure

In case there is a response from the reader to PLC, the structure of the received message is below:

Data length	Data
4 bytes (LSB first)	N bytes (including the brackets <>)

For example, the K command **<K231?>** should receive the following reply from the reader:
<K231, 1, 12345678>
(matchcode info from the reader).

The data length is 17 characters => 17 bytes. So, the first 4 bytes (message length) should be 17 and the next 17 bytes will be the data itself. In total the received message has 17 + 4 = 21 bytes.
See below log data from Weblink terminal, where the Matchcode is inquired and the response arrives:



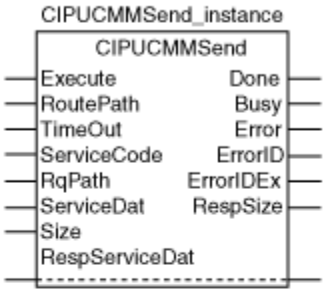
Ladder program implementation – send matchcode / K command

The implementation described below shows how to send a single explicit message from NX or NJ PLC, carrying any K command.

The main function used here is **CIPUCMMSend** which is a standard Sysmac function:

CIPUCMMSend

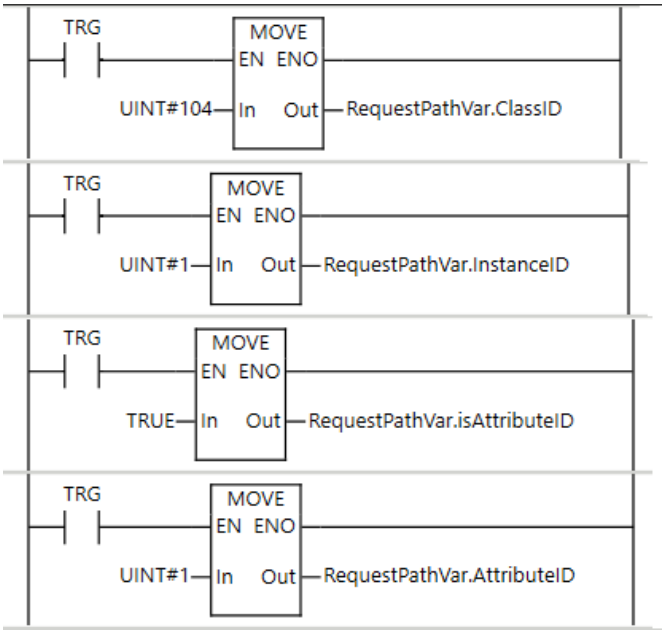
The CIPUCMMSend instruction sends a UCMM CIP message to a specified device on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Send	Send Explicit Message UCMM	FB		CIPUCMMSend_instance(Execute, RoutePath, TimeOut, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

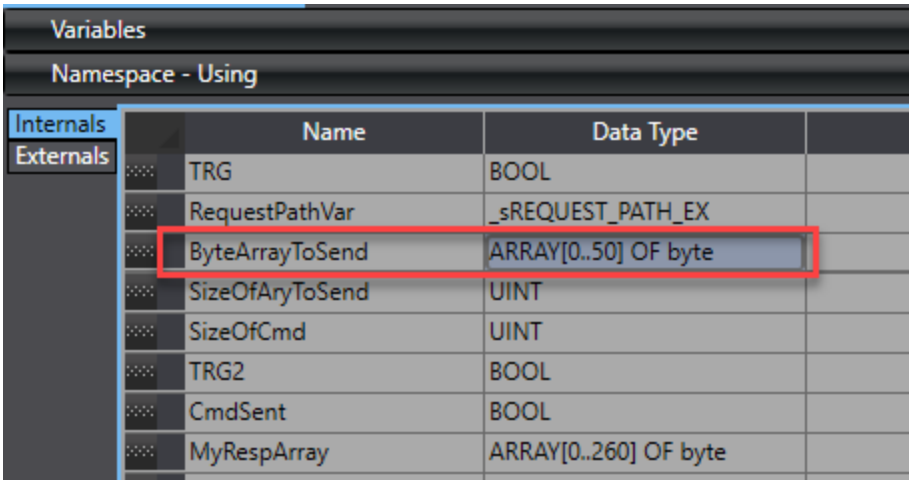
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2.0 s)
ServiceCode	Service code		Service code	Depends on data type.	---	---
RqPath	Request path		Request path	---		
ServiceDat	Command data		Data to send	Depends on data type.		*
Size	Number of elements to send		Number of elements to send			1
RespServiceDat	Response data	In-out	Response data	Depends on data type.	---	---
RespSize	Response size	Output	Response data size	Depends on data type.	Bytes	---

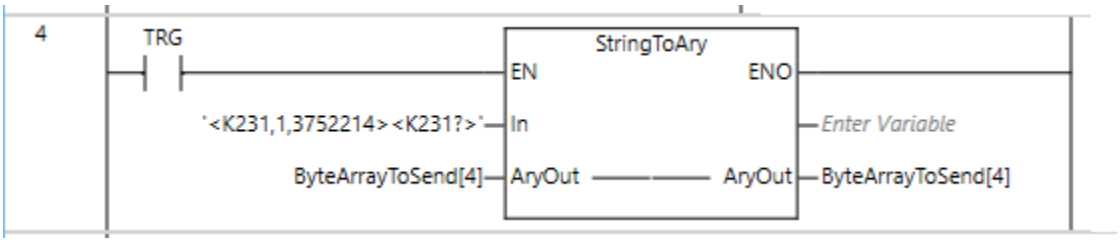
1. In the first 4 rangs, the data structure **RqPath** is assigned with correct values using MOVE functions:



2. In the next rang we define the K command we want to send (send new matchcode data to the reader), then copy the data to the correct place inside **ByteArrayToSend**. In this example, we need to define a local Array with minimum 19 bytes for sending this message:
(4 bytes of length + 17 bytes of the K command itself)
So to be on the safe side, we define a bigger array of 51 bytes. The array should be big Enough to contain all the sent data. It can be bigger but not smaller.



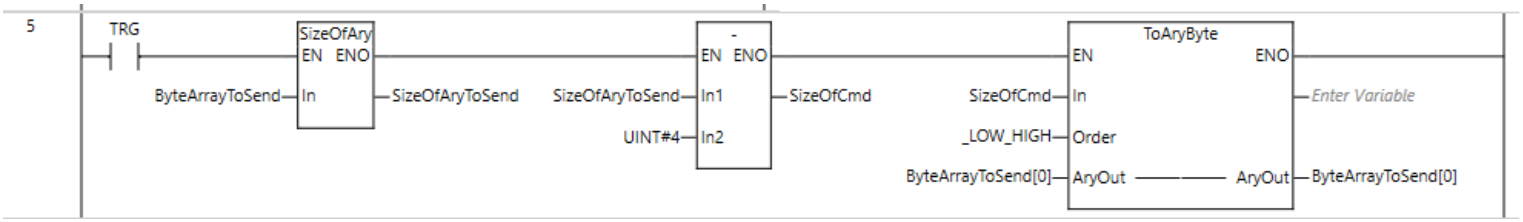
3. Now copy the K command string into the byte array, leaving the first 4 bytes empty.



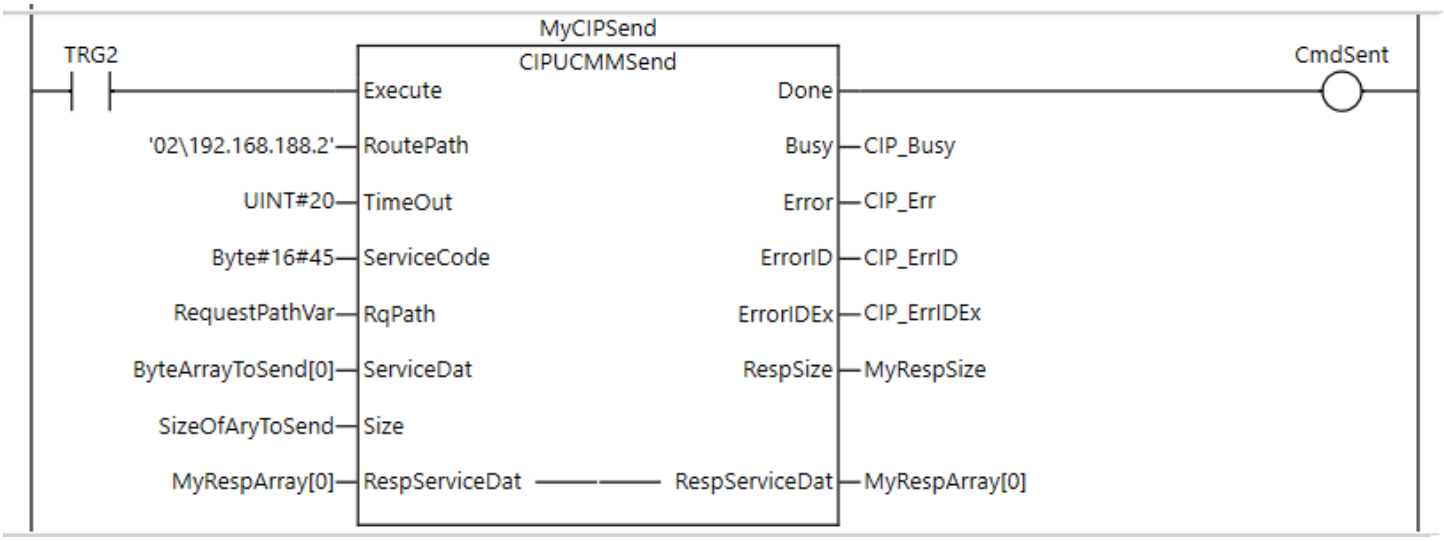
Note that we send at once two K commands. One to send the new match data and another one to inquire the match data. This way we can verify that the new match data is arrived OK to the reader.

In the next rang we extract the command length, then complete our Byte Array with the command length info in the first 4 bytes (LSB first):

4.



5. In the next rang we use the **CIPUCMMSend** function to actually send the data to the reader:

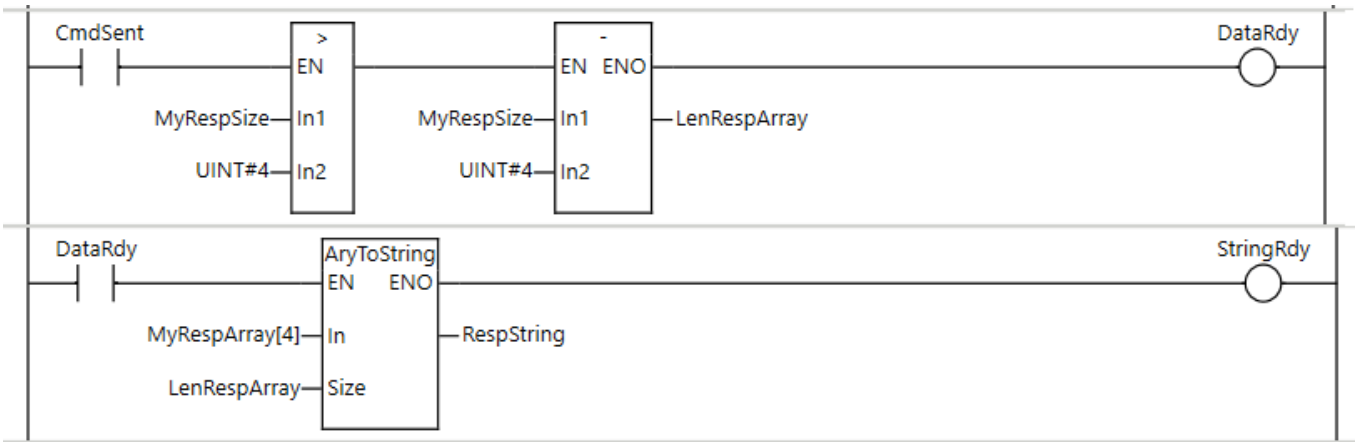


6. When there is a response data sent back from the reader as a reply, we receive it into the array MyRespArray:

MyRespArray	ARRAY[0..260] OF byte

After checking that the response data is not empty (MyRespSize > 4 bytes), the data is separated from the length info and provided in a String variable: RespString

RespString	STRING[257]



See below the complete list of internal variables used in this example:

Variables

Namespace - Using

Internals

Externals

Name	Data Type
TRG	BOOL
RequestPathVar	_sREQUEST_PATH_EX
ByteArrayToSend	ARRAY[0..50] OF byte
SizeOfAryToSend	UINT
SizeOfCmd	UINT
TRG2	BOOL
CmdSent	BOOL
MyRespArray	ARRAY[0..260] OF byte
CIP_Busy	BOOL
CIP_Err	BOOL
CIP_ErrID	WORD
CIP_ErrIDEx	DWORD
MyRespSize	UINT
MyCIPSend	CIPUCMMSend
DataRdy	BOOL
LenRespArray	UINT
StringRdy	BOOL
RespString	STRING[257]

Explicit commands – Common errors

After launching the send function **CIPUCMMSend**, there are possible errors that can occur. The meaning of the error code can be found in the help (Press F1, then search for the error code).
Here are some common errors and the possible root cause:

1. Wrong input parameter

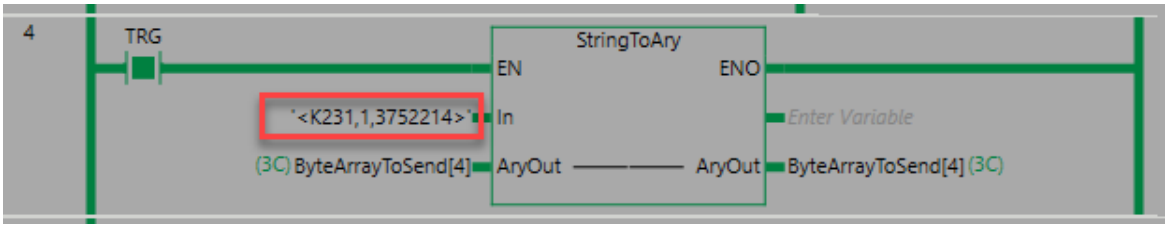
When this error is occurred, it means that one of the input parameters for the send function is out of range or from the wrong type.

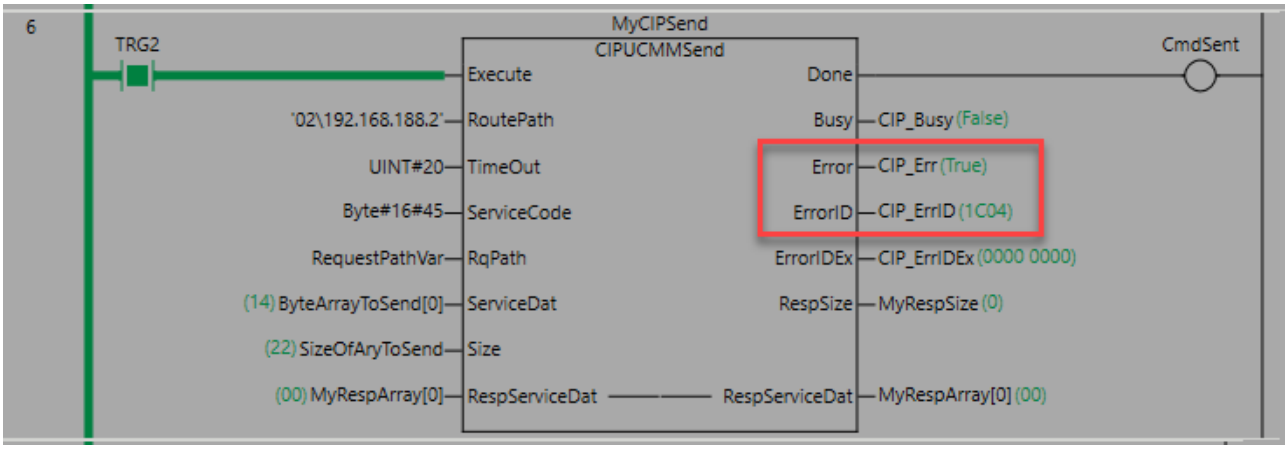
16#0400	Input Value Out of Range

2. Timeout on sending

16#1C04	CIP Timeout

When this error is occurred, it means that there was no response from the reader on this K command. Some commands produce a response and some commands not. For example, the command of sending new matchcode is not producing a response from the reader. Therefore, we should expect a timeout error on sending.

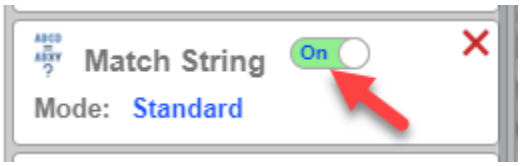


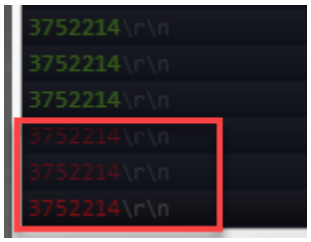


Note that the timeout error happens also when the command is received properly on reader side. So this error should be ignored on PLC side.
A good workaround is sending two K commands together. One to send a new match data and one to inquire the saved match data. This way we can also verify that the change is done.

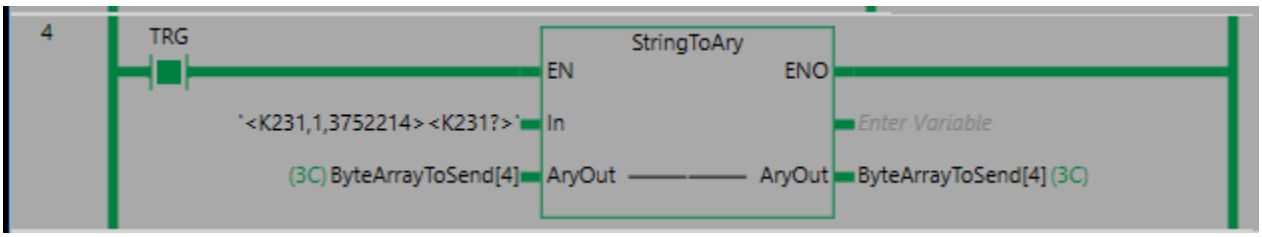
Testing the implementation

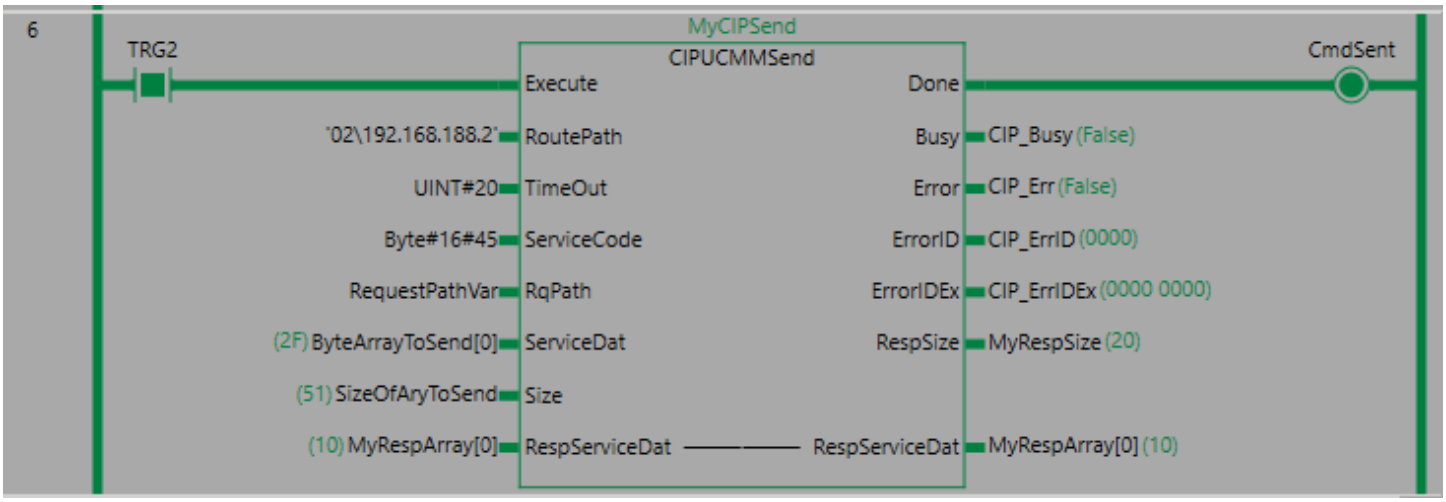
1. In Weblink, verify good read, then enable Matchdata feature and trigger again to verify failure (should fail, as we didn't send yet the Matchdata).



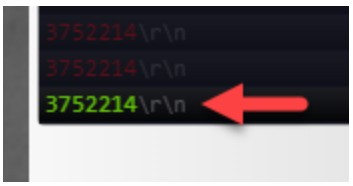


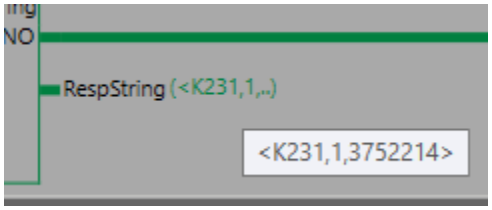
2. Now on PLC side, launch the sending of the K command via explicit, as described above.





3. To verify that the correct matchdata is sent OK, trigger again and view the results in Weblink to see if Pass or Fail. Since the inspection is passed, we can conclude that the matchdata is sent OK to the reader (on PLC side it is a good practice to verify the match data change by comparing the new matchcode we sent to the matchcode we receive in the response on our inquiry.





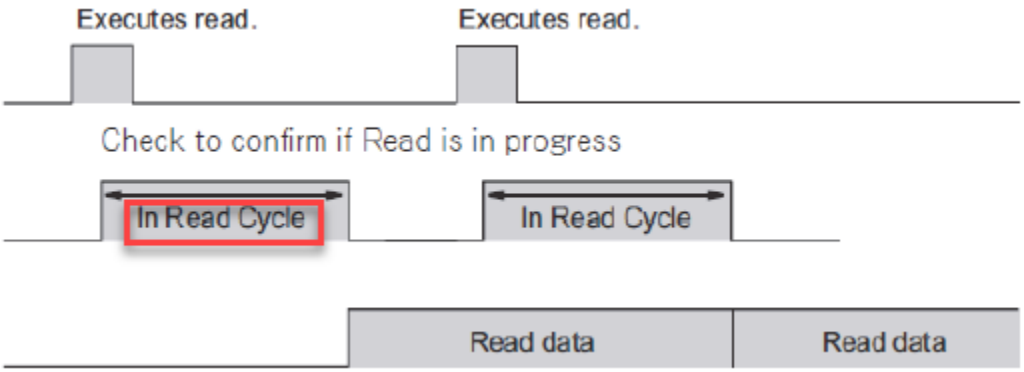
Flow control between PLC and reader

Flow control is handshake implementation between the PLC and the reader, to manage the triggering and reading results in the correct timing and conditions. When we implement such logic on PLC side, we make sure that we trigger the reader only when it is ready and read results only when they are valid.
In order to know the status of the reader in every moment, we monitor specific status signals from the reader. Refer to documentation for details on the available status signals for each one of the Input Assemblies used. When using Small Input, there is no status feedback.

Handshake example when using Big Input Assembly

Big Input assembly provides one status signal which is useful for handshake: **InReadCycle**. When this signal is TRUE, we should not trigger the reader, as then we activate the buffering system (take image and process image simultaneously). The reader can handle this, but it can confuse our PLC logic and we lose track of sent triggers against read results.

See below the timing diagram of a typical read cycle from the Spec:



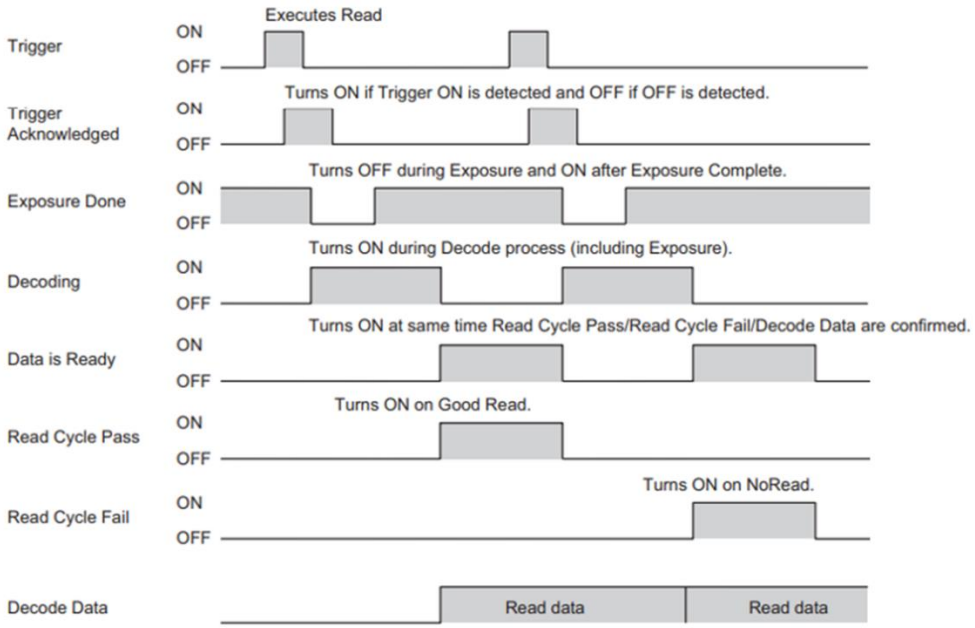
To implement a proper handshake with the reader, we need to implement a state machine on the PLC. See the following example:

State	Description
0	Reader is ready for trigger
1	Trigger sent to reader, waiting for **InReadCycle** to rise
2	**InReadCycle** is TRUE, waiting for **InReadCycle** to fall
3	**InReadCycle** is FALSE, read results and change State to 0

Handshake example when using 1_Decode Input Assembly

When using 1_decode Input assembly, we have several useful status signals from the device, which we can use for the handshake implementation. Some of the signals can be very short, so better to record all signals in the real situation and then check which signals can be safely read on PLC side. Normally The important signals are **DataIsReady**, **TriggerAck** and **Decoding**. Please refer to the documentation for more details.

In the timing diagram below you can see the typical read cycle sequence. After a trigger, the reader is sending **TriggerAck**, then after some decoding time the results will be ready. The reader will signal with **DataIsReady** that the results are valid and ready be read.



See below a simple example for handshake implementation using a state machine:

State	Description
0	Reader is ready for trigger
1	Trigger sent to reader, waiting for **Trigger_Ack**
2	Received **Trigger_Ack**, waiting for **Data_Is_Ready**
3	Received **Data_Is_Ready**, read results and change State to 0

Considerations for handshake implementation:

1. TriggerAck signal exists only when triggering from bit command register over EIP. When the reader is triggered from physical input or serial command, there is no TriggerAck signal.

2. Handshake implementation costs time. So when implementing handshake, note that the inspection rate will be slower (roughly: a quick test using the example handshake, showed a max inspection rate of 100ms instead of 50ms without handshake).

3. EIP refresh rate (RPI setting in Sysmac)

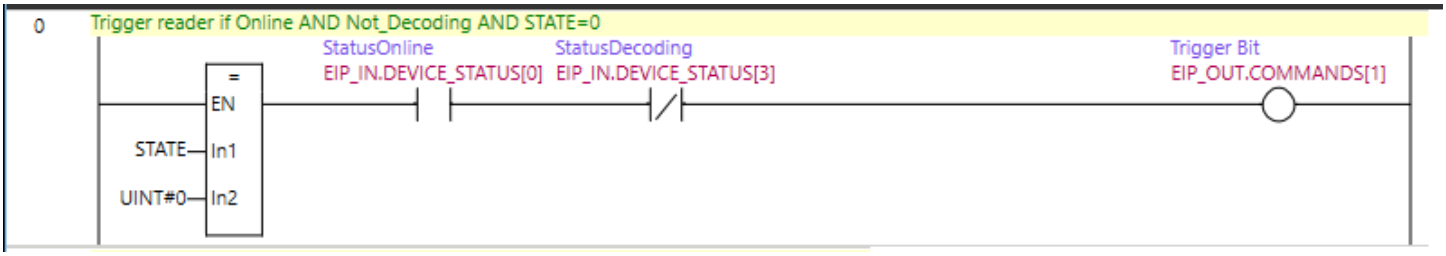
- For working with the reader can be between 10ms – 50ms). For slower refresh rates, note that some of the short status signals will never be sensed by the PLC. In such cases, the implementation should be accordingly.
- When using RPI of 10ms, it takes 20ms for the data to pass back and forth between PLC and reader. Then a thumb rule for ReadCycleTimeout setting on the reader would be: **ReadCycleTimeout <= Trigger Interval - (2 X EIP_RefreshRate)**

Implementation example – ladder logic for flow control

(Implementation using 1_Decode Input assembly)

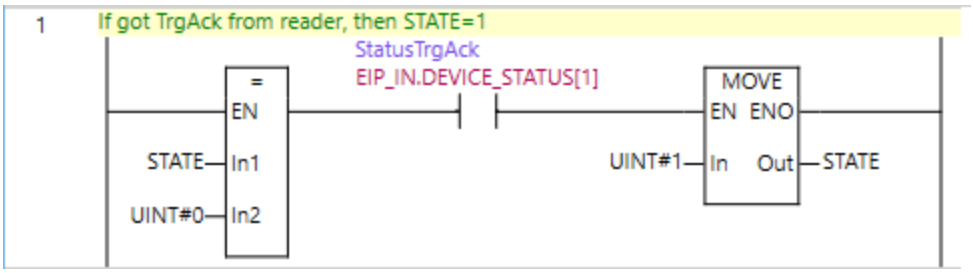
Network 0:
The PLC can send new trigger to the unit only if the following conditions are met:

=>Reader is online
=>Reader is not busy with decoding
=>State = 0



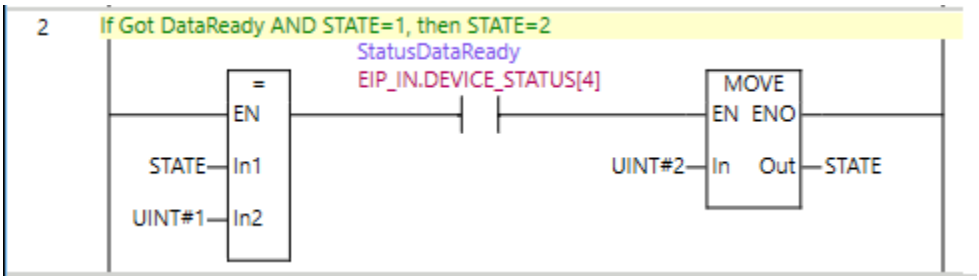
*Note that in this example we trigger from the PLC using Trigger bit command 1. When triggering from Discrete signal, or from Sensor, the safest way is to pass the trigger via the PLC, to create a controlled triggering scheme.

Network 1:
Wait for TriggerAck signal from the reader. When signal is TRUE as well as STATE is 0, then change reader STATE to 1.



*Note that TriggerAck signal exists only when triggering using Trigger bit 1 over EIP. Otherwise, there is no TriggerAck to wait for, so the implementation of such handshake is slightly different. An option is to monitor the Decoding signal instead (StatusBit 3), which goes HIGH when decoding starts, or to monitor the ExposureDone signal which goes LOW when taking a picture.

Network 2:
When DataIsReady signal is TRUE, it means that the decode result can be read from the reader. When TRUE and STATE is 1, change reader STATE to 2.



Network 3:
Read decode results, do whatever is required with the results, and change STATE back to 0.