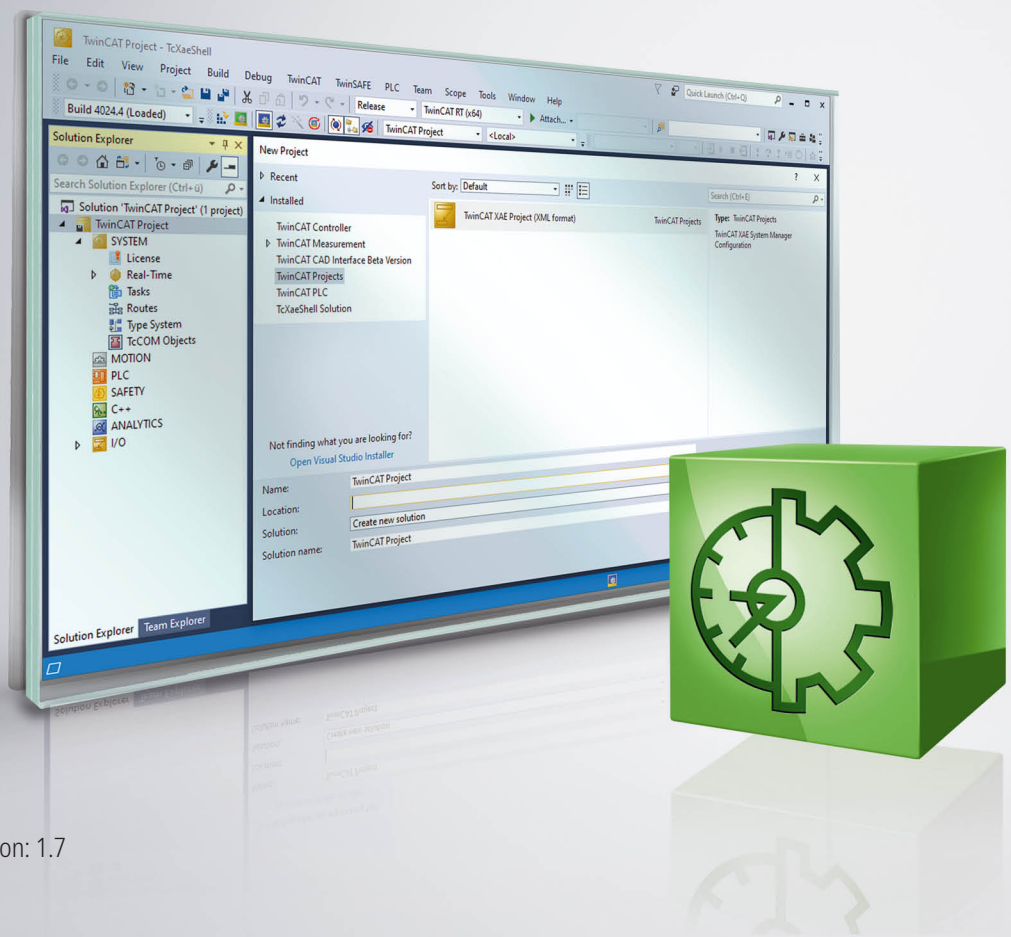
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | EAP



2021-11-17 | Version: 1.7

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

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

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.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Product description

The EtherCAT Automation Protocol (EAP) device enables the cyclic, highly deterministic exchange of any desired variables between PCs that are connected by Ethernet. Communication between EAP devices takes place according to the Publisher/Subscriber principle and is specified by the EtherCAT Technology Group (ETG) (ETG 1005 – see webpage www.ethercat.org).

The real-time Ethernet driver for TwinCAT must be installed for the TwinCAT EAP device in order for highly deterministic communication to take place.

Comparison with TwinCAT 2 network variables

The TwinCAT EAP device is based on the network variables (NWV) familiar from TwinCAT 2 and contains some extensions. Among other things the EAP telegram also slightly expands the NWV telegram. However, this expansion concerns only the contents of the telegram. The structure of the EAP telegram remains identical to the NWV telegram. For this reason network variables are compatible with the EtherCAT Automation Protocol and vice versa. Further details concerning EAP communication between TwinCAT 2 and 3 can be found in chapter EAP between TwinCAT 2 and 3 [▶ 50].

Requirements

The full functional scope of the TwinCAT EAP device as described in this documentation is available from TwinCAT version 3.1 (build 4018.13) or higher.

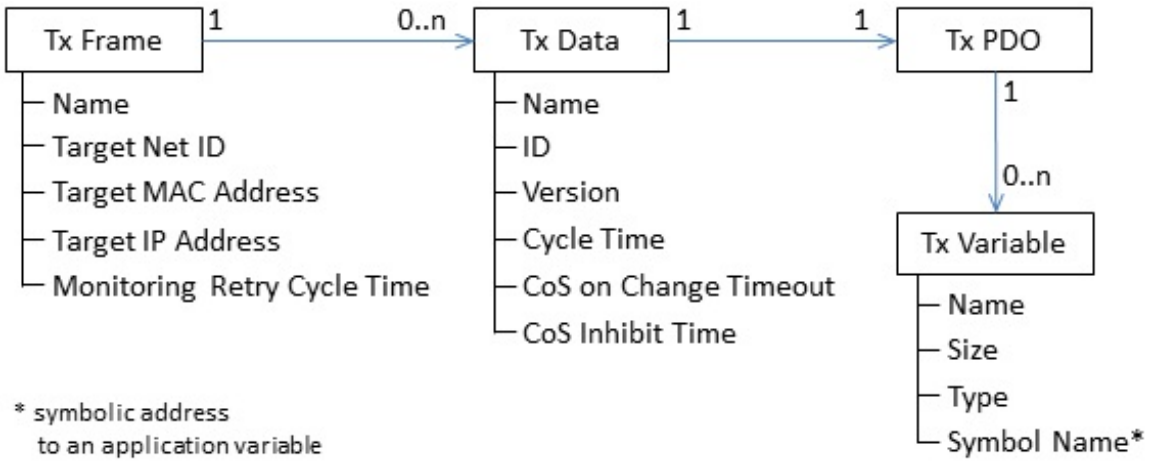
2.1 Basic principles

The TwinCAT EAP device enables data transfer from any variables from a TwinCAT controller A to a TwinCAT controller B via a network. These variables are typically used for controlling the processing operation within a machine. They are therefore also referred to as process variables (*PV*). For a TwinCAT EAP device, sending and receiving can take place via a standard network adapter, which is supported by the TwinCAT real-time Ethernet driver.

The communication between EAP devices takes place based on the Publisher/Subscriber principle. The senders, referred to as Publishers, send messages to all or several network devices; as a rule, a Publisher does not know the receivers or whether a receiver exists at all. On the other side there are receivers, referred to as Subscribers, which are interested in certain messages and receive these, without knowing from which Publisher they originate or whether such a Publisher exists at all.

Structure of an EAP Publisher

An EAP Publisher consists of a number of nested elements, as illustrated in the following illustration. The basic element at the lowest level is a *Tx variable*. It defines an output variable, which is linked to a process variable and has several further properties, such as a data type. The data type can be freely selected; it may be a complex data type, with a size of several hundred bytes. The only condition is that the maximum size of an EAP frame is not exceeded (the size of an EAP frame corresponds to that of a standard Ethernet frame). During operation, the process variable provides the values to be sent by the Publisher.



At the next higher level, *TxVariables* are referenced in the *TxPDO* elements (TxPDO = TxProcessDataObjects). A *TxPDO* can reference several *TxVariables,* thereby consolidating them in an object. The *TxPDO* then defines an ordered set of *TxVariables*. The condition that the maximum size of an EAP frame must not be exceeded also applies to a *TxPDO*.

The *TxData* element (*TxProcessData* = *TxPD*) is located at the next higher level. It represents a *Publisher variable* and is understood as communication unit of the Publisher in EAP. The *TxData* element references a certain *TxPDO* and defines a number of properties, such as the *ID* of the *Publisher variable*, their *version* and the clock cycle, based on which the *Publisher variable* is sent in the first place. Based on these properties, the *Publisher variable* defines an object on the sender side, for which a suitable Subscriber variable must be defined on the receiver side, so that successful data exchange can take place.

The data transfer takes place network-based via the Ethernet protocol or via UDP/IP. Similarly, a *TxFrame* is then assigned a list of TxData, which are to be sent to the same destination address. A *TxFrame* is limited to a maximum data length per data packet. For sending a *Publisher variable*, at least the following properties must be defined:

Destination address:
The destination address is usually a multicast address, so that the *Publisher variable* is automatically sent to a group of receivers. It is also possible to enter the address of an individual receiver.

ID:
For each *Publisher variable* a number is defined, which must be unique across the network. Based on this number, the *Publisher variable* can be identified by a *Subscriber*.

Clock cycle:
The clock cycle defines the interval at which the *Publisher variable* is sent. EAP cycle times generally range between a few milliseconds up to several 100 milliseconds.

Link to a process variable:
Last but not least, a link between the *Publisher variable* and a *process variable* is required, to ensure that process data are actually sent with the aid of the *Publisher variable*. Otherwise the value of the *Publisher variable* would always remain zero.

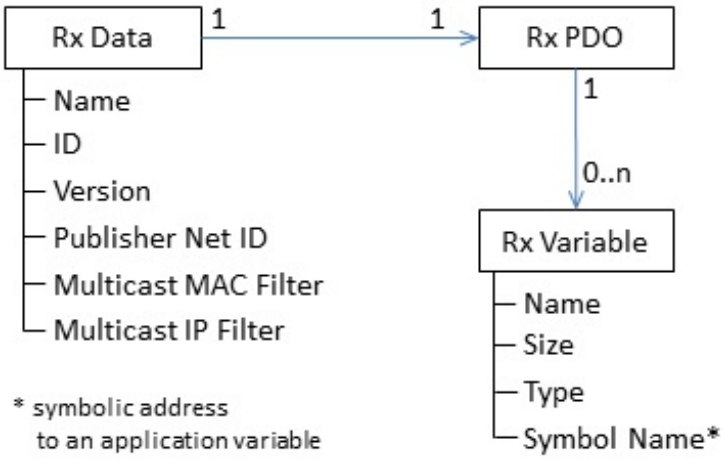
Structure of an EAP Subscriber

The structure of an EAP Subscriber is analogous to that of a Publisher and is illustrated in the following illustration. The basic element at the lowest level of a Subscriber is referred to as *RxVariable*. The RxVariable defines an input variable, which is also linked to a process variable and contains several properties, such as the data type. During operation, the process variable obtains the values, which the Subscriber receives.

Accordingly, the elements at the level above are referred to as *RxPDOs* (RxProcessDataObjects). Each element defines an ordered set of *RxVariables*.

The *RxData* element (*RxProcessData* = *RxPD*) is located at the next higher level. It represents a *Subscriber variable* and is understood as communication unit of the Subscriber in EAP. The *RxData* element references a certain *RxPDO* and defines the required properties (ID and version), which must match the *Publisher variable* to be received. For a successful data exchange, the data types of the referenced *RxVariable* and its

order in the *RxPDO* must be identical to the *TxPDO* of the *Publisher variable*. The *Subscriber variable* thus defines an object on the receiving side, for which a matching *Publisher variable* must be defined on the sender side, in order for data exchange to take place.



Due to the design of the EtherCAT Automation Protocol, for a Subscriber it is irrelevant from which sender the received data originate. In particular, it is irrelevant which *Publisher variables* are sent within a frame. For this reason the Subscriber has no frame element or similar, which would consolidate certain *Subscriber variables* as a unit, so that they would only be received en bloc. Nevertheless, *RxData* offers an option to define an *AMS NetID* as a filter address, in cases where a Subscriber should only receive the *Publisher variables* of a certain sender. In this case, at least the following properties must be defined for a *Subscriber variable*:

ID:
The *ID* of a *Subscriber variable* defines which *Publisher variable* should receive it. The *ID* is a number, which should be unique for each *Publisher variable* across the whole network. It is used to identify the *Publisher variable* at the receiving end.

Link to a process variable:
Finally, using a *Subscriber variable* only makes sense if it is linked to a process variable. Only then will the received data actually be applied by the process variable and taken into account in the machine control.

In addition, the data length of the *Subscriber variable* must be identical to data length of the *Publisher variable*. Otherwise the received *Publisher variable* is discarded.

2.1.1 Communication methods

The TwinCAT EAP device supports two communication types: cyclic process data communication (EtherCAT type 4), and acyclic EtherCAT mailbox communication (EtherCAT type 5). For mailbox communication, the TwinCAT EAP device only supports the AoE protocol (AoE – ADS over EtherCAT). The specification of the AoE protocol is described in the EtherCAT Protocol Enhancements (ETG 1020). For process data communication, a distinction is made between two communication modes:

Pushed Data Exchange mode, in which an EAP sender sends its process information to the network either cyclically or when a change in status is detected, and an EAP receiver expects this process information and receives it accordingly. This mode corresponds to the Publisher/Subscriber principle of the network variables (NWV) of TwinCAT 2.
Polled Data Exchange mode, in which an EAP client sends a request telegram with its process information to an EAP server, which then sends its process information back to the EAP client in a response telegram.

In addition, the TwinCAT EAP device supports different connection types and different addressing modes during process data communication. The supported connection types are:

- **Unicast**: The EAP message is sent from one end point to another end point, in other words: the message is addressed to precisely one PC.
- **Multicast**: The EAP message is sent from one end point to several other end points, in other words: the message is addressed to a group of PCs.

- **Broadcast**: The EAP message is sent from one end point to all accessible end points, in other words: the message is addressed to all devices.

MAC addresses, AMS NetIDs or IP addresses can be used. Depending on the configuration of the connection type and the addressing mode, a particular network protocol is activated for the EAP process data communication. The exact assignment is shown in the following table.

Table 1: Network protocols

Addressing mode Connection type	MAC address	AMS NetID	IP Address
Unicast	Ethernet protocol	Ethernet protocol	UDP/IP
Multicast	Ethernet protocol	*Not possible*	UDP/IP
Broadcast	Ethernet protocol	*Not possible*	UDP/IP

Depending on the different addressing modes (MAC, AMS NetID and IP), the connection types' unicast, multicast and broadcast are supported as follows:

MAC addressing:
The EAP message is transferred based on the Ethernet protocol. The MAC address of the network adapter that is to receive the message is configured as destination address. With this addressing mode, the EAP message cannot be relayed from a router to another subnet, since it operates based on IP addresses. The message can therefore only be sent within a subnet via switches.

Broadcast and multicast

Special MAC addresses are reserved for a broadcast or multicast message:

Broadcast MAC: FF-FF-FF-FF-FF-FF

Multicast MAC: A multicast MAC address must meet the following conditions.

- The lowest-order bit (bit 1) of the first byte has the value 1 (group bit).
- The following bit 2 has the value 0, if the MAC address is globally unique; or the value 1, if the address is only locally unique.
- The first 24 bits (bits 3 to 24) correspond to the manufacturer ID, referred to as Organizationally Unique Identifier (OUI). The OUI for Beckhoff is "00-01-05".
- The remaining 24 bits (bits 25 to 48) can be specified individually for each interface. The sequence "04-00-00" is defined for the EtherCAT Automation Protocol.

⇨ The resulting standard multicast MAC address for TwinCAT EAP devices is 01:01:05:04:00:00.

AMS NetID addressing:
The EAP message is transferred based on a type 4 EtherCAT protocol (EAP). The required target MAC address is determined based on the Address Resolution Protocol (ARP) and the configured AMS NetID. As with MAC addressing, the EAP message can only be sent within the subnet.

Communication via AMS NetID

Using an *AMS NetID* as destination address has the advantage that it is a logical address. The *MAC* address of the target device is determined with the aid of a special ARP request, using the configured *AMS NetID*.

The configuration of an EAP connection does not have to be adapted, even if a control computer or a network adapter of a computer is replaced, resulting in a change of *MAC* address, for example. The only condition is that the new control computer is assigned the original *AMS NetID*.

If the connection type *Unicast* is configured, the *Subscriber Monitoring* mechanism is also configured by default (see Remote station monitoring via ARP [▶ 13]).

IP addressing:
For the EAP message, the Internet protocol (IP) is used in conjunction with the User Datagram Protocol (UDP) for relaying and addressing of the recipient. The required destination MAC address is determined based on the Address Resolution Protocol (ARP) and the configured IP address. With UDP/IP addressing, a router can relay the EAP message to other subnets (including the internet, for example).

Special IP addresses are reserved for a broadcast or multicast message:

Broadcast IP: 255.255.255.255 is specified as broadcast IP address. The broadcast MAC address FF-FF-FF-FF-FF-FF is derived directly from this IP address.

Multicast IP: A multicast IP address must be in the range 224.0.0.0 to 239.255.255.255 (IPv4). In the EAP device, TwinCAT generates a compliant *multicast MAC address* for each configured *multicast IP address*, which is used when TwinCAT starts up (i.e. when the Run mode is activated).

Pushed Data Exchange (n:m connection)

The Pushed Data Exchange mode is based on the same model as the NWV transfer (Publisher/Subscriber principle). It offers the option of an n:m connection in a network. Each EAP device can send one or several EAP telegrams, together with its output process data (*TxData*). Each EAP device can “listen” to ascertain whether the process data contained in a received EAP telegram match its input process data (RxData), so that they can be processed, if applicable. One and the same EAP device can therefore send and receive process data. In this way a bidirectional communication can be established.

With Pushed Data Exchange, the addressing mode (unicast, multicast or broadcast) for each configured EAP telegram can be freely selected as required.

Polled Data Exchange (1:1 connection)

The Polled Data Exchange mode is subject to the client/server architecture principle. With the aid of this architecture, it enables “soft” synchronization. An EAP device can act as client and server at the same time.

Connection type for Polled mode

For the Polled Data Exchange mode, only the connection type unicast is defined uniquely.

Unicast (1:1 connection)
A client sends an EAP telegram together with its output process data to a server, which then returns its input process data to the client in a separate EAP telegram.

Network protocols

Ethernet protocol
The Ethernet protocol is responsible for switching the data packets in the network. It handles the tasks of OSI layers 1 and 2 (physical layer and data link layer). The Ethernet protocol header should contain a sender address, a receiver address and an Ethernet type, which specifies which protocol is used for the next higher OSI layer. The sender and receiver addresses are entered in the form of a MAC address. MAC stands for *media access control* and in this case refers to the unique hardware address assigned to each Ethernet device during production. The Ethernet port of a Beckhoff PC could be assigned the MAC address 00:01:05:34:05:84, for example; "00:01:05" is the Beckhoff ID, and the second part is specified during production. The sender and receiver MAC addresses determine the route of each Ethernet telegram between two PCs in the network. An Ethernet telegram can be processed further via a switch, but usually not via a router.

User Datagram Protocol / Internet Protocol (UDP/IP)
The receiver is identified via an additional IP header in the Ethernet telegram, so that it can be processed further by a router. The telegram has the Ether type 0x0800, which specifies that it is an IP telegram. In the subsequent UDP header, the port number 0x88A4 is used for the source port as well as the destination port. Based on this port number, the TwinCAT system detects that the telegram is a real-time based user datagram.

TwinCAT identifies an EAP telegram either on the basis of Ether type 0x88A4 (if the Ethernet protocol is used) or on the basis of the destination port 0x88A4 (for UDP/IP). Accordingly, the TwinCAT Ethernet driver makes a received EAP telegram bypass the NDIS stack of the operating system, so that TwinCAT treats it preferentially as a Beckhoff real-time Ethernet telegram. When an EAP telegram is sent, the NDIS stack of the operating system is also bypassed.
Once an EAP telegram has been received by a TwinCAT PC and identified as such, during further processing of the telegram the *process data* (*PD*) transported in the telegram are assigned to the RxData configured in the EAP device. This assignment is based on the *PD* ID. The received *PD* is discarded, if no RxData with matching *PD* ID were configured at the receiver.

Finally, the values of the individual *process variables* (*PV*) of a *PD* are only applied if, in addition, the data length and the version number of the received *PD* match the expected data length and version number.

2.1.2 Remote station monitoring via ARP

The EAP is based on the connection-less protocols (Ethernet protocol and UDP/IP). These protocols do not return acknowledgements for messages. The TwinCAT EAP device uses the Address Resolution Protocol (ARP) for remote terminal monitoring, in order to enable the sender of an EAP telegram to detect that the receiver is no longer available. The *ARP Retry Interval* can be used in an EAP Publisher to configure the time frame for checking whether the receiver is still accessible. Remote terminal monitoring (*Subscriber Monitoring*) can only be enabled if a unicast connection is configured.

If *Subscriber Monitoring* is enabled, the Publisher sends an *ARP Request* telegram to the configured target device, based on the configured time interval. If the receiver still operates as expected, it responds with an *ARP reply* telegram. Otherwise there is no response. In the diagnostic variable FrameState (see Publisher), the third bit (0x0004) is set in the event of an error.

ARP handling

The ARP handling for assigning MAC addresses to network addresses (IP addresses) is treated by the operating system (Windows). The ARP handling for assigning MAC addresses to AMS NetIDs is handled by the TwinCAT system.

2.1.3 EAP send mechanism

Sending of an EAP telegram is triggered based on a trigger mechanism. The configuration of an EAP device is used to determine how this *trigger* mechanism works. For each *TxData* a *trigger* condition is defined. If this *trigger* condition is met, *TxData* are sent via an EAP telegram. In other words: In each EAP device, *trigger* conditions are used for each *TxData* to configure the operation of the *trigger* mechanism.

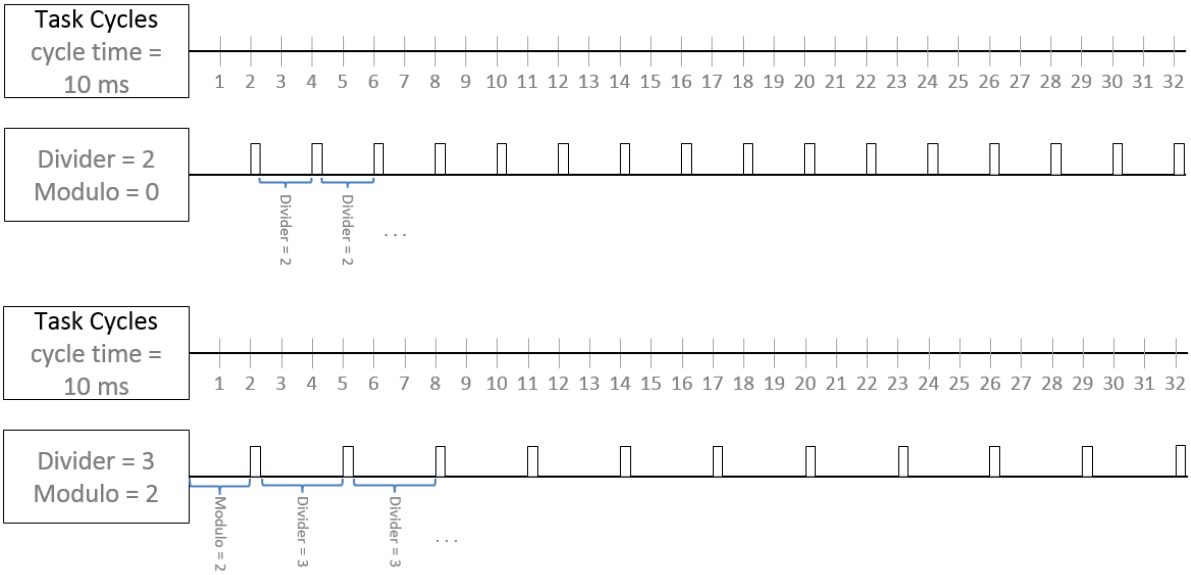
There are 5 different types of trigger conditions, which are described here.

Superposition of trigger conditions

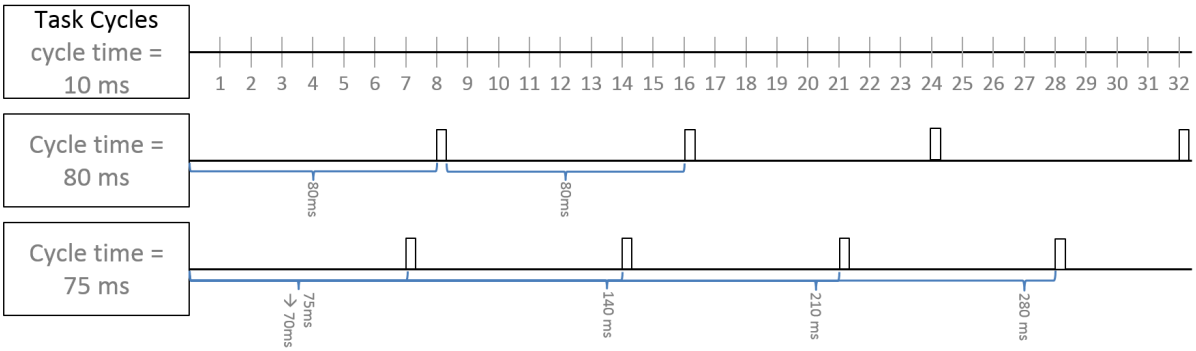
The explanations of the individual trigger conditions indicate which other trigger conditions need to be deactivated. In other words, which conditions are not allowed in combination. The example further below shows that several active trigger conditions mutually overlap. How they overlap is not clearly defined. It is therefore advisable to disable all trigger conditions that are not permitted.

1. **Poll Request Rx PD**
   The trigger condition *Poll Request Rx PD* controls the sending of a response telegram in Polled Data Exchange mode (see section Communication methods [▶ 10]). Once a *TxData* has a valid entry for the trigger condition *Poll Request Rx PD*, the respective *TxData* operates in this mode. A valid entry is present, if it matches the object index of an *RxData* configured in the EAP device. This *RxData* then defines the expected request for returning the *TxData* as response. When the EAP device receives an EAP telegram containing the expected *process data*, in the next cycle the *TxData* is returned in a new EAP telegram to the sender of the request telegram. Consequently, the EAP device serves as *Polled Data Exchange* server for this *TxData*, when the trigger condition *Poll Request RxData* is enabled.
   All other conditions (2 to 5) have to be disabled, if the *Poll Request Rx PD* condition is enabled.
2. **Divider/Modulo**
   A *Divider/Modulo* condition is used to specify the frequency with which a *TxFrame* or a *TxData* is sent (see illustration below). The frequency is always a multiple of the task cycle time driving the EAP device. The *divider* value defines the multiple. The *Modulo* value specifies the start cycle at which the *TxFrame* or the *TxData* is sent for the first time. If the *Divider* has the value 0, this condition is disabled.
   Conditions 3, 4 and 5 are not relevant if the *Divider/Modulo* condition is enabled; they should never-

theless be disabled. Condition 1 must be disabled.



3. **Cycle Time**
   The *TxData* is sent at particular intervals, as defined by the *cycle time* value (unit: µs) (see illustration below). The *cycle time* should be an integer multiple of the task cycle time. If a value is configured that is not an integer multiple of the task cycle time, the next smaller multiple is set automatically, down to 0, if necessary. If the value is 0 µs, this condition is disabled.
   Trigger conditions 1, 2, 4 and 5 should be disabled, if the *cycle time* trigger condition is enabled.

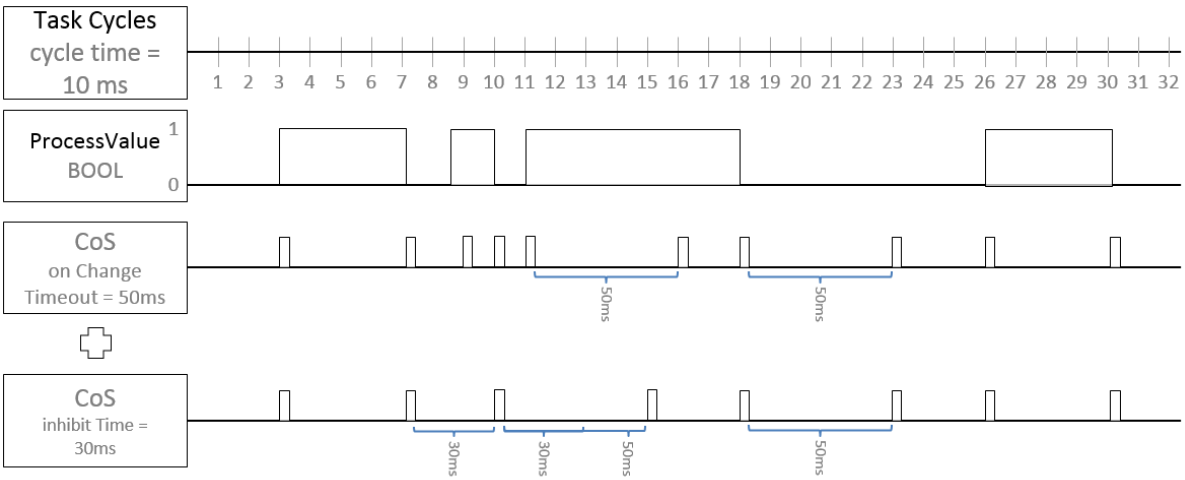


Relationship between cycle time and task cycle time

Assuming the task cycle time is 5 ms (5000 µs), the cycle time of *process data* A is 10000 µs, and the cycle time of *process data* B is 20000 µs. The task cycle time is now slowed down from 5 ms to 15 ms (15000 µs). Neither the cycle time of *process data* A nor that of *process data* B is a multiple of the task cycle time; the cycle time is therefore not divisible by the task cycle time without remainder.
As a result, *process data* A is only sent every 15 ms (15000 µs), *process data* B only every 30 ms (30000 µs).

4. **Change of State (CoS): On Change Timeout**
   The *TxData* is only sent when the value of one of its variables has changed compared with the previous value. A maximum time interval is defined as timeout time (unit: µs). If the value of a variable does not change within this interval, the *TxData* is sent regardless, after the time interval has elapsed (see illustration below). The value for the time interval must be an integer multiple of the task cycle time. If

the time interval is set to 0 µs, the trigger condition *CoS On Change Timeout* is disabled. Trigger conditions 1, 2 and 3 must be disabled, if the trigger condition *CoS On Change Timeout* is enabled.



5. **Change of State (CoS): Inhibit Time**
   The *Inhibit Time* specifies a minimum time interval, so that the *TxData* is not sent before this time interval has elapsed after it was last sent.
   The *Inhibit Time* therefore specifies a minimum time interval in µs, after which the *TxData* is sent - even if one value of the included *Tx variable* has changed (see illustration below). The value for this time interval can only be an integer multiple of the task cycle time, and it must be less than the value of *CoS On Change Timeout*. If the time interval is set to 0 µs, the trigger condition *Inhibit Time* is disabled.
   Trigger conditions 1, 2 and 3 should be disabled, if the inhibit condition *Inhibit Time* is enabled.

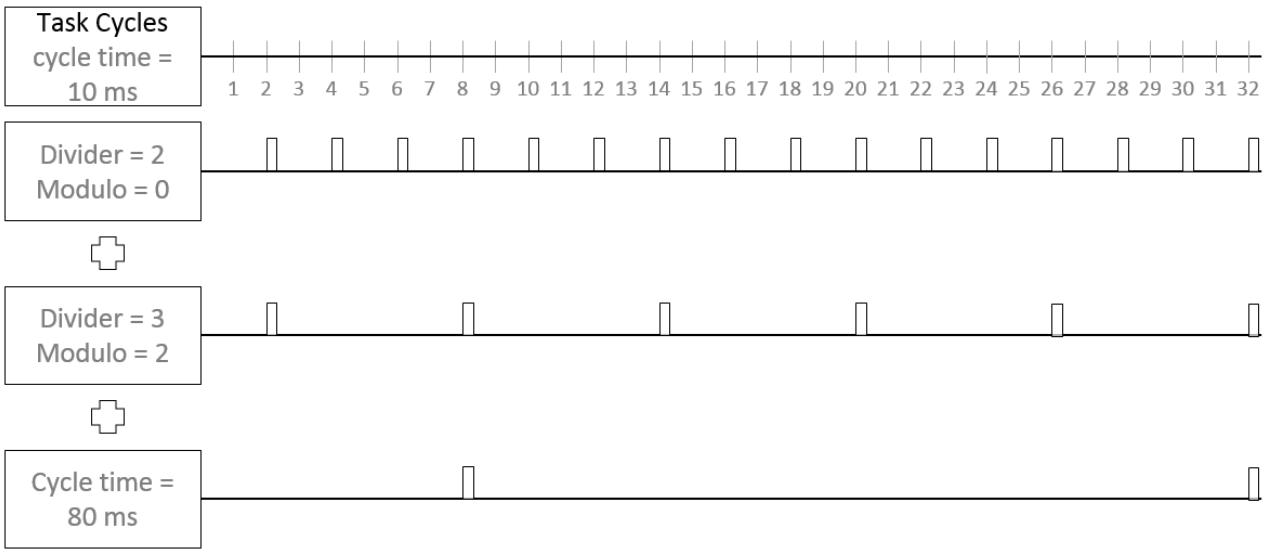
Configuration options for the trigger conditions

The trigger conditions 1, 3 and 5 (Poll Request RxData, Cycle Time and Inhibit Time) can be configured via the EAP object dictionary (see chapter CANopen object dictionary [▶ 52] in the documentation for the TwinCAT EAP device).

Special features of the trigger conditions

For all trigger conditions that define a time interval, this interval cannot be smaller than the task cycle time of the task driving the EAP device.

A combination of the conditions is not recommended because they are not clearly defined. The following gives a good example of the complexity:



The last line clearly shows that the transmission at 160 ms and 240 ms does not take place because it is prevented by the additional divider/modulo conditions.

2.1.4 EAP performance

If the TwinCAT EAP device is used, the temporal boundary conditions of the network architecture used must be taken into account:

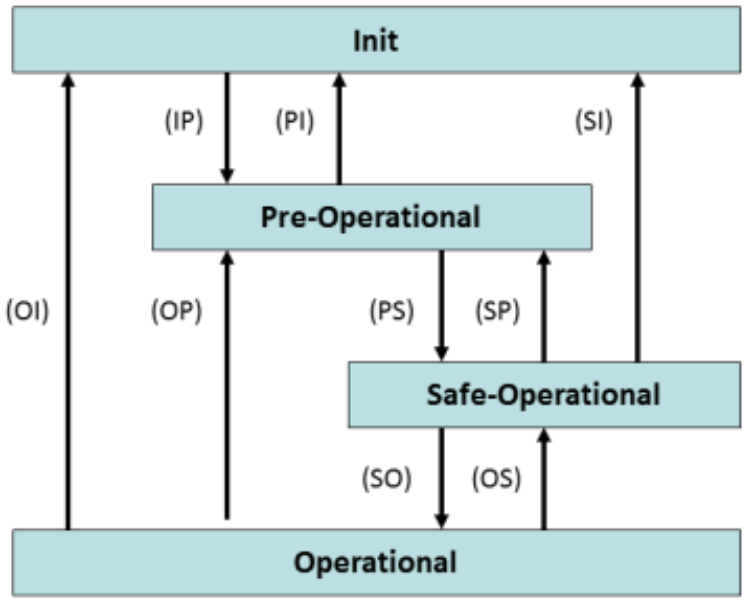
- In a network architecture, in which telegrams are exclusively sent via switches (e.g. per Ethernet protocol), communication cycles of around 10 ms or below can be achieved.
- In a network architecture, in which telegrams can also be sent via a router (i.e. via UDP/IP), a communication cycle time of around 100 ms can be achieved.

In a network, in which other communication takes place in parallel, the performance of the EAP communication can be impaired.

2.1.5 The EAP state machine

The EAP state machine (EAP SM) controls the state of the EAP device. Depending on the state, different functions are accessible or executable in the EAP device. A distinction is made between the following states:

- Init
- Pre-Operational
- Safe-Operational and
- Operational



The regular state of each EAP device after it started is the **OP** state. Until the **OP** state is reached, the EAP device is switched once to each state in turn. During each state transition the EAP device performs certain actions. If an error occurs during one of the transitions, the device cannot be switched to the corresponding subsequent state and therefore remains in the state that was reached last. A readable error code can be used to diagnose the reason for the error.

Init

As a rule, the **Init** state of an EAP device is a temporary state. That is, the EAP device cannot be set to the **Init** state explicitly. Nevertheless, there are cases in which the SM resets the EAP device to the **Init** state. In this state neither mailbox communication nor process data communication with the EAP device is possible.

Pre-Operational (Pre-Op)

During the transition from **Init** to **Pre-Op,** the EAP device checks whether the mailbox was initialized correctly. In **Pre-Op** state, mailbox communication is possible, but not process data communication.

Safe-Operational (Safe-Op)

During the transition from **Pre-Op** to **Safe-Op,** the EAP device checks the internal object references and updates:

- the cycle time-based configuration parameters,
- the reference pointers to the input and output variables of the process image, and
- the mapping of each Publisher/Subscriber variables to its process variables from the PLC.

In **Safe-Op** state, mailbox communication and sending of Publisher variables takes place. No EAP telegrams are received yet.

Operational (Op)

During the transition from **Safe-Op** to **Op**, the EAP device checks once again whether an error has occurred during startup.

In **Op** state the EAP device receives incoming EAP telegrams and copies the received process data to the input variables, if required. Mailbox communication takes place, Publisher variables are sent, and Subscriber variables are received.

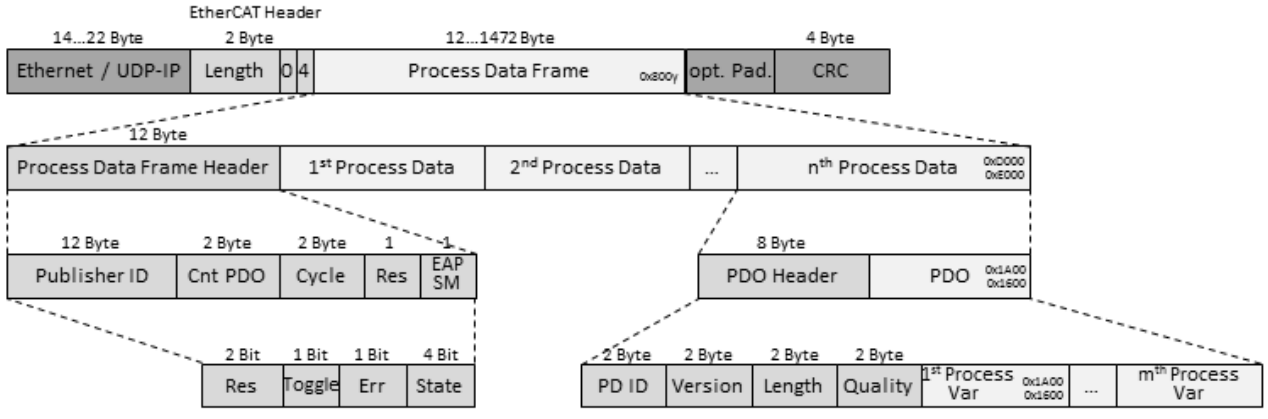
If an error occurs in one of the state transitions, the EAP device remains either in the last reached state, or it is reset to **Safe-Op** state. At the same time, the error bit and a corresponding error code are set (cf. section The CANopen object dictionary [▶ 52] in the documentation for the TwinCAT EAP device). Typical errors occur due to inconsistencies in the CANopen object dictionary, for example, so that the configuration is invalid.

2.2 Technical concept

2.2.1 EAP telegram structure

An EAP telegram can be transmitted by Ethernet (Ethertype = 0x88A4, corresponds to the EtherCAT protocol) or by UDP/IP (UDP port 0x88A4). If EAP is based on the EtherCAT protocol, the EAP-specific telegram parts are embedded in the user data of the EtherCAT protocol.

The EAP telegram consists of a *Process Data Frame Header* and one or more *ProcessData (PD)*. A *PD* is the main unit in the exchange of data via EAP. A *PD* is composed of the so-called *PDO Header* and at least one *Process Variable (PV).* On the whole, the *Process Variables* of a *PD* form the *ProcessDataObject (PDO)*. See the following illustration.



The Process Data Frame Header

The *Process Data Frame Header* consists of five fields (see lines 2 and 3 in the illustration above). The latter field (EAP SM) extends the NWV telegram of twincat 2 with regard to contents.This field contains among other things information about the current state of the EAP device.

Publisher ID
The sender of a telegram is identified on the basis of the Publisher ID. It contains the AMS NetID of the sender. If a receiver is configured so that it only processes telegrams from a certain sender, a check is carried out on the basis of the Publisher ID field as to whether the EAP telegram originated from this sender.

Cnt PDO
This field contains the number of *ProcessData* contained in the telegram, so that the receiver can fully process the telegram.

Cycle
The value of this field is incremented with each task cycle of the sender. On the receiving side the contents of this field can be used as a sequential number in order, for example, to check whether a telegram has been lost. This field should be monitored on the server side in Polled Data Exchange mode.

Res
reserved for future extensions.

EAP SM
This field is composed of 4 subentries.

- **Res:** reserved for future extensions.
- **Toggle:** This bit is toggled for each new EAP telegram.
- **Err:** This field indicates whether an error has occurred with the EAP State Machine.
- **State:** This field contains the value for the current state of the EAP State Machine.

  1 : Init

  2 : Pre-Operational

  4 : Safe-Operational

  8 : Operational

Values other than these are not allowed

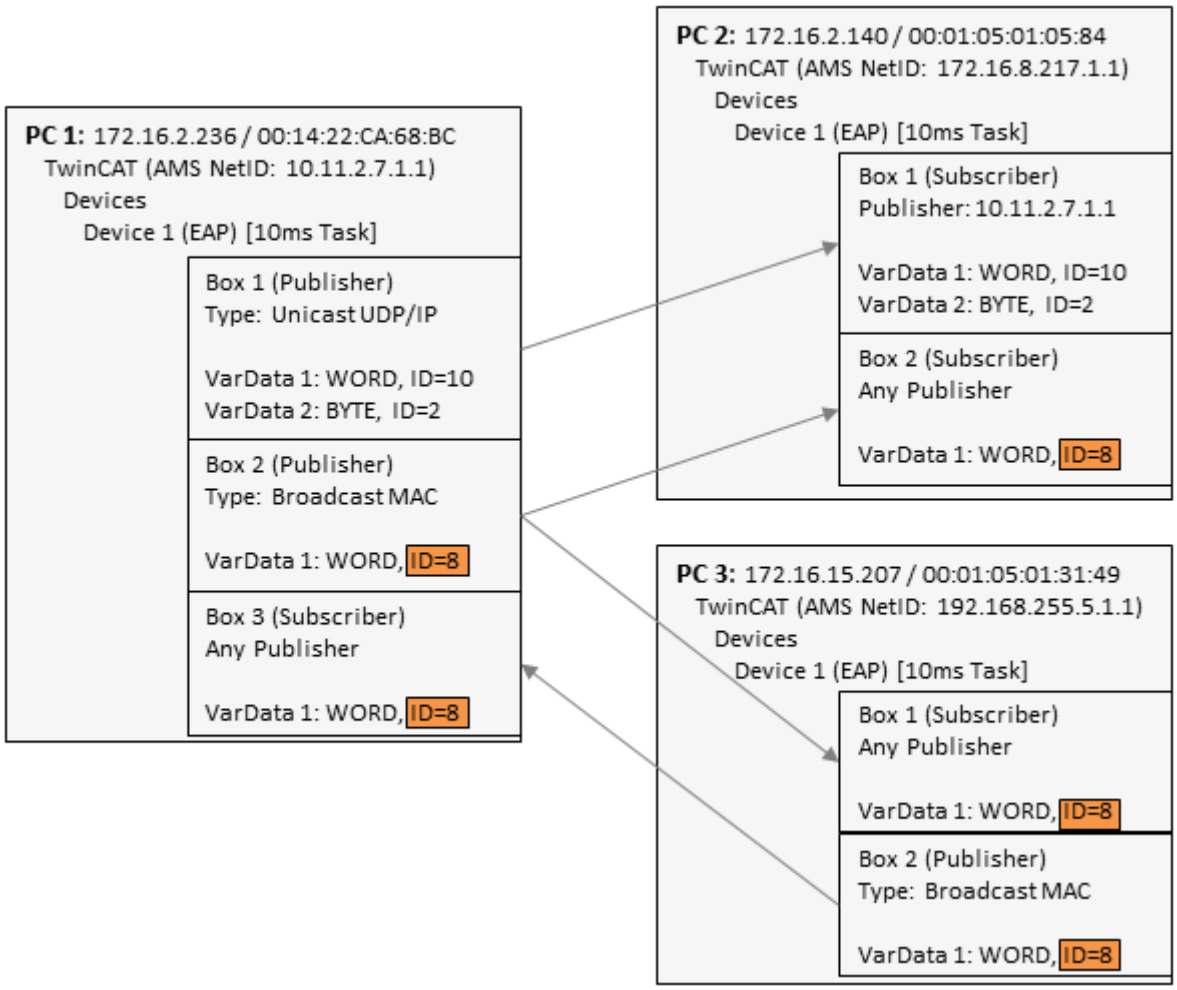
The PDO Header

The *PDO Header* consists of 4 fields, each of which has a data length of 2 bytes (see lines 3 and 4 in the illustration above).

PD ID
The *PD ID* (16-bit) serves as the global identification of the *EAP variables* and should be unique in the network.

Selection of the ProcessData Identifier (PD ID)

In order to achieve unambiguous allocation we recommend using different IDs for each data communication between connected PCs.
Reason: In the following illustration, *PC 2* in *Box 2 (Subscriber)* receives not only the intended variables as *Box 2 (Publisher)* with the ID 8 from *PC 1*, but also, since they are sent as a broadcast(!), the variables from *Box 2 (Publisher)* from *PC 3.* It is then no longer possible to make a distinction in *PC 2*!



Version
A version number can be entered for the *PD* in this field. The influence that the *version* has on the EAP communication is explained in the *Network protocols* section of the chapter Communication methods [▶ 10]. The version number should be consistently increased if at least one of the following changes is made to *ProcessData*.

- The data length of the *ProcessData* is changed.
- The data type of a variable of the *ProcessData* is changed.
- The order of the variables in a PDO is changed.

Length
This field contains the length of the *ProcessData* in bytes. The length of the *Process Data Header* itself is not included.

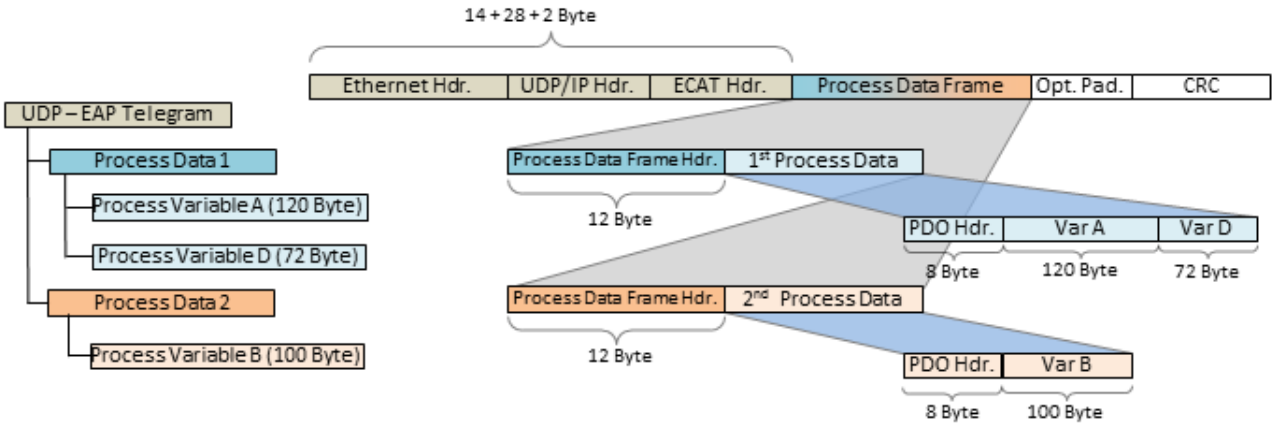
Quality
The value of this field shows the age of the *ProcessData* in the unit [100 µs] if the value lies between 0x0 and 0xEFFF. A value greater than or equal to 0xF000 means that the *ProcessData* is invalid.

The Process Variable (PV)

The *Process Variable* contains the actual data to be transmitted (see *Process Var* in line 4 of the first illustration above). The data length of a *PV* may only be so large that the entire EAP telegram is not larger than 1514 bytes.

Calculation of the size of an EAP telegram

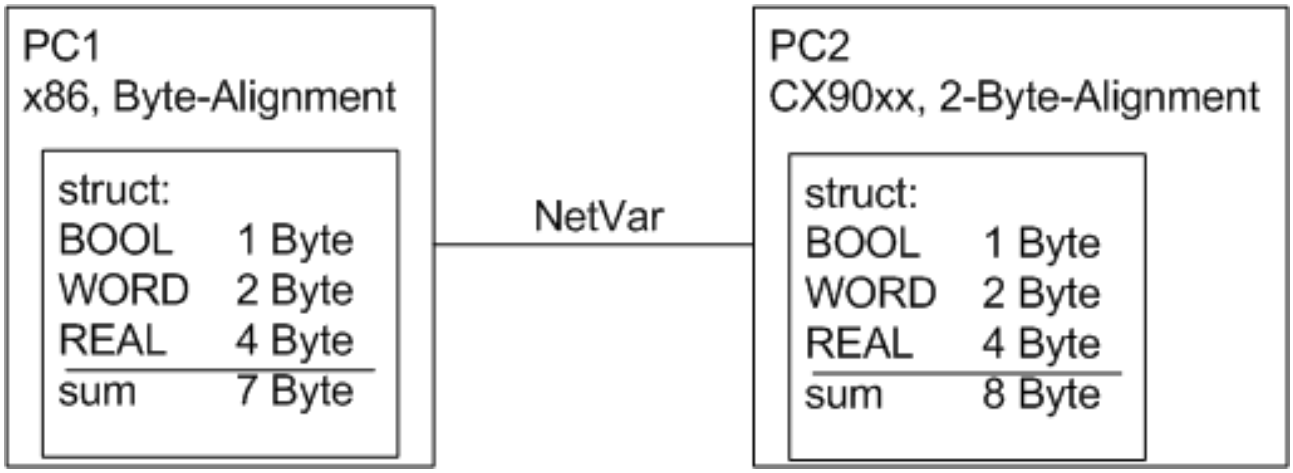
The sum of the lengths of all headers and all publisher variables may not exceed the limit of 1514 bytes. The EAP telegram shown (see following illustration) has a total size of
14 + 28 + 2 + 12 + 8 +120 + 72 + 12 + 8 + 100 = 376 bytes.



(This calculation includes 28 bytes for the UDP/IP Hdr. The 28 bytes are available for data in pure Ethernet communication.)

Data representation on different platforms

Note that both simple and complex data (WORD, ARRAYs, REAL, STRING, user-defined structures) are represented internally in a different manner on different platforms. x86 platforms use byte-alignment, others (ARM) 2-byte or 4-byte alignment.
This means that if a complex structure is created in both an x86/PC PLC project and an ARM PLC project, they can each have a different effective size and a different internal structure.



In the example (see illustration above) the structure in PC2 is larger than in PC1. Not only that, the word and real variables don't match, since a variable can begin at any byte position in PC1 but only at any even-numbered byte position in PC2.

It is recommended to observe the following rules when assembling structures so that no size differences are caused by the alignment:

- firstly, all 4-byte variables (must lie at an address that is divisible by 4)
- then all 2-byte variables (must lie at an address that is divisible by 2)
- then all 1-byte variables

Further recommendation:

- If the data type STRING(x) is used in a PLC, the null termination of the string also belongs to the string itself, so that x+1 must be divisible by 4. Otherwise there is no 4-byte alignment.
- The above rules also apply to substructures.

Other information can be found in the Beckhoff information system under "TwinCAT 3/TExxxx | TC3 Engineering/PLC/PLC/Programming Reference/Data Types/User-defined Data Types/Structures" or "TwinCAT 3/TExxxx | TC3 Engineering/PLC/PLC/Programming Reference/Declaration/Alignment".

Use of Bus Terminal Controllers (BCxxxx, BXxxxx)

Floating point numbers (data type REAL) cannot be transmitted to Bus Terminal controllers (BCxxxx, BXxxxx), since the representation of a floating point number on a Bus Terminal controller differs from that on an x86 platform.
The data type SINT, for example, can be used for signed values.

3 Diagnosis of an EAP connection

The TwinCAT EAP device provides different variables with whose help the quality or failure of an EAP communication can be diagnosed. Such diagnostic variables can be found in both the publisher and the subscriber. They can be programmatically evaluated in order to react to possible malfunctions.

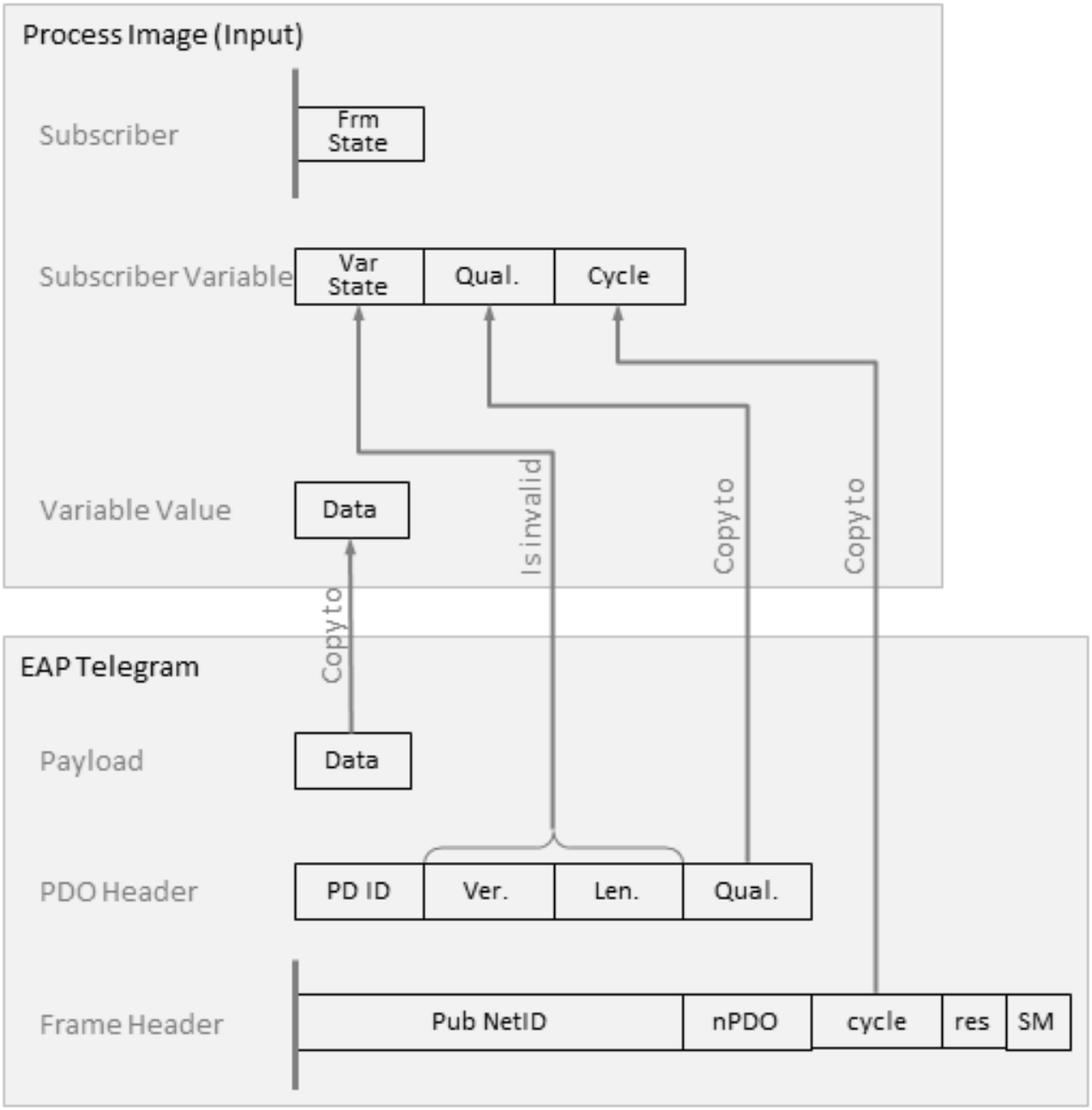
3.1 Subscriber

In the case of the subscriber there are the diagnostic variables *Quality*, *CycleIndex* and *VarState*.

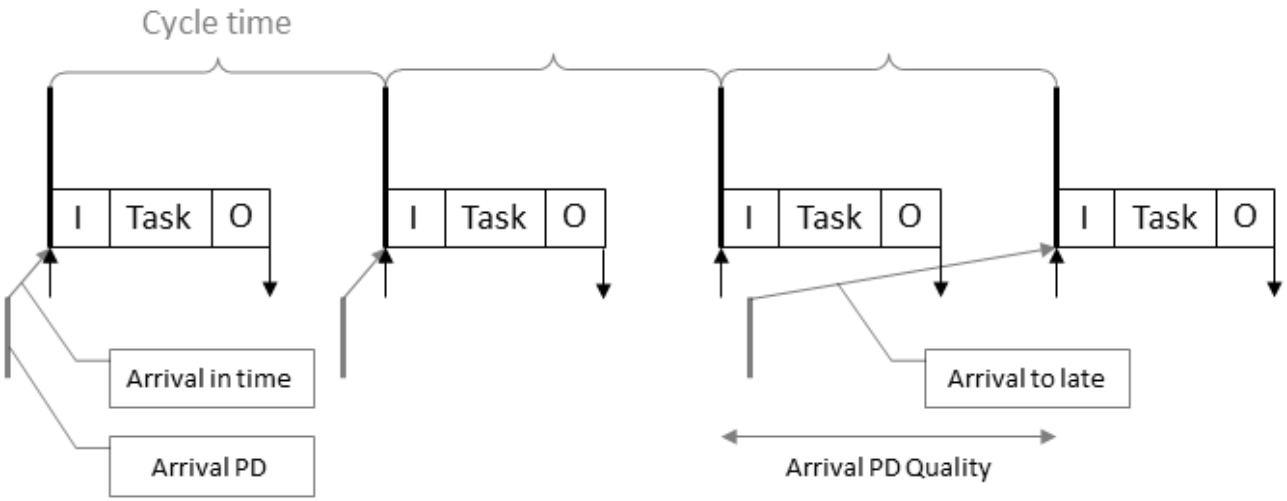
- *Quality*:
  The *Quality* indicates the age of the data received.
- *CycleIndex*:
  Transmission failures can be detected with the help of *CycleIndex*.
- *VarState*:
  The *VarState* variable indicates whether there are discrepancies between the configured *RxData* and the incoming *ProcessData* that are preventing the reception of the data.

Quality

The *Quality* variable contains the length of time in [100 µs] by which this *ProcessData* arrived too late at the receiver. It is incremented in each cycle by the cycle time immediately before the input process image of the TwinCAT EAP device is updated. The updating of the input process image takes place during the processing of the received EAP telegrams. During this processing, several values of each telegram are allocated to the corresponding variables of the input process image.



As shown in the illustration above, the received *Quality* value from the telegram, for example, is allocated to the corresponding *Quality* variable of the input process image.
If the *Quality* in an EAP telegram has the value 0, then the *Quality* of the input process image is reset to the value 0 as soon as the telegram has been received.



The above illustration shows how the delayed arrival of an EAP telegram influences the *Quality* value: at the start, the EAP telegram is received in good time prior to the start of the 1st cycle. Likewise in the 2nd cycle. After that, the telegram doesn't reach the receiver until after the 3rd cycle has already begun. The consequence of this is that the telegram cannot be processed until the 4th cycle. Accordingly, the value of the *Quality* variable is incremented by the *cycle time* during the 3rd cycle, but not reset to the value 0. The *Quality* variable is only reset to the value 0 in the 4th cycle by allocating the *Quality* value of the delayed telegram.

Diagnostic variable quality
Assuming the task cycle of the subscriber is ten times as fast as that of the publisher, then an EAP telegram is received only every tenth cycle. Consequently, no telegram arrives at the subscriber for nine cycles, which also means that the *Quality* variable of the input process image cannot be reset for nine cycles. The *Quality* variable will thus be incremented by the cycle time for nine cycles. It thus increases up to nine times the cycle time.

The result: A "slow" sender (e.g. 100 ms transmission clock in the publisher) leads with a "fast" receiver (e.g. 10 ms receiving clock in the subscriber) to a correspondingly increasing value of the diagnostic variable *Quality*.

It is therefore important to consider different cycle times for the transmission and reception of EAP telegrams. In this respect it is particularly important to pay attention to the *Trigger* conditions (see EAP send mechanism [▶ 13]) that are configured in the sender.

EL6601/EL6614
When using the EL66xx, the time of arrival of a *ProcessData* is precisely when the data are present in the input process image of the EAP device, not when they arrive at the EL66xx or in the input image of the EtherCAT device.

CycleIndex
The *CycleIndex* (size: 16 bits) is a counter that is transmitted by the publisher with the *ProcessData*. It is usually incremented on the sender side with each new cycle before the EAP telegram is sent, thus allowing an inference to transmission interruptions. It can be read on the receiver side (in the subscriber) as *CycleIndex* (cf. uppermost illustration in this chapter).

VarState
The *VarState* (size: 16 bits) supplies information about the current status of the *RxData*.
The following values are possible for *VarState*:

Short description	Bit	Description
Invalid Hash/Version	VS.0	The bit is set to 1 if a *ProcessData* couldn't be received because the version of the received ProcessData didn't correspond to the configured version of this RxProcessData. The bit is otherwise set to 0.
Invalid Variable Length Received	VS.1	The bit is set to 1 if a *ProcessData* couldn't be received because the data length of the received variable didn't correspond to the configured RxVariable. The bit is otherwise set to 0.

EL6601/EL6614

The VarState is not created when using the EL66xx terminals.

3.2 Publisher

The diagnostic variables *VarState* and *FrameState* are available in the publisher. Detailed properties for the transmission of an EAP telegram are clarified by means of these diagnostic variables.

VarState
VarState (size: 16 bits) provides information about the current state of the *TxData*.
The following values are possible for *VarState*:

Short description	Bit	Description
Not Sent (variable skipped)	VS.0	The TxProcessData is transmitted as long as the value is 0. The transmission of the TxProcessData is otherwise suspended.

EL6601/EL6614

This variable is not created when using the EL66xx terminals.

FrameState
FrameState (size: 16 bits) provides information about the current state of the *TxFrame*. The following values are possible for the *FrameState*:

Short description	Bit	Description
Not Sent (frame skipped)	FS.0	The Ethernet frame is transmitted as long as the value is 0. The transmission of the frame is otherwise suspended.
Error (frame oversized)	FS.1	If the value is 1, the maximum size of an Ethernet frame has been exceeded. The linked variable should be smaller.
Subscriber Missing (Unicast only)	FS.2	This bit is only set if the Subscriber Monitoring is activated (see chapter Publisher Box [▶ 44]). The value is set as soon as the Subscriber Monitoring mechanism detects that the EAP device to which the Ethernet frame is sent is no longer reachable. The bit is reset to 0 as soon as the reachability is restored.

EL6601/EL6614

This variable is not created when using the EL66xx terminals.

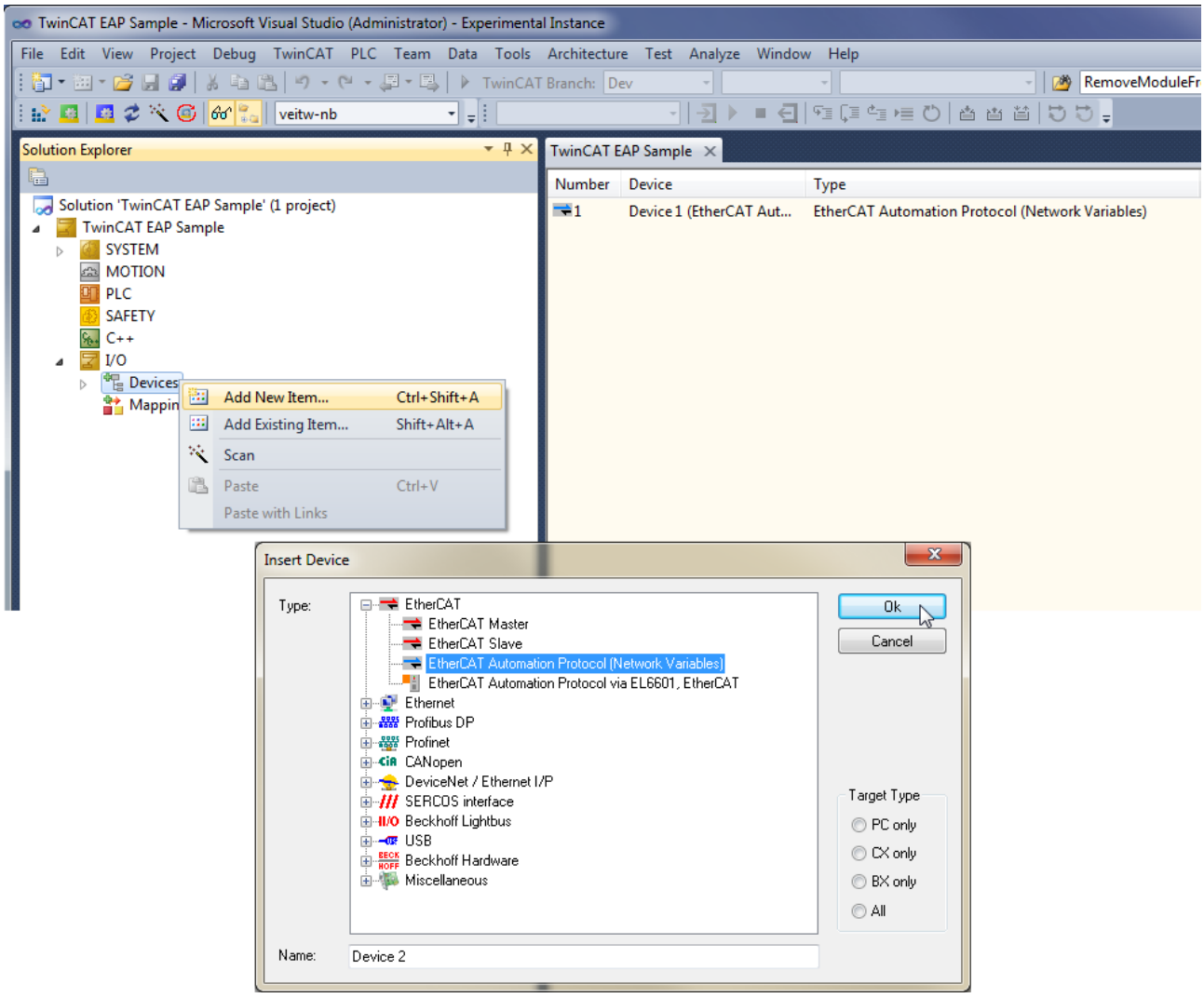
4 Creation of an EAP configuration

A communication connection between TwinCAT controllers via EAP is created with the aid of TwinCAT 3.

4.1 Adding an EAP device

An EtherCAT Automation Protocol device is added in TwinCAT 3 via the path [*I/O*] → [*Devices*] (see following illustration).

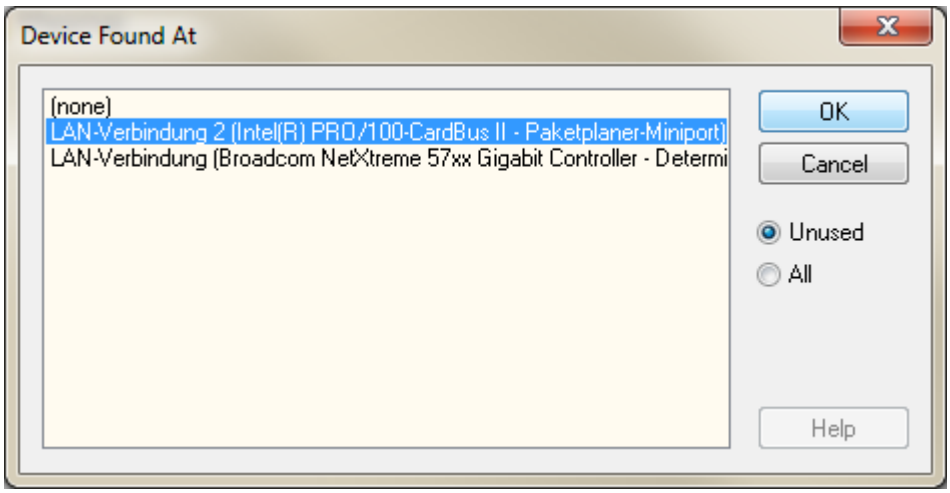
1. In the context menu of the [*Devices*] node, click on the command [*Add New Item...*]
   ⇨ The *Insert Device* dialog opens.
2. Select the *EtherCAT Automation Protocol (Network Variables)* below the EtherCAT node and confirm your selection with [*OK*].



If the PC has several more unused realtime-capable network adapters at its disposal, a dialog appears in which the network adapter can be selected (see following illustration).

3. Select the desired adapter and confirm with [*OK*].

⇨ Subsequently, the EAP device is visible with the designation *Device 1 (EtherCAT Automation Protocol)* underneath the *Devices* node.



In the next step the *Publisher* or *Subscriber Variables* can be configured.

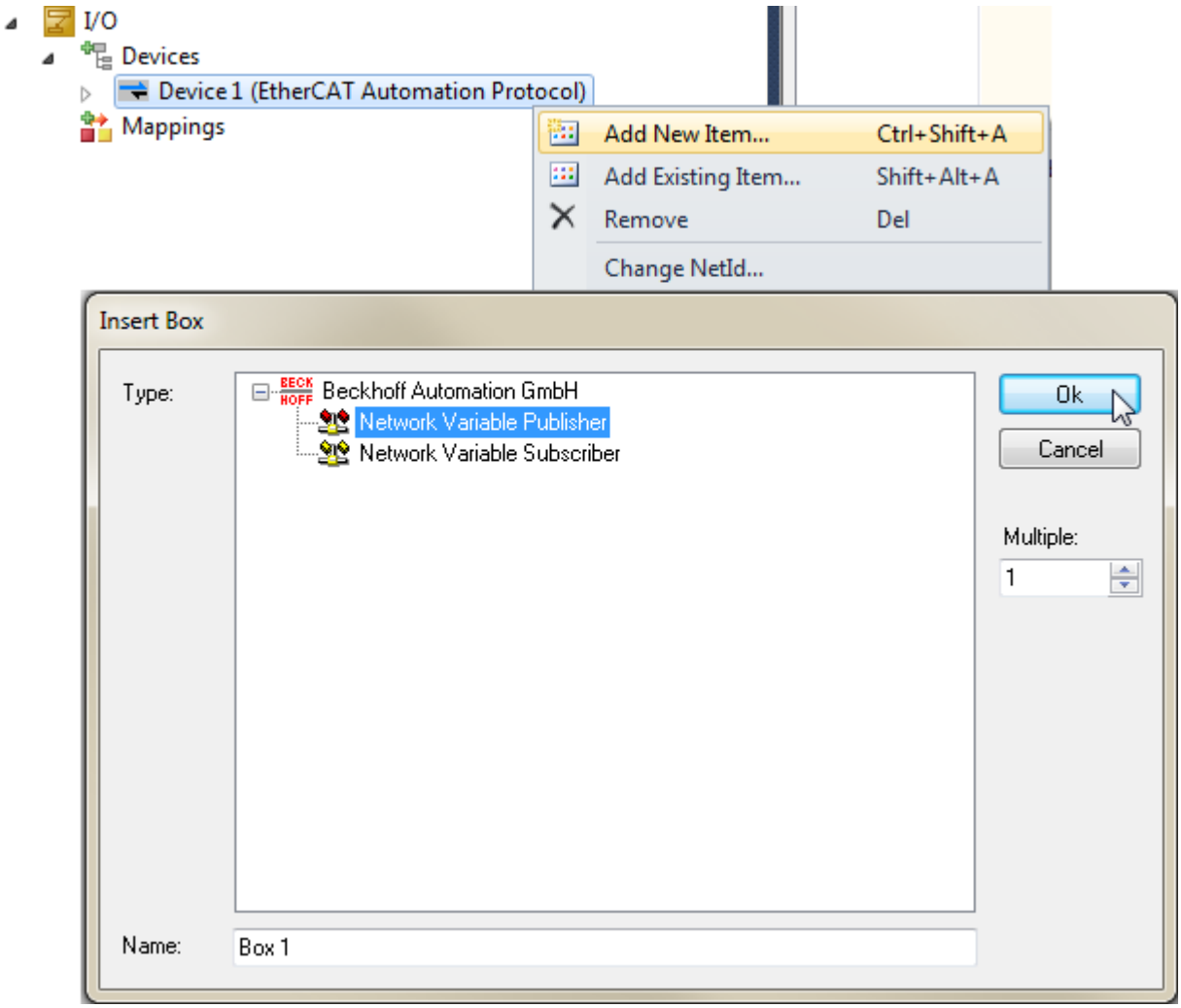
4.2 Addition of publisher variables

If the EAP device is to send variables, *Publisher Variables* must be added in order to complete its configuration (see following illustration).

1. In the context menu of the EAP device node (*Device 1*), click on the entry [*Add New Item*...].
   ⇨ The *Insert Box* dialog opens.
2. Select the *Network Variable Publisher* and click on [*OK*].

Addition of several publishers

If several *Publishers* are to be created, the number can be set using the *Multiple* input field.



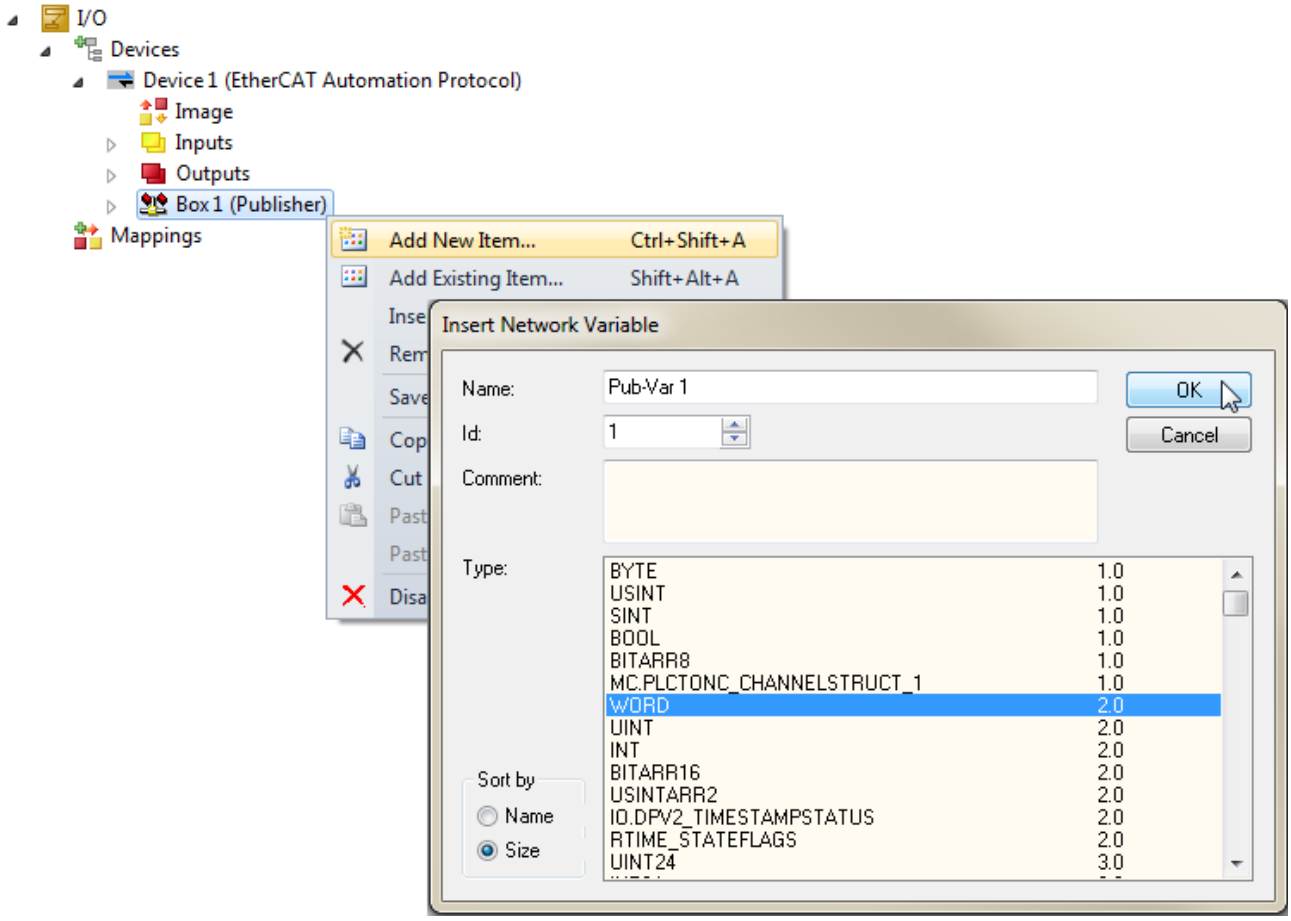
The *Publisher Variables* (*TxData*) are subsequently appended to the created *Publisher* (see following illustration).

3. In the context menu of the *Publisher*, click on the menu item [*Add new Item...*].
   ⇨ The dialog *Insert Network Variable* appears.
4. Select a data type for the *Publisher Variable*, enter a designation in *Name*, define an *ID* and then click on [*OK*].

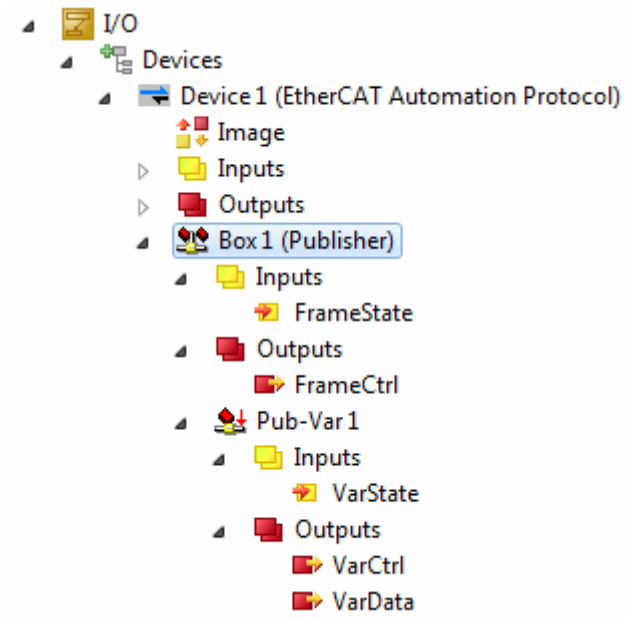
⇨ The node for this *Publisher Variable* appears under the *Publisher* (see next illustration).

Assignment of an ID

The selected *ID* for a *Publisher Variable* should be unique in the network.



In the course of the configuration of publisher variables, input and output variables are generated that belong to the various nodes and sub-nodes of the EAP device (see following illustration). In addition to the unconditional input/output variables there are also conditional ones that are only generated if certain configuration settings are made.



All of the input/output variables together ultimately form the process image of the EAP device. If they are linked with the variables of an application (e.g. a PLC program) in TwinCAT, then the latter can read these contents from the process image or write values into the process image. The purpose of the output variables is to write values that, for example, control the behavior of the EAP device. The purpose of input variables is to read their values in order, for example, to evaluate status information in the PLC program and to react to it. In this way the behavior of the EAP device can be directly influenced from a PLC.

Appearance of the input and output variables

The structure of the process image varies, depending on which port (network adapter of the PLC or an EL66xx Switch Port Terminal) the EAP device uses for communication.

Description of the unconditional input/output variables

FrameState
The *FrameState* input variable below the *Publisher* node indicates the current state of the *TxFrame.* A detailed description can be found in the Publisher [▶ 25] section.

FrameCtrl
The **FrameCtrl** output variable below the *Publisher* can be used to control the transmission of the EAP telegram. The following values are possible for *FrameCtrl*:

Short description	Bit	Description
Disable sending	FC.0	The transmission of the frame is interrupted if the bit is set to 1. The transmission of the frame only restarts when the value falls back to 0.
Remove destination MAC from ARP cache	FC.1	This control box only has an effect if the EAP telegram is sent by IP or *AMSNetID.* The destination *MAC* address entered in the ARP cache is deleted if the bit is set. A new entry can only be made when the bit is reset to 0, provided the EAP device can determine the destination *MAC* address by ARP. (see also Remote station monitoring via ARP [▶ 13])

Appearance of the output variable FrameCtrl

This variable does not exist in the case of EAP communication via the EL66xx terminal! Instead, the output variable *CycleIdx* (size: 16 bits) is created, which is to be used for a diagnosis of the communication connection (see Diagnosis of an EAP connection [▶ 22]).

VarState
The input variable *VarState* below the *Publisher Variable* indicates the current state of *TxData.* A detailed description can be found in the Publisher [▶ 25] section.

VarCtrl
The output variable *VarCtrl* below the *Publisher Variable* can be used to control the transmission of the *Publisher Variable*. The following values are possible for *VarCtrl*:

Short description	Bit	Description
Disable publishing	VC.0	The transmission of the *Publisher Variable* is interrupted. The transmission of the *Publisher Variable* restarts only when the value of the bit has returned to 0.

Appearance of the output variable VarCtrl

This variable does not exist in the case of EAP communication via the EL66xx terminal!

CycleIndex
The output variable *CycleIndex* below the *TxProcessData* should be linked with an application variable that is cyclically incremented, since the *CycleIndex* on the receiving side can be evaluated for diagnostic purposes (see Diagnosis of an EAP connection [▶ 22]).

Appearance of the output variable CycleIndex

The output variable CycleIndex only exists if the EAP communication takes place via the EL66xx terminal!
This variable does not exist if the EAP communication takes place via a PC network adapter. The *CycleIndex* in the EAP telegram is then automatically incremented with each task cycle.

VarData
The output variable *VarData* of the *Publisher Variable* can be linked with any desired variable of a suitable data type (e.g. with the variables of a PLC program).

Description of the conditional output variables

MAC / NetID / IP
The output variable *MAC / NetID / IP* below the *Publisher Box* node only exists if the option *Target Address Online Changeable* is activated. In this case the destination address of the EAP telegram can be changed dynamically (e.g. with the help of a PLC program).
Depending on the type of addressing configured for *TxFrame* (MAC Address, AMS NetID or IP), the data type of the variables is a MAC, a NetID or an IP.

VarId
The output variable *VarId* below the *Publisher Variable* only exists if the option *Online Changeable* is activated for the variable *ID.* In this case the *ID* for the TxData can be changed dynamically (e.g. with the help of a PLC program).

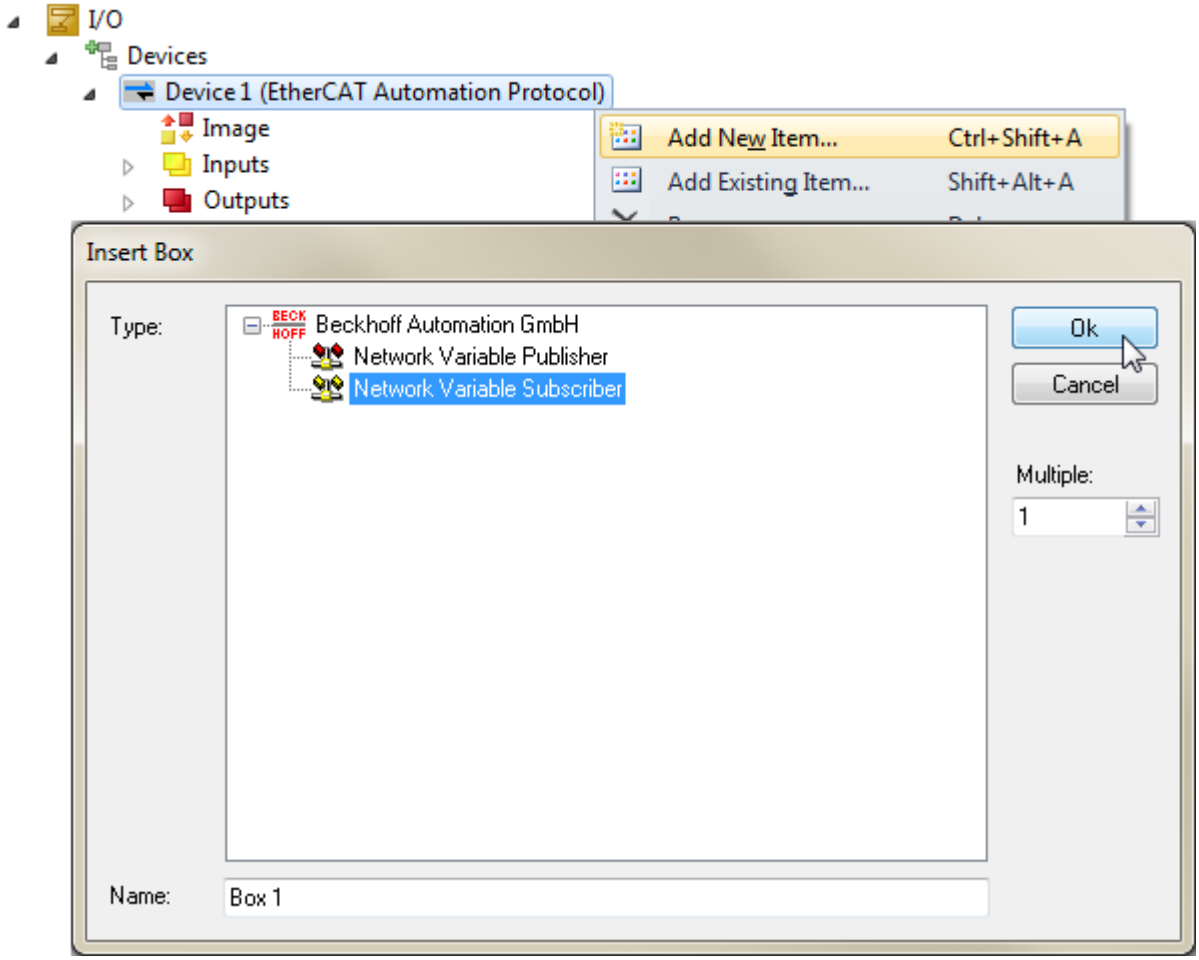
4.3 Addition of subscriber variables

If the EAP device is to receive variables, subscriber variables need to be added in order to complete its configuration (see next illustration).

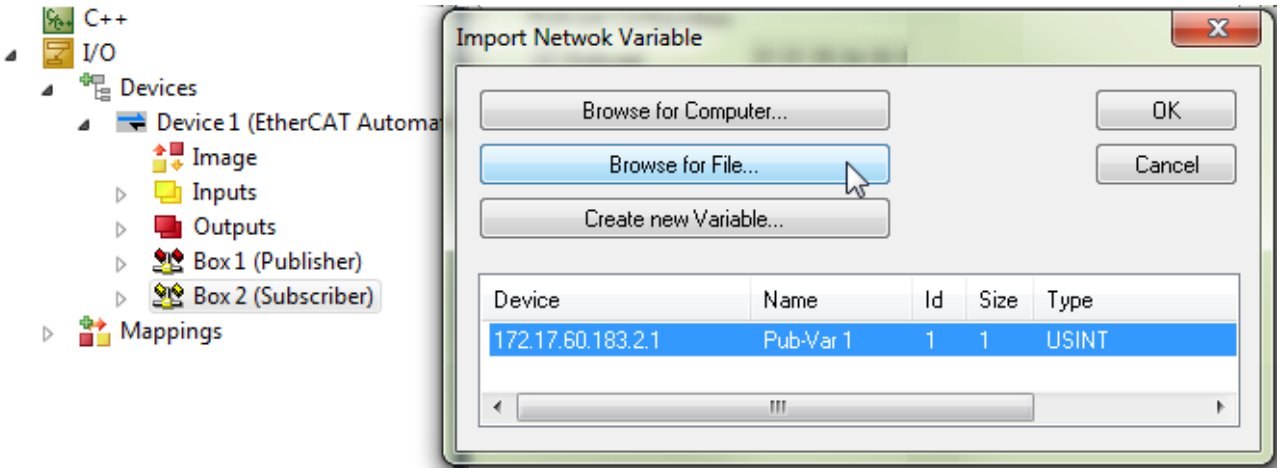
1. In the context menu of the EAP device node (*Device 1*), click on the entry [*Add New Item*...].
   ⇨ The *Insert Box* dialog opens.
2. Select the *Network Variable Subscriber* and click on [*OK*].

Addition of several subscribers
If several *Subscribers* are to be created, the number can be set using the *Multiple* input field.

The *Subscriber Variables* (*RxData*) are finally appended to the *Subscriber* generated (see following illustration).



3. In the context menu of the *subscriber*, click on the menu item [*Add new Item...*].
   ⇨ It opens the *Import Network Variable dialog,* with whose help a *Subscriber Variable* can be imported or defined. There is a choice of three possibilities.



Browse for Computer
The connection to a *Publisher Variable* of another controller in the network can be established automatically.

✓ The other controller must be located in the list of known target systems.

4. Click on [*Browse for Computer…*].
   ⇨ The *Choose Target System* dialog opens.
5. Select the desired control computer from the list and click on [*OK*].
   ⇨ All *Publisher Variables* offered by this computer are listed.
6. Select the desired *Publisher Variable* and confirm with [*OK*].

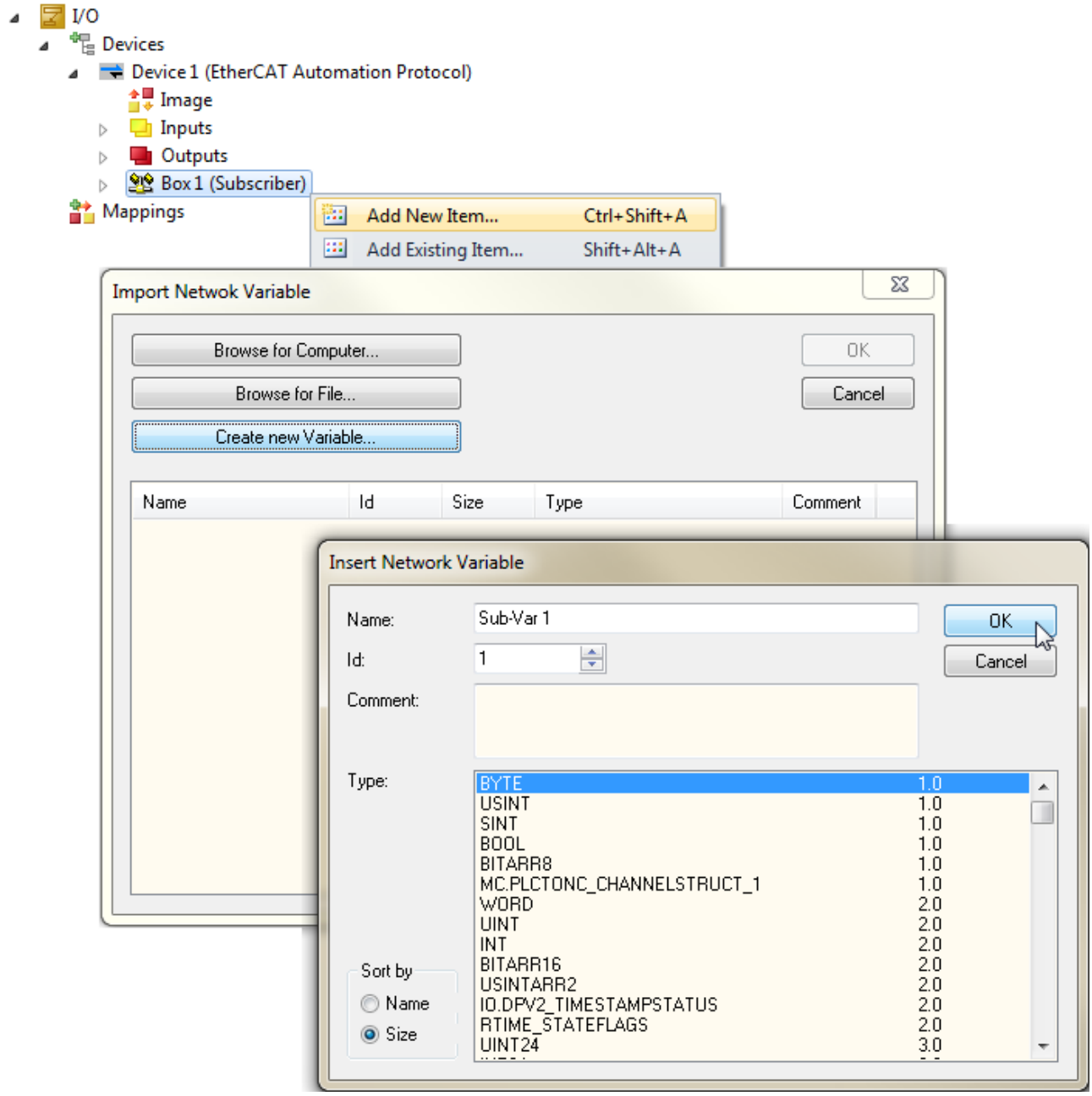
⇨ The *Subscriber Variable* is created to suit the selected *Publisher Variable*.

Browse for File
As an alternative to *Browse for Computer*, the connection to a variable of another control computer can be automatically established by browsing for the project file that is or will be activated on the control computer.

1. Click on [*Browse for File…*].
   ⇨ A dialog opens for browsing the file system.
2. Browse the file system for the desired project file and confirm your selection with [*OK*].
   ⇨ All *Publisher Variables* offered by this computer are listed.
3. Select the desired *Publisher Variable* and confirm with [*OK*].

⇨ The *Subscriber Variable* is created to suit the selected *Publisher Variable*.

Create new Variable
The last option is to manually configure the *Subscriber Variable* (see following illustration).

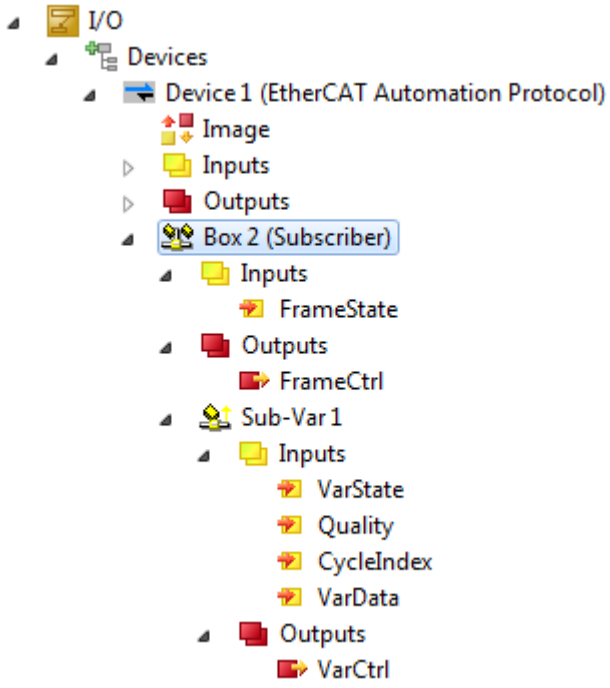


1. Click on [*Create new Variable*].
   ⇨ The dialog *Insert Network Variable* appears.
2. Select a data type for the *Subscriber Variable*, enter a designation in *Name*, define an *ID* and then click on [*OK*].

⇨ The node for this *Subscriber Variable* appears below the *Subscriber* (see following illustration).

Assignment of an ID

The selected *ID* for a *Subscriber Variable* must be identical to the *Publisher Variable ID* that is to be received.



Description of the unconditional input and output variables

FrameState/FrameCtrl
The input variable *FrameState* and the output variable *FrameCtrl* below the *Subscriber* node are reserved and are not used at present.

VarState
The input variable *VarState* below the *Subscriber Variable* indicates the current state of *RxData*. A detailed description can be found in the Subscriber [▶ 22] section.

VarCtrl
The output variable *VarCtrl* below the *Subscriber Variable* can be used to control the receiving. The following values are possible for *VarCtrl*:

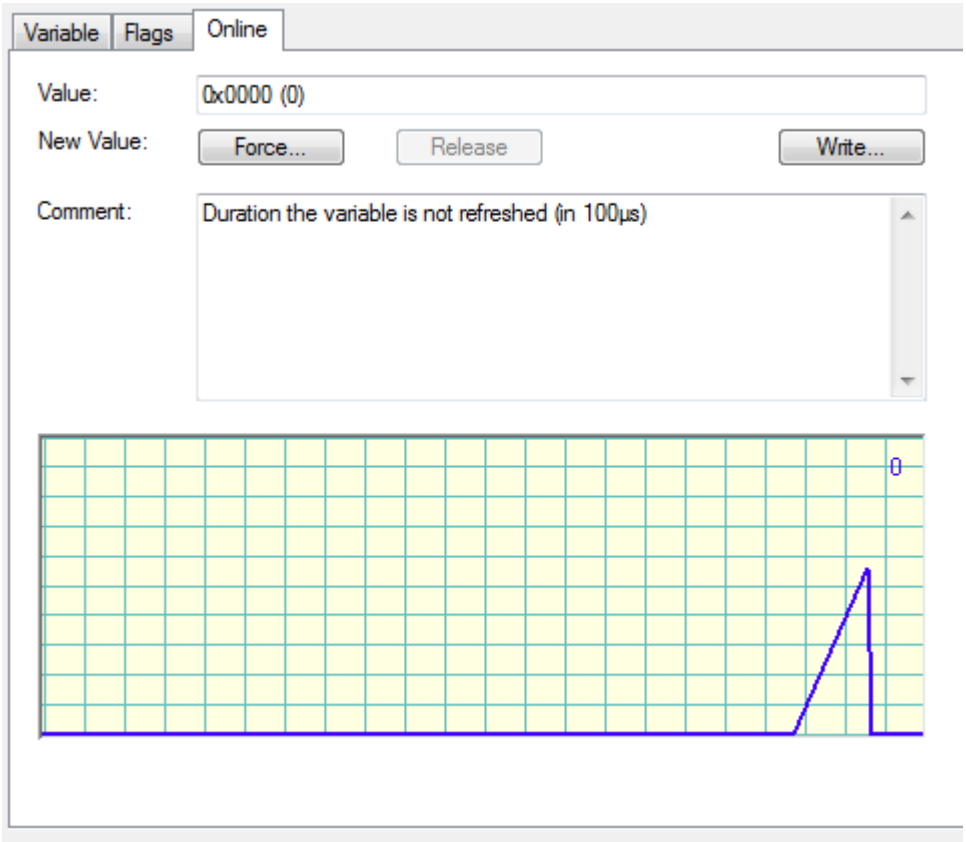
Short description	Bit	Description
Ignore Hash/Version	VC.0	If the bit is set to the value 1, the checking of the version on receiving a *Process Data* is deactivated.

Appearance of the output variable VarCtrl
This variable does not exist in the case of EAP communication via the EL66xx terminal!

VarData
The input variable *VarData* of the *Subscriber Variable* can be linked with any desired variable of a suitable data type in TwinCAT (e.g. with a variable of a PLC program).

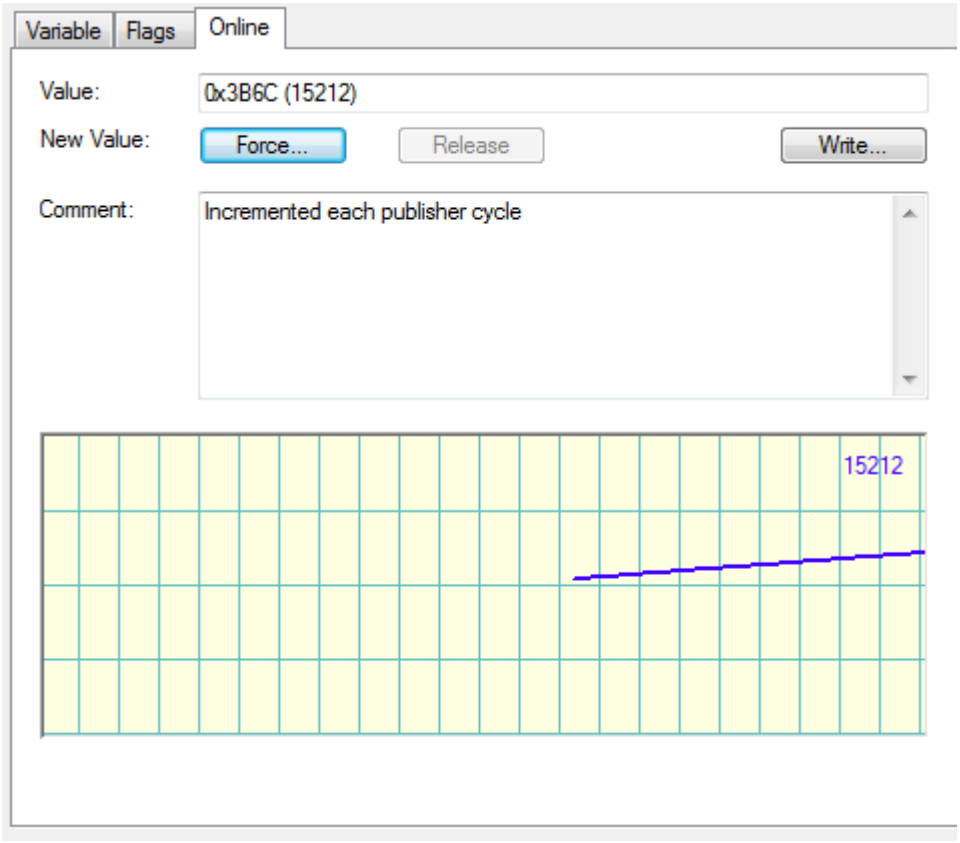
The reception of a variable is diagnosed on the receiving side. The two input variables *Quality* and *CycleIndex* below the *Subscriber Variables* are available for this.

Quality
The *Quality* variable supplies a counter with a resolution of 100 µs. The counter value indicates the length of time since the last data was received for this *Subscriber Variable*. The example in the next illustration shows the online value of the *Quality* variable after disconnecting the network plug (count increases) and reconnection (counter value 0).



CycleIndex

The *CycleIndex* variable is incremented in each *Publisher* cycle. The example in the next illustration shows the typical procedure when viewing the online value of the *CycleIndex* variables.



Description of the conditional output variables

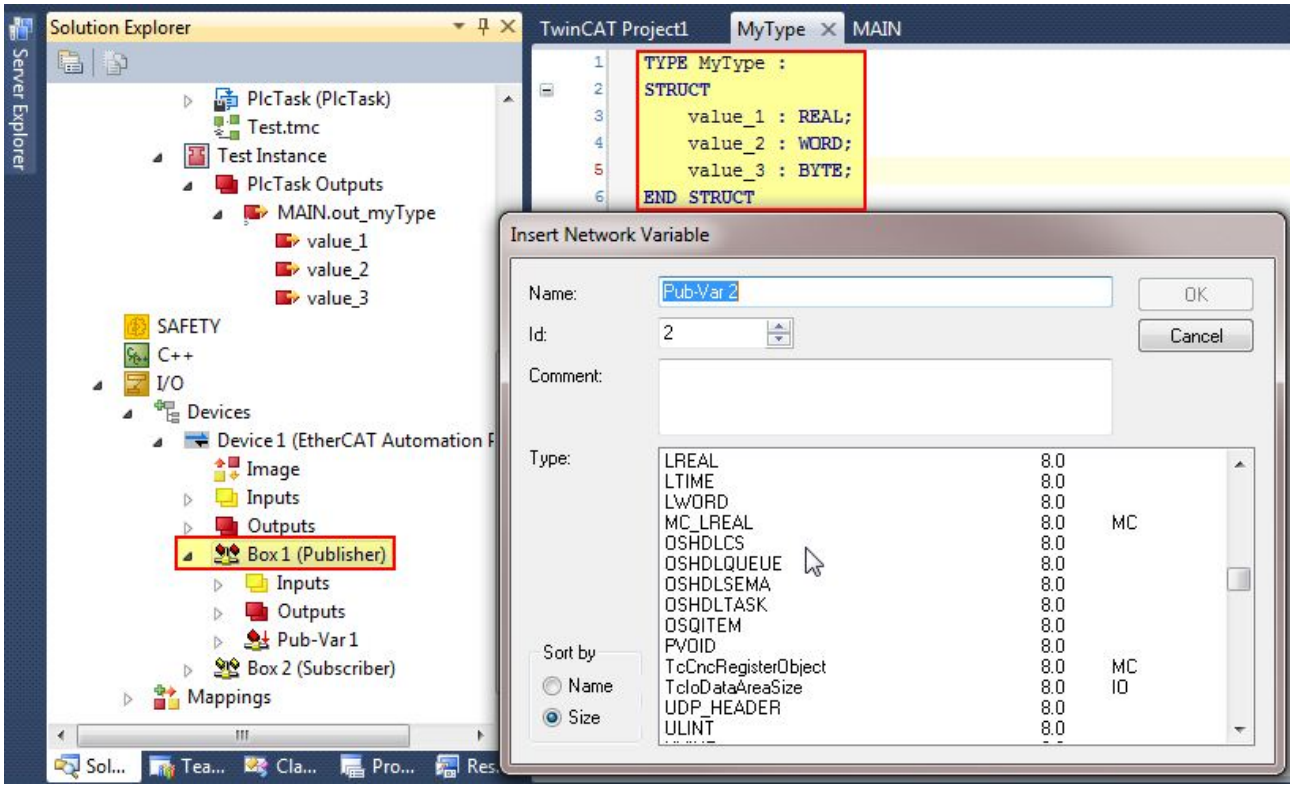
VarId
The output variable **VarId** below the *Subscriber Variable* only exists if the option *Online Changeable* is activated for the *Variable ID*. In this case the *ID* for the *Subscriber Variable* can be changed dynamically (e. g. with the help of a PLC program).

4.4 Use of user-defined data types

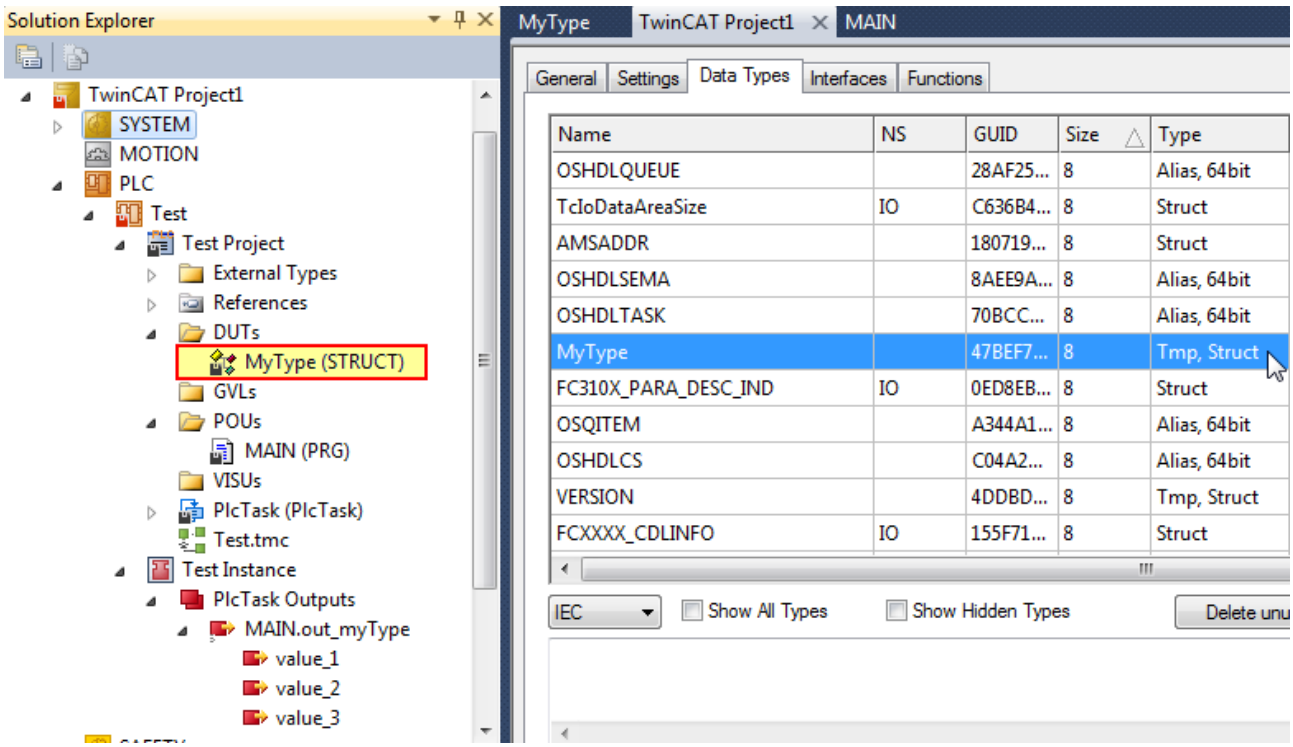
Creation of an EAP variable

There are two common methods in TwinCAT to create user-defined data types. On the one hand a data type of your own can be created via the *System* node of the project tree on the *Data Types* tab (see next but one illustration). Such a data type is then available to all modules of the TwinCAT project. It is thus a global data type.
On the other hand, users often create a data type of their own inside a PLC project by defining a DUT (Data Unit Type). Such a data type is initially only locally available to the PLC project. This data type is hidden to other modules such as the I/O configuration (and thus also the EAP device).

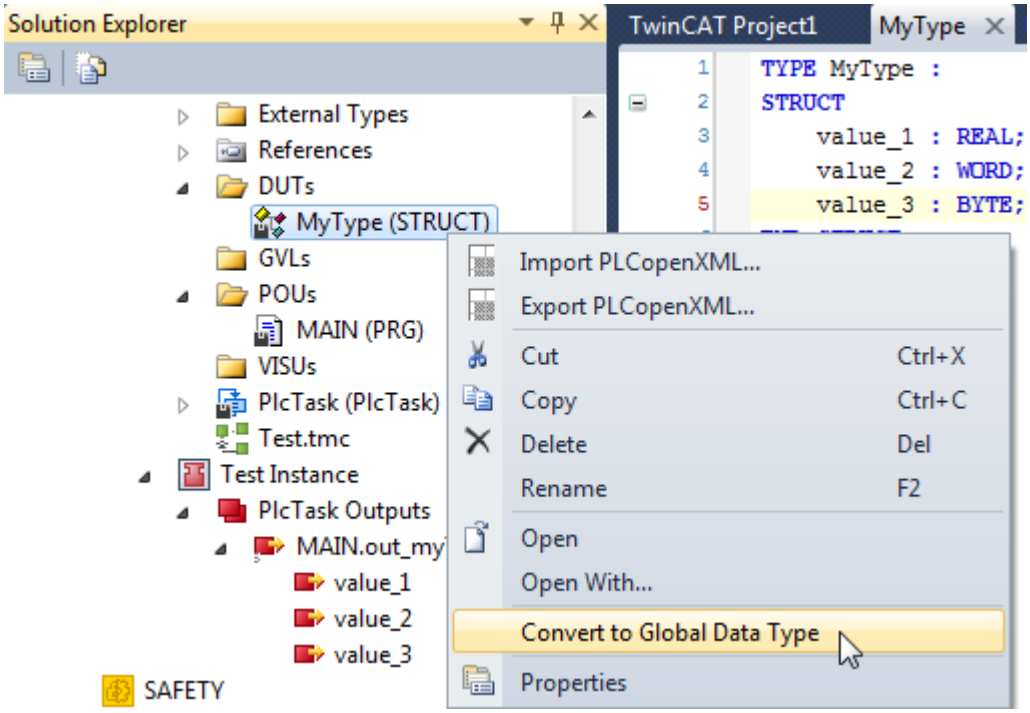
In the following illustration you can see that the data type *MyType* is defined in the PLC. This data type has additionally been used for an output variable of the PLC program. Nevertheless, the user-defined data type does not appear in the list of available data types if a variable of this data type is to be created in the *Publisher Box* of the EAP device.



A corresponding note that a data type is used only locally can be found on the basis of the data type list on the *Data Types* tab via the *System* node. In the following illustration the data type with the name *MyType* appears in the list. However, there is a remark under the *Type* property that the data type concerned is a temporary one *(Tmp)*. This means that the data type concerned is one that is not global.



If it should be necessary to use a local data type from the PLC in a global context, the corresponding *DUT* can be converted into a global data type (see following illustration).



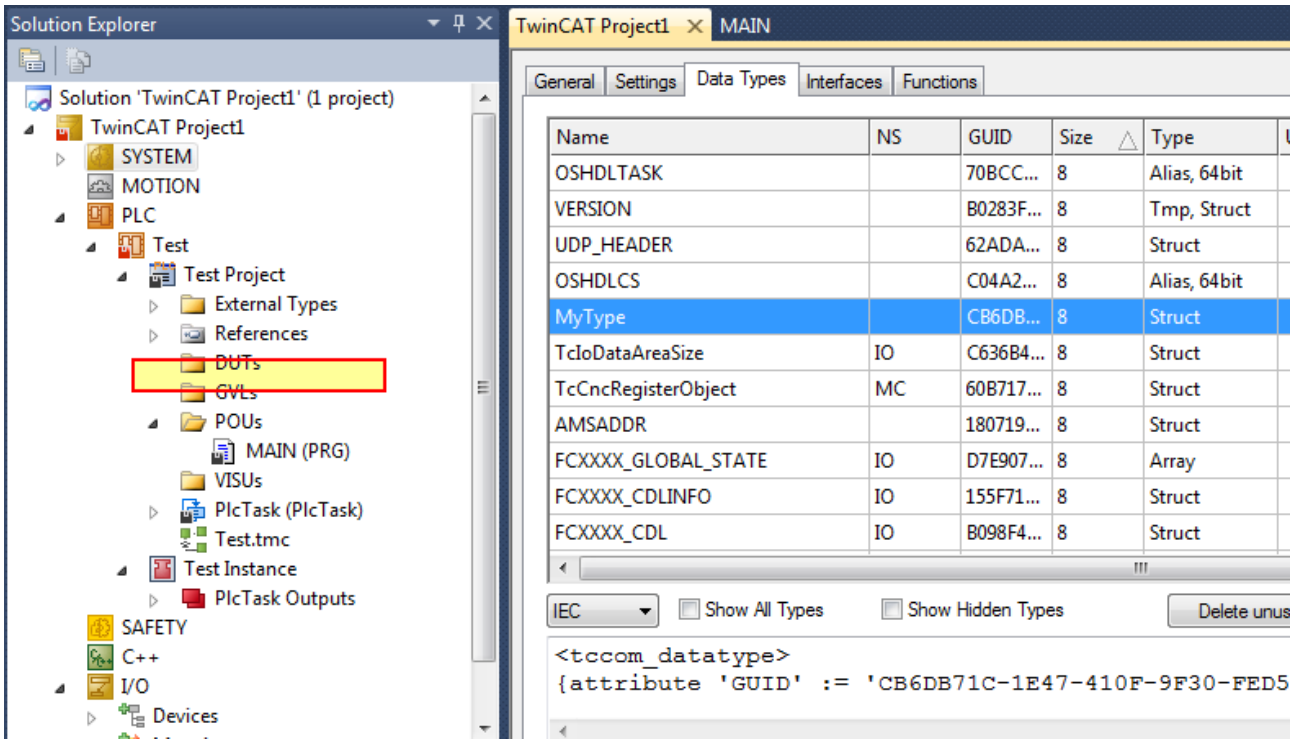
1. Click on the command [*Convert to Global Data Type*] in the context menu of the *DUT*.

⇨ The node of the *DUT* is automatically removed from the PLC project and a global declaration of the type is created (see following illustration).

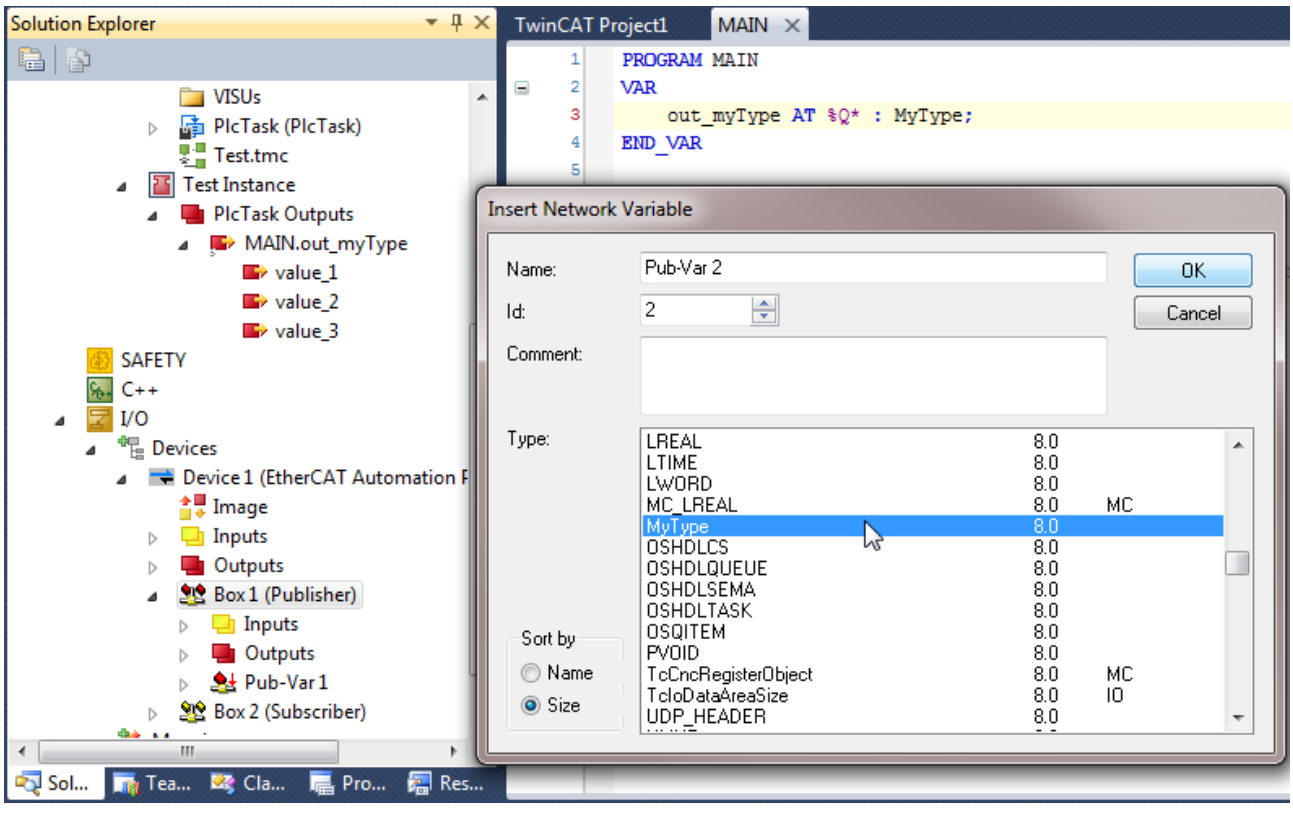
The description for one and the same data type must be unique in TwinCAT. For this reason the definition of the data type (the original DUT) is removed from the PLC. Instead of that, the definition of the data type can be found in the XML-based TwinCAT project file following the conversion, and in this way the data type is globally available to the entire TwinCAT project.

A data type occurs as a local and global variant

Following the conversion of a data type into a global data type, the latter appears twice in the list of data types: as a local data type and also as a global data type. The local data type is only removed from the list if it is no longer referenced. It is usually necessary to compile the PLC project again after the conversion. As a result, the reference from the PLC program to the local data types is deleted and only the global type is referenced. Accordingly, the data type should only appear once in the list as in the following illustration.

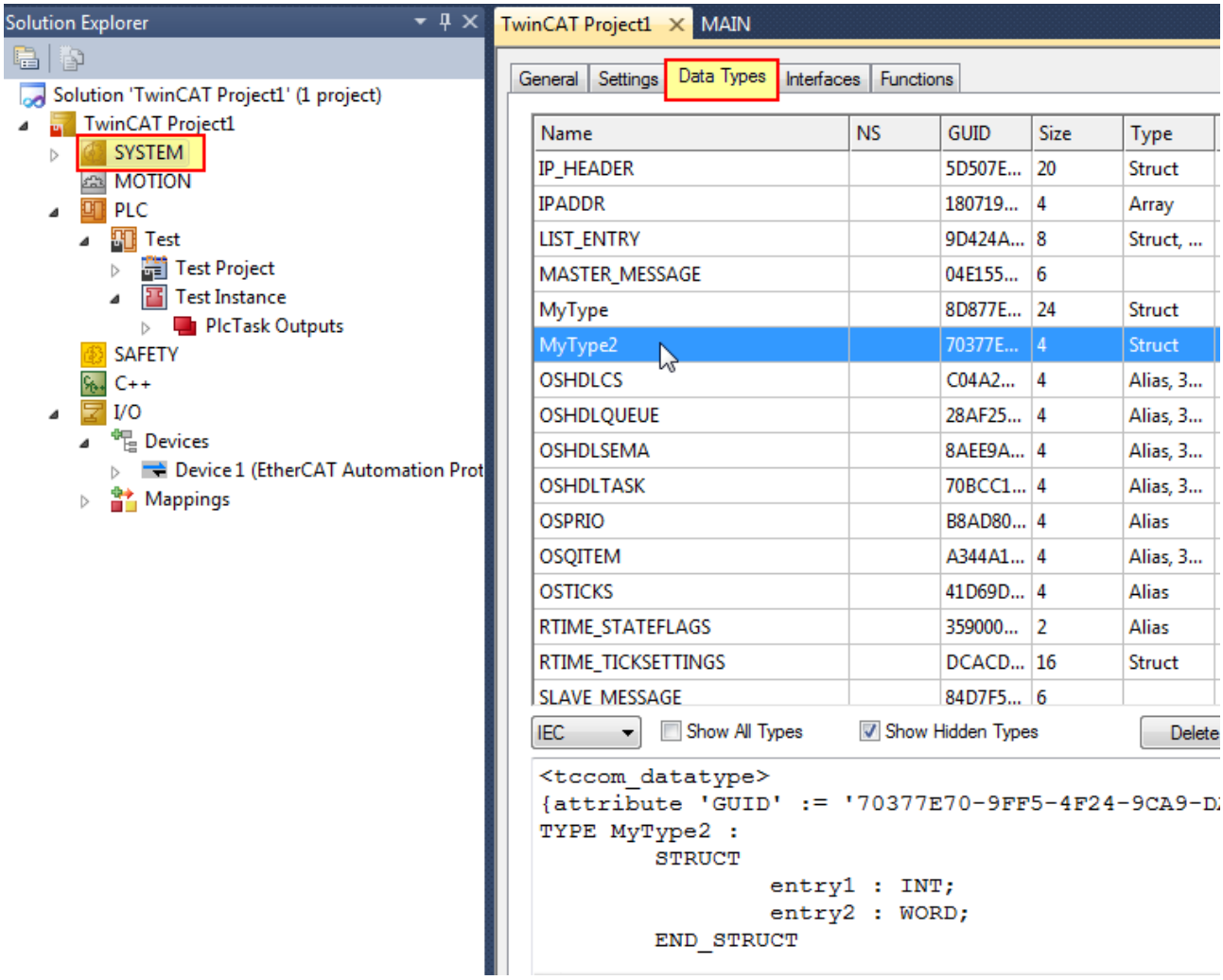


Once the user-defined data type has been converted to a global data type, it can be selected from the list of available data types during the creation of a *Publisher* or *Subscriber Variable* in the EAP device (see following illustration).

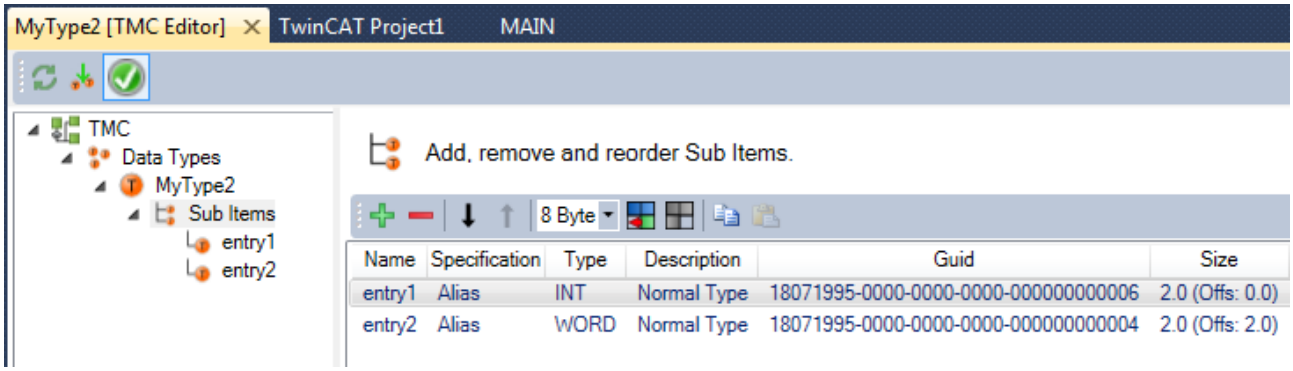


Changing a global data type

Once a data type has been converted to a global data type, its definition is no longer under the control of the PLC project. The definition of the data type is located in the XML-based TwinCAT project file. This data type must now be changed in the TwinCAT project via the *Data Types* tab of the *System* node (see following illustration).



1. Select the data type that you wish to change from the list.
2. Right-click on it and select the command [Edit] in the context menu.
   ⇨ The *TMC Editor* opens (TMC = TwinCAT Module Configuration).



3. With the help of the *TMC Editors* you can change the data type as desired and subsequently save it.

⇨ The change is accepted in the TwinCAT project by creating a new version of the data type following the saving procedure and marking the original version in the TwinCAT project as *Hidden*.

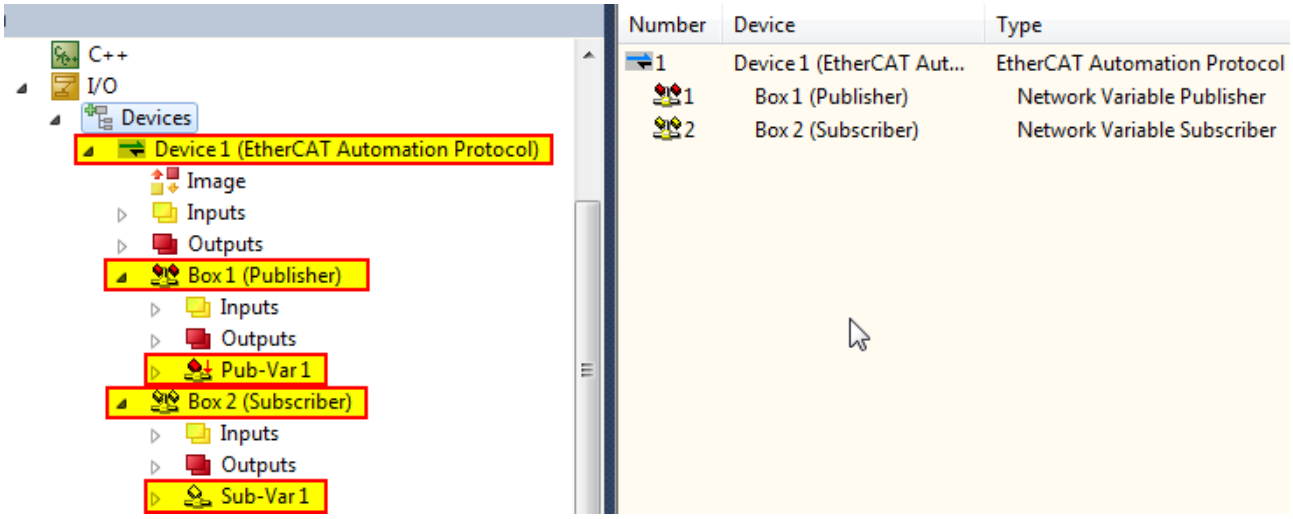
A PLC program that uses the data types concerned automatically uses the latest version of the data types. So that the change of the data type is also accepted in the machine code, the PLC project must be recompiled following the change.

If a PLC variable of this data type exists and it is linked with a corresponding *Publisher* or *Subscriber Variable* of an EAP device, the latest version of the data type is also used with the *Publisher/Subscriber Variable* as soon as the PLC project is recompiled.
If no linking of EAP variables to PLC variables exists, the EAP device continues to use the original old version of the data type for its variables. If the new version is to be used instead, the EAP variable concerned must initially be deleted and a new EAP variable of the desired data type added again. The old version of the data type is retained in the TwinCAT project until no further reference to this old version exists.

5 Configuration of an EAP device

An EAP device created using TwinCAT as described in chapter Creation of an EAP configuration [▶ 26] is initially configured with standard settings. The standard settings are selected such that the user only needs to ensure that the order of the data variables on the receiver side is identical to that on the sender side and that the same data type is selected for the data variables to be transmitted on both the sender and receiver side. An EAP connection configured in this way always communicates in *Pushed Data Exchange Mode* (see Communication methods [▶ 10]).

The EAP communication can be freely configured with the help of the configuration options of the EAP device and the subordinated boxes (*Publisher/Subscriber*, *Publisher/Subscriber Variable* – see following illustration). Furthermore it is possible to read out information about the current configuration of an activated EAP device.



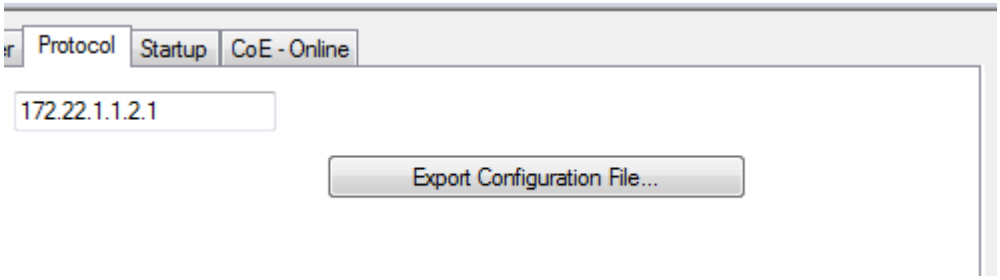
5.1 The TwinCAT EAP device

The Network Interface Card (NIC) via which the EAP telegrams are to be sent and the *AMS NetID* with which the EAP device is to be reachable by *ADS/AMS* are specified with the help of the EAP device's configuration options. By selecting the Network Interface Card, the IP address (in the case of UDP/IP communication) via which the EAP device can be reached is then automatically specified. In addition, there is a possibility to access the object dictionary of the EAP device.

General
The standard dialog on the *General* tab exists for all TwinCAT devices and boxes. A descriptive name and a useful comment on the description of the device or box can be entered in this dialog.

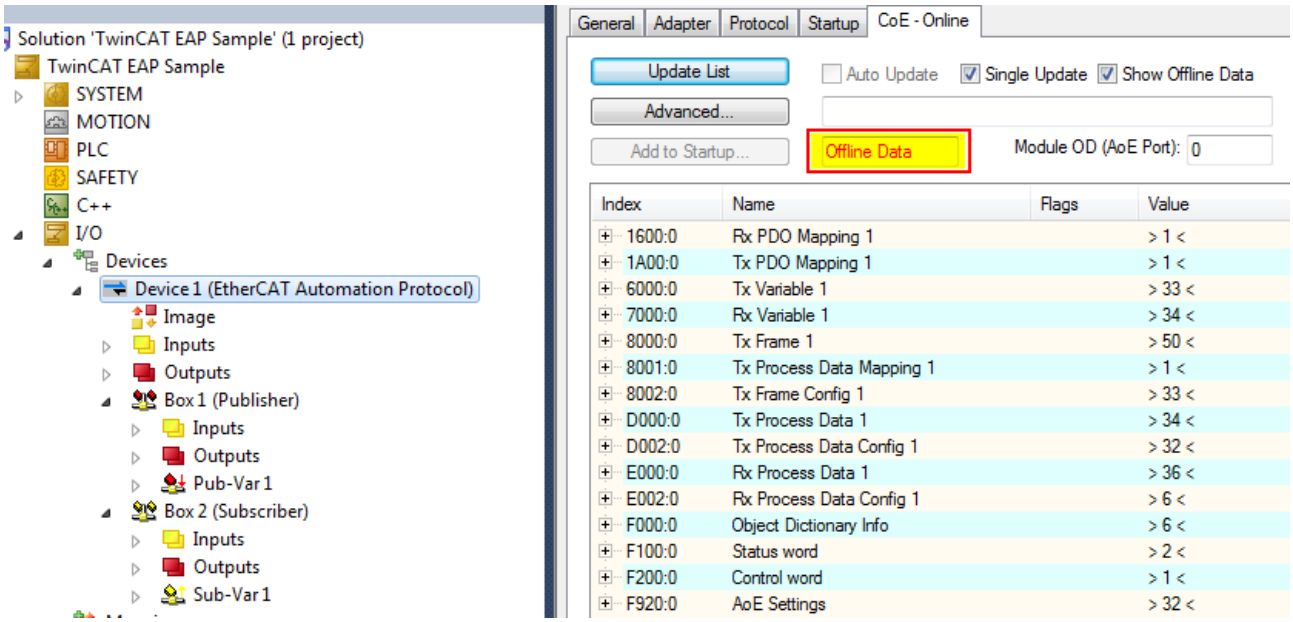
Adapter
The dialog on the *Adapter* tab shows the selected Network Interface Card or enables an adapter to be assigned.

Protocol
The dialog on the *Protocol* tab (see next illustration) enables the assignment of a special *AMS NetID* via which the EAP device can be addressed by *ADS/AMS* during operation.
Furthermore there is an option by means of the [*Export Configuration File…*] button to export the current EAP configuration revision of the loaded project to an XML file. This XML file has a defined scheme and is also called the *EAP Device Configuration* (*EDC*).



CoE – Online

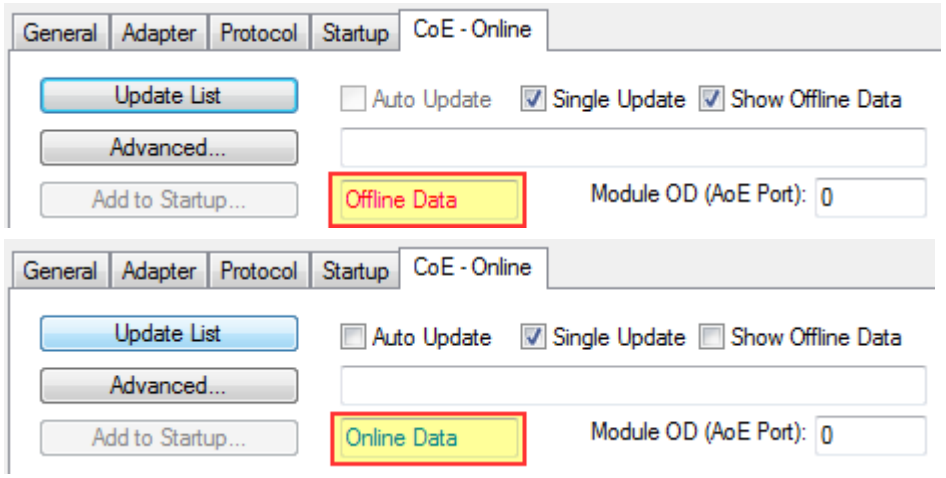
The dialog on the *CoE - Online* tab shows the EAP object dictionary (OD). Compare here the chapter The CANopen object dictionary [▶ 52] and the following illustration.The object dictionary is automatically created as soon as a TwinCAT EAP device is configured with the help of TwinCAT. It includes all configuration information and is automatically extended or reduced by entries as soon as elements are added to or removed from the current configuration.



The control elements of the CoE - Online dialog have the following meanings:

Status

The status of the displayed object dictionary is output in a text field (with a yellow background in the illustration above). The status *Offline Data* is always displayed if TwinCAT has no connection to an activated EAP device. For example, a connection is not established if the configured *AMS NetID* differs from the actual *AMS NetID* of the activated EAP device. Otherwise the status *Online Data* is displayed:



In the online directory	In the offline directory
the real current directory of the EAP device is read out. This may take several seconds, depending on the size and cycle time	the offline directory of the EAP device is displayed. In this case modifications are not meaningful or possible.
a green **Online** can be seen in the TwinCAT dialog *CoE - Online*	a red **Offline** can be seen in the TwinCAT dialog *CoE - Online*

Reading the online data of another EAP device instance

There is an option to set the *AMS NetID* on the Protocol tab to the *AMS NetID* of any desired EAP device within the network in order to read out the online data by TwinCAT via the *CoE – Online* tab. To do this the EAP device must be activated and an *ADS/AMS Route* must exist to the device.

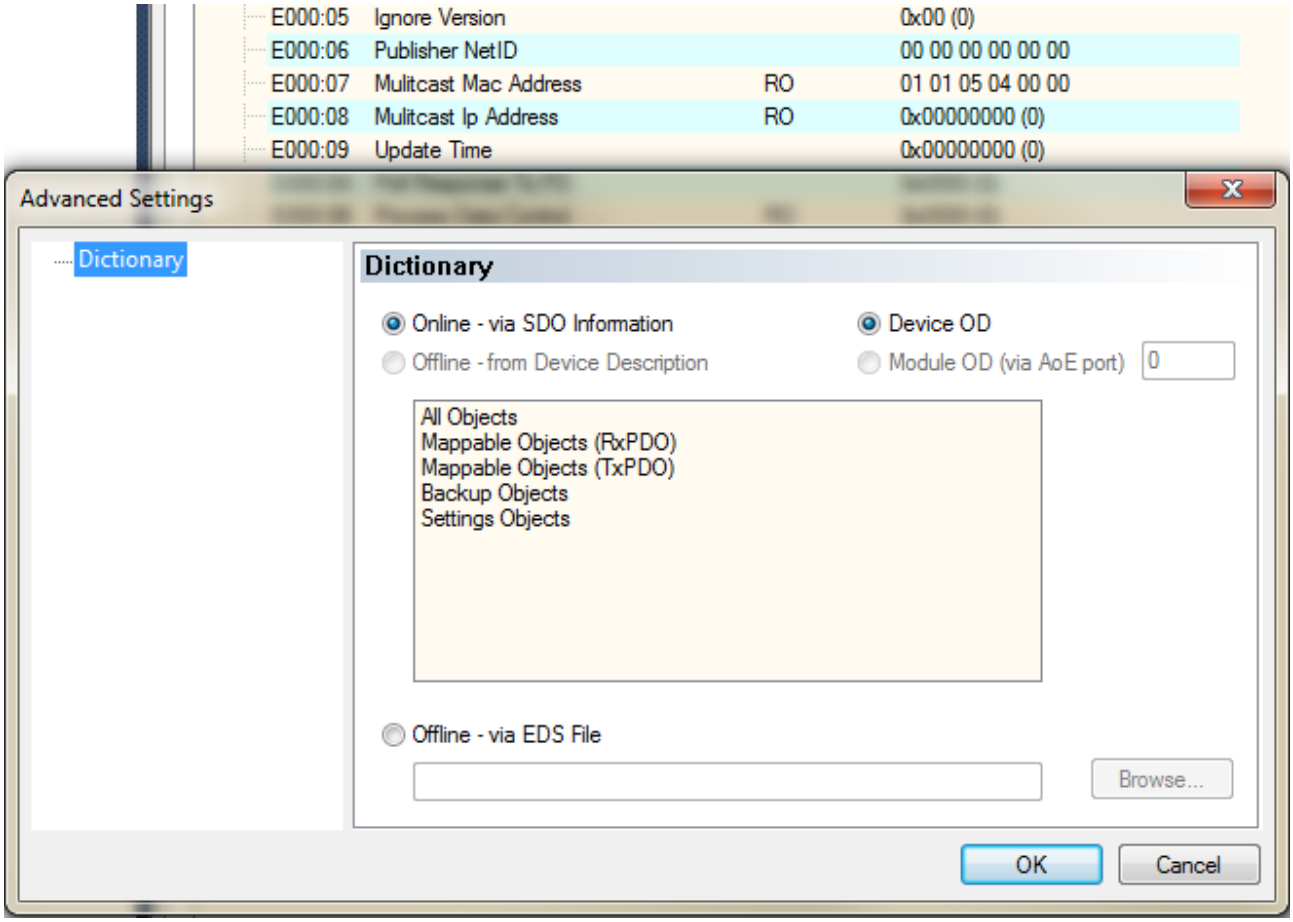
Show Offline Data
With the aid of the *Show Offline Data* option you can set whether the contents of the object dictionary are to be displayed online or offline. Online means that the *OD* contents of the activated configuration are read from the EAP device and displayed. Offline means that the *OD* contents of the configuration that was configured with the help of TwinCAT in the currently loaded TwinCAT project are displayed.

Single Update
If the option *Single Update* is marked, the *OD* contents are always read from the EAP device precisely when an object is expanded (click on the "+" symbol) in order to display its sub-entries, or if you scroll within the window.

Auto Update
If the *Auto Update* option is marked, the *OD* contents for all visible sub-entries are read cyclically from the EAP device and the display is updated.

Update List
The purpose of the *Update List* button is to read the current *OD* contents of all visible object entries from the EAP device and to display them.

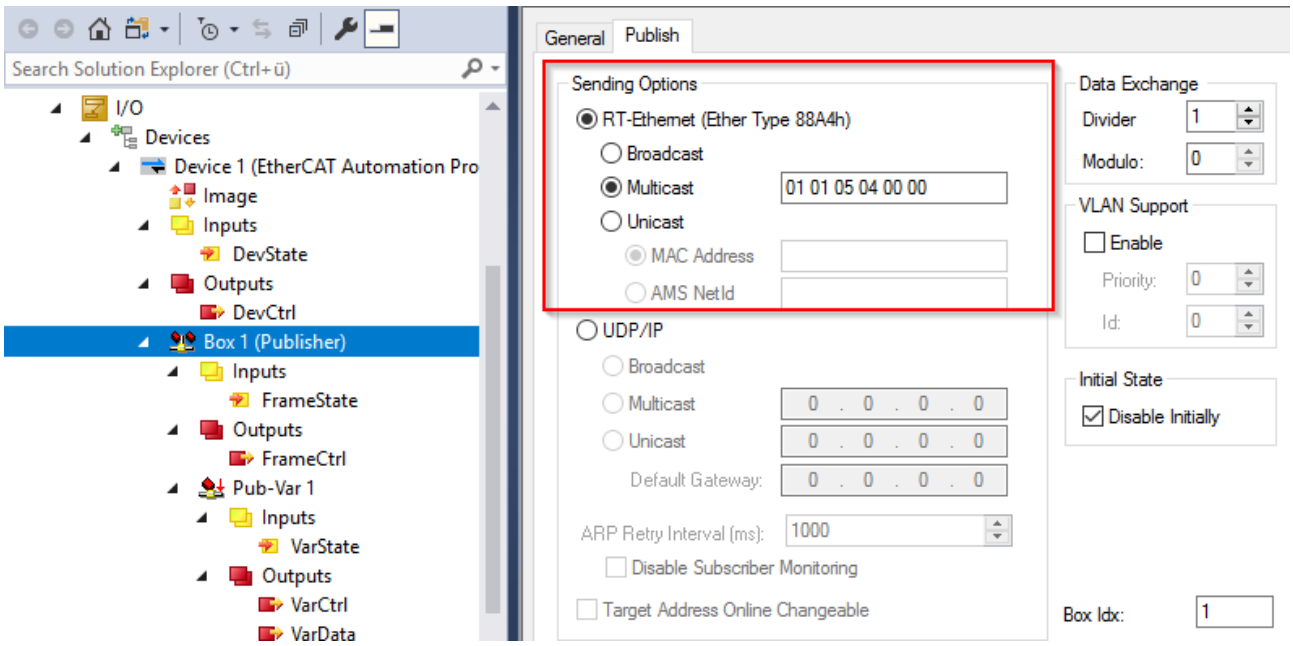
Advanced
The *Advanced Settings* dialog is opened by a click on the *Advanced* button (see following illustration). With the aid of this dialog the entire *OD* description or part of it can be read from the EAP device. This option is advantageous in particular when objects are added to or removed from the object dictionary during operation, because the displayed *OD* description in TwinCAT is then no longer consistent with the current *OD* of the EAP device.



Startup
From the object dictionary, TwinCAT generates a data stream consisting of a series of *Startup* commands that are transmitted to the EAP device. The transmission takes place as soon as the existing configuration is activated. The generated startup commands are displayed in the dialog on the *Startup* tab.

5.2 Publisher Box

The basic protocol to be used for sending the EAP telegram is set on the configuration page of the *Publisher Box* (see following illustration) in the *Sending Options.* The two possible basic protocols *Ethernet Protocol* und *User Datagram Protocol* are introduced in the section Network Protocols of the chapter Communication methods [▶ 10] in the chapter Basic principles [▶ 8]. In each of the two protocols there is a possibility to configure the three different connection methods *Broadcast*, *Multicast* or *Unicast*. In the case of the connection methods *Multicast* and *Unicast*, a destination address must also be defined with whose help the addressee(s) can be reached in the network.



Broadcast
A *Broadcast* telegram is transmitted by one network device to all other devices in the network. Every recipient of a *Broadcast* message decides for itself whether to process the message or not. A *Broadcast* at *Ethernet Protocol* level is sent to the destination *MAC* address FF:FF:FF:FF:FF:FF and at *UDP/IP* level to the *IP* address 255.255.255.255.

Multicast
A *Multicast* telegram is transmitted by one network device to a selected group of devices in the network. A recipient of a *Multicast* message must know the *Multicast* address to which the message was sent and must report it to its network interface card. The network interface card will otherwise discard the *Multicast* message.
Depending on the basic protocol used, either a *Multicast MAC* address will be directly configured as the destination address, or a *Multicast IP* address converted by TwinCAT into a *Multicast MAC* address will be configured as the destination address. A *Multicast IP* address must be in the range 224.0.0.0 to 239,255,255,255 (IPv4).

Unicast
A *Unicast* telegram is transmitted by a network device to precisely one other network device. If the addressing takes place on the basis of the *Ethernet Protocol*, the *MAC* address of the receiver is configured as the destination address. Alternatively, the *AMS NetID* of the receiver can also be configured. If the telegram is sent on the basis of *UDP/IP*, the *IP* address of the receiver is configured as the destination address (see following illustration).

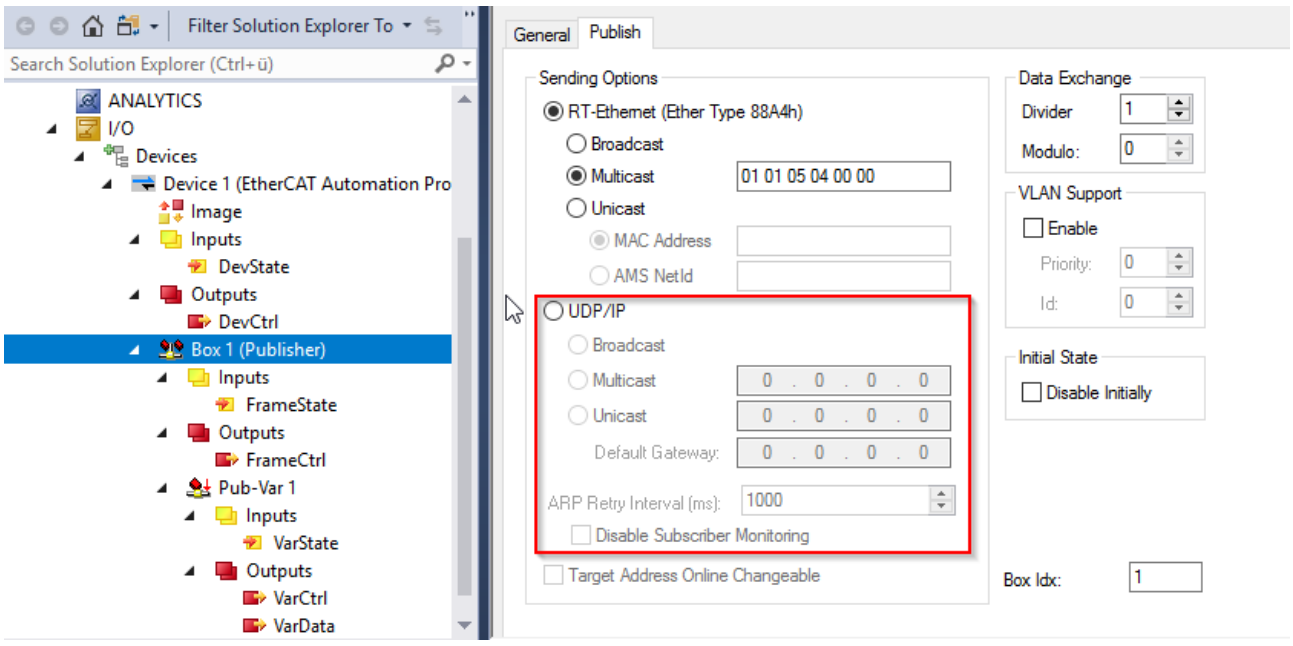
NOTE

Use of broadcast and multicast

EAP telegrams sent as *Broadcast* or *Multicast* at *MAC* or *IP* level cause a higher network load, depending on the cycle time, since they are sent to all network devices! This may cause simple network devices such as printers to crash. With short cycle times all network traffic may become blocked.
In order to avoid a network overload or the overloading of simple, non-realtime-capable network devices, it is recommended

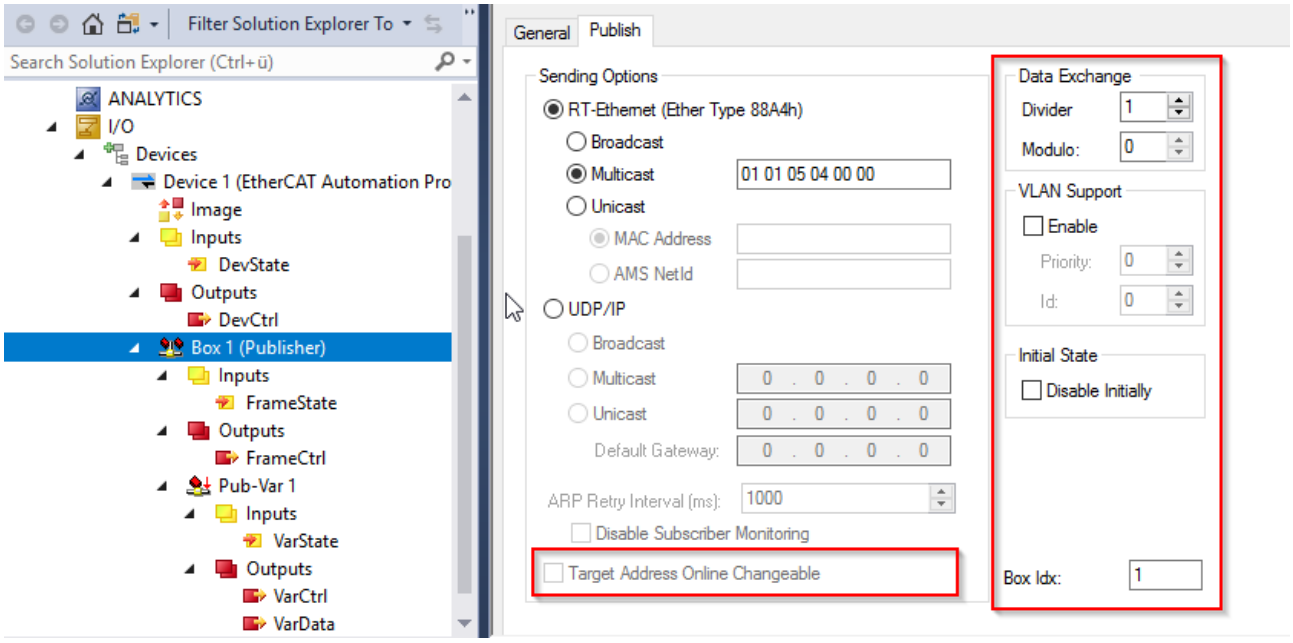
- on the one hand to use *Unicast* addressing and
- on the other to set the cycle time only as small as is absolutely necessary. An explanation regarding the setting of the cycle time can be found further below.

⇨ If the connection type *Unicast* is configured, the *Subscriber Monitoring* mechanism is also configured by default (see Remote station monitoring via ARP [▶ 13]).



Disable Subscriber Monitoring
The *Subscriber Monitoring* mechanism can be deactivated with the help of the option *Disable Subscriber Monitoring.*

ARP Retry Interval
The time set in the input field *ARP Retry Interval* specifies the time interval in milliseconds (*ms*) at which a request is sent to the receiver in order to check its availability.



Target Address Online Changeable
The *Target Address Online Changeable* option is also only usable with a *Unicast*. If this function is activated, a further output variable exists for the *Publisher* in the process image of the EAP device. Depending on which basic protocol is configured, this variable defines an *IP* address, a *MAC* address or an *AMS NetID.* The output variable can be changed with the help of a PLC program. This method can be used to dynamically change the destination address of the configured *Publisher* (refer also to the conditional outputs in the section Addition of publisher variables [▶ 27]).

Data Exchange
The rhythm with which the EAP telegram is sent can be changed with the help of the *Data Exchange* property (see EAP send mechanism [▶ 13]).

Data Exchange

The *Data Exchange* property cannot be used if an EL66xx is in use.

VLAN Support
A fixed route through the *VLAN* (*Virtual Local Area Network*) can be specified for the EAP telegram with the help of the *VLAN Support* property in connection with Managed Switches. If *VLAN* is enabled, the EAP message is furnished with a *VLAN Header*. Accordingly, there are two properties for determining the required *VLAN* and for specifying a *priority* for the processing of the message within a virtual network:

- *VLAN Info ID*: defines the *ID* of the *VLAN* (range between 0 and 4095), in which the message is to be sent, and
- *VLAN Info Priority*: defines a *priority* for the message in the *VLAN* (high priority = 7, low priority = 0).

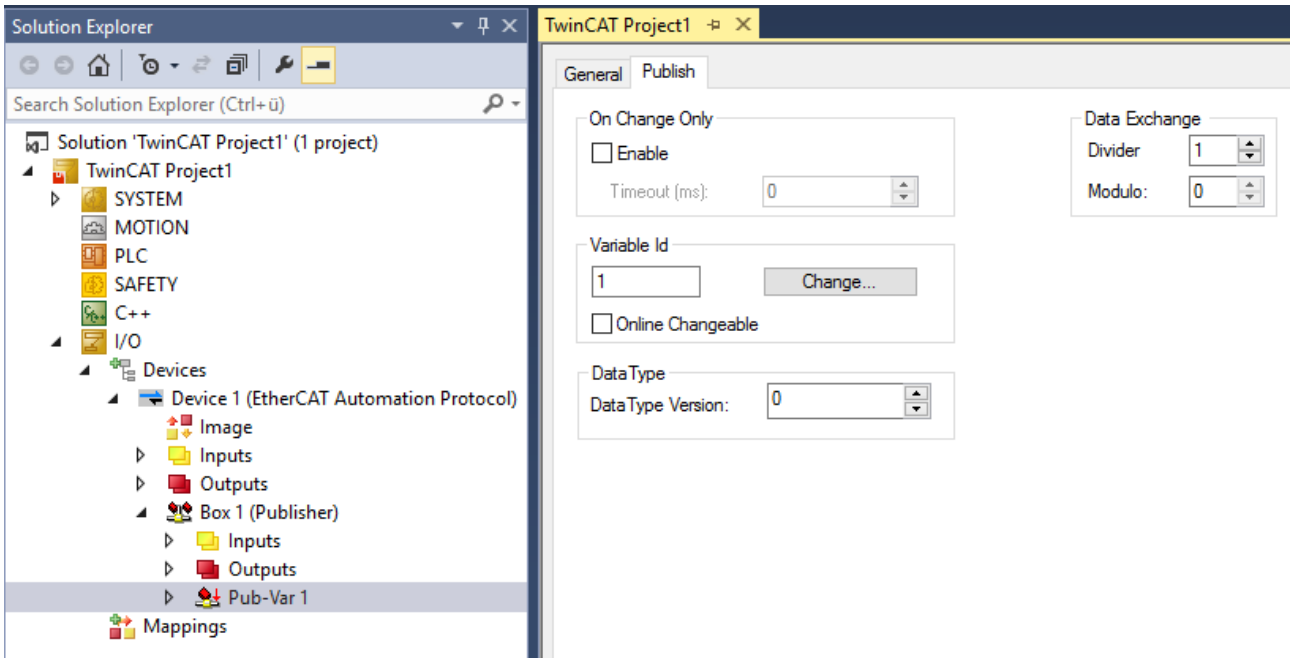
Initial State

With the help of the property “Disable Initially”, you can prevent the publisher from sending packets after starting the system. Sending can subsequently be enabled by setting the FrameState to the value 0.
Box Idx:

Sequential number for the different publishers and subscribers, read-only.

5.3 Publisher Variable

Each *Publisher Variable* additionally has its own special properties that can be parameterized with the help of its own configuration page (see following illustration).



On Change Only
If the *On Change Only* option is activated, the *TxProcessData* is only sent with the *TxFrame* if the value of the *Publisher Variable* has changed. The value in the *Timeout* field specifies the number of milliseconds (ms) that should elapse after a *Publisher Variable* was last sent before sending it again, even if the value of the *Publisher Variable* should not have changed in the meantime (further details in EAP send mechanism [▶ 13]).

On Change Only

The On Change Only property cannot be used if an EL66xx is in use.

Variable Id
The *Variable Id (=ProcessData ID)* is the identification number of the *Publisher Variable.* It can be changed online with the help of a PLC program if the option *Online Changeable* is marked.

DataType Version
A version number can be specified here. The same version number must be configured on the subscriber side, if both version numbers are checked for equality when the variable is received (this comparison takes place by default). The version number is used to ensure that the data type of the publisher variable matches the corresponding subscriber variable. If the data type is only changed on the publisher or subscriber side, the version number must be increased to prevent the variable from being received and its values then being interpreted with the wrong data type.

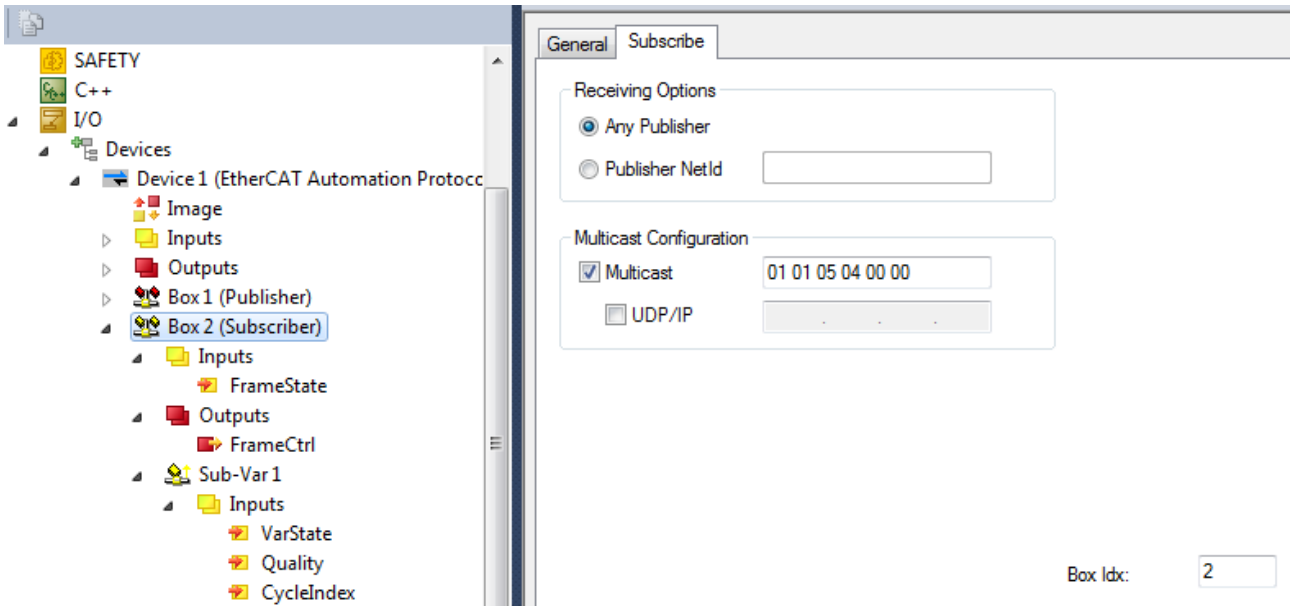
Data Exchange
The rhythm with which the *Publisher Variable* is sent can be changed with the help of the *Data Exchange* property (further details in EAP send mechanism [▶ 13]).

Data Exchange

The *Data Exchange* property cannot be used if an EL66xx is in use.

5.4 Subscriber Box

The properties that generally relate to the reception of an EAP telegram are defined on the configuration page of a *Subscriber* (see next illustration). These properties include:



Receiving Options
With the help of the *Receiving Options* selection you can configure whether all incoming EAP telegrams – i.e. irrespective of the sender – should be received or whether only EAP telegrams from a certain sender should be received. In the latter case the *AMS NetID* of the desired sender should be entered in the input field behind *Publisher NetId*.

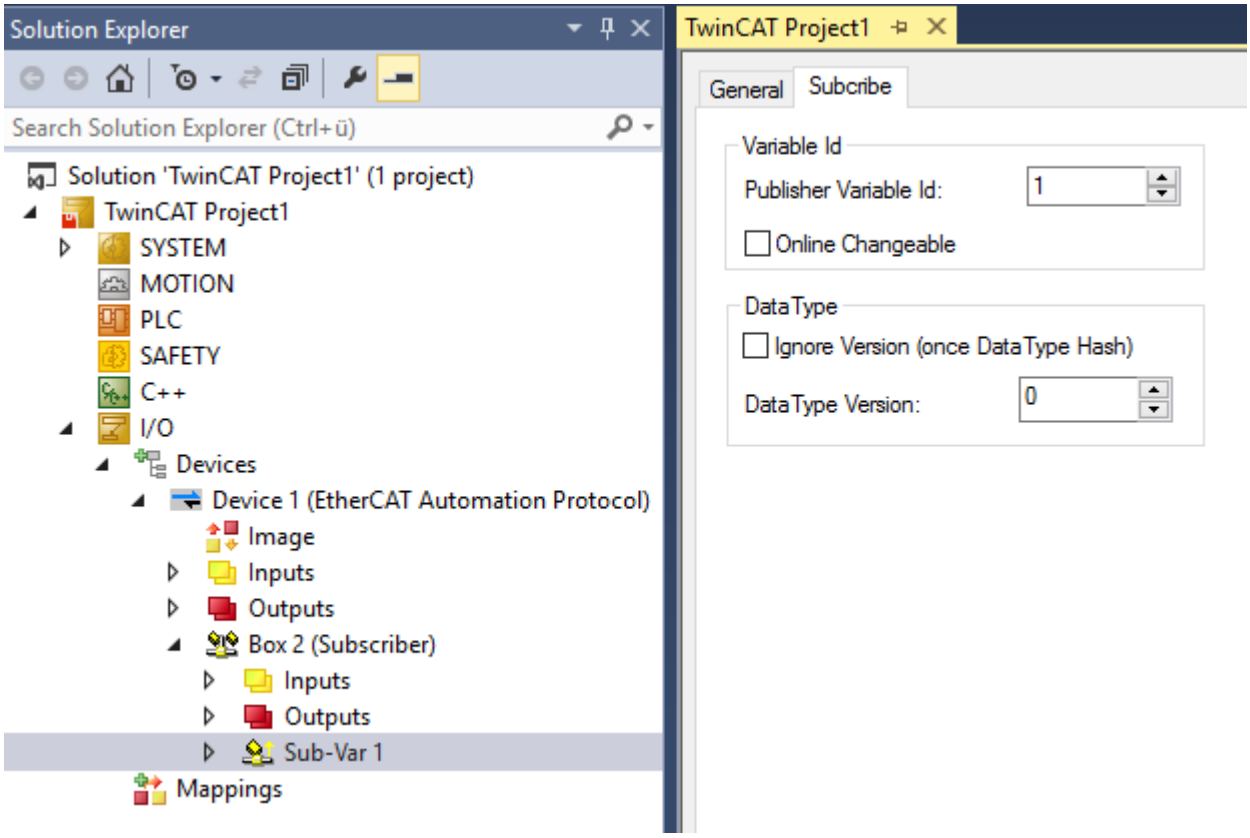
Multicast Configuration
With the help of the *Multicast Configuration* option you can specify whether the *Multicast* address entered should be reported to the configured network interface card. The *Multicast* address parameterized here must be identical to the *Multicast* address configured for the sender if the latter's EAP telegrams are to be received (see *Multicast* in chapter Publisher Box [▶ 44]).

Standard Multicast MAC

The Beckhoff standard *Multicast* address 01:01:05:04:00:00 is always reported to the network interface card for a TwinCAT EAP device, even if the *Multicast* option is not selected.

5.5 Subscriber Variable

The properties that relate especially to a *Subscriber Variable* are defined on the configuration page of the *Subscriber Variable* (see next illustration). These properties include:



Variable Id
The *Variable Id (=ProcessData ID)* is the identification number of the *Subscriber Variable*. It can be changed online with the help of a PLC program if the option *Online Changeable* is marked.

DataType
Each *Subscriber Variable* has a version number (see version in chapter EAP telegram structure [▶ 17]). The version number can be configured in “DataType Version”. It is verified before a *Subscriber Variable* is received (cf. network protocols section in Communication methods [▶ 10]). The incoming *ProcessData* is discarded if the version numbers don't match. This verification is omitted if the option *Ignore Version (once DataType Hash)* is enabled.

Configuration settings for successful data exchange

In order to effect the exchange of data from a *Publisher Variable* to a *Subscriber Variable*, the configurations of the control computers involved must match each other. The section Network Protocol of the chapter Communication methods [▶ 10] contains a description of the sequence of the reception of an EAP telegram or a *Publisher Variable.* In connection with the following aspects, it becomes clear how a data exchange is to be guaranteed:

- The destination address of the *Publisher* must be selected so that the EAP telegram reaches the addressee. Telegrams with *Broadcast* or *Multicast* addressing reach every network device. The exact destination address of the addressee must be configured for a *Unicast* telegram.
- As soon as the *Any Publisher* option has been selected in the *Receiving Options* of the receiver (*Subscriber*), each incoming EAP telegram is received and processed further, irrespective of its sender. An exception only occurs if
  - a *Multicast* address is configured on the sender side that differs from the TwinCAT EAP *Multicast MAC* (01:01:04:05:00:00) and
  - the destination address on the sender side (*Publisher*) does not match the configured *Multicast MAC* address on the *Subscriber* side.

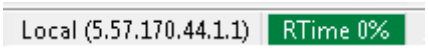
- If a *Publisher NetId* is specified in the *Receiving Options* of the receiver (*Subscriber*), then only EAP telegrams coming from the specified sender (*Publisher*) are received and processed further.
- The *ID* of the *Publisher Variable* and that of the *Subscriber Variable* must be identical and unique in the network. If the *ID* of a sent *Publisher Variable* does not match the *ID* of a *Subscriber Variable*, the variable is discarded when it is received.
- The version (the hash) of the *Publisher Variable* and the associated *Subscriber Variable* must match. A sent *PublisherVariable* whose *Version* does not match the *Version* of a *Subscriber Variable* is discarded during reception unless the *Ignore Data Type Hash* option is activated in the receiver (*Subscriber*).
- The raw data length of a *Publisher Variable* must match the expected raw data length of the *Subscriber Variable*. The *Publisher Variable* will otherwise be discarded during the reception.

5.6 EAP between TwinCAT 2 and 3

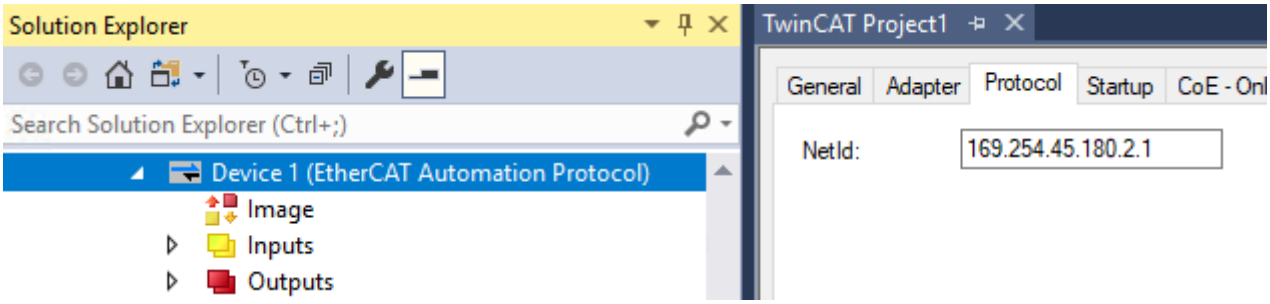
The EtherCAT Automation Protocol is compatible with the conventional network variables (*NWV*) from TwinCAT 2. However, there are some extensions to the network variables with EAP under TwinCAT 3 that require increased attention if communication between TwinCAT 2 *NWV* and TwinCAT 3 EAP is to be established.

Observance of the Publisher NetID

If TwinCAT2 uses a NWV Publisher, the AmsNetID of the overall system is used as AmsNetID. This can be entered accordingly at the receiver.



In contrast, TwinCAT 3 uses for EAP the AmsNetID of the EAP device as Publisher.



"Ignore Data Type Hash" option

A further point that needs to be borne in mind results from the extension of the data type system between the TwinCAT versions 2 and 3:

When using a complex (i.e. non-native) data type, TwinCAT 2 calculates a 16-bit hash value that unambiguously identifies this data type. This data type hash is always 0 for native data types. The data type hash is sent along as a version number when sending a Publisher Variable. This version number is then compared on the receiver side with the data hash type of the configured Subscriber Variable. If the comparison reports correlation, the data are accepted by the subscriber.

In TwinCAT 3 each data type is assigned a Global Unique Identifier (GUID). This has a data length of 128 bits. Accordingly, the value 0 is always used as the version number when configuring a Publisher or Subscriber Variable.

Due to this difference between TwinCAT 2 and 3, you must proceed as follows when using complex data types:

✓ Let's assume that variables of a complex, non-native data type are to be sent between TwinCAT 2 and TwinCAT 3.

1. In this case the "Ignore Data Type Hash" must be activated for the Subscriber Variable.

⇨ This setting suppresses the comparison operation on the receiver side and the data are accepted by the subscriber without a comparison of the versions.

Creating a Subscriber Variable by "Browse for Computer" or "Browse for File".

Due to the change of the file format of a TwinCAT 2 project to a TwinCAT 3 project, the two variants "Browse for Computer" or "Browse for File" for creating a Subscriber Variable to match an existing Publisher Variable are currently only downwardly compatible. This means that it is currently only possible to use these two variants from a TwinCAT 3 system in order to have a Subscriber Variable automatically created from a TwinCAT 2 project on the basis of a Publisher Variable. In the reverse case a Subscriber Variable must be manually created (variant "Create new variable").

6 The CANopen object dictionary

The CiA organization (CAN in Automation) pursues among other things the goal of creating order and exchangeability between devices of the same type by the standardization of device descriptions. For this purpose so-called *CANopen* profiles are defined, which conclusively describe the changeable and unchangeable parameters of a device. Such a parameter encompasses at least the following characteristics:

- An **index number** – for the unambiguous identification of all parameters.
  The index number is divided into a main index and a subindex in order to mark and arrange associated parameters. The subindex is separated by a colon ":".
  This achieves an arrangement in two levels (logical segments). The main index is always used hexadecimally in the value range 0…65535 (0x0…0xFFFF). The subindex is generally used decimally in the value range 0…255 (0x0…0xFF).
- An **official name** - in the form of an understandable, self-descriptive text
- The **access possibility** – e.g. whether the parameter can only be read or also written
- A **data type** – depending on the parameter this can be of the type Text (string), Number (integer, real), Bool or Byte Field.

The assignment of the index numbers to the parameters is defined in a *CANopen* profile. In this way all parameters are organized hierarchically as in a table. This table then contains all of the device-specific parameters. It is called the *CANopen Object Dictionary* (*OD*).

All the parameters of the TwinCAT EAP device are similarly organized with the help of an object dictionary. In terms of the concept its structure is identical to that of the *CANopen OD*. The profile for the *OD* of an EAP device was specified by the EtherCAT Technology Group (ETG) in the specification for the EtherCAT Automation Protocol (ETG 1005, see webpage www.ethercat.org).
This profile is identified by the profile number 5002. It defines the profile type (main profile) and is saved in the Low Word (bits 0-15) of the *OD* parameter *Device Type*. The High-Word (bits 16-31) contains the number 1000. It defines the module profile (subprofile). This produces a value of 0x03e8138a (65541002_{dec}) for the *Device Type* parameter, which is also saved under the *Product Code* in the *Identity Object* (index 0x1018:02).

Example of an object in the OD:
The profile of a TwinCAT EAP device is unambiguously identified on the basis of four parameters. These are summarized in a logical segment called *Identity*, which has the main index 4120 (0x1018). The *Vendor ID* parameter has the index number 4120:01 (0x1018:01) and the entered value *2* as the identifier of a Beckhoff device.
The logical segment *Identity* is also designated as object and is represented from the point of view of the user as follows:

1018:0	Identity		> 4 <
1018:01	Vendor ID	RO	0x00000002 (2)
1018:02	Product Code	RO	0x03E8138A (65541002)
1018:03	Revision Number	RO	0x00030000 (196608)
1018:04	Serial Number	RO	0x00000000 (0)

All the parameters of the *Identity* object have the property *RO* (read only), because the parameters should not be changed by the user.

Quite different properties can be described with the aid of the parameters of an *Object Dictionary* . Examples of such parameters are vendor identifier, version number, process data settings, device name, calibration values, etc. The contents of the *OD* are required for the commissioning as well as the diagnosis of the EAP device and can be very extensive.

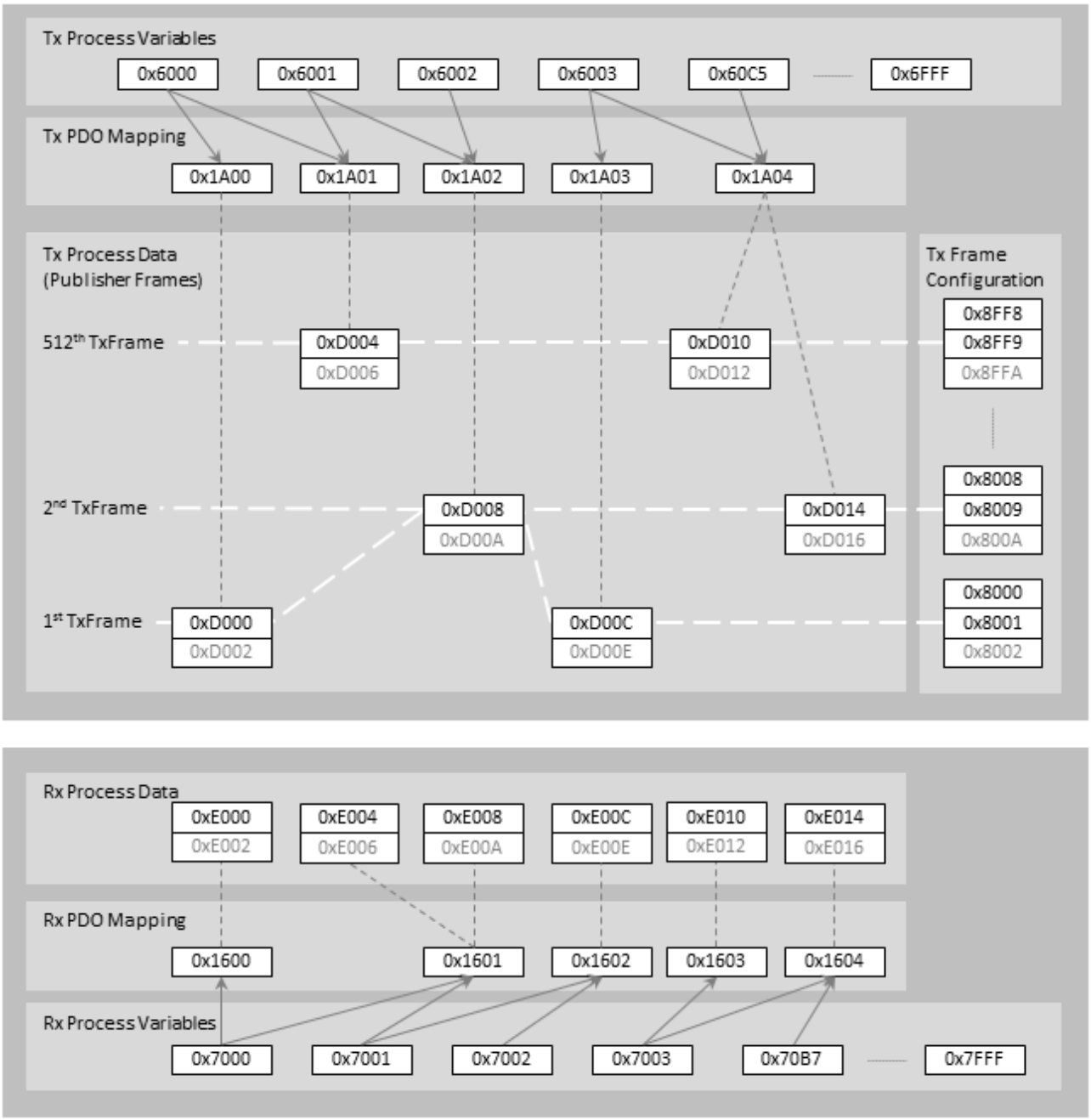
6.1 The EAP Object Dictionary (subprofile 1000)

The EAP *Object Dictionary* is divided into standard and profile-specific objects. Standard objects have the same meaning for all modules. The profile-specific objects have the same meaning for all modules that support the profile type 5002. Beyond that, objects can be static or dynamic. A static object exists as long as the instance of an EAP device itself. A dynamic object can be generated and also deleted again during the runtime of the EAP device.

The division of the object dictionary

The *Object Dictionary* of the EAP device is divided into the following ranges:

- **Index 0x1000** – **0x1FFF**: Range that describes the communication profile.
  General information on the identity of the device such as name, vendor, serial number, etc. are saved in the range 0x1000 – 0x1018.
  Furthermore, *PDO Mapping* objects (*PDO* = *ProcessDataObject*) are defined in the ranges 0x1600 – 0x17FF and 0x1A00 – 0x1BFF. A *PDO Mapping* defines which contents of other objects of the *OD* are summarized to form a *PDO*. A *PDO* then describes the contents of the user data, which is cyclically transmitted in real-time.
- **Index 0x6000** – **0x9FFF**: Range that describes functionally relevant parameters.
  The functionally relevant parameters are specified in the ETG standard 1005. The parameters including their structure are defined under the device profile number 5002, module profile 1000. This definition forms the basis for effecting an exchange of data via the EtherCAT Automation Protocol. The following section deals with the individual object types of the profile as well as their structural relationships.
- **Index 0xF000** – **0xFFFF**: Range that describes the device-specific properties.
  In this range there are objects with whose help diagnostic and control functions can be carried out with the TwinCAT EAP device.



The object types of the standardized profile range and their structure

In the following, the dynamic objects are listed and their relationship with one another is explained. The illustration above shows the relationships:

Objects for parameterizing a *Subscriber*:

- *RxVariable* [0x7000+n … 0x7FFF]:
  An *RxVariable* defines a variable of any type that can be linked with a corresponding input variable of a control application (e.g. PLC).
- RxProcessDataObject (*RxPDO*) [0x1600+n … 0x17FF]:
  An *RxPDO* defines an ordered quantity of *RxVariables* that represent an item of process data as a unit.
- RxProcessData (*RxPD*) [0xE000+4*n … 0xEFFC]:
  An *RxPD* defines the properties for the reception of a *PDO* (see Subscriber Box [▶ 48] and Subscriber Variable [▶ 49]). The *RxPD* thus represents the main reception unit of the EAP communication.
- RxProcessDataInfo [0xE002+4*n … 0xEFFE]:
  An *RxPDInfo* object expands the *RxPD* object by individual properties that are not found in the EAP specification and especially belong to a TwinCAT EAP device.

Objects for parameterizing a Publisher:

- TxVariable [0x6000+n … 0x6FFF]:
  A *TxVariable* object defines a variable of any type that can be linked with a corresponding output variable of a control application (e.g. PLC).
- TxProcessDataObject (*TxPDO*) [0x1A00+n … 0x1BFF]:
  A *TxPDO* defines an ordered quantity of *TxVariables* that represent an item of process data as a unit.
- TxProcessData (*TxPD*) [0xD000+4*n … 0xDFFC]:
  A *TxPD* object defines the properties for transmitting a *PDO* (see Publisher Variable [▶ 47]). The *TxPD* thus represents the main transmission unit of the EAP communication.
- TxProcessDataInfo [0xD002+4*n … 0xDFFE]:
  A *TxPDInfo* object expands the *TxPD* object by individual properties that are not found in the EAP specification and especially belong to a TwinCAT EAP device.
- TxFrame [0x8000+n*8 ... 0x8FF8]:
  A *TxFrame* object defines the transport properties with which one or more *TxPDs* are transmitted within the network (see Publisher Box [▶ 44]).
- TxPD Assignment [0x8001+n*8 … 0x8FF9]:
  A *TxPDAssignment* object is assigned to each *TxFrame* object. The *TxPDAssignment* object has the index one higher than that of the *TxFrame* object. The assignment object specifies which *TxPDs* are sent together in the corresponding *TxFrame.*
- TxFrameInfo [0x8002+n*8 … 0x8FFA]:
  A *TxFrameInfo* object expands the *TxFrame* object by individual properties that are not found in the EAP specification and especially belong to a TwinCAT EAP device.

The standard objects (0x1000-0x1FFF)

Static objects

Index 1000 Device Type

Index	Name	Meaning	Data type	Flags	Default
1000:0	Device Type	EAP device type: The Lo-Word contains the CoE profile used (5002). The Hi-Word contains the CoE profile used (1000).	UINT32	RO	0x03E8138A (65541002_{dec})

Index 1008 Device Name

Index	Name	Meaning	Data type	Flags	Default
1008:0	Device Name	Name of the EAP device	STRING[256]	RO	EtherCAT Automation Protocol

Index 100A Software Version

Index	Name	Meaning	Data type	Flags	Default
100A:0	Software version	Software version of the EAP device	UINT32	RO	0x00000000 (0_{dec})

Index 1018 Identity

Index	Name	Meaning	Data type	Flags	Default
1018:0	Identity	Information for the identification of the EAP device	UINT8	RO	0x04 (4_{dec})
1018:01	Vendor ID	Vendor ID of the EAP device	UINT32	RO	0x00000002 (2_{dec})
1018:02	Product Code	Product code of the EAP device	UINT32	RO	0x03E8138A (65541002_{dec})
1018:03	Product Revision	Revision number of the EAP device	UINT32	RO	0x00030000 (196608_{dec})
1018:04	Serial Number	Serial number of the EAP device. 0 means not used	UINT32	RO	0x0 (0_{dec})

Dynamic objects

Index 1600-17FF RxPDO Mappings

Index	Name	Meaning	Data type	Flags	Default
1600+n:0	Number of used Elements	Number of entries in the RxPDO mapping object	UINT8	RW	#(Subindices)
1600+n:01-255	RxVariable m	Bit 0-7: bit length of the object entered (in the case of a gap in the PDO, corresponds to the bit length of the gap) Bit 8-15: subindex of the object entered (0 in case of a gap in the PDO) Bit 16-31: index of the object entered (0 in case of a gap in the PDO)	UINT32	RW	-

Index 1A00-1BFF TxPDO Mappings

Index	Name	Meaning	Data type	Flags	Default
1A00+n:0	Number of used Elements	Number of entries in the TxPDO mapping object	UINT8	RW	#(Subindices)
1A00+n:01-255	TxVariable m	Bit 0-7: bit length of the object entered (in the case of a gap in the PDO, corresponds to the bit length of the gap) Bit 8-15: subindex of the object entered (0 in case of a gap in the PDO) Bit 16-31: index of the object entered (0 in case of a gap in the PDO)	UINT32	RW	-

Profile-specific objects (0x6000-0xFFFF)

Static objects

Index F100 EAP Status Info

Index	Name	Meaning	Data type	Flags	Default
F100:0	EAP Status	Status information for the EAP device	UINT8	RO	0x02 (2_{dec})
F100:01	Status word	The low byte codes the current state of the EAP device: 0 = Invalid 1 = Init 2 = PreOperational 4 = SafeOperational 8 = Operational The high byte codes whether an error has occurred: 0 = no error 1 = error	UINT16	RO	0x0008 (8_{dec})
F100:02	Status Error Code	An error number that identifies the error that has occurred. 0 means that no error has been identified.	UINT32	RO	0x03E8138A (65541002_{dec})

Index F200 EAP Control Info

Index	Name	Meaning	Data type	Flags	Default
F200:0	EAP Control	Parameter for checking the state of the EAP device	UINT8	RO	0x01 (2_{dec})
F200:01	Control Word	Codes the request to place the EAP device in a desired state: 1 = Init 2 = PreOperational 4 = SafeOperational 8 = Operational	UINT16	RO	0x0008 (8_{dec})

Index F020-F022 Frame List

Index	Name	Meaning	Data type	Flags	Default
F020+n:0	Number of used Elements	Number of configured TxFrames	UINT8	RW	#(Subindices)
F020+n:01-254	Box 1 (Publisher)	Value 0x0000 0000 = first TxFrame object (index 8000) doesn't exist Value 0x0000 03E8 (= 1000): first TxFrame object (index 8000) exists Other values are not permissible. This object can be used to generate/ delete TxFrames	UINT32	RW	0x000003E8 (1000_{dec})

Index F800 EAP Info

Index	Name	Meaning	Data type	Flags	Default
F800:0	Number of used Elements	Number of entries in the EAP Info object	UINT8	RW	0x08 (8_{dec})
F800:01	Available Tx Var	Indicates the maximum number of configured TxVariable objects (0x6nnn).	UINT16	RW	-
F800:02	Available Rx Var	Indicates the maximum number of configured RxVariable objects (0x7nnn).	UINT16	RW	-
F800:03	Available Tx Process Data	Indicates the maximum number of configured Transmit ProcessData objects (0xDnnn).	UINT16	RW	-
F800:04	Available Rx Process Data	Indicates the maximum number of configured RxProcessData objects (0xEnnn).	UINT16	RW	-
F800:05	Available Tx PDOs	Indicates the maximum number of configured TxPDO objects (0x1Ann).	UINT16	RW	-
F800:06	Available Rx PDOs	Indicates the maximum number of configured RxPDO objects (0x16nn).	UINT16	RW	-
F800:07	Available Tx Frames	Indicates the maximum number of configured TxFrame objects (0x8nnn).	UINT16	RO	-
F800:08	Device Cycle Time	Indicates the cycle time with which the EAP device is operated. ProcessData Cycle times (e.g. 0xDnnn:07) can only assume whole-number multiples of this value.	UINT32	RO	-

Index F801 Bitmap

Index	Name	Meaning	Data type	Flags	Default
F801:0	Number of used Elements	Number of entries in the Bitmap object	UINT8	RW	0x06 (6_{dec})
F801:01	Index-Bitmap Tx Var	Bit-coded mapping of existing TxVariable objects. If bit n is set, then index 0x 6000 + n exists.	OCTETE-STRING [512]	RW	-
F801:02	Index-Bitmap Rx Var	Bit-coded mapping of existing RxVariable objects. If bit n is set, then index 0x 7000 + n exists.	OCTETE-STRING [512]	RW	-
F801:03	Index-Bitmap Tx Process Data	Bit-coded mapping of existing TxProcessData objects. If bit n is set, then index 0x D000 + 4*n exists.	OCTETE-STRING [128]	RW	-
F801:04	Index-Bitmap RxProcess Data	Bit-coded mapping of existing RxProcessData objects. If bit n is set, then index 0x E000 + 4*n exists.	OCTETE-STRING [128]	RW	-
F801:05	Index-Bitmap Tx PDOs	Bit-coded mapping of existing TxPDO objects. If bit n is set, then index 0x1A00 + n exists.	OCTETE-STRING [64]	RW	-
F801:06	Index-Bitmap Rx PDOs	Bit-coded mapping of existing RxPDO objects. If bit n is set, then index 0x1600 + n exists.	OCTETE-STRING [64]	RW	-

Index F920 AoE Settings

Index	Name	Meaning	Data type	Flags	Default
F920:0	Number of used Elements	Number of entries in the AoE Settings object	UINT8	RW	0x05 (5_{dec})
F920:01	Local AoE NetID	Local AoE NetID of the EAP device	OCTET-STRING [6]	RW	-
F920:02	Router NetID	AoE NetID of the associated AoE router	OCTET-STRING [6]	RO	-
F920:03	Local MAC Address	Local MAC address of the network card used by this EAP device.	OCTET-STRING [6]	RO	-
F920:04	Local IP Address	Local IP address of the corresponding network card used by this EAP device.	UINT32	RW	-
F920:05	Local Port Name	Name under which the EAP device together with its AoE port is registered with the TwinCAT ADS router.	STRING [31]	RW	EtherCAT Automation Protocol

Dynamic objects

Index 6000-6FFF TxVariables

Index	Name	Meaning	Data type	Flags	Default
6000+n:0	Number of used Elements	Number of entries in the TxVariable object	UINT8	RW	0x22 (34_{dec})
6000+n:01	Size	Length of data (subindex 2) in bits	UINT16	RW	-
6000+n:02	Data	The current data of the variable	OCTET-STRING [Size/8]	RO	-
6000+n:03	Name	Name of the variable	STRING [256]	RW	VarData
6000+n:04	Type	Data type of the object as a GUID	GUID	RW	-
6000+n:05	Reserved	-	UINT32	RW	-
6000+n:29	Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task - PLC-Projectname.MAIN.iCounter)	STRING [256]	RW	-
6000+n:30	AoE Address	Octet 7..2: AoE NetID Octet 1..0: AoE Port Of the object dictionary that contains the current process variable	OCTET-STRING [8]	RW	-
6000+n:32	Image Config	Coding indicating which input/output variables of the process image belong to this object	UINT32	RO	-
6000+n:33	Data Offset	Byte offset within the output process image	UINT32	RO	-
6000+n:34	Reserved	-	UINT32	RO	-

Index 7000-7FFF Rx Variables

Index	Name	Meaning	Data type	Flags	Default
7000+n:0	Number of used Elements	Number of entries in the RxVariable object	UINT8	RW	0x22 (34_{dec})
7000+n:01	Size	Length of data (subindex 2) in bits	UINT16	RW	-
7000+n:02	Data	The current data of the variable	OCTET-STRING [Size/8]	RW	-
7000+n:03	Name	Name of the variable	STRING [256]	RW	VarData
7000+n:04	Type	Data type of the object as a GUID	GUID	RW	-
7000+n:05	Reserved		UINT32	RW	-
7000+n:29	Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task - PLC-Projectname.MAIN.iCounter)	STRING [256]	RW	-
7000+n:30	AoE Address	Octet 7..2: AoE NetID Octet 1..0: AoE Port Of the object dictionary that contains the current process variable	OCTET-STRING [8]	RW	-
7000+n:32	Image Config	Coding indicating which input/output variables of the process image belong to this object	UINT32	RO	-
7000+n:33	Reserved	-	UINT32	RO	-
7000+n:34	Data Offset	Byte offset within the output process image	UINT32	RO	-

Index 8000-8FF8 TxFrame

Index	Name	Meaning	Data type	Flags	Default
8000+n*8:0	Number of used Elements	Number of entries in the TxFrame object	UINT8	RW	0x32 (50_{dec})
8000+n*8:03	Name	Name of the frame	STRING [256]	RW	-
8000+n*8:04	Device type	Subprofile type (identical to the corresponding entry in the object 0xF020-0xF022)	UINT32	RO	0x03E8 (1000_{dec})
8000+n*8:05	Destination Vendor ID	For peer-to-peer communication; 0 = not used Polled Connection: Vendor ID of the communication partner Pushed Connection: not used	UINT32	RW	-
8000+n*8:06	Destination Product Code	For peer-to-peer communication; 0 = not used	UINT32	RW	-
8000+n*8:07	Destination Revision Number	For peer-to-peer communication; 0 = not used	UINT32	RW	-
8000+n*8:08	Destination Serial Number	For peer-to-peer communication; 0 = not used	UINT32	RW	-
8000+n*8:30	Target AMS NetID	AoE NetID (Subscriber Net ID) If the value is not 0, the destination addresses SI 32 and SI 33 must have the value 0.	OCTET-STRING [6]	RW	-
8000+n*8:31	Gateway IP Address	The standard gateway IP address must then be set if SI 33 does not have the value 0.	UINT32	RW	-
8000+n*8:32	Target MAC Address	MAC address If the value is not 0, the destination addresses SI 30 and SI 33 must have the value 0. The MAC address can be a Unicast, Multicast or Broadcast address.	OCTET-STRING [6]	RW	01 01 05 04 00 00
8000+n*8:33	Target IP Address	IP Address If the value is not 0, the destination addresses SI 30 and SI 32 must have the value 0. The IP address can be a Unicast, Multicast or Broadcast address.	UINT32	RW	-
8000+n*8:34	VLAN Info	The VLAN Info is made up of the following fields: Bit 0-15: VLan Type (81 00) Bit 16-18: Priority Bit 19: Reserved Bit 20-31: VLan ID No VLAN header is used if the value is 0	UINT32	RW	0x00000000 (0_{dec})

Index	Name	Meaning	Data type	Flags	Default
8000+n*8:35	Subscriber Monitoring	If the value is 1, an ARP request is sent regularly to the configured destination address in order to ensure that the addressee is still replying. Sending of the TxFrame is ceased if this is not the case. The Subscriber Monitoring can only be used with a Unicast communication.	UINT8	RW	0x00 (0_{dec})
8000+n*8:36	Target Changeable	If Target Changeable has the value 0, no variable is shown for the destination address in the process image. Otherwise the following mapping applies when showing the destination address: 1 : Target MAC address 2 : Target AMS NetID 3 : Target IP address	UINT8	RO	0x00 (0_{dec})
8000+n*8:37	Monitoring Retry Cycles	obsolete	UINT32	RO	-
8000+n*8:38	Monitoring Retry Cycle Time	Waiting time in µs after which a new ARP request is sent if SI 35 = 0x01.	UINT32	RW	0xF4240 (1000000_{dec})
8000+n*8:39	Frame Control	Bit 0 = 1: the sending of the TxFrame is ceased Bit 1 = 1: destination MAC address is deleted from the ARP cache	UINT16	RW	0x0000 (0_{dec})
8000+n*8:40	Frame State	Bit 0 = 1: the TxFrame wasn't sent Bit 1 = 1: error (the frame is too large) Bit 2 = 1: the subscriber is no longer answering (only if SI 35 = 0x01)	UINT16	RO	0x0000 (0_{dec})
8000+n*8:48	Control Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-
8000+n*8:49	State Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-
8000+n*8:50	Target Address Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task) If a symbol name is set, the configured destination address is shown in the process image and SI 36 is set accordingly.	STRING [256]	RW	-

Index 8001-8FF9 TxProcessData Assignment Objects

Index	Name	Meaning	Data type	Flags	Default
8001+n*8:0	Number of used Elements	Number of entries in the TxPD assignment object	UINT8	RW	#(Subindices)
8001+n*8 :01-255	Entry n	1. -255. TxProcessData of the TxFrame	UINT16	RW	-

Index 8002-8FFA TxFrame Info

Index	Name	Meaning	Data type	Flags	Default
8002+n*8:0	Number of used Elements	Number of entries in the TxFrameInfo object	UINT8	RW	0x21 (33_{dec})
8002+n*8:01	Image Config	Coding indicating which input/output variables of the process image belong to this object Lo-Word = Input process image Bit 0 = 1: State Hi-Word = output Process image Bit 0 = 1: Control Bit 1 = 1: Target MAC Address Bit 2 = 1: Target AMS NetID Bit 3 = 1: Target IP Address	UINT32	RO	0x00010001 (65537_{dec})
8002+n*8:02	Control Offset	Byte offset within the output process image	UINT32	RO	-
8002+n*8:03	State Offset	Byte offset within the input process image	UINT32	RO	-
8002+n*8:04	NetID Offset	Byte offset within the output process image	UINT32	RO	-
8002+n*8:32	IoDivMod	The divider/modulo value defines the waiting time in cycles until the next TxFrame is sent. Bit 0-7 (Divider): number of cycles to be waited Bit 8-15 (Modulo): specifies the start cycle from which counting starts	UINT16	RW	0x0000 (0_{dec})
8002+n*8:33	CoE Index	For future purposes	UINT16	RW	-

Index D000-DFFC TxProcessData

Index	Name	Meaning	Data type	Flags	Default
D000+n*4:0	Number of used Elements	Number of entries in the TxPD object	UINT8	RW	0x22 (34_{dec})
D000+n*4:01	Name	Name of the frame	STRING [256]	RW	-
D000+n*4:02	PDO Number	The PDO number defines the object index of the assigned TxPDO	UINT16	RW	-
D000+n*4:03	Process Data ID	The PD ID defines a value in the range 0…65535 that must clearly be within the communication network. The ID is part of the Process Data Frame header.	UINT16	RW	-
D000+n*4:04	Version	The version is a value in the range 0… 65535 and should be consistently incremented as soon as changes are made to this TxPD (e.g. the reference to another TxPDO). The version is part of the Process Data Frame header.	UINT 16	RW	-
D000+n*4:05	CoS On Change Cycles	Obsolete, see subindex 8.	UINT16	RO	-
D000+n*4:06	CoS Inhibit Time	The Inhibit Time specifies the time span in µs during which the TxPD is not sent again, not even if the value of a process variable of the assigned PDO has changed. The transmission of the TxPD is not suppressed if the value is 0. If the value is > 0, then the value of subindex 8 (CoS On Change Timeout) must also be > 0; however, the values of the subindices 7 and 10 must be 0.	UINT32	RW	-
D000+n*4:07	Cycle Time	The Cycle Time defines the time interval in µs at which the TxPD is cyclically transmitted. If the value of Cycle Time is larger than 0, the subindices 6, 8 and 10 must be 0. The TxPD is not transmitted at all if the value is 0.	UINT 32	RW	-
D000+n*4:08	CoS On Change Timeout	On Change Timeout specifies the maximum duration of the time interval in µs during which no TxPD is transmitted, unless the value of a process variable of the assigned PDO changes during that time. If the value is 0, the process variables are not sent in the case of a change of state. If the value is > 0, the values of subindices 7 and 10 must be 0.	UINT32	RW	-

Index	Name	Meaning	Data type	Flags	Default
D000+n*4:10	Poll Request Rx PD	Poll Request RxPD defines the object index of an RxProcessData, which triggers the transmission of this TxPD as soon as the defined RxPD has received a new value. The TxPD then functions as a server in Polled Data Exchange mode. The Polled Data Exchange mode is inactive if the value is 0. If the value is > 0, the values of subindices 6, 7 and 8 must be 0.	UINT16	RW	-
D000+n*4:11	Process Data Control	Bit 0 = 1: deactivate the transmission of the TxPD	UINT16	RW	0x0000 (0_{dec})
D000+n*4:12	Process Data State	Bit 0 = 1: the TxPD was not transmitted	UINT16	RO	-
D000+n*4:32	Control Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-
D000+n*4:33	State Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-
D000+n*4:34	ID Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-

Index D002-DFFE TxProcessData Info

Index	Name	Meaning	Data type	Flags	Default
D002+n*4:0	Number of used Elements	Number of entries in the TxPDInfo object	UINT8	RW	0x20 (32_{dec})
D002+n*4:01	Image Config	Coding indicating which input/output variables of the process image belong to this object Lo-Word = Input process image Bit 0 = 1: State Hi-Word = output Process image Bit 0 = 1: Control Bit 1 = 1: ProcessData ID	UINT32	RO	0x00010001 (65537_{dec})
D002+n*4:02	Control Offset	Byte offset within the output process image	UINT32	RO	-
D002+n*4:03	State Offset	Byte offset within the input process image	UINT32	RO	-
D002+n*4:04	ID Offset	Byte offset within the output process image	UINT32	RO	-
D002+n*4:32	IoDivMod	The divider/modulo value defines the waiting time in cycles until the TxPD is sent again. Bit 0-7 (Divider): number of cycles to be waited Bit 8-15 (Modulo): specifies the start cycle from which counting starts	UINT16	RW	0x0000 (0_{dec})

Index E000-EFFC RxProcessData

Index	Name	Meaning	Data type	Flags	Default
E000+n*4:0	Number of used Elements	Number of entries in the RxPD object	UINT8	RW	0x25 (37_{dec})
E000+n*4:01	Name	Name of the frame	STRING [256]	RW	-
E000+n*4:02	PDO Number	The PDO number defines the object index of the assigned RxPDO	UINT16	RW	-
E000+n*4:03	Process Data ID	The PD ID defines a value in the range 0…65535 that matches the ID of the Process Data received.	UINT16	RW	-
E000+n*4:04	Version	The version is a value in the range 0… 65535 and should be consistently incremented as soon as changes are made to this RxPD (e.g. the reference to another RxPDO).	UINT 16	RW	-
E000+n*4:05	Ignore Version	If this is 0, then the version (hash value) of the Process Data received is checked on the basis of the version from subindex 4. If it is 1, then the version check is deactivated.	UINT8	RW	0x00 (0_{dec})
E000+n*4:06	Publisher NetID	Definition of a Publisher NetID. An EAP telegram is only processed if it was sent from a sender with this NetID. This filter is deactivated if the Publisher NetID has the value 0.	OCTET-STRING [6]	RW	00 00 00 00 00 00
E000+n*4:07	MAC Address	A Multicast MAC address can be defined that uses the NIC (Network Interface Card) as a filter for the reception of Multicast packets. The filter function is deactivated if the value is 0.	OCTET-STRING [6]	RW	01 01 05 04 00 00
E000+n*4:08	IP Address	A Multicast IP address can be defined that uses the NIC (Network Interface Card) as a filter for the reception of Multicast packets. The filter function is deactivated if the value is 0.	UINT32	RW	0x00000000 (0_{dec})
E000+n*4:09	Update Time	The Update Time is used to specify the time interval in µs within which a new ProcessData must be received. This mechanism is deactivated if the value is 0.	UINT32	RW	0x00000000 (0_{dec})
E000+n*4:10	Poll Request TxPD	Poll Request TxPD defines the object index of a TxProcessData, which is sent as a request in order to receive an EAP telegram with the suitable ProcessData. The TxPD then functions as a server in Polled Data Exchange mode. The Polled Data Exchange mode is inactive if the value is 0.	UINT16	RW	0x0000 (0_{dec})
E000+n*4:11	Process Data Control	Bit 0 = 1: The checking of the version number or the hash value is deactivated.	UINT16	RW	0x0000 (0_{dec})

Index	Name	Meaning	Data type	Flags	Default
E000+n*4:12	Process Data State	Bit 0 = 1: A ProcessData with an invalid version number (hash value) was received Bit 1 = 1: A ProcessData with an invalid length was received Bit 2 = 1: The Timeout Poll Response was exceeded	UINT16	RO	-
E000+n*4:13	Process Data Quality	The quality indicates the time in 100 µs since this RxProcessData was last updated (i.e. since data was last received)	UINT16	RO	-
E000+n*4:14	Process Data Cycle Index	On receiving a valid ProcessData, the Cycle Index is assigned the transmitted Cycle Index from the EAP telegram (see Process Data Frame Header)	UINT16	RO	-
E000+n*4:32	Control Symbol Name	Symbol name of the linked variable from an application (e.g. PLC Task)	STRING [256]	RW	-
E000+n*4:33	State Symbol Name	Symbol name of the linked variable from an application (e.g. PLC Task)	STRING [256]	RW	-
E000+n*4:34	ID Symbol Name	Symbol name of the linked variable from an application (e.g. PLC Task)	STRING [256]	RW	-
E000+n*4:35	Quality Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-
E000+n*4:36	Cycle Index Symbol Name	Symbol name of the linked variable from the application (e.g. PLC-Task)	STRING [256]	RW	-
E000+n*4:37	Timeout Poll Response	Specifies the maximum time span in µs within which the response to the Polled Request must be received. If the Value > 0 and the timeout has expired following the sending of the Polled Request, then bit 2 is set in PD State (subindex 12). If the value is 0, then this monitoring is deactivated.	UINT32	RW	0x00000000 (0_{dec})

Index E002-EFFE RxProcessDataInfo

Index	Name	Meaning	Data type	Flags	Default
E002+n*4:0	Number of used Elements	Number of entries in the RxPD Info object	UINT8	RW	0x06 (6_{dec})
E002+n*4:01	Image Config	Coding indicating which input/output variables of the process image belong to this object Lo-Word = Input process image Bit 0 = 1: State Hi-Word = output Process image Bit 0 = 1: Control / Cycle Index Bit 1 = 1: ProcessData ID	UINT32	RO	0x00010001 (65537_{dec})
E000+n*4:02	Control Offset	Byte offset within the output process image	UINT32	RO	-
E002+n*4:03	State Offset	Byte offset within the input process image	UINT32	RO	-
E002+n*4:04	ID Offset	Byte offset within the output process image	UINT32	RO	-
E002+n*4:05	Quality Offset	Byte offset within the input process image	UINT32	RO	-
E002+n*4:06	Cycle Index Offset	Byte offset within the input process image	UINT32	RO	-

6.2 The TwinCAT ADS interface to the EAP device

The TwinCAT EAP device provides a TwinCAT *ADS/AMS* interface for other communication partners (e.g. virtual field devices or Windows programs) and acts as an *ADS/AMS* server. *ADS* stands for *Automation Device Specification.* It describes a device- and fieldbus-independent interface. *AMS* stands for *Automation MessageSpecification* and enables central and decentral systems to be addressed, such as PCs or bus controllers. *ADS/AMS* was specified by Beckhoff and is supported by the TwinCAT router. Messages that are sent in a network beyond the computer boundaries are transferred via *TCP/IP*.

The *CANopen* communication channel *SDO* (*Service Data Object*) can also be used via this interface. The primary purpose of the *SDOs* is to read and write the parameters of the *CANopen Object Dictionary* (*OD*). The transmission of the *SDOs* takes place as a confirmed data transfer in the form of a point-to-point connection between two communication partners and is embedded in ADS:

Using the *ADS Read* or *ADS Write* commands on the port 0xFFFF and the NetID of the EAP device ("Protocol" tab), the parameters, or the *SDO* description of the *CANopen OD* respectively, are read or written. As listed in the table, the communication channel *CANopen SDO* is embedded in the *ADS* protocol as follows:

CANopen SDO communication	ADS Command	Index group	Index offset	Meaning
SDO Upload	Read	0xF302	Index and subindex of an SDO. Bit 16-31: Index Bit 8: Complete Access Bit 0-7: Subindex Sample: 0x16010001: Index = 0x1601 Complete Access = 0 Subindex = 1 The subindex has no meaning if Complete Access = 1	Data Type: UINT8[n] SDO Upload Request: The object is addressed on the basis of the Index Offset and its contents can be read with the help of an ADS Read.
SDO Download	Write	0xF302	Index and subindex of an SDO. Bit 16-31: Index Bit 8: Complete Access Bit 0-7: Subindex Sample: 0x16010001: Index = 0x1601 Complete Access = 0 Subindex = 1 The subindex has no meaning if Complete Access = 1	Data Type: UINT8[n] SDO Download Request: The object is addressed on the basis of the Index Offset and its contents can be written with the help of an ADS Write.
SDO Information Get Object List	Read	0xF3FC	Bit 16-31: List types	Returns the indices of the objects that belong to the list type specified in the Index Offset. Possible list types are: ALL_OBJECTS = 1 RXPD_OBJECTS = 2 TXPD_OBJECTS = 3 BACKUP_OBJECTS = 4 SETTING_OBJECTS = 5
			Samples: 0x00000000: Returns the number of all objects existing for each list type	Data Type: UINT16[6] If list type = 0000: Element = 0 : number of list types Element > 0 : number of existing objects belonging to the nth list type
			0x00010000: Returns the indices of all objects for the specified list type	Data Type: UINT16[n] If list type > 0000: Element n=0 : number of existing objects belonging to this list type plus one. Element n>0: the nth object index belonging to this list type
SDO Information Get Object Description	Read	0xF3FD	Bit 16-31: Index	Read the SDO description of the complete object with the specified index.
SDO Information Get Entry Description	Read	0xF3FE	Bit 16-31: Index Bit 0-7: Subindex	Read the SDO description of the individual entry with the specified subindex belonging to the object with the specified index.

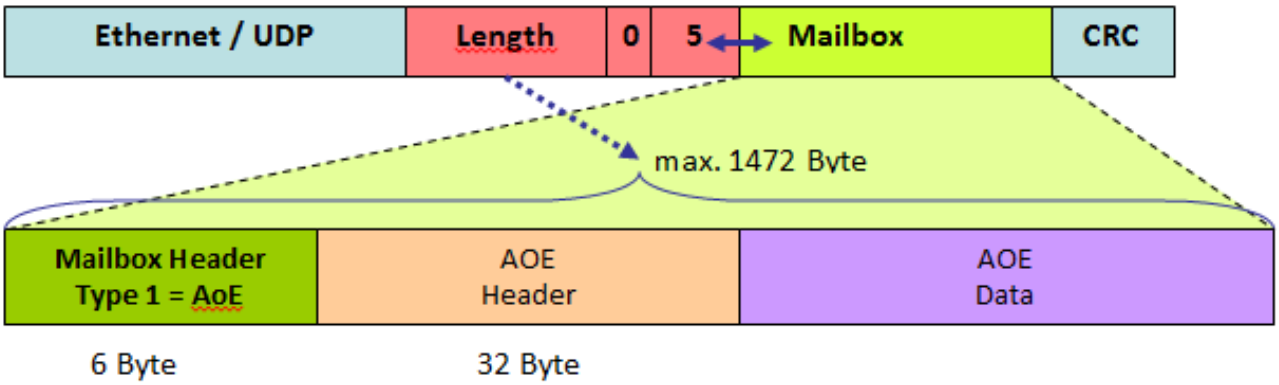
6.3 ADS over EtherCAT (AoE)

The TwinCAT EAP device also supports the *AoE* protocol. The specification of the AoE protocol can be found in the EtherCAT Protocol Enhancements (ETG 1020). The difference between ADS/AMS communication and AoE communication is that, in contrast to ADS/AMS communication, AoE communication requires no TwinCAT router. The AoE protocol is one of the protocols that are classified in TwinCAT under the category Mailbox Communication. An EtherCAT telegram of type 5 (mailbox communication) is used for the communication of AoE. A mailbox telegram can be transmitted from or to the TwinCAT EAP device:

via Ethernet (EtherType = 0x88A4) or

via UDP/IP (UDP Port = 0x88A4)

It is possible to transport (tunnel) various protocols by mailbox communication. The protocol to be tunneled is defined on the basis of the field type in the *Mailbox Header.* The AoE protocol is specified by the value 1 (see *Mailbox Header* in the following illustration). The *Mailbox Header* is directly followed by the *AoE Header* and then by the *AoE* data. Its structure is identical to the *ADS/AMS* protocol (refer also to the TwinCAT *ADS/AMS* specification in the Beckhoff information system). This gives rise to the possibility to also use the *CANopen SDO* communication via the mailbox protocol. Analogous to the *ADS/AMS* protocol, the *CANopen OD* of the TwinCAT EAP device can be accessed as described in the chapter The TwinCAT ADS interface to the EAP device [▶ 67].



Sample:
In order to read the *Vendor ID* of the identity object from the EAP Object Dictionary, the *ADS* command is defined in the *AoE Header* on the basis of the *Command ID*.

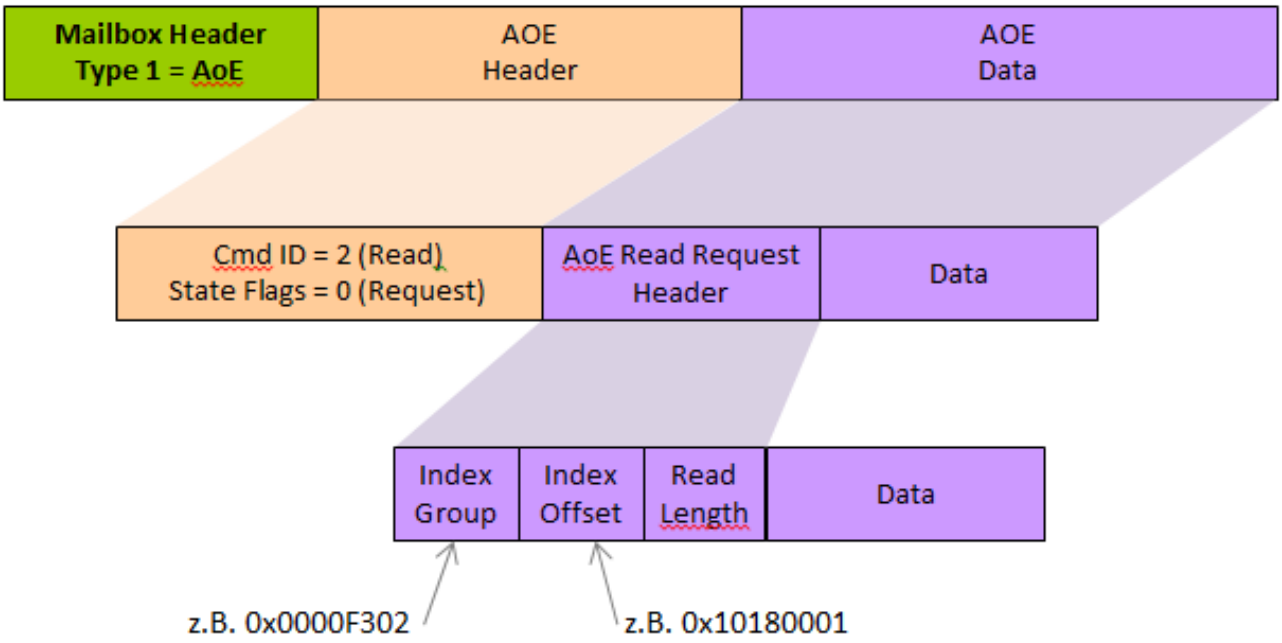
AoE command	Command ID	Description
ADS Read	2	ADS Read command for reading data
ADS Write	3	ADS Write command for writing data

A distinction is made between *Request = 0* and *Response = 1* on the basis of the 0th bit in the *State Flag* of the *AoE Header.* The *AoE Data* field is subsequently populated with the following contents (see following illustration):

At the beginning there is an *ADS Read Request Header* with

- Index Group = 0x0000F302 for reading the contents of the parameter,
- Index Offset = 0x10180001 for addressing the *Vendor ID* parameter and
- Read Length = 4 to inform about the size of the available data buffer.

These are followed by the data buffer for the data to be read.



6.4 Online configuration of the TwinCAT EAP device

The *CANopen OD* offers the great advantage of allowing the virtually unlimited online configuration of a TwinCAT EAP device via a standardized interface. Carrying out an online configuration means that the TwinCAT EAP device is already active and needs to be switched to a safe state before changing the configuration. No process data communication takes place in the safe state.

The configuration is done by setting parameters and takes place

- via TwinCAT:
  the values are then changed directly in the online Object Dictionary.
  This is useful for commissioning of the system. The new value is set in the respective parameter by clicking on the corresponding row of the index to be parameterized and entering an appropriate value in the *Set Value* dialog.
- or from an application via ADS or AoE:
  This is recommended for changes when the system is running or if no TwinCAT or operator is available.

On the one hand, a change can exclusively concern the entries in existing objects of the *OD*. On the other, a change can result in the generation of new objects or the deletion of existing ones.

Changing existing objects

A change that concerns only existing objects serves the adaptation of the current operating behavior (e.g. transmission cycle, destination address, etc.) or to assign purpose-related names to individual objects for a better understanding. The steps that are necessary for such a change can best be illustrated with an example:

Changing the time interval with which a Publisher Variable is transmitted.

Let's assume that the EAP device is configured with a *TxPD* (0xD000) in which an interval of 10000 µs is configured as the *Cycle Time* and this interval is now to be enlarged to 30000 µs. In this case the following steps have to be carried out:

1. Place the EAP device in a safe state.
   Set the object entry 0xF200:01 (*Control Word*) to the value 2 (=PreOperational)
   Check object entry 0xF100:01 to ascertain whether the state is 2 (=PreOperational)
   Carry out this check several times (polling) with a timeout of, for example, 200 ms.
2. Change the time interval of the TxPD to the new value
   Set the object entry *Cycle Time* of the *TxPD* (0xD000:07) to the value 30000

3. Place the EAP device in its operative state again.
   Set the object entry 0xF200:01 (*Control Word*) to the value 4 (=SafeOperational)
   Check object entry 0xF100:01 to ascertain whether the state is 4 (=SafeOperational)
   Carry out this check several times (polling) with a timeout of, for example, 200 ms.
   Set the object entry 0xF200:01 (*Control Word*) to the value 8 (=Operational)
   Check object entry 0xF100:01 to ascertain whether the state is 8 (=Operational)
   Carry out this check several times (polling) with a timeout of, for example, 200 ms.

Generation and deletion of objects

The generation of new objects serves the extension of existing communication connections or the addition of new communication connections. This function makes it possible, for example, to establish a temporary communication connection between two controllers. Only the transmission and reception of the EAP device are briefly interrupted during the configuration phase. The processes of other components and modules of the controller are unaffected by this and can remain active throughout.

Objects are generated or deleted with the help of the objects *Frame List* (index 0xF020 – 0xF022) and *Bitmap* (index 0xF801)
There are a total of 512 entries in the *Frame List* objects – exactly as many as the maximum possible number of *TxFrame* objects (see index 0xF020 in The EAP Object Dictionary (subprofile 1000) [▶ 52]). Each entry is thus assigned precisely one *TxFrame* object. The value 1000 (sub-profile number) is then saved in the corresponding entry for each existing *TxFrame* object. All other entries have the value 0. Let's assume that there are three *TxFrames* with the indices 0x8000, 0x8008 and 0x87F0. The assignment is then to be understood as follows:

No.	Frame List entries	Value	Assigned TxFrame object
1	0xF020:01	1000	0x8000
2	0xF020:02	1000	0x8008
…	…	0	…
254	0xF020:254	0	0x87E8
255	0xF021:01	1000	0x87F0
…	…	0	…
512	0xF022:04	0	0x8FF8

The corresponding *TxFrame* objects are generated as soon as the value of an entry is set to 1000. The object is deleted again when the value is reset to 0.

With the exception of *TxFrames* a separate entry exists for each dynamic object type in the object *Bitmap*. Accordingly, six entries are to be found (excluding the entry 0 – see index 0xF801 in The EAP Object Dictionary (subprofile 1000) [▶ 52]). An entry contains a bit field of a sufficient length to be able to save the maximum number of objects of the corresponding object type in unary notation.

Let's assume that there are three *TxPD* objects with the indices 0xD000, 0xD00C and 0xD014. Then the bit sequence 10010100000000…0 with the total length of 1024 bits is saved in the entry 0xF801:03. The sequence is to be interpreted as follows:

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8		Bit 1024
1	0	0	1	0	1	0	0	…	0
0xD000	0x D004	0x D008	0x D00C	0x D010	0x D014	0x D018	0x D01C	…	0x DFFC

The corresponding object is generated as soon as a bit is set to 1. The object is deleted again on resetting.

NOTE

Deletion of objects configured offline.

It is also possible to delete objects that were configured before commissioning by means of TwinCAT (offline configuration) and as a result are already generated on activating the controller. The following must be considered in this case:

Some objects manage variables (e.g. *CycleIndex*, *Quality*, *Status Word*, *Control Word* etc.) that are located in the process image of the EAP device. TwinCAT accesses the process image by *ADS* when displaying online values. The addresses of the respective variables in the process image that were generated and activated with the help of TwinCAT are calculated after the configuration.

If an object that was configured offline is now deleted during an online configuration, this change remains hidden from TwinCAT. It therefore still accesses the calculated addresses in order to display their contents in the online view. It should be clear at this point that the online values displayed no longer match the configuration generated in TwinCAT.

In this case, therefore, the online view, or the addresses of the process image respectively, are no longer a valid reference source for monitoring or even evaluating the values of the process image.

It is recommended that objects configured offline be deactivated and not deleted.

If objects are deleted or added with the help of TwinCAT, new objects are not directly displayed in the Object Dictionary. Deleted objects remain in the *OD* and no values are displayed for their entries any longer. So that new objects are displayed and deleted objects are no longer displayed, the structure of the *OD* must be read again in TwinCAT. This can be done with the help of the *Advanced Settings* dialog (see fig. *CoEOnline* in chapter The TwinCAT EAP device [▶ 41]).

Activation/deactivation of objects

In case certain *Process Data/PDOs* or complete *Frames* are only to be temporarily removed from the existing EAP communication, it is advisable to deactivate the objects concerned instead of deleting them.

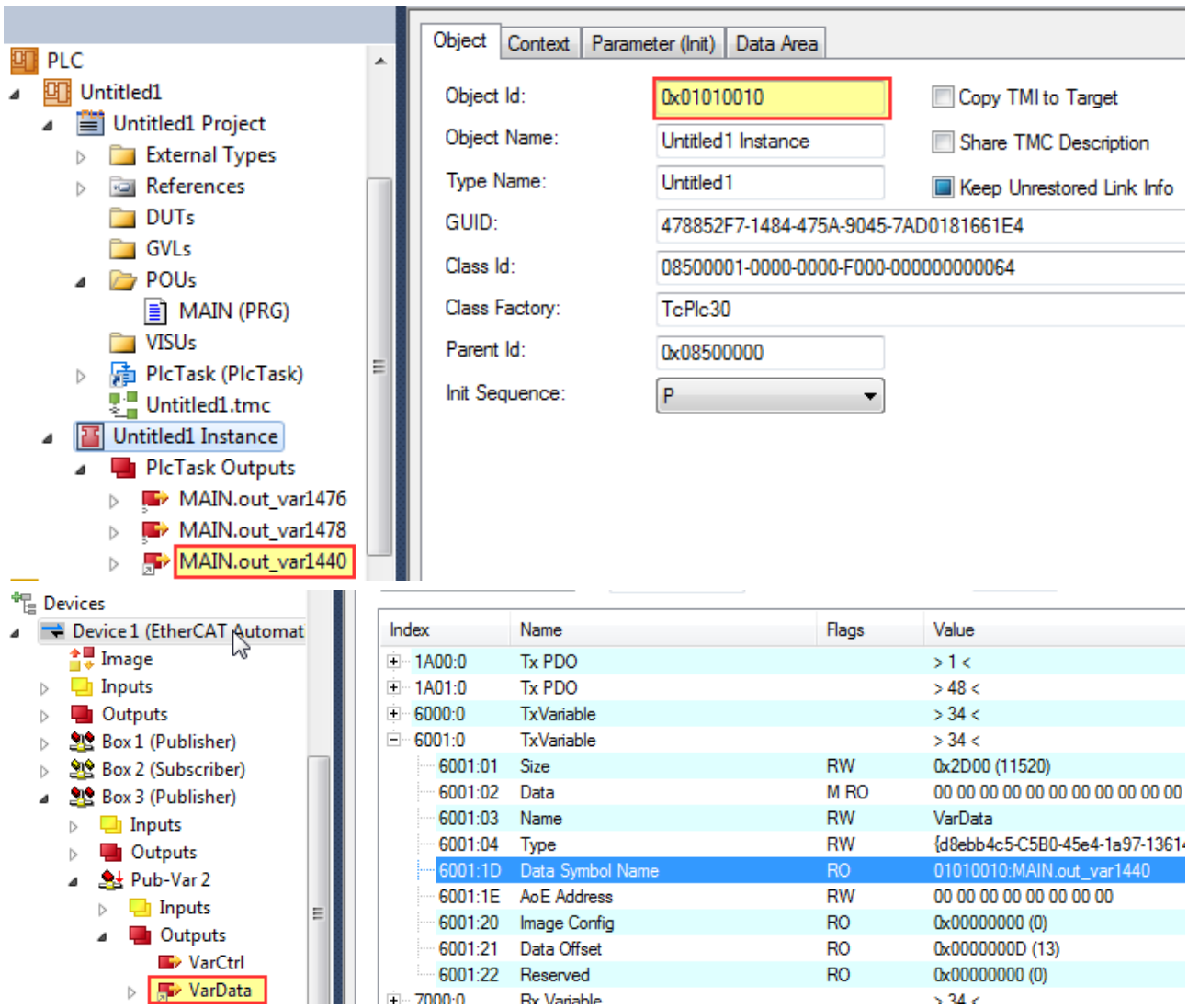
The objects *TxFrame* (0x8000), *TxPD Assignment* (0x8001), *TxPD* (0xD000), *TxPDO* (0x1A00), *RxPD* (0xE000) and *RxPDO* (0x1600) can be deactivated and reactivated by setting their subindex 0 to the value 0. The activation then takes place by setting the original value or any other value greater than 0.

List-based objects

Note with the list-based objects *TxPD Assignment* and *Rx/TxPDOs* that their entries always contain references to other objects. Any change made to subindex 0 will therefore affect which and how many other objects are actually referenced.

Linking of a variable from the EAP process image with a PLC variable.

The linking of variables from the TwinCAT I/O level with variables from a PLC program takes place as standard in TwinCAT. If an *Input/Output Variable* from a PLC program is linked with a *Publisher or Subscriber Variable* of the EAP device, the symbol name of this variable is entered in the corresponding *CANopen* object of the EAP device (see next illustration).



As the previous illustration shows, the symbol name is made up of the *Object ID (OID)* of the PLC instance and the unique name of the PLC program variable, separated by a colon. Using the same syntax a symbol name can also be entered for an EAP object during the online configuration. The PLC variable is then linked accordingly with the *Rx/TxVariable.*

Online configuration of offline links

Links that had already been generated during the offline configuration can neither be changed nor deleted during the online configuration. These symbol names are thus marked with the *Read-Only (RO)* flag.

Reason:
The linking of these variables is defined by TwinCAT with the help of a mapping object when creating a configuration. However, this mapping object cannot be changed during operation.

Access to the CANopen Object Dictionary by Single Access/Complete Access

With the support of the communication channel *CANopen SDO* (see The TwinCAT ADS interface to the EAP device [▶ 67]), the *ADS interface* allows both the values of individual object entries and the data of a complete object to be read or written respectively. Access to the individual object entry (*Single Access*) is explained in the section *Changing existing objects* in the chapter Online configuration of the TwinCAT EAP device [▶ 70].
Complete Access means access to the complete object. In this case the values of all object entries are transmitted in succession in binary in a block. For safety in the case of a write access, the correlation of the block length with the size of the object is checked before the access is enabled. To determine the necessary block length the documentation for the Object Dictionary can be consulted or an offline configuration can be

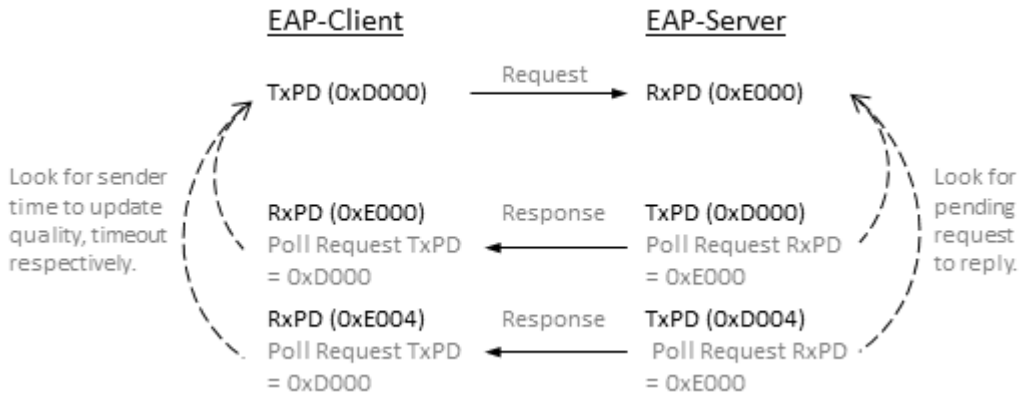
exported to an *EAP Device Configuration* (EDC) file (see section Protocol in chapter The TwinCAT EAP device [▶ 41]). The latter is a file in XML format containing the full Object Dictionary of the current configuration (see chapter The EAP Device Configuration (EDC) File [▶ 76]).

6.5 Configuration of Polled Data Exchange

The Polled Data Exchange mode is configured with the help of the EAP Object Dictionary. To do this the *RxPD* which, as a client, is to receive the response of the server must be configured in one EAP device, while the *TxPD* which, as a server, defines the response to the client's request must be configured in another EAP device.

As shown in the following illustration, a *TxPD* is configured on the client side for the request and is cyclically transmitted as in Pushed Data Exchange mode. For receiving the server response, two *RxPDs* are configured on the client side in this sample. With both *RxPDs* the index of the *TxPD* that serves as the request is entered in the *Poll Request TxPD* property . On the basis of the transmission time of the request, an RxPD can determine a quality value on whose basis the time difference between request and response can be read.

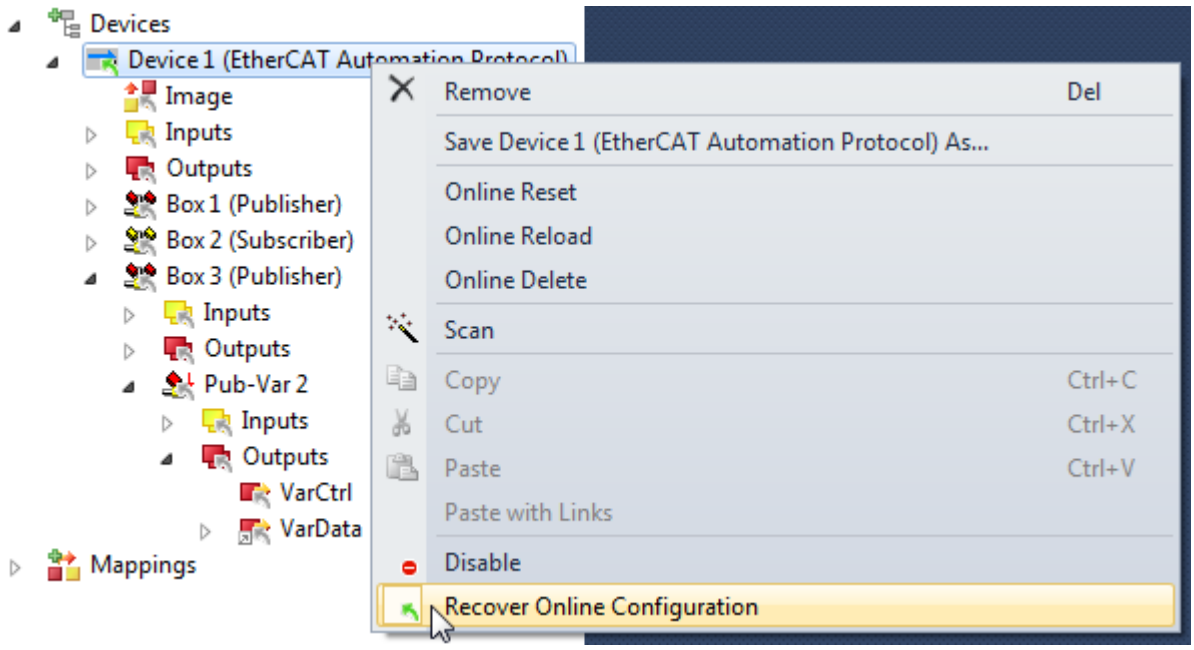
On the server side, an *RxPD* is configured for the reception of the request that works as in Pushed Data Exchange mode. In this example two *TxPDs* are configured for the transmission of the response to the client request. With both *TxPDs* the index of the *RxPD* that serves as the recipient of the request is entered in the *Poll Request RxPD* property. In each processing cycle a *TxPD* checks with its configured *Poll Request RxPD* whether a request has been received. If so, the *TxPD* is sent back in this cycle as the response.



6.6 Restoring the online configuration

If parameters of the *CANopen OD* are changed online for the EAP device, the current state of the *CANopen OD* is archived on the hard drive as soon as TwinCAT is switched back to the *Stop/Config* mode. When restarting TwinCAT in *Run* mode, the created archive is usually irrelevant, because the configuration made offline is simply restarted. All changes configured online are discarded.

If the online configuration is to be restored on restarting TwinCAT, then the option *Recover Online Configuration* must be switched on for this EAP device in TwinCAT (see following illustration) and the project must then be reactivated. In this case, when starting the controller, the offline configuration is initially loaded followed by the archive from the hard disk, so that the online configuration is active at the end of the start procedure.



It is not possible to change the configuration of the TwinCAT EAP device in TwinCAT if the option *Recover Online Configuration* is set. To do this the option must first be deactivated again. The sole purpose of this blocking mechanism is to protect the offline configuration against being changed by the user, even though this change would have no effect when activating the controller since the online configuration is restored.

Recover online configuration

The archived *CANopen* Object Dictionary is coupled to the instance of the EAP device in TwinCAT. As soon as this instance is deleted in TwinCAT and a new instance is created instead, the previously archived *OD* can no longer be restored for this instance.

7 The EAP Device Configuration (EDC) File

The *EAP Device Configuration (EDC)* takes place with the help of TwinCAT. At least the EAP device itself has to be created via TwinCAT so that an instance of the EAP device is generated accordingly when starting TwinCAT. The creation of *TxFrames, Tx/RxProcessData, Tx/RxPDOs* and *Tx/RxVariables* can then also be carried out subsequently by an *ADS-Client* application. TwinCAT must be in *Run* mode for this. In most cases, however, a TwinCAT EAP device is completely configured via TwinCAT and started as described in chapter Creation of an EAP configuration [▶ 26] and Configuration of an EAP device [▶ 41].

A configurator with a graphical user interface (GUI) – the TwinCAT EAP Configurator (Product description TE1610) – has been developed for the online configuration of an EAP device or an entire network of EAP devices. This configurator enables the graphical representation of all EAP devices in the network and their communication connections with one another. Also, each EAP device can be configured via the GUI.
A configuration that has been created in TwinCAT and to which an EDC file has been exported contains all the information required when importing into the configurator in order to display the current configuration of the EAP device. Documentation on the TwinCAT EAP configurator can be found in the Beckhoff Information System under *TE1610 EAP Configurator*.

More Information:
www.beckhoff.com/te1000

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