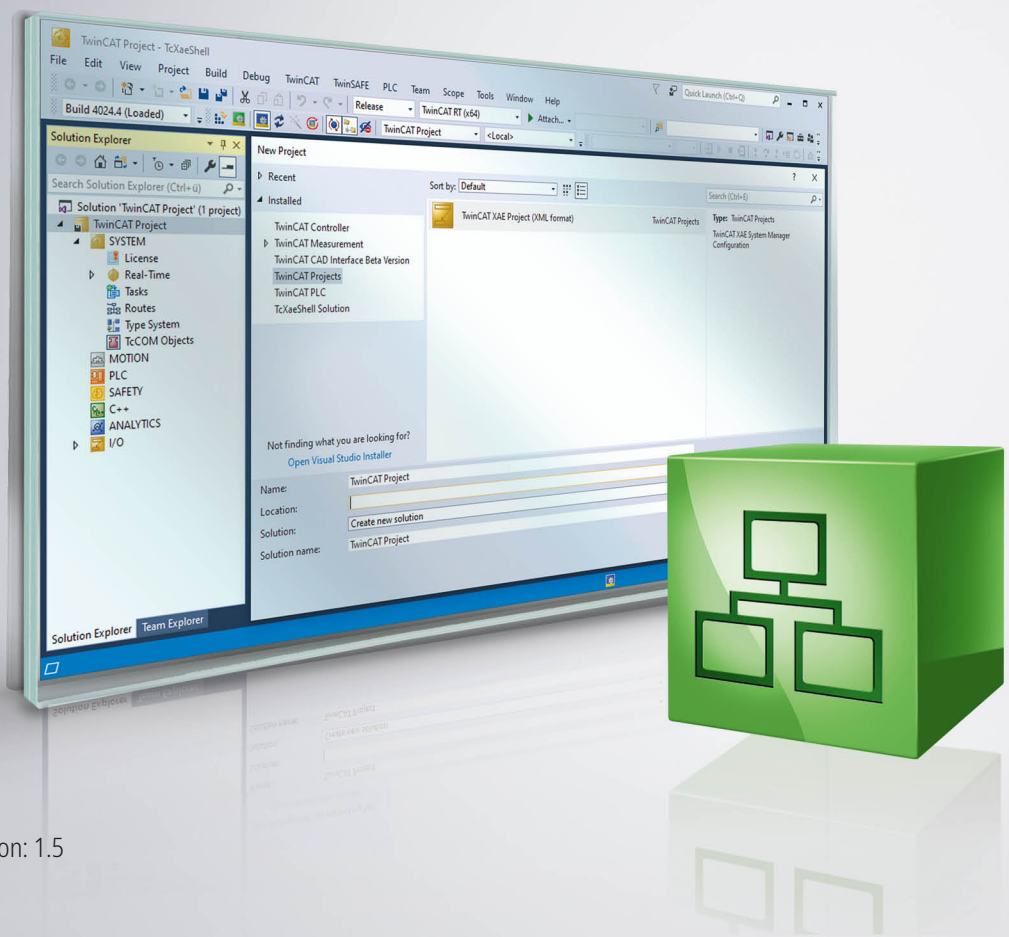
BECKHOFF New Automation Technology

Manual | EN

TF6281

TwinCAT 3 | Ethernet/IP Scanner



2021-08-18 | Version: 1.5

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

2 Overview

The function TF6281 is an EtherNet/IP scanner or master. Here you can connect EtherNet/IP slaves. TF6281 is a software extension that turns an Ethernet interface with Intel chipset into an EtherNet/IP scanner. The real-time driver for the Ethernet interface must be installed for this purpose. The driver is part of the TwinCAT system. This driver is pre-installed on Beckhoff IPCs and can be used on almost all hardware platforms with Intel Ethernet chipset. If you are using a third-party PC, you may need to check or install it.

TC3 function: EtherNet/IP scanner TF6281

Technical data	TF6281							
Requires	TC1200 from build 4022.14, without TC1200 it is not possible to use the full functionality of the function							
Target system	Windows XP, Windows 7/8, Windows CE							
Performance class (pp)	20	30	40	50	60	70	80	90
	–	–	X	X	X	X	X	X

Technical data of the EtherNet/IP scanner

TF6281	4022.0
Remote Nodes (Boxes) [Producer Object counts 1]	128
Client Connections	128
Server Connections	128
CIP Connections	256
Produced Tag	12
Consumed tag for each EtherNet/IP device	12

Ordering information	
TF6281-00pp	TC3 EtherNet/IP scanner

EtherNet/IP

EtherNet/IP (Ethernet Industrial Protocol, EIP) is a real-time Ethernet protocol, which was disclosed and standardized by the ODVA (Open DeviceNet Vendor Association). The protocol is based on TCP, UDP and IPv4.

Further information can be found at www.odva.org or https://en.wikipedia.org/wiki/Ethernet/IP.

3 Requirements

Software

The TF6281 requires **TwinCAT** version **3.1** Build **4022.14** or higher. No further installation is required.

Hardware

To use the TF6281, it is necessary that a real-time driver for the Ethernet interface is installed on the target system.
Beckhoff PC systems are usually preconfigured for the operation of EtherNet/IP devices.

4 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

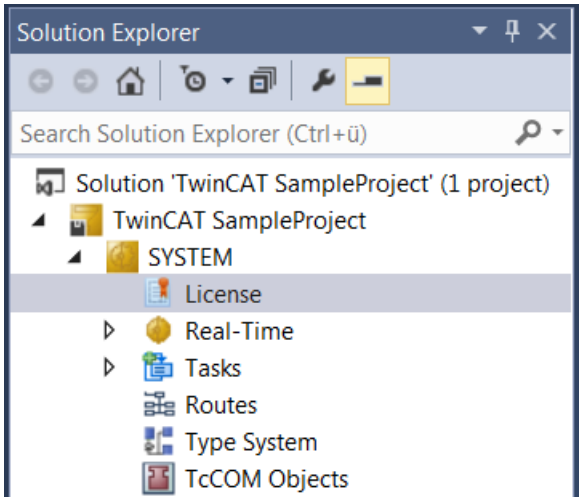
Licensing the full version of a TwinCAT 3 Function

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

Licensing the 7-day test version of a TwinCAT 3 Function

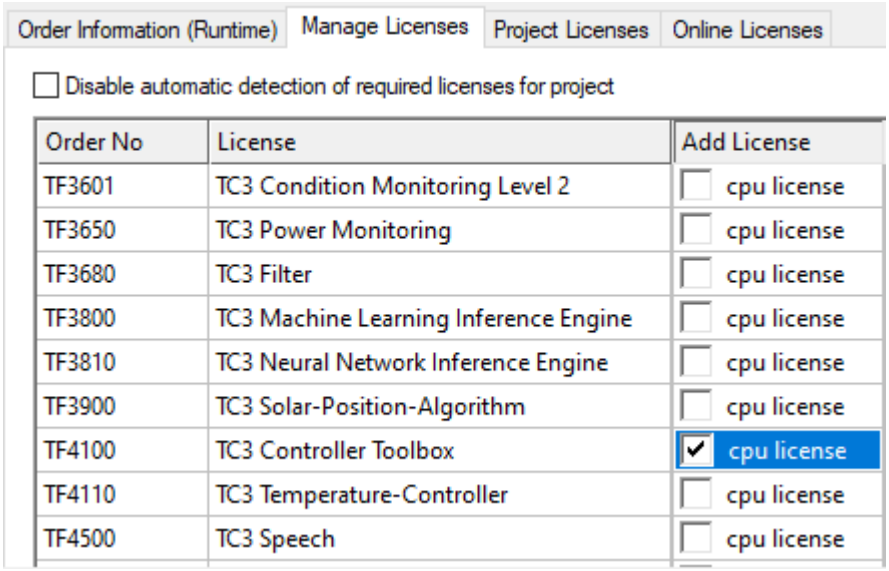
A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.
4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.

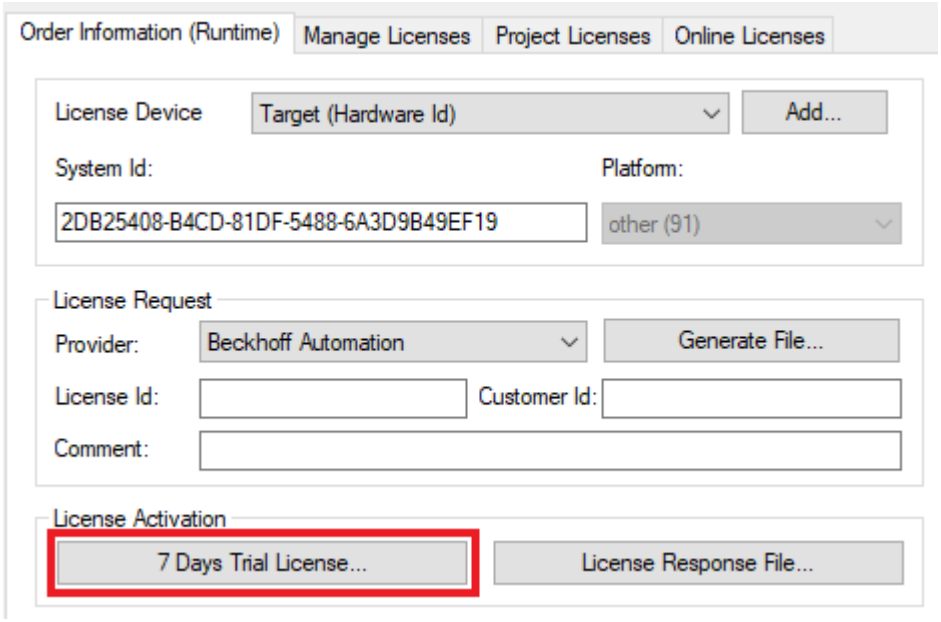


⇨ The TwinCAT 3 license manager opens.

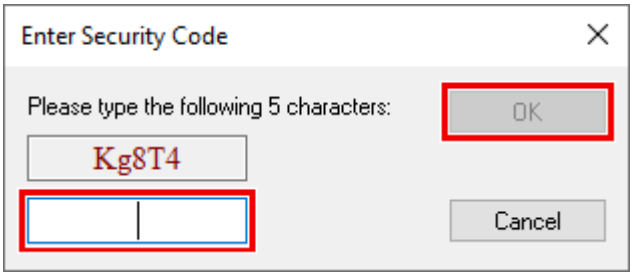
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



6. Open the **Order Information (Runtime)** tab.
   ⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**
7. Click **7-Day Trial License...** to activate the 7-day trial license.



   ⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.

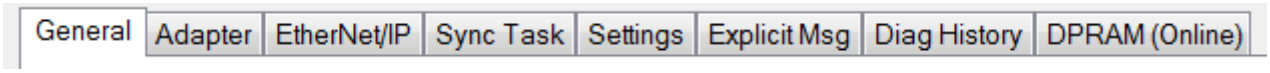


8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.

10. Restart the TwinCAT system.
⇨ The 7-day trial version is enabled.

5 Configuration

The following settings are possible for the EtherNet/IP scanner:



General:
Name and TwinCAT ID of the device

Adapter:
Setting for the Ethernet interface used

EtherNet/IP:
Display of the software version and ADS address of the EtherNet/IP scanner

Sync Task:
Setting indicating which task triggers the EtherNet/IP scanner and the cycle time with which it operates

Settings:
Setting for IP address and other Ethernet-specific services

Explicit Msg:
Only required for Data Table Read/Write (see chapter Data Table Read and Write [▶ 30])

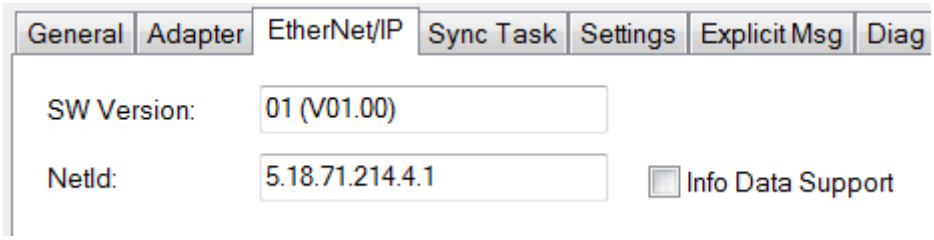
Diag History:
All errors or notes regarding the EtherNet/IP scanner are logged.

DPRAM (online):
Not relevant for the user

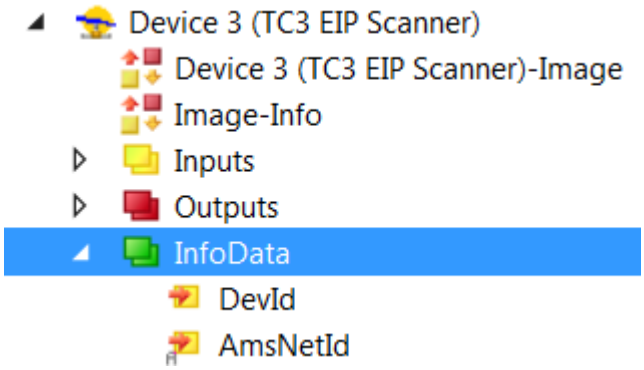
5.1 EtherNet/IP

SW Version: Display of the driver version used for the EtherNet/IP scanner.

NetId: AMSNETID of the EtherNet/IP scanner. This is necessary if the EtherNet/IP-specific function blocks are required.

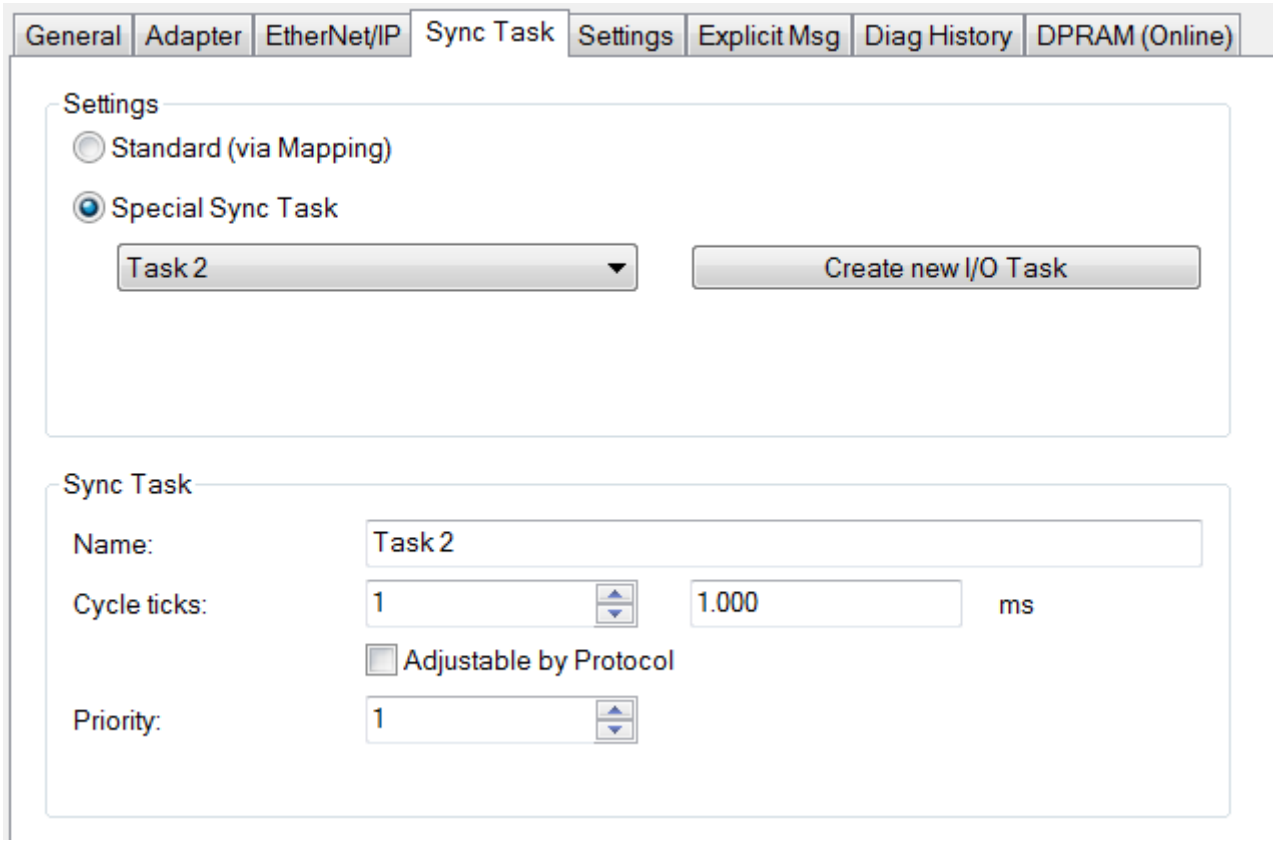


Info Data Support: If this option is activated, the AMSNETID is also available in the TwinCAT tree and can then be linked accordingly.



5.2 Sync Task

The **Sync Task** starts the cyclic call of the EtherNet/IP driver. The Sync Time should be as short as possible, if the processor power allows this. 1 ms is the smallest time base that can be set. It is recommended to create the **Sync Task** via a **Special Sync Task**. If the Sync Task is performed via the mapping of the PLC, a breakpoint in the PLC also causes the EtherNet/IP Task to be stopped, so that the EtherNet/IP devices are no longer addressed. This results in a connection timeout.

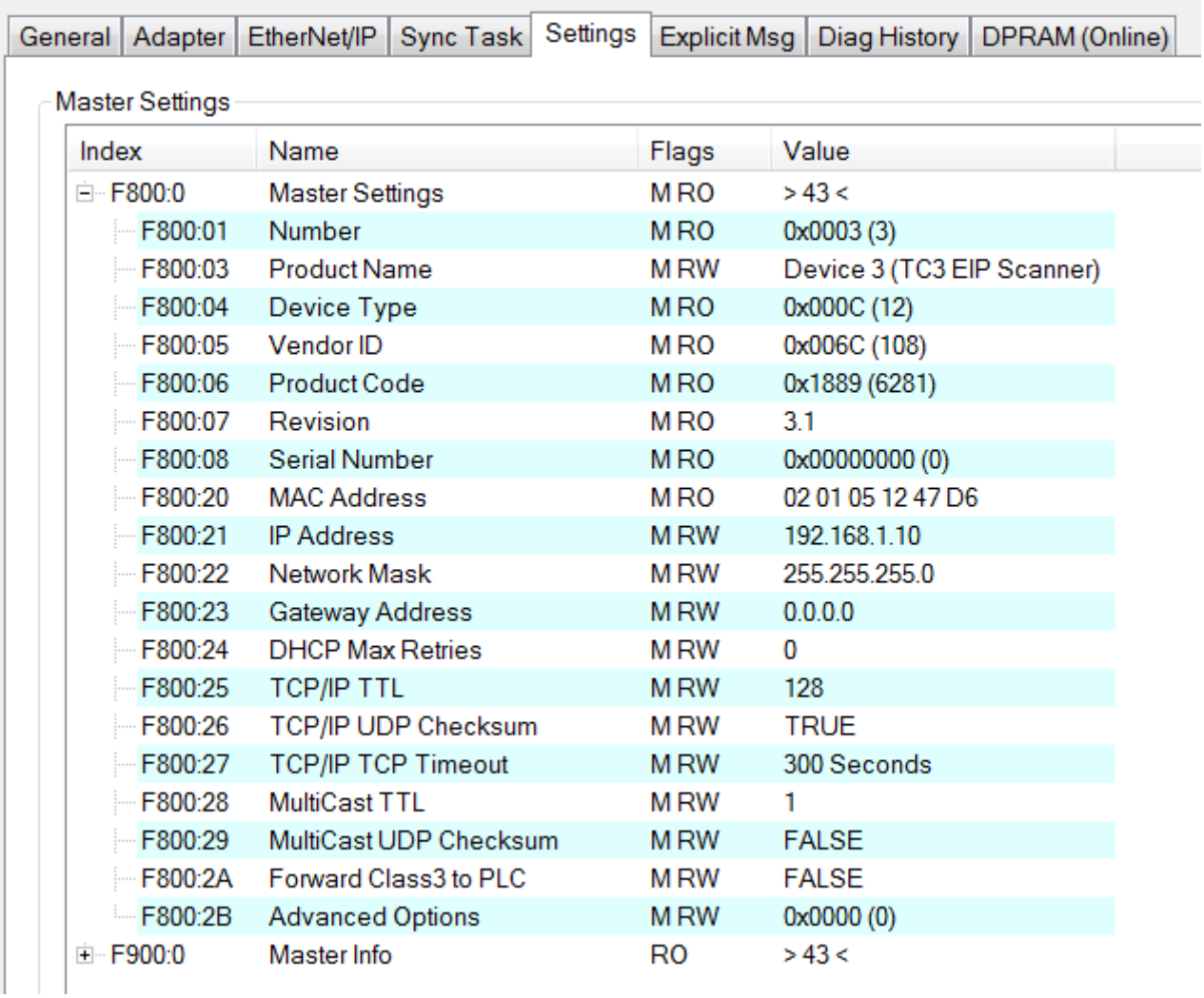


Each slave can run with its own cycle time based on the Sync Task. The **Cycle Time Multiplier** setting is available on each device for this purpose. See chapter Connection of EtherNet/IP slaves [▶ 16].

5.3 Settings dialog

The **Settings** dialog is required for settings such as the IP address and other basic settings. It is divided into two basic settings, which are indicated by the index numbers.
Index `0xF800` contains all the settings used on system startup.
Index `0xF900` contains the actual settings that are valid while the system is running. The actual valid settings are important if basic settings are not made via the **Settings** dialog but have been changed via the PLC.

The IP address is a virtual IP address. In the first step it is unrelated to the IP setting of the operating system (OS). It is recommended to use a different network class than the one selected in the OS. If the IP address of the EtherNet/IP scanner is nevertheless the same as that of the OS, the value `255.255.255.255` should be set under IP address (`0xF800:21`). (See also Firewall recommendation [▶ 15])



Index	Name	Flags	Value
F800:0	Master Settings	M RO	> 43 <
F800:01	Number	M RO	0x0003 (3)
F800:03	Product Name	M RW	Device 3 (TC3 EIP Scanner)
F800:04	Device Type	M RO	0x000C (12)
F800:05	Vendor ID	M RO	0x006C (108)
F800:06	Product Code	M RO	0x1889 (6281)
F800:07	Revision	M RO	3.1
F800:08	Serial Number	M RO	0x00000000 (0)
F800:20	MAC Address	M RO	02 01 05 12 47 D6
F800:21	IP Address	M RW	192.168.1.10
F800:22	Network Mask	M RW	255.255.255.0
F800:23	Gateway Address	M RW	0.0.0.0
F800:24	DHCP Max Retries	M RW	0
F800:25	TCP/IP TTL	M RW	128
F800:26	TCP/IP UDP Checksum	M RW	TRUE
F800:27	TCP/IP TCP Timeout	M RW	300 Seconds
F800:28	MultiCast TTL	M RW	1
F800:29	MultiCast UDP Checksum	M RW	FALSE
F800:2A	Forward Class3 to PLC	M RW	FALSE
F800:2B	Advanced Options	M RW	0x0000 (0)
F900:0	Master Info	RO	> 43 <

Index 0xF800:0 Master Settings
Configuration parameters of the Ethernet/IP scanner

Index 0xF800:1 Number
Box Id

Index 0xF800:3 Product Name
Name of the device

Index 0xF800:4 Device Type
Device type

Index 0xF800:5 Vendor ID
Vendor number

Index 0xF800:6 Product Code
Product code

Index 0xF800:7 Revision
Version

Index 0xF800:8 Serial Number
Serial number (see object `0xF900`)

Index 0xF800:20 MAC Address
MAC address (see object `0xF900`)

Index 0xF800:21 IP Address
Possible values:

- 0: The IP address is assigned dynamically by the DHCP service
- Otherwise: statically assigned IP address.

Index 0xF800:22 Network Mask
Possible values:

- 0: The subnet mask is assigned dynamically by the DHCP service
- Otherwise: statically assigned subnet mask.

Index 0xF800:23 Gateway Address
Possible values:

- 0: DHCP service is used,
- Otherwise: statically assigned gateway address.

Index 0xF800:24 DHCP Max Retries
Possible values;

- 0: Continuous repetition of the DHCP addressing attempts.
- Currently only this mode is implemented, as of: 10-2017

Index 0xF800:25 TCP/IP TTL
"Time to live" value for unicast TCP/UDP communication

Index 0xF800:26 TCP/IP UDP Checksum
function (Unicast)
Possible values:

- 0: UDP checksums disabled,
- 1: UDP checksums enabled

Index 0xF800:27 TCP/IP TCP Timeout
Time switch for inactive TCP connection in seconds

- 0: Time switch disabled

Index 0xF800:28 MultiCast TTL
“Time to live” value for multicast UDP communication

Index 0xF800:29 MultiCast UDP Checksum
function (Multicast):

- 0: UDP checksums disabled
- 1: UDP checksums enabled

Index 0xF800:2A Forward Class3 to PLC
Message forwarding to the PLC
Currently not implemented, as of: 10-2017

Index 0xF800:2B Advanced Slave Options
“Store Category” parameter:

- Bit9=Cat2
- Bit8=Cat1

Index 0xF900 Scanner Info
The current valid settings are displayed here; these can differ from the object 0xF800.

The object 0xF900 displays the active parameters to you.

5.3.1 Firewall setting

The firewall must be enabled, if the EtherNet/IP address is to match the IP address of the operating system (OS). It is advisable to enable the firewall if the IP address of the EtherNet/IP scanner deviates from the IP setting of the operating system.

5.3.2 IP Routing

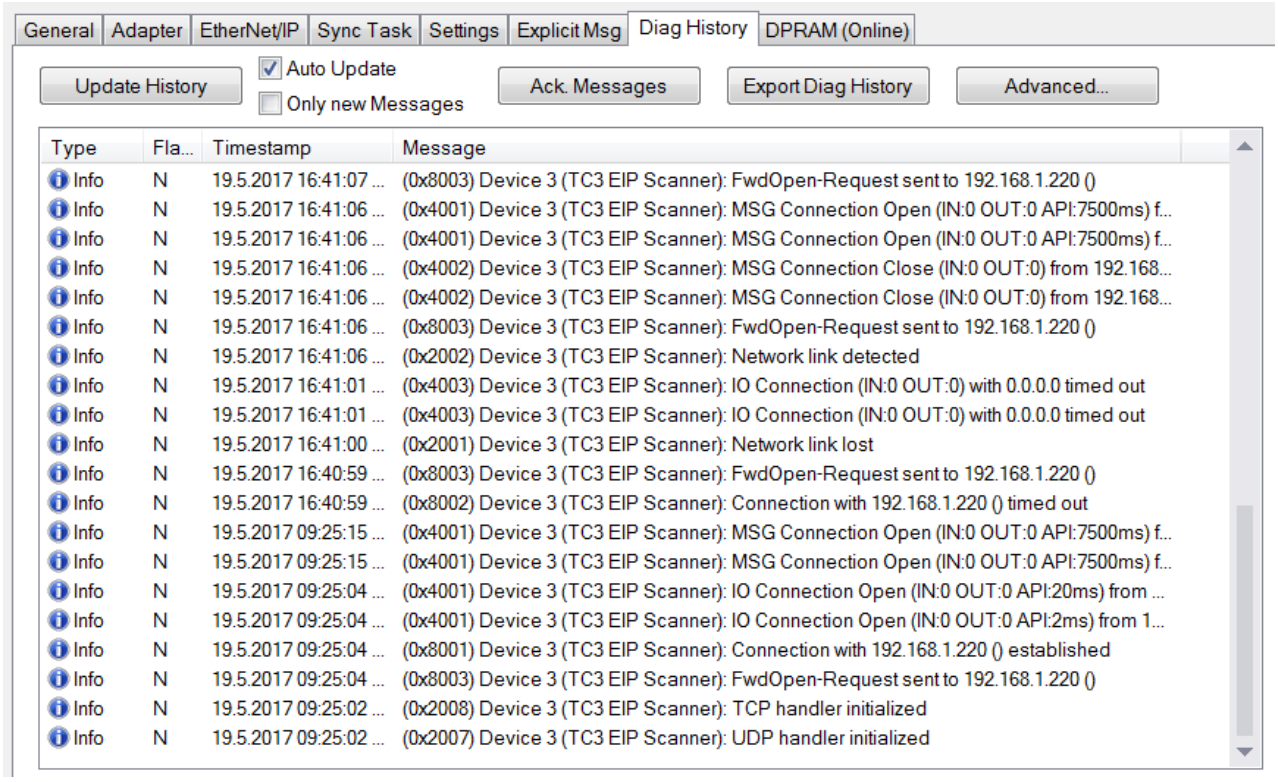
f IP routing is used, the IP address of the OS must be in a different subnet than the IP address of the Ethernet/IP adapter/scanner.
The Regkey can be different depending on the operating system and version, here only as an example, default is "0".
HKEY_LOCAL_MACHINE\ System\ CurrentControlSet\ Services\ Tcpip\ Parameters "IPEnableRouter"

5.4 Diag History

The diagnostic history (**Diag History**) is a tool for monitoring the status of the Ethernet/IP interface and displaying the diagnostic messages with time stamp in plain text.

In addition, information / errors that occurred in the past are logged, in order to enable precise troubleshooting at a later stage. This also applies for errors that only occurred for such a short time that any corresponding messages were not visible.

The diagnostic history is part of the TwinCAT system, where it can be found under Devices, EtherNet/IP in the **Diag History** tab:



5.5 Connecting EtherNet/IP slaves

An EtherNet/IP slave can be integrated as a generic node with EDS (Electronic Data Sheet), or without an EDS file. Not all EtherNet/IP slaves currently available on the market are supported. It should be possible to integrate Ethernet/IP devices that are delivered with an EDS file via the EDS import, provided they are supported by the TF6281. If this is not the case, you can send the EDS file to Beckhoff Support for verification.

If the EDS file can be integrated without errors, communication to the slave should be possible. If you use a slave that can only be integrated via the generic node (i.e. without an EDS file), it is to be assumed that it should also be usable.

The following slaves cannot be used:

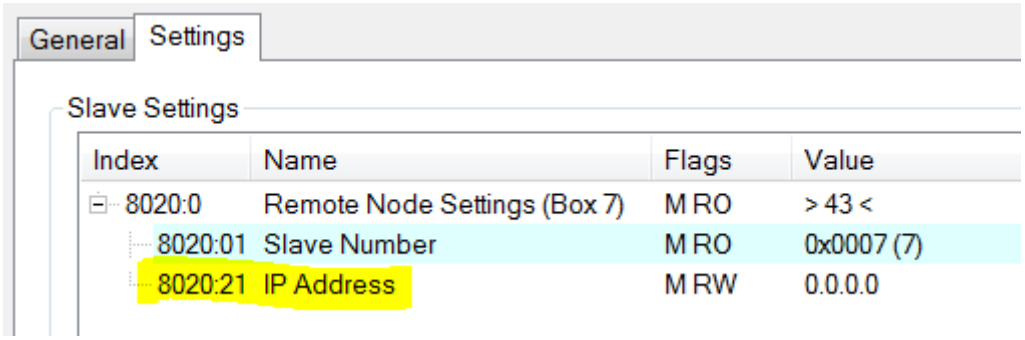
- Slaves that use CIP Sync, CIP Motion or CIP Safety
- Slaves with modular EDS file

Integrating EtherNet/IP slave without EDS file

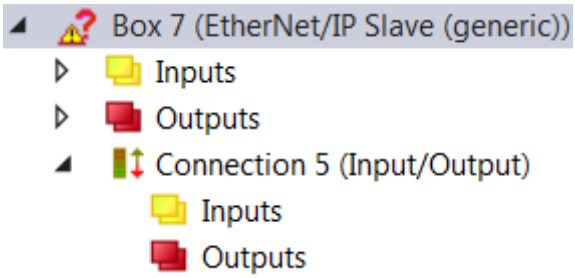
Slaves that do not use an EDS file, or for which the manufacturer does not provide an EDS file, are integrated via a generic node. The following manufacturer information is required for this purpose:

- IP address of the slave
- Maximum RPI time, i.e. the maximum or minimum time with which the slave can work
- The Assembly Instance Number for config, input and output data and their length
- Description of the data

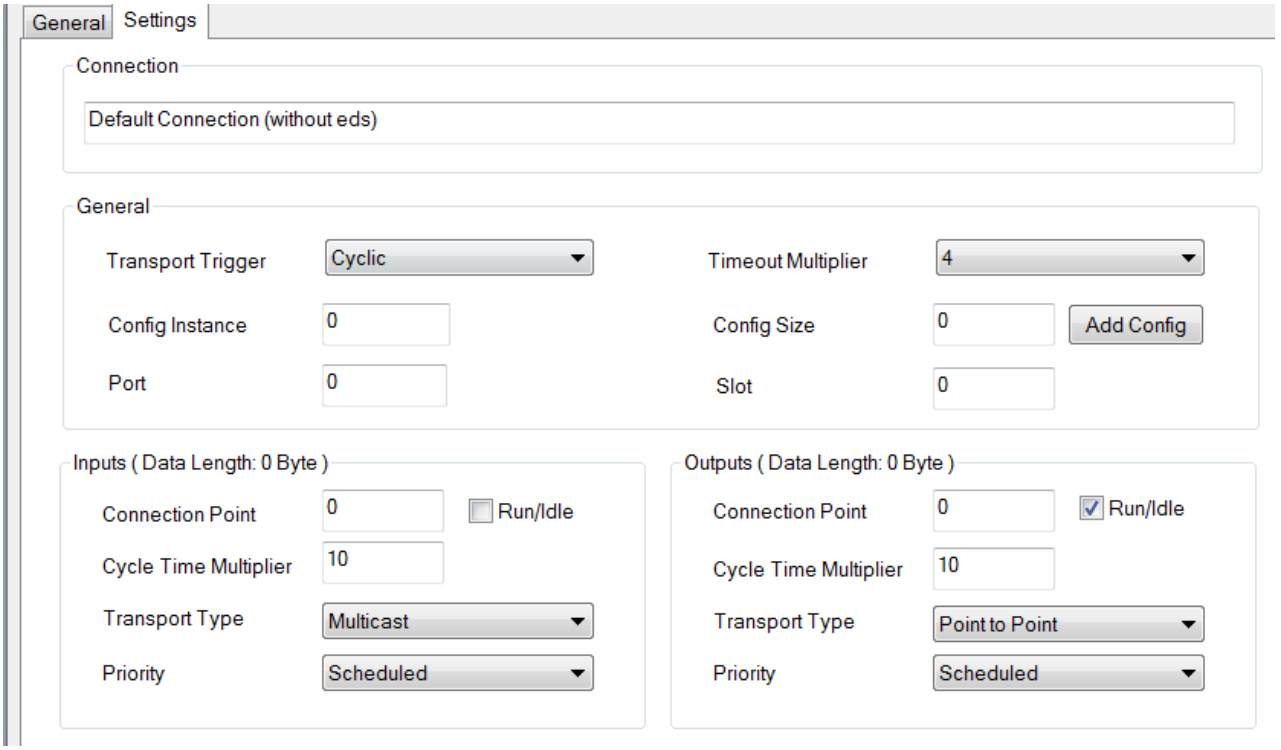
Add a generic node under the EtherNet/IP scanner. As long as you have not specified an IP address, the symbol is identified by a warning and question mark  . Enter the **IP address** under **Settings**.



An "IO Connection" must first be created under the node. This IO Connection contains the inputs and outputs, which can now be created. The variable type is freely selectable, only the size has to match.



Furthermore, the EtherNet/IP specific entries have to be made now.



It is sufficient to specify the values for **Config Instance** and **Config Size**. The **Connection Points** must be created for the inputs and outputs.
The data length results from the length you have previously created. You can verify it in this dialog.

Cycle Time Multiplier

With EtherNet/IP it is allowed to operate the adapters (slaves) with a different cycle time. You can set this individually with the **Cycle Time Multiplier**.

The created **Sync Task** (-> see Sync Task [▶ 13]) specifies the basic cycle time, with which the EtherNet/IP Master is operated.

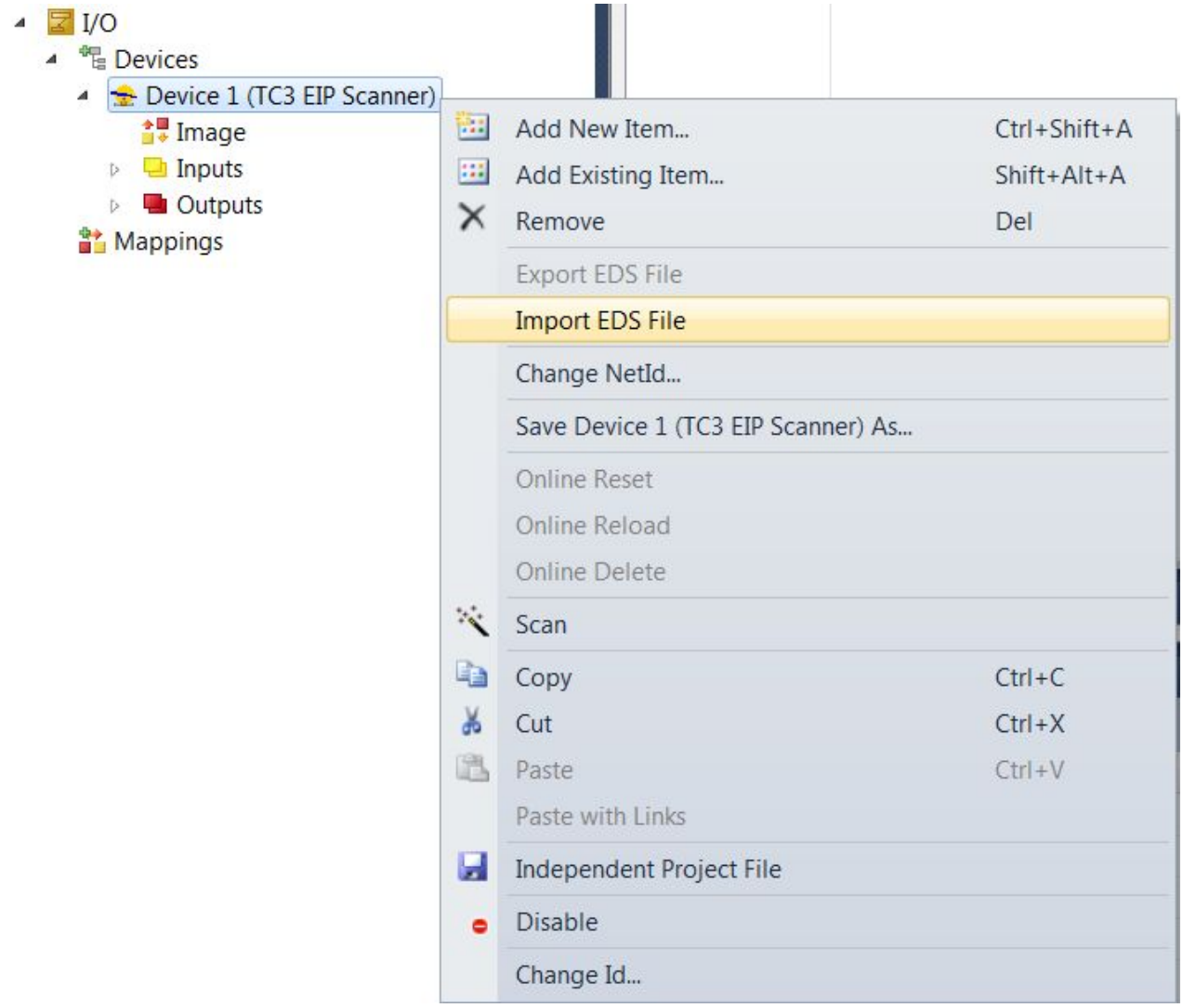
The **Cycle Time Multiplier** is in this case a multiplier of the cycle time in accordance with the inputs or outputs.

The **Timeout Multiplier** is in turn based on the multiplier of the **Cycle Time Multiplier**.

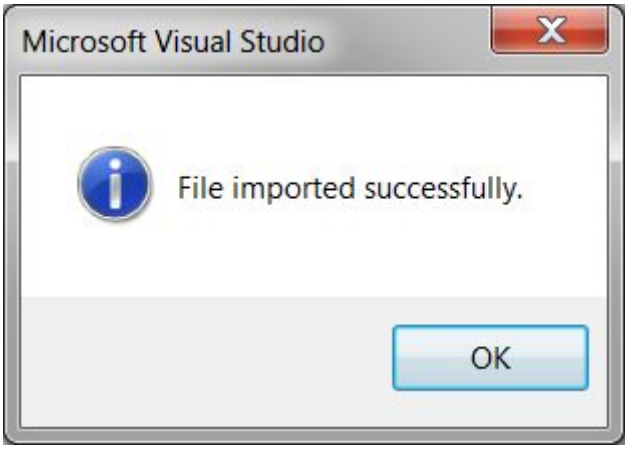
Example: If the **Sync Task** is set to 2 ms and the **Cycle Time Multiplier** is set to 10, the slave is operated with 20 ms. If the connection is interrupted here and the **Timeout Multiplier** is set to 4, the system detects this after 80 ms (20 ms * 4 = 80 ms).

Integration of EtherNet/IP slave with EDS file

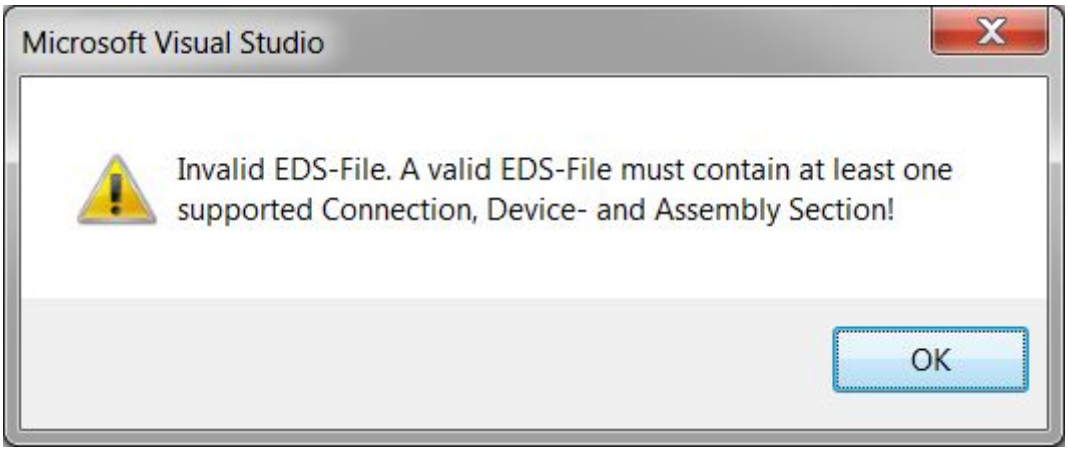
TwinCAT offers the option of integrating EDS files. The **Import EDS File** dialog is used for this purpose.



The files are checked and copied to the directory \TwinCAT\3.1\Config\Io\EtherNetIP after successful import.

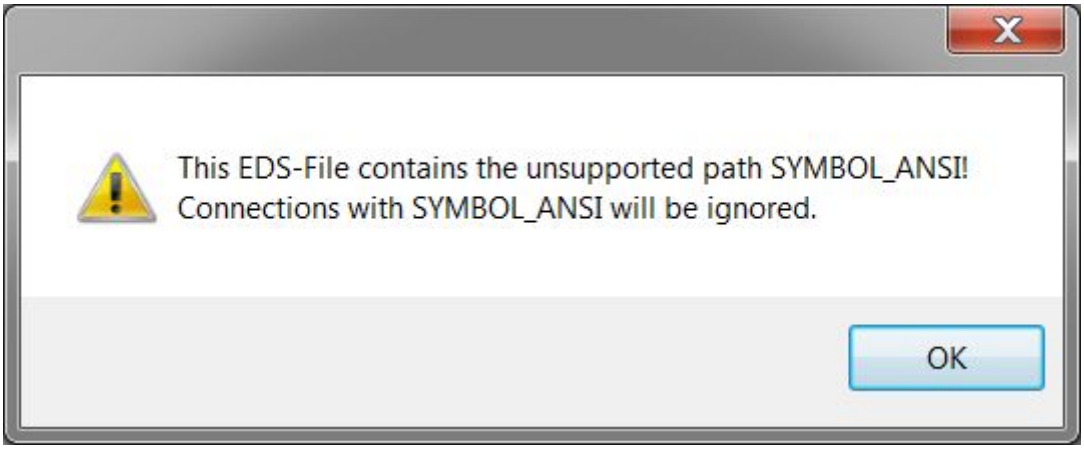


EDS files must have an IO connection, otherwise this error message appears:



These types of devices are not supported by the TF6871 Ethernet/IP scanner.

For EDS files that support symbols, the symbolism is ignored. The symbolism is therefore not usable:



After you have created the slave, the connection must be added. Only the connections described in the EDS file are displayed. Only one connection is allowed.

5.6 PLC to PLC communication

Consumed and Produced tags

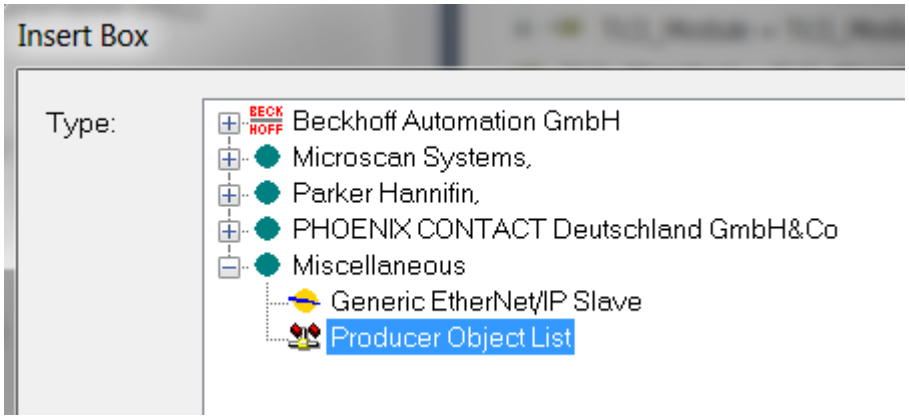
This type of communication is used for PLC – PLC communication. Data is exchanged in real-time between the two controllers. The data exchange takes place via the so-called Consumed and Produced tags. Tag stands for a variable name. The Consumed tag receives the data. The Produced tag provides the data. This means that a Produced tag is created on one controller first, the opposite side that is supposed to receive the data "consumes" the data, hence Consumed tag. This type of communication always requires two EtherNet/IP scanners.
In the following paragraph this is explained by means of a TC3 controller (CX2020 in this case) with the function EtherNet/IP Scanner TF6281 and an Allen-Bradley CompactLogix from Rockwell (RSLogix5000 V20.03.00).

Both sides are described here to set up a communication as described above.

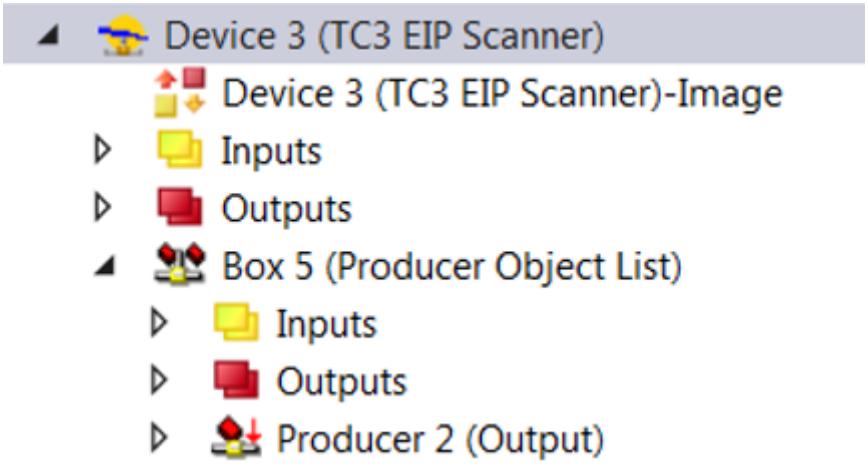
TwinCAT 3.1 Build 4022.x

ProduceTag in TwinCAT

First, the EtherNet/IP scanner is created in TwinCAT (IP address and further settings can be found in the previous chapter Settings dialog [▶ 13]). Right-clicking on the EtherNet/IP Scanner opens a dialog. Select **Add New Item…**.
Then select **Producer Object List**:



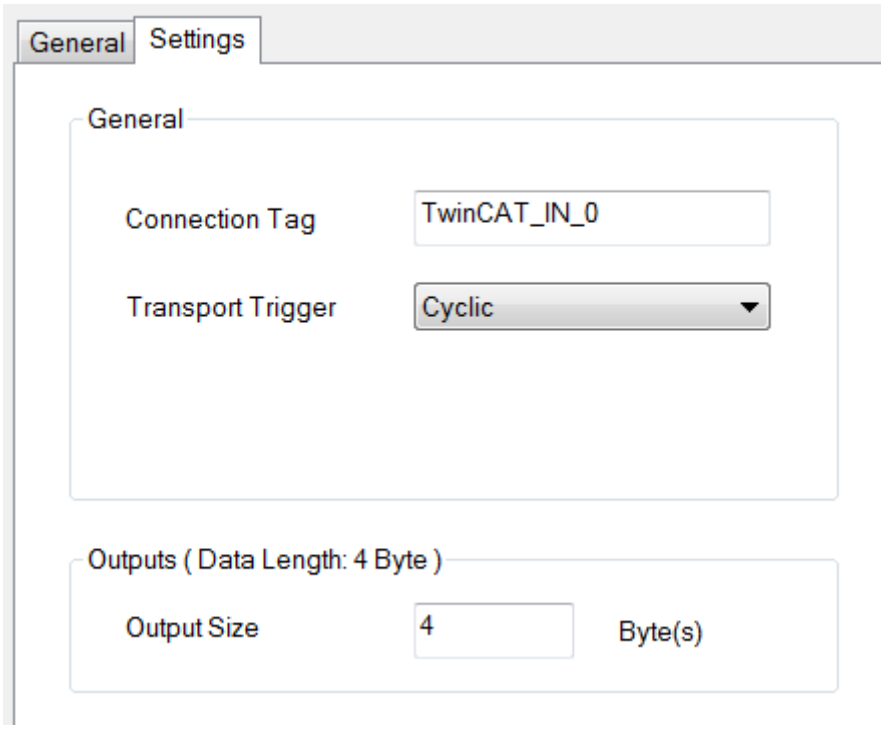
A **Producer Object List** is then created below the scanner. This is available only once, even if the data is sent to more than one controller. Right-click on **Producer Object List** and select **Append Producer Connection**.



Now specify the name of the **Connection Tag**. This must be identical to the name of the consumer.

Then define the number and type of data. It is only possible to use DINT or larger variables.

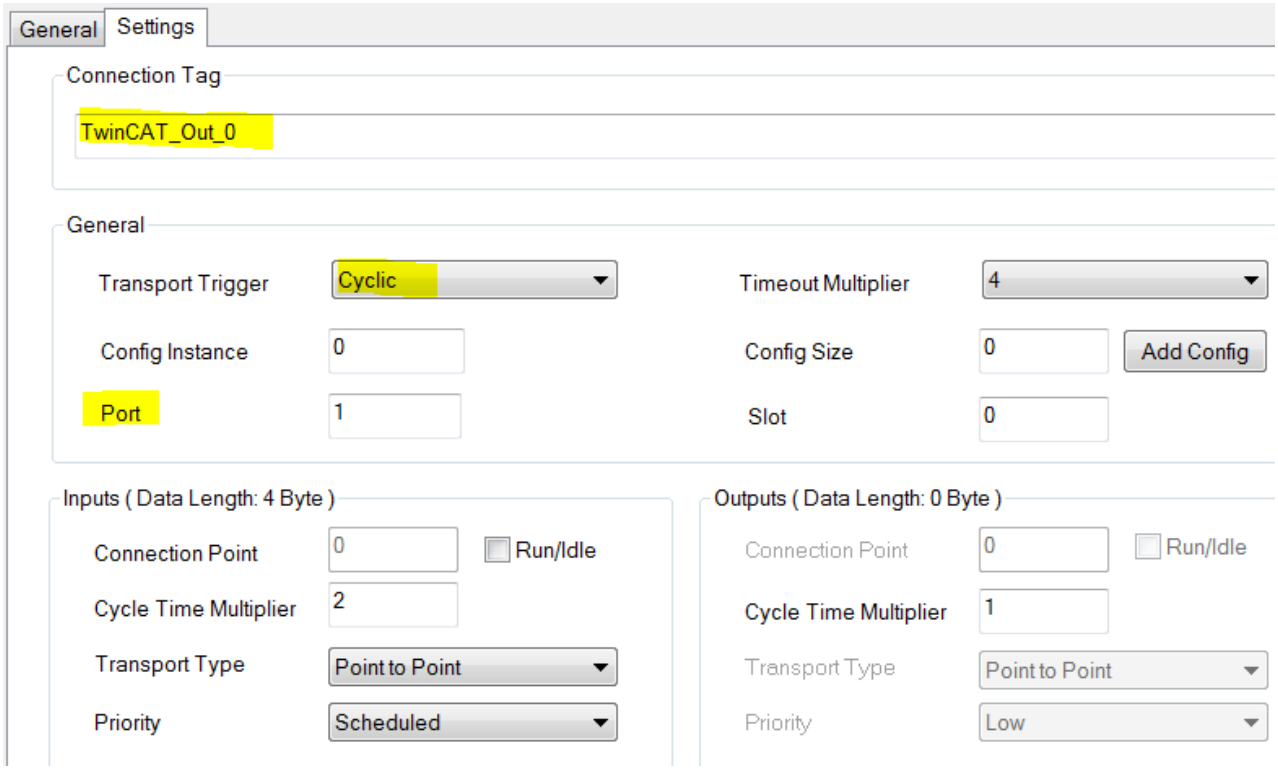
For the further steps, the name **TwinCAT_IN_0** and a variable of type **DINT** were selected. To do this, navigate to the outputs of the **Producer Object** and insert a variable of type **DINT**.



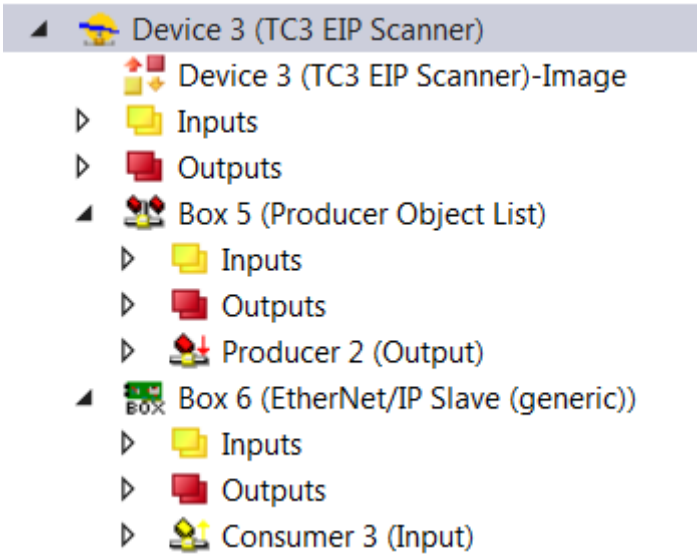
Set the **Transport Trigger** to **Cyclic**. Other operation modes are currently not supported.

Consumer Tag in TwinCAT

Next, create a **Consumer Tag**. To do this, create a **Generic EtherNet/IP Slave** in the EtherNet/IP Scanner. It requires the IP address of the Allen-Bradley CPU. Enter the address and add an **Append Consumer Connection** Consumer tag under the newly created slave. The name is important because it must later be specified as a Produced variable in the Allen-Bradley CPU.
The **Port** is the CPU port on which the variable will be used later. Usually this is **1**.



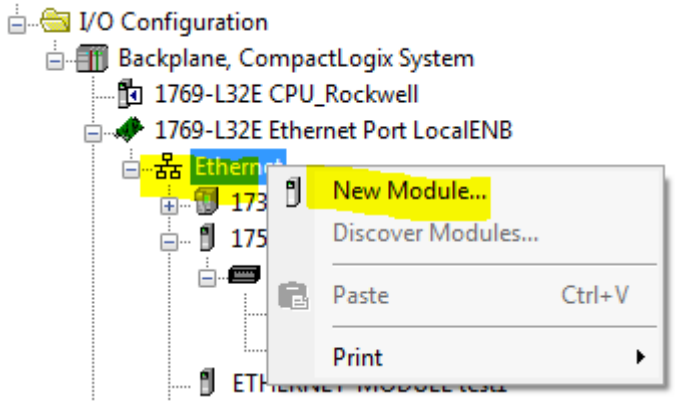
Now you have created a producer in the TwinCAT tree and a consumer for the other EtherNet/IP controller.



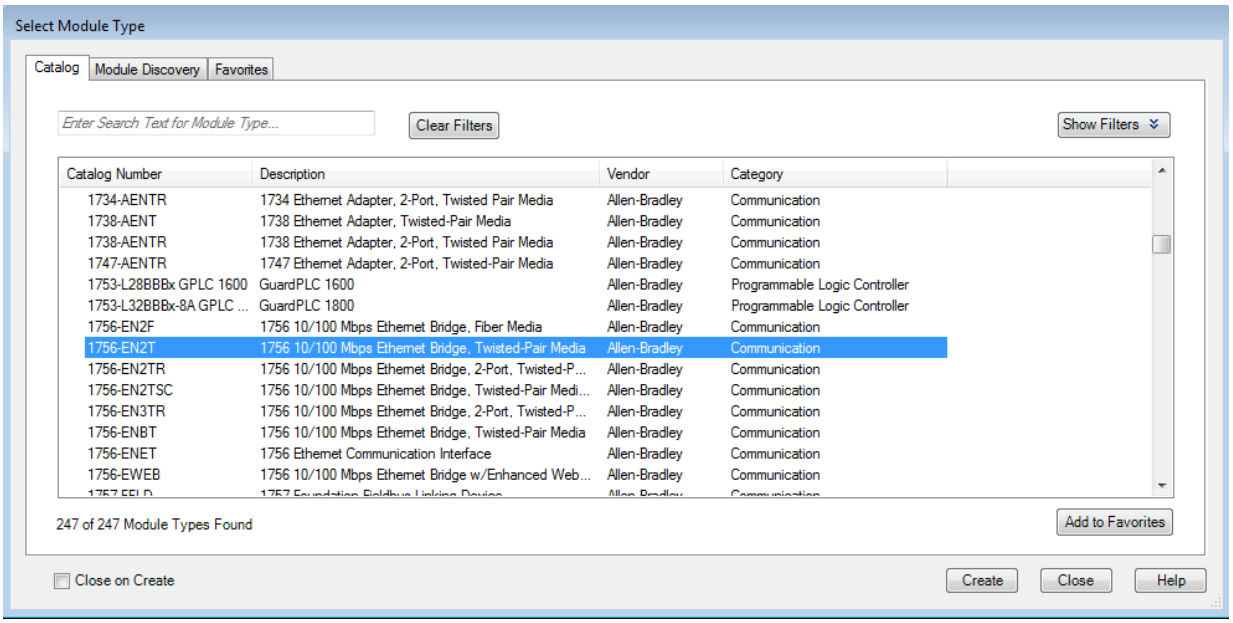
5.6.1 Allen-Bradley CompactLogix

In order to enable PLC – PLC communication using the Consume and Produce tags, an EtherNet/IP controller must be installed at Allen-Bradley (AB). It is not possible to use a Beckhoff controller with AB, therefore an Allen-Bradley controller must be created in the configuration tool.

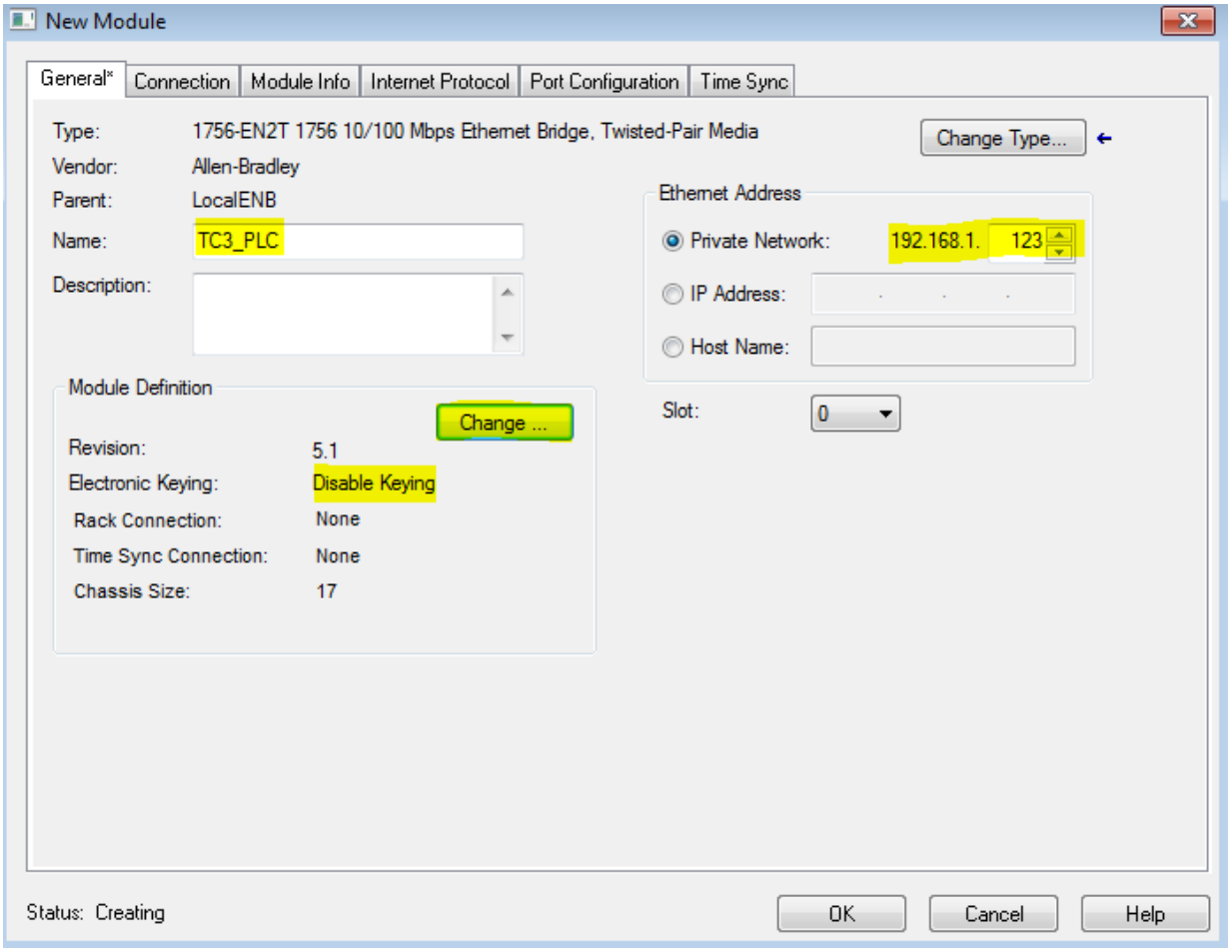
1. Click **Ethernet**; you can create a new module with the right mouse button. Select **New Module…**



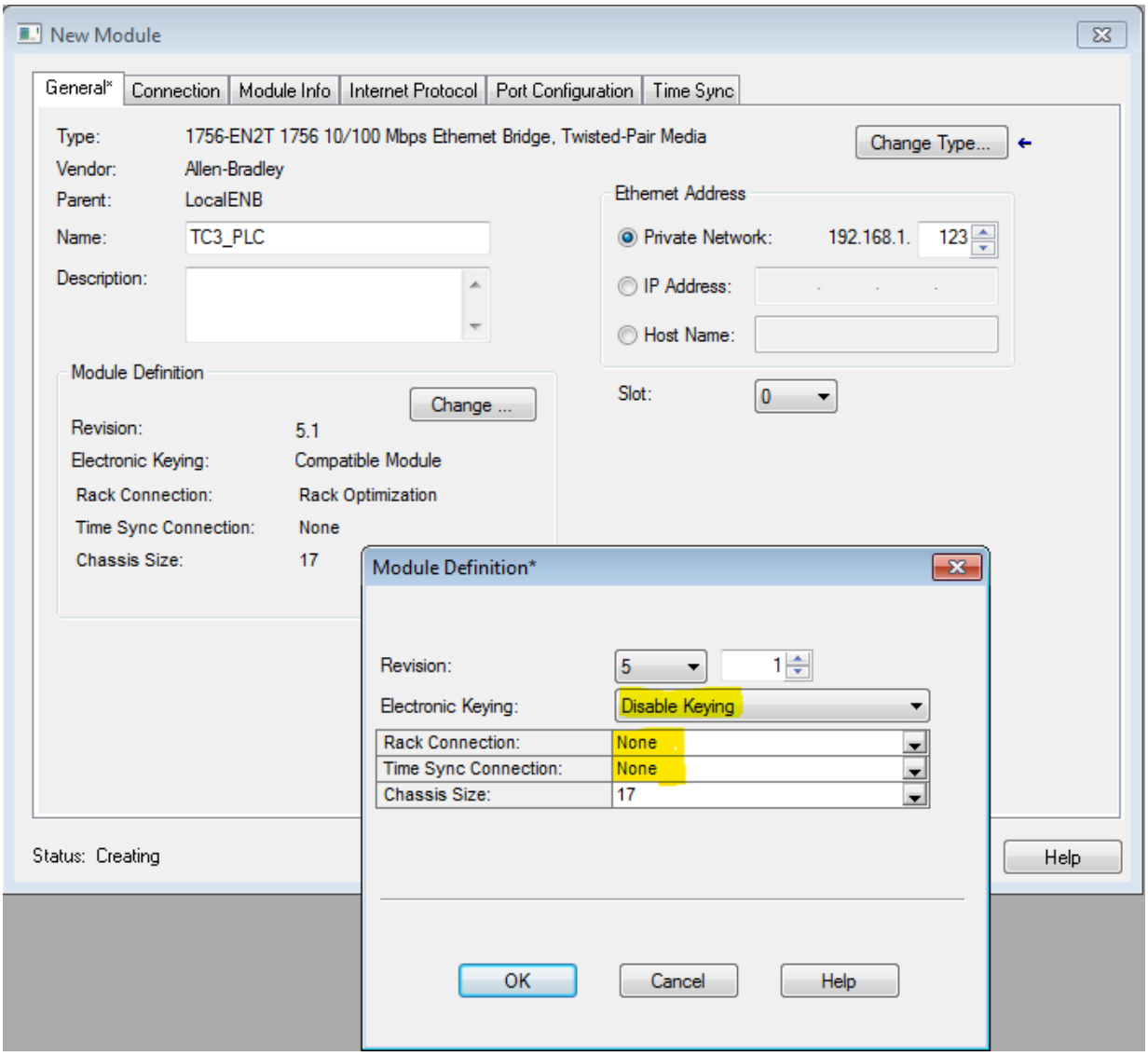
2. Then select a controller, for example 1756-EN2T.



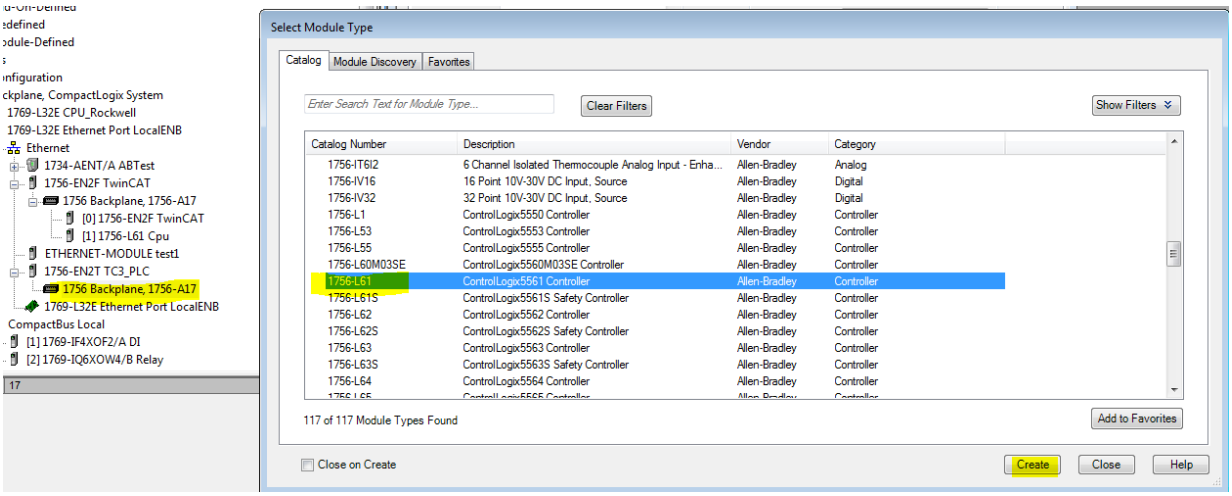
3. Now enter the IP Address of the Beckhoff controller or the IP Address of the Beckhoff EtherNet/IP Scanner. In addition, the controller requires a name.



4. Select **Disable Keying** under **Module Definition**. In addition, select the value “None” for both **Rack Connection** and **Time Sync Connection**.

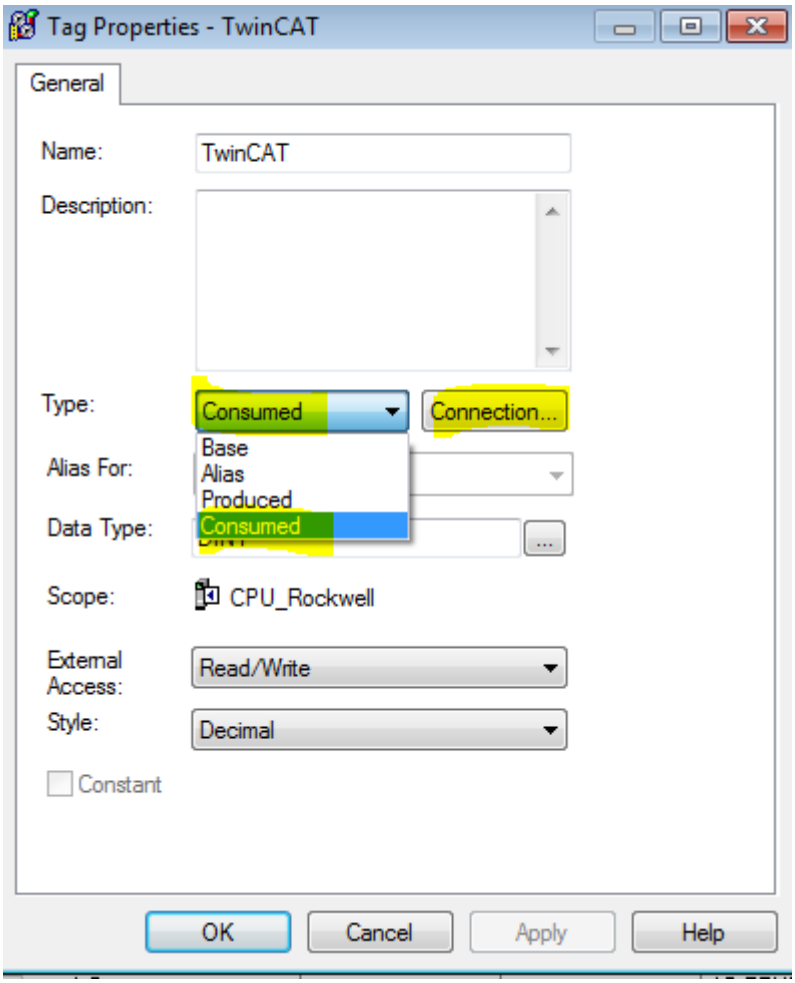


5. Now you have to create a PLC. Select 1756-L61, for example, and click **Create**:

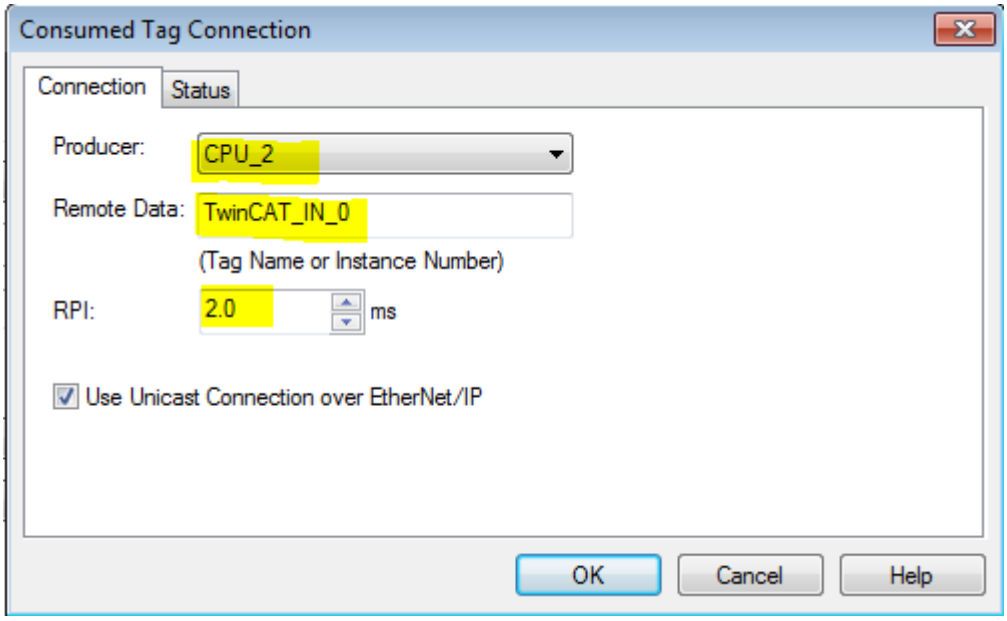


6. Enter a name for the controller, e.g., **CPU_2**; this name is still needed later when you create the **ConsumedTags**.

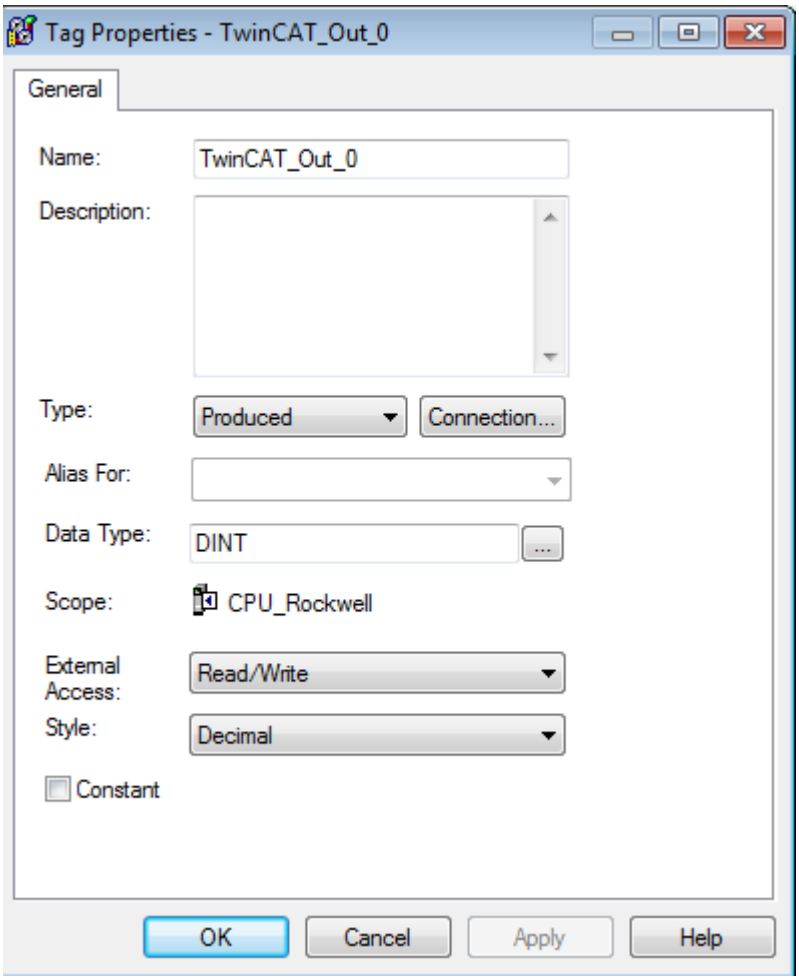
7. Insert a new DINT variable under **Controller Tags**. Create it as type **Consumed**.



8. Now click **Connection**. Select the controller from which you want to receive the data. This requires the name that was assigned during configuration (in this example **CPU_2**). Furthermore, the tag name, which was also assigned in the TwinCAT controller (here: **TwinCAT_IN_0**) and the RPI time are required. The RPI time should always be greater than or equal to the SyncTask of the EtherNet/IP Scanner in TwinCAT.



9. Now insert another **DINT** variable and configure it as **Produced**. It is only important to use the same name as in TwinCAT for the Consumed connection (here **TwinCAT_Out_0**).



5.7 Acyclic communication via CIA

5.7.1 Common Industriual Protocol (CIP)

The Common Industrial Protocol (CIP) is an object-oriented peer-to-peer protocol that enables connections between industrial devices (sensors, actuators) and higher-level devices (controllers). CIP is independent of physical media and the data link layer. CIP has two main purposes: transport of control-oriented data connected to I/O devices, and transport of information relating to the system to be controlled, such as configuration parameters or diagnostics.

CIP uses abstract objects to describe a device. A CIP device consists of a group of objects. Objects describe the available communication services, the externally visible behavior of the device, and a way in which information can be retrieved and exchanged. CIP objects are divided into classes, instances and attributes. A class is a set of objects that all represent the same component. An instance is the current representation of a particular object. Each instance has the same attributes, but possibly with different attribute values. The individual objects are addressed via a node address, which for EtherNet/IP is the IP address, plus a class, instance and attributes.

- Object
  - An abstract representation of a particular component within a product.
- Class
  - A set of objects that all represent the same type of system component. A class is a generalization of an object. All objects in a class are identical in form and behavior, but can contain different attribute values.

- Instance
  - A specific and real specimen of an object.
    Example: Berlin is an instance of the Capital object class.
- Attribute
  - A description of a property or characteristic of an object. Typically, attributes provide status information or control the operation of an object.

(Source: The CIP Networks Library Volume 1: Common Industrial Protocol, Edition 3.22)

The following objects are used internally by Beckhoff and are therefore reserved:

1. Identity Object → Class 0x1
2. Message Router Object → Class 0x2
3. Assembly Object → Class 0x4
4. Connection Manager Object → Class 0x6
5. TCP/IP Interface Object → Class 0xF5
6. Ethernet Link Object → Class 0xF6

5.7.2 Forward Message to AMS Port via CIA

The feature "FwdMsgToAmsPort" enables processing of acyclic requests from Ethernet/IP devices via CIA. To activate the feature, enter the AmsPort of the PLC (in the sample 851) in the slave/master settings (0x8000:2A/0xF800:2A) for the first PLC port.

Requirement:

- Ethernet/IP driver version, V1.23 or higher

The following sample shows how the interface can be implemented in the PLC.

1. ADSRDWRT requests from the Ethernet/IP driver (IDGRP: 0x848180E9 IOFFS: SlaveId (Adapter) or 0xFFFF (Scanner)) to the PLC task are registered as indications and enable their processing. The ADSRDWRTIND function block is used for this purpose.
2. The first entry in the indication registered by the Ethernet/IP driver is a 32-byte (8xULONG) header:

```
TYPE DUT_MsgToAmsPortHeader:
STRUCT
    nServiceCode:UDINT;
    nClassId:UDINT;
    nInstanceId:UDINT;
    nAttributeId:UDINT;
    nReservedId:UDINT;
    nGeneralStatus:UDINT;
    nAdditionalStatus:UDINT;
    nDataLen:UDINT;
END_STRUCT
END_TYPE
```

The same header is also used for the response.

3. The actual read/write data follows directly after the header (nDataLen <> 0 should be set according to the data length). The maximum supported data length is 992 bytes (+ 32-byte header = 1042 bytes).

Table 1: Possible classes/instances/attribute values

	min	max
Class	1	0xFFFF
Instance	1	0xFFFF
Attribute	1	0xFFFF

4. After an indication has been processed a response must be sent to the source device via the ADSRDWRTRES function block.

```
PROGRAM MAIN
VAR
    i                : INT;
    IdxGroup            : UDINT;        //Ethernet/IP-Treiber -> 16#848180E9
    IdxOffset            : UDINT;        //SlaveId (Adapter) bzw. 0xFFFF (Scanner)
    fbAdsRdWrInd        : ADSRDWRTIND;
    fbAdsRdWrRes        : ADSRDWRTRES;
    request                : DUT_IncomingMsgRequest;
    response            : DUT_OutgoingMsgResponse;
    nResponseLen        : UINT;
    nAdsResult        : UDINT:=0;
    nAdsResponsesSent    : UDINT;
    Attributes        : ARRAY [1..4] OF STRING :=['TestReadOnlyAttribute1','TestReadOnlyAttribute2',
'TestReadOnlyAttribute3','TestReadWriteAttribute4'];
END_VAR
```

```

CASE i OF
0:    //check for ADSReadWrite-Requests
fbAdsRdWrInd(
CLEAR:=FALSE ,
VALID=> ,
NETID=> ,
PORT=> ,
INVOKEID=> ,
IDXGRP=> ,
IDXOFFS=> ,
);

IF fbAdsRdWrInd.VALID THEN
IdxGroup:=fbAdsRdWrInd.IDXGRP;
IdxOffset:=fbAdsRdWrInd.IDXOFFS ;
        MEMSET(ADR(request), 0, SIZEOF(request));
        MEMSET(ADR(response), 0, SIZEOF(response));
        nResponseLen:=0;
        //check for Indication Request = Ethernet/IP-driver -> 16#848180E9
        IF IdxGroup = 16#848180E9 THEN
            //check for Indication.datalength >= DUT_MsgToAmsPortHeader
            IF fbAdsRdWrInd.WRTLENGTH >= SIZEOF(request.reqHdr) THEN
                MEMCPY(ADR(request.reqHdr), fbAdsRdWrInd.DATAADDR, SIZEOF(request.reqHdr));
            END_IF
            //check for Indication.datalength > DUT_MsgToAmsPortHeader >>> save additional data
            IF fbAdsRdWrInd.WRTLENGTH > SIZEOF(request.reqHdr) THEN
                MEMCPY(ADR(request.reqData), fbAdsRdWrInd.DATAADDR+SIZEOF(request.reqHdr), fbAdsRdWr
Ind.WRTLENGTH-SIZEOF(request.reqHdr));
            END_IF
            i:=10;
        ELSE
            i:=20;
        END_IF
END_IF

10:    //new Ind from EthIp-Drv received
    response.resHdr.nServiceCode := request.reqHdr.nServiceCode OR CONST.CN_SC_REPLY_MASK;
    response.resHdr.nGeneralStatus := 0;
    response.resHdr.nAdditionalStatus := 0;
    response.resHdr.nDataLen := 0;
    IF request.reqHdr.nServiceCode = CONST.CN_SC_GET_ATTR_SINGLE OR request.reqHdr.nServiceCode = CO
NST.CN_SC_SET_ATTR_SINGLE THEN
        i:=11;
    ELSE
        response.resHdr.nGeneralStatus := CONST.CN_GRC_BAD_SERVICE;
        nResponseLen := SIZEOF(response.resHdr);
        i:=20;
    END_IF

11:    //case decision for request
    CASE request.reqHdr.nClassId OF
    16#1000:    //erlaubte Beispiel Class 0x10000
            CASE request.reqHdr.nInstanceId OF
            16#1:    //erlaubte Beispiel Instance 0x1
                    CASE request.reqHdr.nAttributeId OF    //
Attributes 1-4 erlaubt; only attr 4 is settable
                    1,2,3:    IF request.reqHdr.nServiceCode = CONST.CN_SC_SET_ATTR_SINGLE THEN
                                response.resHdr.nGeneralStatus := CONST.CN_GRC_ATTR_NOT_SETTABLE;
                                nResponseLen := SIZEOF(response.resHdr);
                                i:=20;
                            ELSE
                                i:=12;
                            END_IF
```

```
                    4:        IF request.reqHdr.nServiceCode = CONST.CN_SC_SET_ATTR_SINGLE THEN
                                i:=14;
                              ELSE
                                i:=12;
                              END_IF
                    ELSE
                        response.resHdr.nGeneralStatus := CONST.CN_GRC_UNDEFINED_ATTR;
                        nResponseLen := SIZEOF(response.resHdr);
                        i:=20;
                    END_CASE
            ELSE
                response.resHdr.nGeneralStatus := CONST.CN_GRC_BAD_CLASS_INSTANCE;
                nResponseLen := SIZEOF(response.resHdr);
                i:=20;
            END_CASE
    ELSE
        response.resHdr.nGeneralStatus := CONST.CN_GRC_BAD_CLASS_INSTANCE;
        nResponseLen := SIZEOF(response.resHdr);
        i:=20;
    END_CASE

12:    //GetAttribute
    response.resHdr.nGeneralStatus         := CONST.CN_GRC_SUCCESS;
    MEMCPY(ADR(response.resData), ADR(Attributes[request.reqHdr.nAttributeId]), SIZEOF(Attributes[re
quest.reqHdr.nAttributeId]));
    response.resHdr.nDataLen := INT_TO_UINT(LEN(Attributes[request.reqHdr.nAttributeId]));
    nResponseLen := UDINT_TO_UINT(response.resHdr.nDataLen) + SIZEOF(response.resHdr);
    i:=20;

14:    //SetAttribute
    response.resHdr.nGeneralStatus         := CONST.CN_GRC_SUCCESS;
    IF request.reqHdr.nDataLen <= SIZEOF(STRING)-1 THEN
        MEMCPY(ADR(Attributes[request.reqHdr.nAttributeId]), ADR(request.reqData), request.reqHdr.nD
ataLen);
    ELSE
        response.resHdr.nGeneralStatus     := CONST.CN_GRC_BAD_DATA;
    END_IF
    nResponseLen := SIZEOF(response.resHdr);
    i:=20;

20:    //response to Ethernet/IP-driver
     fbAdsRdWrRes(
NETID:=fbAdsRdWrInd.NETID ,
PORT:=fbAdsRdWrInd.PORT ,
INVOKEID:=fbAdsRdWrInd.INVOKEID ,
RESULT:=nAdsResult ,
LEN:=nResponseLen,
DATAADDR:=ADR(Response) ,
RESPOND:=TRUE );
i:=21;
        nAdsResponsesSent:=nAdsResponsesSent+1;
fbAdsRdWrInd(CLEAR:=TRUE);
21: i:=0;
fbAdsRdWrRes(RESPOND:=FALSE);
END_CASE
```

5.8 Data Table Read and Write

Please note the system requirements

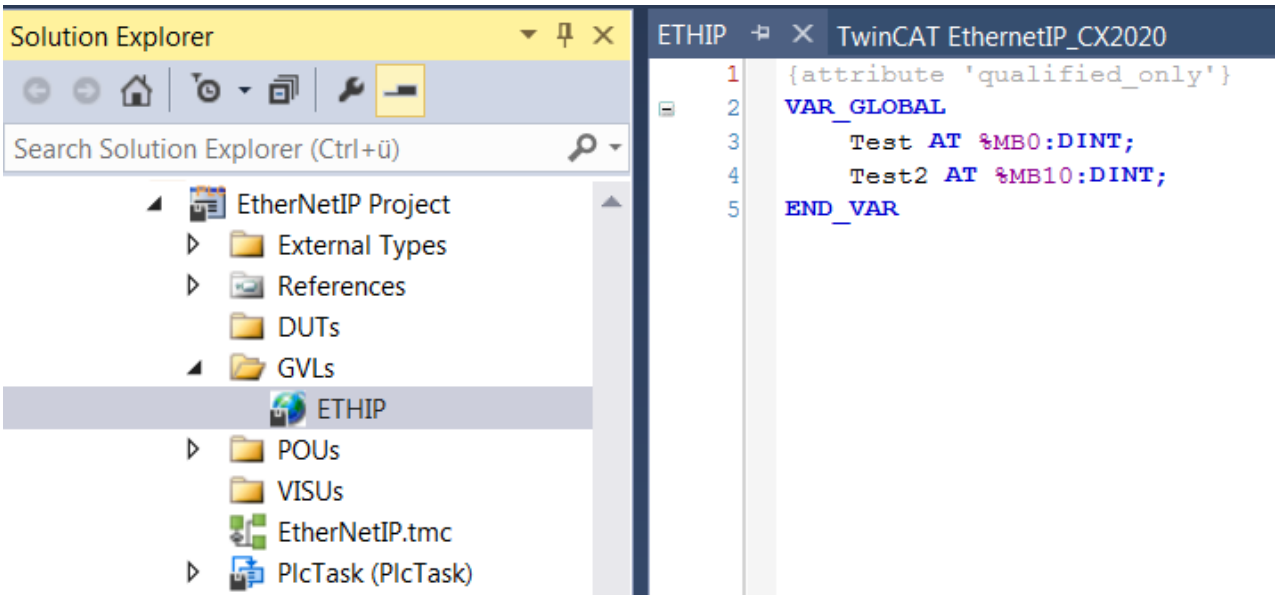
Data Table Read and Write can only be used with the TC1200.

Like the Consumed and Produced tag, this function is used for communication between two EtherNet/IP controllers, with the difference that it is an acyclic communication. This enables data to be exchanged between two controllers which do not have to be transmitted cyclically, such as parameters, recipes or any other data. The data can be structures, arrays or a combination of both. TwinCAT enables data to be read from and written to a controller, and it is also possible to read or write data from TwinCAT using remote control. This is explained below by way of example:

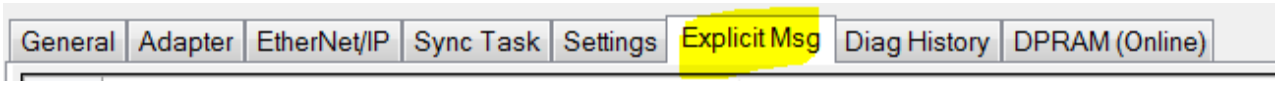
Data that is to be sent or received via this service must be made known in the TwinCAT system. This data must be stored as a global variable in a folder ETHIP and in the flag area. The library Tc2_EthernetIP must also be included. It contains a function block for the DataTable read/write.
The data types must match in both PLCs.

Creating the variables:

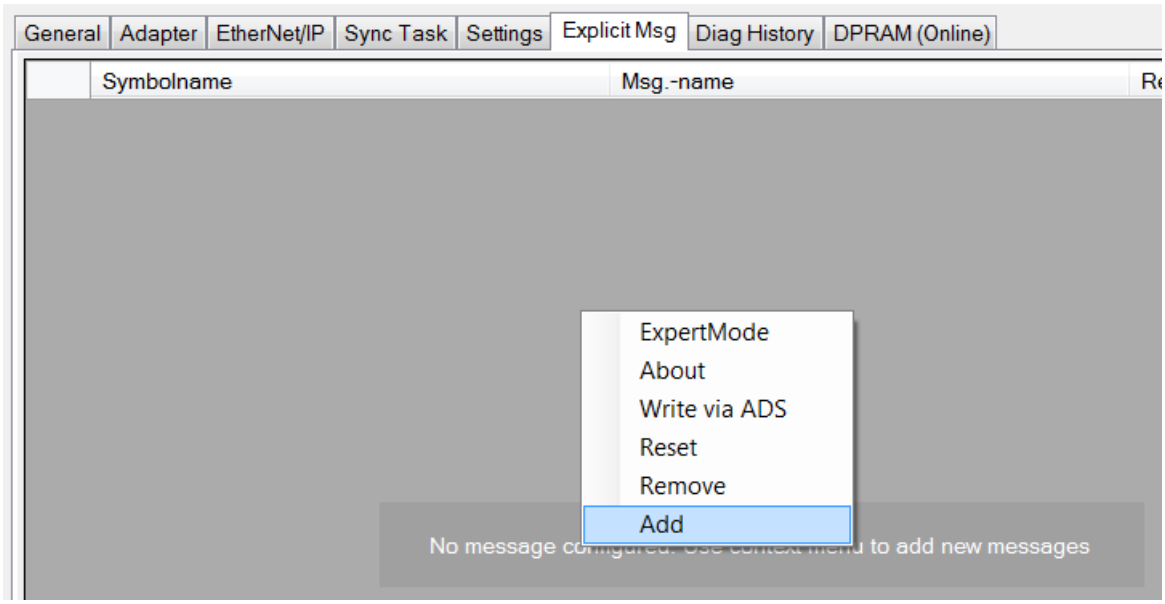
Create a global variable list with the name ETHIP. Now add two variables as shown in the image below. The variables must have a fixed address and lie within the flag area (%MBx, x address). For non-located variables, the internal address could change during an online change; such variables are currently not supported. Now compile the project and switch to the EtherNet/IP scanner.



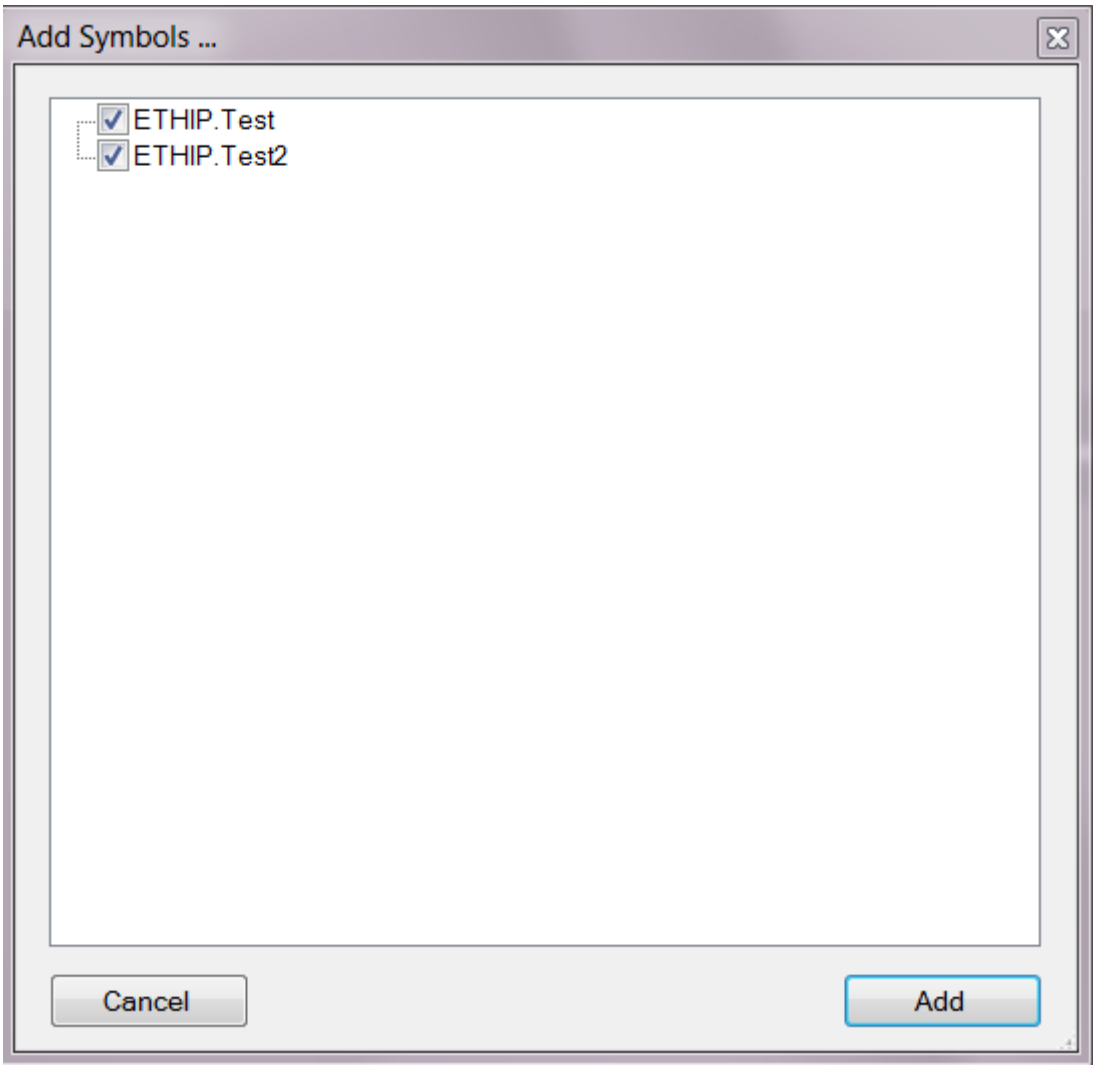
Open the **Explicit MSG** tab:



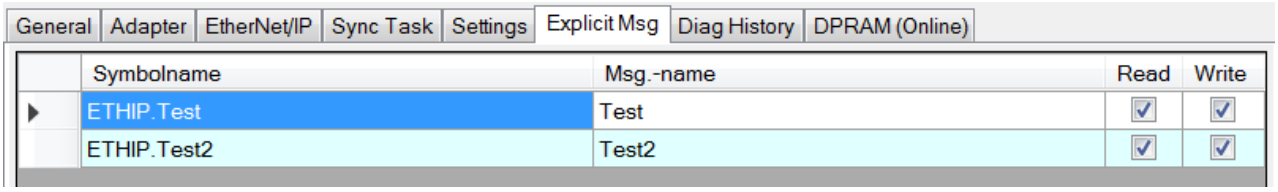
Move the mouse over the empty box, right-click and select **Add** to add the data:



The dialog **Add Symbols …** appears Tick the data you want to use later:



The data are now available in the dialog.



Next, recompile and restart the TwinCAT project. This is necessary if you change the data, e.g. the name, flag, address, type of variable, etc.

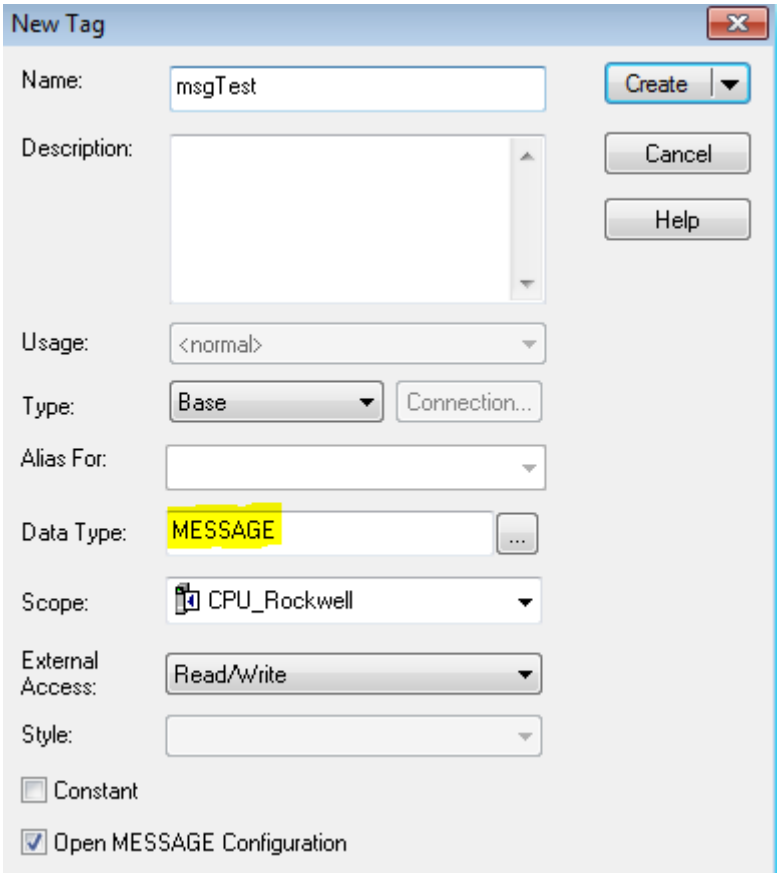
Read a TwinCAT variable from the Allen-Bradley controller

First, enter the TwinCAT controller in the configuration, as for the Consumed and Produced tags; proceed in the same way.

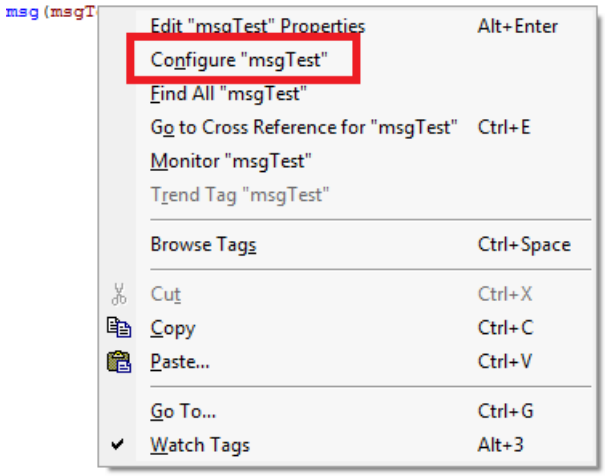
Under **Controller Tags** enter variables **Test** and **iTest**, both as DINT. Now some code has to be written for the Allen-Bradley (AB) controller.

msg(msgTest); (* Program language: Structured Text *)

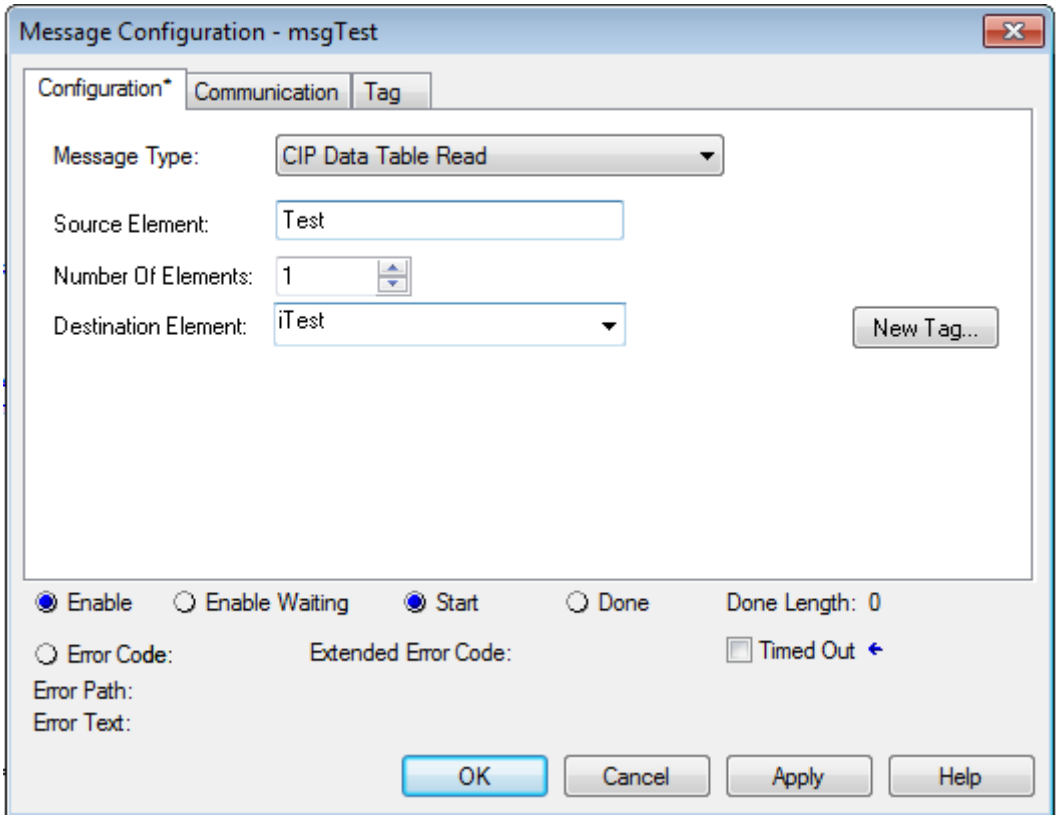
"msgTest "must be of type **MESSAGE**.



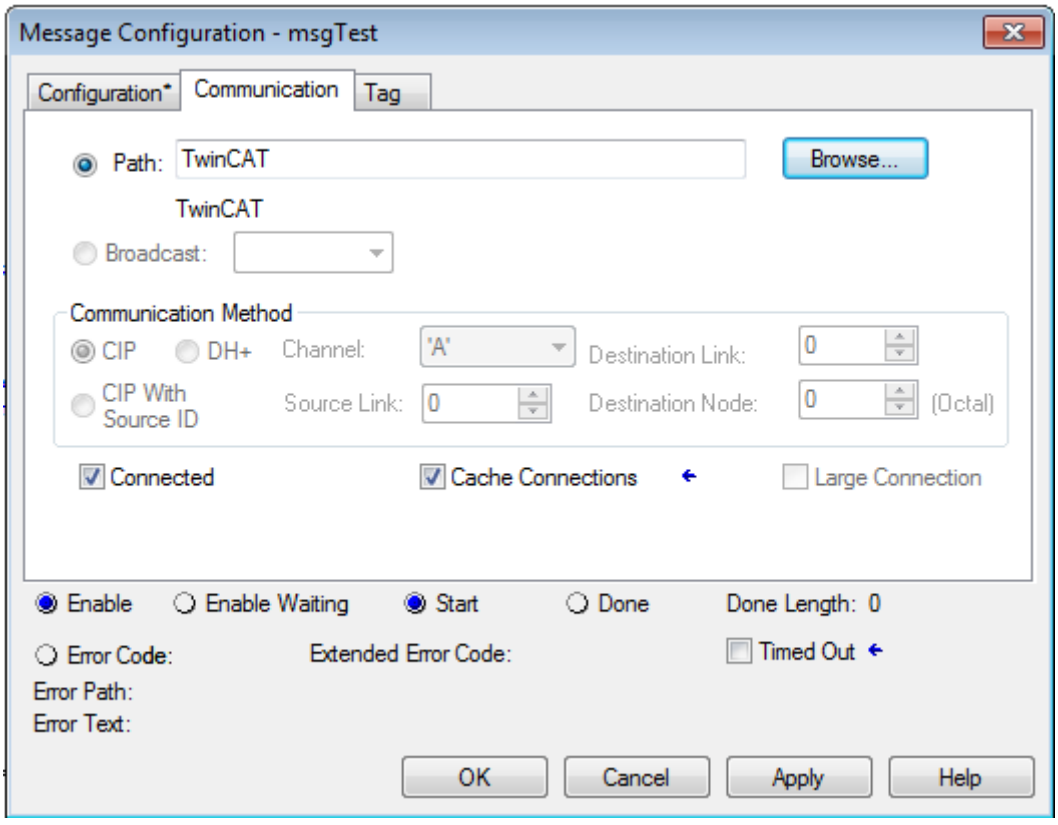
Then click on the **msgTest** variable and configure the function block.



Set the message type to **CIP Data Table Read**. Under **Source Element** enter the name that you used in the TwinCAT project.



Then open the **Communication** tab. Here you set the controller from which you want to read the variable **Test**.

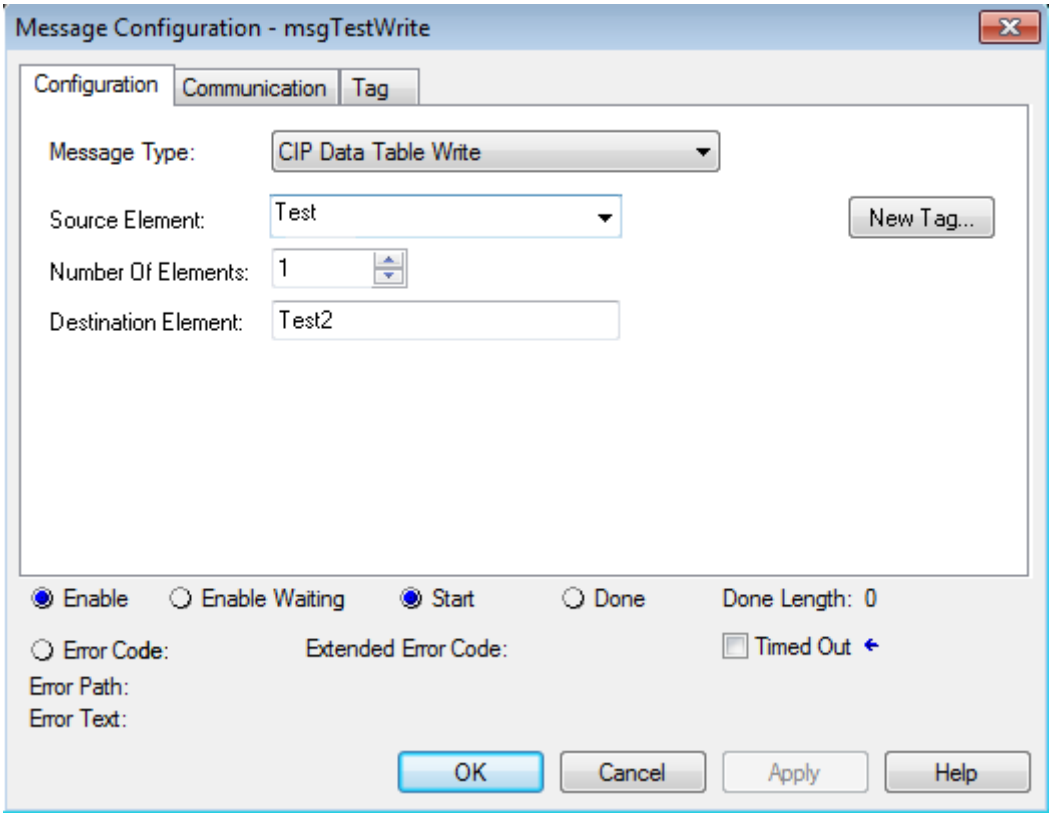


Everything is now prepared for reading the variable.

The variable **Test** is read (on the Beckhoff side) and copied (on the AB side) to the variable **iTest**.

Writing a TwinCAT variable from the Allen-Bradley controller

A similar procedure must be followed when writing. In this case, the MSG command must describe the Data Table Write. The source element is the variable in the Allen-Bradley controller. The **Destination Element** is the TwinCAT variable. Again, select the TwinCAT controller under **Communication**.



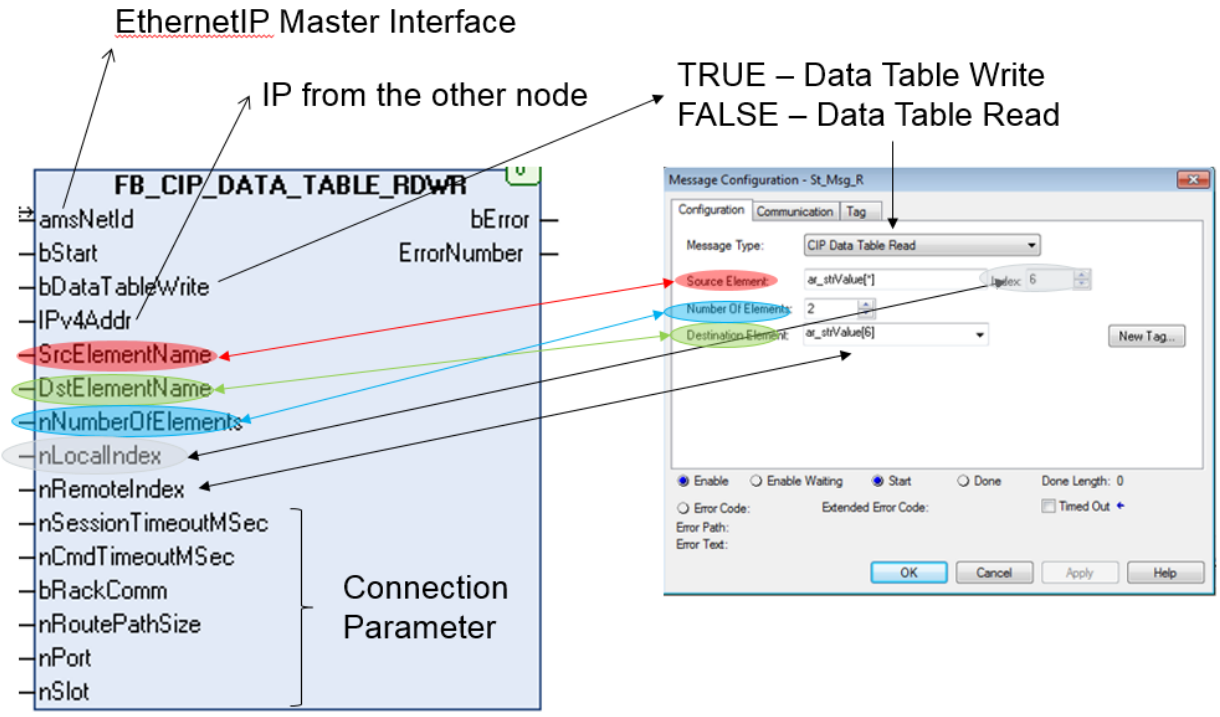
The variable **Test** (on the AB side) is copied to the variable **Test2** (on the Beckhoff side).

Transferring STRING variables

On the Rockwell controller, STRINGS have a different data format than on the TwinCAT controller. The library Tc2_EthernetIP features a data structure called **RSL5K_STRING**, which facilitates the use of STRINGS. You must use this in order to be able to use STRINGS. The corresponding conversions are also available in the library. Only STRINGS with 82 characters or less may be used.

Data Table READ/WRITE from the Beckhoff controller

The PLC function block FB_CIP_DATA_TABLERDWR [▶ 42] is used for DataTableRead/Write from the library Tc2_EthernetIP (see DataTableRDWR). The usage is very similar to that of the AB controller and is shown here as an example:



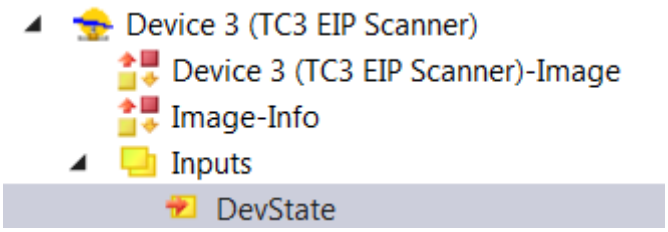
As shown in the image above, a **[*]** placeholder can also be used with ARRAYs on the TwinCAT side. To this end, the ARRAY value is entered with an * in the variable name. The advantage is that only parts or just one element of an ARRAY is read or written. In other words, it is not necessary to read or write the complete ARRAY.
If you use an ARRAY in an ARRAY with * in each case, the index is entered for all **[*]** values. Example **DataARRAY[*].ValueArray[*]**: the index is entered for both.

5.9 Diagnostics

There are several diagnostic options for EtherNet/IP. The diagnosis is divided into two areas, i.e. diagnosis for the scanner (master), and diagnosis for the adapters (slaves) that are connected to the scanner. These are cyclic diagnostic data which can be linked to the PLC. A further diagnosis is available via DiagHistory. Errors in the EtherNet/IP system are logged and can be evaluated for diagnostic purposes.

Diagnosis of the master (scanner)

The scanner diagnosis contains information about the status of the EtherNet/IP scanner. If the value is 0x0000, everything is OK and there is no error.



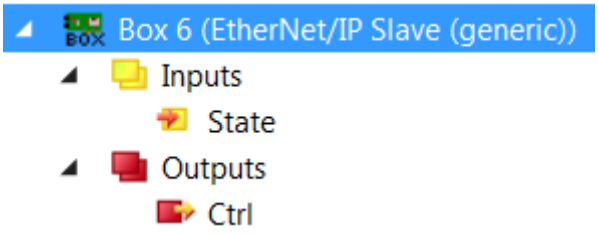
Values that the DevState can take:

0x0001 = Link error
0x0010 = Out of send resources (I/O reset required)
0x0020 = Watchdog triggered
0x8000 = reserved

0x4000 = Master has no valid IP Addr - pending DHCP request
0x2000 = TCP server: unable to listen on local EtherNet/IP Port (44818)
0x1000 = UDP server: unable to listen on local EtherNet/IP Port (44818)

Diagnosis of the slave (adapter)

Each slave has a state and a Ctrl word.



The Ctrl word currently has no purpose. In an error-free state, the value of the state is 0x0000. The state has the following meaning:

0x8000 = Remote Node has no connections
0x4000 = Remote Node is not reachable
0x2000 = TCP Client: initialization failed
0x1000 = UDP Client: initialization failed
0x0X00 = reserved
0x0001 = 1st Connection disconnected
0x0002 = 2nd Connection disconnected
0x0004 = 3rd Connection disconnected
...
0x0080 = 8th Connection disconnected

Producer State

0x8000 = Producer has no valid Producer Objects configured
0x4000 = Producer has no valid IP Addr - pending DHCP request
0x2000 = TCP server: unable to listen on local EtherNet/IP Port (44818)
0x1000 = UDP server: unable to listen on local EtherNet/IP Port (44818)
0x0001 = 1st Connection disconnected
0x0002 = 2nd Connection disconnected
0x0004 = 3rd Connection disconnected
...
0x0800 = 12th Connection disconnected

Consumer State

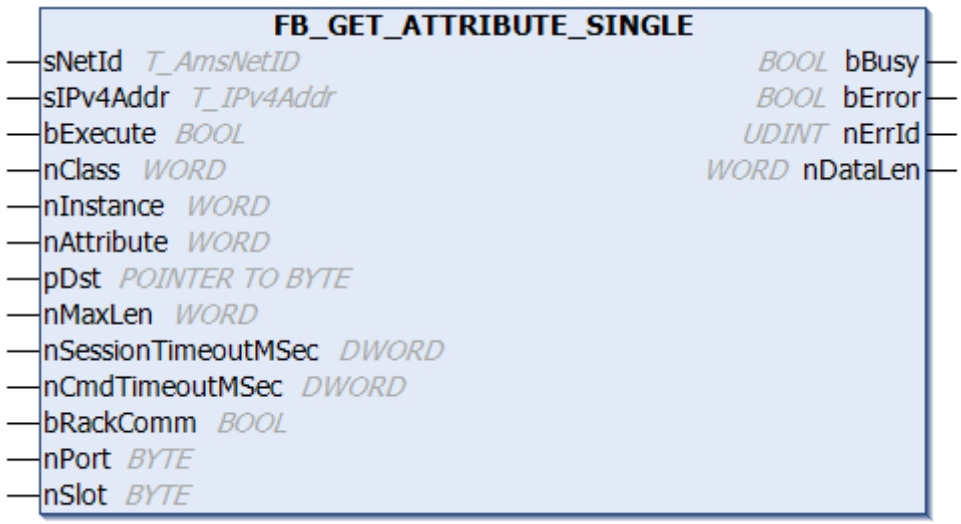
0x0X00 = reserved
0x0001 = 1st Connection disconnected
0x0002 = 2nd Connection disconnected
0x0004 = 3rd Connection disconnected
...
0x0800 = 12th Connection disconnected

6 PLC API

The TwinCAT function blocks can only be used in conjunction with the TC1200. The library Tc2_EthernetIP can be found under **Communication**. It is part of the TC1200 TwinCAT installation.

6.1 Function blocks

6.1.1 FB_GET_ATTRIBUTE_SINGLE



The function block FB_GET_ATTRIBUTE_SINGLE enables reading of parameters from an EtherNet/IP device.

Service code: `0x0E`

VAR_INPUT

```
VAR_INPUT
    sNetId              : T_AmsNetID;
    sIPv4Addr           : T_IPv4Addr;
    bExecute            : BOOL;
    nClass              : WORD;
    nInstance           : WORD;
    nAttribute          : WORD;
    pDst                : POINTER TO BYTE;
    nMaxLen             : WORD;
    nSessionTimeoutMSec : DWORD;
    nCmdTimeoutMSec     : DWORD;
    bRackComm           : BOOL;
    nPort               : BYTE;
    nSlot               : BYTE;
END_VAR
```

sNetId: AMSNetId of the TwinCAT EtherNet/IP scanner through which the command is to run

sIPv4Addr: IP address of the target device

bExecute: A positive edge starts the command

nClasss: Class number of the CIP service

nInstance: Instance number of the CIP service

nAttribut: Attribute number of CIP service

pDst: Pointer to the variable to which the value is be copied (the pointer is determined with ADR)

nMaxLen: Size of the variable to which the pointer pDst points (determined with SizeOf)

nSessionTimeoutMSec: Timeout for the session; the default is 30 seconds

nCmdTimeoutMSec: Timeout for the command; the default is 7.5 seconds
bRackComm: TRUE if the CPU is modular, i.e. a CPU with a rack design, for example a CompactLogix

nPort: Port number of the CPU (the TF6281 currently only supports port 1)

nSlot: Slot number if the CPU is not plugged into slot 0

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    nDataLen  : WORD;
END_VAR
```

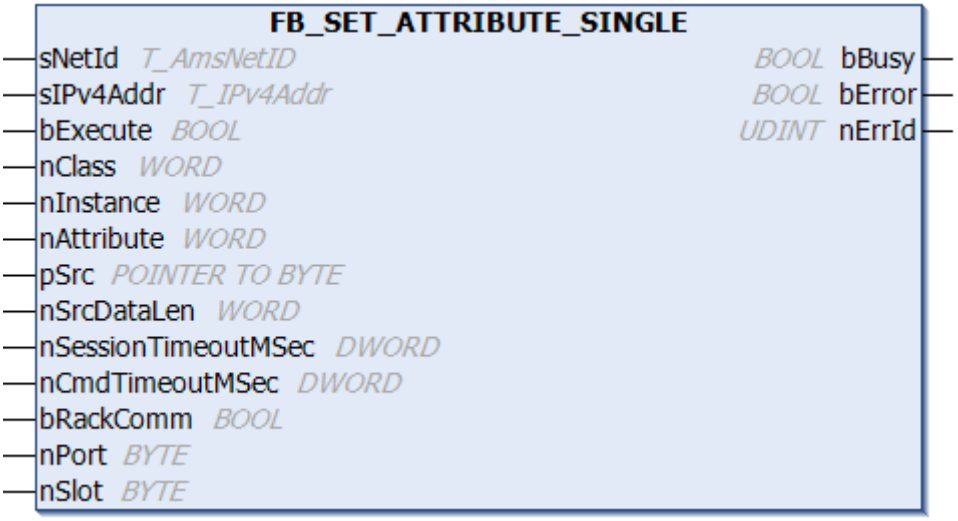
bBusy: When the function block is activated this output is set. It remains set until a feedback is received. While Busy = TRUE, no new command will be accepted at the inputs.

bError: If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.

nErrId: If an bError output is set, this parameter supplies an error number.

nDataLen: Returns the number of valid data (number of bytes).

6.1.2 FB_SET_ATTRIBUTE_SINGLE



The function block FB_SET_ATTRIBUTE_SINGLE enables writing of parameters in an EtherNet/IP device.

Service code: `0x10`

VAR_INPUT

```
VAR_INPUT
    sNetId              : T_AmsNetID;
    sIPv4Addr           : T_IPv4Addr;
    bExecute            : BOOL;
    nClass              : WORD;
    nInstance           : WORD;
    nAttribute          : WORD;
    pSrc                : POINTER TO BYTE;
    nSrcDataLen         : WORD;
    nSessionTimeoutMSec : DWORD;
    nCmdTimeoutMSec     : DWORD;
```

```
    bRackComm           : BOOL;
    nPort               : BYTE;
    nSlot               : BYTE;
END_VAR
```

sNetId: AMSNetId of the TwinCAT EtherNet/IP scanner through which the command is to run
sIPv4Addr: IP address of the target device
bExecute: A positive edge starts the command
nClasss: Class number of the CIP service
nInstance: Instance number of the CIP service
nAttribut: Attribute number of CIP service
pSrc: Pointer to the variable containing the value for sending the service (the pointer is determined with ADR)
nSrcDataLen: Size of the variable to which the pointer pSrc points (determined with SizeOf)
nSessionTimeoutMSec: Timeout for the session; the default is 30 seconds
nCmdTimeoutMSec: Timeout for the command; the default is 7.5 seconds
bRackComm: TRUE if the CPU is modular, i.e. a CPU with a rack design, for example a CompactLogix
nPort: Port number of the CPU (the TF6281 currently only supports port 1)
nSlot: Slot number if the CPU is not plugged into slot 0

VAR_OUTPUT

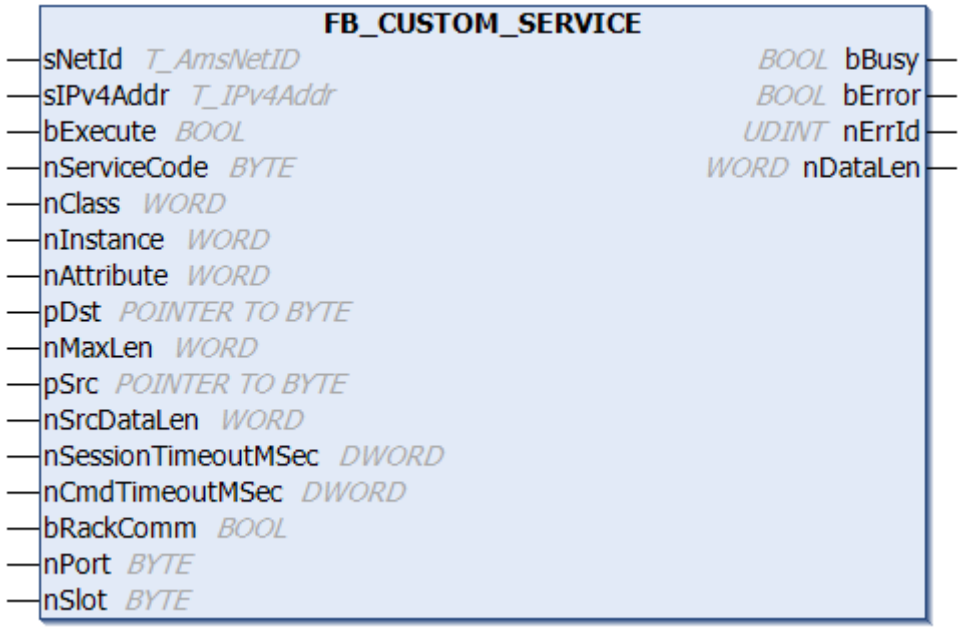
```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until a feedback is received. While Busy = TRUE, no new command will be accepted at the inputs.

bError: If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.

nErrId: If an bError output is set, this parameter supplies an error number.

6.1.3 FB_CUSTOM_SERVICE



The function block FB_CUSTOM_SERVICE enables virtually any CIP services to be created by the user.

VAR_INPUT

```
VAR_INPUT
    sNetId              : T_AmsNetID;
    sIPv4Addr           : T_IPv4Addr;
    bExecute            : BOOL;
    nServiceCode        : BYTE
    nClass              : WORD;
    nInstance           : WORD;
    nAttribute          : WORD;
    pDst                : POINTER TO BYTE;
    nMaxLen             : WORD;
    pSrc                : POINTER TO BYTE;
    nSrcDataLen         : WORD;
    nSessionTimeoutMSec : DWORD;
    nCmdTimeoutMSec     : DWORD;
    bRackComm           : BOOL;
    nPort               : BYTE;
    nSlot               : BYTE;
END_VAR
```

sNetId: AMSNetId of the TwinCAT EtherNet/IP scanner through which the command is to run

sIPv4Addr: IP address of the target device

bExecute: A positive edge starts the command

nServiceCode: Service code of the CIP service

nClasss: Class number of the CIP service

nInstance: Instance number of the CIP service

nAttribut: Attribute number of CIP service

pDst: Pointer to the variable to which the value is be copied (the pointer is determined with ADR)

nMaxLen: Size of the variable to which the pointer pDst points (determined with SizeOf)

pSrc: Pointer to the variable containing the value for sending the service (the pointer is determined with ADR)

nSrcDataLen: Size of the variable to which the pointer pSrc points (determined with SizeOf), or the number of bytes to be sent. Usually this is the size of the variable.

nSessionTimeoutMSec: Timeout for the session; the default is 30 seconds

nCmdTimeoutMSec: Timeout for the command; the default is 7.5 seconds

bRackComm: TRUE if the CPU is modular, i.e. a CPU with a rack design, for example a CompactLogix

nPort: Port number of the CPU (the TF6281 currently only supports port 1)

nSlot: Slot number if the CPU is not plugged into slot 0

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrId    : UDINT;
    nDataLen  : WORD;
END_VAR
```

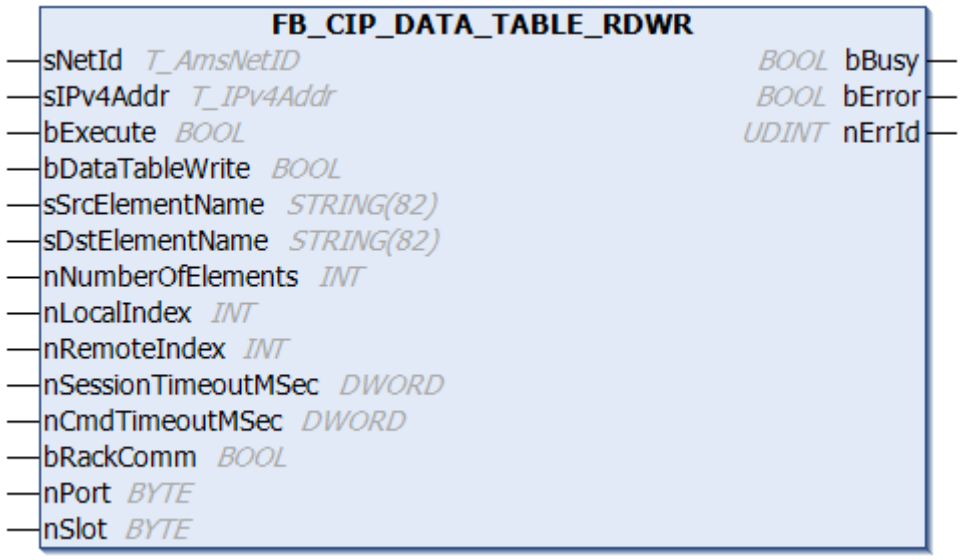
bBusy: When the function block is activated this output is set. It remains set until a feedback is received. While Busy = TRUE, no new command will be accepted at the inputs.

bError: If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.

nErrId: If an bError output is set, this parameter supplies an error number.

nDataLen: Returns the number of valid data (number of bytes)

6.1.4 FB_CIP_DATA_TABLE_RDWR



Variables are read and written from TwinCAT via a function block that is part of the Tc2_EthernetIP.

The function block FB_CIP_DATA_TABLE_RDWR can be used for reading and writing.

VAR_INPUT

```
VAR_INPUT
    sNetId               : T_AmsNetID;
    sIPv4Addr            : T_IPv4Addr;
    bExecute             : BOOL;
    bDataTableWrite      : BOOL;
    sSrcElementName      : WORD;
    sDstElementName      : WORD;
    nNumberOfElements    : POINTER TO BYTE;
    nLocalIndex          : WORD;
    nRemoteIndex         : DWORD;
    nSessionTimeoutMSec  : DWORD
    nCmdTimeoutMSec      : DWORD;
    bRackComm            : BOOL;
    nPort                : BYTE;
    nSlot                : BYTE;
END_VAR
```

sNetId: AMSNetId of the TwinCAT EtherNet/IP scanner through which the command is to run

sIPv4Addr: IP address of the target CPU

bExecute. A positive edge starts the command

bDataTableWrite: FALSE triggers a DataTableRead, TRUE a DataTableWrite

sSrcElementName: String for the source name

sDstElementName: String for the target name

nNumberOfElements: Number of elements

nLocalIndex: For ARRAYs the start index has to be set to indicate from which ARRAY index the data should be taken (local system)

nRemoteIndex: For ARRAYs the start index has to be set to indicate from which ARRAY index the data should be taken (remote system)

nSessionTimeoutMSec: Timeout for the session; the default is 30 seconds
nCmdTimeoutMSec: Timeout for the command; the default is 7.5 seconds

bRackComm: TRUE if the CPU is modular, i.e. a CPU with a rack design, for example a CompactLogix

nPort: Port number of the CPU (usually 1)

nSlot: Slot number if the CPU is not plugged into slot 0

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy   : BOOL;
    bError  : BOOL;
    nErrId  : UDINT;
END_VAR
```

bBusy: When the function block is activated this output is set. It remains set until a feedback is received. While Busy = TRUE, no new command will be accepted at the inputs.

bError: If an error should occur during the transfer of the command, then this output is set once the bBusy output was reset.

nErrId: If an bError output is set, this parameter supplies an error number.

Example

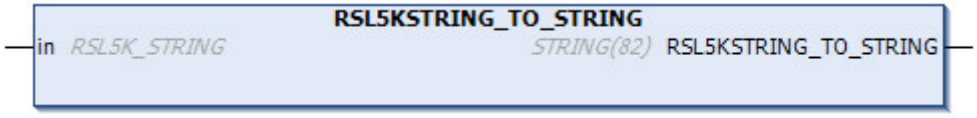
> **Removing test code**
>
> If you have already tested the communication from AB to Beckhoff, you should remove the function calls to DataTable Read/Write from the AB project

```
VAR
    FB_CIP_DATA_TABLE_RDWR: FB_CIP_DATA_TABLE_RDWR;
    SourceName: STRING := 'Test';
    DestName: STRING := 'ETHIP.Test';
END_VAR
```

```
FB_CIP_DATA_TABLE_RDWR(
  sNetId:='5.18.71.214.4.1' ,
  sIPv4Addr:='192.168.1.220' ,
  bExecute:=TRUE ,
  bDataTableWrite:= ,
  sSrcElementName:=(SourceName) ,
  sDstElementName:=(DestName) ,
  nNumberOfElements:=1 ,
  nLocalIndex:= ,
  nRemoteIndex:= ,
  nSessionTimeoutMSec:= ,
  nCmdTimeoutMSec:= ,
  bRackComm:=TRUE ,
  nPort:= ,
  nSlot:= ,
  bBusy=> ,
  bError=> ,
  nErrId=> );
IF NOT FB_CIP_DATA_TABLE_RDWR.bBusy THEN
  FB_CIP_DATA_TABLE_RDWR(bExecute:=FALSE);
  Error:=F_GET_ETHERNETIP_ERROR_HELPSTRING(FB_CIP_DATA_TABLE_RDWR.nErrId;
END_IF
```

6.2 Functions

6.2.1 RSL5KSTRING_TO_STRING



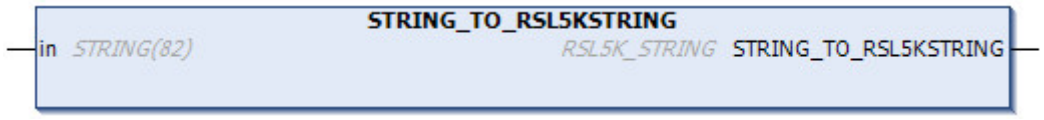
The function converts an RSL5KString value [▶ 44] to a string value.

FUNCTION RSL5KSTRING_TO_STRING : STRING(82)

VAR_INPUT

```
VAR_INPUT
    in   : RSL5K_STRING;
END_VAR
```

6.2.2 STRING_TO_RSL5KSTRING



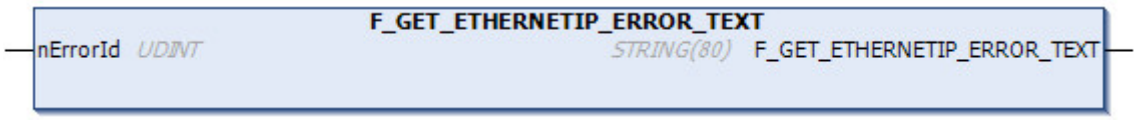
The function converts an RSL5KString value [▶ 44] to a string value

FUNCTION STRING_TO_RSL5KSTRING: RSL5K_STRING

VAR_INPUT

```
VAR_INPUT
    in   : STRING(82);
END_VAR
```

6.2.3 F_GET_ETHERNETIP_ERROR_TEXT



This function returns a descriptive text based on an error number.

See list of TF6281 error codes [▶ 46]

FUNCTION F_GET_ETHERNETIP_ERROR_TEXT: STRING(80)

VAR_INPUT

```
VAR_INPUT
    nErrorId  : UDINT;
END_VAR
```

6.3 Data types

6.3.1 RSL5K_STRING

```
TYPE RSL5K_STRING
  STRUCT
    LENGTH  : DINT;
    DATA    : ARRAY [0..81] OF SINT
  END_STRUCT
END_TYPE
```

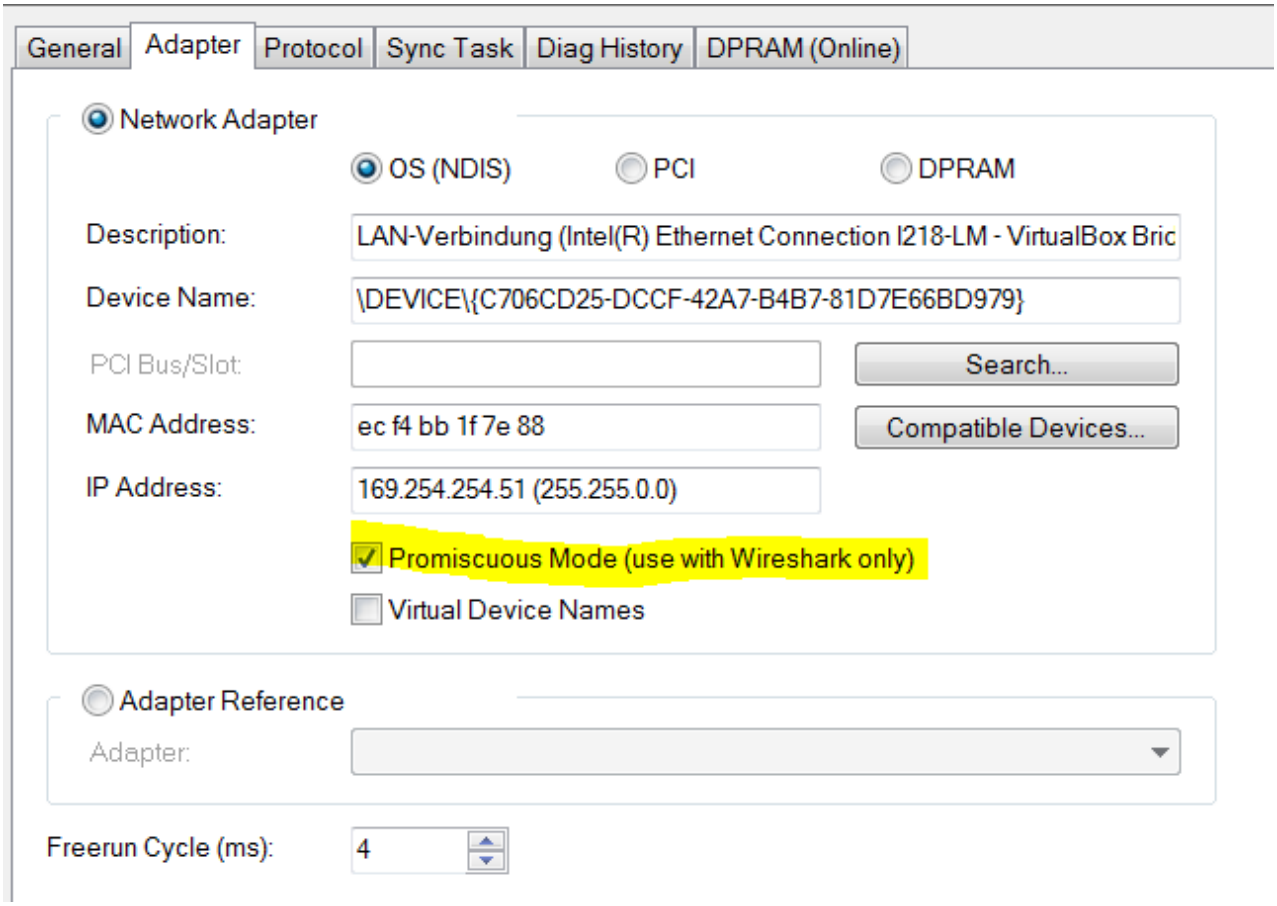
Length: Length of char characters contained in the data (max. 82)

Data: Chat characters

7 Appendix

7.1 Prepare Wireshark recording

The Wireshark recording can be created with a network hub, a network switch with port mirroring, e.g. the Beckhoff ET2000, or with the **Promiscuous Mode** of the TwinCAT system. In **Promiscuous mode**, it can happen that the telegrams are not recorded in the correct order, depending on the system performance and traffic. It is recommended to use an ET2000 for the recording.



7.2 Error Codes TF6281

Error	Code hex / (decimal)	Description	Remedy/meaning
CN_ORC_ALREADY_USED	0x100 / (256)	Connection already in use	The connection is already established; use another connection or close this one.
CN_ORC_BAD_TRANSPORT	0x103 / (259)	Transport type not supported	The transport type is not supported
CN_ORC_OWNER_CONFLICT	0x106 / (262)	More than one guy configuring	A connection already exists; a further connection cannot be established
CN_ORC_BAD_CONNECTION	0x107 / (263)	Trying to close inactive connection	Faulty connection
CN_ORC_BAD_CONN_TYPE	0x108 / (264)	Unsupported connection type	The connection type is not supported; check your setting.
CN_ORC_BAD_CONN_SIZE	0x109 / (265)	Connection size mismatch	The connection size does not fit; check your setting.
CN_ORC_CONN_UNCONFIGURED	0x110 / (272)	Connection unconfigured	Connection was not configured
CN_ORC_BAD_RPI	0x111 / (273)	Unsupportable RPI	The task time usually doesn't match; make sure that the EL6652 operates internally with 1 ms and that you can adjust this with the Cycle Time Multiplier. Otherwise, adjust the task time.
CN_ORC_NO_CM_RESOURCES	0x113 / (275)	Conn Mgr out of connections	No further resources are available
CN_ORC_BAD_VENDOR_PRODUCT	0x114 / (276)	Mismatch in electronic key	Incorrect manufacturer number
CN_ORC_BAD_DEVICE_TYPE	0x115 / (277)	Mismatch in electronic key	Incorrect device type
CN_ORC_BAD_REVISION	0x116 / (278)	Mismatch in electronic key	Incorrect revision number
CN_ORC_BAD_CONN_POINT	0x117 / (279)	Non-existent instance number	Incorrect connection number
CN_ORC_BAD_CONFIGURATION	0x118 / (280)	Bad config instance number	Faulty configuration
CN_ORC_CONN_REQ_FAILS	0x119 / (281)	No controlling connection open	The connection could not be established
CN_ORC_NO_APP_RESOURCES	0x11A / (282)	App out of connections	No further free connections available.

If you cannot fix this error yourself, Support will require the following information:

- TwinCAT version and build number and a
- Wireshark recording

7.3 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: https://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963 157
Fax:	+49 5246 963 9157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963 460
Fax:	+49 5246 963 479
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963 0
Fax:	+49 5246 963 198
e-mail:	info@beckhoff.com
web:	https://www.beckhoff.com

More Information:
www.beckhoff.com/tf6281

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com