BECKHOFF New Automation Technology

Manual | EN

TS8020

TwinCAT 2 | BACnet/IP

Supplement | Building Automation

2023-08-11 | Version: 1.3

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

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 Introduction

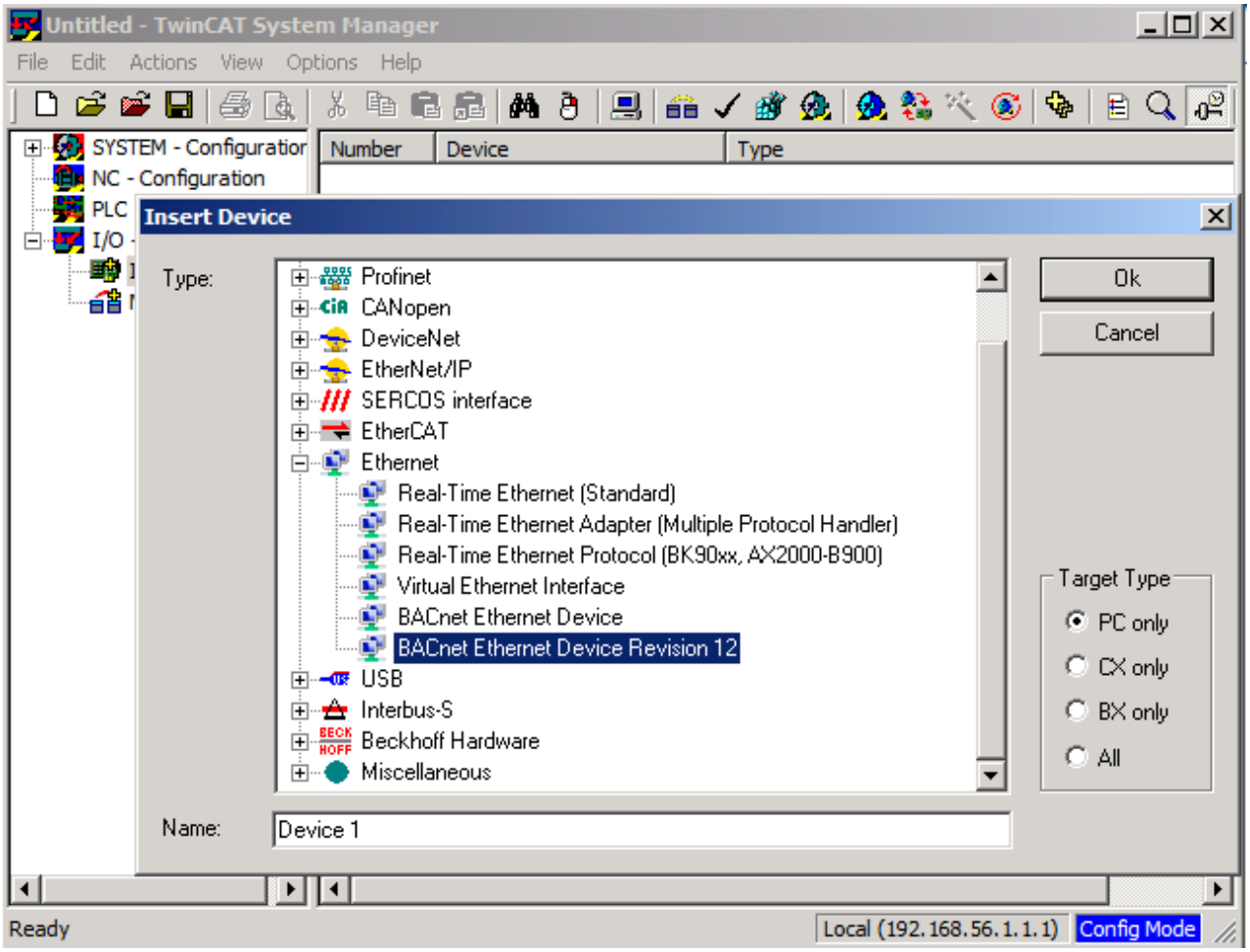
BACnet (Building Automation Control Network) is a standardized, manufacturer-independent communication protocol for building automation. Areas of application include HVAC, lighting control, safety, and fire alarm technology. Implementation of this protocol is carried out as server as well as client and can be run on all Beckhoff Industrial PCs and Embedded PCs. All services of a BBC (BACnet Building Controller) are supported such as for example, common data use (DS), alarm and event processing (AE), schedule (SCHED), trend recording (T) as well as device and network management (DM).

This documentation explicitly refers to the technical specification of TwinCAT BACnet/IP and therefore the implementation of the BACnet standard in the TwinCAT runtime and configuration environment. Sound knowledge of BACnet is a prerequisite. There is plenty of literature users may wish to consult in advance:

- Book: BACnet Gebäudeautomation 1.4, Hans R. Kranz, published by CCI Promotor, 2nd edition, ISBN: 3-922420-09-5
- Book: BACnetandBACnet/IP - A Clear Description, F. Tiersch, C.Kuhles, published by Desotron ISBN 978-3-932875-31-1
- External link: ASHRAE BACnetBibliography
- External link: BACnet Interest Group - BACnet Literature

2.1 BACnet device

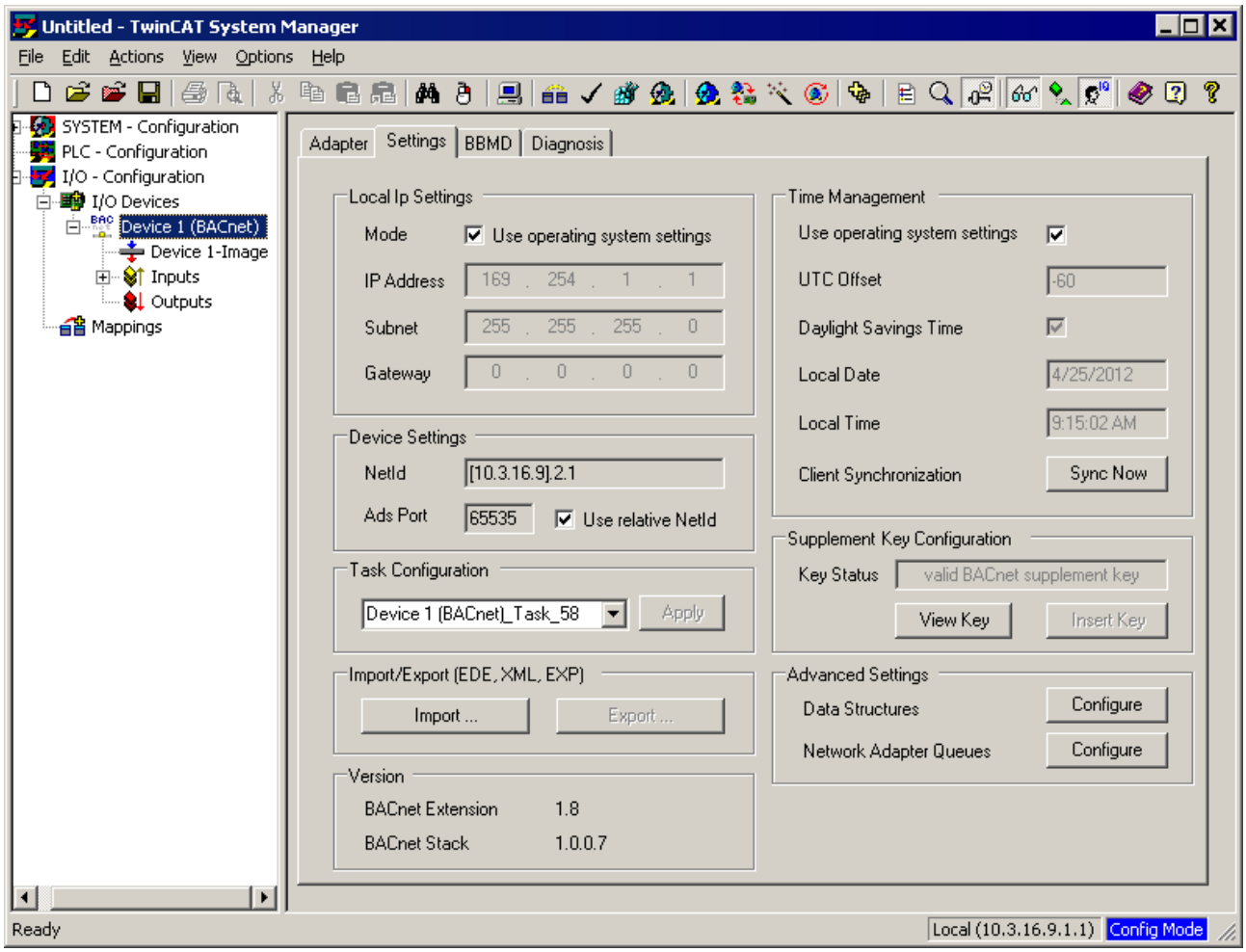
BACnet/IP is an Ethernet based protocol and can therefore be configured in TwinCAT in the "Ethernet" category. A BACnet device can be added to a configuration via the configuration element "I/O Devices". The BACnet device establishes the connection with a network adapter.



BACnet device settings

IP settings

The IP address of a BACnet device can be displayed and modified via the Settings tab. The IP address is read from the selected network adapter when the device is created. The IP address used for BACnet does not have to match the IP address of network adapter. It can be from a separate IP address space. A TwinCAT system can accommodate several BACnet devices, if several network adapters are available. The IP settings of the operating system can be acquired via the option "Use operating system settings". This option lends itself if a device obtains its IP address via DHCP, for example.



AmsNetID settings

Once a BACnet device has been created, further settings can be made via the Settings tab. Each BACnet device has an AmsNetID, which is used as access point for acyclic services. The AmsNetID has 6 digits. The first 4 digits are taken from the system NetID, the 5th is determined by the item ID (1st device: 2, 2nd device: 3, etc.), the 6th is always 1. To activate the BACnet configuration on another target system, the first 4 digits of the AmsNetID have to be adjusted accordingly. A corresponding dialog appears during connection with the new target system. Generally, we recommend activating mode "Use relative NetID". In this case the first 4 digits of the NetID are initialized with 0 and replaced with the current system NetID when the BACnet device is loaded. In this way it is possible to transfer an active system configuration to another target system by copying corresponding files (CurrentConfig.xml).

Task configuration

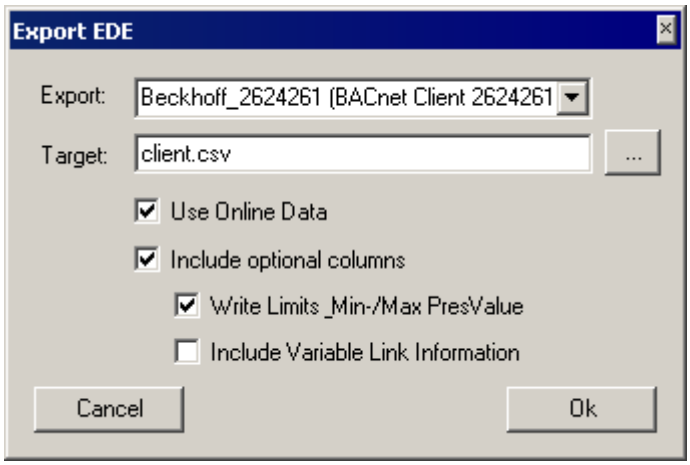
A BACnet device is linked with a task through asynchronous mapping. The corresponding task can be set in the field "Task Configuration". For a BACnet device any task with port >350 can be used. When a BACnet device is created, a default task with priority 58 is automatically created and allocated to the BACnet device.

Import/Export

The "Settings" tab of the BACnet device can also be used to import and export various file formats. A .exp file export, for example, makes a BACnet configuration available in a PLC. Declarations of function blocks of the TcBACnet.lib are created for all configured BACnet server and client objects, which can then be used in the PLC program. This enables convenient access to remote BACnet objects, for example, since process data variables are automatically linked during PLC automapping.

During XML export and import BACnet configurations are processed in a manufacturer-specific format.

Version 2.2 also supports *Engineering Data Exchange* (EDE) import and export. The EDE format is a comma- or semicolon-separated list with fixed and optional columns. Whether comma or semicolon is used as separator depends on the country-specific setting. The list separator specified in the operating system (Regional and Language Options) is used. Additional optional columns can be selected during export ("Include optional columns"). The optional column "vendor-specific address" can be used to copy information relating to the linked variables of the TwinCAT configuration to the EDE file. During EDE export of a BACnet server the user can choose whether offline data (settings dialogs) or online data are used for the export.



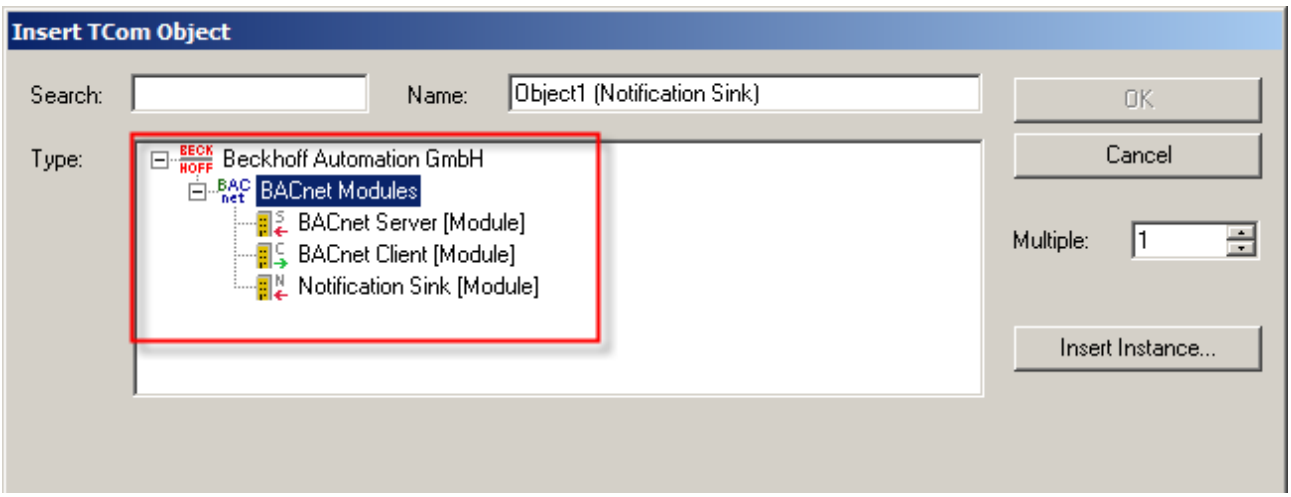
Time management

The "Time Management" group in the "Settings" tab of the BACnet device has an option to specify whether the time of the underlying operating system should be used. Summer and winter time is updated automatically, if configured accordingly in the operating system. For manual time setting the UTC offset and the DayLightSavingsStatus (summertime/wintertime) can be specified in this dialog. During the manual configuration of the time parameters the UTC offset and DayLightSavingsStatus can be modified via BACnet. Switching between summer and winter time leads to a 60-minute time switch.

Further BACnet device setting options are presented in the sections below.

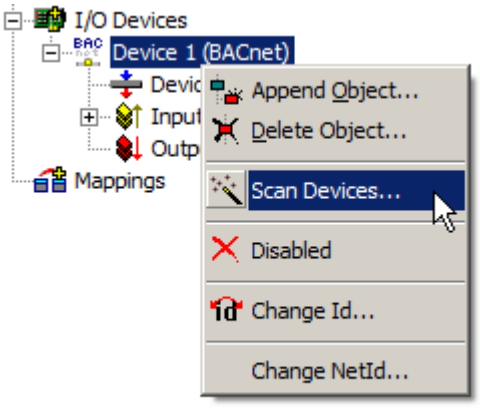
BACnet server/client

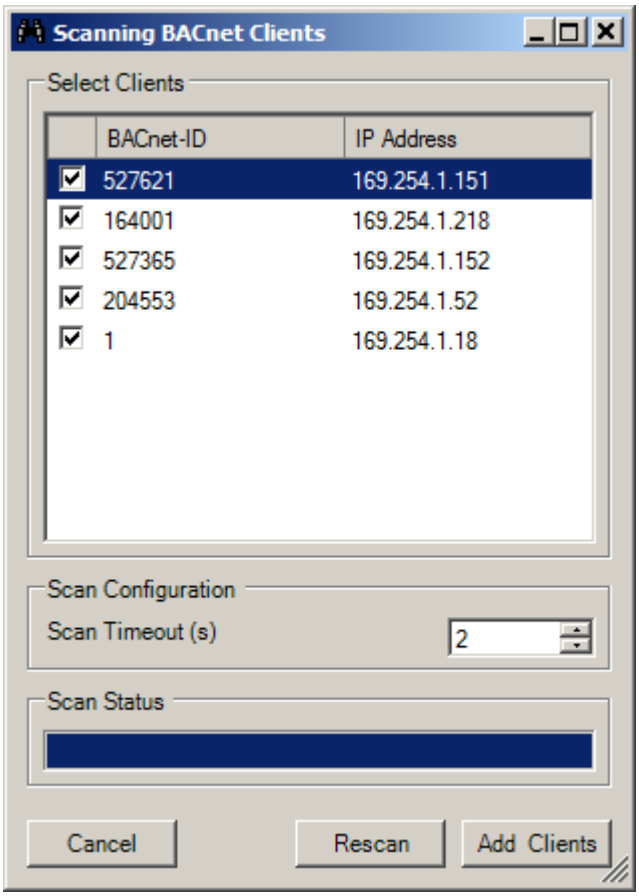
Under a BACnet device BACnet clients and server can be configured (see also example: "Create BACnet adapters and servers [▶ 92]"). A server makes BACnet objects available. A BACnet client enables access to BACnet objects of other BACnet devices (servers) in the network.



BACnet clients can be detected automatically by scanning the network. To this end a BACnet device is created, and the Scan function of the TwinCAT System Manager is used to find other devices (servers). For each found device the loaded BACnet objects are detected and created in the System Manager. Clients can also be configured manually with suitable BACnet device IDs, whereby the IP address is automatically determined with the BACnet service Who-Is. If the Scan function of the System Manager is used for a select client, the objects are detected so that no manual configuration is required.

Scanning of clients and their objects and properties requires an existing BACnet installation.



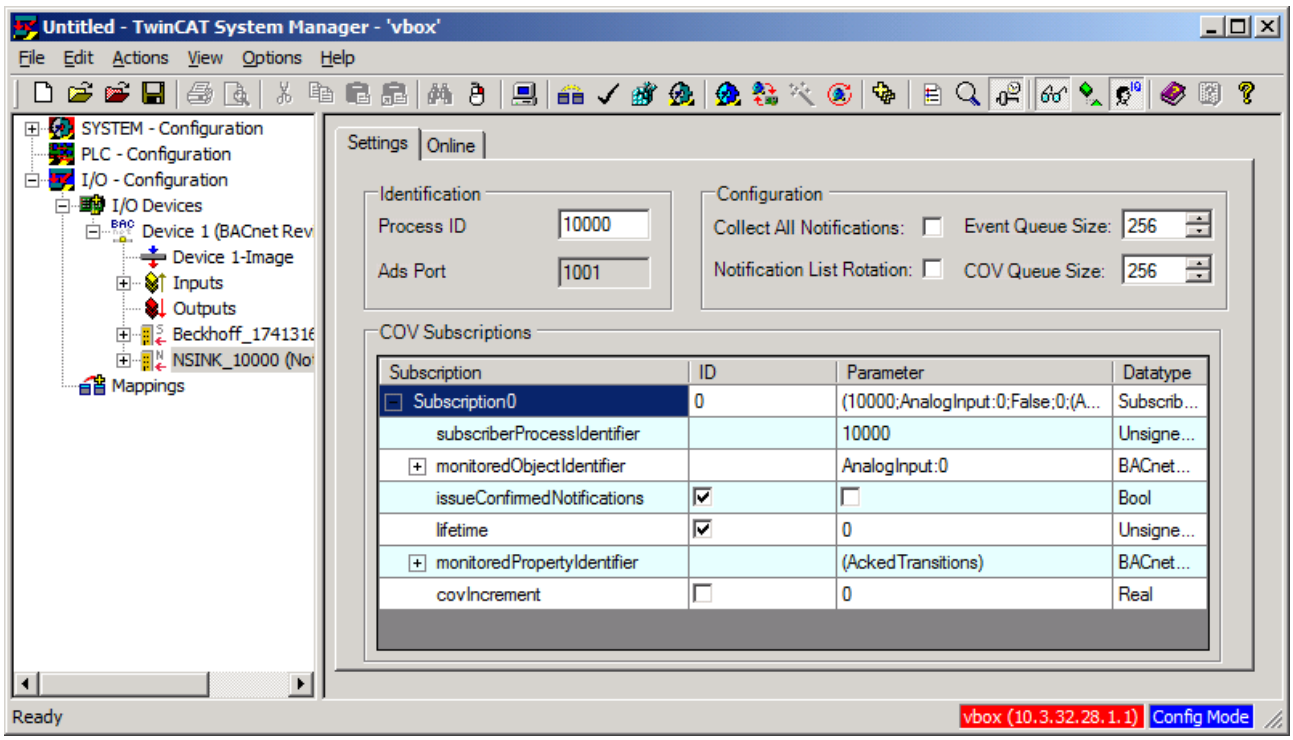


BACnet notification sink

The notification sink is a special feature of BACnet modules. It can be used to collect events and facilitate COV subscription. Functionally it is not a BACnet object, so that the notification sink is not visible for other clients from BACnet. However, the notification sink can be accessed via ADS services (e.g. TCP/IP, see section "ADS interface [▶ 513]").

Notification sink and certification

The notification sink is not included in the certification. Application of the notification sink is the responsibility of the user.



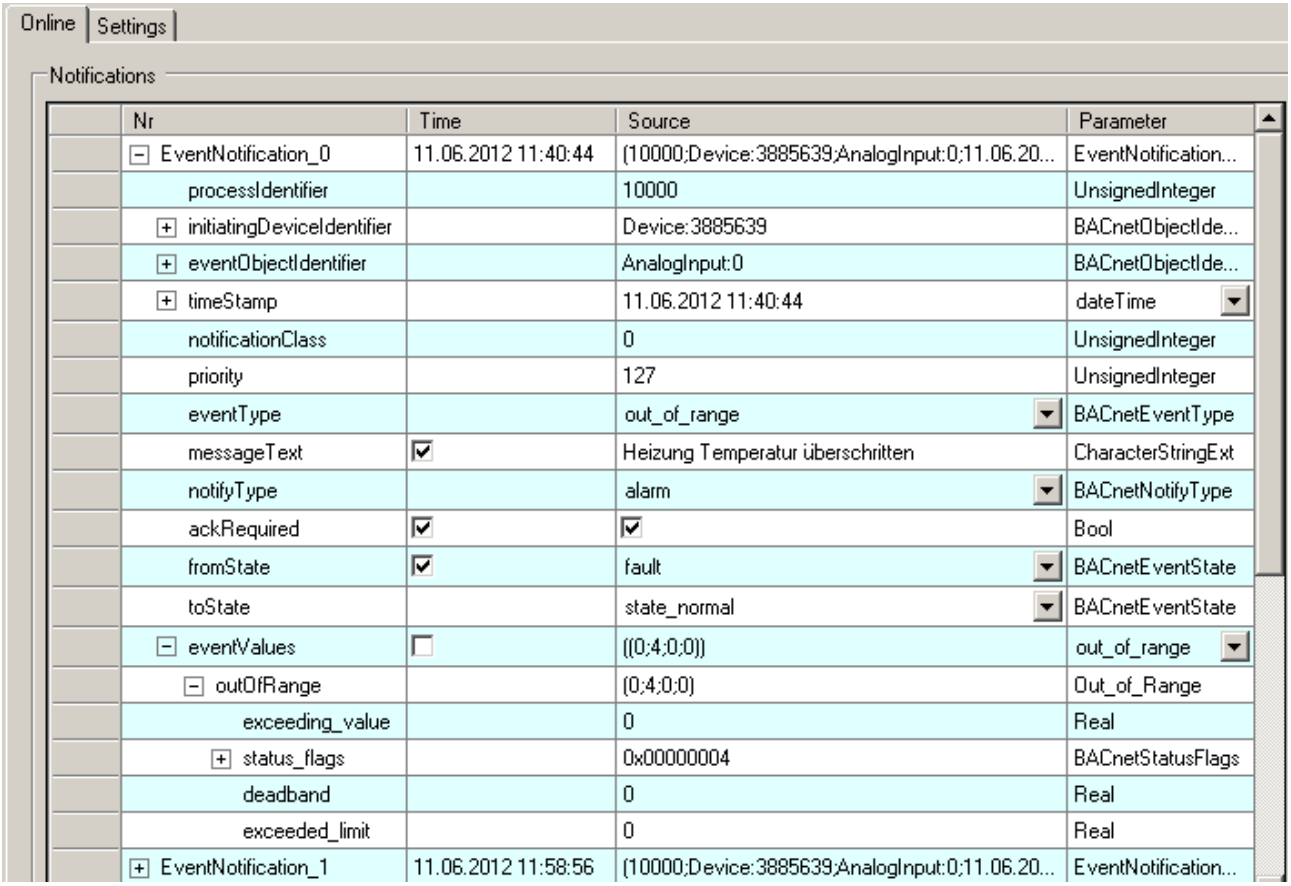
For each BACnet device several notification sinks can be configured (with different process IDs), for example to distinguish different weightings of event priorities. For each notification sink a unique ADS port is generated and displayed under "Identification". The size of the queues for COV and event notifications can be specified via the Configuration field. If a queue is full, further notifications are discarded. Via the ADS interface the user must ensure that the queues are processed suitably quickly, and entries are cleared (via ADS). Regarding the queues it should be noted that in the current implementation router memory is used for the notifications. The memory allocation for each notification is approx. 3 kbytes. The size of the router memory should match the queue configuration.

The functions Event and COV auto-acknowledge are currently not used. Acknowledgement of individual notifications is also not implemented at present.

The main use of notification sinks is processing and visualization of alarm messages (event notifications). COV notification subscriptions should only be used if state changes of complex BACnet properties must be implemented. This can be used to monitor whether new BACnet objects were dynamically added to a device, for example. For properties with simple data types of process data of remote objects should be used, since here too efficient COV notifications are implemented.

A different configured client can be set as target for COV subscriptions by double-clicking on the Identification field.

Received notifications (including receive time stamp) can be viewed in the Online tab. Individual event and COV notifications can be cancelled, and all queues can be cleared with a command available in the context menu of the Online tab.



The figure shows an EventNotification in the Online tab of a notification sink. TwinCAT BACnet/IP devices read the message text of an EventNotification from the Description property of the BACnet object sending the event.

EventNotifications can also be deleted via the function block FB_BACnet_NotificationSinkDelEntry [▶ 489] of the PLC library TcBACnet.lib.

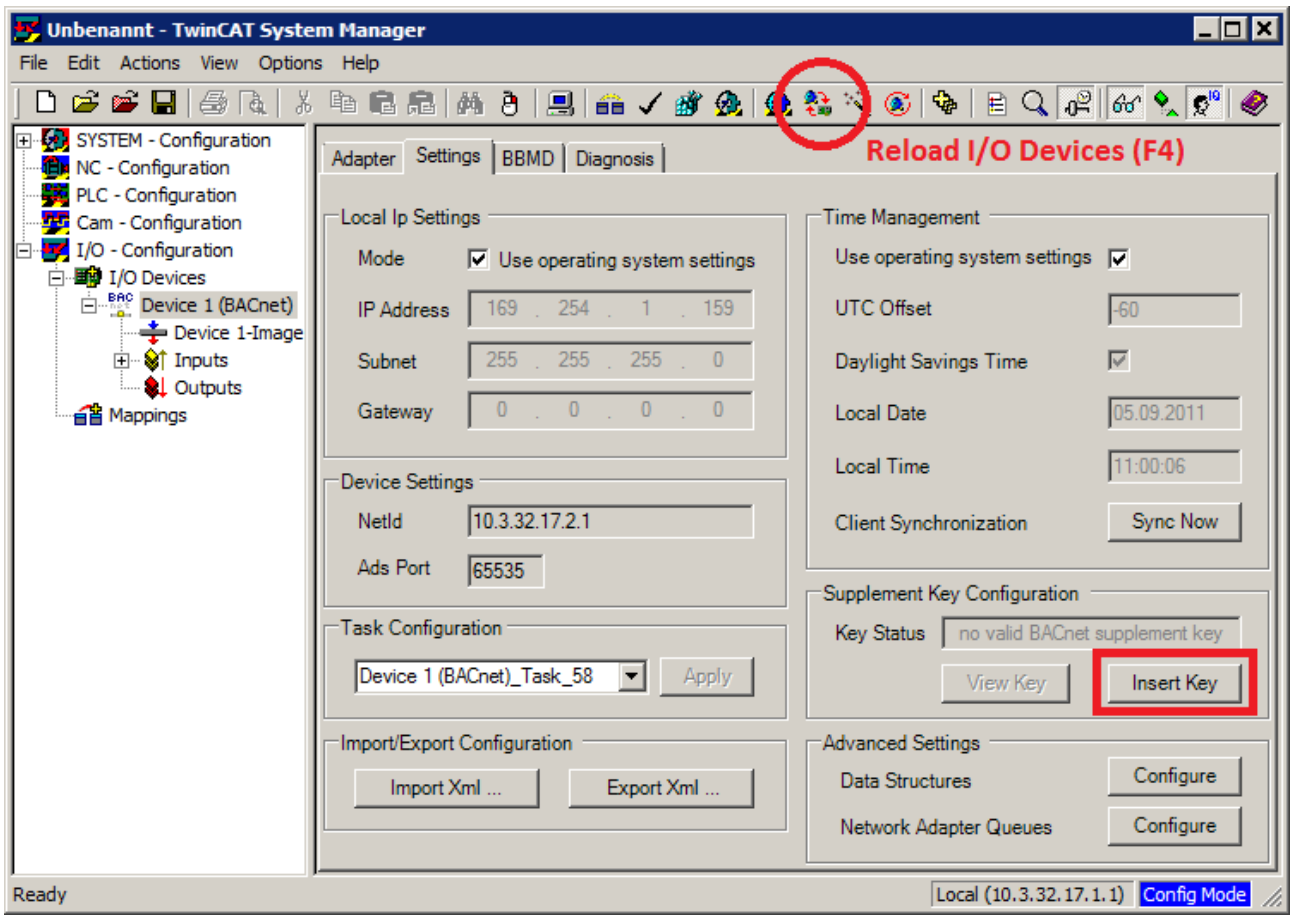
An example that illustrates the application of the NotificationSink and generation of EventNotifications can be found in section Application: Example: NotificationClass and NotificationSink. [▶ 129]

2.2 Setting up a BACnet Supplement Key

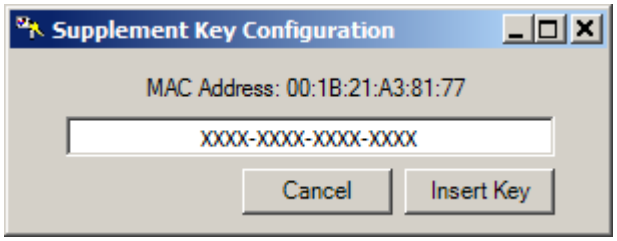
In order to be able to execute a TwinCAT BACnet/IP installation in RUN mode, it is necessary to obtain a supplement key. This key is linked to the MAC address of the network adapter. Supplement keys for several network adapters can be set up on the same computer.

Windows CE images

For Windows CE-based images the supplement key is already pre-installed in corresponding BACnet versions (special order number) and does not have to be entered retrospectively.



To add a supplement key a BACnet device must be created and loaded ("Reload Devices"). The key can then be added with the button "Insert Key" in the Settings tab of the BACnet device.



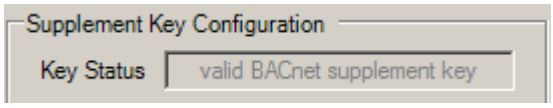
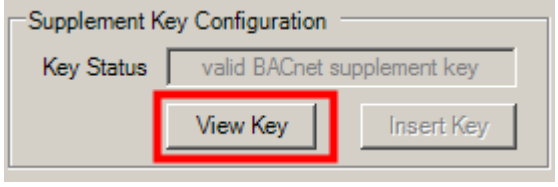


Image updates

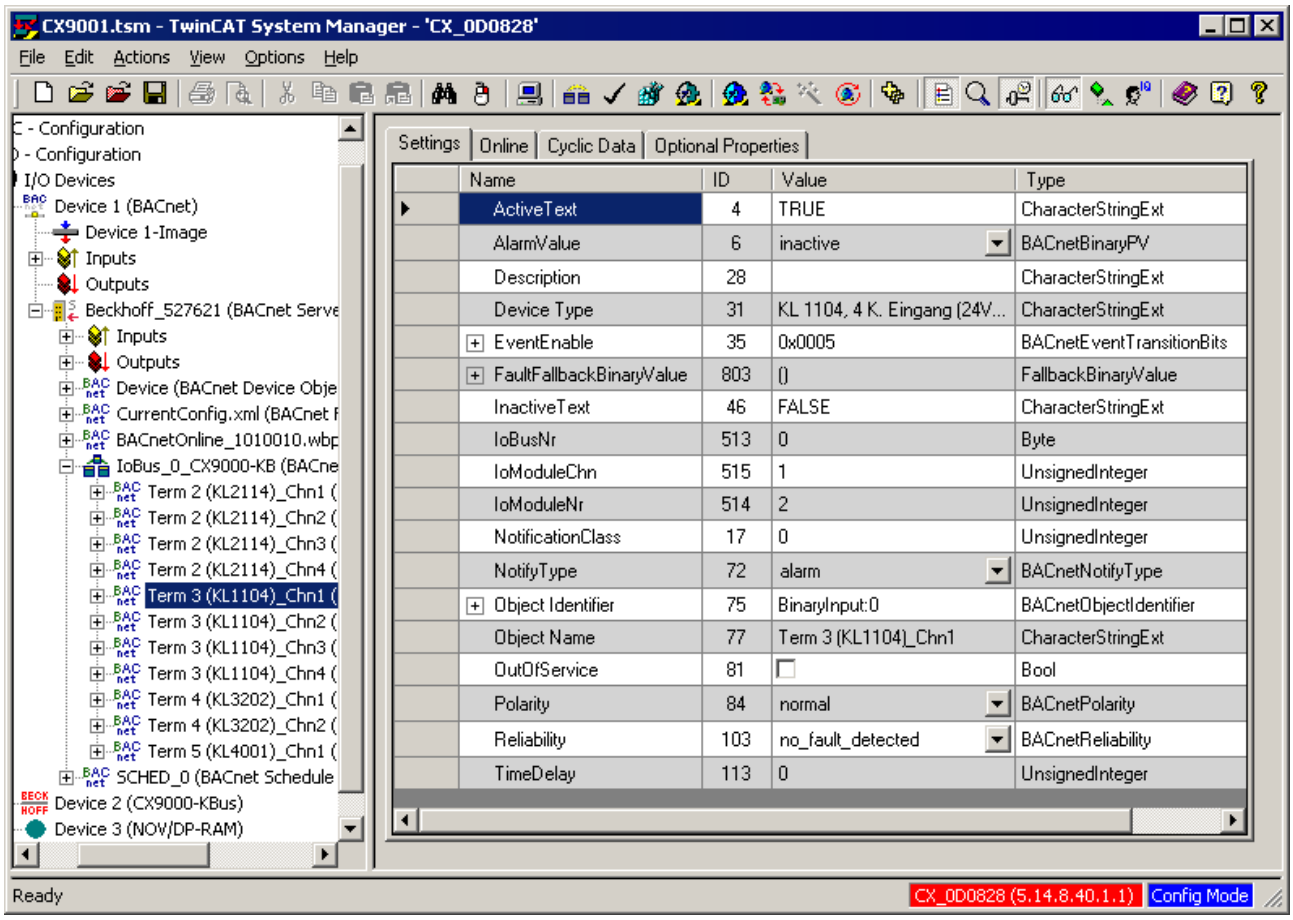
If a valid key has been entered, it can subsequently be displayed with the button "View Key". In Windows Embedded-based devices (CX5010,CX5020) a image update may result in the supplement key being lost. It is advisable to make a note of the supplement key before the update.



2.3 BACnet objects and properties

General information on handling

Each BACnet object has a defined number of properties associated with it. These properties are configured offline on the BACnet server side and made available to other BACnet devices in the BACnet network during operation (RUN or Free RUN). These offline data, or the initial values of the object properties are displayed in the "Settings" tab. The Online tab shows the current values of the object properties at runtime. Some properties are read-only (and identified as such by a lock symbol) and cannot be changed.

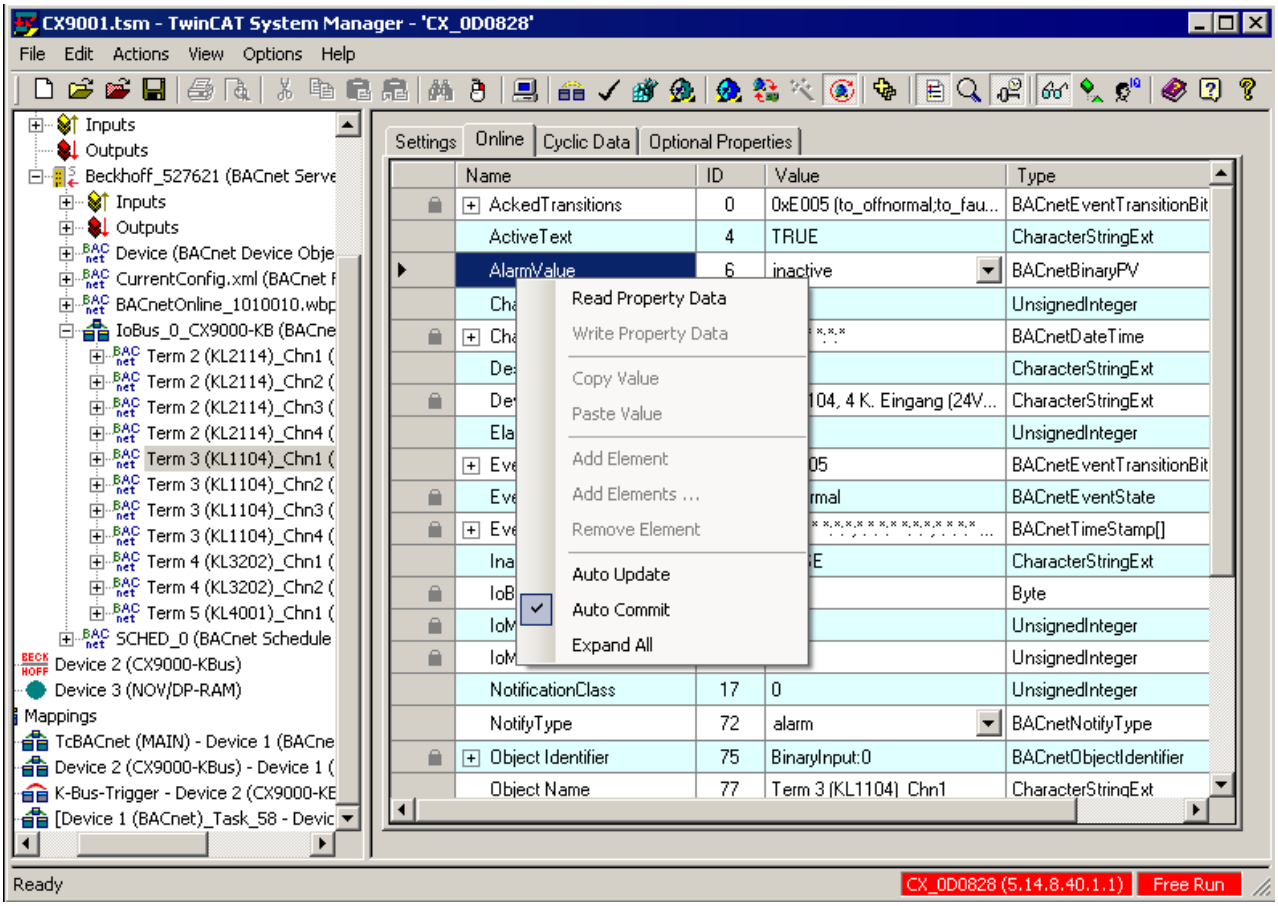


Each object property has a unique number. In TwinCAT BACnet/IP properties are subdivided into groups, which are identified by numbers as follows:

- Property ID 0...511: BACnet standard-compliant properties:
  - defined in ISO 16484-5
  - specified, defined manufacturer-spanning behavior
  - can be configured offline in part, in some cases only representation of an online state
- Property ID 512...799: manufacturer-specific properties with BACnet visibility:
  - visible in the BACnet network as properties of the corresponding objects
- Property ID 800...1000: manufacturer-specific properties without BACnet visibility:
  - used for configuring internal parameters that are not relevant for BACnet but are allocated to objects as parameters
  - configurable in the Settings tab of corresponding objects; not visible in the Online tab
  - includes fall-back values for I/O modules
- Property ID >1000: internal properties without BACnet visibility:
  - not visible in the Settings or Online dialog
  - used internally for special process data (e.g. "RawIo*")

- used in PLC automapping [▶ 72] for priority-based links and a device ID

In Online view a context menu can be opened by right-clicking.



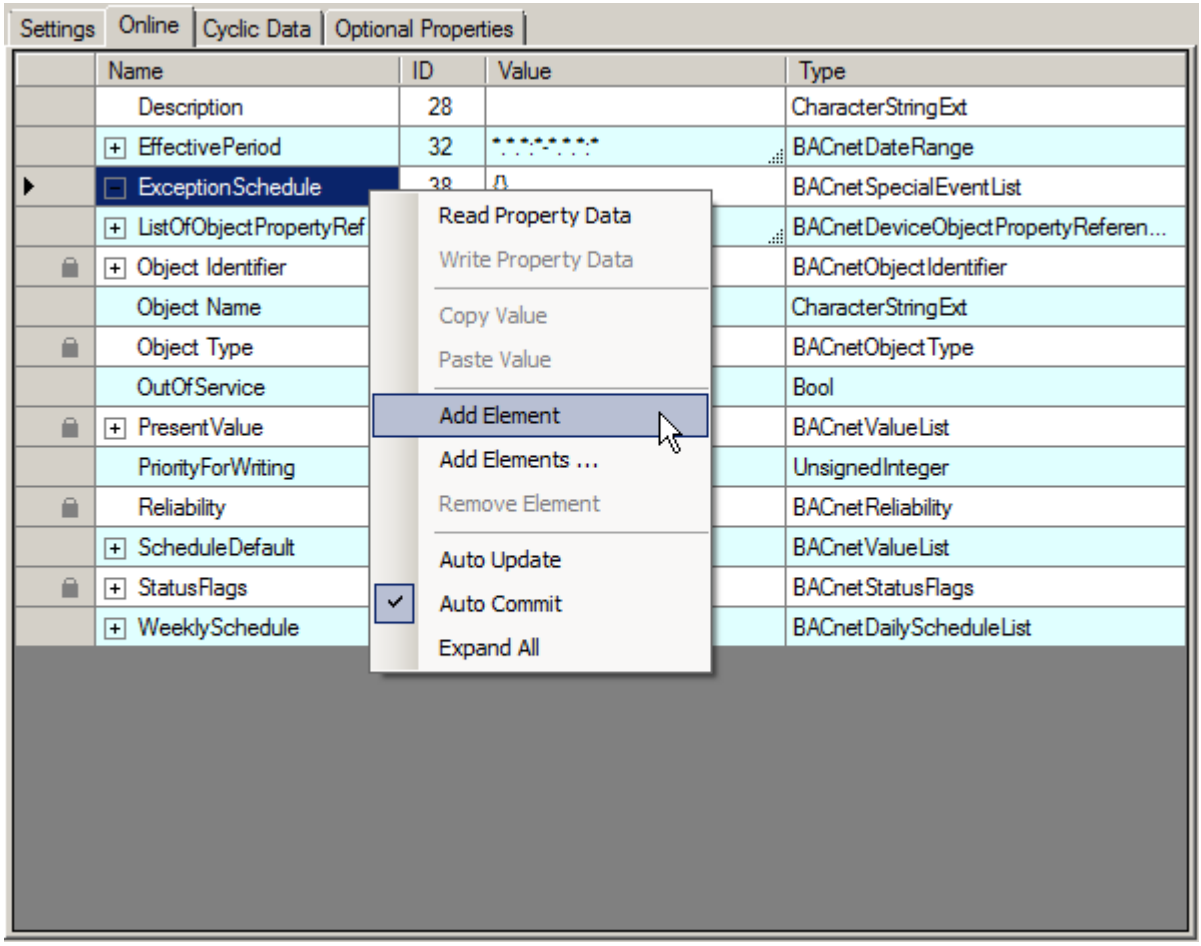
The menu entry "Read Property Data" can be used to read the value of an individual property.

ReadPropertyAll

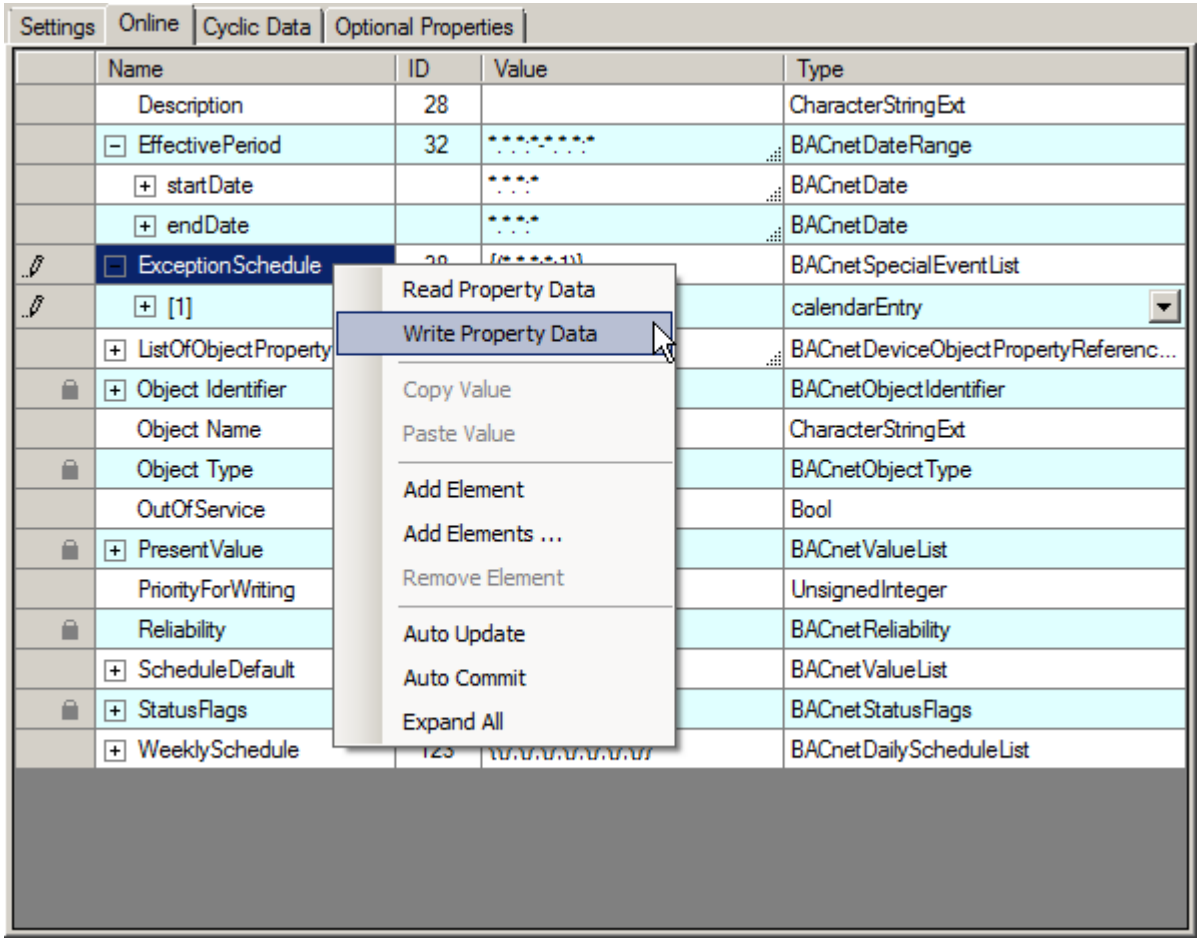
The overall view is read with the BACnet service ReadPropertyAll. If a device does not support this service, the properties are read individually.

If the function Auto Update is activated, all properties are read and displayed at 1-second intervals. In this case the individual properties no longer have to be read explicitly via the menu entry "Read Property Data". This feature is enabled by default.

List and array property elements can be added/removed with the Add/Remove function.



By default, changes in the property data are written immediately. Data can be manipulated without triggering immediate writing by deactivating the function "Auto Commit". This enables to make complex changes in the properties (e.g. in the property ExceptionSchedule), and the changes can subsequently be written in a single operation. The edited properties are identified with a pen symbol on the left-hand side until the changes have been written. The context menu option "Write Property Data" can be used to write the changes. Please note: the option WritePropertyMultiple is currently not implemented. Instead, individual WriteProperty calls are used.

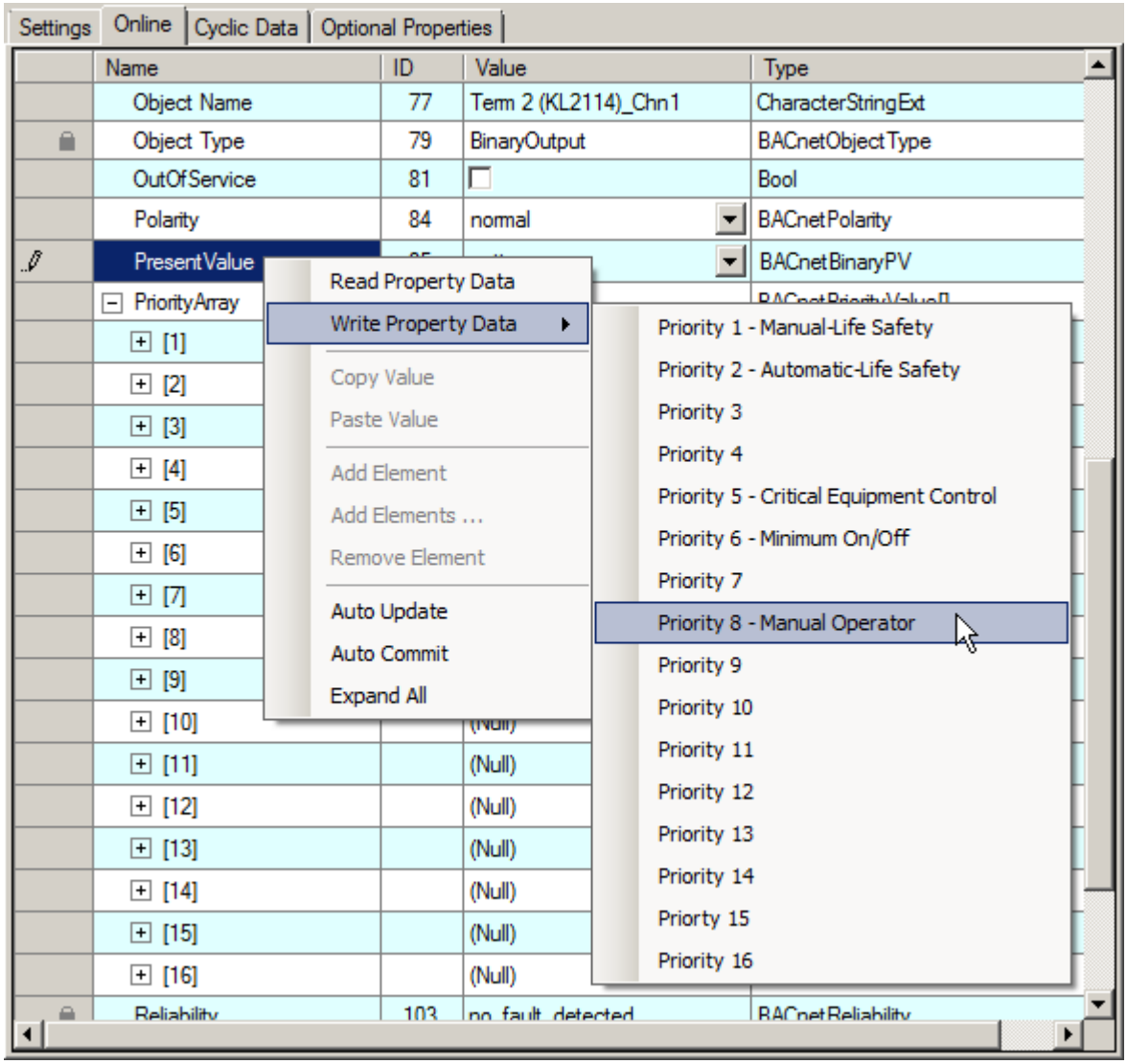


NOTICE

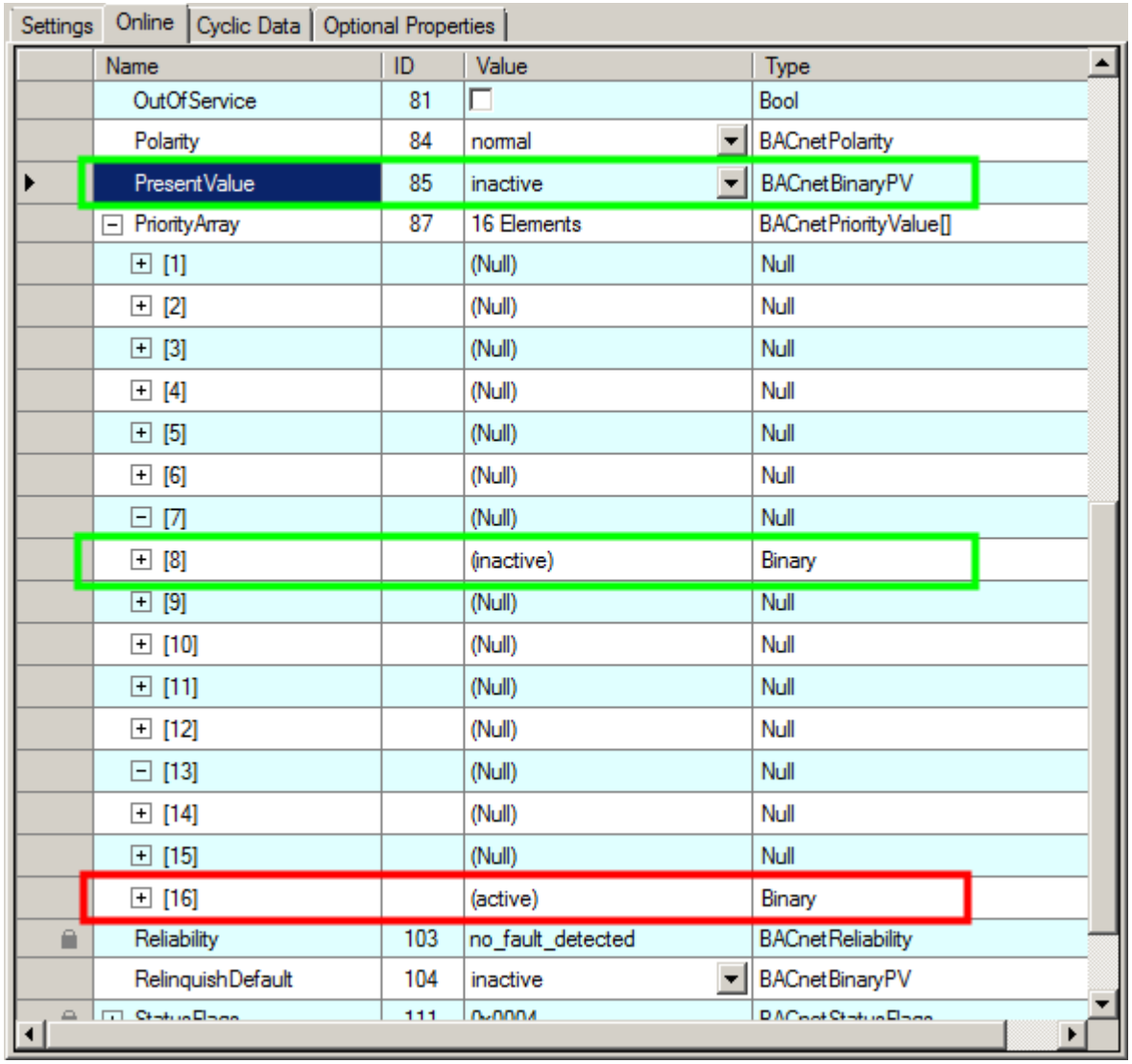
Deactivate auto update!

If "Auto Commit" is deactivated, the function "Auto Update" should also be deactivated in order to prevent overriding of edited data on reading.

Certain objects have commandable properties. In these cases a write access priority can be specified. This value is written to the PriorityArray via the index linked with the priority.



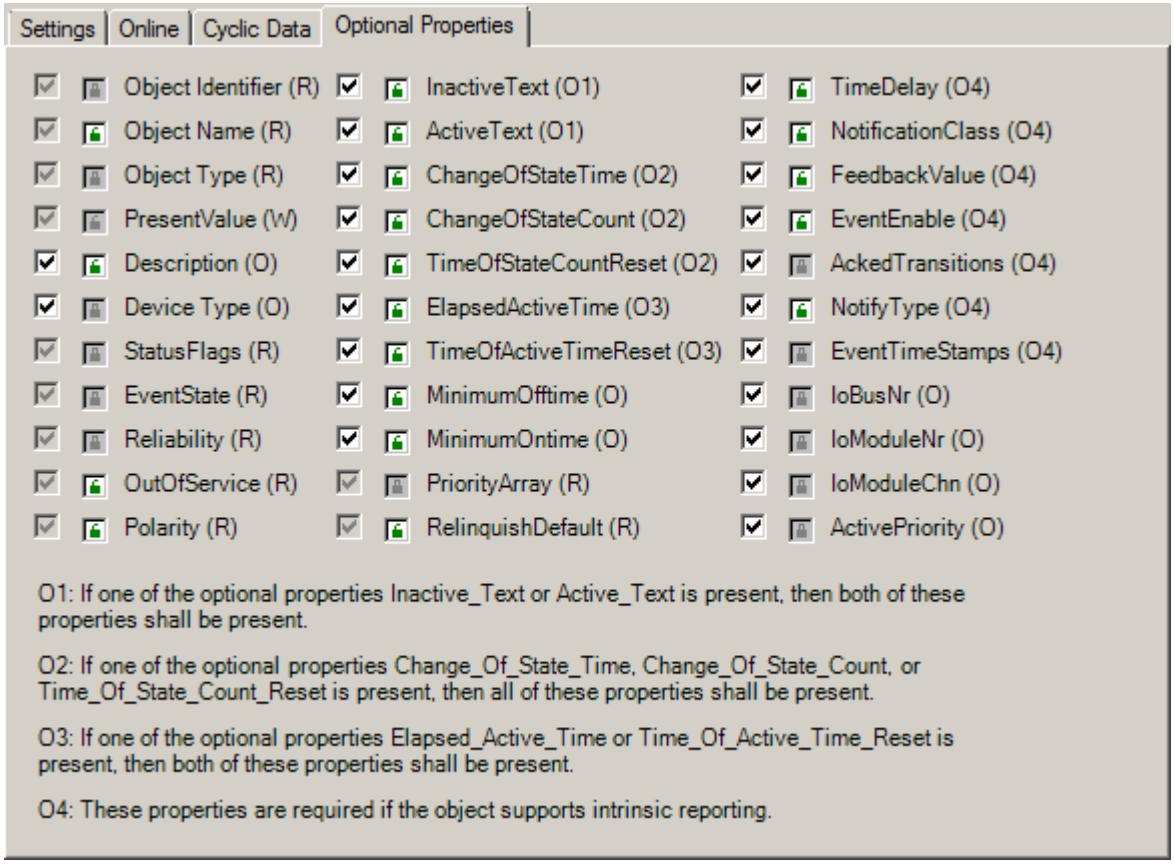
The PriorityArray also offers direct write access. All entries with the value "(Null)" are ignored. Depending on the priority, entries with a concrete data type and a valid value act on the corresponding, commandable property or are overridden by another entry with higher priority. The entries from the PriorityArray determine the value of the PresentValue of a BinaryOutput object, for example.



To remove an entry from the PriorityArray, set the type to NULL in the Type column. Entries can also be deleted from the PriorityArray via the context menu of the commandable property. To this end select NULL as type and write it to the corresponding priority with "Write Property Data" or automatically with "Auto Commit".

Configuration of optional properties and property write protection

The BACnet standard defines the standard objects and their properties (property ID 0 - 511). Some properties are identified as optional, other as obligatory. In order to achieve maximum flexibility in the configuration of objects, optional properties can be set to be in view or hidden and the access type can be limited, if permitted (read-only or read/write) in the Object view under the "Optional Properties" tab.



Information shown after the property name (R/W/O)

The letter after the property name (R/W/O) indicates the minimum access right according to the BACnet specification. "R" stands for readable (or higher) properties, "W" for readable and writeable and "O" for optional readable and/or writeable properties. The identification number of optional properties indicates their interdependence. Optional properties with the same number can only be activated or deactivated as a group. Notwithstanding the BACnet specification some normally optional properties are obligatory in TwinCAT BACnet/IP and can therefore not be disabled. This due to the internal implementation.

For some properties write access can be specified. The green, open lock symbol indicates write access.

The red, closed lock symbol indicates read only access. Click on the symbol to toggle the state. In case of greyed out symbol the permission cannot be changed. The access permission is indicated by "W" or "R" or the greyed out lock symbol ("W" and open, grey lock = write/read access, "R" and closed, grey lock = read only access).

The availability of optional properties can be controlled by clicking on the symbol checkbox . Deselecting the checkbox disables the corresponding property.

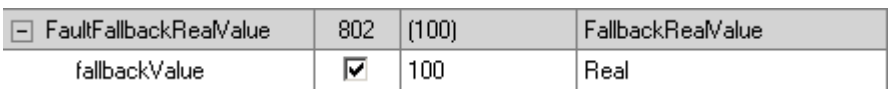
Manufacturer-specific properties

TwinCAT BACnet/IP uses a number of manufacturer-specific properties, which are explained below. Vendor-specific properties are distinguished through their property IDs. Properties with IDs between 512 and 799 are visible across the BACnet network. Properties with higher IDs are used for the configuration within TwinCAT BACnet/IP and are not displayed via BACnet. These properties are visible in the Settings tab, but not yet in the Online tab.

ScaleOffset (512)	ScaleOffset is used for AnalogInput and AnalogOutput objects and determines the offset for the conversion between the hardware range to BACnet and vice versa.

	For AnalogInput objects: PresentValue = RawIoSingedValue * Resolution + ScaleOffset For AnalogOutput objects: RawIoSingedValue = PresentValue / Resolution - ScaleOffset The same calculation algorithms apply for the process data properties RawIoUnsingedValue.
IoBusNr [0..255] (513)	IoBusNr is used for mapping a fieldbus state, e.g. during K-bus I/O automapping. Further details can be found in sections IO Automapping [▶ 72] and Process data [▶ 42]. 255 indicates that a BACnet object is not allocated to any IoBus.
IoModuleNr [0..65535] (514)	IoModuleNr indicates the position of a terminal within an I/O bus. For I/O automapping consecutive numbers are used, starting with 1. This property is for information purposes only and is not used in the BACnet stack.
IoModuleChn [0..255] (515)	IoModuleChn indicates the channel number of an I/O terminal. For I/O automapping consecutive numbers are used, starting with 1. This property is for information purposes only and is not used in the BACnet stack.
PersistentData (516)	Persistent Data indicates whether changes have occurred since the last saving of persistent data. In addition, this property can be used to trigger saving of persistent data via BACnet. Details relating to this property can be found in section Persistent data [▶ 57]. This property is only visible if persistent data were activated and the configuration was loaded in RUN mode.
ActivePriority (517)	ActivePriority indicates the active priority level for objects with commandable PresentValue properties. If all entries in PriorityArray are NULL, the value of Active Priority is 17, and ReliquishDefault is used as value for PresentValue.
LastConfirmedServiceAccess (518)	The property LastConfirmedServiceAccess of the device object can be used to display the IP address of the last successfully executed and confirmed service (e.g. read, write property) in the form of a string. In COV mode a TrendLog object can be used to set up an access log for documenting all network access.
DataRequestMode (519) DataPollingInterval (520) EnrollmentCovResubscriptionInterval (521)	EventEnrollment object - monitoring of external objects. These properties can be used to configure the data transfer method of an EventEnrollment object. The property DataRequestMode of data type Enum specifies the service used to read data (ReadProperty, COV, ReadPropertyMultiple). The property DataPollingInterval specifies the interval at which data are requested in transmission types ReadProperty and ReadPropertyMultiple. EnrollmentCovResubscriptionInterval specifies the ResubscriptionIntervall for the COV mode.
FileName (522)	In the File object, the corresponding file is identified via the property ObjectName. If the operator requires a fixed naming scheme for the File object, the file can no longer assigned via the property ObjectName. The property FileName can be optionally enabled. If this property is active, the corresponding file, relative to the boot folder, is allocated via the value of the character string, and the property ObjectName can be used freely.

Requirements

FaultFallbackRealValue (802)	FaultFallbackRealValue can be used for AnalogInput objects, so that valid values are provided in the event of an I/O bus error in the higher-level PLC or BACnet. If an IoBus error is detected, TwinCAT BACnet/IP no longer copies the corresponding process data of the fieldbus, but the configured fall-back values.  The fall-back value can be activated in the Settings tab by ticking the box for the optional value. If no fall-back value is active, in the event of an IoBus error no process data are copied, and the last valid value remains active.
FaultFallbackBinaryValue (803)	FaultFallbackBinaryValue has the same purpose as FaultFallbackValueReal and is used for BinaryInput objects.

OutOfServiceFallbackRealValue (804)	OutOfServiceFallbackRealValue can be used for generating valid output process data if an AnalogOutput object is in OutOfService mode. If OutOfServiceFallbackRealValue is active, the corresponding value for the I/O process data is used. Otherwise, if OutOfService is active, the last valid PresentValue value is used.
OutOfServiceFallbackBinaryValue (805)	OutOfServiceFallbackBinaryValue has the same purpose as OutOfServiceFallbackRealValue and is used for BinaryOutput objects.
TimeSynchronizationUseUTC (806) *From revision 12 this property is no longer used. The specified properties UTCTimeSynchonisationRecipients and TimeSynchronizationRecipients are now supported. The functionality is retained as described.*	TimeSynchronizationUseUTC is used for configuring a TimeMaster and indicates whether the TimeSynchronization service or the UTCTimeSynchronization service is used for sending synchronization telegrams to the TimeSynchronizationRecipients. Revision 6, which is implemented in TwinCAT BACnet/IP, does not yet make provision for TimeMaster-specific properties. In this context TwinCAT BACnet/IP implements the standard-compliant properties AlignIntervals, IntervalOffset and TimeSynchronizationInterval without BACnet visibility. The property TimeSynchronizationInterval (204) indicates the interval (in minutes) at which time synchronization telegrams are sent. If the value of this property is 0 the TimeMaster function is disabled. The property AlignIntervals (193) can be used to enable the property IntervalOffset (195), which can be used to specify when exactly time synchronization telegrams are sent. If the property TimeSynchronizationInterval has a value that is divisible by 60, time synchronization messages are sent when the current time in minutes modulo matches the value of TimeSynchronizationInterval and IntervalOffset. The following example explains the configuration of a Timemaster, which sends UTC time synchronization telegrams to all BACnet devices in the network every day at 2 am: TimeSynchronizationUseUTC = TRUE AlignIntervals = TRUE TimeSynchronizationInterval = 1440 IntervalOffset = 120 TimeSynchronizationRecipients = {((FF:FF:FF:FF:FF:FF;0))}
PredictScheduleValueTime [sec] (807) PredictedScheduleBoolValue (808)	In building automation applications, HVAC systems can save energy through "optimized/sliding" switch-on. If, for example, a set temperature of 21°C is required at 8:00, a heating system can be activated earlier or later depending on the season. At low outside temperatures the heating will come on earlier than at higher outside temperatures. The heating activation time can be calculated based on a suitable mathematical equation or through learning systems. To implement this functionality the vendor-specific properties *PredictScheduleValueTime* and *PredictedScheduleBoolValue* are available in the BACnet object Schedule. The calculated system pre-activation time can be transferred from the PLC to BACnet via the process data property PredictScheduleValueTime. The BACnet stack uses the process data property PredictedScheduleBoolValue to calculate the predicted value (PresentValue) of the Schedule object in the future. The unit of *PredictScheduleValueTime* is second. If, for example, the time is 7:51 and the value of PredictScheduleValueTime is 540 (9 minutes), PredictedScheduleBoolValue will be assigned the *value of the PresentValue* of the Schedule object at 8:00.
AccumulatorIntegrationMode (811)	The Accumulator object can be used to accumulate count values via the process data property RawIoAccumulatorValueUSINT. In normal mode the counter value of the accumulators is incremented by 1, if the value of RawIoAccumulatorValueUSINT is incremented by 1. Integration mode can be activated for measuring quantities per time. In this case, the count value of the accumulator is incremented if the value of RawIoAccumulatorValueUSINT was 1 over one millisecond. This functionality can be used to calculate an air volume, for example, if RawIoAccumulatorValueUSINT is linked to the current air flow. The time base is 1 ms. The BACnet property *PreScale* can be used for scaling, e.g. on an hourly basis.

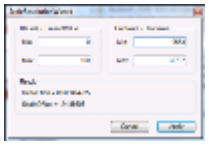
<table>
<tr><td>AvgFilterCycles (812)</td><td>Analog input values may result in fluctuating values, due to long cables or interference. When AnalogInput objects are used in BACnet, the property CovIncrement can be used to limit the transfer of value changes. If the fluctuations exceed the resolution of values to be displayed, it may be advisable to smooth out the analog input values through filters. The property AvgFilterCycles can be used to activate an averaging filter for AnalogInput objects, which forms the average value of the raw value over AvgFilterCycles. With a BACnet cycle time of 50 ms, a value of AvgFilterCycles = 20 indicates a filter over 1 second.</td></tr>
<tr><td>FaultDeadband (813)</td><td>A further potential problem caused by fluctuating analog values is violation of a value range defined as valid. For AnalogInput objects the properties MinPresValue and MaxPresValue define a valid range for the value of the property PresentValue. If the value scaled from the raw* value is outside the defined range, the BACnet object assumes Fault state. Since the properties MinPresValue and MaxPresValue are also read as value range by the control system, these limits should not be shifted in an attempt to tolerate values that fluctuate slightly below the limit as valid. Similar to the property Deadband, which determines the transition to OffNormal state, the property FaultDeadband can be used to specify an extended limit for the Fault state. If the value calculated by the scaling falls below the value of MinPresValue by less than FaultDeadband, the PresentValue assumes the value MinPresValue, and the object does not assume Fault state. The exact calculation formula is implemented as follows:

IF MaxPresValue + FaultDeadband > PresentValue > MaxPresValue THEN
PresentValue = MaxPresValue
IF MinPresValue - FaultDeadband < PresentValue < MinPresValue THEN
PresentValue = MinPresValue

Below is a small example outlining the use of this property:

Example:
- Analog input signal 2V-10V
- Input terminal 0V-10V [0..32767]
- Mapping to 0% - 100% (PresentValue)
- Values up to 1.8V should be tolerated as valid ?

Calculation:
- 2V = 6553
- 10V = 32767
- MinPresValue = 0 (0%)
- MaxPresValue = 100 (100%)
- FaultDeadband: 1.8V corresponds to a value of 5898. FaultDeadband takes effect in the value range of PresentValue; the scaled value is: 5898 *0.0038 – 24.99 = 2.5%. In the example, the FaultDeadband therefore has to have a value of 2.5, in order for 1.8 V still to be displayed as a valid value.

  </td></tr>
<tr><td>Pt1DampFactor [0..1] (814)</td><td>Similar to the property AvgFilterCycles, the property Pt1DampFactor can be used to activate an analog input filter. The Pt1 filter is applied after the scaling and smoothes the analog value. The damping factor can have values between 0 and 1. No filtering takes place if the value is 1. If the value is 0, the new input value is only applied at infinity. Meaningful values can be 0.01 or 0.001, for example. A TrendLog object can be used to verify the effect of the filter for a simple pulse response. (Change of the raw value from 0 to 1). The implemented formula is:
$PresentValue_{new}=Pt1DampFactor*(Input-PValt)+PresentValue_{old}$
PT1 and averaging filter can be operated in parallel. The averaging filter is more runtime-efficient, since integer arithmetic is used.</td></tr>
</table>

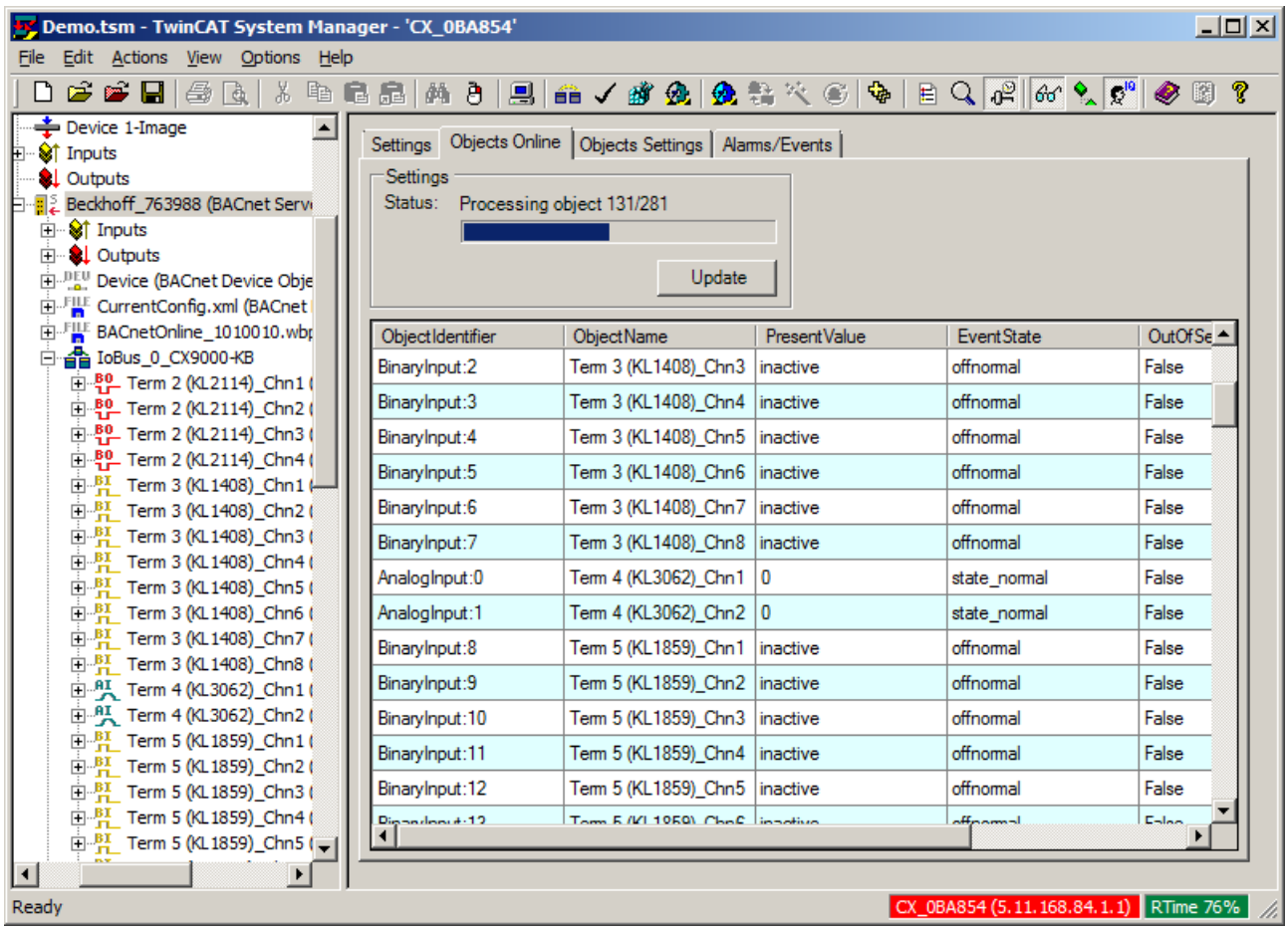
EnableInternalLoopCtrl (815)	The property EnableInternalLoopCtrl can be used to activate an internal control algorithm in the Loop object. Normally, the loop is implemented as an empty shell for a controller implemented in the PLC. A basic controller was integrated in the BACnet stack, in order to provide support for dynamic generation of Loop objects. This PID controller additively links the P, I and D components and implements a limitation for the integral and sum component. The formula implemented in the Loop object when the EnableInternalLoopCtrl is activated (and set to TRUE) is shown below: `IF (Action = = direct) E = W - X IF (Action = = reverse) E = X - W Yp = Kp * E // Proportional Yi = Yi(n-1) + (Ki * tCycle * E) // Integral Max >= Yi >= Min // Anti-Wind-Up Yd = Kd * (E - E(n-1)) / tCycle // Differenzial Y = Yp + Yi + Yd + Bias Max >= Y >= Min // Ausgangslimitierung`
Pt1DerivativeDampFactor [0..∞] (816)	In order to be able to handle step changes in the setpoint specification for the control algorithm implemented in the Loop object, the error value (E) of the D component can be attenuated via a Pt1 factor. This enables the effect of the D component to be amplified when step changes occur in setpoints.

2.4 Help functions and wizards

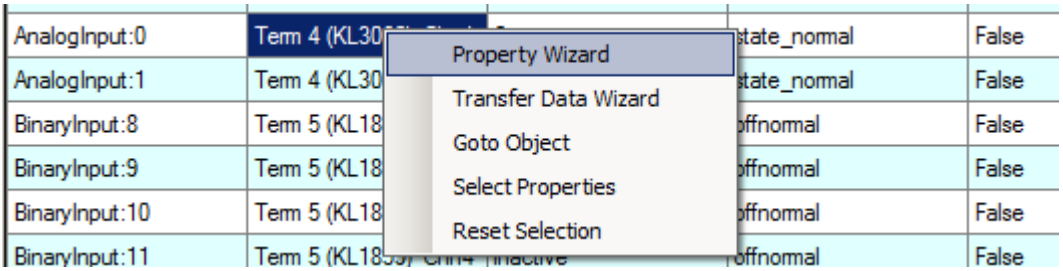
Wizards were implemented to simplify the configuration of TwinCAT BACnet/IP and make it more transparent. They are briefly described in this section. Th availability of a wizard is indicated by a small triangle () on the bottom right of the Property view in the Setting and Online tabs of the objects. The wizards can be used for client and server objects.

Object overview

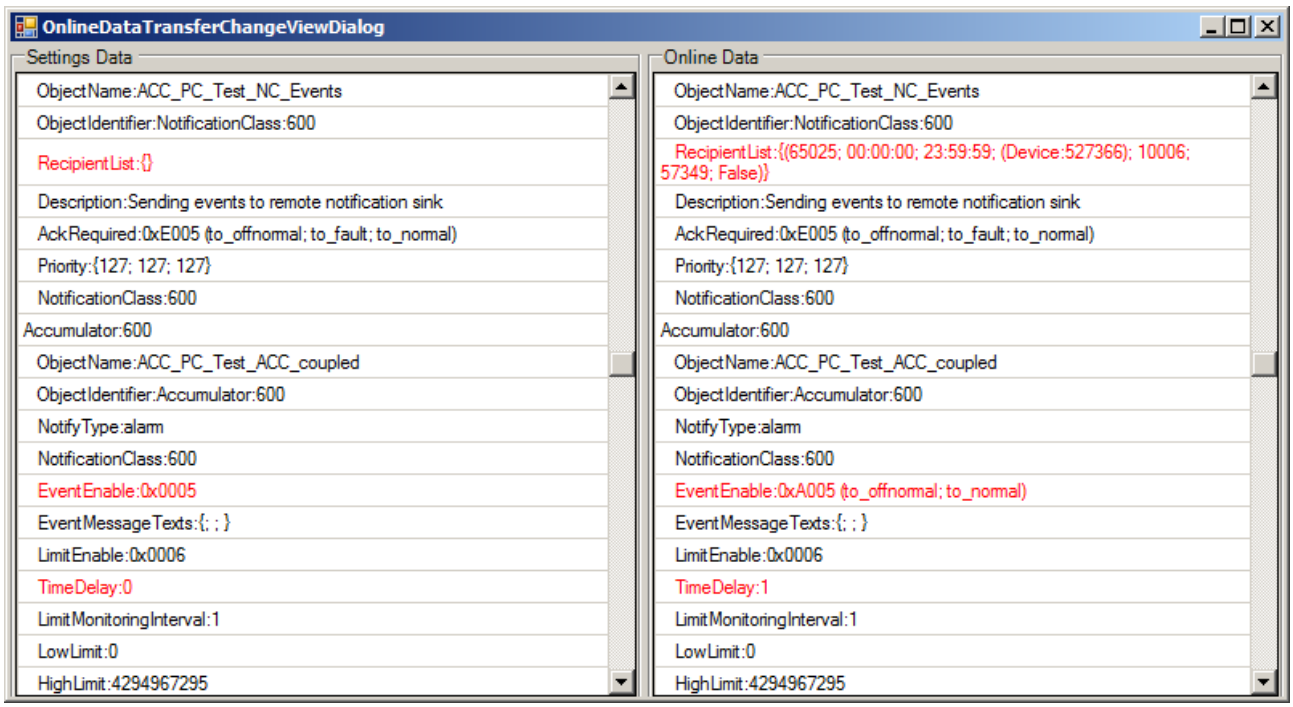
Under BACnet client and server, there are tabs "Objects Online" and "Objects Settings" respectively. In this view selected properties of the objects of a BACnet client or server are shown. BACnet objects can be sorted based on different criteria by clicking on the table header. In this way it is possible to show all objects with *fault* state at the top, for example. In general, "Objects Online" represents all online values of the BACnet objects; "Objects Settings" represents the settings values. The Copy&Paste feature (CTRL-C, CTRL-V) can be used to copy property values in this view. If several cells are selected, CTRL-V will change several property values.



Right-clicking on the object overview opens up a context menu through with the Process Data wizard for BACnet clients or the Property wizard for BACnet servers can be activated.



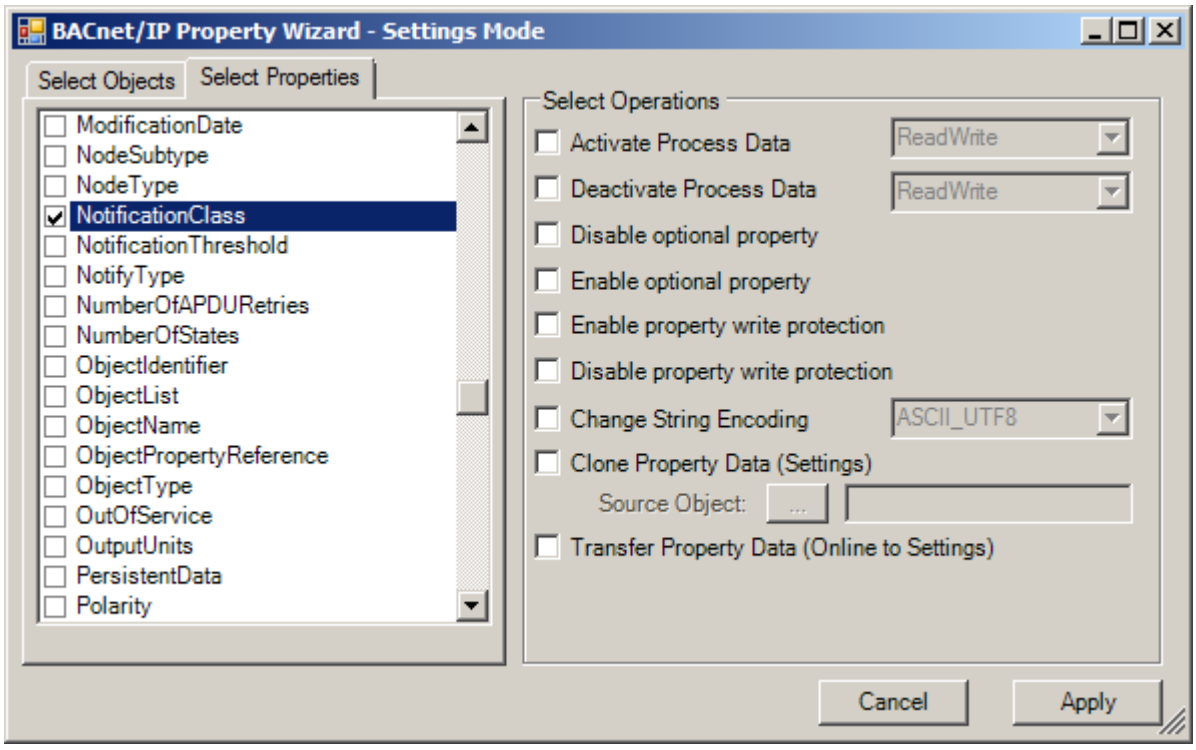
The context menu can also be used to jump directly to a selected object in the System Manager tree ("Goto Object"). The button "Select Properties" can be used to select which properties are displayed in the object overview. The Property Transfer wizard can be used to display differences between Settings and Online data. Data can be copied from Online to Settings via a context menu. This feature is also available via the Property wizard. In large configurations, the Transfer wizard may take some time to open.



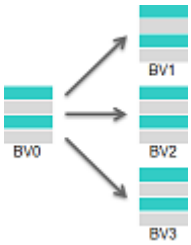
Property wizard

The Property wizard enables activation/deactivation of process data, activation/deactivation of optional properties and activation/deactivation of the write protection for several properties of several objects. This makes it very easy to deactivate Intrinsic Reporting (alarms) for all BinaryValue objects, for example, by deactivating the corresponding optional properties. The Property wizard is available in an Online version and a Settings version, depending on which object overview was used to activate the wizard. In Online mode all values are manipulated "online"; in Settings mode "Offline". In Online mode not all operations are available.

First, the objects ("Select Objects" tab) and the properties ("Select Properties" tab) for which the actions should be executed have to be selected. Click on "Apply" to execute the selected operations.



The function "Change String Encoding" can be used to change the character set for multiple BACnet properties. The following table explains the difference between the functions "Clone" and "Transfer".

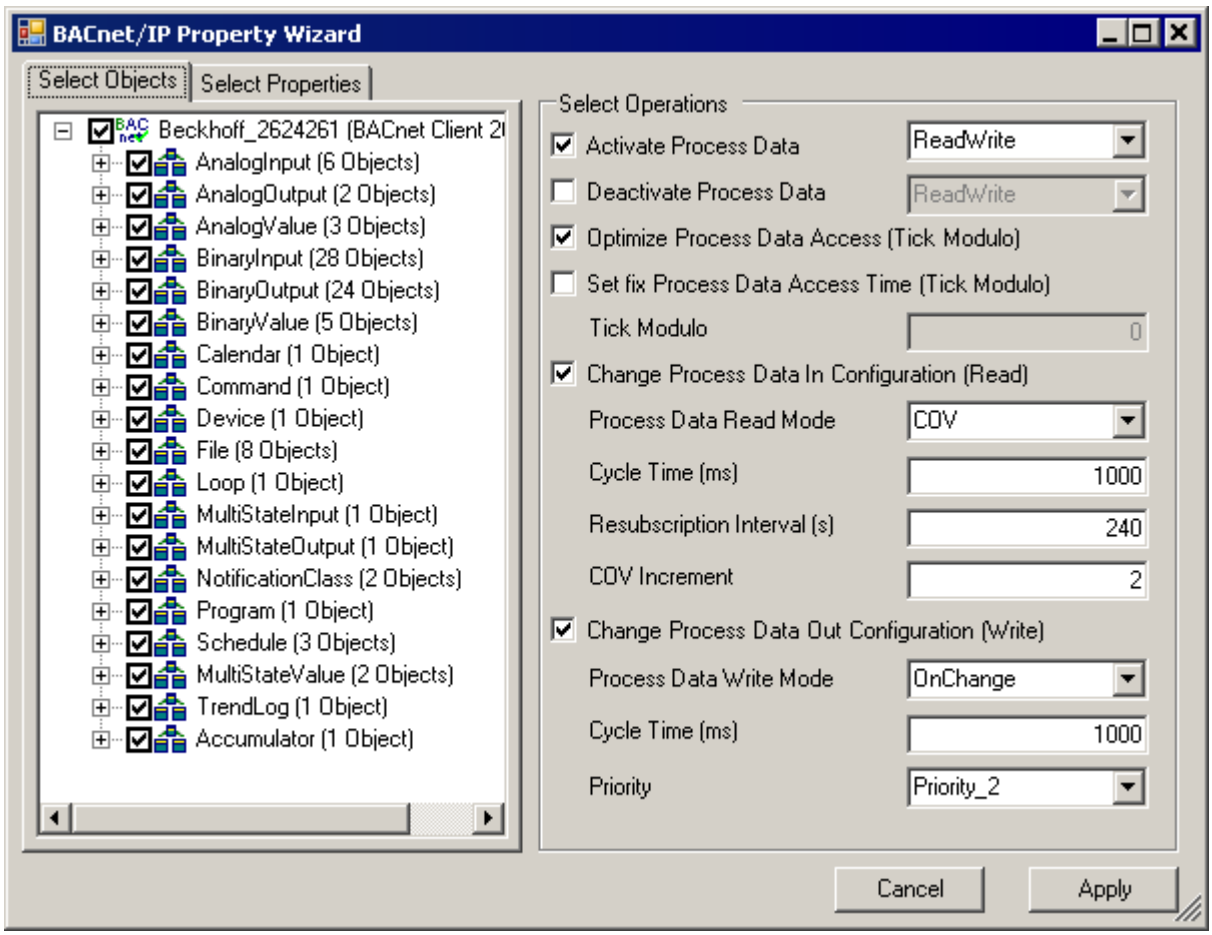
Clone 	During "cloning", the property values of a source object are transferred to several target objects. This function can be used to change the NotificationClass property of several BACnet objects simultaneously. To this end, the new value is written manually to the BACnet object and then selected as source object. Another example is retrospective activation of a smoothing filter for all AnalogInput objects. Since further operations can run in parallel with copying of the Property value, the property AvgFilterCycles can be enabled ("Enable optional Property") and at the same time initialized with a preconfigured value from the source object.
Transfer 	During "Transfer" the Property values of an object (source and target object are the same) are transferred. Either from Settings to Online (Online mode), or from Online to Settings (Settings mode). This function can be used to transfer modified online data into the configuration (.tsm). The values of all selected properties are copied.

Process Data wizard

The process data wizard enables activation/deactivation of process data, activation/deactivation of optional properties and activation/deactivation of the write protection for several properties of several objects. This makes it very easy to deactivate *Intrinsic Reporting* (alarms) for all BinaryValue objects, for example, by deactivating the corresponding optional properties. The Property wizard is available in an Online version and a Settings version, depending on which object overview was used to activate the wizard. In Online mode all values are manipulated "online"; in Settings mode "Offline". In Online mode not all operations are available.

The Process Data wizard for BACnet clients enables the configuration of process data-specific settings of client objects. Parameters that can be configured include the data processing mode (COV etc.), cycle times, COV increment etc.

For very large BACnet client configurations (with many COV process data) we recommend executing the function "Optimize Process Data Access" on all objects and properties. The corresponding TickModulo factors for the client interactions are then calculated automatically. The space available for modulo factors is determined automatically through the BACnet cycle time and the "cycle time" of the process data. With a BACnet cycle time of 10ms and a process data "cycle time" of 1000 ms the client interactions can be apportioned to 100 cycles. For large configuration the cycle times should be adjusted in order to produce a modulo factor that is as large as possible. The maximum number of cycles over which client interactions can be distributed is 255. The "cycle time" is also used in COV mode. In conjunction with the TickModulo the login (COV subscription) is sent in the assigned cycle. In systems with many COV process data, bursts can be reduced by distributing the COV subscriptions over several cycles. **From revision 12, the "Optimize Process Data Access" function is automatically executed when a configuration is activated and no longer has to be triggered via the wizard!**

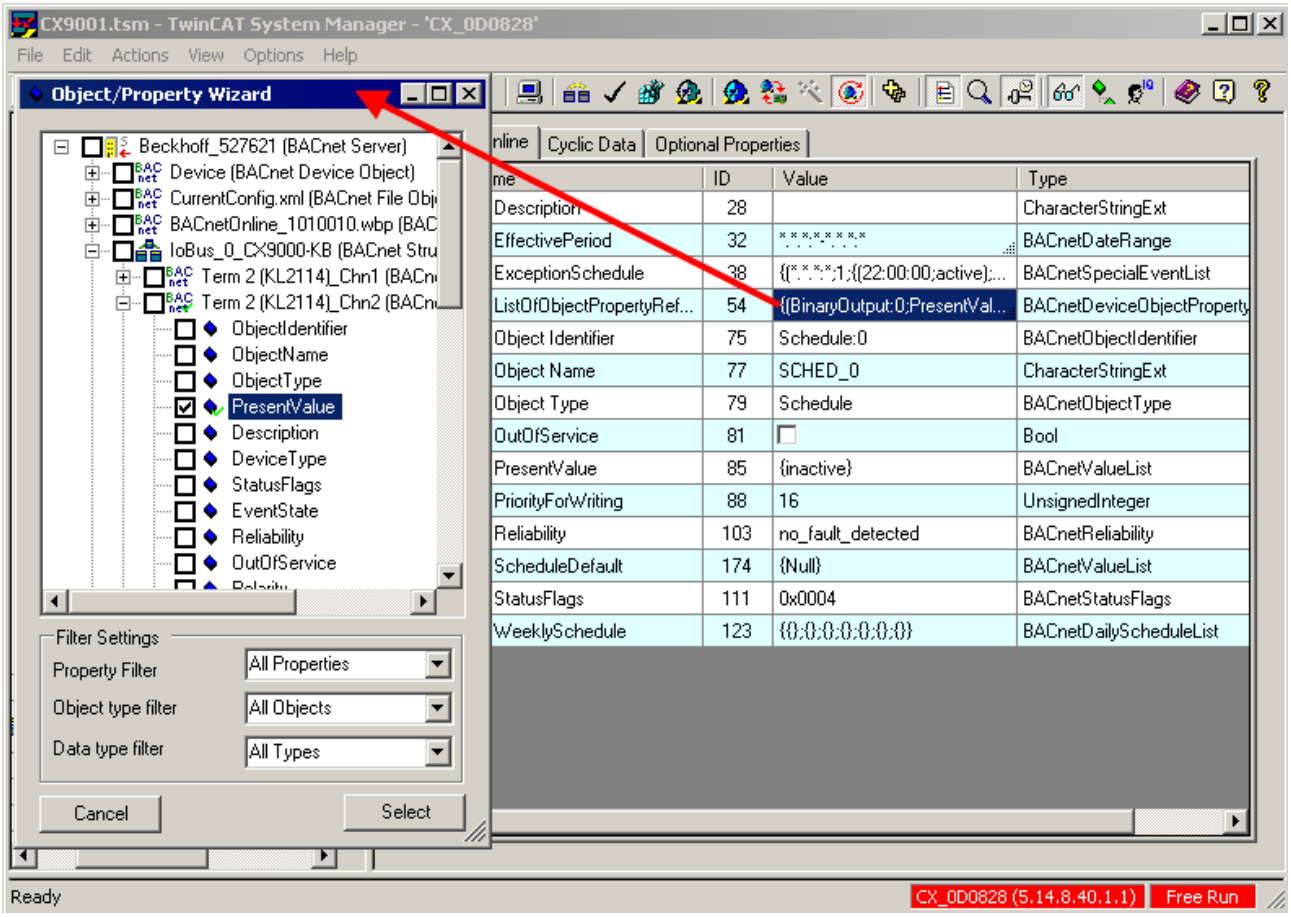


The process data wizard is very well suited to reconfigure the type of process data transfer for several or all BACnet remote objects.

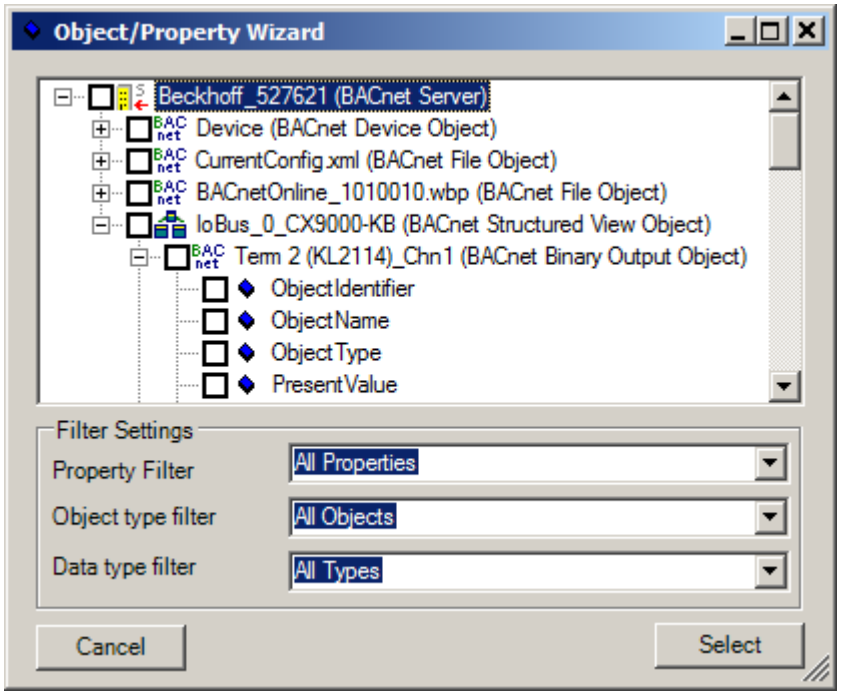
Property Reference wizard

In order to facilitate the handling of property and object references, it is possible to configure these properties by **double-clicking on** Value. This opens the wizard dialog. The availability of the wizard is indicated by a

triangle symbol in the bottom right corner of the Value field  . Examples are the ListOfObjectPropertyReferences property in the Schedule object or the LogDeviceObject property in the TrendLog object. The wizard dialog simplifies the selection of the corresponding properties and objects.

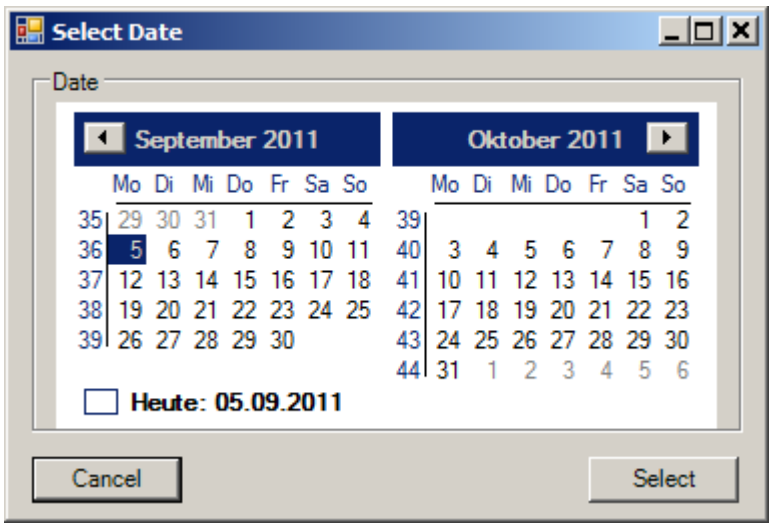


Filter options can be used to simplify the selection of several properties and objects. If the filter settings are changed, the options for selecting properties and objects are adjusted dynamically.



Calendar wizard

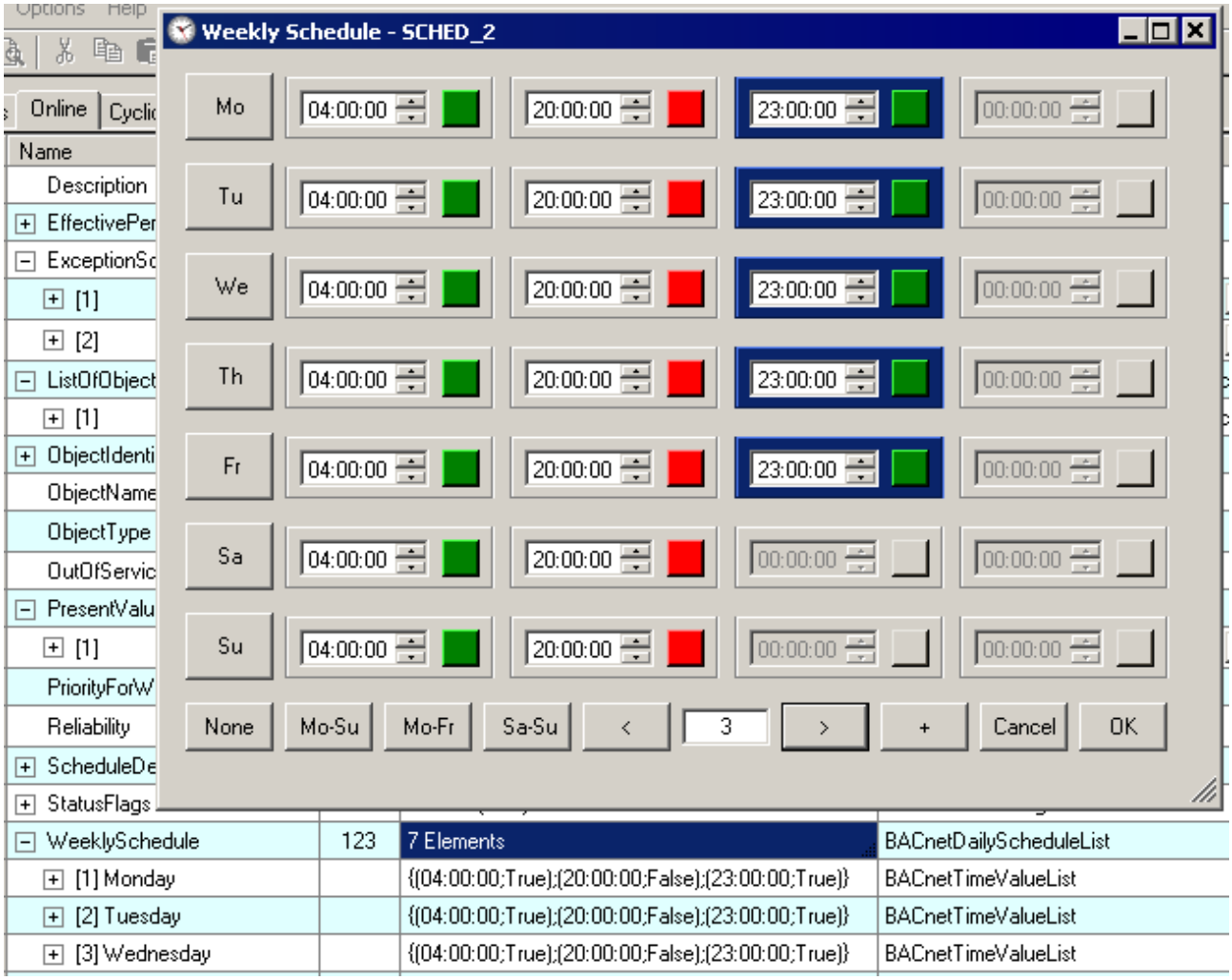
Properties with data type BACnetDate can be configured via the Calendar wizard. A day can be selected via a displayed calendar. The corresponding day of the week is determined automatically. For data with data type BACnetDateRange the wizard also enables configuration of date ranges.



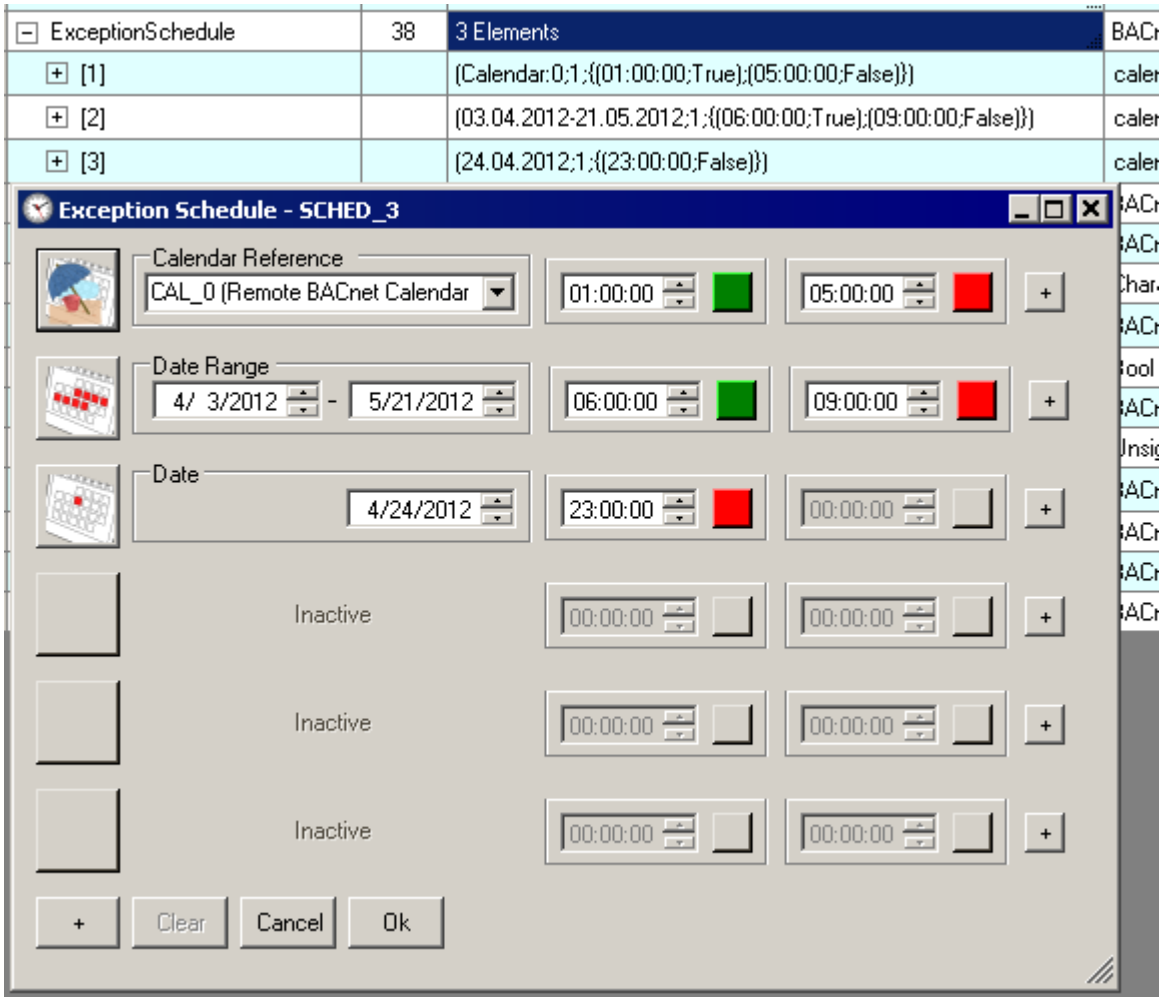
Schedule Object wizards

Wizards for the properties WeeklySchedule and ExceptionSchedule are available for configuring timer-based switching schedules. Currently these wizards are only available if a property of data type BinaryPV or Boolean is set as property reference. Wildcards ("*") are not allowed in the time entries.

The WeeklySchedule wizard supports editing of several day entries at the same time. Several weekdays can be selected by selecting "Mo-Su", "Mo-Fr" or "Sa-Su". Changes in an entry are then automatically transferred to all select entries. Other entries can be selected or deselected by clicking on them.

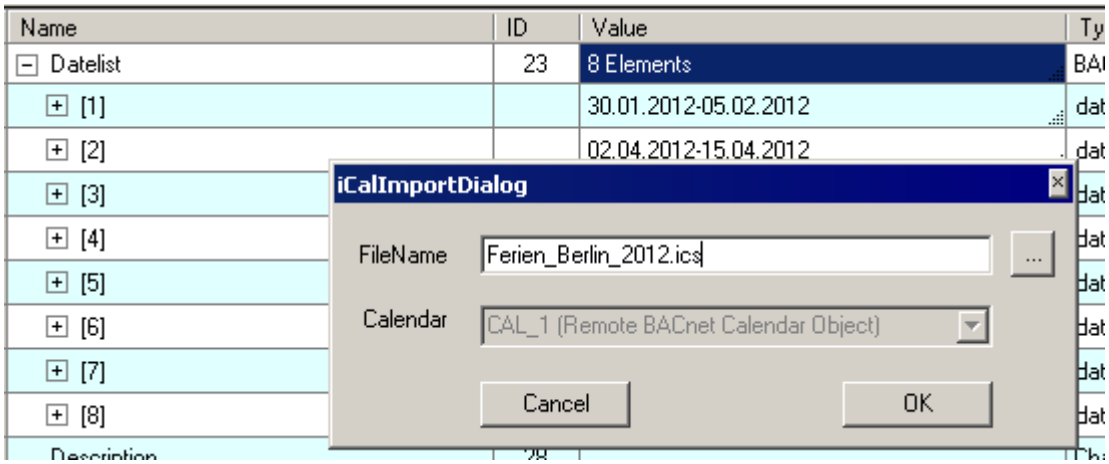


The ExceptionSchedule wizard can be used to edit exceptions. Four modes are available for selection: Calendar reference, Date, DateRange and WeekNDay. Further entries can be added by clicking on the "+" symbols.



Calendar/Schedule iCal/.ics Import

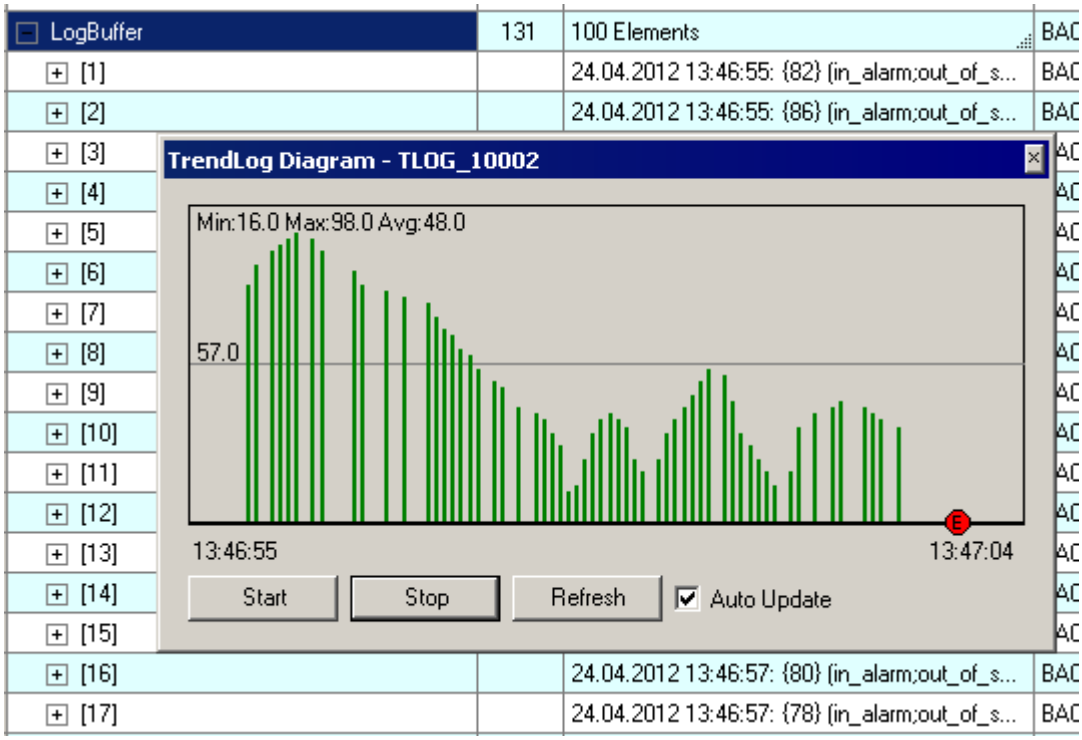
.ics import was implemented for efficient development of holiday calendars. Suitable .ics holiday files can be downloaded from the Internet and imported. The .ics import can be activated via a wizard of the DateList property in the Calendar objects. In addition, .ics can be selected via the "Import" button of the BACnet device, and the entries can either be stored in a new or an existing calendar.



The .ics import is also supported for the ExceptionSchedule and WeeklySchedule properties. Using the Windows Drag&Drop function, an .ics file can be dragged onto one of the properties. Corresponding Boolean entries are then imported for each contained period. .ics can be created, for example, using the Microsoft Outlook program.

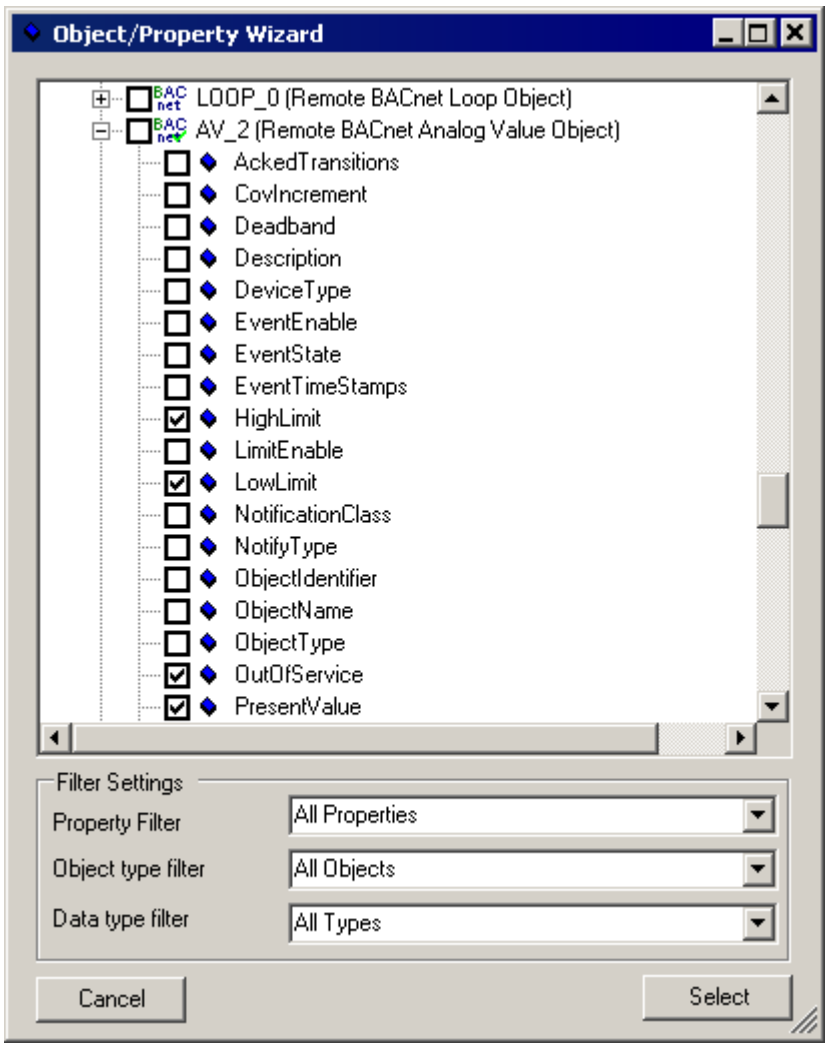
TrendLog-LogBuffer wizard

The property LogBuffer of the TrendLog objects contains the log entries. A diagram view for TrendLog objects was integrated as a wizard in order to provide a better overview. In addition to mean, min. and max. values, the diagram shows the entries of the LogBuffer as lines. Special entries (Start - B, Stop - E, Time synchronization - S) are shown on the x axis of the diagram. A special feature of this wizard is that several windows can be shown in parallel. In conjunction with the function "Auto Update" this enables certain features to be visualized in a BACnet configuration.

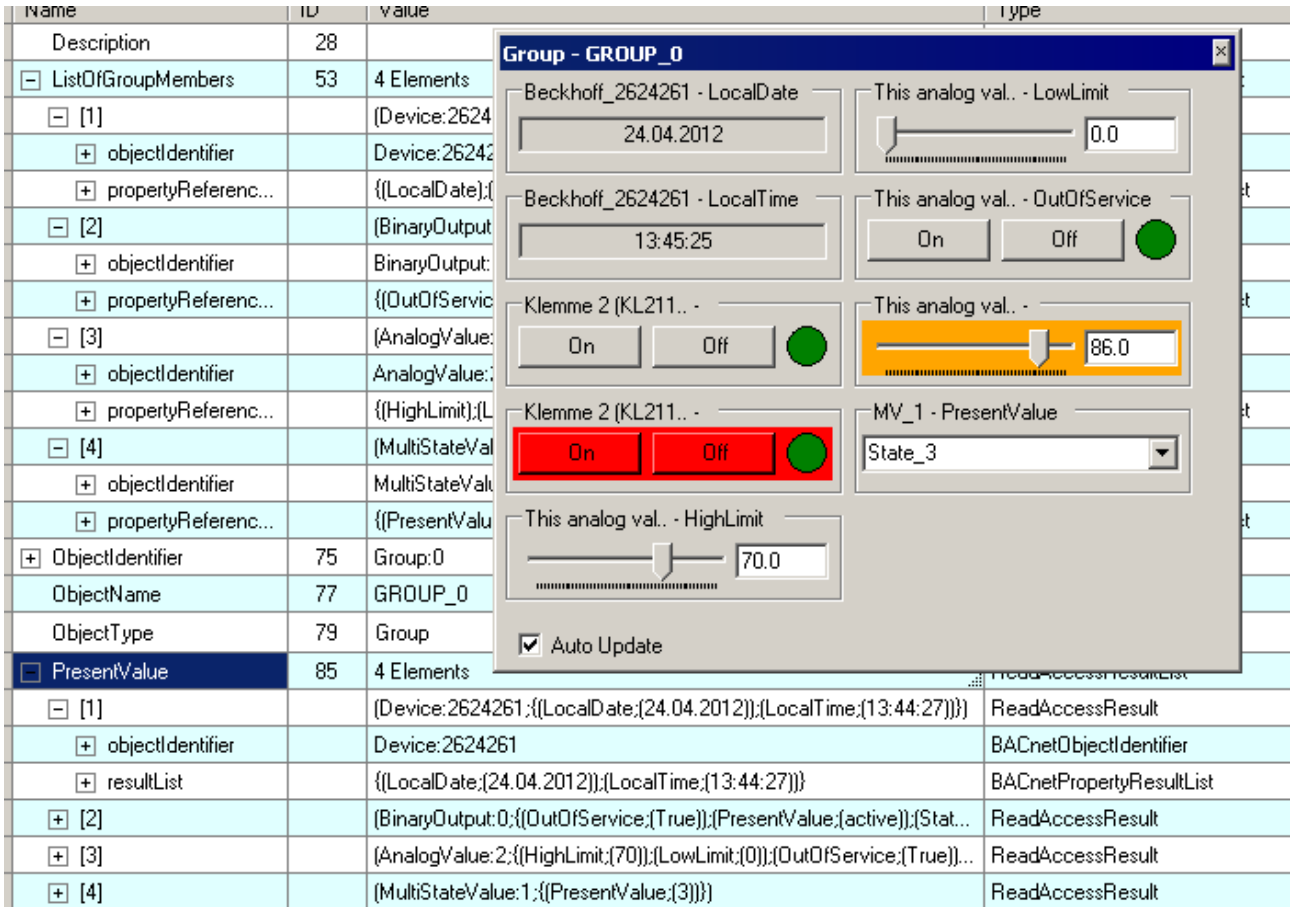


Group Object wizards

Group objects enable consolidation of several property values of multiple BACnet objects. The Property wizard described in section "Objects and Properties" can be used to configure the property ListOfGroupMembers of the Group objects.



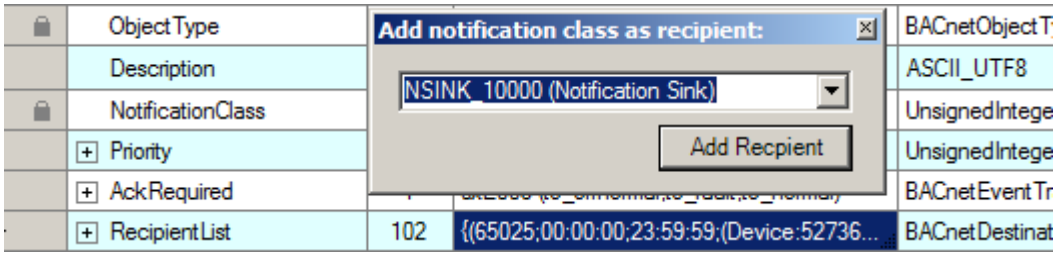
At runtime the property PresentValue of a Group object contains the values of the configured properties. The Group-PresentValue wizard displays the included property values in a window. In addition it is possible to change the property values. For properties of data type Boolean and BinaryPV the buttons "On" or "Off" and the current state are displayed. Properties of type Real are shown as sliders and value input fields. The states of MultiState* objects and enumerator properties (except BinaryPV) are visualized as a combo box. If the property StatusFlags of an object was configured in addition to a PresentValue, the *Offnormal* state becomes orange and the *Fault* state is highlighted in red.



Clicking on the property name of a GroupBox in the Group PresentValue-wizard while pressing the SHIFT key results in dynamic generation and starting of a TrendLog object on the associated BACnet client/server. In this way certain properties can conveniently monitored during commissioning. When the diagram window is closed, the dynamically generated TrendLog objects are deleted again.

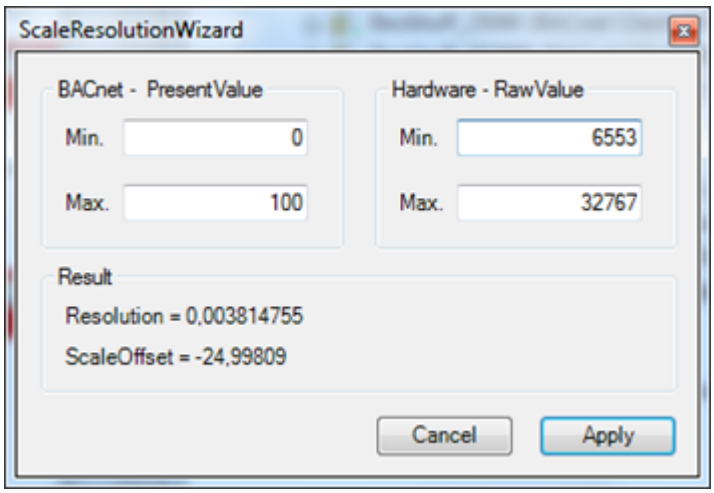
Notification Recipient Wizard

The Notification Recipient wizard facilitates entries in the RecipientList of a NotificationClass object. In this way it is possible to enter local NotificationSinks in remote BACnet stations. The required NotificationSink can be selected in the wizard.



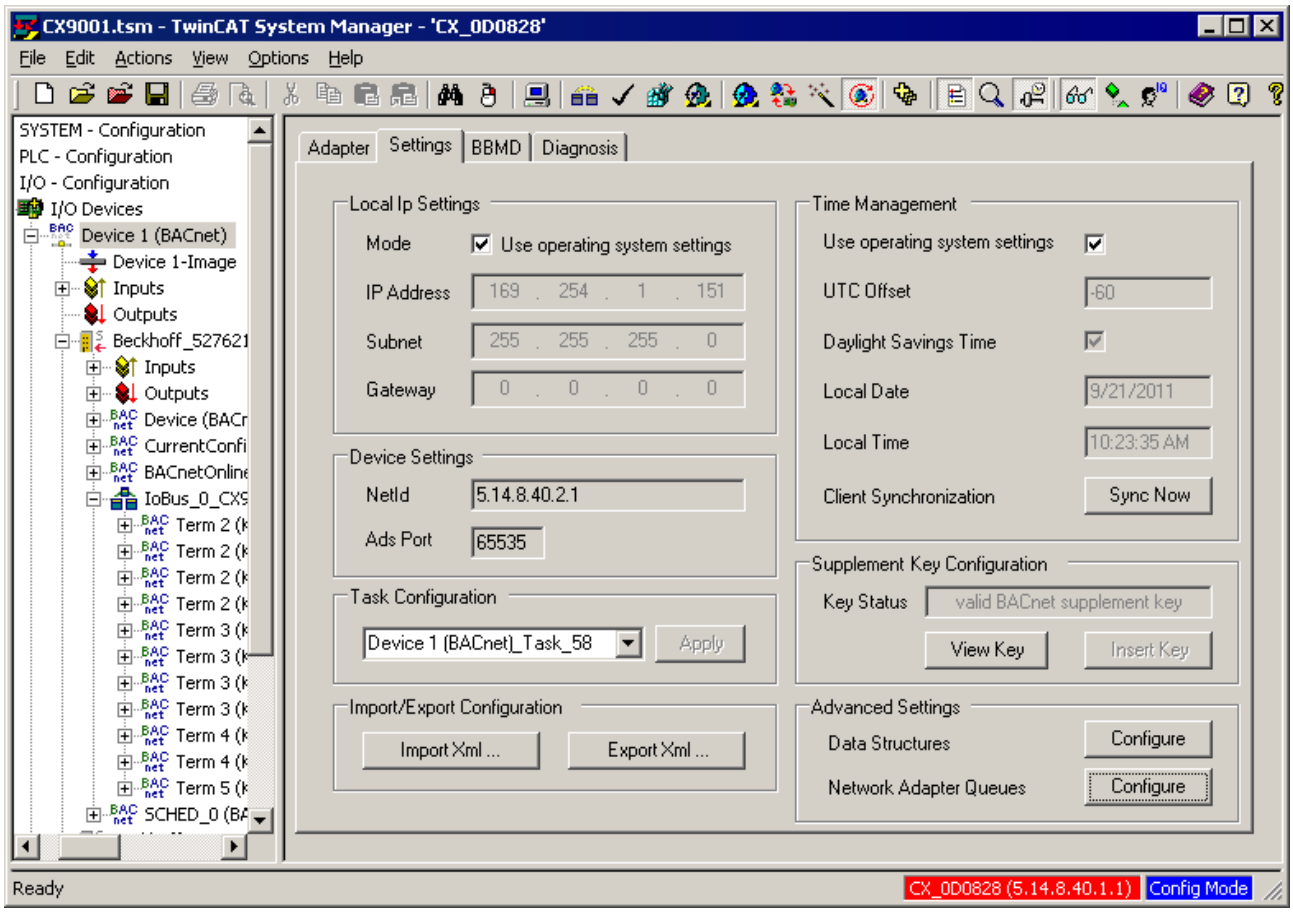
Scale/Resolution wizard

The ScaleResolution wizard can be used to calculate the properties ScaleOffset and Resolution of AnalogInput and AnalogOutput objects. Enter the range of values for BACnet and the hardware used (i.e. the range of values of the linked Raw* variable). Press "Apply" to activate the values automatically. In the Online or Settings part, depending on where the wizard was opened.



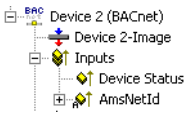
2.5 Process data

This section describes the process data variables of the individual TwinCAT BACnet/IP elements. In general, process data of a BACnet/IP device are linked with the PLC and a fieldbus via asynchronous mapping. This is necessary since BACnet has acyclic characteristics, and certain services may trigger many changes in other BACnet properties and therefore the sending of many BACnet Ethernet frames. Asynchronous mapping is used to prevent links with BACnet process data resulting in cycle time violations in the PLC or watchdog timeouts of the fieldbuses. It means that BACnet, PLC and fieldbus tasks run independent of each other.



Linking to a task takes place under the BACnet device via the Settings tab and "Task Configuration". This is where a task can be selected. The selection dialog shows all tasks with a port greater than 350. The bottom left of the screenshot shows a PLC task ("BACnet_Demo") that is linked asynchronously with the BACnet task, which by default is created with priority 58 when a BACnet object is added.

BACnet Device

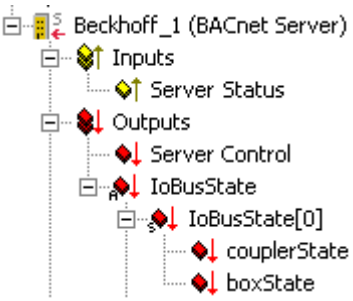


Under a BACnet device the process data variables are the Device Status, which includes the link status of the linked Ethernet adapter, and the state of the internal state machine. The device status is explained in the following table.

In addition the AmsNetID of the BACnet device can be linked with a PLC that accesses BACnet functions via ADS.

Bit	Description
0	**Ethernet link status:** Represents the current link status of the linked Ethernet adapter: • 0 - link is present. • 1 - no link is present.
1	**Gateway status:** represents the current status of the IP gateway: • 0 - gateway not available, or MAC address not resolved • 1 - gateway available and MAC address resolved
8 - 15	**Device State Machine**: Reflects the state of the internal BACnet device state machine: • Init (0) - Initial state • CheckIpAddress (1) - initialize the IP address (among others wait for DHCP - address) • CheckParameter (2) - Verification of gateway and BBMD parameters • GetGatewayMAC (3) - If a gateway has been configured: send out an ARP request • WaitForGatewayMAC (4) - Wait for gateway MAC address to be resolved (ARP reply) • Complete (8) - Initialization completed

BACnet Server



The ServerStatus contains among other things the current state of the internal server state machine. The server status from a BACnet view can be determined with the help of the property SystemStatus, which can be linked cyclically.

The state of an I/O link can be monitored centrally via the IoBusState process data variables. When linking a K-bus or a BK9000, for example, CouplerState or BoxState maps the state of several I/O modules. Fieldbuses that are linked with BACnet via I/O automapping can map Reliability automatically with the aid of these process data. Each fieldbus strand is allocated an IoBusNr, which is also a vendor-specific BACnet property. If the value of the process data variable couplerState or BoxState is not equal 0, all relevant (I/O) BACnet objects of the server with the corresponding IoBusNr are notified, and the Reliability is set to NO_SERVER or NO_OUTPUT. By default 8 IO buses are available.

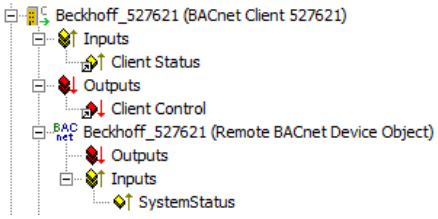
BACnet Server Status

Bit	Description
2	**WriteOnChange disabled**: WriteOnChange is disabled via the server control bit.
8 - 15	**BACnet Server Statemachine**: Reflects the state of the internal BACnet server state machine: • Init (0) - Initial state • Send I-Am Request (1) - An initial I-Am request is sent to the BACnet network. • Wait IO Timeout (2) - Waiting for PLC process data and object state machines to be released • Set System Status (3) - property SystemStatus of the Device object is set to Operational • Complete (4) - The BACnet server is ready for use and functional. • Backup (5) - A BACnet backup is executed.

BACnet Server Control

Bit	Description
0	**Backup PLC feedback Enable:** When backing up data generated by the PLC through a BACnet backup, this bit ensures that the PLC initially has write access to the backup-relevant files (configuration files). If this bit is set, the BACnet stack waits for bit 1 of the server control word being set before acknowledging the backup request. If this bit is not set, the BACnet stack omits this handshake procedure. Further information on backup and restore can be found in the corresponding chapter of this documentation.
1	**Backup PLC feedback Complete:** Once the backup-relevant PLC data were backed up, this bit instructs the BACnet stack to continue with the backup.
2	**Disable WriteOnChange**: Disables writing of cyclic process data. Can be used by the PLC to reset the writing process data without manipulating BACnet
3	**Elapse IO startup timeout**: At system startup, process data in the BACnet stack are only processed once the IO startup time has elapsed. This is configured as advanced parameter and is set to 2000 ms by default. It enables the PLC to prepare writing process data for BACnet before potential error messages are sent to the BACnet network. For faster startup, the PLC can reduce this time by setting this bit to 1. TcBACnet.lib uses this functionality automatically.

BACnet Client



In addition to monitoring the client status, the SystemStatus of the device object of the client should always be activated as process data. Since errors in the client communication are only shown in the client status if errors in the communication occur, regular communication should be ensured. To this end it is advisable to read or poll the property SystemStatus cyclically (with a suitably low cycle time).

The process data SystemStatus is a enumeration with the name BACnetDeviceStatus and is defined in the BACnet standard. BACnet makes provision for the following values:

- operational (0)
- operational-read-only (1)
- download-required (2)
- download-in-progress (3)
- non-operational (4)
- backup-in-progress (5)
- ...

BACnet Client Status

Bit	Description
0	**Client TimeOut:** A timeout has occurred in a BACnet service request.
1	**Client Service Failure:** A service with the configured client could not be carried out successfully. This error bit is set if: • A client cannot be reached. • A COV subscription could not be completed successfully. • A WriteProperty/ReadProperty request had a negative result.
2	**Client WriteOnChange State**: Activation control for WriteOnChange, which is specified under client control. • 0 - WriteOnChange is active. If the client process data change, corresponding BACnet network services are triggered. • 1 - WriteOnChange is disabled. Process data can be changed without triggering BACnet network services.

Bit	Description
3	**Client COV Subscription Control State**: Displays the state of the COV subscription control. If the **Client COV Subscription Control** bit is set, this bit is set to 1 when the COV subscription data structures were reset. If the **Client COV Subscription Control** bit is not set, the state is 0. This bit does not indicate whether all COV subscriptions were completed successfully, it only indicates that all COV subscription will be executed again in future.
4	**Client Force Output Update State**: Force Output Update is active.
8 - 15	**Client Remote State**: Reflects the state of the internal client state machine: • Init (0) - initial state • Connecting (1) - Sending out a Who-Is request for the client ID • WaitingForConnection (2) - Waiting for I-Am and IP address of the client. The system waits 10 seconds for a response, then the WhoIs request is repeated (connecting). • RequestDeviceInformation (3) - Reading of the properties ApduTimeout and NumberOfApduRetries, if segmentation is supported additionally MaxSegmentsAccepted, ApduSegmentTimeOut • Complete (4) - Client completely initialized. Interaction with the BACnet client can take place.

BACnet Client Control

Bit	Description
0	**Client TimeOut Reset:** Reset the Client-TimeOut bit in the client status.
1	**Client Service Failure Reset:** Reset the service failure bit in the client status.
2	**Client WriteOnChange Control:** Activation control for WriteOnChange. WriteOnChange is disabled if the corresponding bit is set in client status. • 0 - WriteOnChange is active. If the client process data change, corresponding BACnet network services are triggered. • 1 - WriteOnChange is disabled. Process data can be changed without triggering BACnet network services.
3	**Client COV Subscription Control:** Resets the status of the COV subscriptions and triggers the new registration of COV subscriptions. The internal data structures are reset once (which triggers the COV subscriptions) if a change of value 0 -> 1 is detected.
4	**Client Force Output Update State:** When the value changes from 0 to 1, at the next possible point in time all writing process data on a BACnet client are updated. This feature can be used to ensure that values for the whole cross communication are applied on a remote station, without overwriting by a different device in the meantime.

Notification Sink

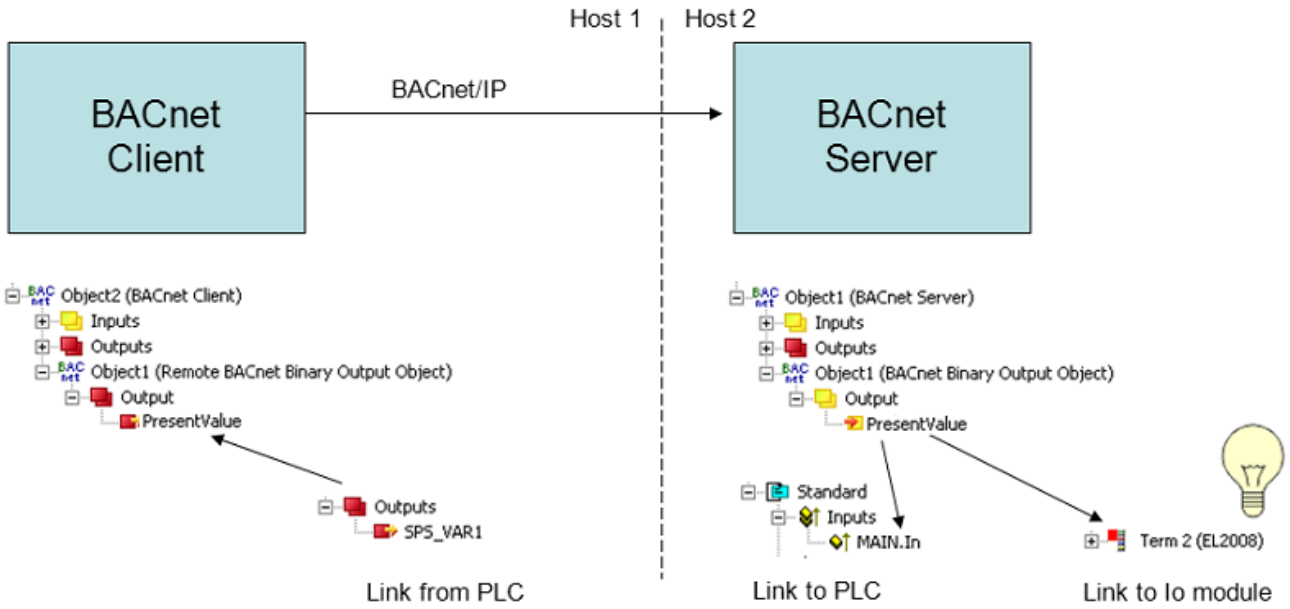
The TwinCAT BACnet/IP notification sink enables the receipt of COV and event notifications. The number of received notifications can be determined via the process data CovNotifications and EventNotifications. Notifications can be deleted and read via the ADS interface of the notification sink.

BACnet objects - properties as process data

Properties with non-complex data types (Signed Integer, Unsigned Integer, BitField, Boolean, Real, Enumerated) can be configured as cyclic process data. In the property process data a distinction has to be made between objects under a server (local objects) and objects under a client (remote objects).

- The properties of local objects are configured under a BACnet server. The dialog "*Cyclic Data*" can be used to configure which properties are linked as process data. Connectable properties appear under an object as process data variables and can be linked with variables of a PLC or with other I/O devices (e.g. EtherCAT terminals or K-bus terminals). Properties can be activated as a reading or writing process data:
  - Output data ("*PLC/IO -> BACnet*") are data that are copied from a PLC into a BACnet object. They are shown as red variables.
  - Input data ("*BACnet -> PLC/IO*"), shown in yellow, are copied from BACnet to the PLC. These include the StatusFlags of an object, which can be processed in the PLC.

For *remote* objects, the properties of a remote device can also be made available as cyclic process data. The data of the properties can either be transferred acyclically via COV (Change of Value) or cyclically at a fixed interval. In remote objects data that are consumed by a PLC are shown in yellow, produced data in red.



The schematic diagram illustrates the application of process data in client and server objects. The example shows a BinaryOutput object created on a server. The property PresentValue is linked as a process data variable. The PresentValue implemented as *RawInBinaryBoolValue* can be used in a PLC on the server in order to influence a functionality in a PLC program, for example, or it can be linked with an EtherCAT/K-bus terminal to activate lighting, for example. In this case, *RawInBinaryBoolValue* is a yellow PLC input variable on the server side. On a second device the object is configured as a remote object. In this example the value of PresentValue of the server object is to be changed with the aid of a PLC on the client. To this end the PresentValue is mapped as a red output variable on the client and can be linked with the PLC.

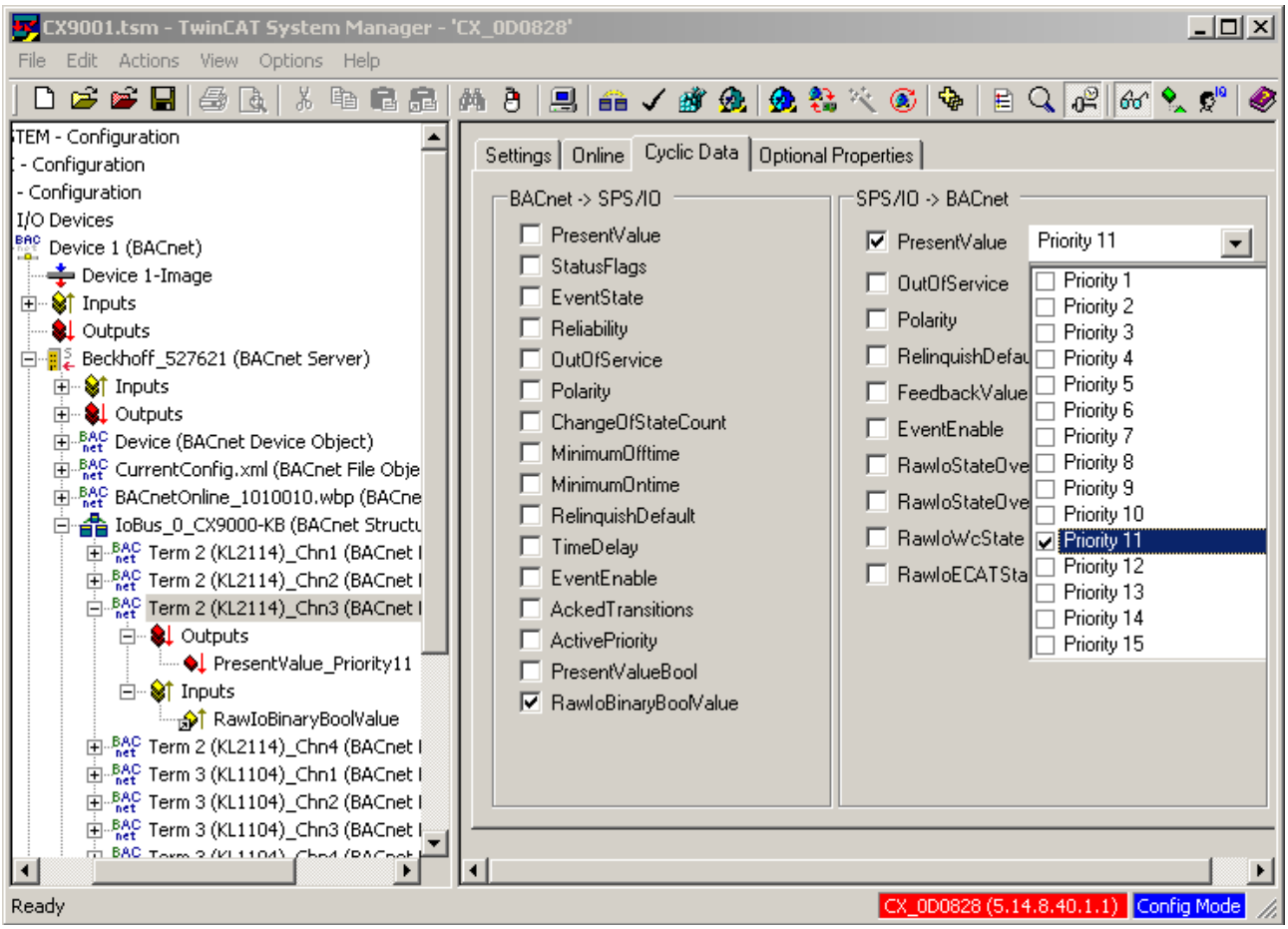
The following table shows the **relationship between BACnet data types** and PLC data types:

BACnet data type	PLC data type
Boolean	BOOL
Unsigned Integer	UDINT
Signed Integer	DINT
Real	REAL
Double	not used
Bit string (StatusFlags, EventEnable, LimitEnable etc.)	WORD (For bit strings it should be noted that the lowest-order byte is used to display the number of unused bits of the highest-order byte. An example is 0xE005 of an EventEnable property: 5 indicates that 3 bits are used in the highest-order byte [to_offnormal, to_fault, to_normal]. 0xE0 indicates that all 3 bits are TRUE. If a bit string property is written cyclically from the PLC to BACnet, these bits have to be coded accordingly. Valid values can be found in the Online tab of the BACnet objects.)
Enumerated (EventState, Reliability, Polarity, BinaryPV etc.)	WORD

Process data of local objects

For local objects, (writing) process data are generally written once on system startup and subsequently when a change occurs. This means that potentially the value written by the PLC and the value of the described property can differ. Among other things because e.g. via BACnet or ADS another 'actor' has changed the value. For writing process data it is therefore advisable to implement feedback in the PLC. This behavior was modified compared with the implementation in Revision 6, where all properties were written in each cycle. An exception are the Raw* properties, which are written in each cycle.

Commandable properties



BACnet offers the option to control certain (commandable) properties via a priority mechanism. In this case, the value of the property *PresentValue* of the objects BO, BO, AO, AV, MV and MO is controlled via the entries of the property *PriorityArray*. The different entries correspond to different importance and can be written by different 'actors' in the BACnet network. With TwinCAT BACnet/IP, the PLC can write entries in the *PriorityArray*. To this end, a priority level can be selected in the tab "Cyclic Data" for writing the property *PresentValue* (the values are then copied internally to the property *PriorityArray*).

To activate a lower priority level, a priority level with the value NULL can be written (enabled).

- From the PLC this is possible for BinaryOuput/Value objects. If a value not equal 0/1 is written to the PresentValue of a linked priority level of a BinaryValue or BinaryOutput object, the value ZERO is copied to the PriorityArray.
- For AnalogOutput/Value objects the value ZERO is assumed in the PriorityArray, if the special coding of REAL values NaN (not a number) is used as process data. For server objects NULL is also written if the REAL value of the process data variable is outside the Min/MaxPresValue range.
- In MultistateOuput/Value objects the value NULL is assumed in the PriorityArray, if 0 is written as process data. For server objects NULL is also assumed if the process data takes on a value greater than NumberOfStates.

For each BACnet object, several priority levels can be written simultaneously by the PLC.

RawIoProperties

Process data properties from this category are used for linking of BACnet objects with I/O modules and PLC variables. These properties do not directly represent properties that are visible in BACnet, but they act on standardized properties of the corresponding objects or are formed from them. The general rule for RawIo properties is that the BACnet property OutOfService takes effect. An object is then referred to as OutOfService, since the BACnet property OutOfService has the value TRUE.

Property	Description
RawIoSignedValue	Signed 16-bit value (INT) used for analog* objects. If an AnalogInput object is not OutOfService, the value of this process data variable forms the value of PresentValue. The value of RawIoSignedValue is converted to a REAL data type, the result is multiplied with the property Resolution and then added to the property ScaleOffset. For AnalogOutput objects the value of RawIoSignedValue is formed from PresentValue based on the inverse procedure. If the value of the PresentValue exceeds the value range of the data type INT (-32768 to 32767), RawIoSignedValue has the value 0. If an object is OutOfService, depending on the configuration of the properties FaultFallbackValue or OutOfServiceFallbackValue, the last value remains or the configured value is used.
RawIoUnsingedValue	Unsigned 16-bit value (UINT) that is used for analog* objects. If an AnalogInput object is not OutOfService, the value of this process data variable forms the value of PresentValue. The value of RawIoSignedValue is converted to a REAL data type, the result is multiplied with the property Resolution and then added to the property ScaleOffset. For AnalogOutput objects the value of RawIoSignedValue is formed from PresentValue based on the inverse procedure. If the value of the PresentValue exceeds the value range of the data type UINT (0 to 65535), RawIoSignedValue has the value 0. If an object is OutOfService, depending on the configuration of the properties FaultFallbackValue or OutOfServiceFallbackValue, the last value remains or the configured value is used.
RawIoBinaryEnumValue	Enumeration type (in the PLC processed as WORD) that is used for binary* objects. Applies if OutOfService is not TRUE. For **BinaryOutput** objects the value of the property RawIoBinaryEnumValue is formed as follows: • RawIoBinaryEnumValue is 0 if PresentValue is ACTIVE and the polarity is NORMAL, or if PresentValue is INACTIVE and the polarity is REVERSE. • RawIoBinaryEnumValue is 1 if PresentValue is INACTIVE and the polarity is NORMAL, or if PresentValue is ACTIVE and the polarity is REVERSE. If OutOfService is TRUE, for BinaryOutput objects the property RawIoBinaryEnumValue assumes the value of the property OutOfServiceFallbackValue if this is activated, otherwise the last valid value is maintained. For **BinaryInput** objects the PresentValue is formed from the property RawIoBinaryEnumValue. • PresentValue is Active if RawIoBinaryEnumValue is 0 and the polarity is NORMAL or if RawIoBinaryEnumValue is greater than 0 and the polarity is REVERSE. • PresentValue is INACTIVE if RawIoBinaryEnumValue is greater than 0 and the polarity is NORMAL or if RawIoBinaryEnumValue is 0 and the polarity is REVERSE. If OutOfService is TRUE, PresentValue always assumes the value of FaultFallbackValue if this is active, otherwise PresentValue assumes the last value when OutOfService was FALSE.
RawIoBinaryBoolValue	Binary value (BOOL) that is used for binary* objects. The property PresentValue of the binary* objects in BACnet is an enumeration type. The property RawIoBinaryBoolValue was introduced to optimize the handling of binary objects. It can be used instead of RawIoBinaryEnumValue. For **BinaryOutput** objects the value of the property RawIoBinaryBoolValue is formed as follows: • RawIoBinaryBoolValue is FALSE if PresentValue is ACTIVE and the polarity is NORMAL, or if PresentValue is INACTIVE and the polarity is REVERSE. • RawIoBinaryBoolValue is TRUE if PresentValue is INACTIVE and the polarity is NORMAL, or if PresentValue is ACTIVE and the polarity is REVERSE.

Property	Description
	If OutOfService is TRUE, for BinaryOutput objects the property RawIoBinaryEnumValue assumes the value of the property OutOfServiceFallbackValue if this is activated, otherwise the last valid value is maintained. For **BinaryInput** objects the PresentValue is formed from the property RawIoBinaryBoolValue. • PresentValue is ACTIVE if RawIoBinaryBoolValue is FALSE and the polarity is NORMAL, or if RawIoBinaryBoolValue is TRUE and the polarity is REVERSE. • PresentValue is INACTIVE if RawIoBinaryBoolValue is TRUE and the polarity is NORMAL, or if RawIoBinaryBoolValue is FALSE and polarity is REVERSE. If OutOfService is TRUE, PresentValue always assumes the value of FaultFallbackValue if this is active, otherwise PresentValue assumes the last value when OutOfService was FALSE.
RawIoStatus	Analog input modules can have a process data that indicates the state of the physical connection (e.g. wire break detection, range violation). If the property RawIoStatus (data type USINT) is active, it influences the property Reliability. The property Reliability is formed from RawIoStatus if the object is not OutOfService, no IoBus error is pending (see RawIoECATState, IoBusState) and no status override is active, (see RawIoStateOverride*) as follows: • If bit 0 is set, Reliability is UNDER_RANGE • Otherwise Reliability is OVER_RANGE, if bit 1 is set • Otherwise Reliability is UNRELIABLE_OTHER, if bit 6 is set
RawIoECATState	EtherCAT modules have an operation status that maps the current operating state. BACnet objects that are linked with EtherCAT modules can form their Reliability via this operation status. RawIoECATState enables unplugged modules (module plug) and partial bus failures to be detected and mapped to BACnet. The property Reliability is formed as follows from RawIoECATState if the object is not OutOfService: • For output objects (BO, AO, MO) the following applies: if the value of bit 0 - bit 3 does not equal 8 (OP), Reliability is NO_OUTPUT. • For input objects (BI, AI, MI) the following applies: if the value of bit 0 - bit 3 does not equal 8 (OP), Reliability is NO_SENSOR.
RawIoWcState	WcState monitoring is currently not supported.
RawIoStateOverride	Some I/O modules have a mechanical manual mode for overriding the fieldbus process data locally. Manual override can be mapped in BACnet via the property StatusFlags with the bit OVERRIDE. If the process data variable RawIoStateOverride is TRUE, in the allocated BACnet object the bit OVERRIDE is set in the property StatusFlags, otherwise not, when the BACnet object is not OutOfService.
RawIoStateOverrideInverted	Some I/O modules have a mechanical manual mode for overriding the fieldbus process data locally. Manual override can be mapped in BACnet via the property StatusFlags with the bit OVERRIDE. If the process data variable RawIoStateOverride is FALSE, in the allocated BACnet object the bit OVERRIDE is set in the property StatusFlags, otherwise not, when the BACnet object is not OutOfService. Examples for using RawIoStatusOverrideInverted can be found in modules KM2642 and KM4602.
RawIoAccumulatorUnsignedValue	RawIoAccumulatorUnsignedValue enables acquisition of count values via a 16-bit process data variable. This can be linked to a KL5101, for example, or counting pulses can be specified by the PLC. The BACnet stack performs a rotation detection at 32767. In other words, two pulses are counted (after 0 and after 1) during the transition from 32767 to 1. To protect against data loss when the PLC is stopped, no operations are carried out when the current value is 0. Only with a new value of 1 would accumulation continue accordingly. The AccumulatorIntegrationMode property can be used to activate an

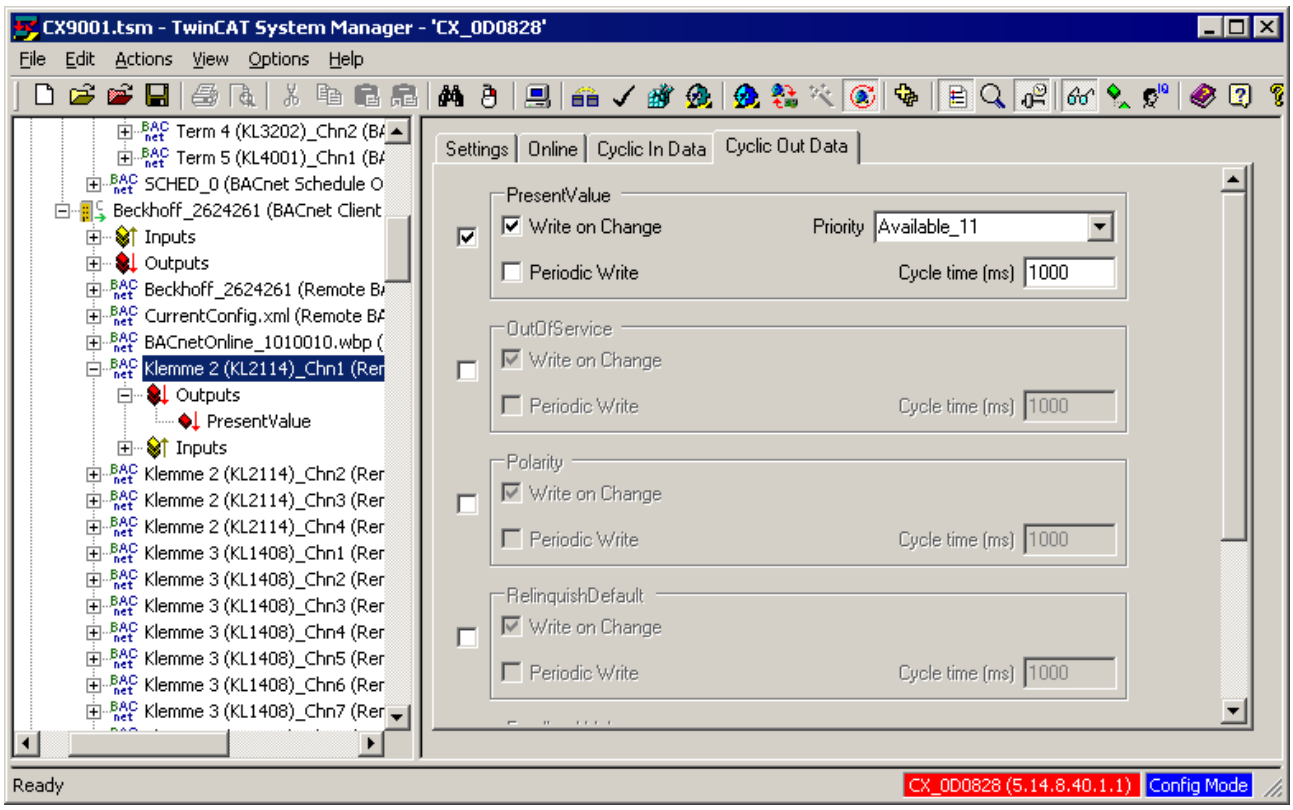
Property	Description
	integrating mode. If this property is TRUE, it is not modifications of the variable RawIoAccumulatorUnsignedValue that are accumulated, but the product of RawIoAccumulatorUnsignedValue * elapsed time (in ms). For example, if the value is 1, 1000 is accumulated after one second. This function enables acquisition of air volumes based on a current air volume.
RawIoPulseConverterUnsignedValue	RawIoPulseConverterUnsignedValue has the same mode of operation as RawIoAccumulatorUnsignedValue. Integration mode is not supported for the PulseConverter object.

Special data versions of PresentValue

Property	Description
PresentValueBool	**Schedule object**: If the PresentValue of a schedule object of the BACnet data type is Boolean, the process data property PresentValueBool reflects the current value of the property PresentValue. **Binary* objects**: In BACnet the property PresentValue of binary* objects is of data type BinaryPV and therefore an enumeration. The corresponding PresentValue process data is therefore of data type WORD, since enumerations are processed in the PLC as 16-bit values. As an optimized version the process data PresentValueBool of data type BOOL can be used. PresentValueBool is TRUE if PresentValue is ACTIVE and FALSE if PresentValue is INACTIVE.
PresentValueEnumerated	If the PresentValue of a schedule object is a BACnet enumeration, the process data property PresentValueEnumerated reflects the current value of the property PresentValue.
PresentValueReal	If the PresentValue of a schedule object is of BACnet data type Real, the process data property PresentValueBool reflects the current value of the property PresentValue.
PresentValueSigned	If the PresentValue of a schedule object is of BACnet data type SignedInteger, the process data property PresentValueBool reflects the current value of the property PresentValue.
PresentValueUnsigned	If the PresentValue of a schedule object is of BACnet data type UnsignedInteger, the process data property PresentValueBool reflects the current value of the property PresentValue.

Remote process data

For remote objects, data have to be transferred to the remote BACnet server (with the aid of the BACnet WriteProperty service). TwinCAT BACnet/IP offers two data transfer options. Data can be written "*On Change*", i.e. when changes occur. During processing of the process data within TwinCAT BACnet/IP, write access to the corresponding BACnet property is triggered if a change in the process data is detected. This procedure is equivalent to the processing of local objects. As a second option the data can be written cyclically. If "*Write On Change*" is active the data are written once when a change occurs. If another BACnet device subsequently overrides the value, the value of the client PLC may differ from the property value. It may be advisable to feed the BACnet properties back as input variable. With "Write On Change" data are transferred at most with the set cycle time. The dialog *"Cyclic Out Data*" can be used to configure writing process data for remote objects:



For commandable properties a priority can be selected for the write access. This priority is then transferred in the WriteProperty request.

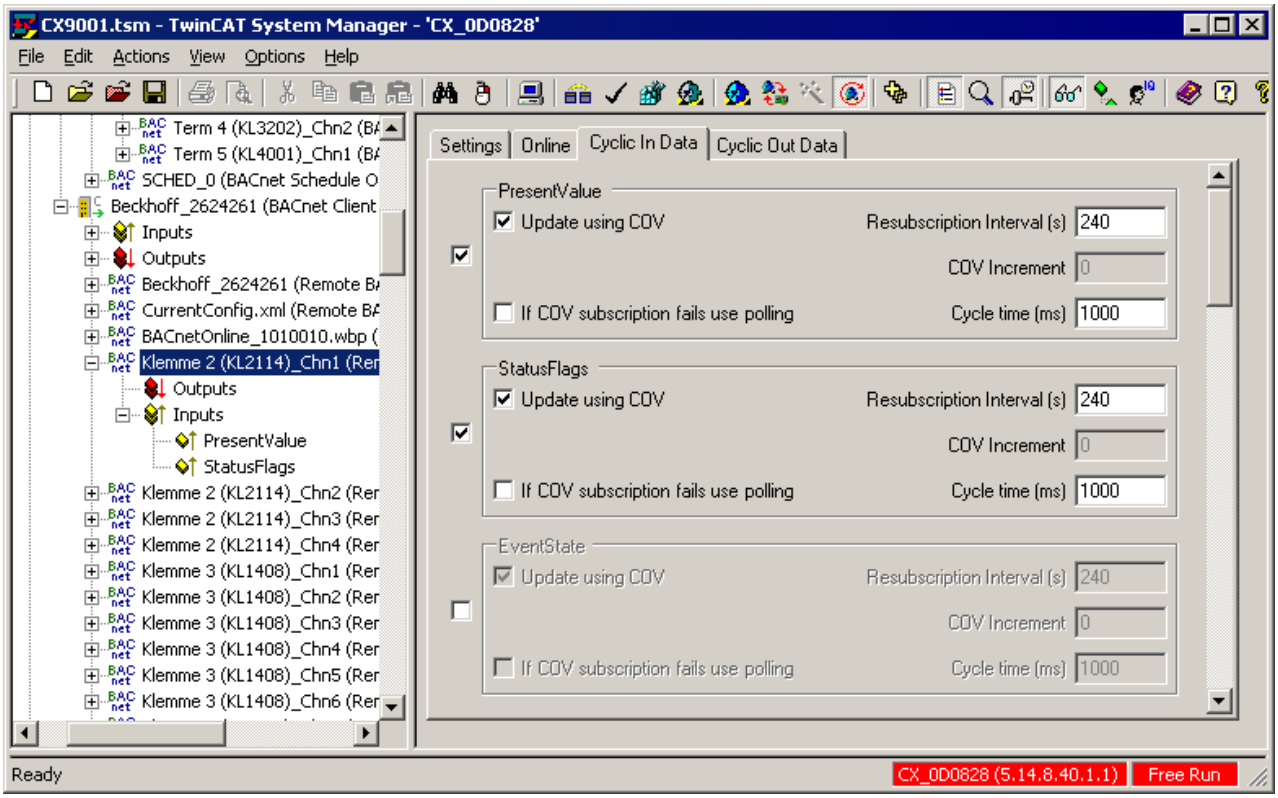
A special case for writing remote process data is an automatic locking when the PLC stops. If Bit3 (ElapseIoActiveTime) in the BACnet server control word changes from 1 to 0, the *DisableWriteOnChange* function is automatically activated for each BACnet client. Writing of remote process data can only recommence once this bit has been set to 1 again. This protective logic is required to prevent flooding of the BACnet network with "0" if the PLC stops.

Reading process data of remote BACnet objects are configured via the tab "*Cyclic In Data*". This dialog can be used to select as a basic configuration whether data are transferred when changes occur (COV) or cyclically. In addition, the interval for new subscriptions to change messages (*Resubscription interval*) can be specified. For properties with floating-point values, the COV increment can be used to specify the "resolution" at which data are to be transferred. *CycleTime* defines the period for reading cyclic data.

BACnet supports two services for transferring change messages. The service COV-P enables transfer of changes of all properties. In addition, a COV increment per *Subscription* can be specified for properties with floating point values. The service COV only allows the transfer of specially specified properties for selected objects (e.g. *PresentValue*, *StatusFlags*). COV is supported by a wide range of manufacturers; COV-P only by a few. To achieve maximum compatibility, the COV service should be used whenever possible.

COV-P is used if:

- a property is not supported by COV (e.g. *EventState*)
- a COV increment other than 0.0 was entered



The TwinCAT BACnet/IP stack in Revision 12 supports extensive further modes for process data transfer of remote BACnet objects. These transfer modes cannot be configured in the "*Cyclic In Data*" tab, since they always affect several BACnet properties at the same time. Some of the modes shown in the following table can only be activated via PLC automapping or with the Process Data wizard.

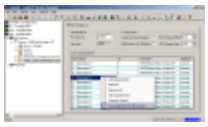
Requirements

Transmission type	Description
Unconfirmed COV	This mode is activated in the "*Cyclic In Data*" dialog. Property values are sent as unconfirmed change messages (*UnconfirmedCOV*). In other words, the sender does not ensure that the data is actually delivered. In order to be able to use this mode, the remote station must support the service COV or COV-P for the activated property. This mode can be used in reliable networks.
Confirmed COV	In this mode change messages are delivered confirmed (*ConfirmedCOV*). After submitting a change message, the sender waits for the configured *APDU timeout* (device object). If the change message is not confirmed, the message is sent again until the number of APDURetries is reached (device object). In order to be able to use this mode, the remote station must support the service COV or COV-P for the activated property. This mode should be use in networks with temporary overload, or in cases where confirmation is required that change messages were delivered. Note that the maximum number of simultaneous changes that can be transferred to the remote station is 255.
Unsolicited COV	In modes *Unconfirmed COV* and *Confirmed COV*, changes are subscribed to via the *SubscribeCOV* service. The remote station stores the subscription in a table, which often has a limited size. In cases where many devices in a network register for changes of a particular property (e.g. weather data), it may be sensible to use the service *Unsolicited COV*. In this case, no registration is required. The source device automatically sends changes as broadcast (message to all devices in the network). If this mode is used for client process data, the corresponding property is read once via *ReadProperty* when TwinCAT starts (initial value). Subsequently, the corresponding broadcast telegrams are evaluated. Details about the configuration of *Unsolicited COV* as client and server can be found at the end of this chapter.

Transmission type	Description
ReadProperty (Polling)	This mode is activated in the "*Cyclic In Data*" dialog. Property values are transferred cyclically via *ReadProperty*.
ReadPropertyMultiple per Object (RPM-O)	In BACnet, the service *ReadPropertyMultiple* can be used to transfer several property values in one message. RPM-O mode can be used in cases where many property data have to be transferred as process data from a remote BACnet device. In this case, cyclic read access operations to several properties are consolidated in one message per object.
ReadPropertyMultiple per Client (RPM-C)	In BACnet, the service *ReadPropertyMultiple* can be used to transfer several property values in one message. RPM-C mode can be used in cases where many property data have to be transferred as process data from a remote BACnet device. In contrast to RPM-O, in RPM-C all property values of all objects are consolidated to one message. The BACnet stack automatically optimizes the request size to avoid segmentation at BACnet level (messages are limited to approx. < 1400 bytes per message). This transmission mode can be used for slow networks where a lot of property data has to be transmitted.

Unsolicited COV - client configuration

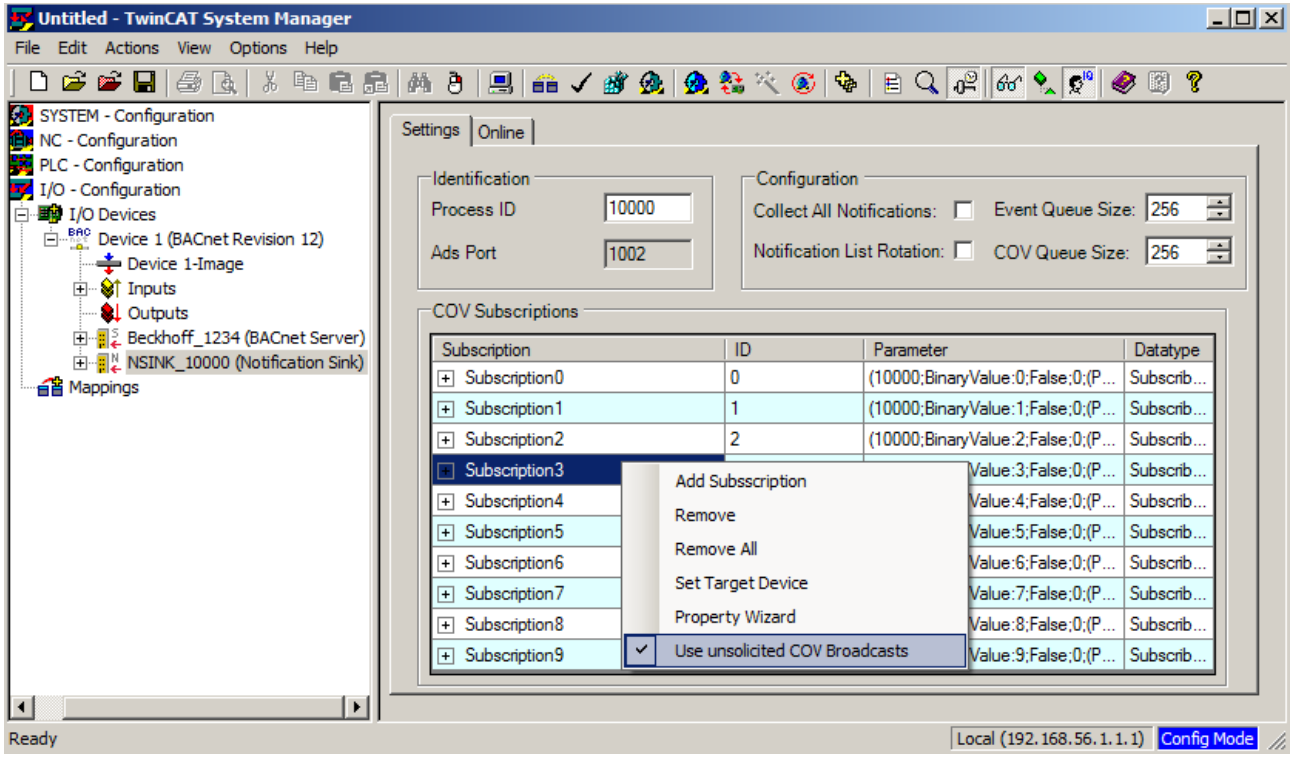
With the *Unsolicited COV* feature, change messages are sent without the need to register in the BACnet network (*COV subscription*). This function is used by room control units or weather stations, for example. The Unsolicited COV mode can be used to 'consume' these change messages as client process data. This mode can be selected via the Property wizard by selecting all required objects of the BACnet client, selecting properties and then activating the new mode in the field "Change Process Data In Configuration (Read)". In this mode an initial value is read once on startup via *ReadProperty*. Subsequently the change messages are processed.



Unsolicited COV - server configuration

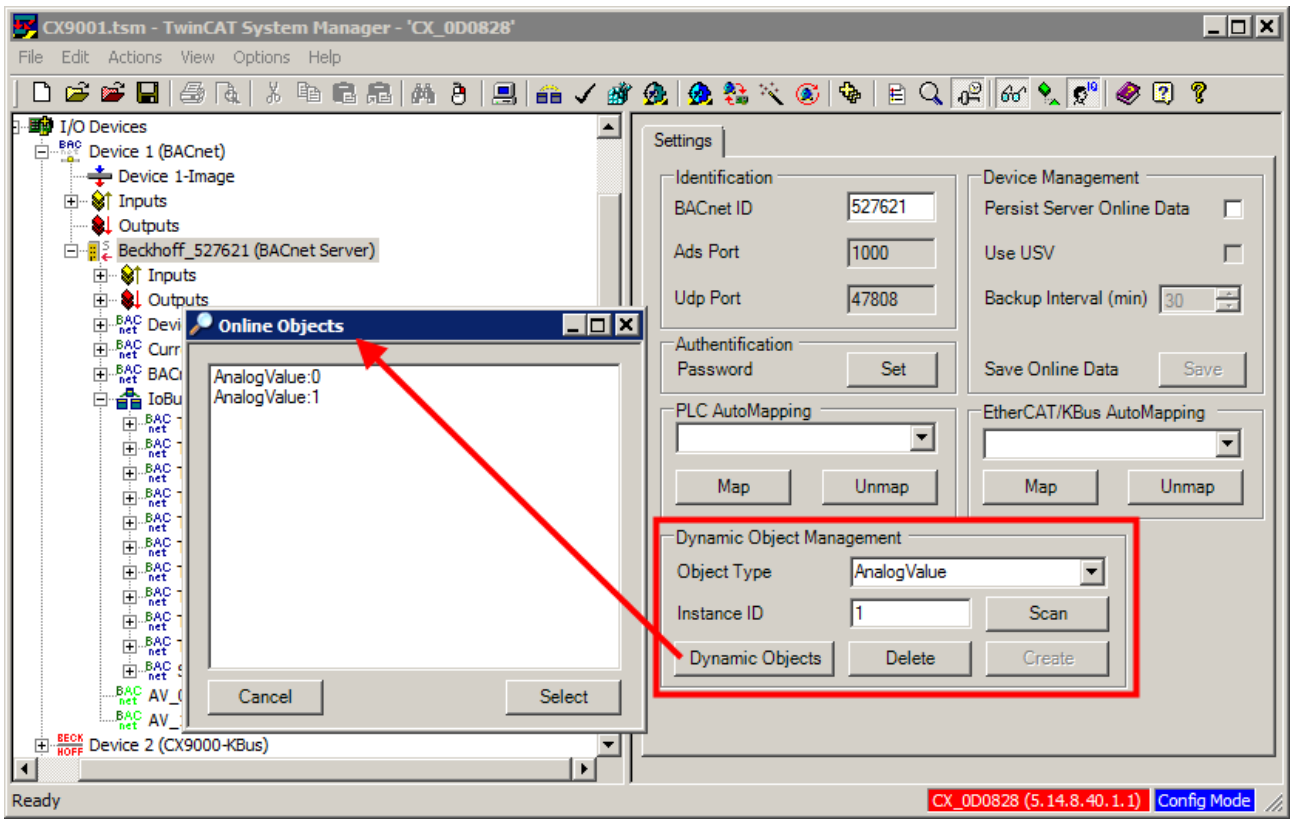
TwinCAT BACnet/IP also supports the *Unsolicited COV* server functionality. This means that change messages can be sent as broadcast in the BACnet network without registration. *Unsolicited COV* is configured via a Notification Sink (or several Notification Sinks). The context menu in the "Settings" tab can be used to activate Unsolicited COV via the checkbox "*Use unsolicited COV Broadcasts*". This causes all configured COV *subscriptions* to be stored and processed internally with broadcast *recipient*. The COV subscription list can also be used for settings such as the COV *increment*.

The Property wizard, which is available in the context menu, can be used to select the properties for which change notifications are to be sent.

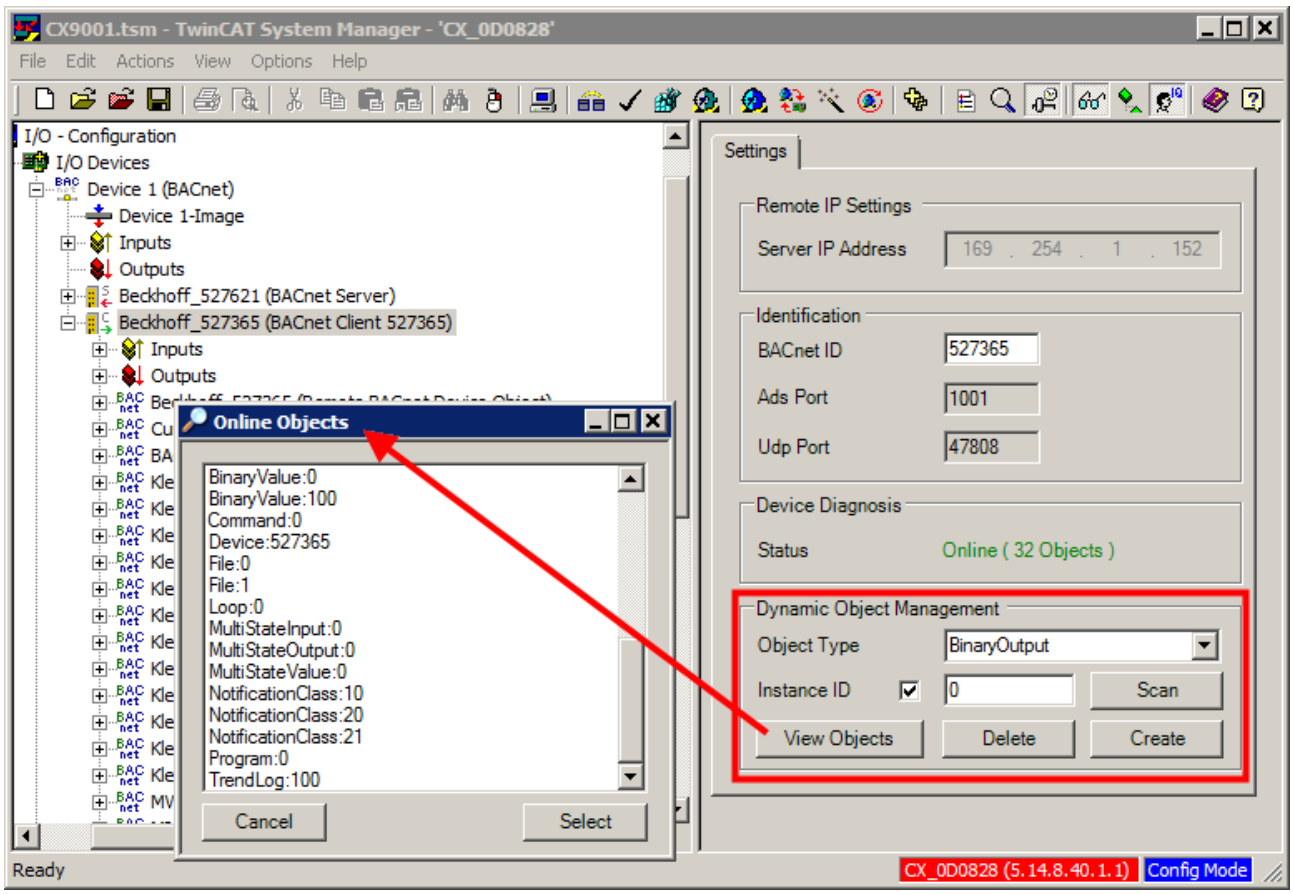


2.6 Managing dynamic objects

Dynamic objects are BACnet objects that can be created during runtime (RUN or Free RUN) on a server (local) or a client (remote) without having to reactivate/restart TwinCAT. Dynamic objects of a local server are accessed via the Settings tab of the BACnet server. In the Settings tab the dynamic objects can be displayed, created or deleted. In addition, the Scan button can be used to scan existing objects that are not displayed and integrate them into the tree diagram under the server (e.g. dynamic objects that were generated by a remote system). To create an object the corresponding Object Typeand an Instance ID (object number) have to be entered. To delete, enter an existing object number with associated object type, or conveniently select it with the button "Dynamic Objects".



The procedure for accessing the dynamic objects of a client (remote access) is similar. The management function can be found in the Settings tab of the associated BACnet client. The difference in the management of a server is in the distinction between dynamic and static (configured offline) objects. In general all dynamic objects can be created/deleted, except loop, device and program objects. Static objects can neither be created nor deleted at runtime. However, since the BACnet communication makes no distinction between dynamic and static objects, there is no difference in how the objects are displayed in the tab "Settings" → "View Objects" of the BACnet client. The "Online Objects" dialog therefore lists all objects that the client side makes available. An attempt to delete a static object is acknowledged with fault feedback from the BACnet client (error result). The Settings tab of the BACnet client offers an option to create several dynamic objects simultaneously. To this end press the shortcut CTRL+SHIFT when pressing the Create button.



Dynamic objects are indicated with a green symbol under a BACnet server. Due to the client behavior mentioned above, on the client side there is no distinction in the tree view.

During the dynamic creation of client objects the user can specify whether or not an object ID has to be specified in the CreateObject service. It should be noted that the devices from some manufacturers only support dynamic object creation without specification of an object ID. In this case the "Instance ID" option can be deselected.

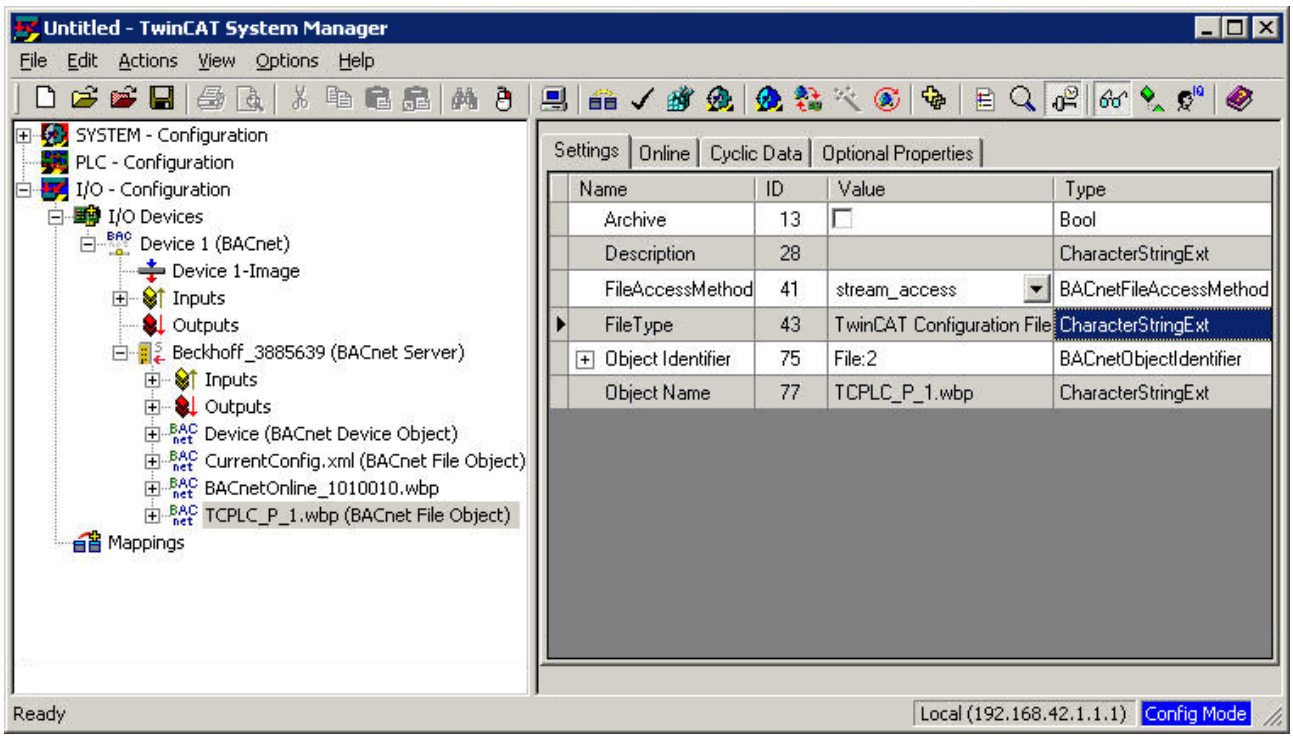
Since dynamic objects are only created at runtime, there is no process data mapping [▶ 42] option. The linking of process data must be known during loading of the TwinCAT configuration. This in turn precludes offline configuration of dynamic objects. The configuration therefore always takes place under the Online tab. Please note that the configuration of all dynamic objects and their properties is only retained after a restart if data persistence is activated (for information on data persistence please refer to chapter "Persistent data [▶ 57]").

2.7 Backup and restore

BACnet enables the creation of backup for the configuration of a BACnet controller. In TwinCAT BACnet/IP the configuration may consist of several files. One of the main files is CurrentConfig.xml, which contains the general TwinCAT configuration. In addition, TwinCAT BACnet/IP uses a further BACnetOnline file which contains the relevant data that were modified during runtime and differ from the original configuration of CurrentConfig.xml. Further files may be used to determine a configuration, e.g. a PLC boot project or project-specific files.

The BACnet backup and restore mechanism enables a customizable number of files to be backed up and restored using the file services. Once a backup was selected via the corresponding BACnet services, a BACnet client can determine which BACnet file objects belong to a backup configuration belong by reading the device object property ConfigurationFiles. The BACnet file objects refer to the corresponding file names. Please note that TwinCAT BACnet/IP treats all files relative to the boot project folder (default: "C:\TwinCAT\Boot"). Backup-relevant files should therefore be stored in this folder.

Creating a BACnet server automatically creates two file objects, which refer to the file CurrentConfig.xml and the BACnetOnline file.



BACnet TwinCAT/IP enables further files to be included in a backup configuration. To this end can a new BACnet file object can be added. The file name can be configured in the property ObjectName. If the property FileType is initialised with the string "TwinCAT Configuration File", during a backup request this file object is included in the property ConfigurationFiles so that it can be backed up by a BACnet client and subsequently restored. The screenshot illustrates how the boot project for PLC runtime 1 can be included in the backup configuration.

Backup-relevant data can also be stored in the PLC. A feedback mechanism between BACnet and a PLC ensures that the PLC backup data are written to the storage medium during a backup operation. Cyclic monitoring of the property SystemStatus of the server device object enables the timing of a backup to be detected (BackupInProgress (5)). A feedback procedure can be activated via the server control status word (ServerControl) by setting bit 0. If bit 0 is set in the control status word, the BACnet stack waits for the PLC data to be written before processing the backup request of a BACnet client (once bit 1 is set in the control status word). A corresponding PLC sequence is shown below:

1. Set bit 0 in the control status word
2. Monitor SystemStatus - when value BackupInProgress (5): writing the relevant data
3. After the write operation set bit 1 in the control status word
4. When SystemStatus has the value operational (0) again: delete bit 1

A further backup-relevant file is "BACnet_LastRestoreTime.wbp", which is stored in the boot folder of TwinCAT. This file contains data that have to survive a restore. They include the properties LastRestoreTime and DatabaseRevision.

TwinCAT BACnet/IP currently does not support triggering of a backup or restore operation. A corresponding building management tool can be used to backup and restore the Beckhoff controllers. In other words, only the server is supported in this case, not the client side.

2.8 Persistent data

If BACnet properties are changed via BACnet or the online dialog of server objects, the changes are initially lost when TwinCAT is restarted, since the corresponding values from the Settings dialog are used for initialization. In order to retain the data persistently during a device restart, the function Persist Server Online Data can be used in the Settings tab of the BACnet server. If this function is activated, during shutdown and perhaps cyclically the values of all BACnet properties that have changed compared with the initial configuration are saved in a file and reloaded when a configuration is loaded. The file is created in the boot folder of the TwinCAT installation ("C:\TwinCAT\Boot") and has the name "BACnetOnline_1010010.wbp". The number code identifies the server via a TwinCAT(TcCOM) object ID.

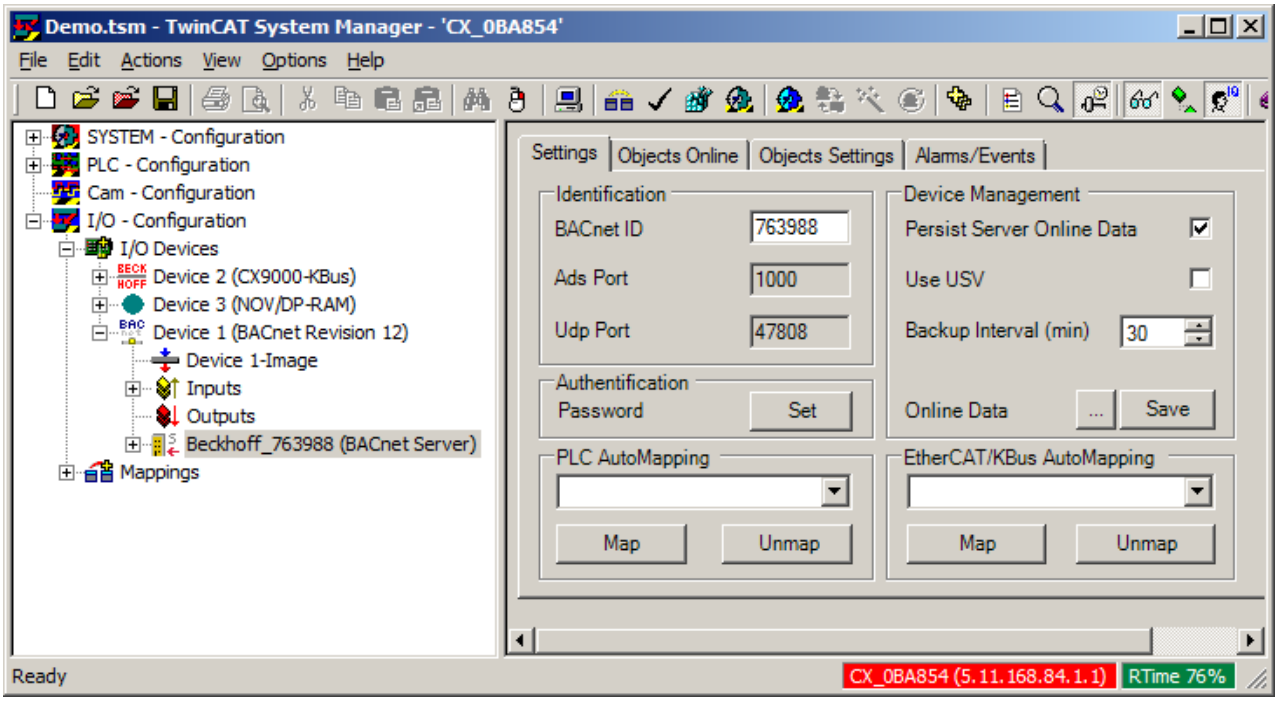
Devices with UPS can be shut down even if the supply voltage is lost, so that the storage of persistent data is ensured. For devices without UPS the data have to be stored cyclically or in the event of changes. The "*Use USV*" option in the Settings tab of the BACnet server can be used to specify whether data should be stored at a specified interval during operation. The *"Backup Interval*" dialog box can be used to specify the interval (in minutes) at which modified BACnet properties should be saved. Data are only saved if a BACnet property has been changed since the last backup.

The option "Use USV" does not result in automatic shutdown of the device in the event of a power failure. Monitoring of the UPS status and associated tasks can be implemented in the PLC with the aid of corresponding libraries. The option "Use USV" only disables periodic saving of persistent data. Writing of persistent data has to be affected manually or through the PLC via ADS or the property PersistentData. The PLC library TcBACnetRev12.lib provides functions for triggering the saving process.

NOTICE

Note the maximum number of write cycles

If data are cyclically saved to a flash medium, the storage medium may be damaged or destroyed if the backup intervals are too short. It is therefore important to ensure that the number of write cycles specified by the flash manufacturer is not exceeded over the expected operating period.



Note that with this functionality all settings that were made in the "Settings" tabs of the objects after the current TwinCAT configuration has been activated become invalid, since all settings are taken from the BACnetOnline file. If a configuration has to be changed, first deactivate the function and delete the BACnetOnline file. Attention: Once the function has been disabled, the configuration is still loaded with activated persistence function, i.e. the BACnetOnline file is generated again when the system is shut down. Writing of the file is only disabled when the configuration is activated without the persistence function. With the new dialog for managing persistent data, changes can nevertheless be easily adopted. A description follows at the end of this chapter.

Dynamically generated BACnet objects are also included in the persistent data and re-generated during a restart.

When a new TwinCAT BACnet/IP configuration (or a BACnet server, to be more precise) is created, a unique Config ID is generated within the project, through which the BACnetOnline files are allocated to a configuration (CurrentConfig.xml). This is checked when the online data are loaded. If the Config ID does not match the loaded project, the online data are not applied. In this case the data from the Settings tabs are used.

Loading and saving persistent data

This section describes the procedure for loading and saving the online data. It ensures that a valid version of the persistent data is always available, even in the event of a power failure. Loading and saving of persistent data always takes place in RUN mode, and only when persistent data were activated.

Sequence: loading the BACnetOnline file

1. If present: load the file "BACnetOnline_XXXXXXXX.wbp"
2. After successful loading: rename the file "BACnetOnline_XXXXXXXX.wbp" to "BACnetOnline_XXXXXXXX.wb~"
3. If "BACnetOnline_XXXXXX.wbp" does not exist: load the file "BACnetOnline_XXXXXX.wb~"

Sequence: writing the BACnetOnline file:

1. Rename an existing file "BACnetOnline_XXXXXXXX.wbp" to "BACnetOnline_XXXXXXXX.wb~"
2. Write the modified property and object data to the file "BACnetOnline_XXXXXXXX.wbp.tmp"
3. After the write operation: rename the file "BACnetOnline_XXXXXXXX.wbp.tmp" to "BACnetOnline_XXXXXXXX.wbp"

Please note that the described procedure requires twice the memory on the flash memory.

Relevant changes

As already mentioned, online data are only saved if appropriate changes were made. Changes can be triggered by:

- Writing of a BACnet property via WriteProperty or WritePropertyMultiple
- List manipulation via AddListElement, RemoveListElement
- Creation of dynamic objects via CreateObject
- Deleting of dynamic objects via DeleteObject
- Changing of BACnet properties through state machines within objects (e.g. ElapsedActiveTime of binary objects, LogBuffer in the TrendLog object)
- Changing of the Broadcast Distribution Table (BDT) if BBMD is activated

Changes in the following BACnet properties generally do not lead to persistence-relevant changes:

- Permanently read-only BACnet properties (e.g. Protocol Revision, Object Type, ObjectIdentifier)
- ActiveCOVSubscription: as specified by the BACnet specification, COV subscriptions that have been carried out are not persistent.
- BACnet properties resulting from other BACnet properties or internal system states (LocalDate, LocalTime, StatusFlags, EventState, DeviceAddressBinding, ObjectList, ModificationDate, SystemStatus, PersitentData, ActivePriority)
- DaylightSavingsStatus and UtcOffset if operating system time settings are used (Time management "Use operating system settings")

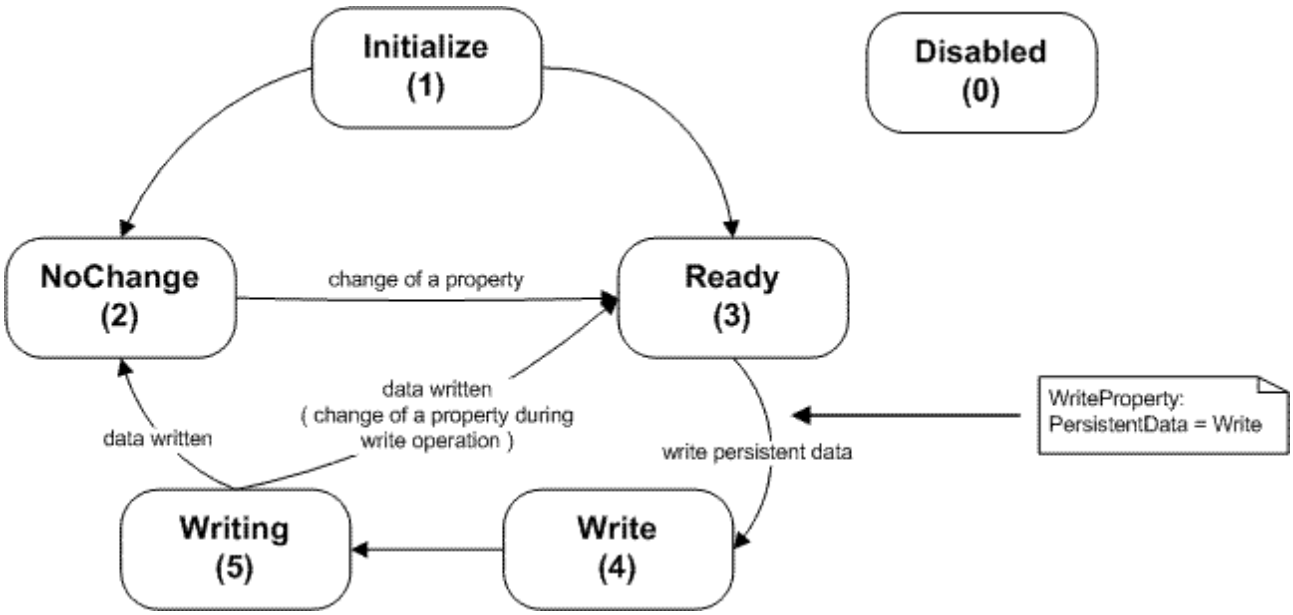
Manual saving of persistent data

Saving of persistent data can be triggered manually/programmatically. An acyclic (ADS) interface is available via which writing of the BACnet online data can be triggered:

IndexGroup	IndexOffset	Description
0x80C00000	always 0	Triggering saving of the persistent data. Only valid for BACnet servers if Persist Online Data is active.

Saving can be triggered via ADS with the aid of the System Manager (BACnet server Settings tab, "Save" button).

Saving of the persistent data can also be triggered via the BACnet network and the vendor-specific BACnet property PersistentData (of the device object). A state diagram for the BACnet property PersistentData is shown below.



The BACnet property PersistentData is only visible if the persistent data are activated and the configuration is executed in RUN mode. After the initialization the property of the BACnet-data type Enumerated indicates whether a relevant BACnet property has changed or whether dynamic objects were created since the persistent data were last saved. If the data are unchanged the property PersistentData has the value NOCHANGE (2). If data are changed via BACnet (WriteProperty, WritePropertyMultiple, CreateObject, DeleteObject, WriteBDT), via ADS Write or through an object state machine, this is indicated by the property PersitentData with the value Ready (3).

If the property PersistentData has the value Ready (3), writing can be triggered via BACnet by writing the value Write (4). During writing PersitentData has the value Writing (5). After successful writing of the persistent data PersistentData enters the state NoChange (2) or Ready (3), depending on whether relevant data have changed during writing.

If persistent data is saved by manually triggering the write process, it is recommended to set the backup interval to a correspondingly high value or to deactivate the periodic backup with the function "*Use USV*".

Change management for persistent data

A new dialog was integrated in TwinCAT BACnet/IP to assist with the management of persistent data. This dialog can be enabled via the button "..." next to the Save button on the BACnet server "Settings" tab. The dialog can be used to deactivate or activate temporary storage of the persistent data at runtime (even if it was not active in the configuration). Temporary deactivation is reset at the next restart or when this function is temporarily activated. The transfer function can be used for (online) transfer of modified data into the Settings dialog (and therefore into the .tsm file). Attention: In contrast to the transfer function of the Property wizard, only data that were modified online are transferred. Property data that were changed in the Settings dialog, but not online, are preserved. "Delete" can be used to delete the ".wbp" files. Save triggers the saving of the peristent data.

In a project with activated persistent data, the following procedure is recommended for loading changes, or for storing data, which were changed online after commissioning, in the .tsm file:

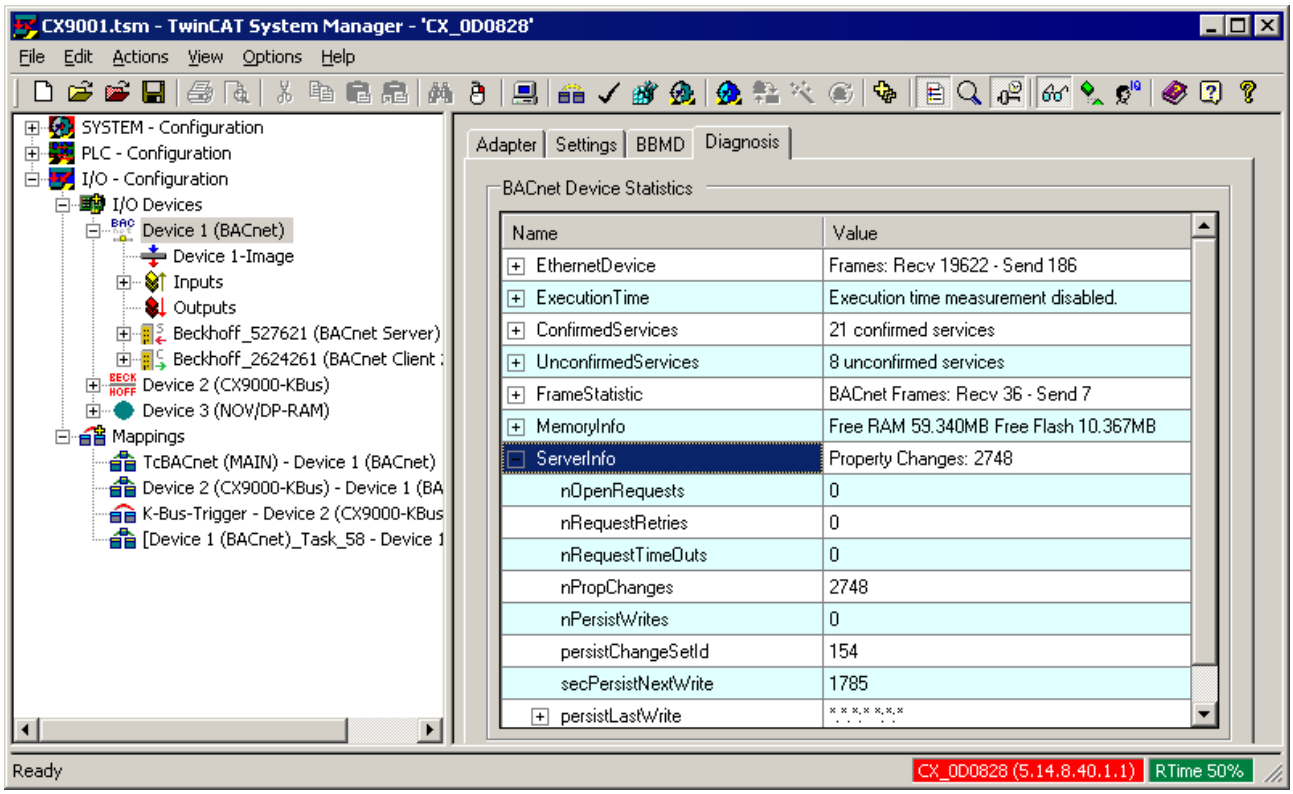
1. Trigger the transfer function (changed data are transferred from Online to Settings)
2. Deactivate writing of persistent data (Disable button)
3. Delete the persistent data, since the changes are now in .tsm (Delete button)
4. Save the configuration (.tsm) (optional)
5. Activate the configuration (System Manager - Actions - Activate Configuration ...)
6. Restart the controller

Diagnosis persistent data

The online tab of the BACnet objects shows for each BACnet property whether it has been changed compared to the Settings dialog. A * means that a BACnet property has been changed. An additional check mark means that the change was saved in the .wbp file.

	InactiveText	46	FALSE
*	ChangeOfStateTime	16	20.06.2014 09:46:12
*	ChangeOfStateCount	15	1
*	TimeOfStateCountReset	115	20.06.2014 09:45:57
*	ElapsedActiveTime [s]	33	16
*	TimeOfActiveTimeReset	114	20.06.2014 09:45:57
	MinimumOfftime [s]	66	0

The diagnostic functions of the BACnet device enable information relating to the processing of persistent data to be viewed.



Under ServerInfo you can find persistence specific information:

- **nPropChanges**: Indicates how often BACnet properties were modified since the system startup.
- **nPersistWrites**: Indicates how often the BACnetOnline file was written since the system startup.
- **persistChangeSetId**: After each writing of the online file a ChangeSet ID is incremented in the BACnet driver, in order to enable detection of changes of BACnet properties since the last writing. This ChangeSet ID is written to the online file and indicates how often a project was saved since the first start. During a BACnetrestore the ChangeSet ID of the written online file is used.
- **secPersistNextWrite**: Indicates in how many seconds the next cyclic writing of persistent data is triggered.
- **persistLastWrite**: Indicates when the online file was last written. If it has not been written since the system startup, this timestamp is initialized with *.*.*.* *.*.*.* .

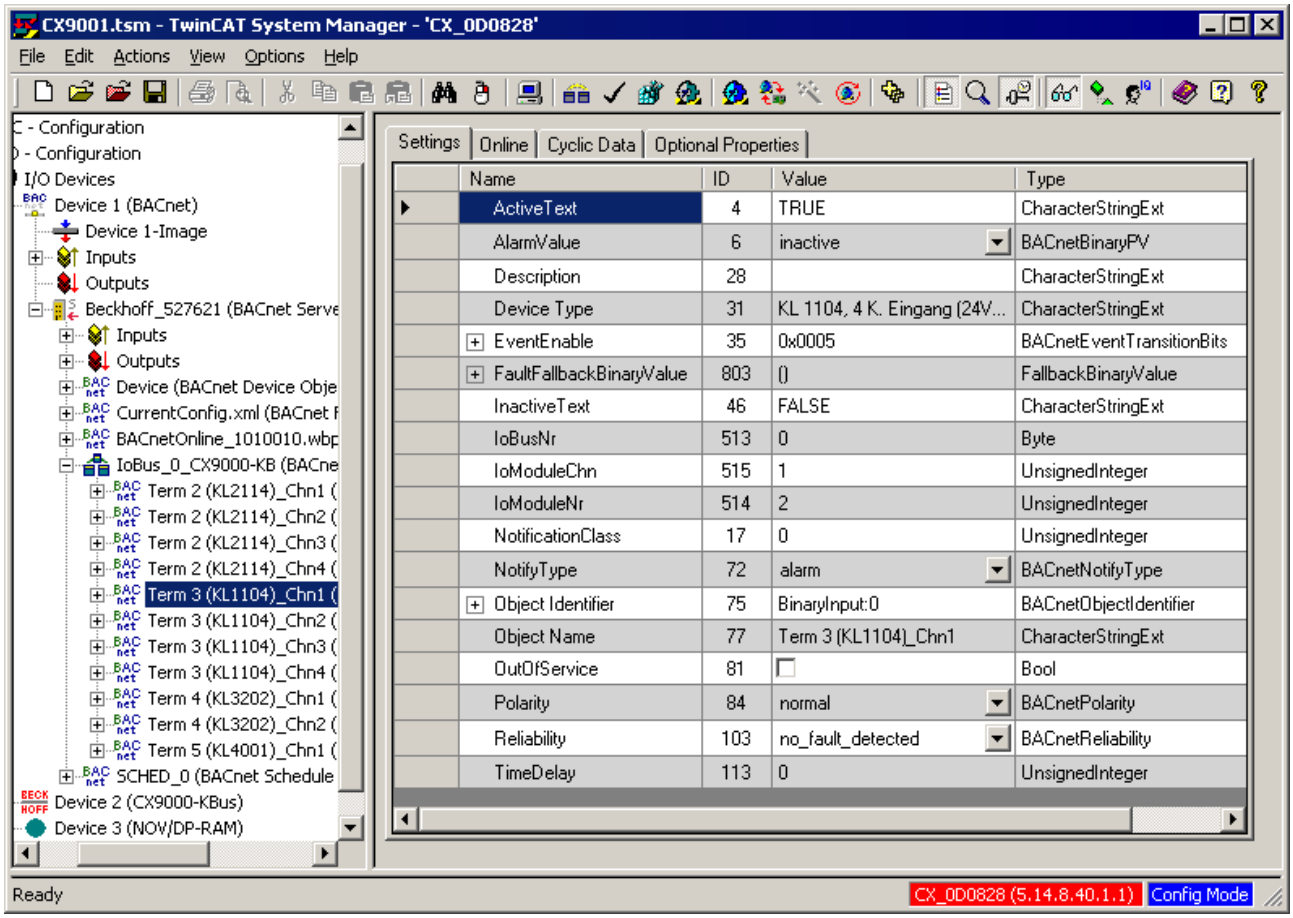
The WbpView dialog enables a more exact analysis of which property data were modified and are stored in the .wbp file. It can be activated as a wizard via the File object of the .wbp file. The wizard shows an overview of the contents of the binary wbp data in text format.

2.9 PLC automapping

Introduction

Special comments that are added after the PLC variable declaration can be used for automated configuration of BACnet objects from the PLC program. The procedure can be used to configure BACnet objects, link BACnet property variables with PLC variables and initialize BACnet properties.

PLC automapping can be carried out via the BACnet server in the Settings tab, once the PLC project has been added to the hardware configuration. Automapping is triggered by pressing the Map button once the required PLC program has been selected:



PLC automapping for BACnet servers

The following source code extract illustrates the application of automapping comments. An allocated variable "av0" of type REAL created. The used comments define BACnet properties for use with automapping. The general comment syntax is defined by the PLC. Comments start with "(*" and end with "*)". Comments with the special prefix "~" are transferred by the PLC compiler into the TPY file for the respective symbols. These special comments, referred to as property, always have the following structure: "(NAME : VALUE : COMMENT)". For BACnet automapping the name is composed of "BACnet_[BACnet-Property-Name]". "VALUE" depends on the BACnet property and is explained accordingly.

In the sample the symbol "av0" is annotated with 3 BACnet properties. It is specified that the object is an AnalogValue object (AV). The object instance is 100. For the property ObjectIdentifier it should be noted that in the PLC comment it only contains the instance number, not the coded object type, as in BACnet. In the sample, automapping creates a BACnet object of type AnalogValue with the instance number 100. For each symbol a link can be made with a cyclic BACnet property variable. For variables that are not to be linked the comment "NOLINK" (see note in the appendix) can be used. For each symbol only one property may exist for linking. In the sample the property PresentValue of the AnalogValue object is linked with the PLC variable "av0".

```
av0 AT %I* : REAL;          (* ~ (BACnet_ObjectType     :  AV        : NOLINK)
                    (BACnet_ObjectIdentifier   :  100       : NOLINK)
                    (BACnet_PresentValue       :        : )
                *)
```

A further sample is provided below. In this case a BACnet object of type BinaryInput is created. No object instance is defined. During automapping an available object instance number is generated automatically. The sample demonstrates how the values of certain BACnet properties can be pre-initialized. The BACnet properties Description and ObjectName are initialized with the values "Hello, BACnet" and "BinaryInput_1". After automapping the corresponding values appear in the generated BACnet object (Settings tab) when the configuration is activated "Online". In addition the object name is used in the System Manager tree. In the sample a variable of type BOOL is linked with the property PresentValueBool. Although the BACnet data type of the PresentValues of binary objects is an enumeration type, the auxiliary property PresentValueBool was introduced for efficient handling of the PLC, in order to be able to link Boolean variables. For further details please refer to chapter "Process data [▶ 42]".

```
bi0 AT %I* : BOOL;          (* ~ (BACnet_ObjectType     : BI          : NOLINK)
                    (BACnet_Description    : Hello, BACnet  : NOLINK)
                    (BACnet_ObjectName     : BinaryInput_1  : NOLINK)
                    (BACnet_PresentValueBool   :        : )
                *)
```

Even if variables of primitive type are used, several variables can be linked with a BACnet object. To this end identical object type and object instance parameters should be specified for two variables. In the sample the BACnet properties PresentValue and StatusFlags are linked with AnalogValue object "101". In order to be able to report object instance numbers that have been assigned twice by mistake as errors, the comment "OBJREF" must be added to the 2nd symbol that refers to a previously created BACnet object. "OBJREF" must always be declared for the BACnet property "ObjectIdentified". If an object is to be accessed by name via 2 symbols, "OBJREF" must be used when declaring the property *ObjectName*.

```
av0 AT %I* : REAL;          (* ~ (BACnet_ObjectType     : AV          : NOLINK)
                    (BACnet_ObjectIdentifier   : 101        : NOLINK)
                    (BACnet_PresentValue       :        :)
                *)
av0_StatusFlags AT %I* : WORD; (* ~ (BACnet_ObjectType     : AV          : NOLINK)
                    (BACnet_ObjectIdentifier   : 101        : OBJREF,NOLINK)
                    (BACnet_StatusFlags    :        :)
                *)
```

Commandable properties

BACnet objects can also be accessed in write mode from the PLC. Commandable BACnet properties are linked with a PriorityArray and support write access with priorities 1 to 16. By using priorities, other system components can override signals from the PLC. The general rule for write access from the PLC to BACnet is that a value is only applied from BACnet when the program is started or changed. If the BACnet network writes competitively to the same priority level, the PLC-generated value may differ from the BACnet value. However, for prioritized accesses, a priority level should be used exclusively in most cases. In the case of several potential same-priority change sources, the BACnet property *RelinquishDefault* can also be used.

In the sample a "Q"-allocated variable is defined, which is linked with the PresentValue of an AnalogOutput object. Writing from the PLC results in an entry with priority level "8" in the PriorityArray. The PresentValue properties of BinaryValue, BinaryOutput, AnalogValue, AnalagOutput, MultistateValue and MultistateOutput objects are commandable. The highest active priority (1=highest, 16=lowest) is set as PresentValue. A priority in the PriorityArray is active if it is not NULL. If all elements in the PriorityArray are NULL *PresentValue* takes the value of the BACnet property *RelinquishDefault*.

```
ao0_PresentValue8 AT %Q* : REAL;(*~ (BACnet_ObjectType     : AO          : NOLINK)
                    (BACnet_PresentValue_Priority8 :        :)
                *)
```

A special feature of commandable properties is the entry NULL in the PriorityArray, which is a separate data type in BACnet - in contrast to the respective value data types (REAL, BINARY_PV, UNSIGNED). The value NULL, which deactivates a priority level in the PriorityArray, can be written for the respective objects via the linked variable as follows:

- **BinaryValue, BinaryOutput** : priority-based PresentValues are linked as WORD (or an enumeration type, e.g. E_BACNET_BINARYPV). A 0 entry results in an INACTIVE entry in the PriorityArray, 1 results in ACTIVE and values greater than 1 result in NULL.

- **AnalogValue, AnalogOutput**: to generate a NULL entry in the PriorityArray a special bit coding of REAL values can be used, which are defined as NaN (not a number). This value can be generated via the function "F_BACnet_NAN()" in the BACnet PLC library "TcBACnet.lib". Note that this value is not used for calculations in the PLC, since they would trigger a floating-point exception. A value of 0.0 also results in a value of 0.0 in the PriorityArray. In addition, the following applies for AnalogOutput objects: writting values outside the Min/MaxPresValue range leads to NULL in the PriorityArray.
- **MultistateValue, MultistateInput**: is the value 0 or a value that is greater than the BACnet property NumberOfStates is written, a NULL entry is generated in the PriorityArray.

Structured programming

Function blocks (FBs) are also supported for structured linking of BACnet objects during automapping. The source code extract shown below illustrates the declaration of a BACnet function block. In order to be considered in automapping, these function blocks must begin with the prefix "FB_BACnet". The sample defines an FB for an AnalogOutput object, which is available as a template in the BACnet PLC library "TcBACnet.lib". The properties PresentValue, StatusFlags, Reliability and OutOfService are linked as input variables to the PLC, the property PresentValue with write-access, priority 8.

```
FUNCTION_BLOCK FB_BACnet_AnalogOutput
VAR_OUTPUT
    ObjectType: E_BACNETOBJECTTYPE;        (* ~ (BACnet_ObjectType       : AO   : NOLINK) *)
    PresentValue AT%I*: REAL;          (* ~ (BACnet_PresentValue     :      :) *)
    StatusFlags AT%I*: UINT;           (* ~ (BACnet_StatusFlags      :      :) *)
    Reliability AT%I*: E_BACNETRELIABILITY;    (* ~ (BACnet_Reliability      :      :) *)
    OutOfService AT%I*: BOOL;          (* ~ (BACnet_OutOfService     :      :) *)END_VAR
VAR_INPUT
    PresentValue_Priority8 AT%Q*: REAL;        (* ~ (BACnet_PresentValue_Priority8 ::) *)END_VAR
```

This FB can be instantiated in the main program (or another FB, program). Additional comments can be specified during instantiation. Comments at the instantiation level have higher priority than comments at FB declaration level. In the sample an AnalogOutput object is created, and the object instance and the BACnet property Description are initialized at instantiation level. Through the FB declaration the properties PresentValue, StatusFlags, Reliability, OutOfService and PresentValue_Priority8 are linked in the generated BACnet object.

```
ao1 : FB_BACnet_AnalogOutput;          (* ~ (BACnet_ObjectIdentifier : 100 :NOLINK)
                            (BACnet_Description      : FB-basiertes AO: NOLINK)
                           *)
```

BACnet FBs can be used hierarchically, i.e. an "FB_BACnet_Pump" containing several FB_BACnet_AnalogOutput instances can be defined, for example. For nesting depths greater than one a StructuredView object is created in the System Manager. BACnet comments are not handed down to several hierarchical levels. Only the instantiation and declaration levels are supported. An exception is the BACnet comment BACnet_ObjectIdentifier, which is resolved over several hierarchical levels, in order to support ID allocation for nested FBs.

If an object instance is specified at a higher level, it may potentially affect many objects of the same type. The generation of an object ID is based on the specified ID (e.g. if an object instance of 1000 is specified, a new free ID is searched for from 1000). An FB_BACnet_Room with 2 AV and 2 BV objects and a start ID of 1000 will therefore generate BACnet objects AV:1000, AV1001, BV:1000 and BV:1001. The resulting object IDs are calculated before the actual mapping operation. If objects with the determined IDs already exist, e.g. from an IO mapping, the existing objects are used/linked, and no new objects are created.

In addition to nested FBs, one-dimensional arrays are also supported. The following sample shows the declaration of an array *raeume* of type FB_BACnet_Raum that contains further BACnet objects. In this way systematic units with identical functionality can be managed. Arrays at different nesting levels are also supported. For example, a BACnet object *fbHaus[1].EtageU.Raum[10].tmpSoll* can be generated.

```
raeume : ARRAY[0..10] OF FB_BACnet_Raum;
```

Structured programming: configuration of FB instances

BACnet comments at FB declaration level apply equally to all instances of the FB. For certain BACnet properties, an instance-level definition may apply. For example, instance-based ObjectIdentifiers can be defined or the hardware connection can be configured to Io modules per instance. The following is a sample where BACnet properties are set at the instance level.

```
FUNCTION_BLOCK FB_BACnet_SubSub
VAR
  bi  : FB_BACnet_BinaryInput; (*~(BACnet_OutOfService : TRUE: NOLINK)*)
  bi2 : FB_BACnet_BinaryInput;
END_VAR
```

```
FUNCTION_BLOCK FB_BACnet_Sub
VAR
  sub1   : FB_BACnet_SubSub;
  sub2   : FB_BACnet_SubSub;
  subArr : ARRAY[0..1] OF FB_BACnet_SubSub;
END_VAR
```

```
PROGRAM SIMPLE
VAR
s1 :FB_BACnet_Sub; (* ~ { BACnet_ObjectIdentifier<sub1/bi>    : 1017   : }
            { BACnet_ObjectIdentifier<sub2>       : 1028   : }
            { BACnet_ObjectIdentifier<subArr[0]/bi>   : 5555   : }
            { BACnet_ObjectName<sub2/bi>          : biname : NOLINK }
           *)
END_VAR
```

By defining an instance path after the BACnet property name, BACnet property definitions can be applied to sub-elements of an FB. For this purpose, the instance path can be specified in "<" and ">". In the above sample, for example, the BACnet object bi of the instance sub1 is given an ObjectIdentifier 1017. Individual sub-levels are separated by "/". BACnet property definitions can also be configured at intermediate levels. In the sample the ObjectIdentifier 1028 is transferred to the FB instance sub2. In this sample, this causes the BACnet objects s1.sub2.bi to be given Object ID 1028 and s1.sub1.bi2 to be given Object ID 1029. The instance path specification can also contain array elements.

```
ARRAY[01..03] OF FB_BACnet_AnalogValue; (* ~
   (BACnet_Description<[1]>:1. OG Raum 1 Temp: NOLINK)
   (BACnet_Description<[2]>:1. OG Raum 2 Temp: NOLINK)
   (BACnet_Description<[3]>:1. OG Raum 3 Temp: NOLINK)
*)
```

Direct parameterization of array elements at program or global level is also supported. In the sample, the BACnet property Description of three array elements is initialized.

Structured programming: View configuration

During PLC automapping, StructuredView objects are created for each nesting depth in the form of programs and function blocks, in order to map the PLC structure to BACnet. Depending on the data point addressing description used, it may be necessary to adapt the names BACnet objects and their structure with a name prefix. The PLC comment "BACnet_StructuredViewPath" can be used to define dedicated name prefixes for PLC automapping. "BACnet_StructuredViewPath" can be used to annotate any variable at the required structuring levels.

```
PROGRAM MAIN
VAR
  viewDummy : BOOL; (* ~     ( BACnet_StructuredViewPath        : \/Building1\/Floor2 : )
( BACnet_StructuredViewPathDescription : \/Haus 1\/Flur 2 : )
            ( BACnet_StructuredViewPathObjectId    : \/1000\/9: )
             *)
  room1 : FB_BACnet_Room; (* ~( BACnet_Description : Description of structured view object room1: )
*)
  room2 : FB_BACnet_Room;
END_VAR
```

In the sample the name Building1.Floor2.room1.XXX is used for all BACnet objects within the function block FB_BACnet_Room, instead of the name MAIN.room1.XXX. In addition to the use for the names of BACnet objects, a StructuredView object is also created for each hierarchical level. Hierarchical levels are defined by the string "\/".

The PLC comments "BACnet_StructuredViewPathDescription" and "BACnet_StructuredViewPathObjectId" can be used to specify the BACnet property Description and the ObjectIdentifiers of the generated StructuredView objects. StructuredView objects that are generated through complex function blocks; in sample room1 and room2 can be configured via the PLC comments BACnet_Description and BACnet_ObjectIdentifier.

Data point addressing description

Depending on the required data point addressing description, it may be necessary to adapt the generated BACnet object names. By default, points (".") are used to separate structural levels. By specifying a "StructureSeparator" another symbol can be used instead of the point or the separator symbol can be removed completely. The following sample shows the definition of the separator symbol "_". The separator symbol should be defined once per project, e.g. at a global variable.

```
akzDummy : BOOL; (* ~( BACnet_StructureSeparator :_: ) *)
```

Write protection and optional properties

Special comment options were introduced for controlling write protection and optional properties. Additional options can be specified in the form of a comma-separated list in the comment part of the PLC comments. DISABLE deactivates an optional property. WRITEPROTECT activates write protection for a writeable property. Make sure that the option NOLINK is specified when configuring multiple properties. The following sample shows the configuration of write protection and optional properties.

```
bo1: FB_BACnet_BinaryOutput; (* ~(BACnet_NotifyType          :    : DISABLE,NOLINK)
                  (BACnet_ChangeOfStateTime      :    : DISABLE,NOLINK)
                  (BACnet_ChangeOfStateCount     :    : DISABLE,NOLINK)
                  (BACnet_ActiveText         : AN     : WRITEPROTECT,NOLINK)
                  (BACnet_InactiveText       : AUS    : WRITEPROTECT,NOLINK)
                   *)
```

The comment ENABLE can be used for BACnet properties with inverted optional logic, i.e. BACnet properties that are initially disabled when a BACnet object is created. An example of this is activation of a mean value-forming filter in the AnalogInput object by means of the vendor-specific BACnet property *AvgFilterCycles:*

```
fbAI : FB_BACnet_AnalogInput; (* ~(BACnet_AvgFilterCycles     : 20    : ENABLE,NOLINK ) *)
```

Initialization of complex property values

The initialization of property values via PLC automapping is limited by the PLC comment syntax. The reserved characters "(", ")", ":", "*" and " cannot be used. In addition, for BACnet properties with selection type (Choice) or optional fields it is not always possible to unambiguously analyze each string, if an optimized display form is used, as in the System Manager.

For certain properties special parsers were implemented, which enable simplified configuration of property values. The following sample shows how the StateText property of a MultiStateValue object can be initialized.

```
mv : FB_BACnet_MultiStateValue;  (* ~(BACnet_StateText   : {Low;Medium;High}       : NOLINK )*)
```

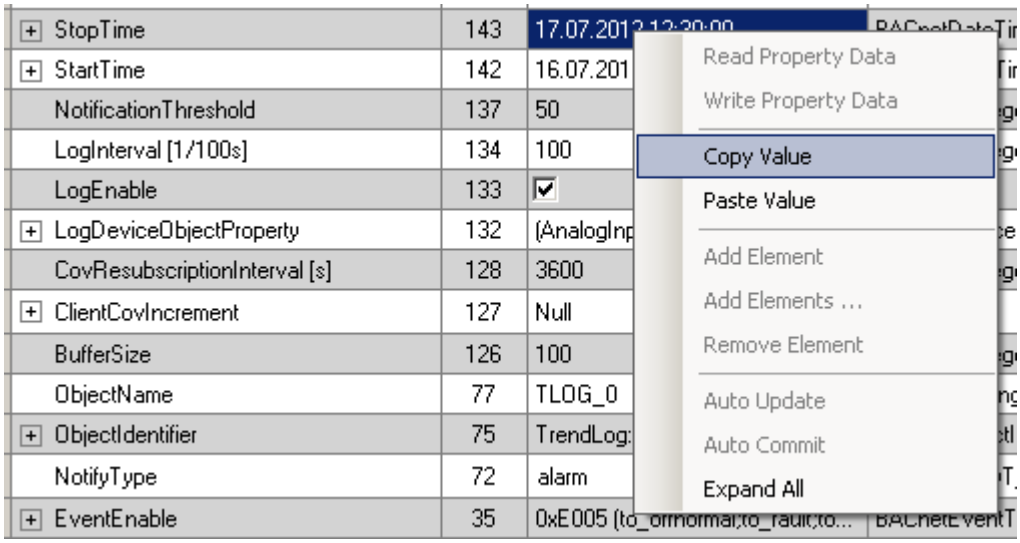
BitField properties (EventEnable, AckRequired) can be initialized by specifying a hexadecimal number. In this case it is advisable to use the corresponding value from the System Manager:

```
bi : FB_BACnet_BinaryInput;      (* ~(BACnet_EventEnable : 0xE005          : NOLINK )*)
```

The Priority property of the NotificationClass object can also be initialized in abbreviated form. In the sample this array property is initialized with the values 12, 12 and 122. For the initialization of BitField properties the more transparent form of array-like initialization was implemented. In the sample the bits for "to_offnormal" and "to_fault" are set in the property AckRequired. A BitField property without set bit can be initialized with the string "{}".

```
nc : FB_BACnet_NotificationClass;(* ~(BACnet_Priority    : {12;12;222}          : NOLINK)
                     (BACnet_AckRequired : {to_offnormal;to_fault} : NOLINK)
                  *)
```

Since it is not possible to implement dedicated parsers for all complex properties, the option of initialization with the aid of XML was introduced. XML has the advantage that it is not necessary to use the reserved comment syntax characters, so that there are no restrictions for initialization of the property values. The XML values for the property initialization can simply be created via the Copy&Paste function for the property values. A property can be created in the System Manager using wizards. Via the context menu in the property view ("Online" or "Settings") the current value of a property can be added to the clipboard via "Copy Value" (or via the shortcut CTRL-C) and used in PLC Control via "Paste" or CTRL-V.



Is the SHIFT key is pressed while using the "Copy Value" function, the XML string is copied as a line. Otherwise a structured variant is generated in multi-line mode. Using single-line mode makes the variable declaration in the PLC program more transparent. The following sample illustrates how the property StartTime of a TrendLog object is initialized via the XML syntax:

```
tl : FB_BACnet_TrendLog; (* ~(BACnet_StartTime :
<BACnetDateTime>
  <date>
    <year>112</year>
    <month>7</month>
    <day>16</day>
    <dayOfWeek>1</dayOfWeek>
  </date>
  <time>
    <hour>12</hour>
    <minute>30</minute>
    <second>0</second>
    <hundredths>0</hundredths>
  </time>
</BACnetDateTime>
: NOLINK )*)
```

Property references

The REFERENCE option was introduced for the initialization of reference properties. Automatic generation of object IDs means that direct value initialization of properties cannot be used for referencing properties of other PLC BACnet objects, since the object IDs are unknown when the PLC program is implemented. Specification of the REFERENCE option and property name as a suffix enables referencing of automatically generated objects. A simple heating system is described below as an example. Set values, actual values and control values are specified via analog* BACnet objects, which are linked with a Loop object via references.

```
fbSollwert   : FB_BACnet_AnalogValue;
fbStellwert  : FB_BACnet_AnalogOutput;
fbIstwert    : FB_BACnet_AnalogInput;
fbTempRegler : FB_BACnet_Loop; (* ~(BACnet_ManipulatedVariableReference : ./
fbStellwert : REFERENCE_PresentValue)
                  (BACnet_SetpointReference        : ./fbSollwert  : REFERENCE_PresentValue)
                  (BACnet_ControlledVariableReference  : ./fbIstwert   : REFERENCE_PresentValue)
                *)
```

In terms of REFERENCE functionality, a distinction is made between absolute and relative referencing. Relative references always refer to the current context and start with "./". The current context always refers to the current declaration level in the program or function block. Using relative referencing, objects can also be used in other programs or function blocks. One example is the declaration of a LOOP object in the "MAIN" program; to reference an object in the program "IO_OBJECTS", the syntax "./../IO_OBJECTS/fbAO" can be used. Absolute references always start directly with the program name (e.g. MAIN.fbBvI or MAIN.haus1.raum1.uv24Vok). For global variables, the "." must be removed at the beginning.

Generally, all references refer to symbol names in the PLC, not to the BACnet object name!

In general, references that may potentially contain BACnet objects in other devices (BACnetDeviceObjectPropertyReference), can also be initialized with the REFERENCE function. If the target symbol is a FB_BACnet_Remote_* function block, the target device ID is entered automatically.

The REFERENCE option also works for Schedule and TrendLog objects:

```
bi      : FB_BACnet_BinaryInput;
bv      : FB_BACnet_BinaryValue;
tl      : FB_BACnet_TrendLog; (* ~(BACnet_LogDeviceObjectProperty       : ./
bi      : REFERENCE_PresentValue) *)
sched   : FB_BACnet_Schedule; (* ~(BACnet_ListOfObjectPropertyRefs      : ./
bv      : REFERENCE_PresentValue) *)
```

The following section shows a sample for remote objects. The sample uses an absolute reference within the MAIN program.

```
bvRemote : FB_BACnet_RemoteBinaryValue; (* ~ (BACnet_ObjectType       : BV : NOLINK)
                         (BACnet_DeviceID     : 32 : NOLINK)
                         (BACnet_ObjectIdentifier : 2 : NOLINK) *)
fbTLog   : FB_BACnet_TrendLog;      (* ~ (BACnet_LogDeviceObjectProperty : MAIN.bvRemote : REFERENCE
_PresentValue) *)
```

Several references in the property *ListOfObjectPropertyRefs* of the Schedule object can be specified via the following syntax:

```
 schedMultiRef : FB_BACnet_Schedule;     (* ~ (BACnet_ListOfObjectPropertyRefs : Object=COOLING.bPrg
Cool;Property=PresentValue;
                                        Object=fbGlob;Property=PresentValue; : REFERENCE) *)
```

A special initialization is also available for other complex BACnet properties with references. These are described below, based on brief samples. Initially, the initialization follows the *Action* property of the Command object, which can be used for controlling process-based procedures.

```
fbCmdBo  : FB_BACnet_BinaryOutput;
fbCmdBV19: FB_BACnet_BinaryValue; (* ~(BACnet_ObjectIdentifier : 19 : ) *)fbCmdAv  : FB_BACnet_Analo
gValue;
fbCmdMv  : FB_BACnet_MultiStateValue;
fbCmd    : FB_BACnet_Command; (* ~(BACnet_Action :
   Action=1;Command=1;Object=./
fbCmdBo;Property=PresentValue;Value=ACTIVE;Priority=16;Delay=0;Quit=TRUE;
   Action=1;Command=2;Object=./fbCmdBo;Property=PresentValue;Value=NULL;Delay=1;Quit=FALSE;
   Action=1;Command=3;Object=BinaryValue_19;Property=PresentValue;Value=NULL;Delay=1;Quit=FALSE;
   Action=1;Command=4;Object=./fbCmdAv;Property=PresentValue;Value=NULL;
   Action=1;Command=5;Object=./fbCmdAv;Property=PresentValue;Value=12.3;Delay=1;Quit=FALSE;
   Action=2;Command=1;Object=./fbCmdMv;Property=PresentValue;Value=2;Delay=1;Quit=FALSE;
: REFERENCE )*)
```

The following sample illustrates the initialization of the ListOfGroupMembers Property of the Group object.

```
fbGroupAv:  FB_BACnet_AnalogValue;
fbGroupBv : FB_BACnet_BinaryValue;
fbGroup   : FB_BACnet_Group;(* ~(BACnet_ListOfGroupMembers :
   Object=./
fbGroupAv;Property=PresentValue;Property=StatusFlags;Property=HighLimit;Property=LowLimit;
   Object=./fbGroupBv;Property=PresentValue;Property=StatusFlags;Property=OutOfService
: REFERENCE ) *)
```

The EventEnrollment object can be used to define alarm conditions that go beyond the conditions covered by Intrinsic Reporting. The following section demonstrates the initialization of the EventEnrollment objects with the aid of an abbreviated syntax:

```
fbEEAv   : FB_BACnet_AnalogValue;
fbEEWarn : FB_BACnet_EventEnrollment;(* ~
   (BACnet_ObjectPropertyReference : ./fbEEAv : REFERENCE_PresentValue)
   (BACnet_EventParameters     : EventType=out_of_range;timeDelay=5;lowLimit=0;highlimit=10;deadband
=0;:REFERENCE )
   (BACnet_NotificationClass       : 10 : nolink )
   (BACnet_EventMessageTexts       : {Warning;Sensorfault;Normal operation} : nolink )
*)
```

```
fbEEBo   : FB_BACnet_BinaryOutput;
bEEHours : FB_BACnet_EventEnrollment; (* ~
   (BACnet_ObjectPropertyReference : ./fbEEBo : REFERENCE_ElapsedActiveTime)
   (BACnet_EventParameters     : EventType=unsigned_range;highlimit=60; :REFERENCE )
   (BACnet_NotificationClass       : 10 : nolink )
   (BACnet_EventMessageTexts       : {Maximum operating hours exceeded; ; } : nolink )
*)
```

Integration of hardware modules

The connection of hardware modules can basically be done via IO automapping and the assignment to PLC objects by specifying ObjectIdentifiers in the PLC code. To enable efficient code generation of BACnet projects, it is also possible to configure the hardware connection directly behind the declaration of BACnet objects in the PLC. The LINKPATH option exists for this purpose. With this option it is possible to link a BACnet property variable of a BACnet object directly to an IO module. In this case, no link to a PLC variable is created, but a pure device-device link.

The following sample shows the linking of a BACnet object BinaryInput with the 3rd terminal (of a KL1002) on the 1st channel. The hardware value can then be accessed directly in the PLC program via the PresentValue of the FB_BACnet_BinaryInput.

```
bi : FB_BACnet_BinaryInput;(* ~{ BACnet_RawIoBinaryBoolValue : TIID^Rt-
Ethernet^BK9000^Term 3 KL1002^Channel 1^Input : LINKPATH } *)
```

By specifying an additional parameter of the LINKPATH option, Offset1, Offset2 and the bit size for the link can also be specified directly (LINKPATH<[Offset1],[Offset2],[Bit size]>)... If the bit size is set to the value 0, the maximum possible number of bits is automatically linked. Without specifying the additional parameters, the value 0 is used for Offset1 and Offset2. Offset1 refers to the BACnet property variable and Offset2 to the external link.

```
bx : FB_BACnet_BinaryInput;(* ~{ BACnet_RawIoBinaryBoolValue : TIID^Rt-
Ethernet^BK9000^Term 3 KL1002^Channel 1^Input : LINKPATH<0,0,1> } *)
```

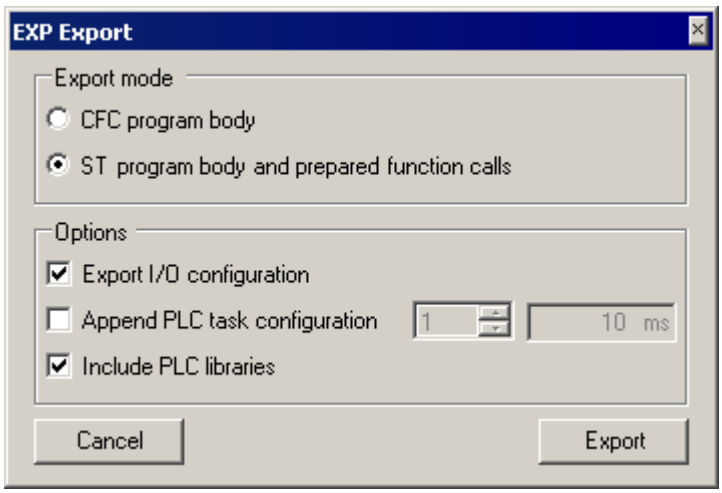
In connection with function blocks, sub-elements of FB instances can also be directly connected to a hardware module. The instance path syntax "<[]>" after the BACnet property to be linked can be used for this.

```
PROGRAM SIMPLE
VAR
s1 :FB_BACnet_Sub; (* ~ { BACnet_RawIoBinaryBoolValue<sub1/bi>  : TIID^Rt-
Ethernet^BK9000^Term 3 KL1002^Channel 1^Input : LINKPATH }
            { BACnet_RawIoBinaryBoolValue<sub2/bi>  : TIID^Rt-
Ethernet^BK9000^Term 3 KL1002^Channel 2^Input : LINKPATH }
            { BACnet_RawIoBinaryBoolValue<sub1/bi2> : TIID^Rt-
Ethernet^BK9000^Term 2 KL1002^Channel 2^Input : LINKPATH }
            *)
END_VAR
```

In addition, PLC auto-mapping also allows variables to be linked outside of BACnet. With the option "ExtLink" any local PLC symbol can be linked to an external variable e.g. a hardware module. With the option "ExtLink2" 2 PLC-external variables can be linked together. Additional parameters (as with LINKPATH) can be used to specify Offset1, Offset2 and the bit size for the link. Below is a sample of how to use each of these options.

```
b1 AT%I*: BOOL;   (* ~{ ExtLink  : TIID^Rt-Ethernet^BK9000^Term 2 (KL1002)^Channel 2^Input  : } *)
bDummy  : BOOL;   (* ~{ ExtLink2 : TIID^Rt-Ethernet^BK9000^Term 5 (KL2012)^Channel 1^Output :
                   TIID^Rt-Ethernet^BK9000^Term 2 (KL1002)^Channel 2^Input }
            *)
b1 AT%I*: BOOL;   (* ~{ ExtLink<0;0;1>  : TIID^Rt-
Ethernet^BK9000^Term 2 (KL1002)^Channel 2^Input  : } *)
bDummy  : BOOL;   (* ~{ ExtLink2<0;0;1> : TIID^Rt-
Ethernet^BK9000^Term 5 (KL2012)^Channel 1^Output :
                     TIID^Rt-Ethernet^BK9000^Term 2 (KL1002)^Channel 2^Input }
            *)
```

The EXP export can be used as a starting point for linking hardware modules. In the EXP export configuration dialog, the "Export I/O configuration" option can be used to generate the BACnet object configuration for the TcBACnet.lib. Here, a corresponding program with the hardware BACnet objects and their link is created for each IO bus present in the project. The procedure EXP-Export, PLC Automapping then generates the same BACnet configuration that generates an IO automapping. The advantage of the PLC variant is that hardware objects can be integrated into structured function blocks of the PLC or hardware channels that are not to be visible via BACnet can be deleted.



PLC automapping for BACnet clients

To link PLC variables with process data properties of remote BACnet objects, Device ID can be specified as a comment, in addition to the object type and instance. In this case the PLC variables are linked with previously created BACnet objects (e.g. through scanning of clients). If a client does not yet exist, it will be created automatically. Objects that do not exist are also created. The hierarchy created is flat. StructuredView object for clients are currently not supported by PLC automapping.

In the following source text extract a variable of type REAL is created. During automapping it is linked with the property PresentValue of object AnalogValue:100 under the client with the BACnet-ID "42". The process data variable of the remote BACnet object is activated automatically.

```
av0 AT %I* : REAL;          (* ~ (BACnet_ObjectType          : AV     : NOLINK)
                    (BACnet_DeviceID           : 42     : NOLINK)
                    (BACnet_ObjectIdentifier       : 100    : NOLINK)
                    (BACnet_PresentValue       :        :)
                *)
```

Auxiliary properties exist for the configuration of the process data-specific parameters of client properties in order to be able to automatically configure all options that can be set in the System Manager. These are write priority for PresentValue properties, cycle time, COV increment, COV resubscription interval for COV subscriptions and flags for configuring whether data are to be processed cyclically or via COV or OnChange. The following source code extract illustrates the application of these auxiliary properties.

```
ao0 AT %Q* : REAL;          (* ~ (BACnet_ObjectType          : AO     : NOLINK)
                    (BACnet_DeviceID           : 42     : NOLINK)
                    (BACnet_ObjectIdentifier       : 0      : NOLINK)
                    (BACnet_RemotePriority     : 8      : )
                    (BACnet_RemoteCycleTimeWrite   : 1000   : )
                    (BACnet_RemoteFlagsWrite       : WRITEONCHANGE : )
                    (BACnet_PresentValue       :    : )
                *)

ao1 AT %I* : REAL;          (* ~ (BACnet_ObjectType          : AO     : NOLINK)
                    (BACnet_DeviceID           : 42     : NOLINK)
                    (BACnet_ObjectIdentifier       : 0      : NOLINK)
                    (BACnet_PresentValue       :    : )
                    (BACnet_RemoteCycleTimeRead    : 1000   : )
                    (BACnet_RemoteFlagsRead    : COV    : )
                    (BACnet_RemoteCovincrement     : 0,5    : )
                    (BACnet_RemoteResubscriptionInterval: 60: )
                *)
```

For RemoteFlagsWrite, the following values can be used for output variable (%Q), i.e. from the configured controller to the remote device:

- onchange, periodic

For RemoteFlagsRead - input variable (%I) - reading from a remote device the following applies:

- cov, covpolling, polling, confirmedcov, rpm-o, rpm-c

If no auxiliary properties are specified for remote objects, the default settings of the System Manager are used:

- RemotePriority : 16
- RemoteFlagsRead : 1 (COV)
- RemoteFlagsWrite : 1 (OnChange)
- RemoteCycleTimeRead : 1000
- RemoteCycleTimeWrite : 1000
- RemoteResubscriptionInterval : 240
- RemoteCovIncrement : 0.0

If the remote-blocks of TcBACnet.lib are used, the process data transfer type can also be specified during FB instantiation. The remote configuration parameters refer to all activated BACnet properties. In the following sample, the transmission type of the process data is set to ConfirmedCOV for the properties of the *EX function block: *StateOfChangeCount*, *PresentValue*, *StatusFlags*, *EventState* and *Reliability*.

```
bvRemote : FB_BACnet_RemoteBinaryValue_EX; (* ~ (BACnet_ObjectType : BV : NOLINK)
                        (BACnet_DeviceID   : 32 : NOLINK)
                        (BACnet_ObjectIdentifier : 20 : NOLINK)
                        (BACnet_RemoteFlagsRead : ConfirmedCOV : )
                        (BACnet_RemoteResubscriptionInterval : 3600 : )
                        *)
```

For priority-based properties (PresentValue) a write priority can be configured, in order to enable access to higher priorities of the PriorityArray. As for servers, the value NULL can be written to the associated PriorityArray in order to disable this priority level. For analog and binary objects this is done in the same way as for servers (value > 1 for binary objects; NaN for analog objects). For AnalogOutput objects the Min/MaxPresValue range is not monitored, which means that NULL can only be generated via NaN. In contrast to servers, for multistate objects only writing of 0 results in NULL in the PriorityArray. For performance reasons the property NumberOfStates is not read by the remote device.

In conjunction with the TwinCAT BACnet PLC library "TcBACnet.lib" an efficient option for using "scanned" client configurations in the PLC is available. Once the clients to be used have been added to the System Manager (e.g. through scanning of a BACnet network), the variable declaration and associated annotation with automapping comments can be generated by specifying a destination file with the extension "EXP" in the Export option of the Settings tab in the BACnet device. This can then be imported in a PLC project in PLC Control. The basis BACnet FBs of the library are used (e.g. "FB_BACnet_RemoteAnalogInput").

Remap

When using persistent data, subsequent extensions pose a challenge in terms of establishing an unambiguous relationship between the persistent data and their counterparts in the PLC program. Persistent data are unambiguously assigned via the ObjectIdentifier. If an extension involves adding an object in a program, subsequently generated BACnet ObjectIdentifiers may be shifted, thereby corrupting the allocation of the persistent data. The Remap function was implemented to resolve this issue. Remap can be run on an already mapped PLC project that is subject to modifications/extensions.

The PLC project is examined for modifications and:

- New objects are added (in order to avoid collisions with old projects, new ObjectIdentifiers are allocated that are higher than the largest previously generated instance number, see "System Manager Settings")
- Modified property initialization are written (unchanged initializations are not written again)
- Remote PLC objects that were previously generated through PLC automapping are removed
- Attention during renaming: when moving objects in the tree, property values may be lost. (e.g. by changing the StructuredViewPath or symbol name)

Appendix

- Overview object types for the "BACnet_ObjectType" : AI,AO,AV,BI,BO, BV, CAL,CMD,DEV,FILE,GROUP,LOOP,MI,MO,NC,PROG,SCHED,MV,TLOG

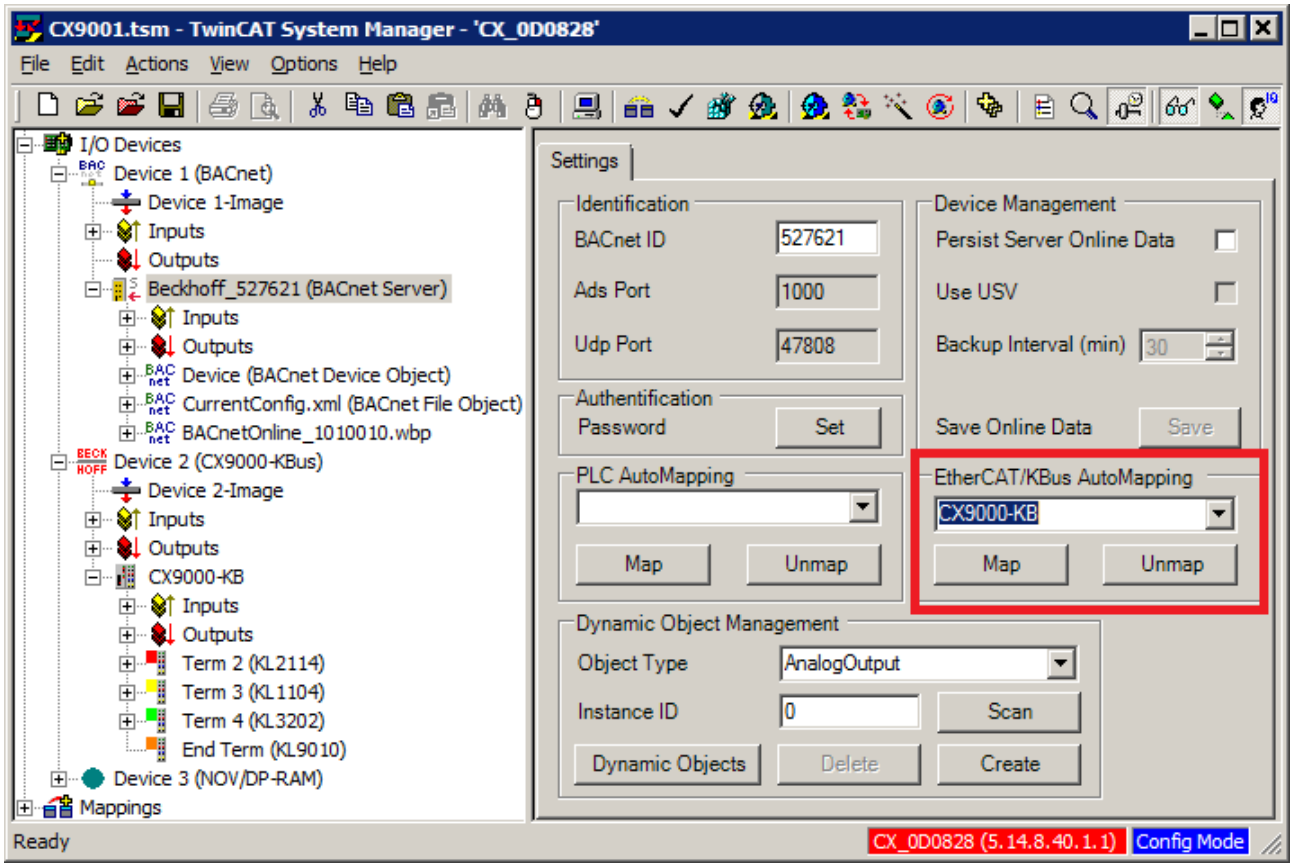
- Note regarding NOLINK: the comment "NOLINK" is used to identify BACnet properties that are not to be linked with a variable, if multiple annotations exist for the variable. For selected BACnet properties "NOLINK" is set automatically, in cases where linking would not be meaningful. The following properties are implicitly annotated with "NOLINK": ObjectIdentifier, ObjectType and all remote* properties of the client-specific configuration.
- In TwinCAT BACnet/IP an option was added to enable the property *ObjectIdentifier* as process data. Since this property is automatically treated as NOLINK, the LINK option was introduced for compatibility reasons, which overrides NOLINK for *ObjectIdentifier*. This means that this property can also be activated and linked as a process date via PLC automapping.
- Example hierarchical FBs: see sample "PLC Automapping [▶ 118]"

2.10 I/O automapping

TwinCAT BACnet/IP supports automated mapping of the modular IO systems K-bus and EtherCAT to BACnet. For each I/O channel a corresponding BACnet object of type BinaryInput, BinaryOutput, AnalogInput or AnalogOutput is configured automatically, process data are linked accordingly through Device2Device mapping, and the status of the I/O systems is transferred to the BACnet properties Reliability and StatusFlags, if applicable.

During mapping of I/O systems a distinction is made between I/O busses that consolidate several I/O modules and abstract an I/O strand in each case. By assigning IoBusNr BACnet I/O objects are allocated to an I/O bus, via which the status is mapped, for example. If, for example, a K-bus strand is linked, the process data BusState can be used to detect whether the K-bus is ready for operation, and the property Reliability of all BACnet objects with the respective IoBusNr can be adjusted to the value NO_FAULT_DETECTED, otherwise NO_SENSOR or NO_OUTPUT. With TwinCAT BACnet/IP the I/O busses can also be linked to a K-bus, e.g. a BK9000, or further EtherCAT strands can be linked with a BACnet controller.

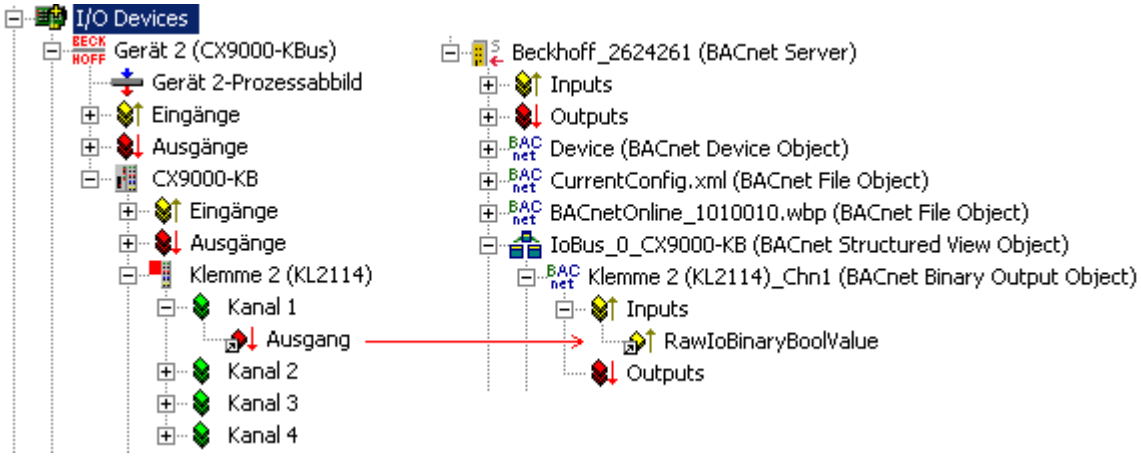
I/O bus automapping can be triggered via the "Settings" tab of a BACnet server. In the corresponding dialog box, an I/O bus can be selected and the link can then be initiated via "Map" button:



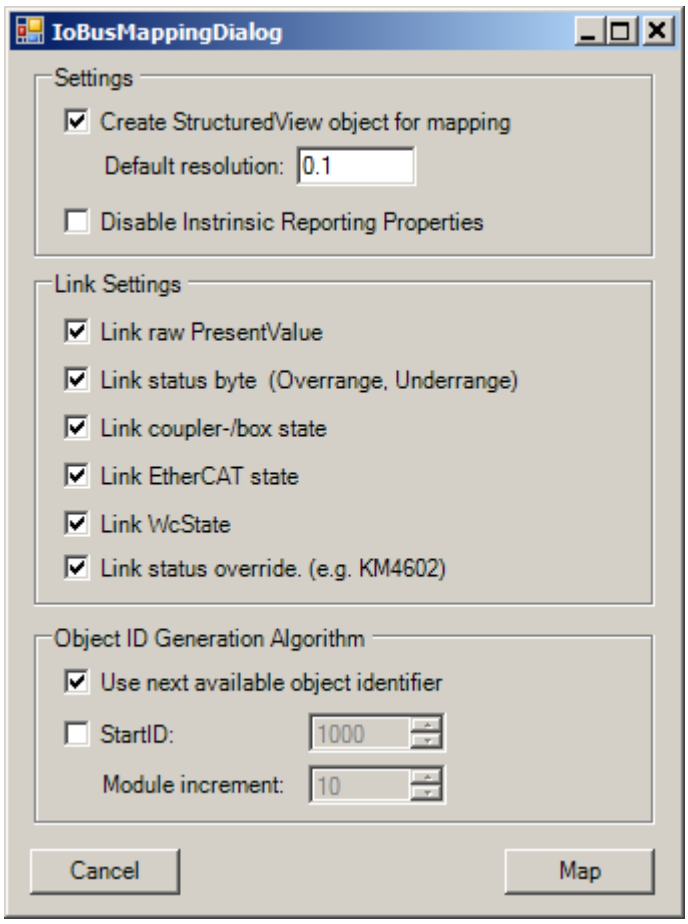
The following I/O bus types are currently supported:

- EtherCAT

- BK1120, BK1250
- K-BUS: CX1100-BK, CX5000-BK, CX-8000-BK
- BK9000, BK9100, BK9050



In general, during I/O automapping the System Manager analyses all terminals under an I/O bus, creates corresponding BACnet objects and links process data. As an example, the figure shows how a digital output terminal (KL2114) at a CX9001-KBus is mapped to BACnet. The BACnet object "Terminal 2 (KL2114)_Chn1" of type BinaryOutput was created, and the output process data variable "Output" was linked with the BACnet process data variable "RawIoBinaryValue". This link ensures that the status of the digital output signal always matches the PresentValue of the BACnet object, if the object is not OutOfService. The polarity of the "BinaryOutput" is also considered. For further details please refer to section "Process data [▶ 42]".



Automapping can be configured via an associated dialog, and the user can specify whether the bus status should be monitored automatically, or whether the process data can be linked automatically. In addition, the object ID allocation algorithm of the created BACnet objects can be specified. The following options are available:

- **"Use next available object identifier"** - For each new BACnet object the next free object ID for the respective object type is used, starting from 0.
- Module-based ID allocation - Based on a start ID the object ID for each new I/O module is incremented by a "Module increment". This can be used to ensure that the BACnet object ID can always be used to determine which terminal was linked. Example: With a start ID of 1000 and a module increment of 10, a BinaryInput object with ID 1113 is allocated to the third binary input channel of the eleventh terminal.

The other configuration options of I/O automapping are explained below:

- **"Create StructuredView object for mapping"** - All BACnet objects generated for I/O mapping are created below a newly generated StructuredView object, to increase the transparency within a project
- **"Default Resolution"** - Analog objects have a BACnet property resolution, which determines the scaling of I/O process data for the PresentValue of the BACnet object and vice versa. The parameter configured here is entered for all generated analog objects. To use the original hardware values, a resolution value of 0 should be set.
- **"Link raw PresentValue"** - To link the PresentValue of the BACnet objects with the I/O modules, the corresponding RawIoPresentValue properties of the BACnet objects must be activated and linked. This can be activated with this option.
- **"Link Status Byte"** - Analog input terminals can use a status byte to indicate whether the respective process data value is valid or whether a limit was exceeded, or a wire is broken, for example. If this option is activated, for analog input terminals (KX3XXX, EL3XXX) the data status of the terminal is linked with the BACnet I/O property "RawIoStatus". With EtherCAT the variables "Underrange" and "Overrange" are linked bitbased. If the corresponding status bits are set, the reliability is formed as follows during runtime:
  - If bit 0 is set, Reliability is UNDER_RANGE
  - Otherwise, Reliability is OVER_RANGE, if bit 1 is set
  - Otherwise, Reliability is UNRELIABLE_OTHER, if bit 6 is set
- **"Link coupler-/box state"** - In a K-bus or Bus Couplers (BK9000) the process data BusState/ couplerState and "BoxState" can be used to determine the communication and operating state of all terminals of the bus. Monitoring of the couplerState is realised centrally via the BACnet server. If this option is activated, available BusState/couplerState or "BoxState" are linked with the IoBusState process data of the BACnet server with the respective IoBusNr. The BusState of a K-bus is linked with the couplerState. If couplerState or "BoxState" for an I/O bus in the BACnet server is not equal 0, the reliability of all objects under the server with the corresponding IoBusNr is set to the value NO_SENSOR for input objects or NO_OUTPUT for output objects, provided the objects are not OutOfService.
- **"Link EtherCAT state"** - For EtherCAT, a mechanism similar to Coupler/Box state for K-bus is available. In contrast to the K-bus, with EtherCAT the state of each terminal can be determined, which could enable disconnected terminals to be detected. With EtherCAT the status identification is therefore not realised centrally via the server, but takes place in each object by linking the process data property "RawIoECATState". Is the value of "RawIoECATState" is not equal 8 (OP), the reliability of the object is set to the value NO_SENSOR for input objects or NO_OUTPUT for output objects, provided the objects are not OutOfService.
- **"Link Wc state"** - Monitoring of the EtherCAT Wc state is currently not supported.
- **"Link Status Override"** - For terminals with manual operating mode the respective manual operation status can be mapped in BACnet to the StatusFlags via the override bit. For the terminals KM4602 and KM2652 a link with the process data property "RawIoStateOverride" is generated, if this option is activated.

Automatic mapping of EtherCAT and K-bus devices to BACnet objects is not always possible. In certain cases, and with complex terminals it is in not always clear whether a BACnet object should be created for process data, or which BACnet object should be created. For many basic input/output terminals the mapping is based on the following algorithm:

- For all 1-series modules (KX-1XXX, EL-1XXX) a BACnet BinaryInput object is created for all input variables of type BIT that are found and the input data is linked if the variable name is not equal "WcState" or "InputToggle".
- For all 2-series modules (KX-2XXX, EL-2XXX) a BACnet BinaryOutput object is created for all output variables of type BIT that are found, and the process data is linked.

- For all 3-series modules (KX-3XXX, EL-3XXX) a BACnet AnalogInput object is created for all input variables of type INT16 that are found, and the process data is linked.
- For all 4-series modules (KX-4XXX, EL-4XXX) a BACnet AnalogOutput object is created for all output variables of type INT16 that are found, and the process data is linked.

The following terminals are treated in a special way:

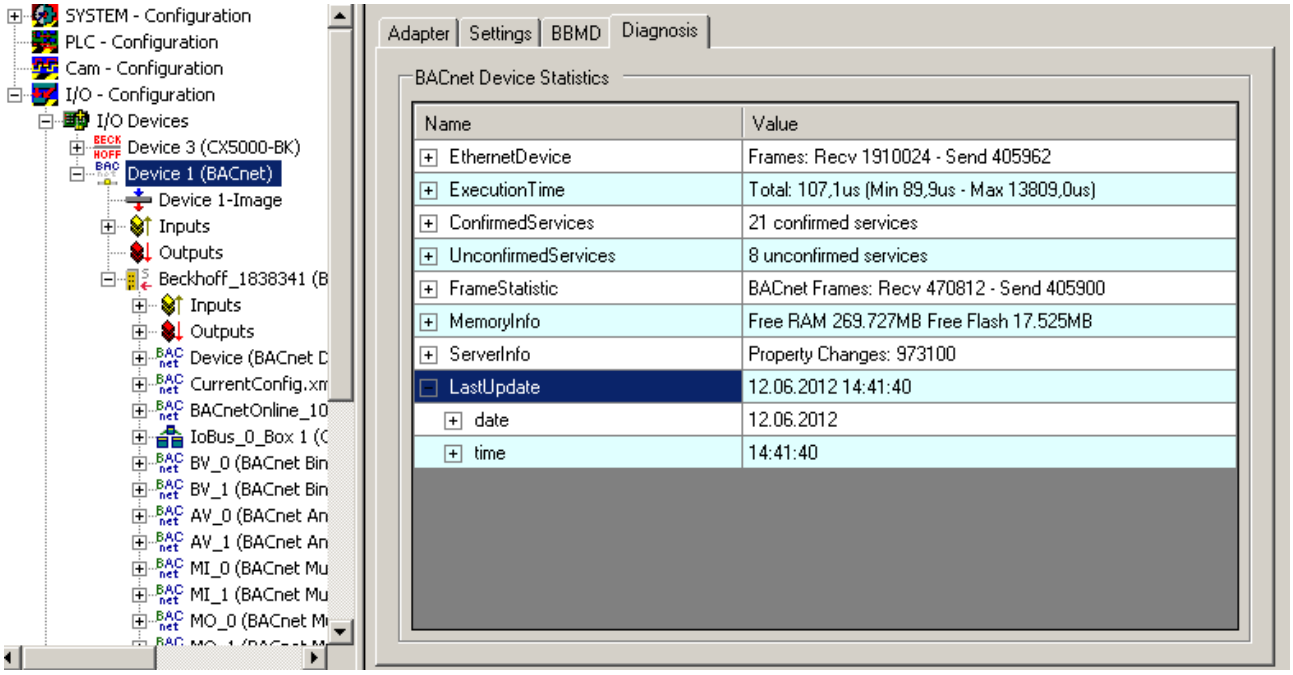
- KL1859 - 8 BinaryInput and 8 BinaryOutput BACnet objects are created.

With automapping the following BACnet properties are configured automatically:

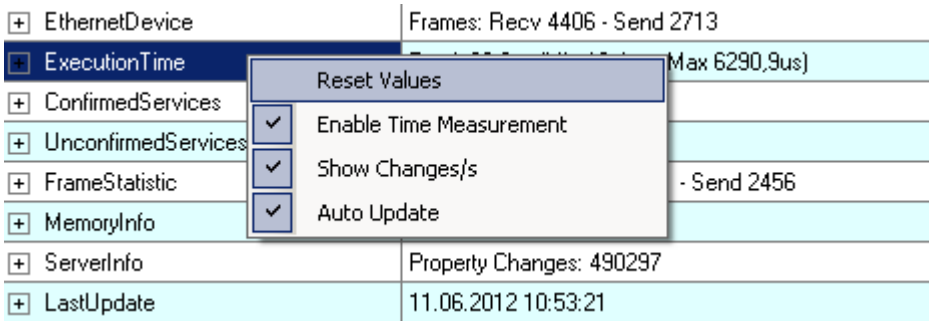
- ObjectIdentifier - is determined by the object ID allocation algorithm
- ObjectName - is formed from the terminal name plus "_ChnX" for the channel number
- IoBusNr - is incremented with each I/O automapping, starting from 0
- IoModuleNr - indicates the number of the linked terminal, starting from 1
- IoChannelNr - indicates the channel, starting from 1
- Resolution - as specified in the configuration dialog

2.11 Diagnosis

This section explains how BACnet configurations can be analyzed at runtime. Device status in the form of process data, and for BACnet clients the client status, are available for this purposes (see section Process data [▶ 42]). Further details can be analyzed via the "Diagnosis" tab of the BACnet device. This section describes the diagnosis via the "Diagnosis" tab and other troubleshooting strategies.



The diagnosis of the BACnet device is subdivided into several categories. These are described in detail in the following sections. Various options can be activated via a context menu:



"Auto Update" updates the diagnosis display in 1-second intervals. The category "Last Update" always shows the time at which the diagnosis was last read. The option "Enable Time Measurement" enables the category "ExecutionTime", through which the runtimes of the BACnet stack can be determined. For some values the option "Show Changes/s" activates an additional display showing the changes per second. It is formed over the last 10 read diagnostics and only works if "Auto Update" is activated. The option "Reset Values" resets all diagnostic values.

Diagnostic data in the PLC

The diagnostic data presented in this section can also be accessed from the PLC via an ADS interface. The function block FB_BACnet_GetDiagInfo offers convenient access.

EthernetDevice category

The EthernetDevice category enables the number of sent and received messages (frames) and any receive and send errors to be determined. These data are read directly from the network driver, which means that they are not reset when a BACnet configuration is reloaded, although the option "Reset Values" deletes these values. This category counts all frames that are processed via the network driver, including non-BACnet frames.

⊟ EthernetDevice	Frames: Recv 3727 - Send 2313
⊟ ethStat	(3727;2313)
sendFrames	2313 (0.00 Frames/s)
recvFrames	3727 (0.40 Frames/s)
⊟ txRxErrCnt	(0;0)
txCnt	0 (0.00 /s)
rxCnt	0 (0.00 /s)

The counter "sendFrames" indicates how many frames were used, "recvFrames" indicates the number of received frames. "txCnt" and "rxCnt" indicate the number of faulty received and sent frames. Unexpectedly high numbers may indicate network faults or overload (e.g. CRC errors, dropped frames).

ExecutionTime category

⊟ ExecutionTime	Total: 71,6us (Min 13,4us - Max 5898,7us)
ns100IoInCurExecutionTime	359
ns100IoInMinExecutionTime	78
ns100IoInMaxExecutionTime	53663
ns100IoOutCurExecutionTime	157
ns100IoOutMinExecutionTime	27
ns100IoOutMaxExecutionTime	2841
ns100CycleCurExecutionTime	200
ns100CycleMinExecutionTime	29
ns100CycleMaxExecutionTime	2483
ns100InputCurDistanceTime	8088
ns100InputMinDistanceTime	1912
ns100InputMaxDistanceTime	66939

The option "Enable Time Measurement" enables runtime information for TwinCAT BACnet/IP to be determined. For each runtime a minimum (Min), maximum (Max) and current (Cur) value is determined. All runtimes are determined with a resolution of 100 ns. The following values are available as runtimes:

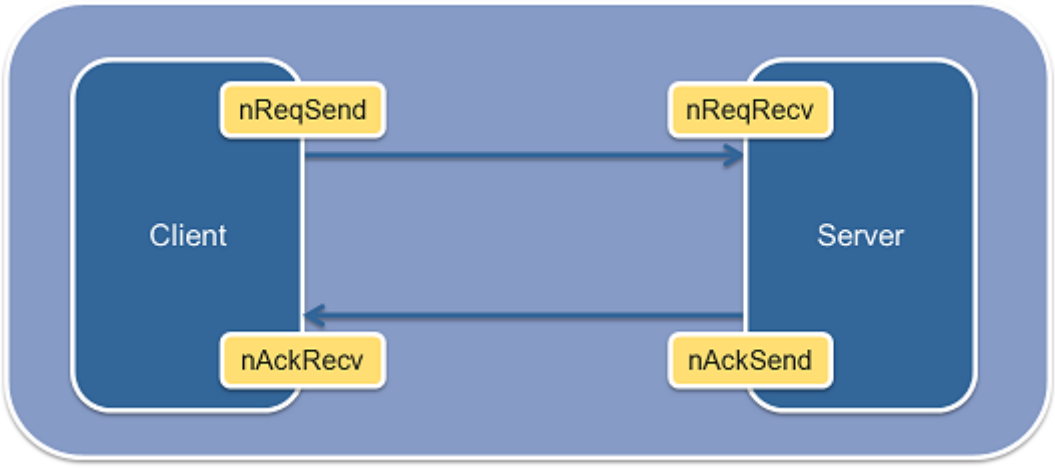
- **IoIn**XXXExecutionTime: determines the runtime of the function InputUpdate of the BACnet device, which also executes the InputUpdate functions of all BACnet objects. This is where process data are copied from BACnet to the PLC/IO, among other tasks. In addition, the IoBusStatus is checked in the BACnet server and updated in the BACnet objects. For BACnet remote objects ReadProperty requests und CovSubscriptions for activated property process data are sent from this context.

- **IoOut**XXXExecutionTime: determines the runtime of the function OutputUpdate of the BACnet device, which also executes the OutputUpdate functions of all BACnet objects. In objects process data are copied from the PLC/IO and compared for changes. If a change has occurred, EventState machines can be executed and COV and event messages may be sent, depending on the BACnet property. For BACnet remote objects, write property requests are sent for activated property process data and detected in case of "WriteOnChange" modification of relevant properties (comparison).
- **Cycle**XXXExecutionTime: determines the runtime of the function CycleUpdate of the BACnet device, which also executes the CycleUpdate functions of all BACnet objects. In the CycleUpdate context:
  - device, client, server state machines are executed,
  - received frames are processed,
  - Object EventState machines may be executed (in the event of property modifications),
  - COV and event messages are sent,
  - ADS requests are processed,
  - segmented frames are processed,
  - and persistent data are written.
- InputXXX**Distance**Time: indicates the interval between two calls of function InputUpdate of the BACnet device. In the event of "Exceeds" of the BACnet task this can be used to determine at which maximum interval the BACnet stack is executed.

In the top row of the category Execution Time the values IoIn, IoOut and cycle XXXExecution time are consolidated in the form of minimum, maximum and current values (in microseconds).

ConfirmedServices category

This category can be used to determine statistics for sent and received (acknowledged) BACnet messages. Acknowledged services operate in BACnet based on the client-server principle. A client sends a request, the server receives the request, processes it and sends a response (acknowledgement). The client receives the response and can process it further accordingly. Depending on the services a TwinCAT BACnet/IP device can act as a client or a server. The following figure shows the procedure for acknowledged services.



In the ConfirmedServices category, a list with the number of sent and received requests and sent and received responses is maintained for each supported service. The list shows how many requests/responses were processed successful and how many errors occurred. The 8 values for each BACnet service shown in the screenshot below are derived from these combinations. The top row of each service shows the total number of successful requests and responses and errors.

SubscribeCOVProperty	Req 2232/136 - Ack 1602/630
nReqSend	2232 (0.00 Frames/s)
nReqSendFail	136 (0.00 Frames/s)
nReqRecv	0 (0.00 Frames/s)
nReqRecvFail	0 (0.00 Frames/s)
nAckSend	0 (0.00 Frames/s)
nAckSendFail	0 (0.00 Frames/s)
nAckRecv	1602 (0.00 Frames/s)
nAckRecvFail	630 (0.00 Frames/s)

In the classification of BACnet services a distinction is made between so-called A-services and B-services, depending on which role (client or server) a device assumes for a service. If a device acts as a client, i.e. if it uses a request and receives a response (Ack), it uses an A-service. If a device receives a request, processes it and sends a response (Ack), it provides the B-service. If, for example, a client reads the value of a property of another device, the service ReadProperty-A is used. The remote terminal, the server, implements the ReadProperty-B service in this case. For diagnostics it is important to distinguish between A and B services. The following table is therefore subdivided into these service categories.

For acknowledged services an A-service is always associated with the counters ReqSend and AckRecv, a B-service always with ReqRecv and AckSend. The table indicates for the potential errors in which cases the error counter was incremented (e.g. ReqSendFail>0). In A-services an important error indicator may be a mismatch between the number of sent requests and received responses (Ack). Thus error type is referred to as (Req<>Ack).

Service diagnosis

With BACnet it should be noted that increased error counters in the diagnosis not necessarily indicate a misconfiguration. In BACnet networks short load peaks and lost messages are not uncommon. Suitable repetition mechanisms have been implemented to deal with this (ApduRetries and ApduTimeout). However, if the error rate is excessive it is advisable to find the cause.

The following table provides an overview of relevant BACnet services and their diagnosis. The section explains which function the services perform, where the services are used in TwinCAT BACnet/IP, which possible causes of errors can be diagnosed and possible solutions.

AcknowledgeAlarm-B (ReqRecv, AckSend)	**Use:** Acknowledging pending alarms **Possible errors:** nReqRecvFail>0: incorrect request parameters (wrong EventState, wrong Timestamp, Instrinsic Reporting for the object is disabled) nAckSendFail: no frame memory, no router memory **Error handling:** Check remote terminal
ConfirmedCOVNotification-A (ReqSend, AckRecv)	**Use:** Remote terminal has subscribed to change of value. Service sends message in the event of change. **Possible errors:** ReqSendFail>0 : no frame memory, no router memory, no available Invoke-IDs Req <> Ack: remote terminal not available or overloaded **Error handling:** Check/increase COVIncrement property for analog* objects. Reduce BACnet cycle time: for writing process data fewer modifications are then generated. Check configuration for continuously changing properties. Check whether remote terminal can use non--acknowledged COV subscriptions.
ConfirmedCOVNotification-B (ReqRecv, AckSend)	**Use:** Receipt of a change of value

	Possible errors: nReqRecvFail>0: faulty request parameters (process ID not available, unknown object, unknown property, message does not contain the expected data type), no router memory **Error handling:** Check remote terminal
ConfirmedEventNotification-A (ReqSend, AckRecv)	**Use:** Message for change in event state (Instrinsic Reporting) **Possible errors:** ReqSendFail>0: no frame memory, no router memory, no available Invoke-IDs Req <> Ack: remote terminal not available or overloaded **Error handling:** Check whether sending of event notifications is meaningful for all objects. Perhaps deactivate Instrinsic Reporting properties, deactivate event notification via property EventEnable, or allocate a NotificationClass without recipients to all objects that should not send event notifications. Perhaps use unconfirmed event notifications (issueConfirmedNotifications in NotificationClass:RecipientList.)
ConfirmedEventNotification-B (ReqRecv, AckSend) *Not covered by certification*	**Use:** Receipt of event messages in the notification sink **Possible errors:** nReqRecvFail>0: faulty request parameters (process ID not available, receive buffer for event notifications full), no router memory **Error handling:** check remote terminal, increase buffer size for EventNotifications, increase delete interval for message buffer
GetAlarmSummary-B (ReqRecv, AckSend)	**Use:** Call up summary of all objects in alarm state **Possible errors:** nReqRecvFail>0: no router memory nAckSendFail: no frame memory, no router memory
GetEnrollmentSummary-B	**Use:** Call up a summary of event-specific information (with comprehensive filter options) **Possible errors:** nReqRecvFail>0: no router memory nAckSendFail: no frame memory, no router memory
SubscribeCOV-A (ReqSend, AckRecv)	**Use:** Service for subscribing to a change of value (COV) of properties "PresentValue" and "StatusFlags" for analog*, binary* and MultiState* objects. With remote objects for process data in COV mode. TrendLog for remote objects with LogInterval=0. COV subscriptions configured in NotificationSink. **Possible errors:** ReqSendFail>0: connection data cannot be resolved (client status not Complete), no frame memory, no router memory, no available Invoke IDs Req <> Ack: remote terminal not available or overloaded. **Error handling:** Activate optimization of the process data in the "PropertyWizzard", increase cycle times, subscribe to fewer properties, read properties cyclically that cannot be subscribed to via COV (polling)
SubscribeCOV-B	**Use:**

(ReqRecv, AckSend)	Receipt of a subscription to a change of value (COV) of properties "PresentValue" and "StatusFlags" for analog*, binary* and MultiState* objects. **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unsupported property, invalid array index), maximum number of supported COV subscriptions for object reached, no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase the number of supported COV subscriptions per object in Advanced Parameter [▶ 531] (MAX_OBJ_SUBS_COV_ENTRIES)
AtomicReadFile-B (ReqRecv, AckSend)	**Use:** Reading a file **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unsupported object, invalid FileStart position, access via record), no router memory nAckSendFail: no frame memory, no router memory, local segment buffer full, too many data for supported segmentation of the remote terminal **Error handling:** Increase Advanced Parameter [▶ 531] MAX_RECV_BAC_SEGM_FRAMES
AtomicWriteFile-B (ReqRecv, AckSend)	**Use:** Writing a file **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unsupported object, invalid FileStart position, access via record), no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase Advanced Parameter [▶ 531] MAX_RECV_BAC_SEGM_FRAMES
AddListElement-B (ReqRecv, AckSend)	**Use:** Adding a list element **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unsupported property, property is not a list, property is read-only), property memory exceeded, no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase property list memory (LIST_ALLOC_MEM_BYTES). Some lists are managed in TwinCAT BACnet/IP as arrays (e.g. RecipientList, DateList). Increase MAX_PROPERTY_ARRAYELEMENTS.
RemoveListElement-B (ReqRecv, AckSend)	**Use:** Removing a list element **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unsupported property, property is not a list, property is read-only, element not available), no router memory nAckSendFail: no frame memory, no router memory
ServiceCreateObject-B (ReqRecv, AckSend)	**Use:** Generating a dynamic object

	Possible errors: nReqRecvFail>0: faulty request parameters (unsupported object type, invalid property parameter data, object name or ObjectIdentifier already allocated, maximum number of objects exceeded), no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase Advanced Parameter [▶ 531] MAX_OBJECTS
ServiceDeleteObject-B (ReqRecv, AckSend)	**Use:** Unloading a dynamic object **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, static object - only dynamic objects can be deleted), no router memory nAckSendFail: no frame memory, no router memory
ReadProperty-A (ReqSend, AckRecv)	**Use:** Reading a property of a remote device **Possible errors:** ReqSendFail>0 : connection data cannot be resolved (client status not Complete), no frame memory, no router memory, no available Invoke IDs Req <> Ack: remote terminal not available or overloaded. **Error handling:** Activate optimization of the process data in the "Property Wizzard. Increase cycle times. Request fewer properties during polling: use SubscribeCOV service
ReadProperty-B (ReqRecv, AckSend)	**Use:** Reading a local property **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unknown property, no read access, invalid array index), no router memory nAckSendFail: no frame memory, no router memory, local segment buffer full, too many data for supported segmentation of the remote terminal
ReadPropertyMultiple-B (ReqRecv, AckSend)	**Use:** Reading several local objects and properties with one access **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown objects, unknown properties, no read access, invalid array index), no router memory nAckSendFail: no frame memory, no router memory, local segment buffer full, too many data for supported segmentation of the remote terminal **Error handling:** in the event of excessive data, the remote terminal should use ReadProperty. Perhaps read list and array properties for each index. Increase Advanced Parameter [▶ 531] MAX_SEND_BAC_SEGM_FRAMES
WriteProperty-A (ReqSend, AckRecv)	**Use:** Writing a property of a remote device **Possible errors:** ReqSendFail>0: connection data cannot be resolved (client status not Complete), no frame memory, no router memory, no available Invoke IDs Req <> Ack: remote terminal not available or overloaded.

	Error handling: Activate optimization of the process data in the "PropertyWizzard. Increase cycle times. Subscribe to fewer properties. Activate WriteOnChange in the process data configuration (with WriteOnChange a maximum of WriteProperty per client property "cycle time" is executed.)
WriteProperty-B (ReqRecv, AckSend)	**Use:** Writing a local property **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unknown property, no write access, invalid array index, wrong data type), no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase property list memory (LIST_ALLOC_MEM_BYTES). Increase Advanced Parameter [▶ 531] MAX_PROPERTY_ARRAYELEMENTS. Perhaps remove write protection ("Optional Properties" tab). Examine remote terminal!
WritePropertyMultiple-B (ReqRecv, AckSend)	**Use:** Writing several local objects and properties with one access **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown objects, unknown properties, no write access, invalid array index, wrong data type), no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase property list memory (LIST_ALLOC_MEM_BYTES). Increase Advanced Parameter [▶ 531] MAX_PROPERTY_ARRAYELEMENTS. Perhaps remove write protection ("Optional Properties" tab). Use WriteProperty. Examine remote terminal!
DeviceCommunicationControl-B (ReqRecv, AckSend)	**Use:** Communication control of a device (switching off message processing) **Possible errors:** nReqRecvFail>0: faulty request parameters (invalid password), no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Synchronize password configuration in remote terminal with password (BACnet server "Authentication"). Check password encoding.
ReinitializeDevice-B (ReqRecv, AckSend)	**Use:** Initiates a device restart or changes the device status to backup or restore mode. **Possible errors:** nReqRecvFail>0: faulty request parameters (invalid password, device is not in operational state, device cannot restart since it is in backup or restore state), no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Synchronize password configuration in remote terminal with password (BACnet server "Authentication"). Check password encoding. Some functions of ReinitializeDevice-B (backup, restore, restart) function only in RUN mode. Check whether enough data memory is available for backup/restore.

ReadRange-B (ReqRecv, AckSend)	**Use:** Reading of lists (in particular the property LogBuffer) **Possible errors:** nReqRecvFail>0: faulty request parameter (unknown object, unknown property, property is not a list, attempt to access unsupported properties with parameter "TimeRange" and "BySequence"), no router memory nAckSendFail: no frame memory, no router memory
SubscribeCOVP-A (ReqSend, AckRecv)	**Use:** Service for subscribing to a change of value (COV-P) with any objects and properties. With remote objects for process data in COV mode. For TrendLog for remote objects with LogInterval=0. COV subscriptions configured in NotificationSink. **Possible errors:** ReqSendFail>0 : connection data cannot be resolved (client status not Complete), no frame memory, no router memory, no available Invoke IDs Req <> Ack: remote terminal not available or overloaded. **Error handling:** Activate optimization of the process data in the "PropertyWizzard. Increase cycle times. Subscribe to fewer properties. Read properties cyclically that cannot be subscribed to via COV-P (polling).
SubscribeCOVP-B (ReqRecv, AckSend)	**Use:** Receiving of a subscription for a change of value (COV-P) with any objects and properties. **Possible errors:** nReqRecvFail>0: faulty request parameters (unknown object, unsupported property, invalid array index), maximum number of supported COV subscriptions for object reached, no router memory nAckSendFail: no frame memory, no router memory **Error handling:** Increase the number of supported COV subscriptions per object in the Advanced Parameters [▶ 531] menu (MAX_OBJ_SUBS_COV_ENTRIES).
GetEventInformation-B (ReqRecv, AckSend)	**Use:** Call up summary of all objects in event state **Possible errors:** nReqRecvFail>0: no router memory nAckSendFail: no frame memory, no router memory

UnconfirmedServices category

Unlike confirmed services, unconfirmed services are unidirectional. A client will therefore send request, and a server receives it. The following table provides a detailed description of the unconfirmed services of the BACnet diagnosis.

IAm-A (ReqSend)	**Use:** Announcing the server in the network **Possible errors:** ReqSendFail>0: no frame memory, no router memory
IAm-B (ReqRecv)	**Use:** Announcing a device from the network **Possible errors:** nReqRecvFail>0: DeviceAddressBinding list is full, no router memory

	Error handling: Increase the number of supported AddressBindings in the Advanced Parameters [▶ 531] (MAX_DEVICE_BINDINGS)
IHave-A (ReqSend)	**Use:** Reports an object that was searched in the network **Possible errors:** ReqSendFail>0: no frame memory, no router memory
UnconfirmedCOVNotification-A (ReqSend)	**Use:** Remote terminal has subscribed to change of value. Service sends unconfirmed message in the event of change. **Possible errors:** ReqSendFail>0: no frame memory, no router memory **Error handling:** Check/increase COVIncrement property for analog* objects. Reduce BACnet cycle time: for writing process data fewer modifications are then generated. Check configuration for continuously changing properties.
UnconfirmedCOVNotification-B (ReqRecv)	**Use:** Unconfirmed receipt of a change of value **Possible errors:** nReqRecvFail>0: faulty request parameters (process ID not available, unknown object, unknown property, message does not contain the expected data type), no router memory
UnconfirmedEventNotification-A (ReqSend)	**Use:** Unconfirmed message for change in event stage (Instrinsic Reporting) **Possible errors:** ReqSendFail>0: no frame memory, no router memory **Error handling:** Check whether sending of event notifications is meaningful for all objects. Perhaps deactivate Instrinsic Reporting properties, deactivate event notification via property EventEnable, or allocate a NotificationClass without recipients to all objects that should not send event notifications.
UnconfirmedEventNotification-B (ReqRecv)	**Use:** Unconfirmed receipt of event notifications in the notification sink **Possible errors:** nReqRecvFail>0: faulty request parameters (process ID not available, receive buffer for event notifications full), no router memory
TimeSynchronisation-B (ReqRecv)	**Use:** Receipt of a timestamp for time synchronization **Possible errors:** nReqRecvFail>0: system time could not be synchronized since synchronization is already running or TwinCAT is in free-run state, no router memory **Error handling:** Switch TwinCAT to the "Run" state
WhoHas-B (ReqRecv)	**Use:** Search for certain objects in a network **Possible errors:** nReqRecvFail>0: device does not process notifications since this function was switched off via the service "DeviceCommunicationControl", no server created, no router memory
WhoIs-A	**Use:**

(ReqSend)	Scanning of a network for devices and querying of the device communication parameters **Possible errors:** ReqSendFail>0: no frame memory, no router memory
WhoIs-B (ReqRecv)	**Use:** Querying the server for device communication parameters **Possible errors:** nReqRecvFail>0: device does not process notifications since this function was switched off via the service "DeviceCommunicationControl", no server created, no router memory
UTCTimeSynchronisation-A (ReqSend)	**Use:** Sending a UTC timestamp for time synchronization **Possible errors:** ReqSendFail>0: no frame memory, no router memory
UTCTimeSynchronisation-B (ReqRecv)	**Use:** Receipt of a UTC timestamp for time synchronization **Possible errors:** nReqRecvFail>0: system time could not be synchronized since synchronization is already running or TwinCAT is in free-run state, no router memory **Error handling:** Switch TwinCAT to the "Run" state

FrameStatistic category

This category consolidates counters of sent and received BACnet messages. In addition the system can detect how often during operation the physical link of the network adapter was lost.

FrameStatistic	BACnet Frames: Recv 3704 - Send 3024
nSegmSendTimeouts	0
nFrameAllocFailCnt	143 (0.00 Frames/s)
nFrameSendCnt	3024 (2.40 Frames/s)
nFrameSendFailCnt	143 (0.00 Frames/s)
nFrameRecvCnt	3704 (2.40 Frames/s)
nFrameRecvFailCnt	0 (0.00 Frames/s)
nLinkStatusChangedCnt	1

The individual counters are described in more detail below, and possible problem indicators are listed.

nSegmSendTimeouts	**Purpose:** Indicates how often timeouts occurred during waiting for confirmation of a segment. **Problem indicator:** the remote terminal may be overloaded or no longer reachable.
nFrameAllocFailCnt	**Purpose:** Indicates how often no memory for sending of BACnet messages could be allocated. **Problem indicator:** The number of messages sent per cycle may be too high. It may be possible to activate the network adapter queues (see Advanced parameters [▶ 531]). Check whether a connection to the network exists. A high value for this counter typically indicates a local overload situation.
nFrameSendCnt	**Purpose:** Indicates how many BACnet messages were sent successfully.

nFrameSendFailCnt	**Purpose:** Indicates how many BACnet messages could not be sent. **Problem indicator:** The number of messages sent per cycle may be too high. It may be possible to activate the network adapter queues (see Advanced parameters [▶ 531]). Check whether a connection to the network exists. A high value for this counter typically indicates a local overload situation.
nFrameRecvCnt	**Purpose:** Indicates how many BACnet messages were received without error.
nFrameRecvFailCnt	**Purpose:** Indicates how many BACnet messages were received with error. This counter indicates the total of the AckRecvFail and ReqRecvFail counters for all services. **Problem indicator:** The problem should be analyzed in more detail via the categories ConfirmedServices and UnconfirmedServices.
nLinkStatusChangedCnt	**Purpose:** Indicates how often the LinkStatus of the network adapter has changed. **Problem indicator:** Normally this counter has the value 1 (link on system startup.)

MemoryInfo category

The MemoryInfo category consolidates memory-specific information. In the top row of Windows CE devices the available main and flash memory is indicated. The individual counters are explained in the following table.

nAmsAlloc **nAmsAllocCntMax** **nAmsAllocFailCnt**	**Purpose:** These counters represent the allocation behavior of the BACnet stack for the router. nAmsAlloc indicates the current number of allocations. nAmsAllocCntMax indicates the maximum number of activated allocations at one time. nAmsAllocFailCnt indicates how often it was not possible to allocate memory. These counters do not provide information about the size of the allocated memory. **Problem indicator:** Router memory is used for temporary operations, e.g. for processing of messages and for saving EventNofitications and COV notification in the NotificationSink. A consistently high counter nAmsAlloc indicates an overload situation.
nCeAvailPhysMemBytes **nCeAvailFlashMemBytes**	**Purpose:** These counters are only implemented for Windows CE. They indicate how many RAM and flash memory is available. **Problem indicator:** If not enough RAM (nCeAvailPhysMemBytes) is available, it may not be possible to allocate dynamic objects. Flash memory is important for saving persistent data and for backup and restore operations.
nRecvFrameFifoSize **nEmptyFrameFifoSize**	**Purpose:** If network adapter queues are used, these counters indicate the available list memory for received (nRecvFrameFifoSize) and to be sent messages (nEmptyFrameFifoSize).

	Problem indicator: If the memory is inadequate, the size can be increased via the parameter dialog of the network adapter queues.
nRecvSegmUDPFramesListSize **nRecvSegmFramesListSize** **nSendSegmFramesListSize**	**Purpose:** These counters provide information about the messages segments currently stored intermediately. A distinction is made between IP segmentation (UDP) and BACnet NPDU segmentation, and between segments that are earmarked for delivery (Send) and received segments (Recv). **Problem indicator:** In the event of problems with the segmentation, the Advanced Parameters [▶ 531] MAX_RECV_BAC_SEGM_FRAMES and MAX_SEND_BAC_SEGM_FRAMES can be adjusted as required.
nClientScanListSize	**Purpose:** Indicates the size of the list of found devices in the network.
nAdsRequestListSize	**Purpose:** ADS requests to the BACnet stack are stored intermediately in a list and processed in the context of the BACnet task. This counter indicates the current number of pending ADS requests.

ServerInfo category

A detailed description of the category ServerInfo can be found in section Persistent data [▶ 57].

ClientInfo category

In this category information on the current communication state for each configured BACnet client is displayed.

instanceNumber	**Purpose:** Contains the device ID (instance number of the device object) of the BACnet client as assignment feature.
ipAddr udpPort networkNumber	**Purpose:** Contains information on communication addresses. BACnet only uses the device ID to identify devices. The IP address can be allocated dynamically. The current IP address can be determined via this diagnostic field. In addition, the UDP port and the network number (for communication via BACnet router, e.g. MS/TP devices) can be read here.
isReachable clientState	**Purpose:** ClientSTMState reflects the current state of the client state machine**.** ClientSTMState can take the following values: Init, Connecting, WaitingForConnection, RequestDeviceInformation, Complete. The operating principle of the ClientState machine is described in the Client Control section. ClientState indicates whether a connection to the remote BACnet device could be established. ClientState is TRUE if the ClientSTMState has reached the value "Complete". **Problem indicator:** If a BACnet device cannot be reached, the ClientState machine remains in state "*WaitingForConnection*". In this case, the reason why the BACnet client is unavailable should be ascertained: • Do the subnet masks of all devices match (e.g. 255.255.255.0)? This is an important prerequisite in BACnet, since it is used to calculate the broadcast addresses used for device searches. • Is a gateway configured, if the devices are in different IP ranges? Can the gateway be reached?!!

	• Is the use of a BBMD required (because the devices are in different networks (LANs))? • Was a physical connection established? (LinkStatus in the device status of the BACnet adapter; are all cables connected).
nOpenRequests **nFreeOpenRequestData** **nRequestRetries** **nRequestTimeOuts**	**Purpose:** This group of diagnostic values shows general information on the communication state of configured process data of the BACnet client. nOpenRequests indicates how many sent, so far unanswered requests exist for this client. This value ultimately reflects the number of invokeIDs used. nFreeOpenRequestData indicates how many invokeIDs remain available for the client. nRequestRetries indicates how often repetitions had to be sent for the process data-specific requests (ReadProperty, WriteProperty, ReadPropertyMultiple, SubscribeCOV). nRequestTimeOuts indicates how many process data-specific requests have failed, despite repetitions. The intervals between request repetitions are derived from the property APDU-Timeout of the server device object. **Problem indicator:** nRequestTimeOuts should never be greater than 0. This would indicate problematic communication errors. Occasional repetitions (nRequestRetries) may be acceptable. Solutions can be derived from the following 3 table rows, which list details of problems relating to the individual communication services.
nCyclicCovRequests **nCyclicCovPendingCounter** **nFailedCovRequests**	**Purpose:** These values provide information on COV-specific communication services. These values refer to registering of COV messages (SubscribeCOV). nCyclicCovRequests indicates how many property process data were configured with mode COV (Confirmed, Unconfirmed, Unsolicited). nCyclicCovPendingCounter indicates how many open SubscribeCOV requests exist for this BACnet client. The maximum number of parallel (open) requests per client can be specified via the Advanced Parameter MAX_PARALLEL_COV_SUBSCRIPTIONS. nFailedCovRequests indicates how many registrations of COV messages have failed. **Problem indicator:** nFailedCovRequests should never differ from 0. To subscribe to a large number of properties of a remote device, in some cases ReadPropertyMultiple can be used. If necessary, check whether the remote terminal supports COV and whether the subscribed properties are supported via COV or COV-P.
nCyclicReadPollingRequests **nFailedReadRequests**	**Purpose:** These values provide information on polling-based process data (ReadProperty, RPM-O, RPM-C). nCyclicReadPollingRequests indicates how many messages were configured for this process data type. Attention: this value does not contain the number of configured properties, but the number of messages required for data transfer. With RPM-O and RPM-C, several property values are read with one message. nFailedReadRequests contains the number of failed read attempts. **Problem indicator:** nFailedReadRequests should never be greater than 0, if possible. In the event of an error, check whether the remote terminal supports RPM. In the event of overload, the number of process data properties should be reduced, or the polling interval should be increased.
nCyclicWriteRequestsn **FailedWriteRequests**	**Purpose:** These values refer to writing process data. nCyclicWriteRequests indicates how many writing properties were configured under this BACnet client. nFailedWriteRequests indicates how many write requests (WriteProperty) failed, despite repetitions. **Problem indicator:** nFailedWriteRequests should never differ from 0. In the event of overload, the

	cycle time should be increased. It determines how often changes can be written. If necessary, check whether attempts to write non-existing or read-only BACnet properties are made.

2.12 BACnet Broadcast Management Device

TwinCAT BACnet/IP supports the "BACnet Broadcast Management Device" (BBMD) functionality. BBMD can be used to transfer IP broadcast messages (e.g. 192.168.1.255), which are used by certain BACnet services, beyond the boundaries of local networks. One example is the *Who-Is* service, which is used to find other BACnet devices via an IP broadcast message. A *Who-Is* message sent by a BACnet device is relayed to the local network with a broadcast MAC address (FF:FF:FF:FF:FF:FF). All BACnet devices in this network receive the message and can respond accordingly. However, the message is not relayed by IP routers that communicate with other IP subnets. BBMD was introduced to resolve this issue. In a local network only one device with BBMD functionality can be configured, which relays the IP broadcast messages to remote IP subnets via a *Broadcast Distribution Table* (BDT).

Two procedures are available for transferring BBMD messages, referred to as *One-Hop* and *Two-Hop*. With the *Two-Hop* procedure a BBMD device transfers broadcast message to a further BBMD in the remote IP subnet, which then distributes the messages in its local network as IP broadcast messages for its IP subnet. With the *One-Hop* procedure the local BBMD transfers the message directly to the remote devices. In this case the IP routers must support the transfer of IP broadcast messages to remote IP subnets.

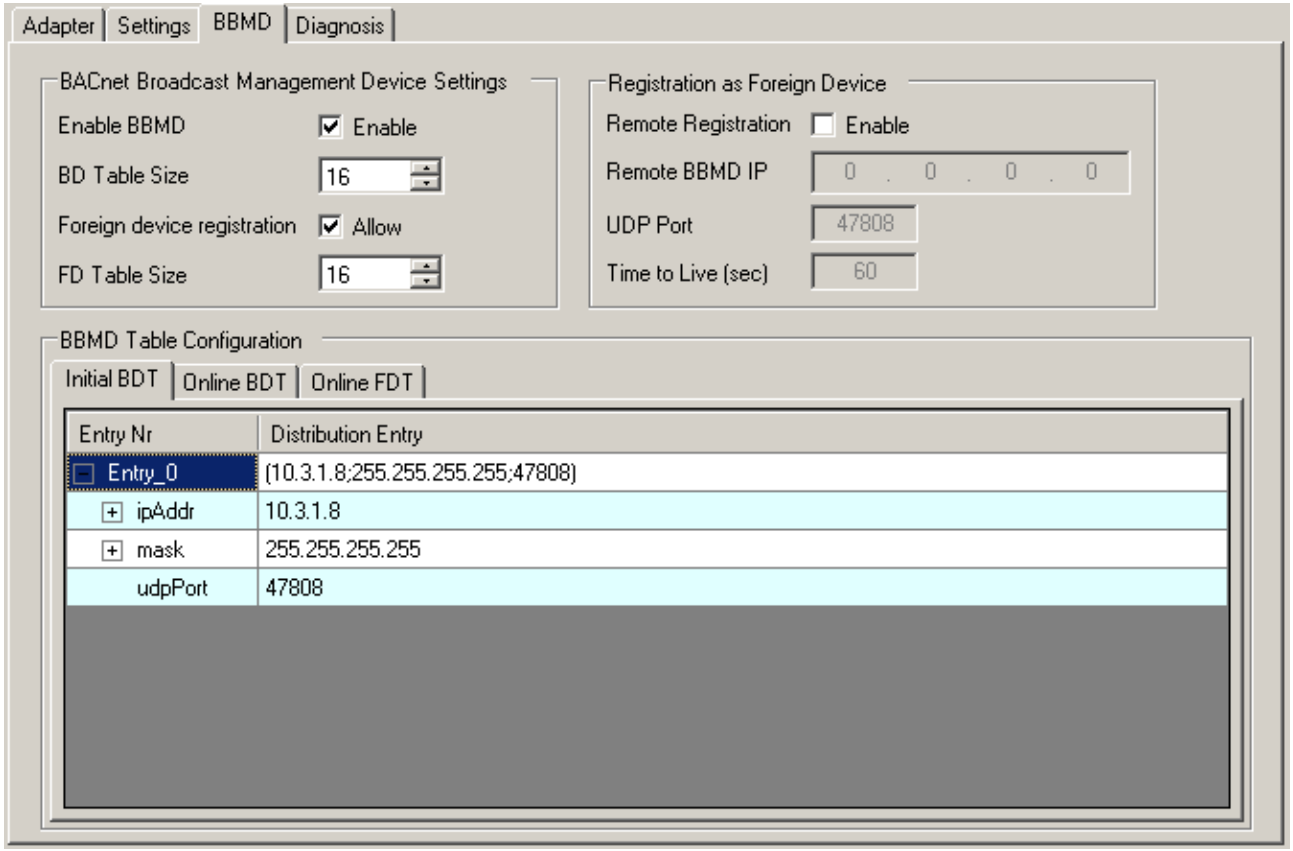
Gateway configuration

The BBMD functionality requires a gateway to be entered in the IP configuration of a BACnet device, since IP messages must be relayed to remote IP subnets. If no gateway is configured, the BBMD functionality is automatically disabled.

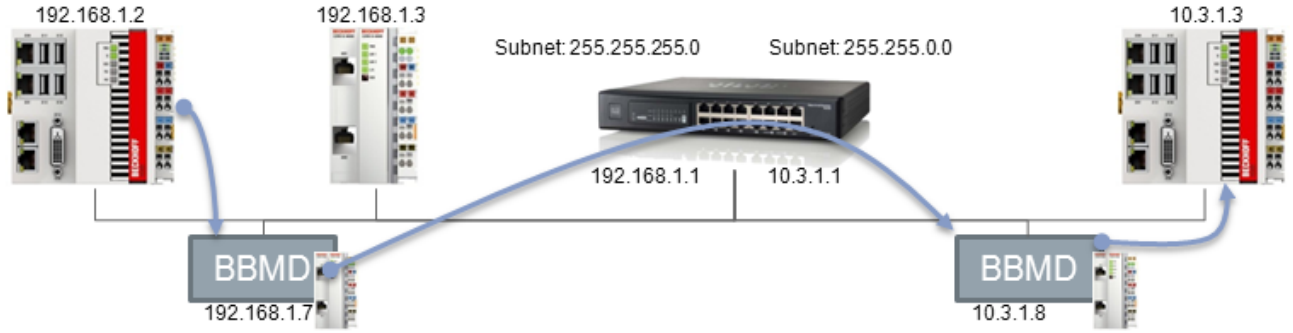
The "Foreign Device" procedure can be used to enable remote devices that dial in via an Internet connection, for example, to link dynamically with a BACnet configuration. Special BACnet services exist, which are also supported by TwinCAT BACnet/IP, for generating an entry in the *Foreign Device Table* (FDT). Like the BDT procedure, a BBMD IP broadcast relays messages to all FDT entries. In contrast to BDT, FDT entries have a limited lifetime. This means that login has to be refreshed after a time interval (*Time To Live*) has elapsed.

The BACnet device has a BBMD tab for configuring the BBMD. Each TwinCAT BACnet/IP device can therefore operate as a BBMD. Only one BBMD may exist for each subnet. The BBMD functionality can be activated with the option "Enable BBMD". In addition, the size of the BDT and FDT can be configured. During runtime new entries can be added to the BDT and FDT. This should be considered when configuring the size. Entries that are added to the BDT at runtime are stored as persistent data if this option is activated. As for BACnet properties, BDT entries are distinguished between offline and online data. The "Initial BDT" describes entries that are used on startup. During runtime the state of the BDT can be monitored via the tab "Online BDT". The FDT can be viewed via the tab "Online FDT".

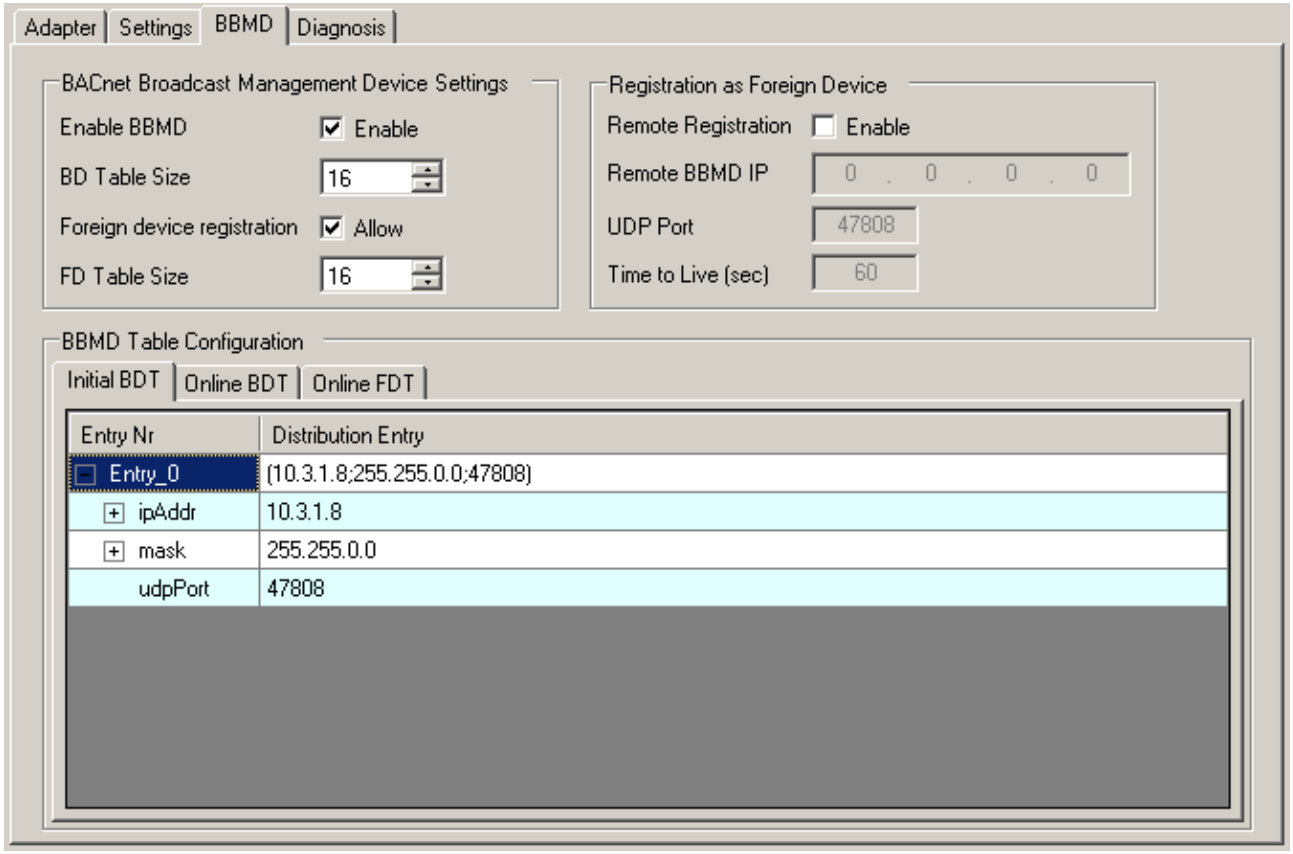
Via the submenu "Registration as Foreign Device" a TwinCAT BACnet/IP can register devices in an existing BACnet network. To this end the IP address of the remote BBMD must be activated.



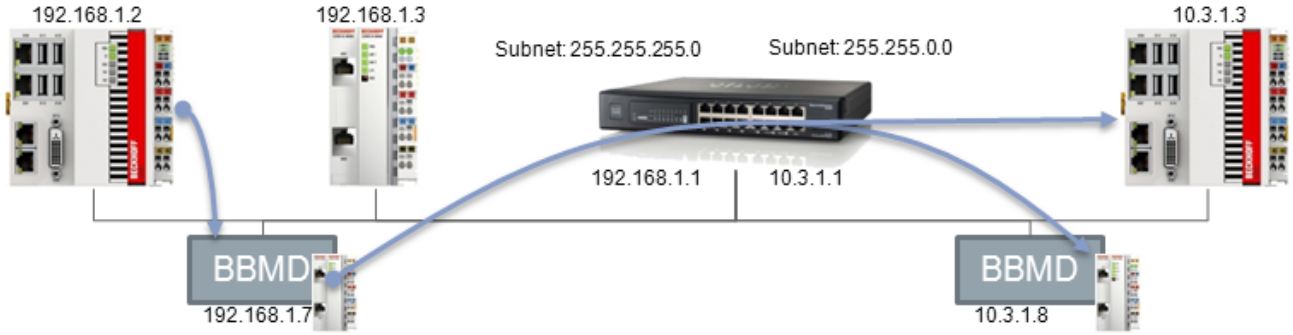
The above screenshot shows the configuration of the BDT in two-hop mode. A small example will be used to describe the BBMD functionality. The following figure shows two IP subnets that are connected via a router. In the example the device 192.168.1.2 sends an IP broadcast message (in this case 192.168.1.255) to the local network. The BBMD (IP address 192.168.1.7) relays the message as forwarded NPDU via the router/ gateway (192.168.1.1) to the BBMD of IP subnet 10.3.xxx.xxx. In this case it is the configured BBMD with the IP address 10.3.1.8. The remote BBMD distributes the relayed message as an IP broadcast (10.3.255.255) to the local network.



The following screenshot shows the BBDM configuration in one-hop mode. In contrast to the two-hop procedure, the subnet mask of the target system is entered to indicate that a target IP broadcast message should be generated directly. In this case the last two digits of the IP address are not relevant since they are replaced with 255. In other words, it would make no difference if 10.3.0.0 was used, for example.



In this case the BBMD (192.168.1.7) sends the message with the MAC address of router/gateway to the IP broadcast address (10.3.255.255) of the remote IP subnet. In this mode the remote BBMD does not relay messages.



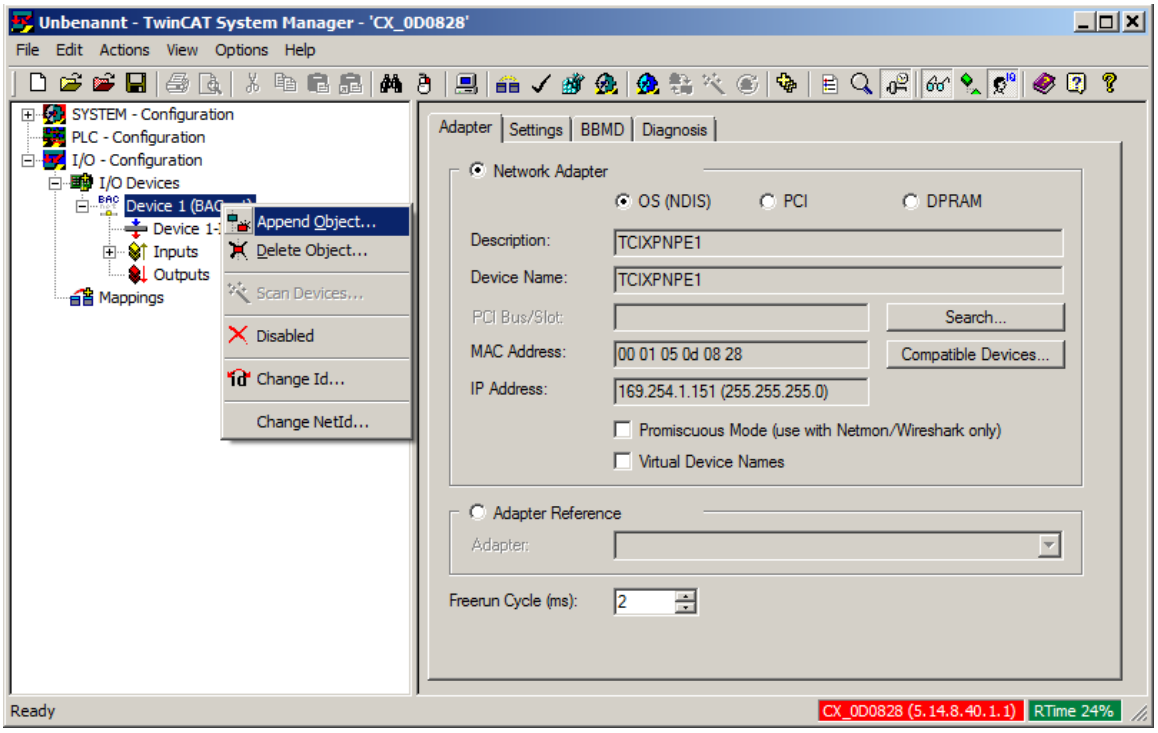
3 Application

3.1 Example: Create BACnet adapters and servers

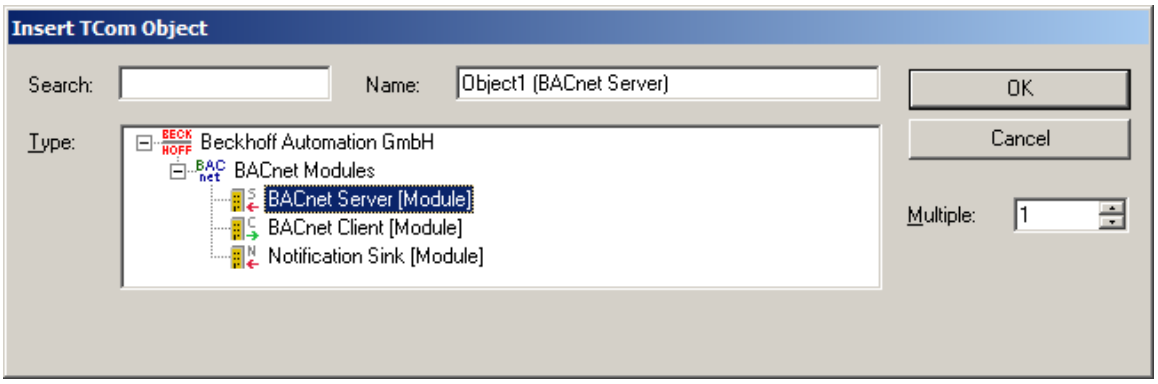
As already described in section BACnet device, server, client [▶ 14], the BACnet adapter is added into a project as an Ethernet device. Step-by-step instructions for creating a server under the BACnet adapter are provided below.

1. Creating a BACnet server under the used BACnet adapter

   a) Right-click on the adapter and select "Append Object..."



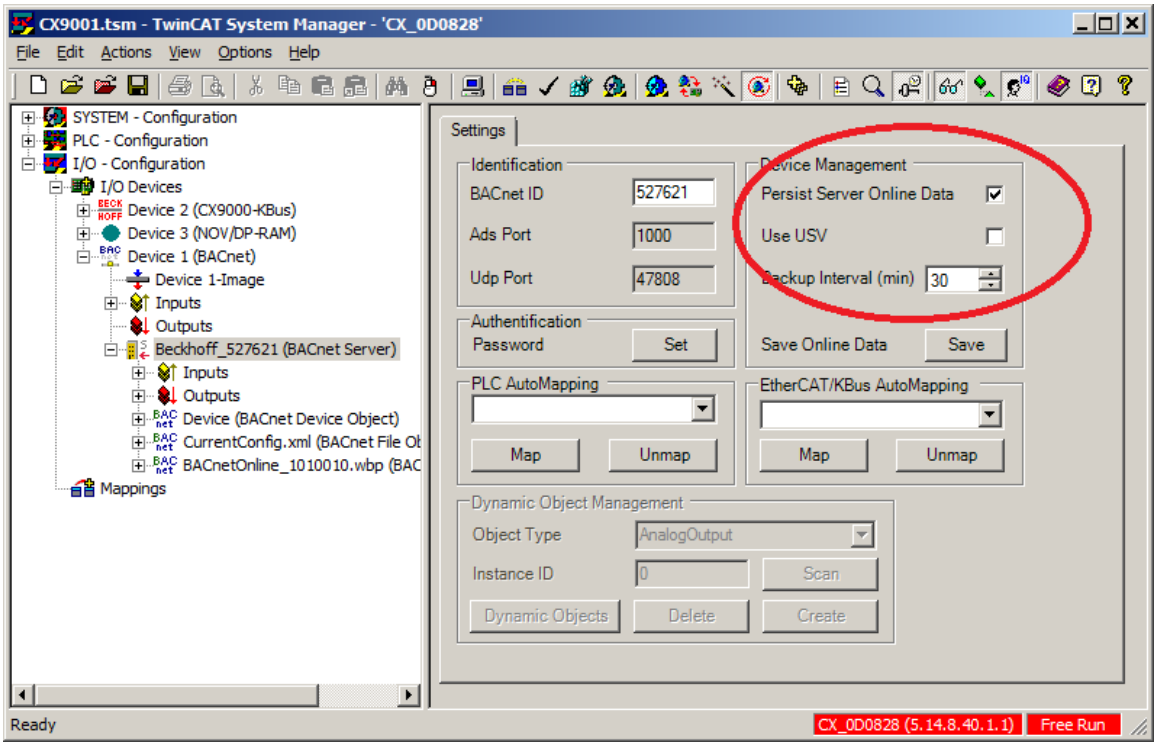
   b) In the subsequent dialog select the module "BACnet server" and confirm with "OK"



Supplement Key

Each configured BACnet adapter requires a license key, the so-called "Supplement Key". See section "Set up BACnet Supplement Key" for further information.

2. The option Persist Server Online Data should be activated if the server has to store the configuration data of all subordinate objects persistently. The screenshot below shows the Settings tab under which the option can be activated (see also section "Persistent data [▶ 57]"):

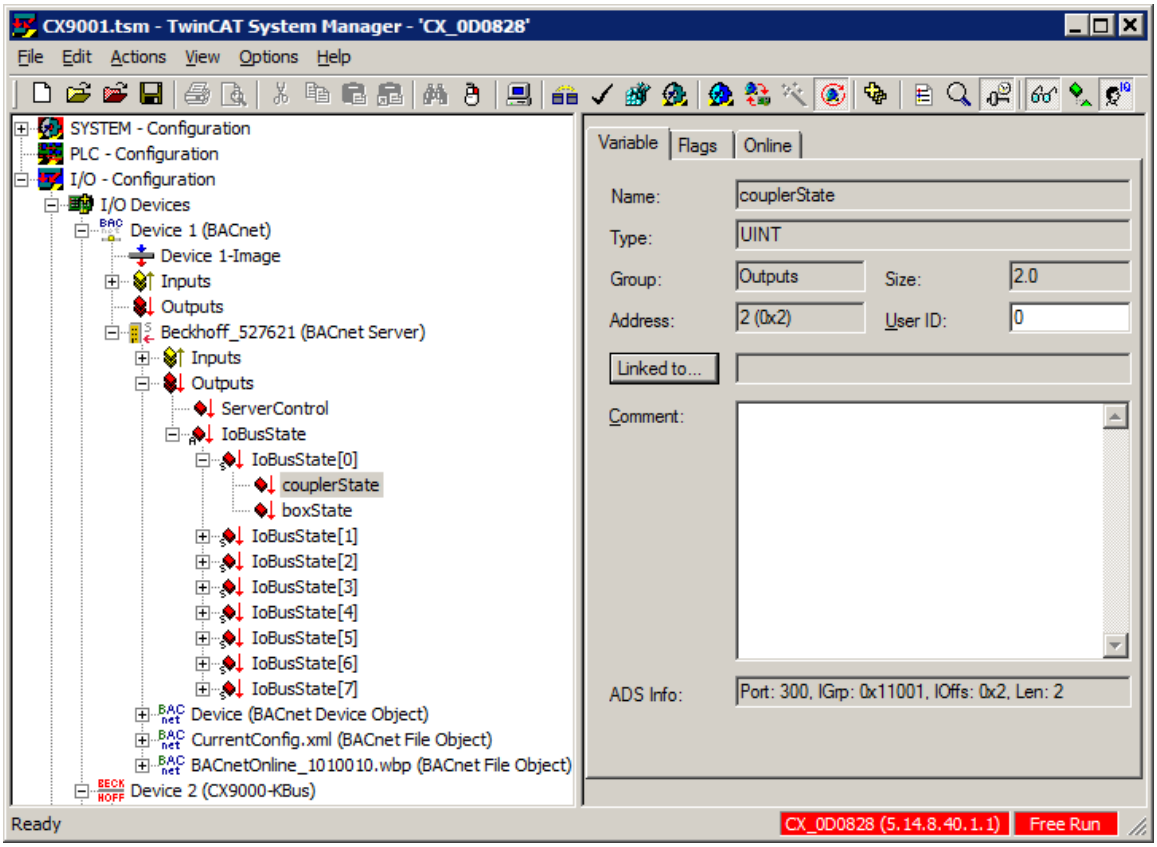


3.2 Example: Manual linking of hardware (terminal), BACnet BinaryInput and PLC program

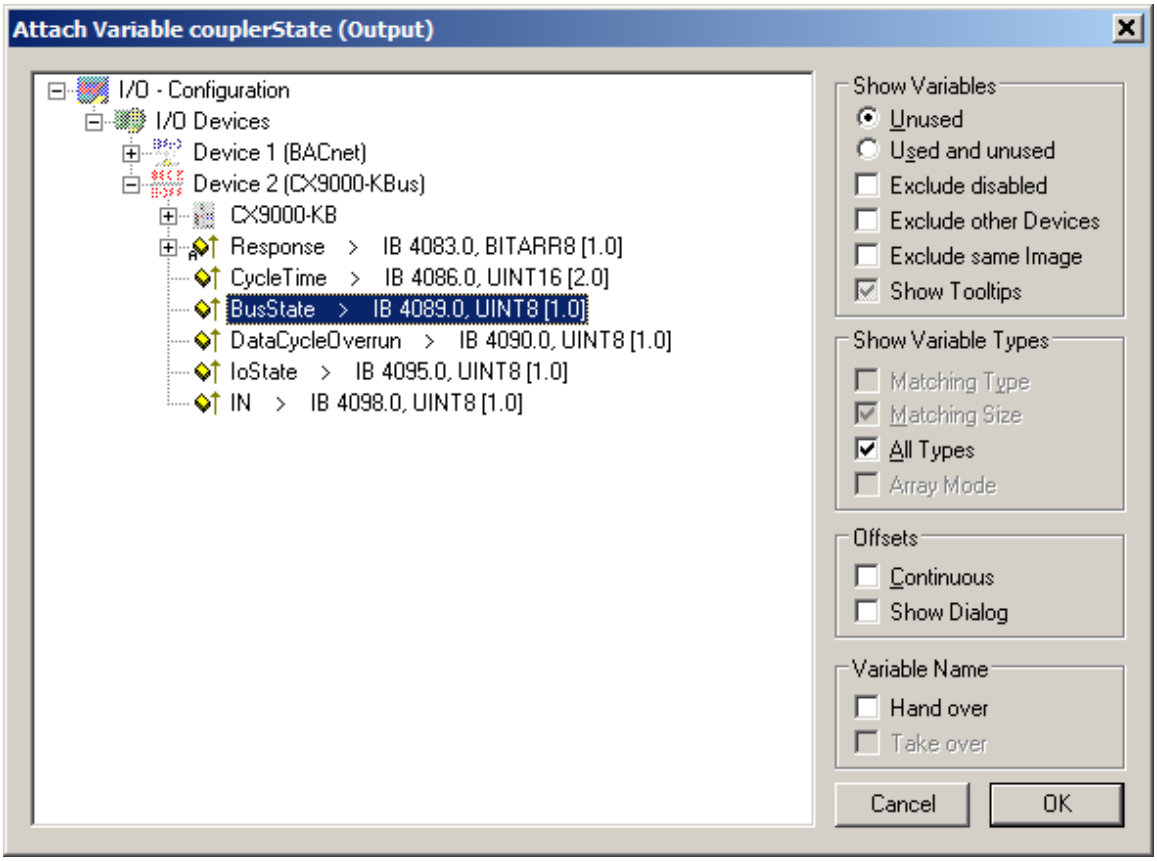
The following example describes the linking of two digital inputs with the corresponding objects and the PLC.

1. Create a BACnet adapter and server (see "Example: Create BACnet adapters and servers [▶ 92]")
2. Link the state of the bus coupler under which the hardware terminal was added with the BACnet server

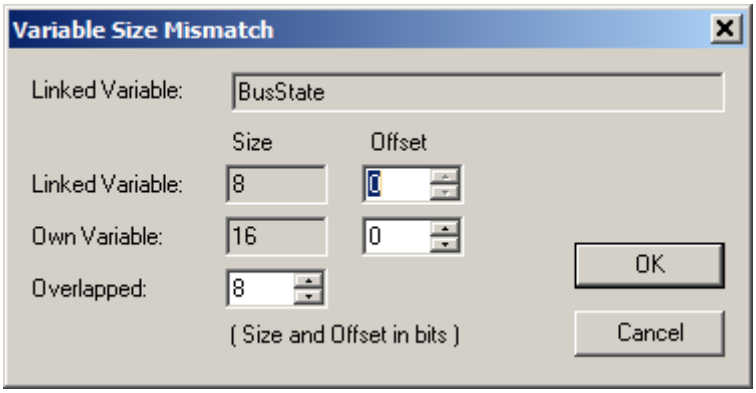
    a) Right-click on couplerState



b) In the subsequent dialog select BusState of the bus coupler selected (ATTENTION: deselect "Exclude other Devices" and "Exclude same Image" and select "All Types") and confirm with "OK"

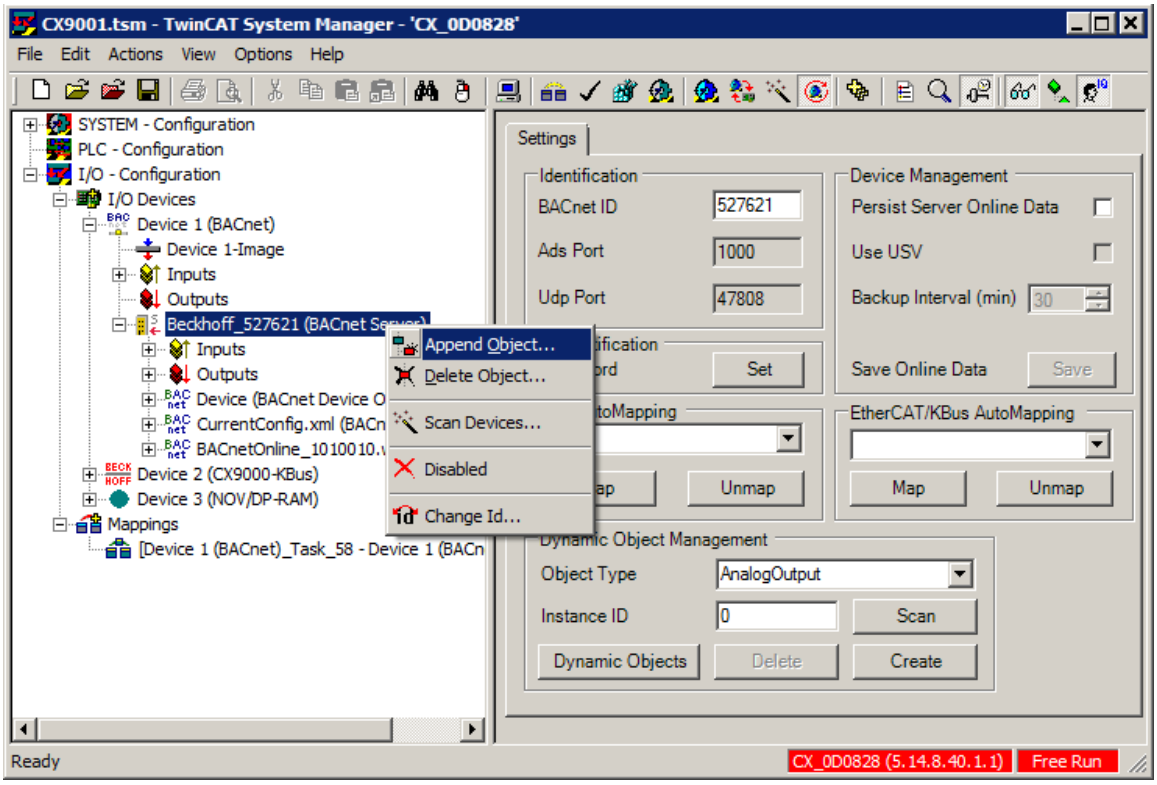


c) Confirm the subsequent dialog with "OK"

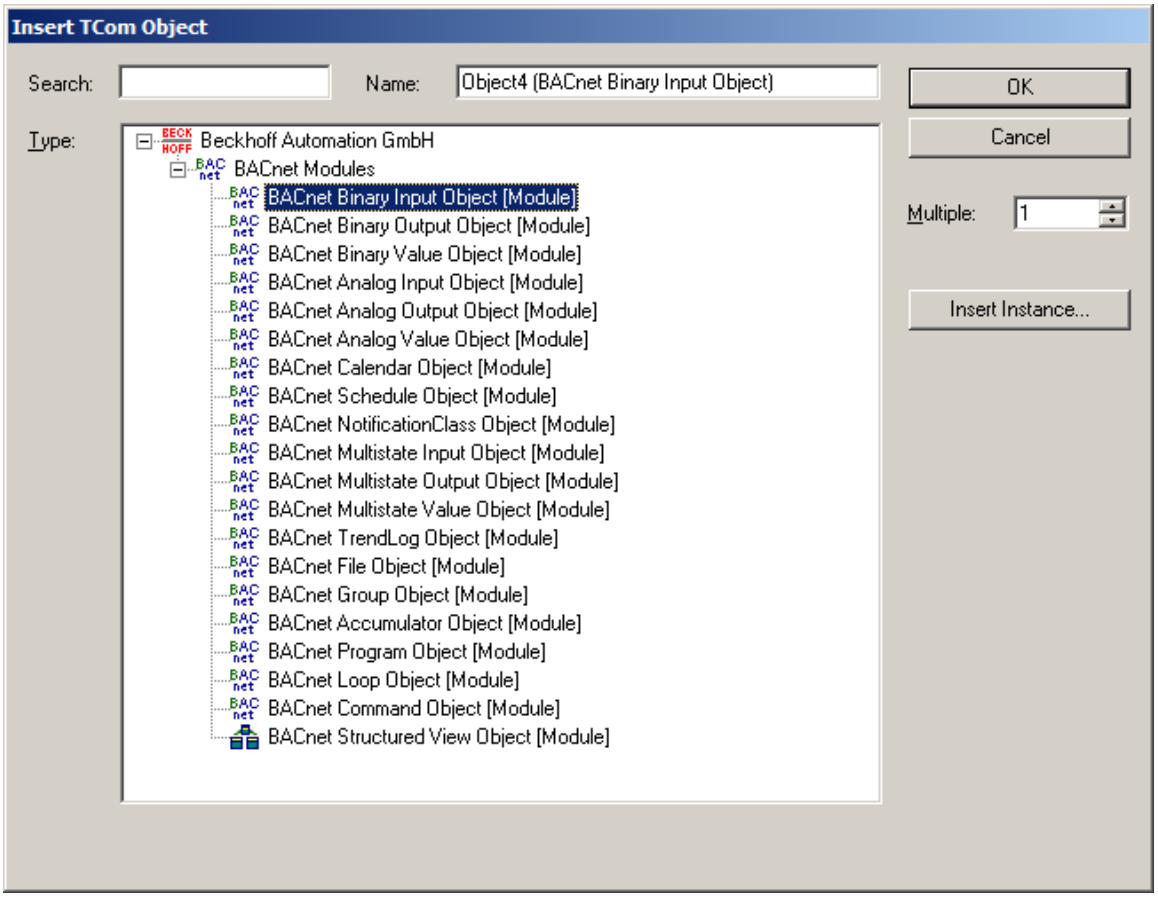


3. The first BinaryInput object under the previously created BACnet server is now added

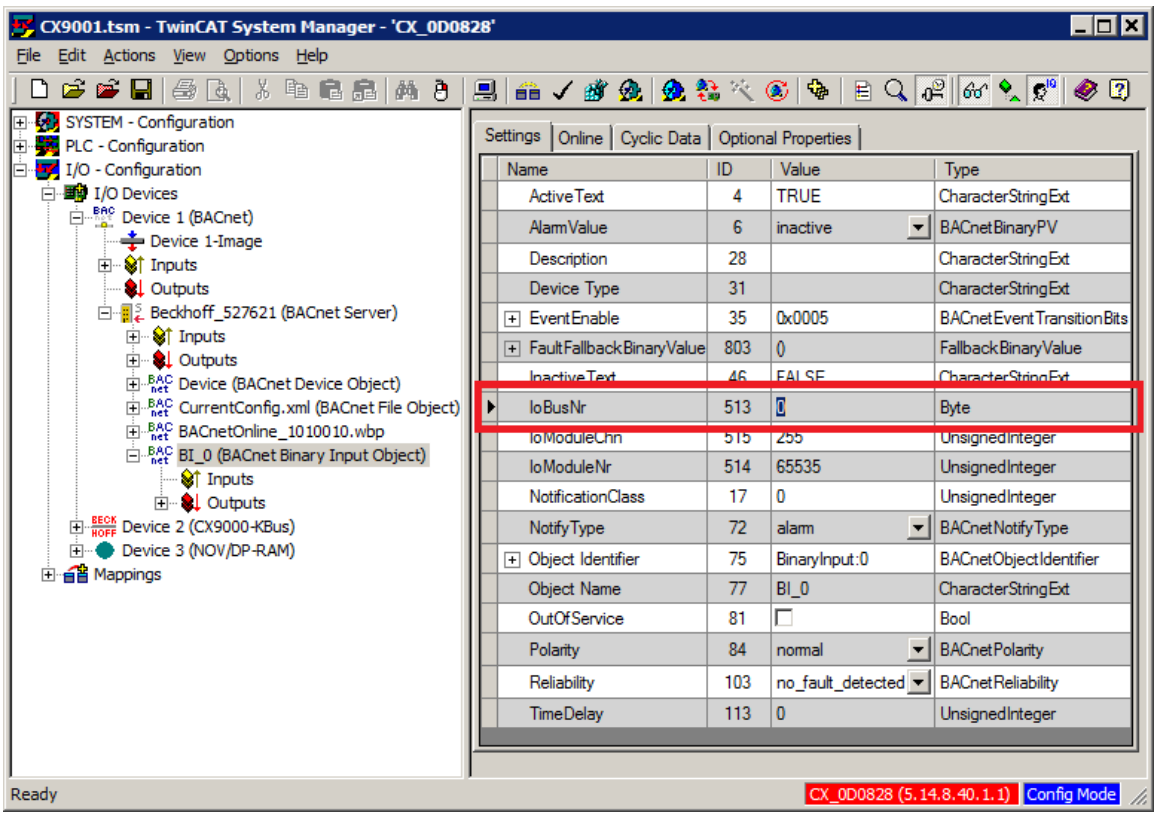
a) Right-click on the BACnet server and select "Append Object..."



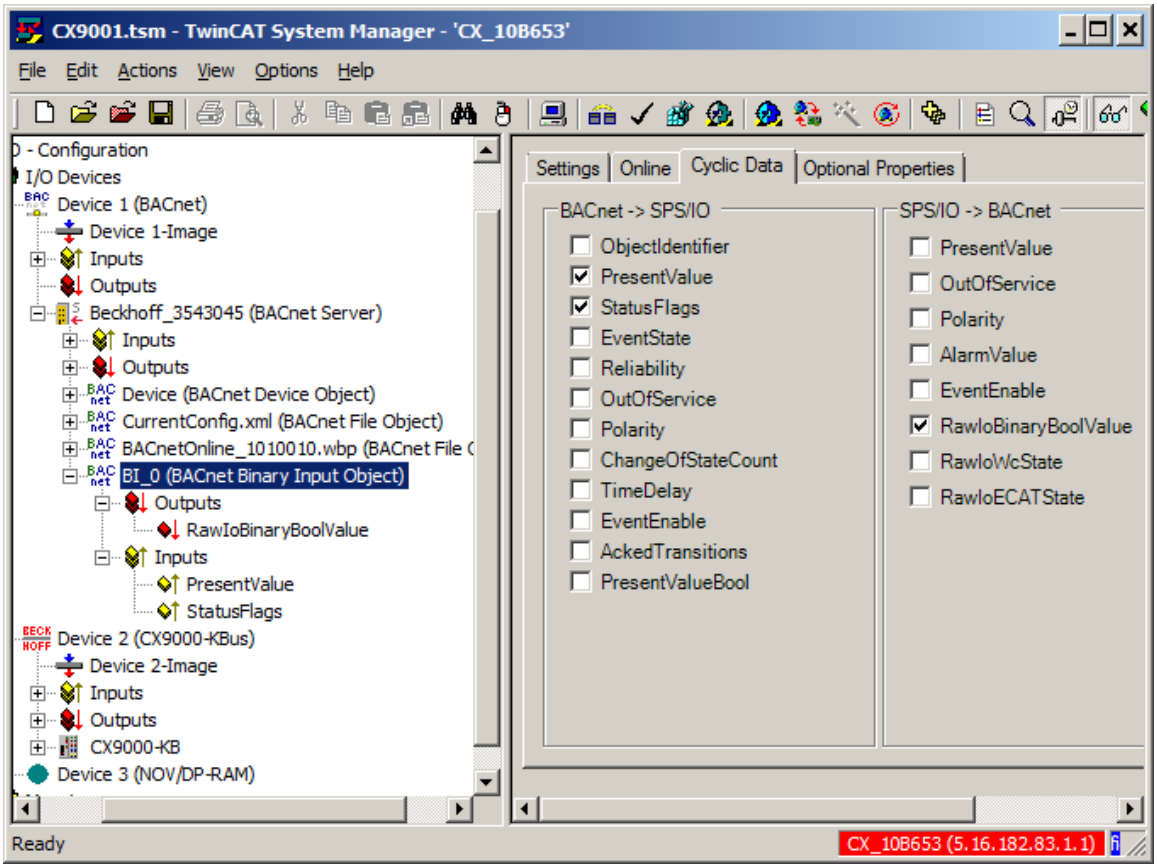
b) In the subsequent dialog select the module "BACnet Binary Input Object" and confirm with "OK"



4. For setting the property "Reliability" of the BinaryInput object the bus number of the bus state linked under 2 above has to be set in the BACnet object (bus index "0"):

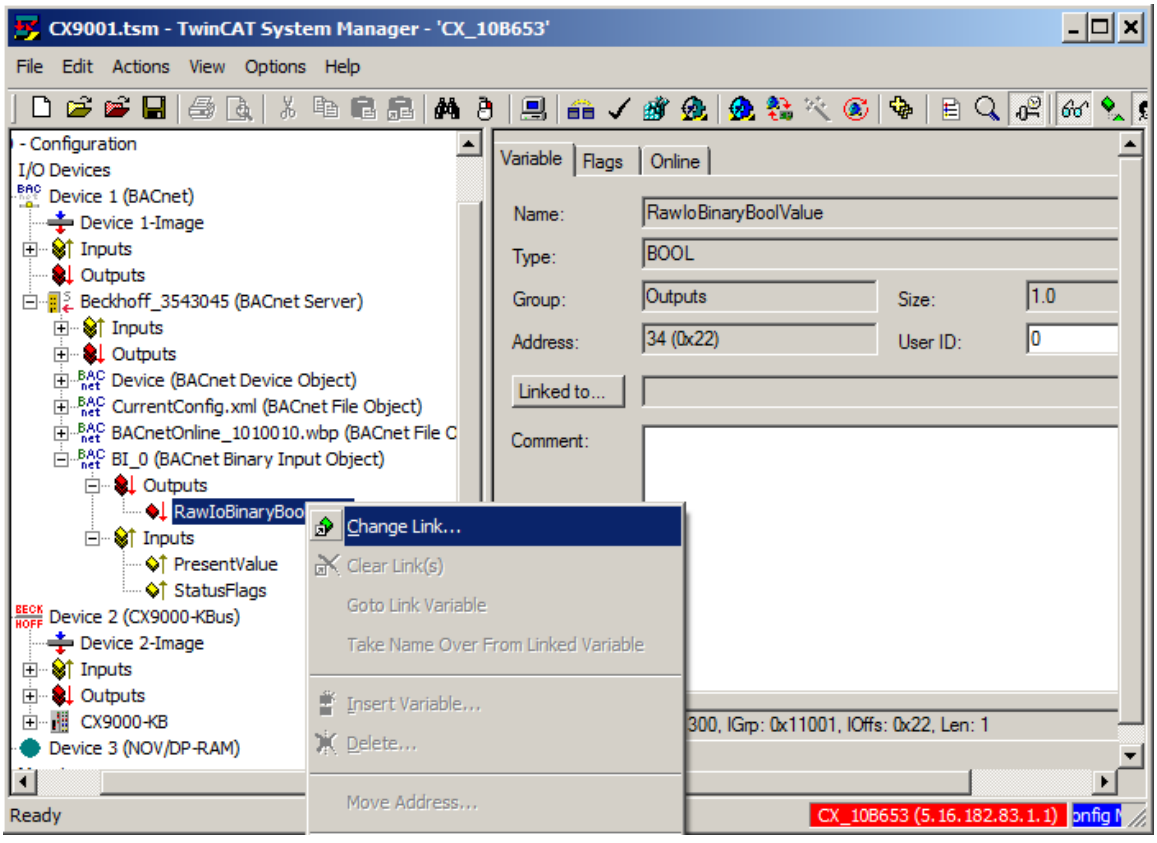


5. For subsequent mapping onto the PLC the corresponding properties should be added to the cyclic I/O mapping. To this end the following properties should be selected as a minimum:

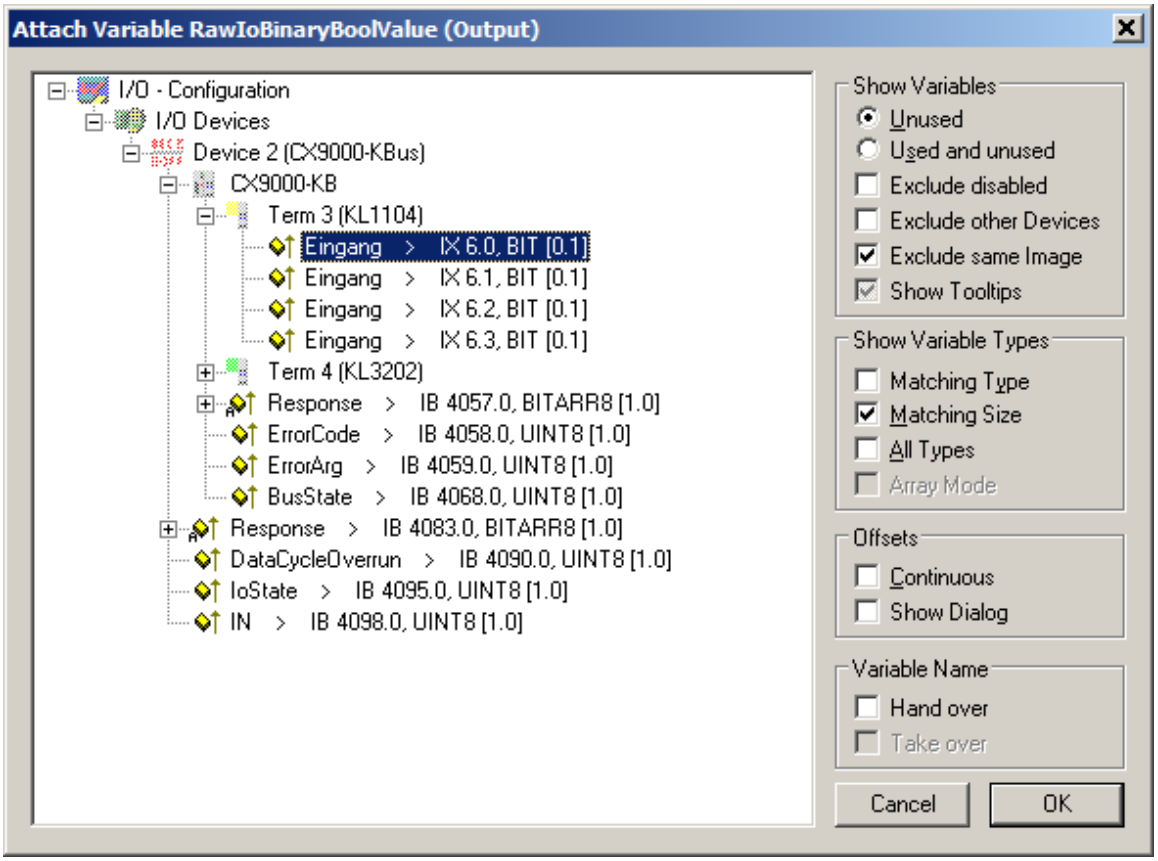


6. Subsequently the link between the hardware terminal and the BACnet object is created. To this end RawIoBinaryBoolValue has to be linked with the corresponding bit of the hardware terminal:

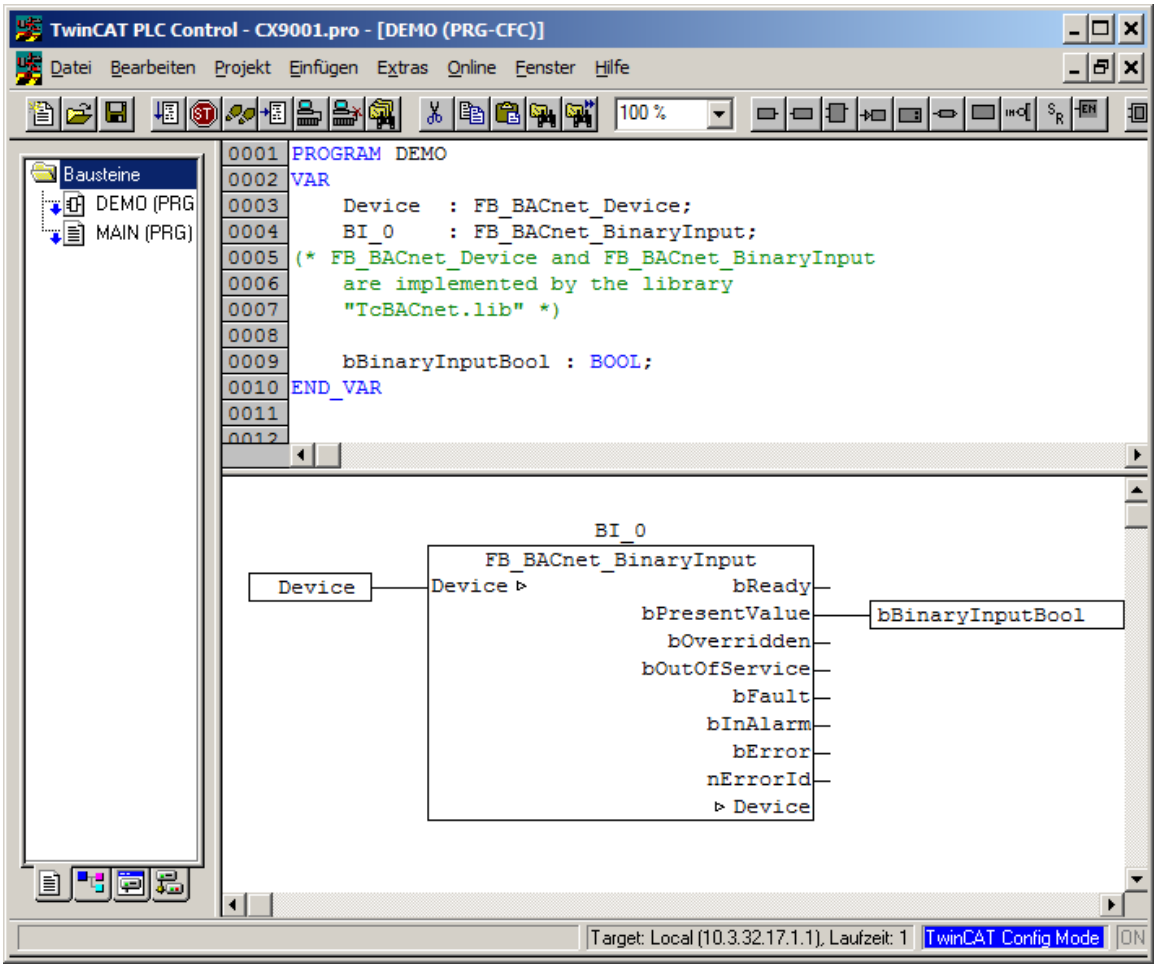
   a) Right-click on RawIoBinaryBoolValue under the BACnet object "BI_0" and select "Change Link..."



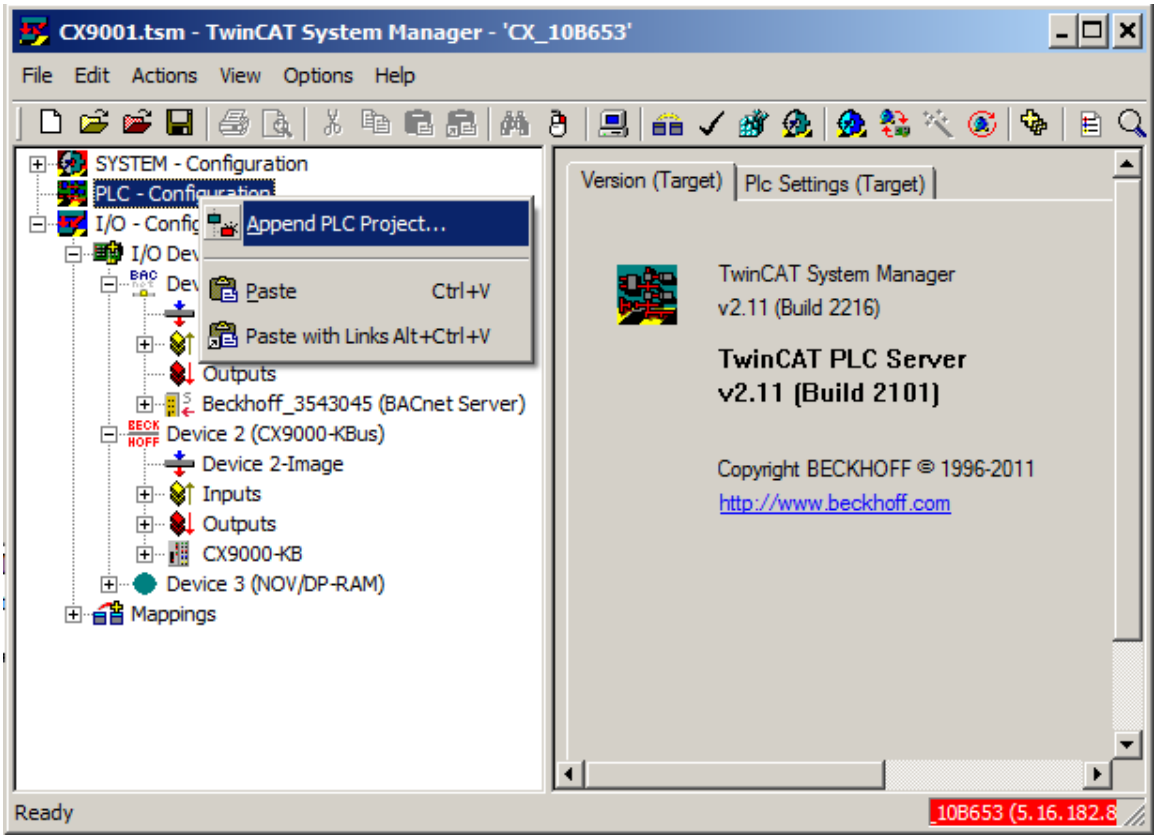
b) In the subsequent dialog select the symbol "Input" of the corresponding hardware terminal (ATTENTION: deselect "Exclude other Devices" and "Exclude same Image" and select "Matching Size") and confirm with "OK"



7. The library "TcBACnet.lib [▶ 371]" can be used for accessing BACnet objects from a PLC runtime. It makes all objects available as function blocks with the corresponding I/O points for the mapping in the System Manager. In the following example a PLC project with an instance of the function "FB_BACnet_BinaryInput [▶ 396]" with name "BI_0" is created. The function is inserted into the PLC program "DEMO" as follows (The "DEMO" program is in turn called in the "MAIN" program. Program "MAIN" is added to the task configuration as a task). The function block instance "Device" enables access to the device object and the server state and has to be transferred to the function block instance "BI_0":

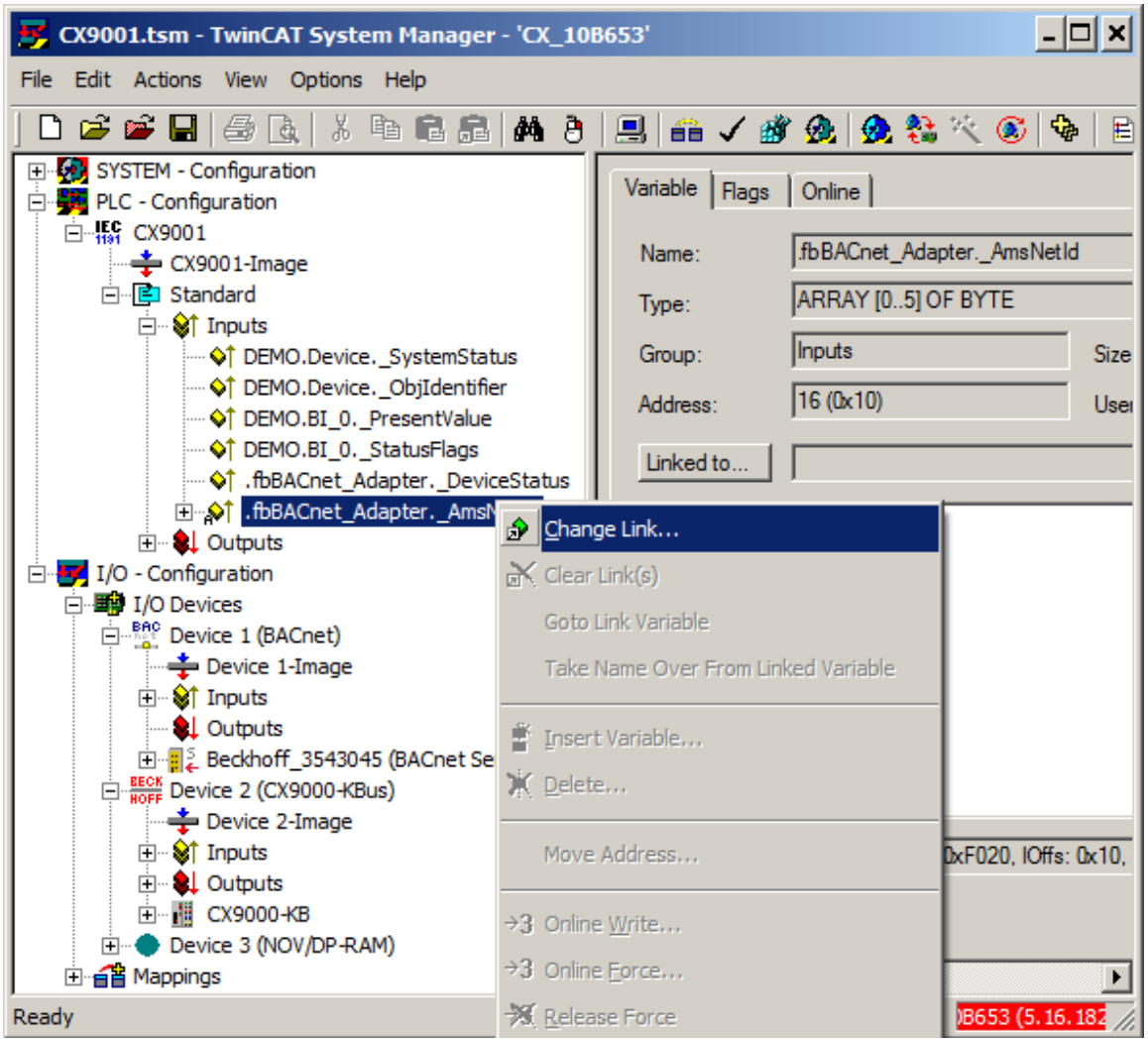


8. Once the PLC project is compiled it has to be made available in the System Manager. Right-click on the symbol "PLC - Configuration". Afterwards by means of "Append PLC Project..." add the TPY file of the previously compiled PLC project:

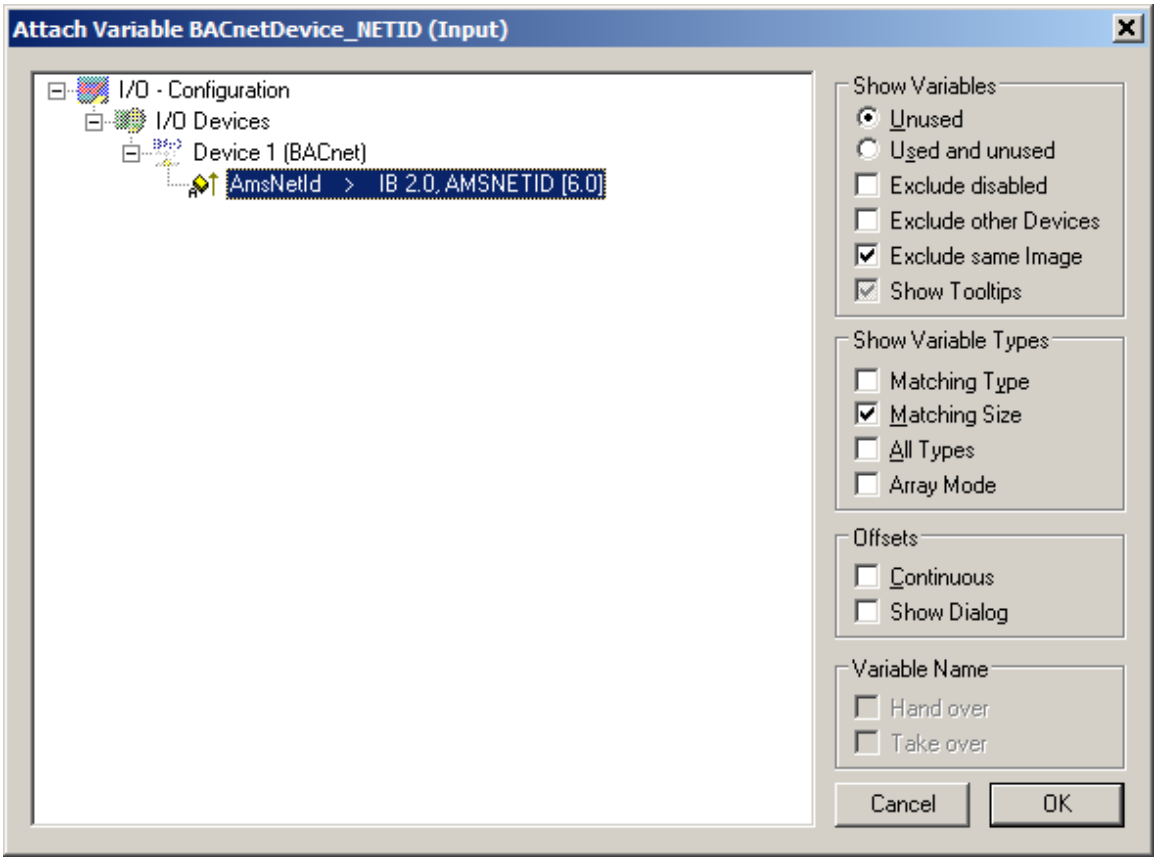


9. Once the PLC project has been added, all I/O variables of the PLC program are available in the System Manager. The global instance "fbBACnet_Adapter" of type FB_BACnet_Adapter [▶ 375] is available in each PLC project linked via the library "TcBACnet.lib". It is used for accessing the BACnet network adapter, its AMS NetID and the link status. The I/O variables under the instance now have to be linked with the I/O points of the BACnet adapter:

   a) Right-click on the symbol ".fbBACnet_adapter._AmsNetID" and select "Change Link..."

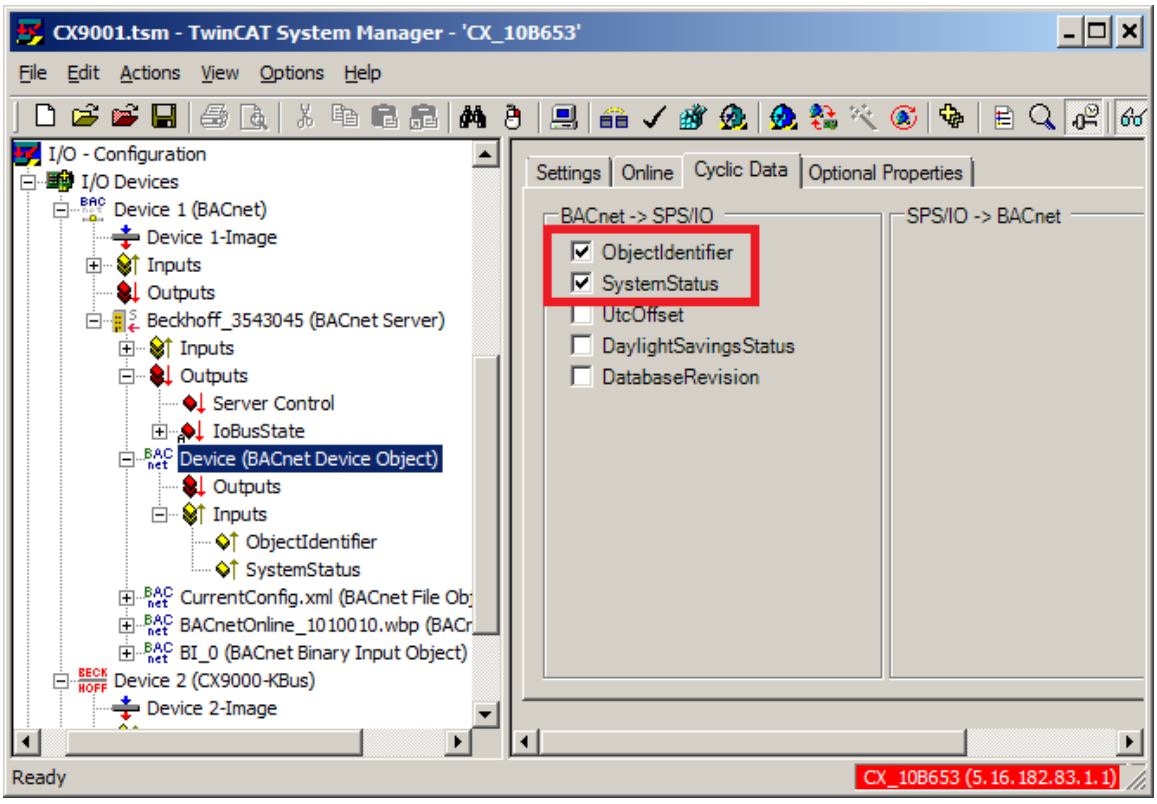


b) In the subsequent dialog select the AmsNetID of the BACnet device (ATTENTION: deselect "Exclude other Devices" and select "Matching Size") and confirm with "OK"



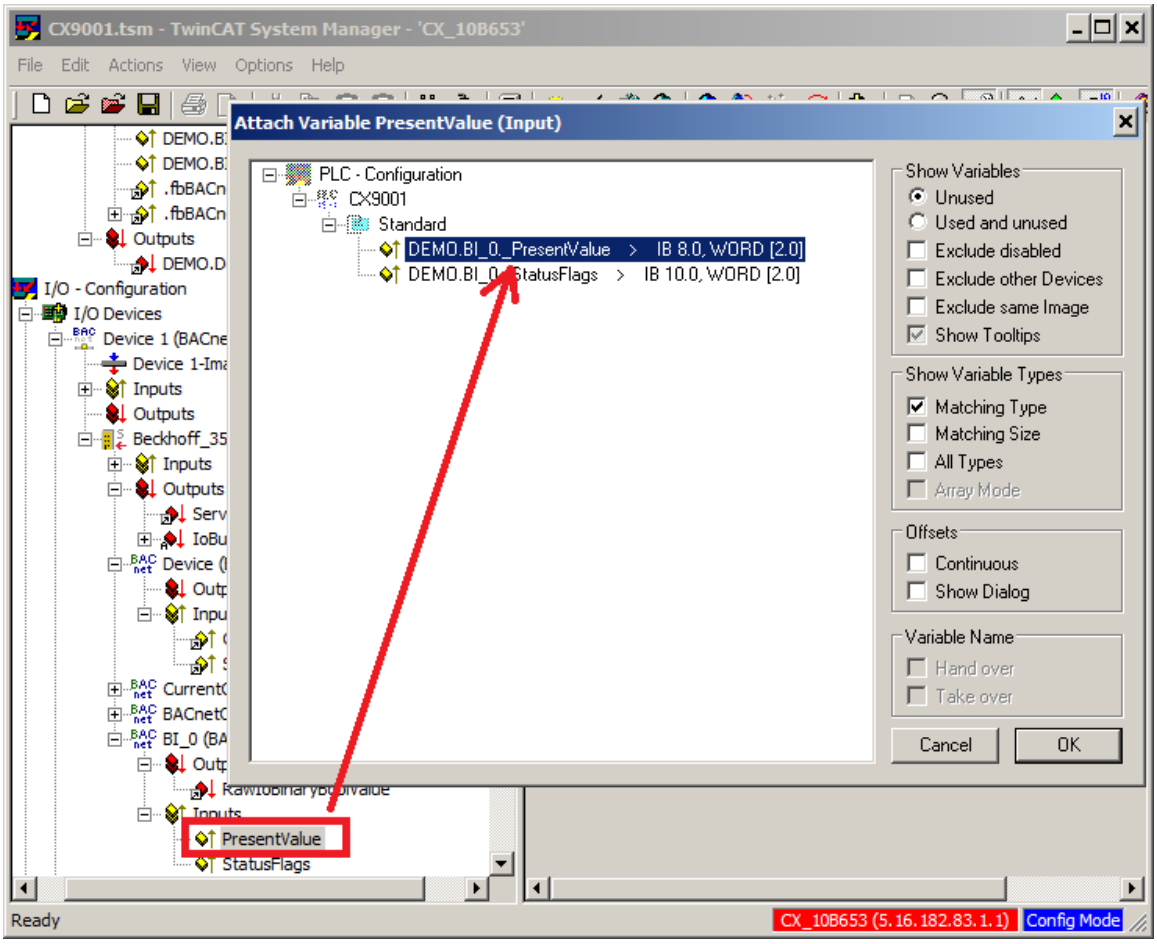
10. Repeat step 9 with "DeviceStatus" of the BACnet adapter
11. The process data inputs: "_SystemStatus", "_ObjIdentifier" and outputs: "_ServerControl" of the device object have to be linked in the same way. This maps the status of the BACnet server and the local device object to the PLC.

    a) Create process data for the device object:



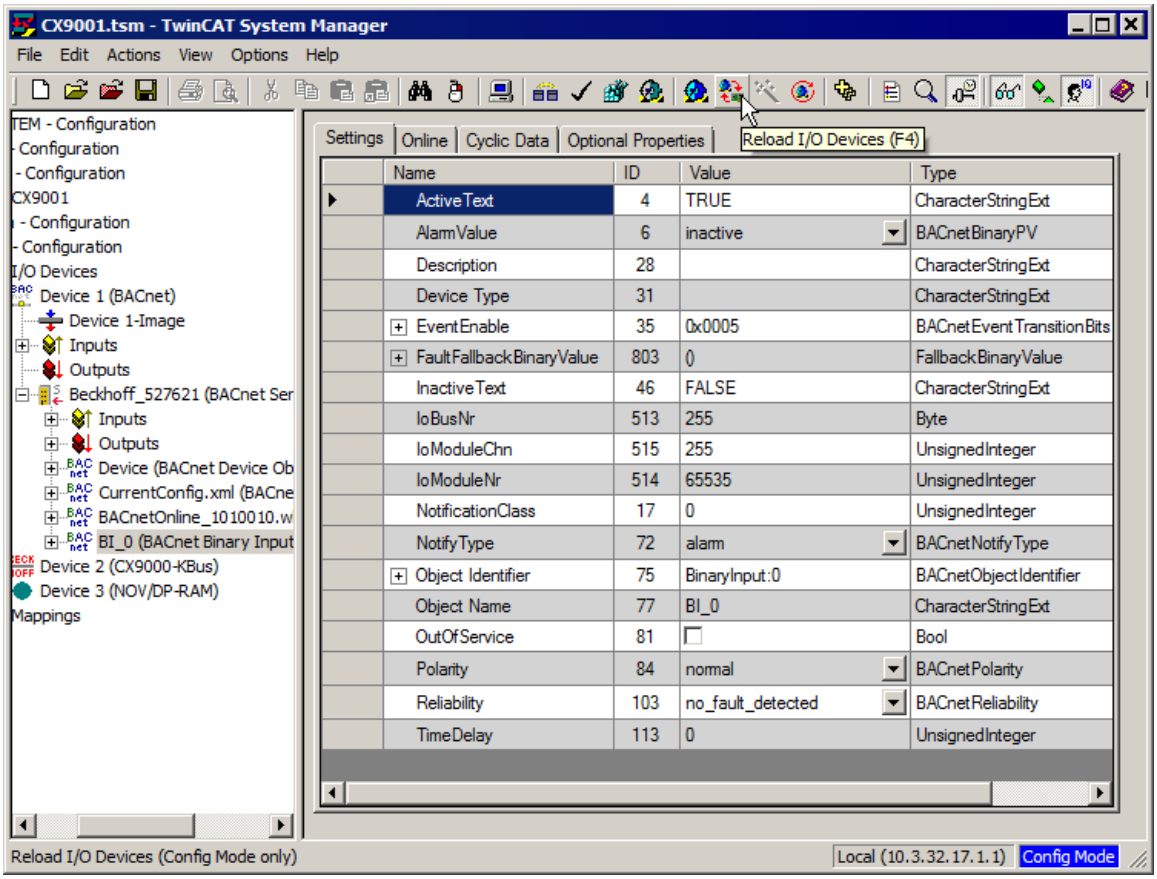
    b) Create link between PLC and I/O process data.

12. All I/O signals of the PLC/BACnet function block "BI_0" now have to be linked with the actual BACnet object. The links are established in the same way as specified under item 9 ("_PresentValue" and "_StatusFlags"):

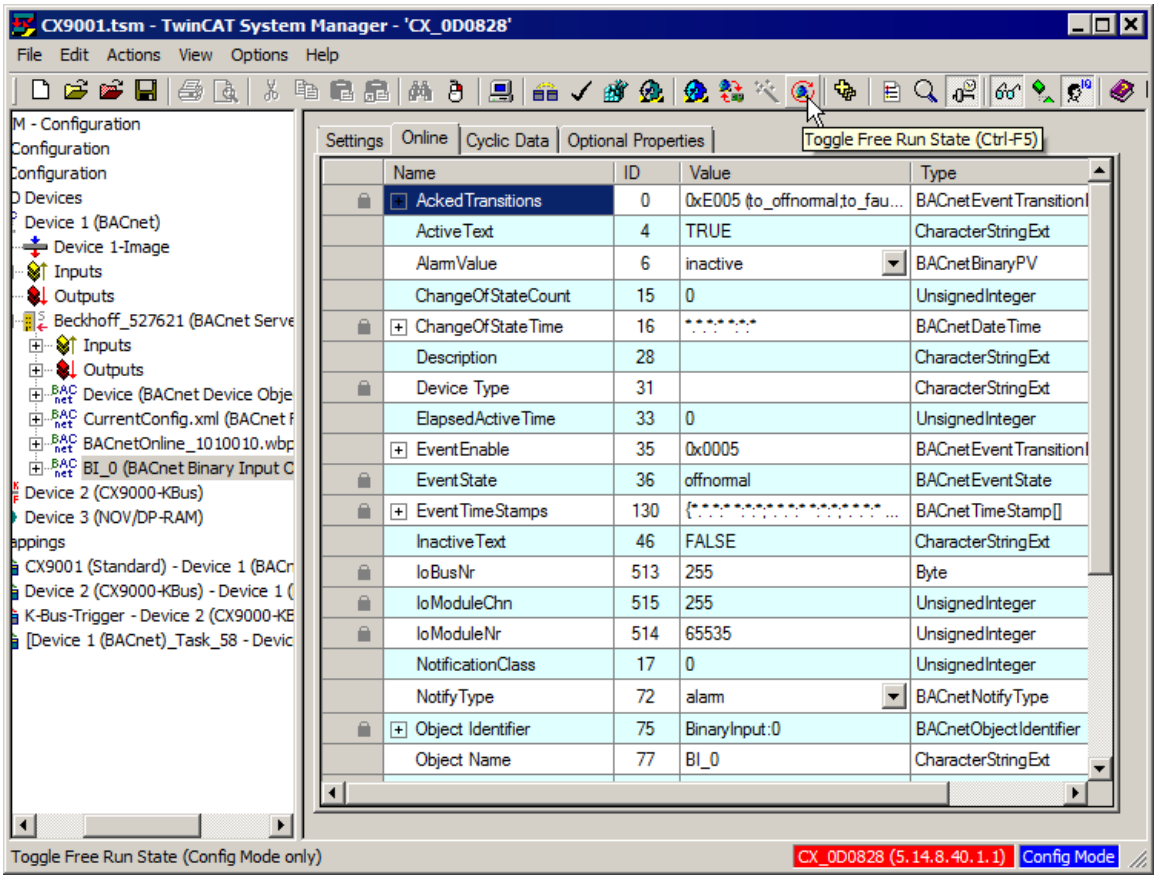


13. To check the function without active PLC runtime, the configuration of the BACnet device and the hardware terminals can be tested in Free Run mode:

    a) Select "Reload I/O Devices (F4)" from the toolbar or press F4

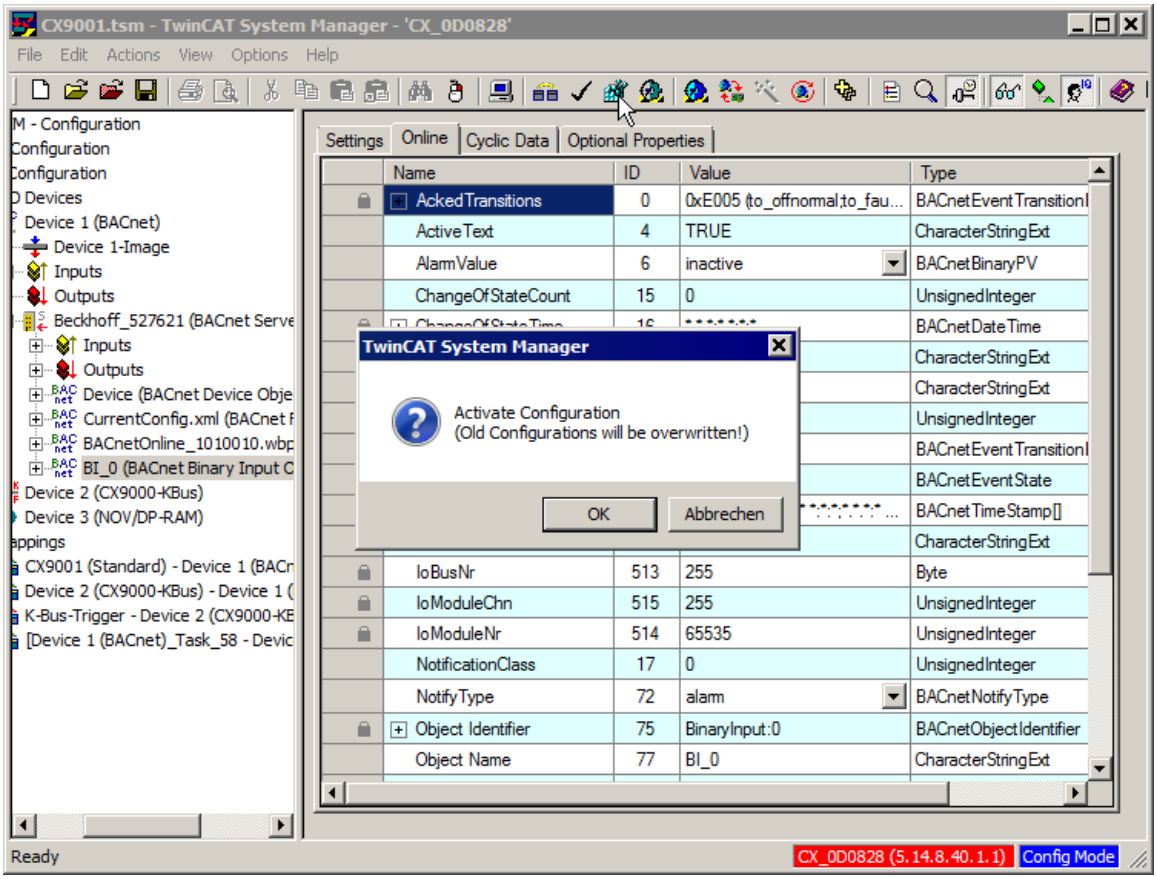


b) Select "Toggle Free Run State (Ctrl-F5)" from the toolbar or press CTRL+F5

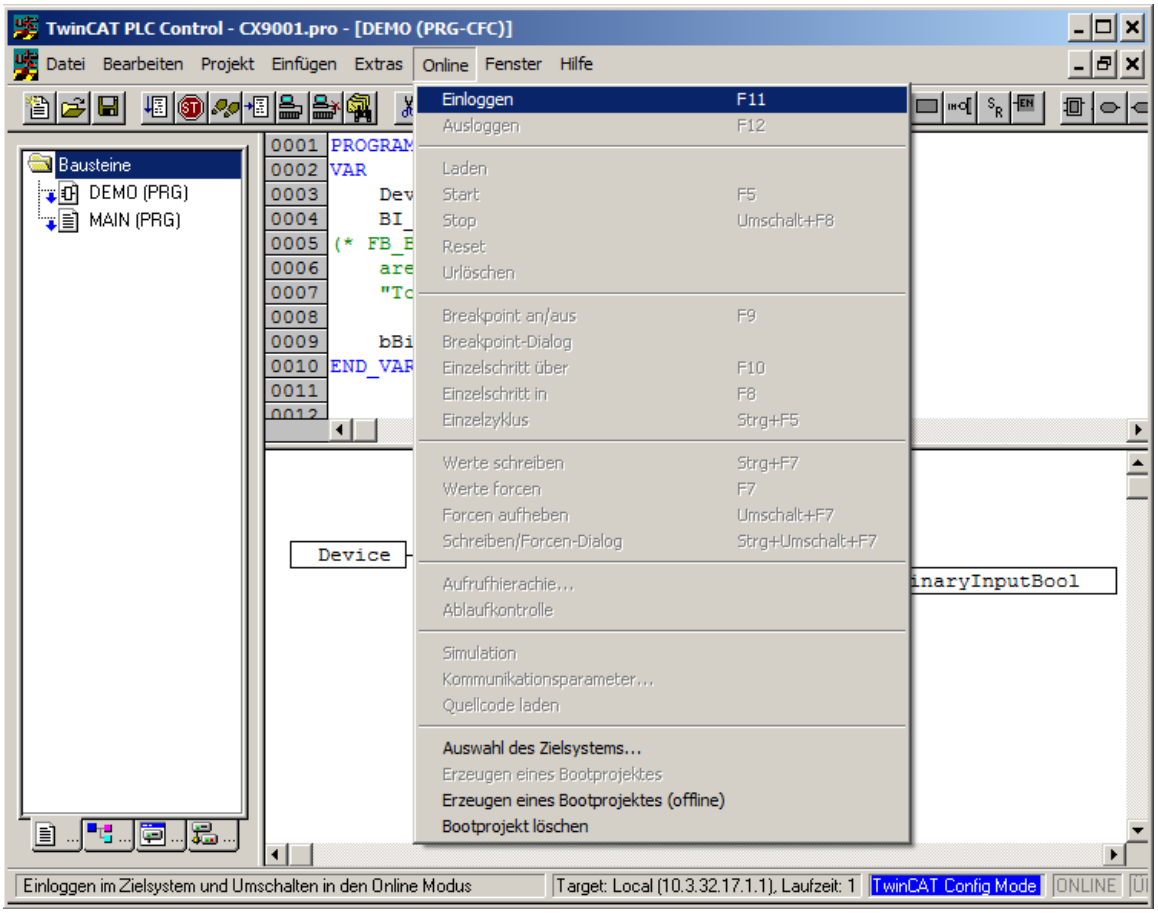


c) The I/O bus of the target system should then switch to RUN. The state of the hardware input can now be monitored in the Online tab of the BACnet object "BI_0". The property Present_Value no. 85 represents the logical state of the hardware input. Using the context menu of "Online" tab, the Auto Update mode should be activated in advance (see chapter "BACnet objects and properties [▶ 21]").

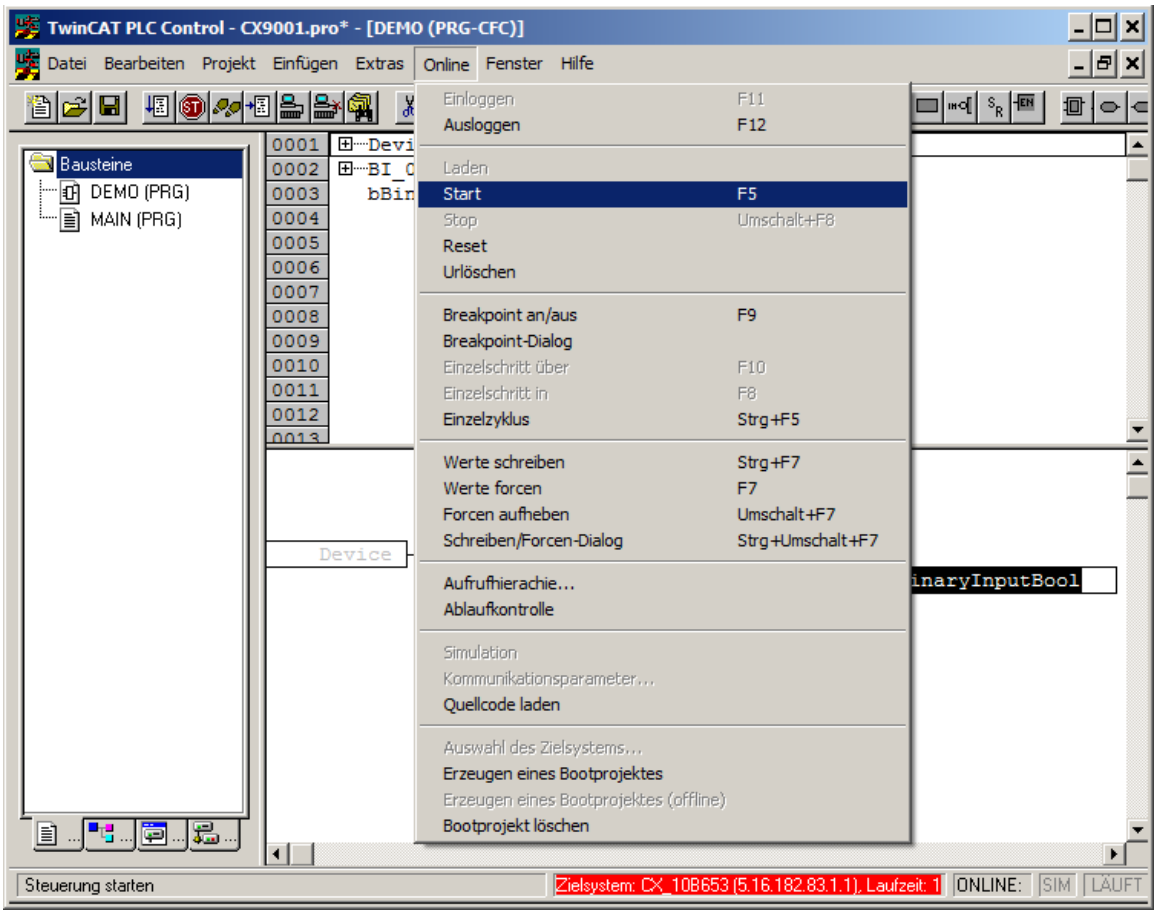
14. After a successful test in "Free Run" mode the configuration has to be activated permanently. To this end activate the configuration as usual via the symbol "Activate Configuration" from the toolbar or by pressing CTRL+SHIFT+F4 (this loads the configuration into the target system):



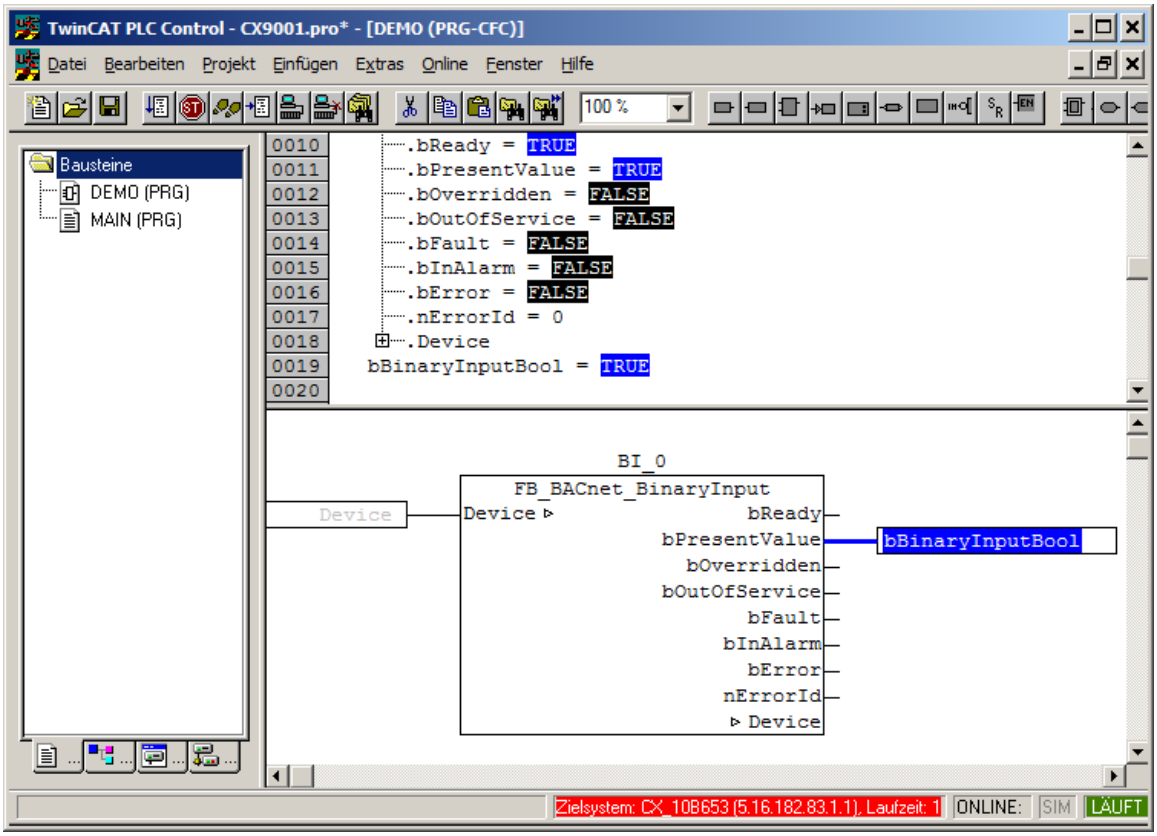
15. This is followed by loading of the PLC runtime:



16. and starting in RUN mode:



17. The states of the BACnet object can be monitored via the Online view of the "DEMO" program. The state of signal Present_Value represents the logical state of the hardware input:



3.3 Example: Linking of BinaryInput and BinaryOutput objects in the PLC program

The following example illustrates how the link of a binary input with a binary output can be realized with the aid of BACnet objects and a PLC program. The procedure is based on PLC automapping.

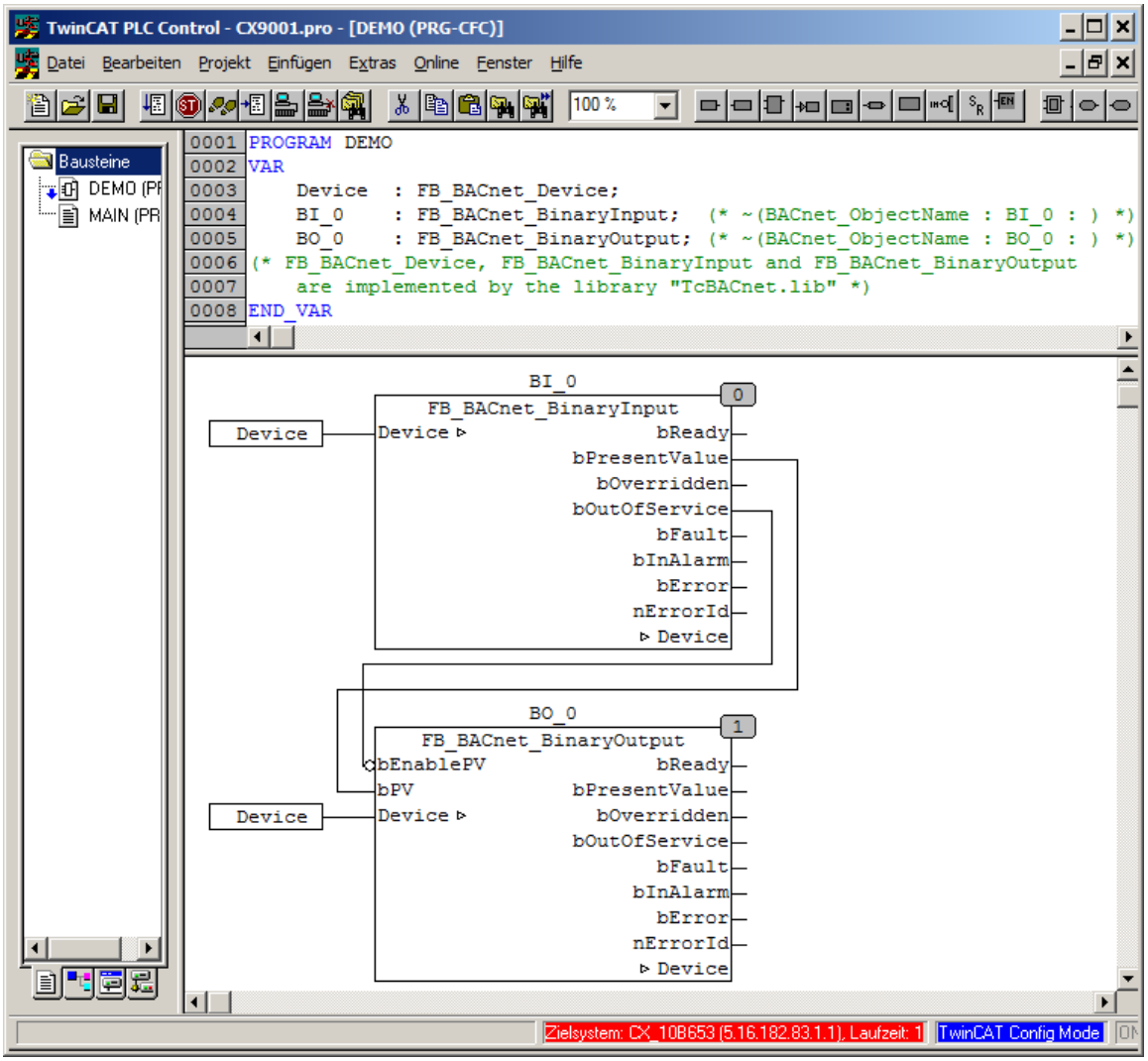
Summary:

- A binary input, represented by the BACnet object "BI_0", is to write its Present Value to the binary output, represented by the BACnet object "BO_0", with priority "12", if the object "BI_0" is not in state OutOfService
- If the object "BI_0" is in state OutOfService, PresentValue access with priority "12" to object "BO_0" should be deleted.

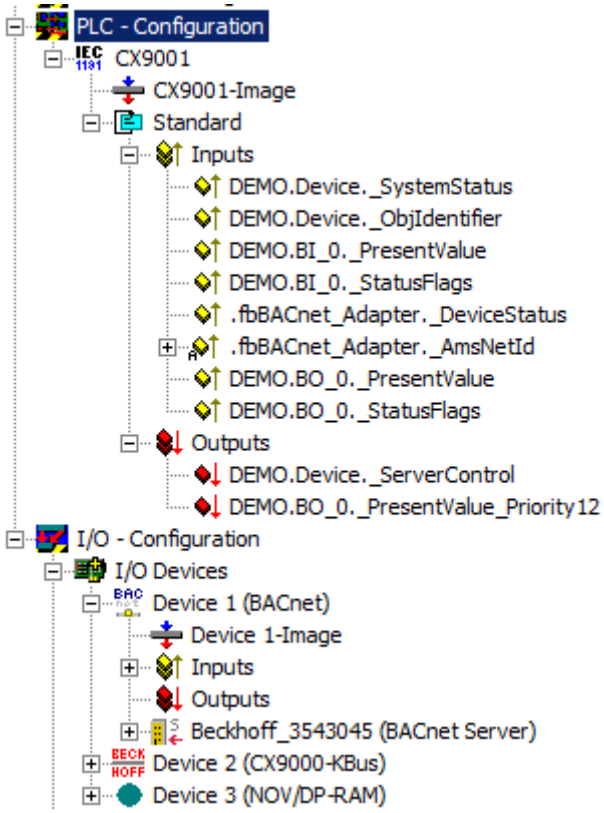
The example https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749043083/.zip can be downloaded from herehttps://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749043083/.zip.

The strategy is explained step-by-step with the aid of screenshots (For creating servers and objects see also examples "Create BACnet adapters and servers [▶ 92]" and "Manual linking of hardware (terminal), BACnet BinaryInput and PLC program [▶ 93]"):

1. Create BACnet adapters and servers (see Example "Create BACnet adapters and servers [▶ 92]")
2. Creating an I/O bus (K-bus, E-bus, BK90xx)
3. Execute I/O automapping [▶ 72] of the bus on BACnet (see "example: I/O automapping [▶ 115]")
4. Creating a PLC project and inserting object instances (The link to the input/output objects is realized using the object names "BI_0" and "BO_0" in the comment, see also "PLC automapping [▶ 62]"):

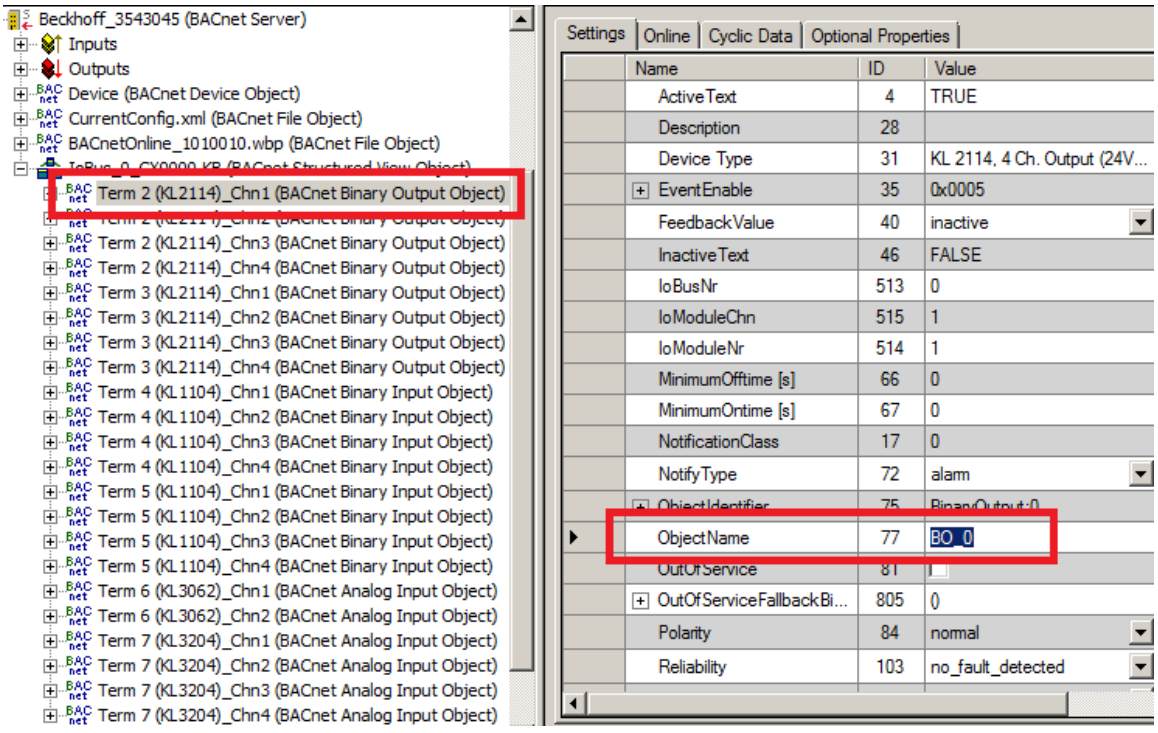


5. Adding the PLC project (.tpy) to the System Manager configuration:

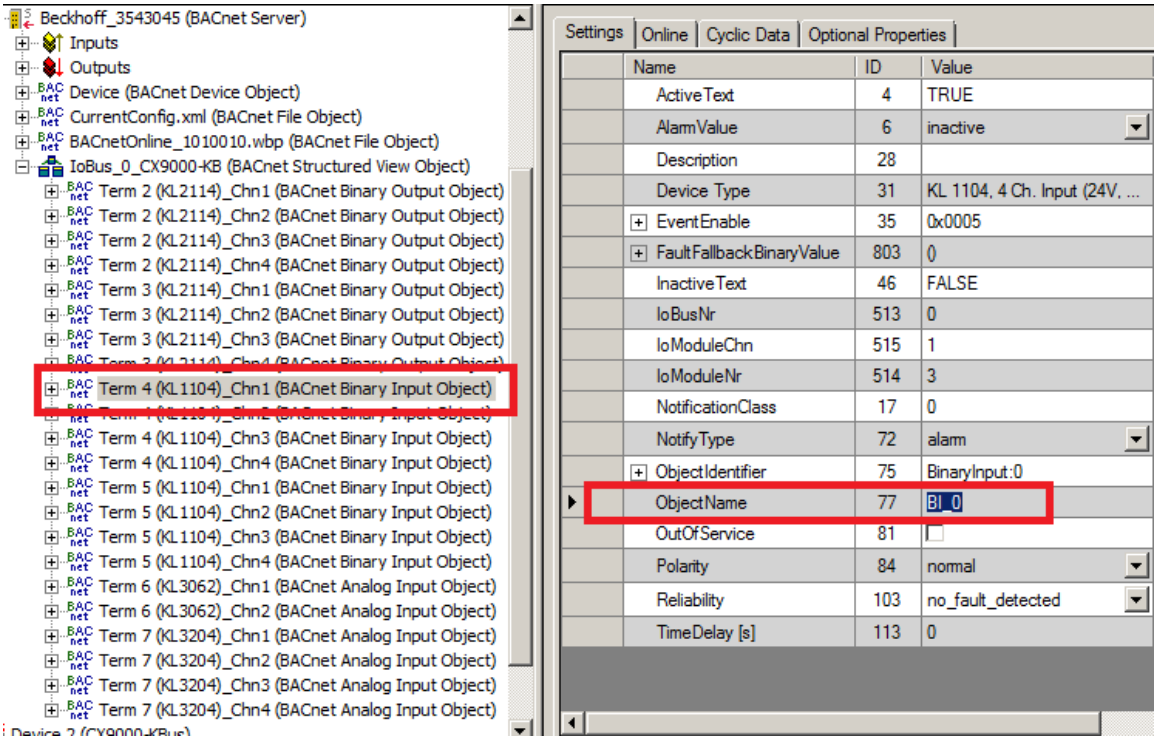


6. Important: To ensure correct linking, the object names should correspond to the PLC comment (assignment using the object ID would also be possible):

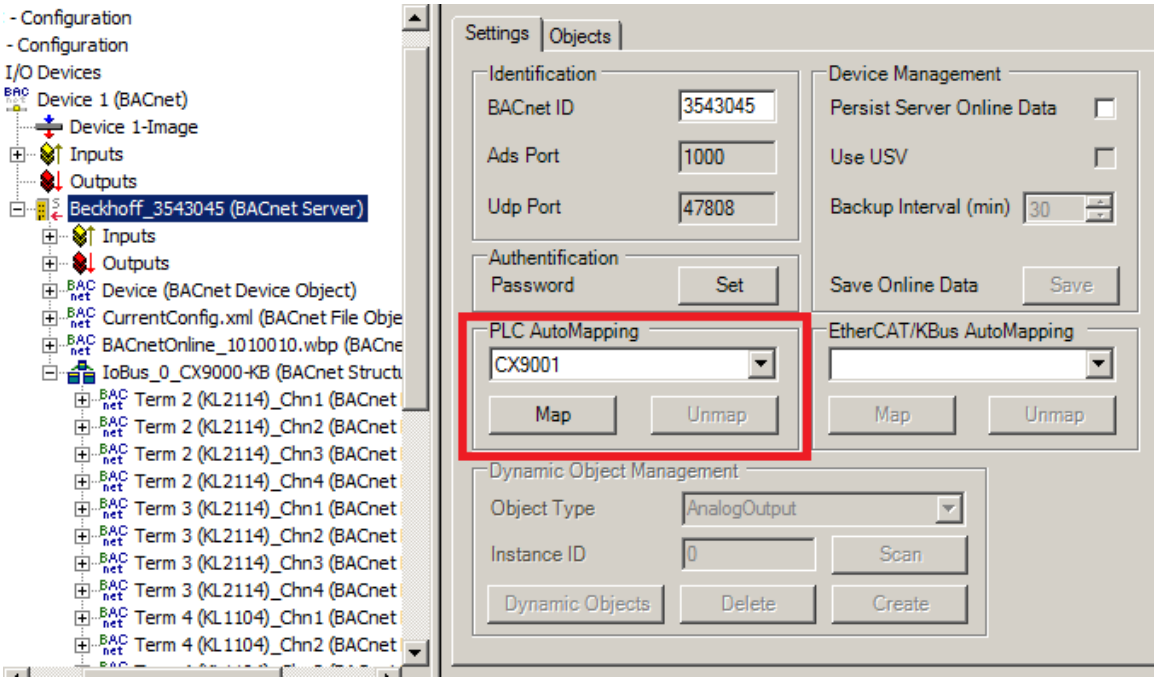
   a) Binary output "BinaryOutput: 0" is to be linked with the PLC instance "BO_0":



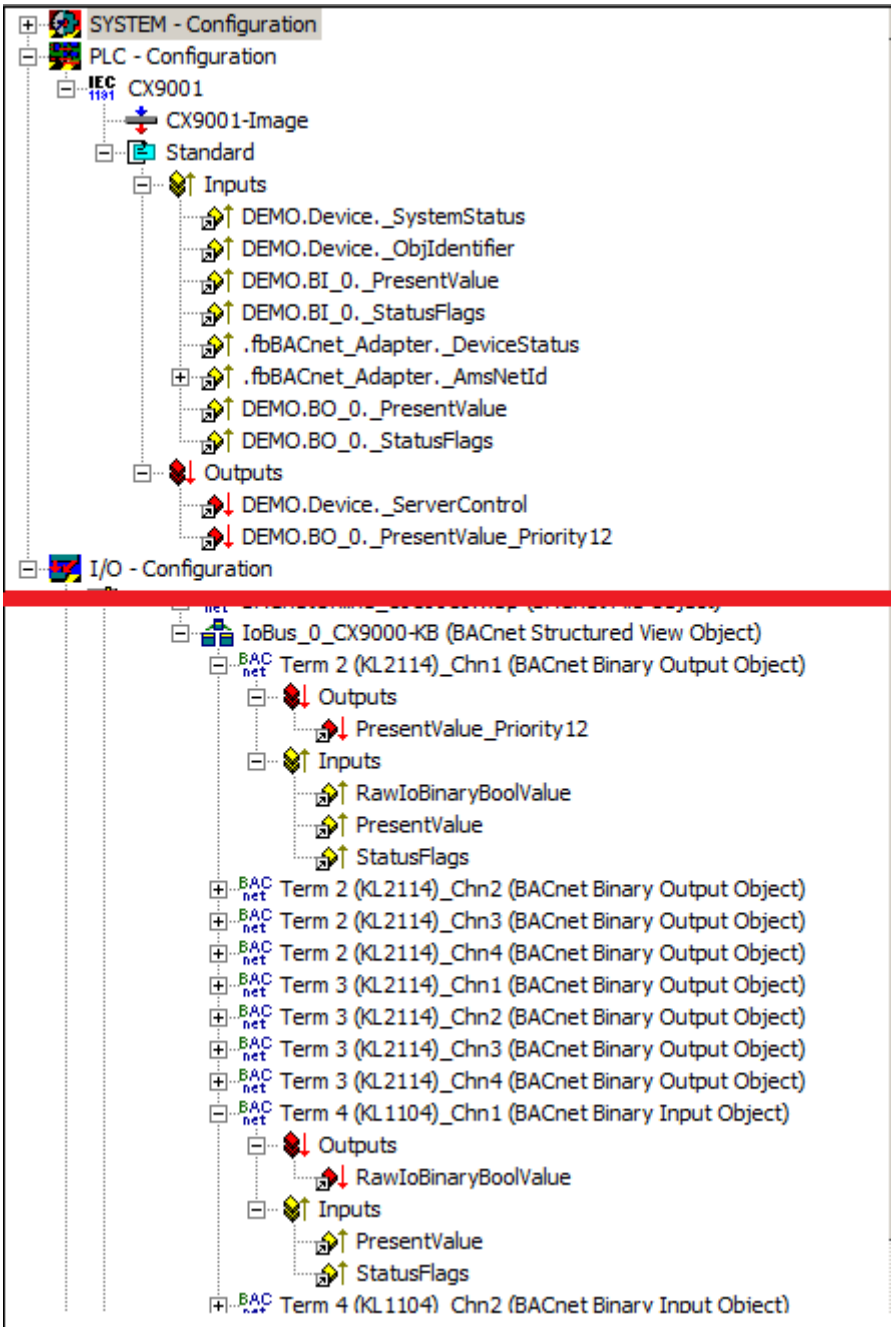
   b) Binary input "BinaryInput: 0" is to be linked with the PLC instance "BI_0":



7. Execute PLC automapping via the Settings tab of the BACnet server:

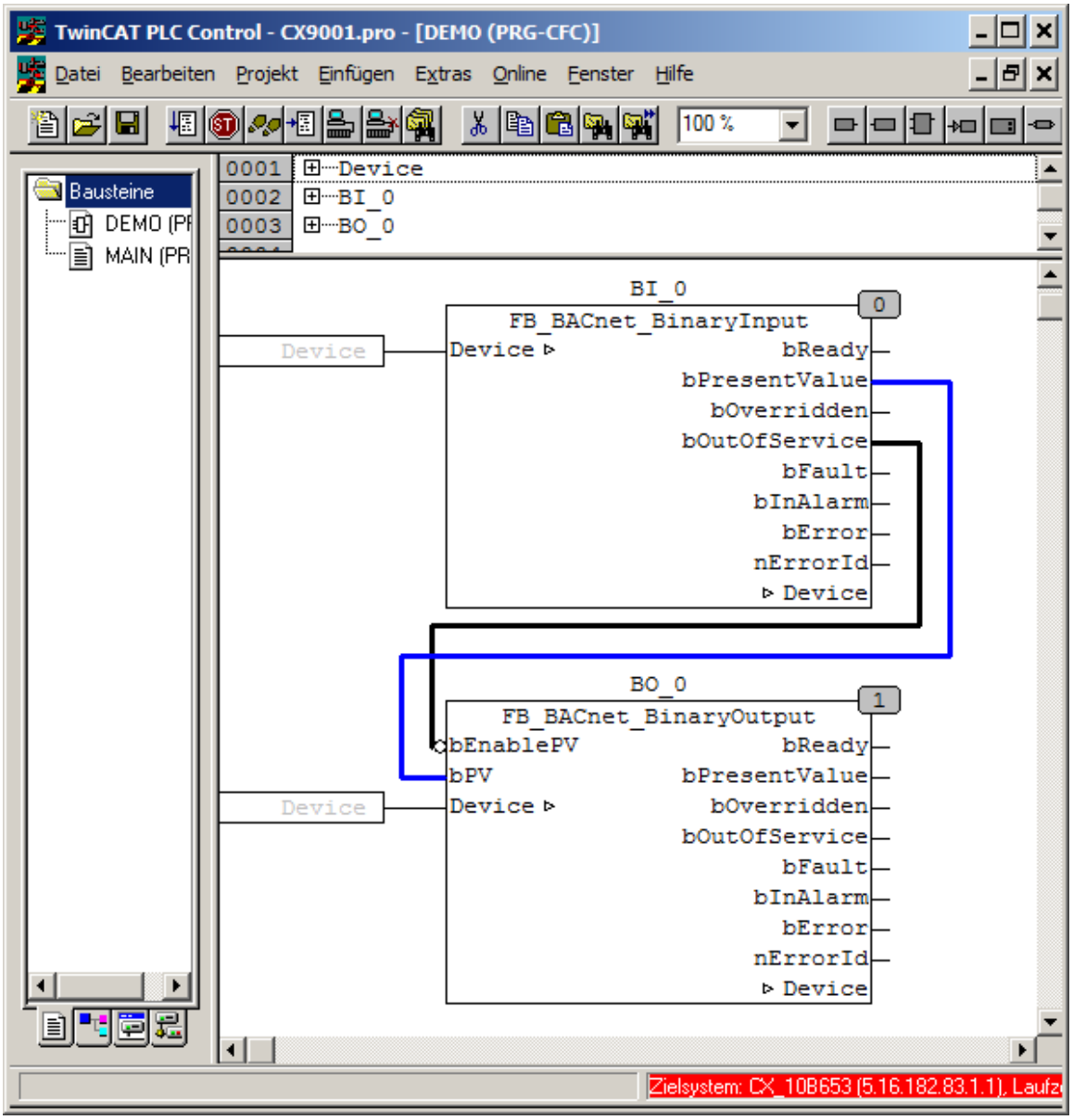


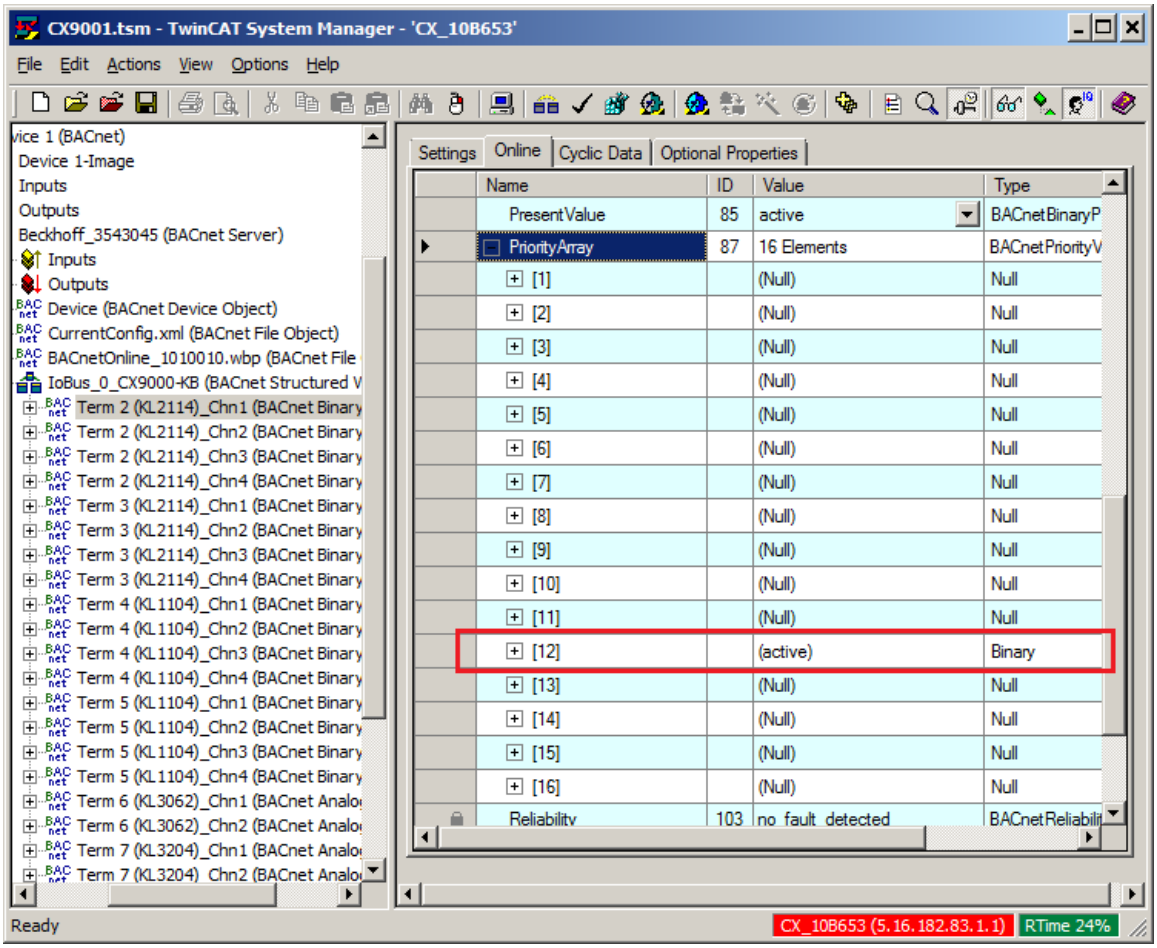
8. The process data of the BACnet objects and the corresponding data of the PLC program are then linked:



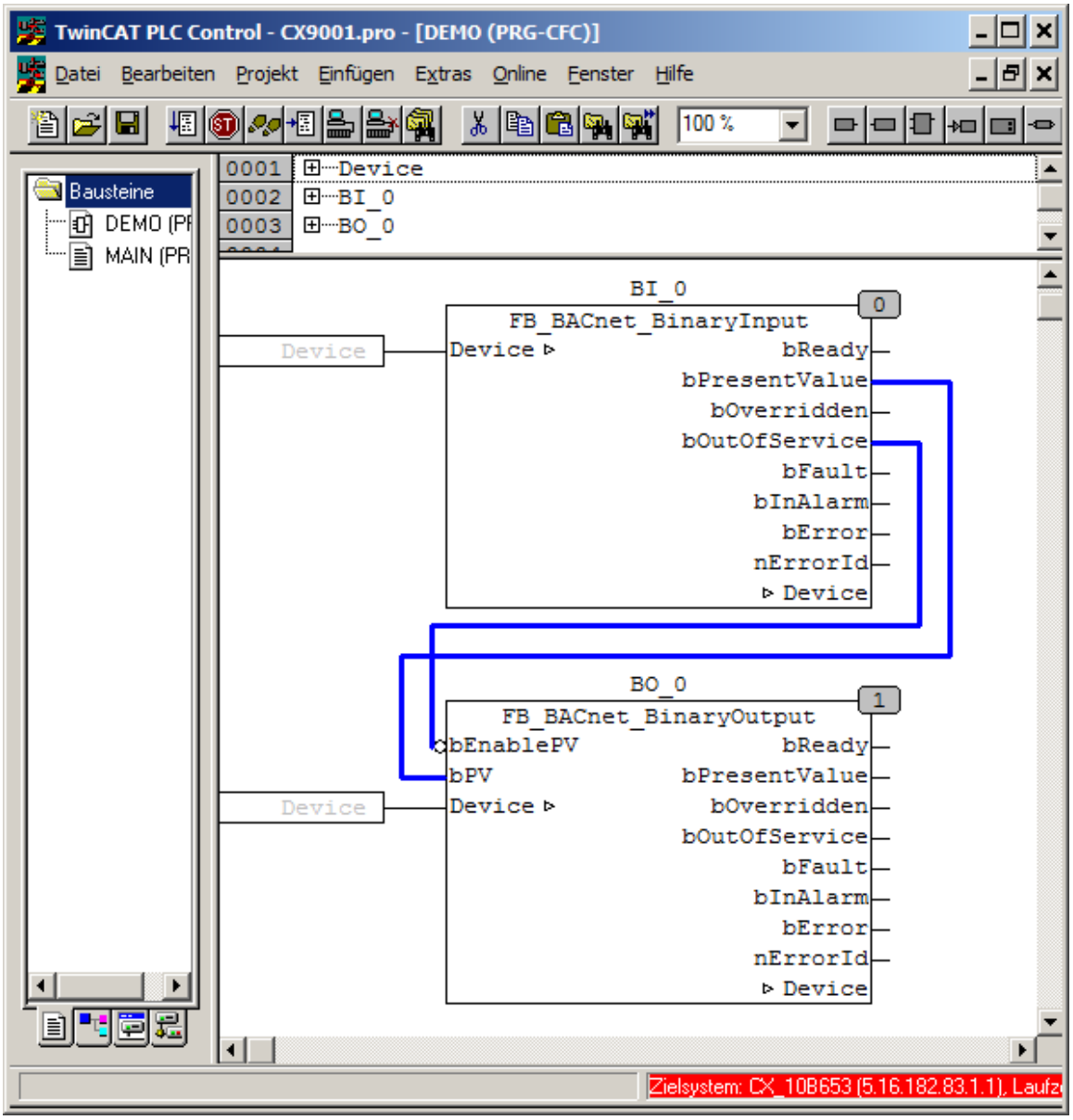
9. The configuration has to be activated (Ctrl+Shift-F4), and the PLC program has to be loaded (PLC Control via menu "Online --> Login" or F11). The PLC is then switched to Run state with F5 or "Online --> Start".
10. The different PLC states and the corresponding BACnet properties are described below:

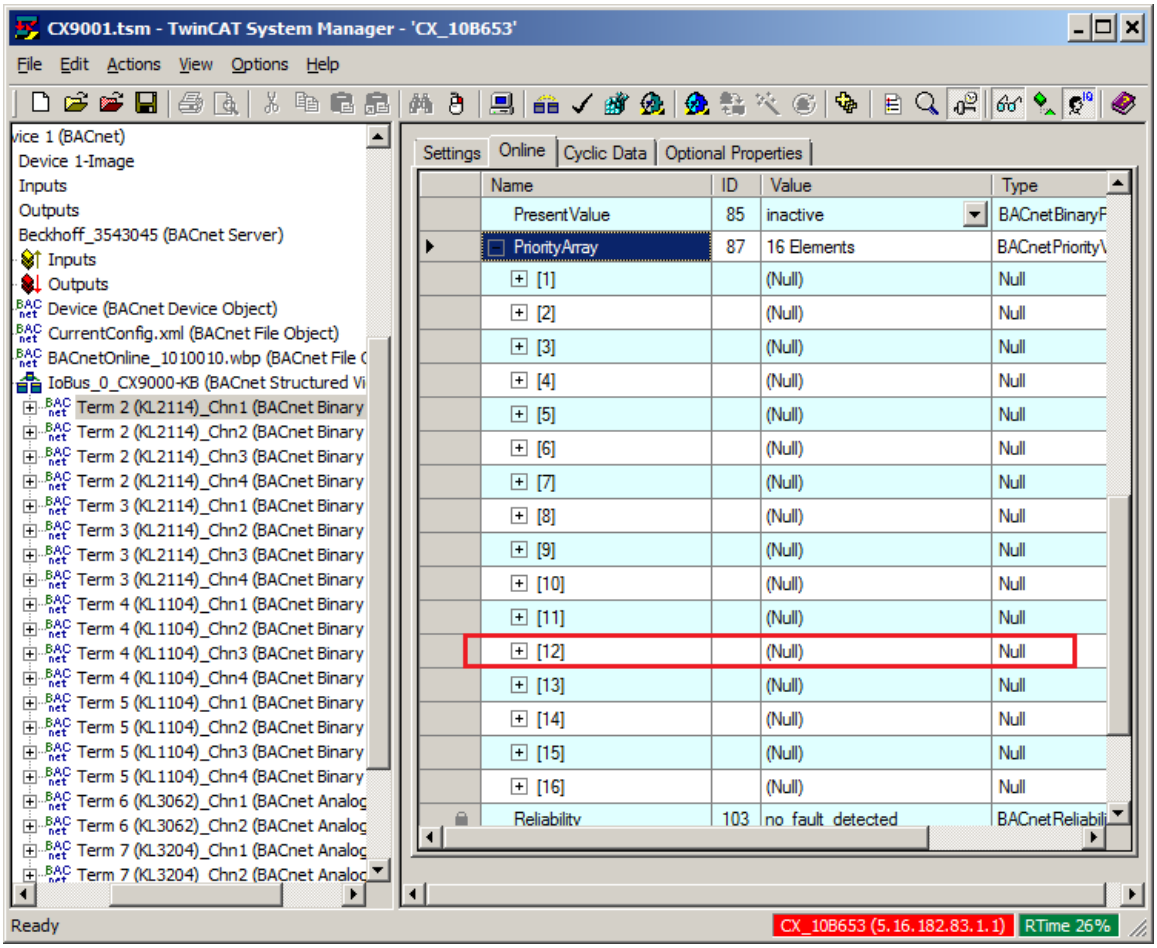
    a) **Case 1**: "BI_0" is not OutOfService and *ACTIVE* or *INACTIVE* → Priority "12" of "BO_0" is set to *ACTIVE* or *INACTIVE* and acts on the hardware terminal:


TwinCAT PLC Control - CX9001.pro - [DEMO (PRG-CFC)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
100 %
Bausteine
DEMO (PR
MAIN (PR
0001 Device
0002 BI_0
0003 BO_0
BI_0
FB_BACnet_BinaryInput
0
Device
Device
bReady
bPresentValue
bOverridden
bOutOfService
bFault
bInAlarm
bError
nErrorId
Device
BO_0
FB_BACnet_BinaryOutput
1
bEnablePV
bPV
Device
Device
bReady
bPresentValue
bOverridden
bOutOfService
bFault
bInAlarm
bError
nErrorId
Device
Zielsystem: CX_10B653 (5.16.182.83.1.1), Laufz



b) **Case 2**: "BI_0" is OutOfService → Priority "12" of "BO_0" is deleted *(Zero)* and the value from property *RelinquishDefault* of "BO_0" acts on the hardware terminal:

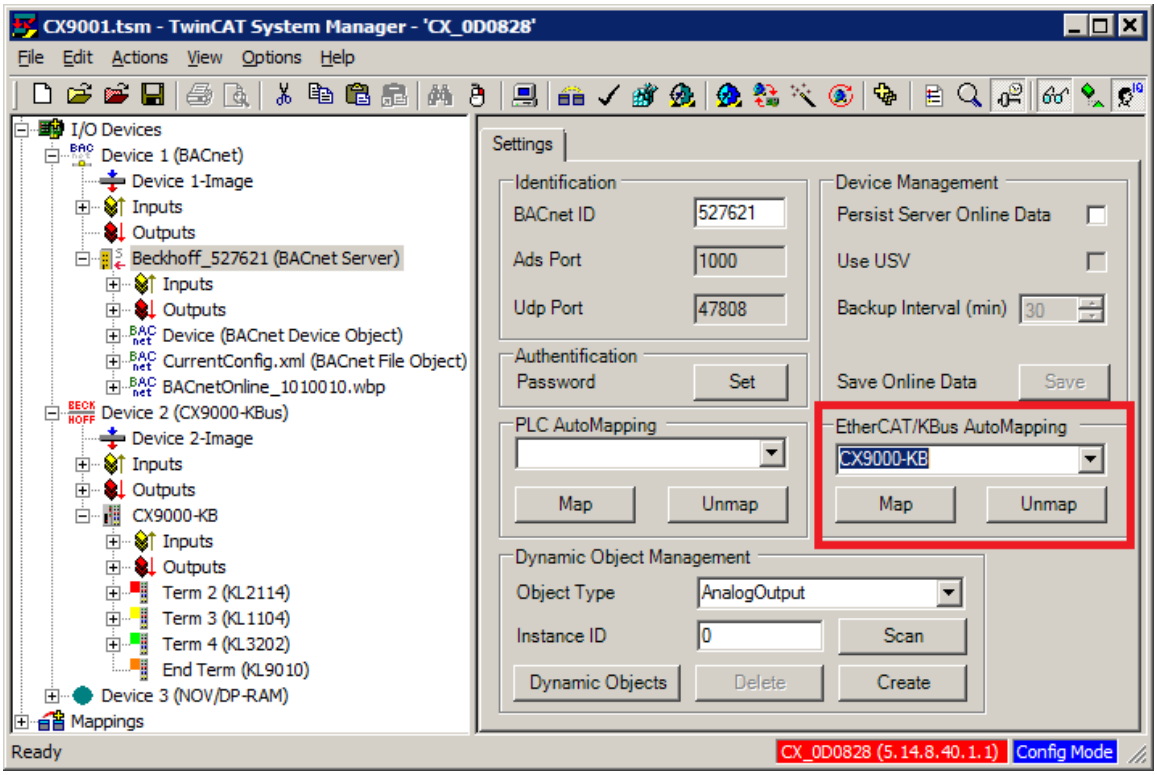

TwinCAT PLC Control - CX9001.pro - [DEMO (PRG-CFC)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
100 %
Bausteine
DEMO (PR
MAIN (PR
0001 Device
0002 BI_0
0003 BO_0
BI_0
FB_BACnet_BinaryInput
0
Device
Device
bReady
bPresentValue
bOverridden
bOutOfService
bFault
bInAlarm
bError
nErrorId
Device
BO_0
FB_BACnet_BinaryOutput
1
bEnablePV
bPV
Device
Device
bReady
bPresentValue
bOverridden
bOutOfService
bFault
bInAlarm
bError
nErrorId
Device
Zielsystem: CX_10B653 (5.16.182.83.1.1), Laufz



3.4 Example: I/O automapping

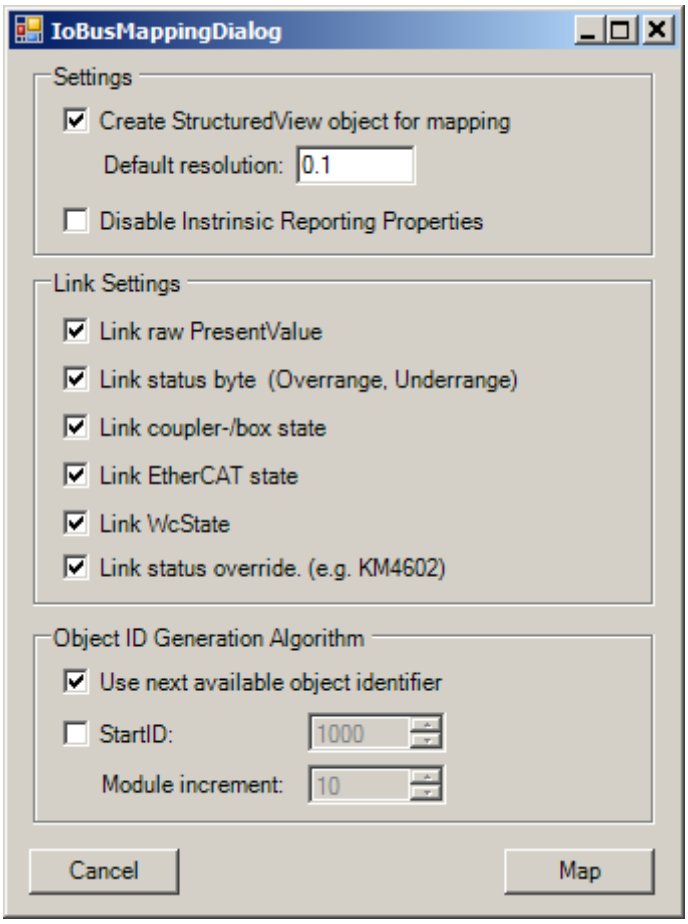
In a BACnet environment data are always represented via objects and their properties. This applies to memory states (variables) as well as hardware inputs and outputs. Essentially it means that all hardware terminals of the I/O system must be mapped via BACnet objects. The automapping option minimizes the complexity of linking between BACnet objects and hardware terminals. The procedure is described below by means of screenshots. Further technical information can be found in section "I/O automapping [▶ 72]". An example for manual linking between hardware terminals and BACnet objects can be found in section "Manual hardware linking (terminal), BACnet BinaryInput and PLC runtime [▶ 93]".

1. Create BACnet adapters and servers (see Example "Create BACnet adapters and servers [▶ 92]")
2. Create an I/O bus via the Scan function in Config mode or manually (see also section "I/O automapping [▶ 72]")
3. Activate EtherCAT or K-bus automapping
   a) Select the required bus system via the dropdown box

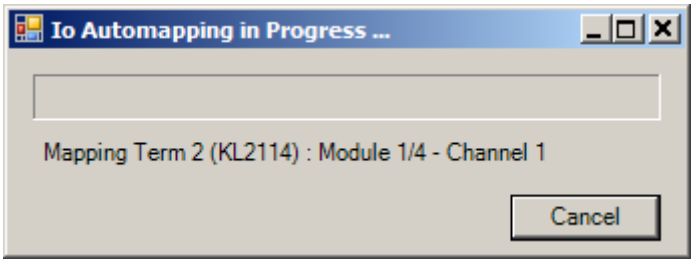


b) Press the "Map" button

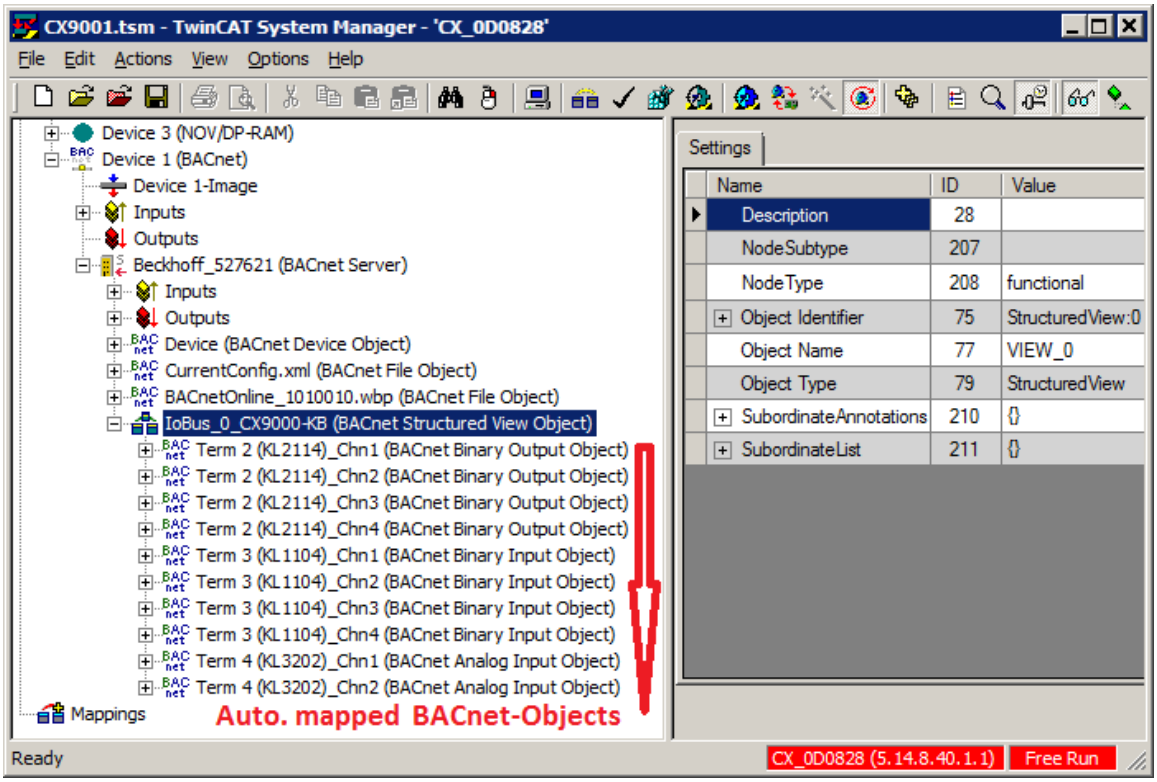
4. The mapping parameters can be set in the Automapping dialog and confirmed with the "Map" button



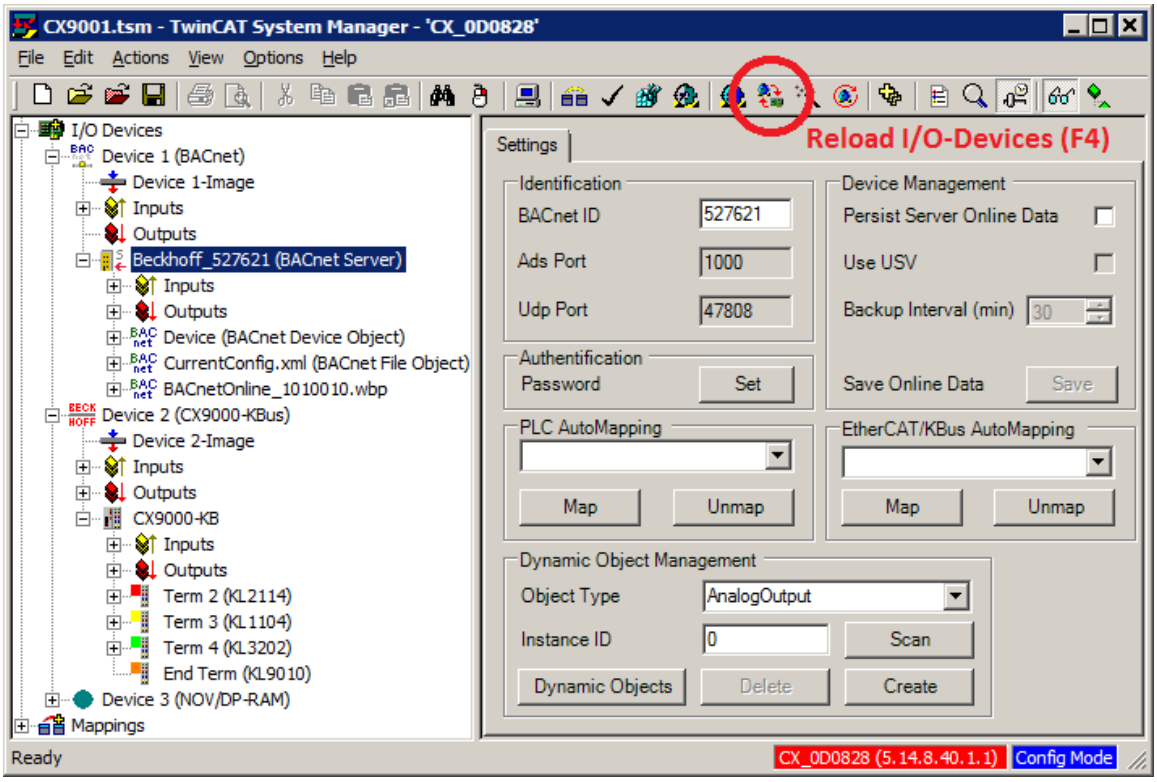
5. A separate dialog indicates the mapping progress



6. The BACnet objects representing the the terminals of the I/O bus are located under a Structured View



7. The objects are available "Online" once the configuration has been loaded with "Reload I/O Devices" and the Free RUN mode has been activated



3.5 Example: PLC automapping

In a BACnet environment, data are represented via objects and their properties. This applies to memory states (variables) and PLC signals that are meant to be visible in BACnet. Essentially this means that all states of the PLC runtime which are meant to be visible in BACnet must be mapped via BACnet objects. The automapping option minimizes the complexity of linking between BACnet objects and SPS signals. The procedure is described below by means of screenshots. For further information please refer to section "PLC automapping [▶ 62]".

The example https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749044491/.zip can be downloaded from herehttps://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749044491/.zip.

1. Creating BACnet adapters and servers (see: "Example: Create BACnet adapters and servers [▶ 92]")
2. Create a PLC project with the following block instances

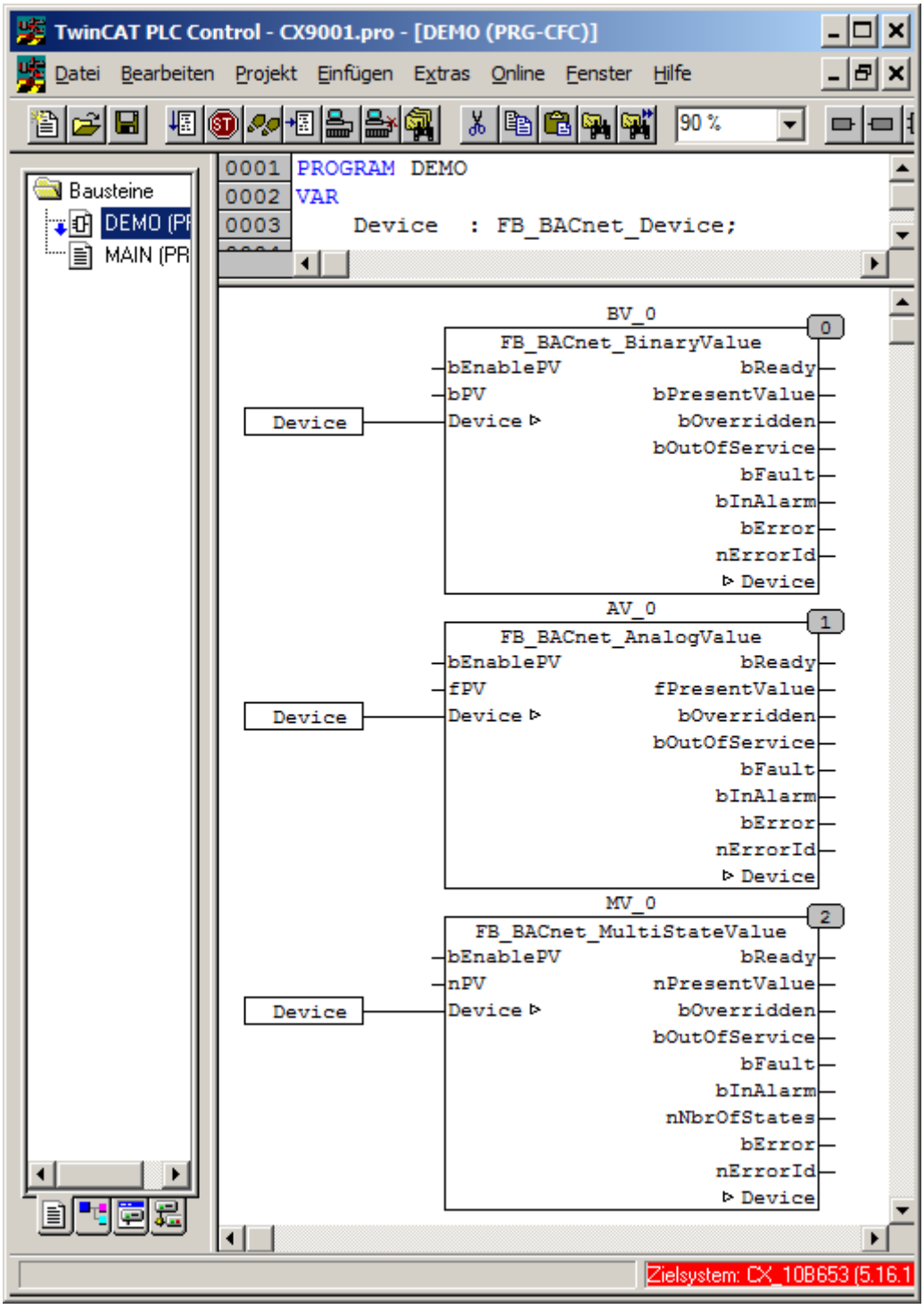
```
0001 PROGRAM DEMO
0002 VAR
0003     Device   : FB_BACnet_Device;
0004     BV_0     : FB_BACnet_BinaryValue;
0005     AV_0     : FB_BACnet_AnalogValue;
0006     MV_0     : FB_BACnet_MultiStateValue;
0007 (* FB_BACnet_Device, FB_BACnet_BinaryValue,
0008     FB_BACnet_AnalogValue and FB_BACnet_MultiStateValue
0009     are implemented by the library "TcBACnet.lib" *)
0010 END_VAR
```

   The aim is to create the PLC instances of the BACnet blocks as BACnet objects in the System Manager configuration, pre-configure the process data and properties and link the process data between the PLC and the objects. The required process data definitions (AT%I* / AT%Q*) are already included in the library blocks of the PLC library. Initialisation of the object properties is described below.
3. Add automapping comments to the PLC instances and call up the block instances in the PLC program

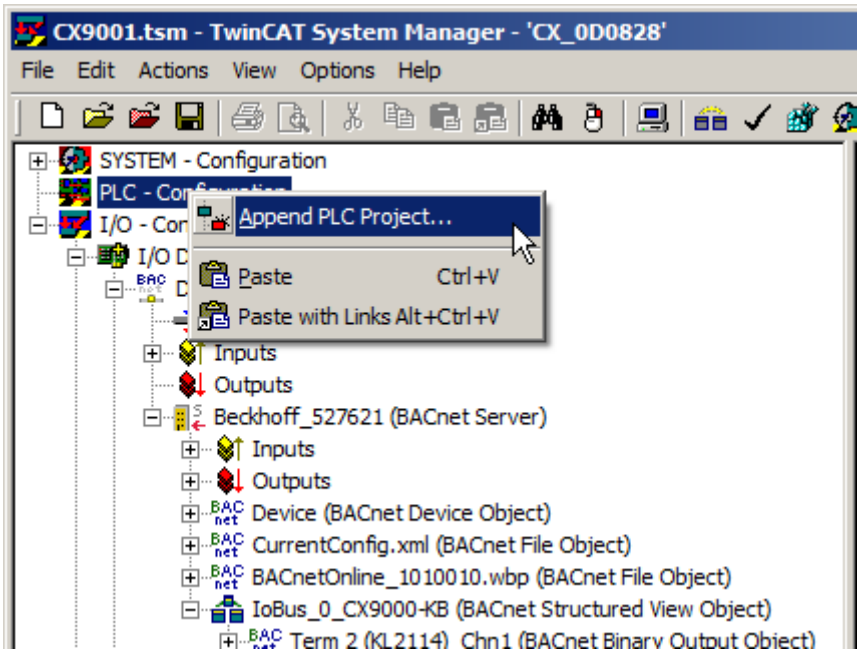
```
PROGRAM DEMO
VAR
    Device   : FB_BACnet_Device;

    BV_0     : FB_BACnet_BinaryValue;
    (* ~(BACnet_ObjectName       : DEMO.BV_0 : nolink)
        (BACnet_ObjectIdentifier: 10        : nolink)
        (BACnet_ActiveText       : An        : nolink)
        (BACnet_InactiveText     : Aus       : nolink)
        (BACnet_MinimumOntime    : 2         : nolink)
        (BACnet_MinimumOfftime   : 5         : nolink)
        (BACnet_RelinquishDefault: active    : nolink) *)

    AV_0     : FB_BACnet_AnalogValue;
    (* ~(BACnet_ObjectName       : DEMO.AV_0 : nolink)
        (BACnet_Description      : Ein Test-Objekt. : nolink)
        (BACnet_HighLimit        : 100       : nolink)
        (BACnet_LowLimit         : 0         : nolink)
        (BACnet_LimitEnable      : 0xC006    : nolink)
        (BACnet_Units            : Other_percent : nolink) *)

    MV_0     : FB_BACnet_MultiStateValue;
    (* ~(BACnet_ObjectName       : DEMO.MV_0 : nolink)
        (BACnet_NumberOfStates   : 5         : nolink)
        (BACnet_AlarmValues      : {2;4}     : nolink)
        (BACnet_FaultValues      : {1;5}     : nolink)
        (BACnet_StateText        : {eMin;wMin;OK;wMax;eMax} : nolink) *)

(* FB_BACnet_Device, FB_BACnet_BinaryValue,
    FB_BACnet_AnalogValue and FB_BACnet_MultiStateValue
    are implemented by the library "TcBACnet.lib" *)
END_VAR
```

Note:

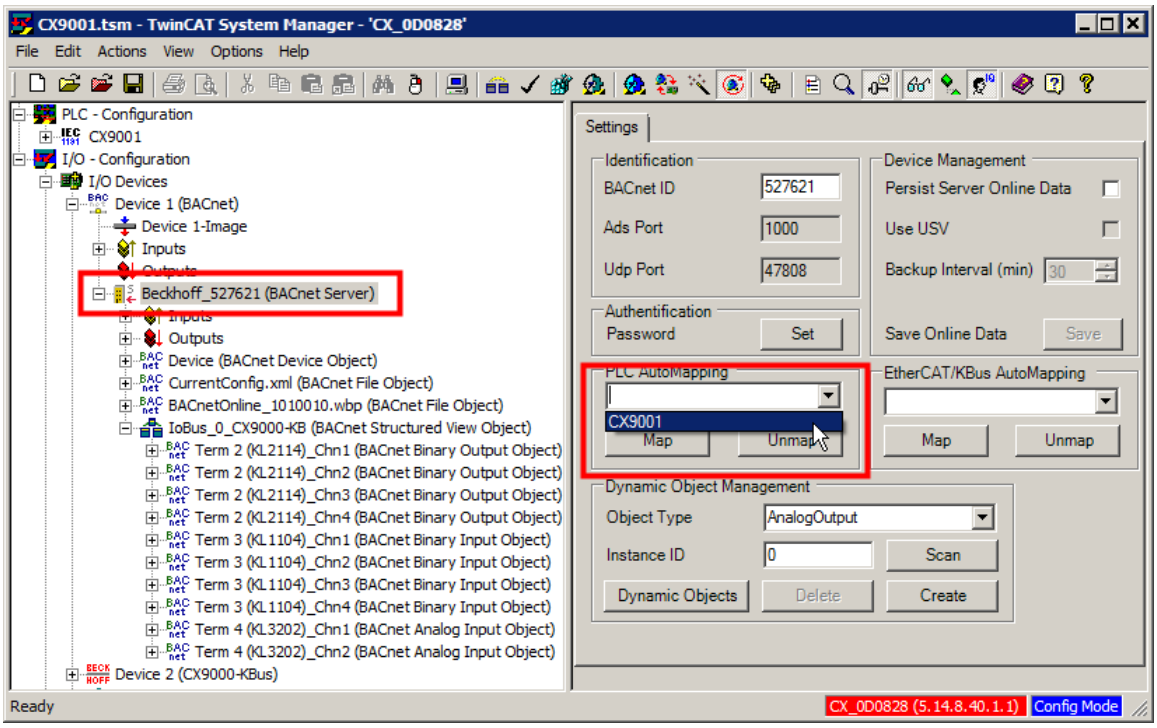
- The PLC program "DEMO" is called from program "MAIN". The program "MAIN" is entered as a task in the task configuration
- The test program contains the library "TcBACnet.lib" as reference



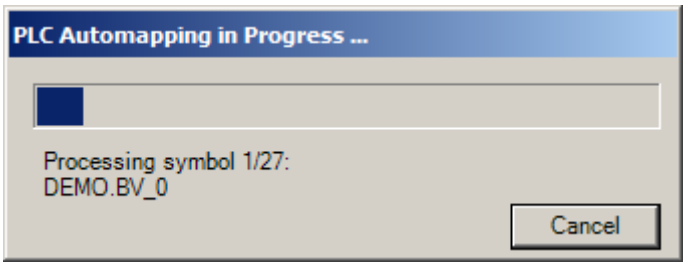
4. Compile the PLC project (CTRL+F8)
5. Add the PLC project to the hardware configuration



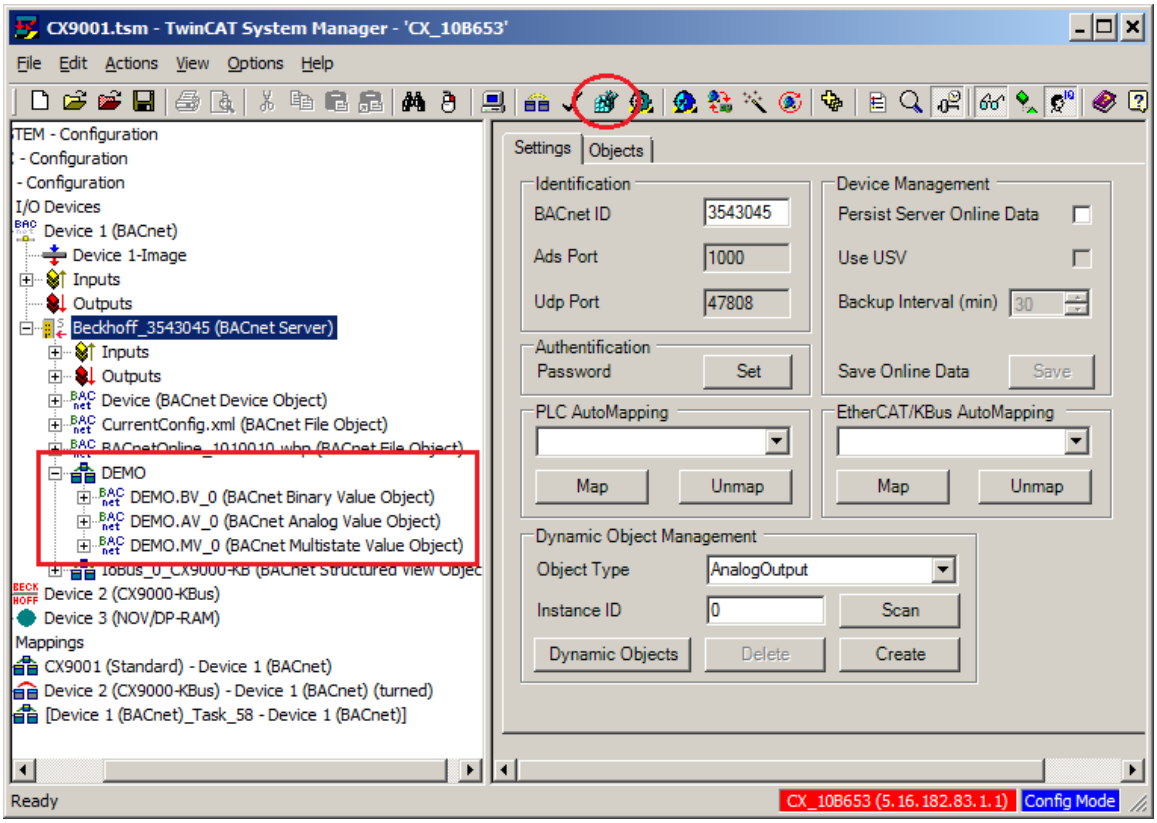
6. In the Settings tab of the BACnet server select the PLC configuration and execute automapping by pressing the "Map" button



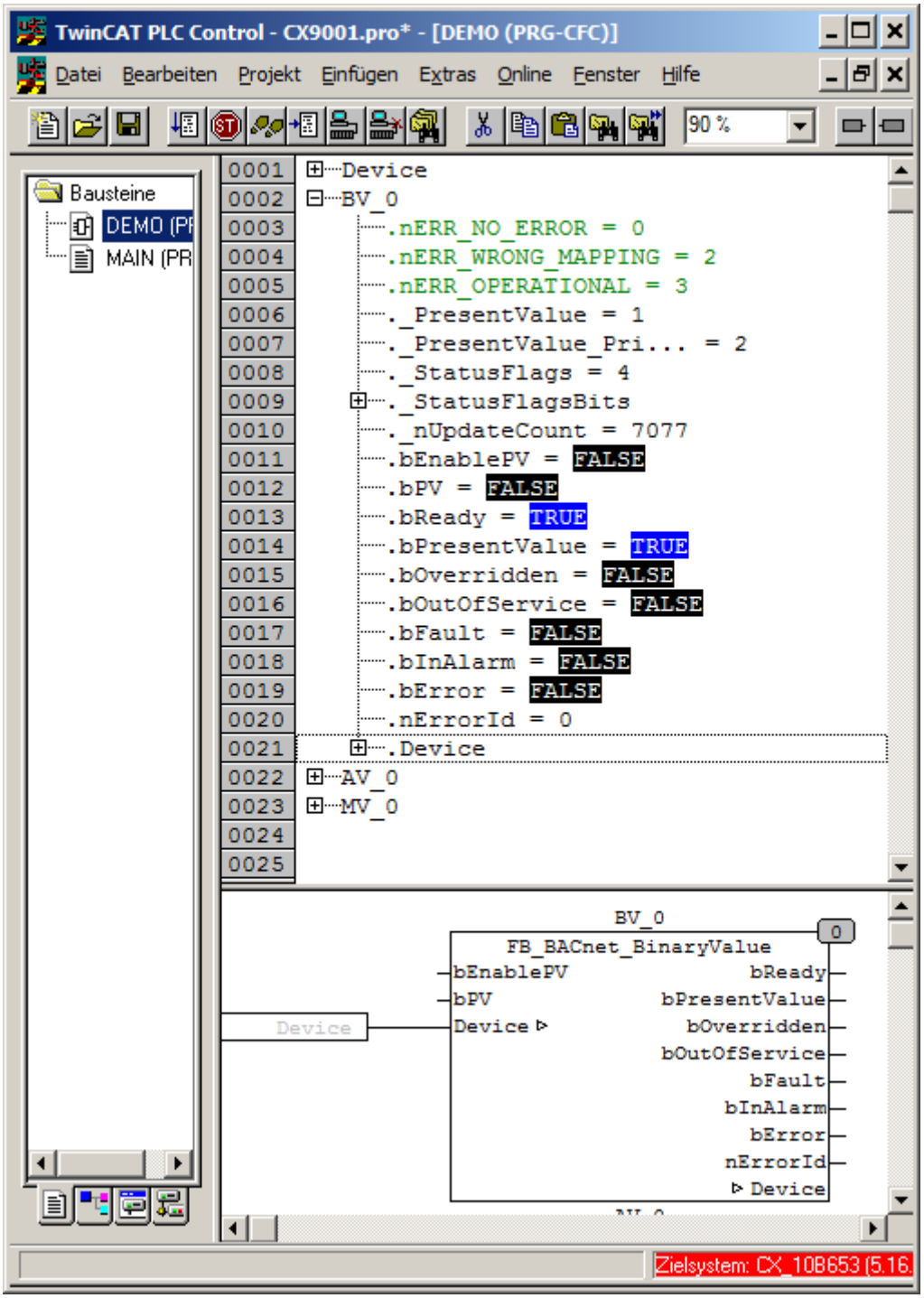
7. Wait until the mapping is complete (dialog with progress bar appears)



8. The links have now been created. Finally, activate the configuration by pressing the button "Activate Configuration" button in the toolbar



9. Log into the PLC (F11) and load the program (then start the PLC project with F5, if required)



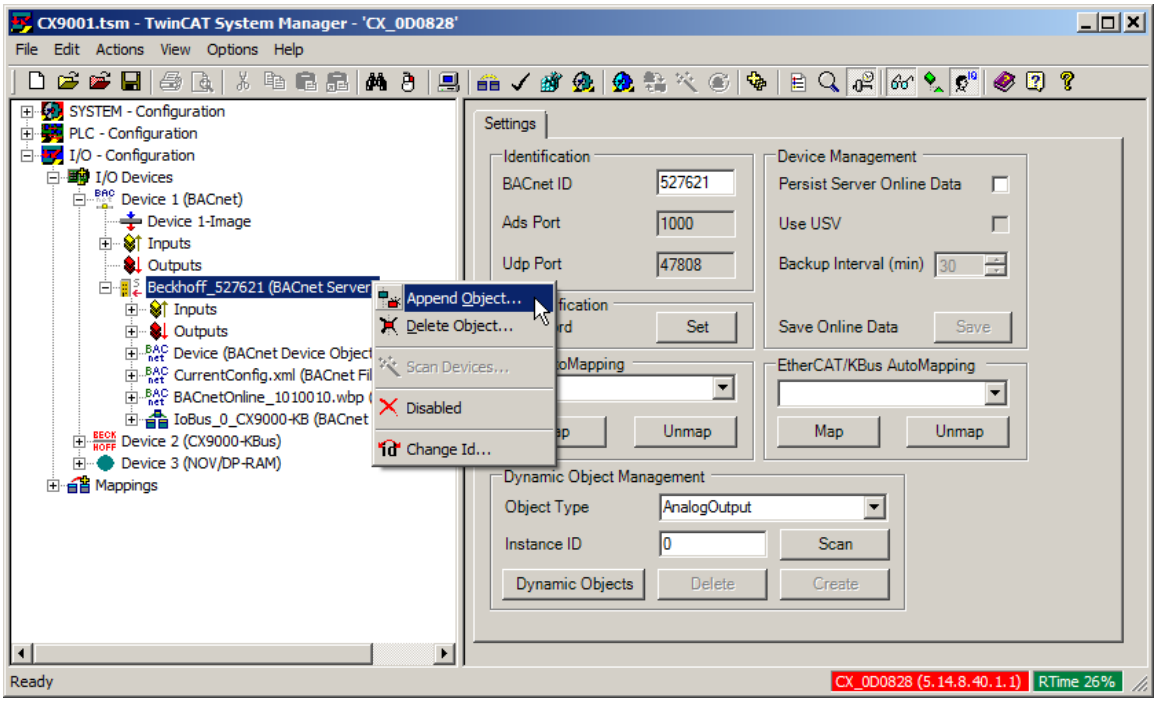
3.6 Example: BACnet day/night control

The example below illustrates how to configure simple day/night control with the aid of a BACnet schedule and output object. This can be used for lighting control, for example, which is active during the night. The time for activating the binary output is set to 22:00. The output is reset at 4:00.

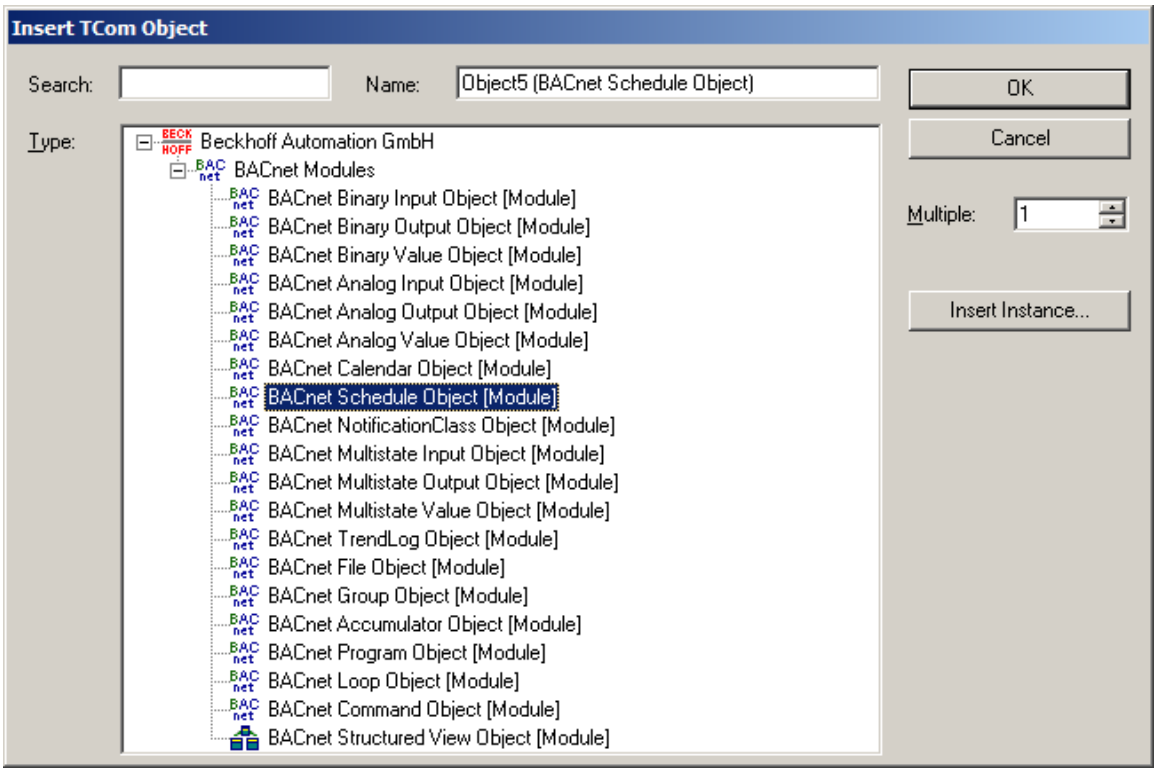
The example https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749045899/.zip can be downloaded here https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749045899/.zip.

1. Create a BACnet adapter and server (see "Example: Create BACnet adapters and servers [▶ 92]")
2. Adding an I/O bus via I/O automapping (see: "Example: I/O automapping [▶ 115]")
3. Creating a schedule object under the server

a) Right-click on the server and select "Append Object..." to create a new BACnet object

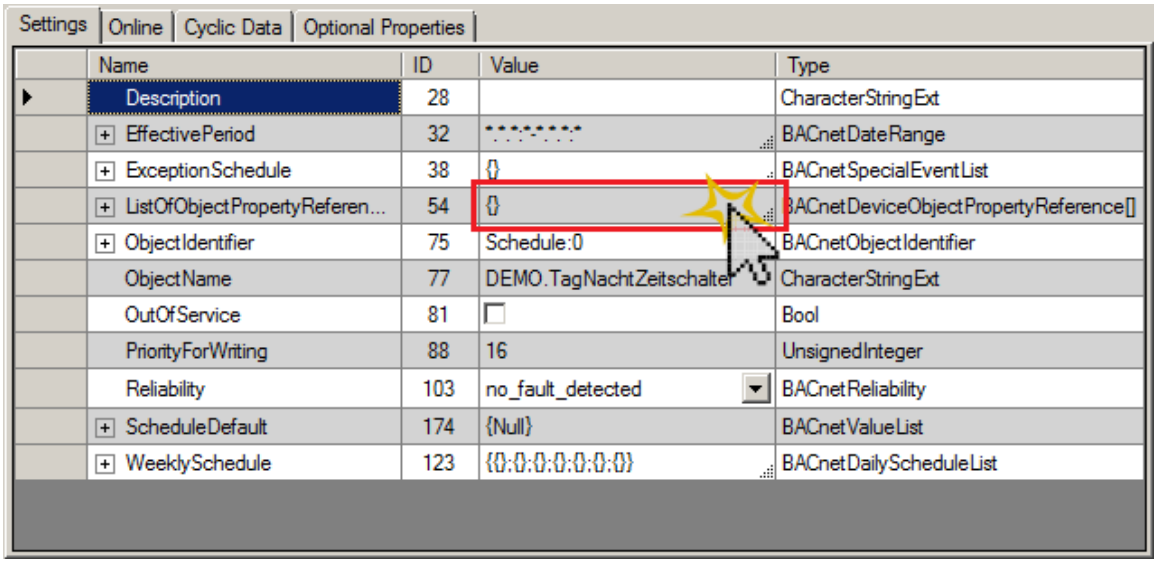


b) Select module "BACnet Schedule Object" and add with "OK"

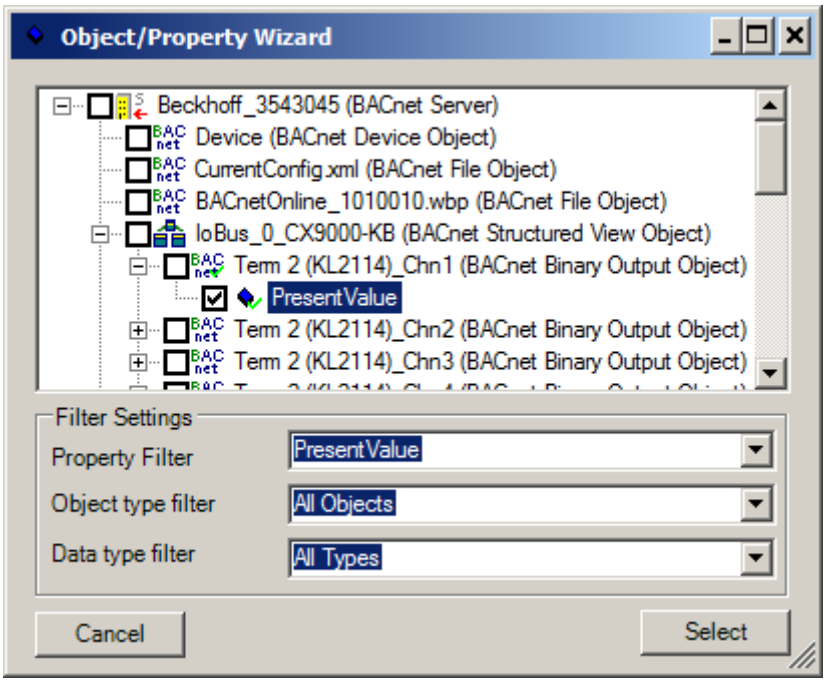


4. The object of type BACnetBinaryOuput, which represents the binary output for the lighting control, is entered as object reference for writing the schedule value (*ACTIVE* / *INACTIVE*):

   a) Double-click on the property *ListOfObjectReferences*:

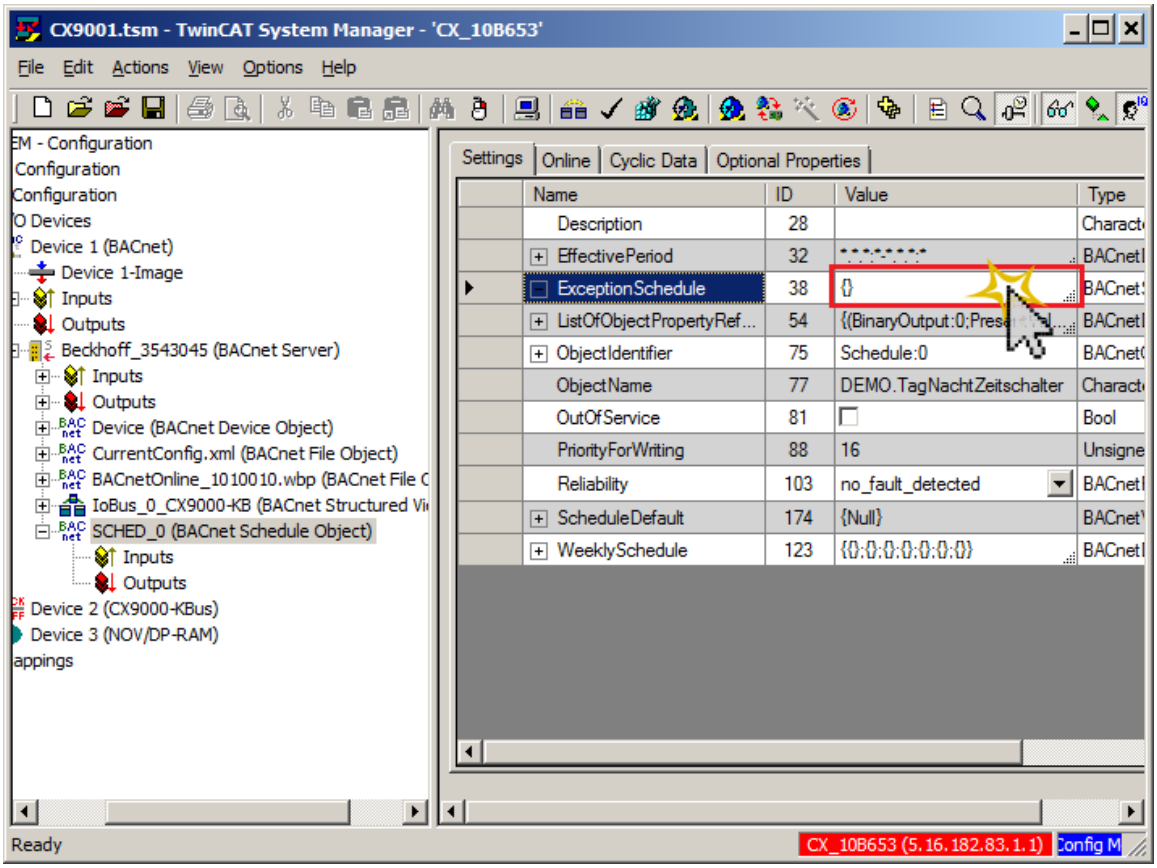


b) Then select the *PresentValue* of the required *BinaryOutput* object:

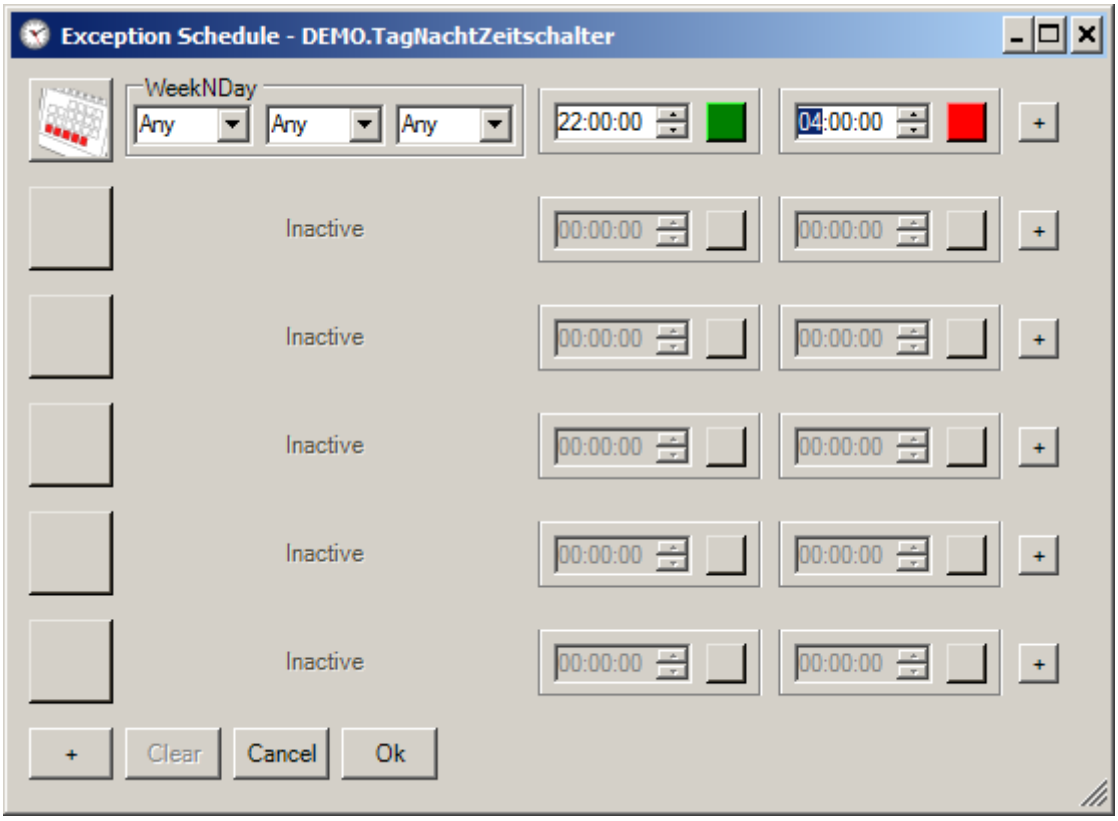


The selection of the object reference is important for the subsequent steps. It determines the data type of the TimeValue entries in the property ExceptionSchedule for the following wizard. If the PresentValue of a binary* object is selected, the data type BACnetBinaryPV is otherwise added to Bool.

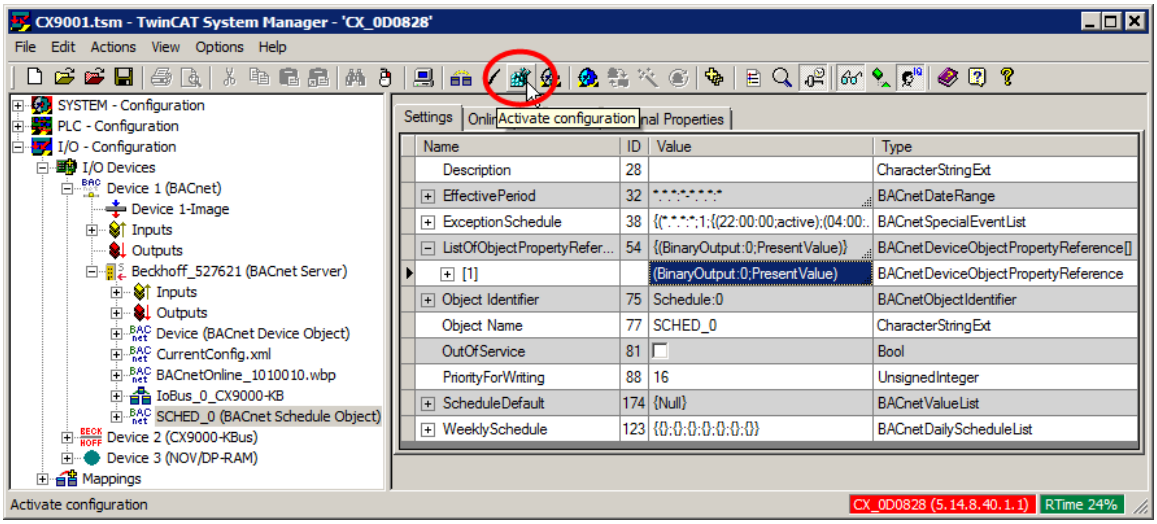
5. Double-click on the property ExceptionSchedule:



6. The wizards facilitates configuration of the switch-on and switch-off times:



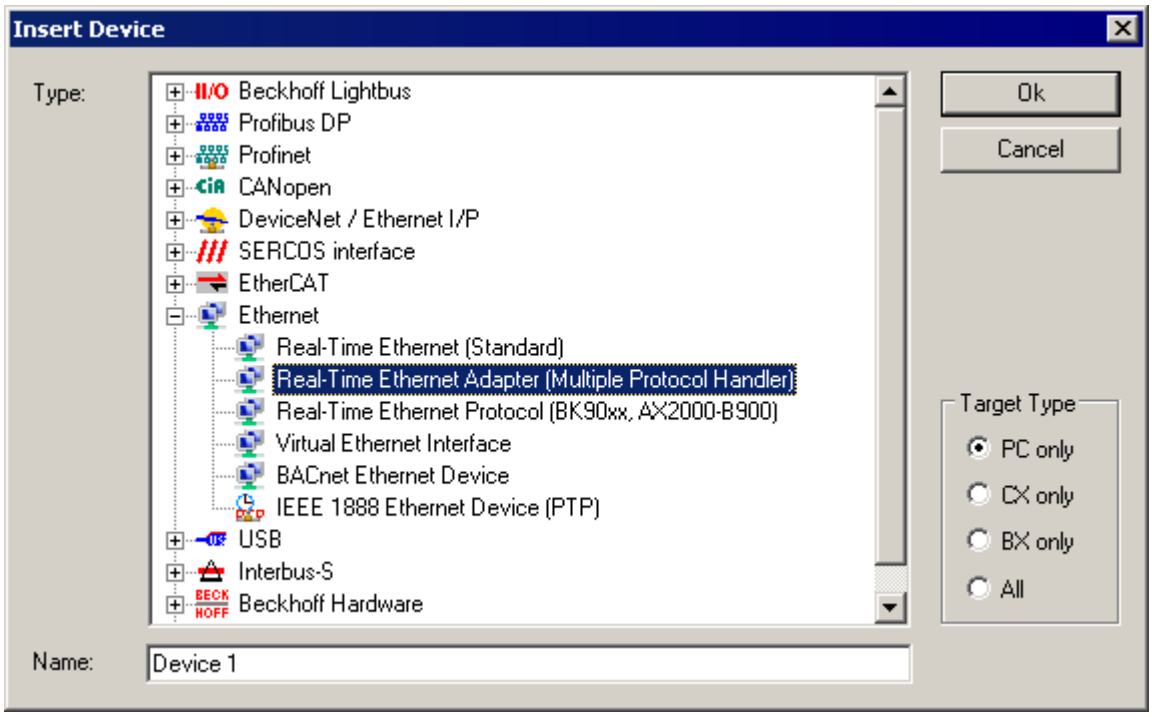
7. The configuration now has to be loaded into the target system by clicking on "Active configuration" on the toolbar:



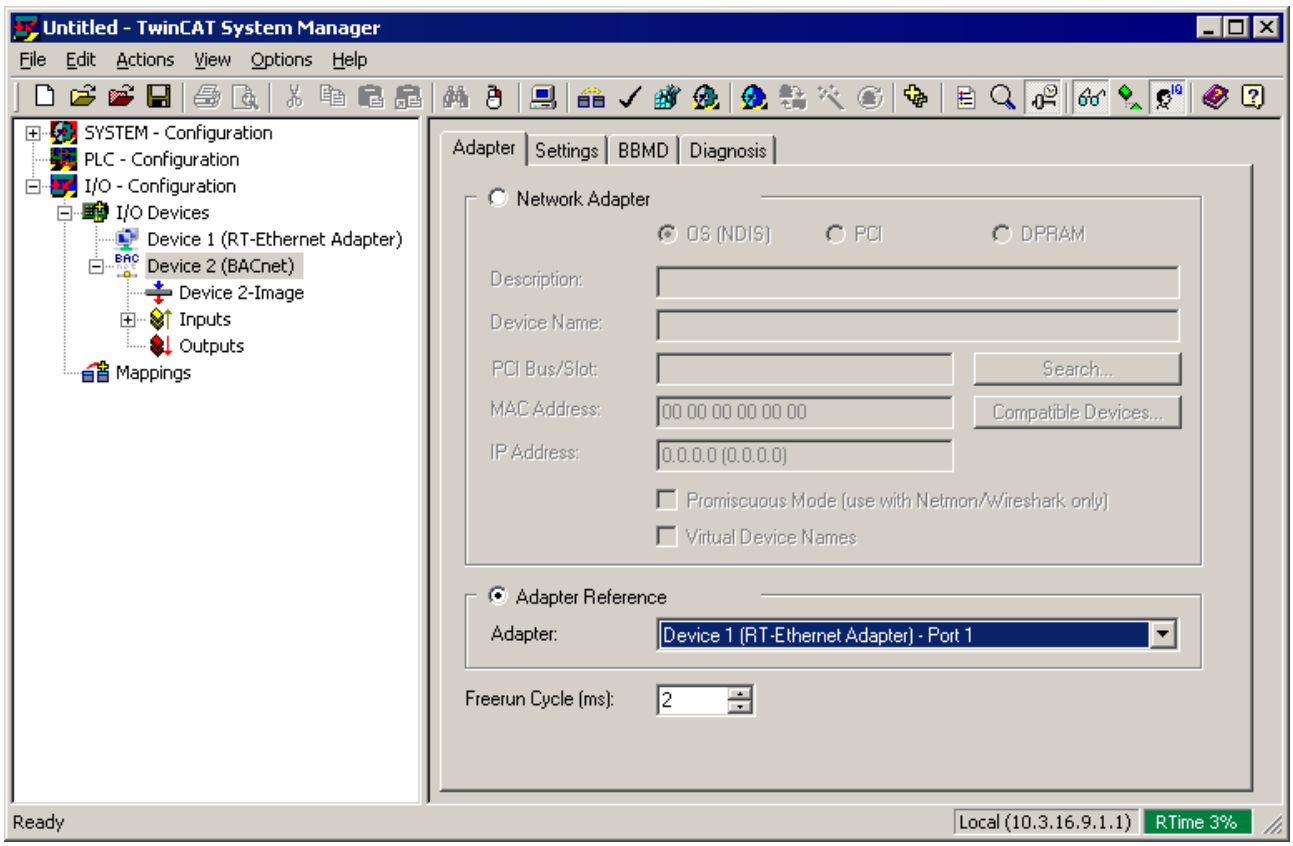
3.7 Example: Operating BACnet and BK90XX via an Ethernet interface

This section explains how a BACnet device can be operated in parallel with one or several BK90XX (real-time Ethernet) via an Ethernet interface.

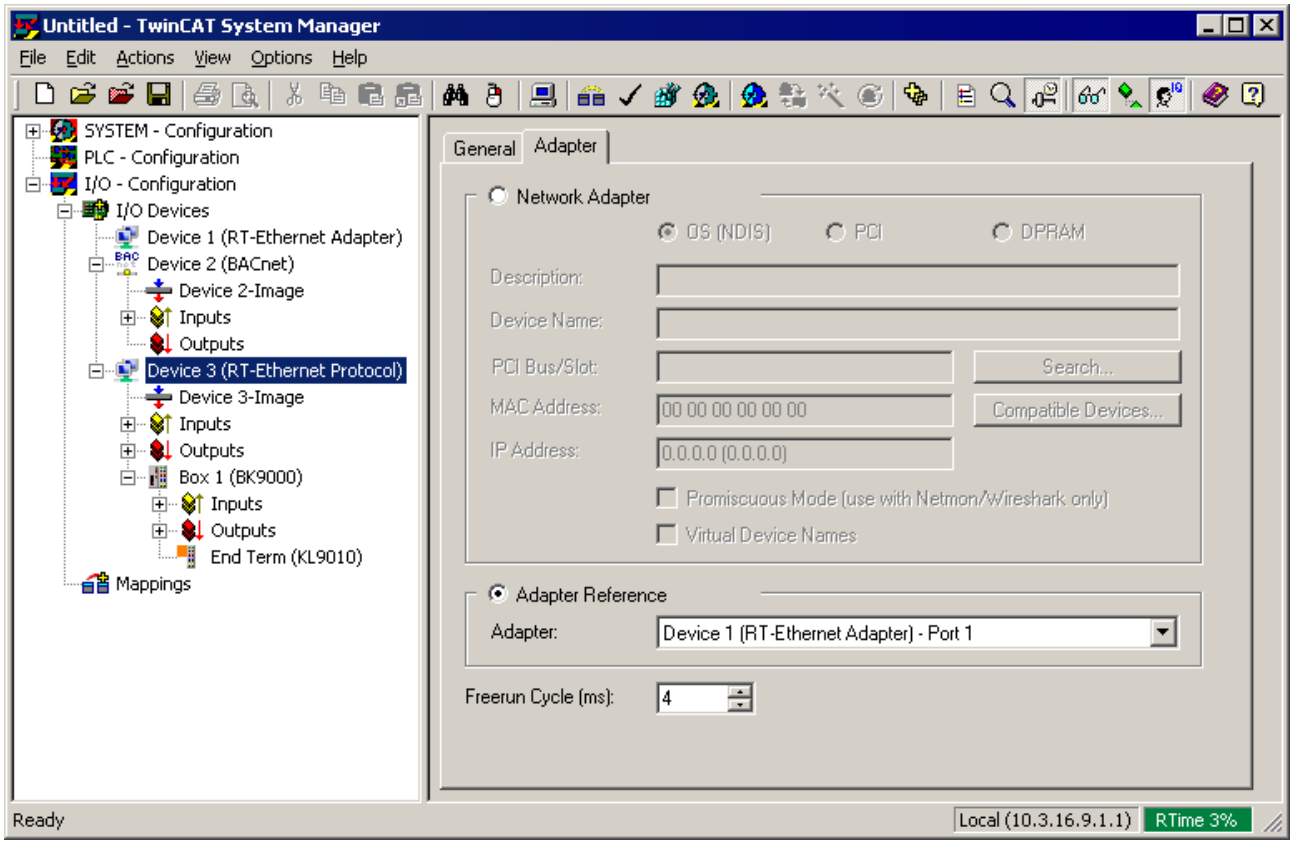
First, a **real-time Ethernet adapter (multi-protocol handler)** is created and linked with the required Ethernet interface:



Then a BACnet device is created and used as Ethernet adapter **Port 1** for the multi-protocol handler:



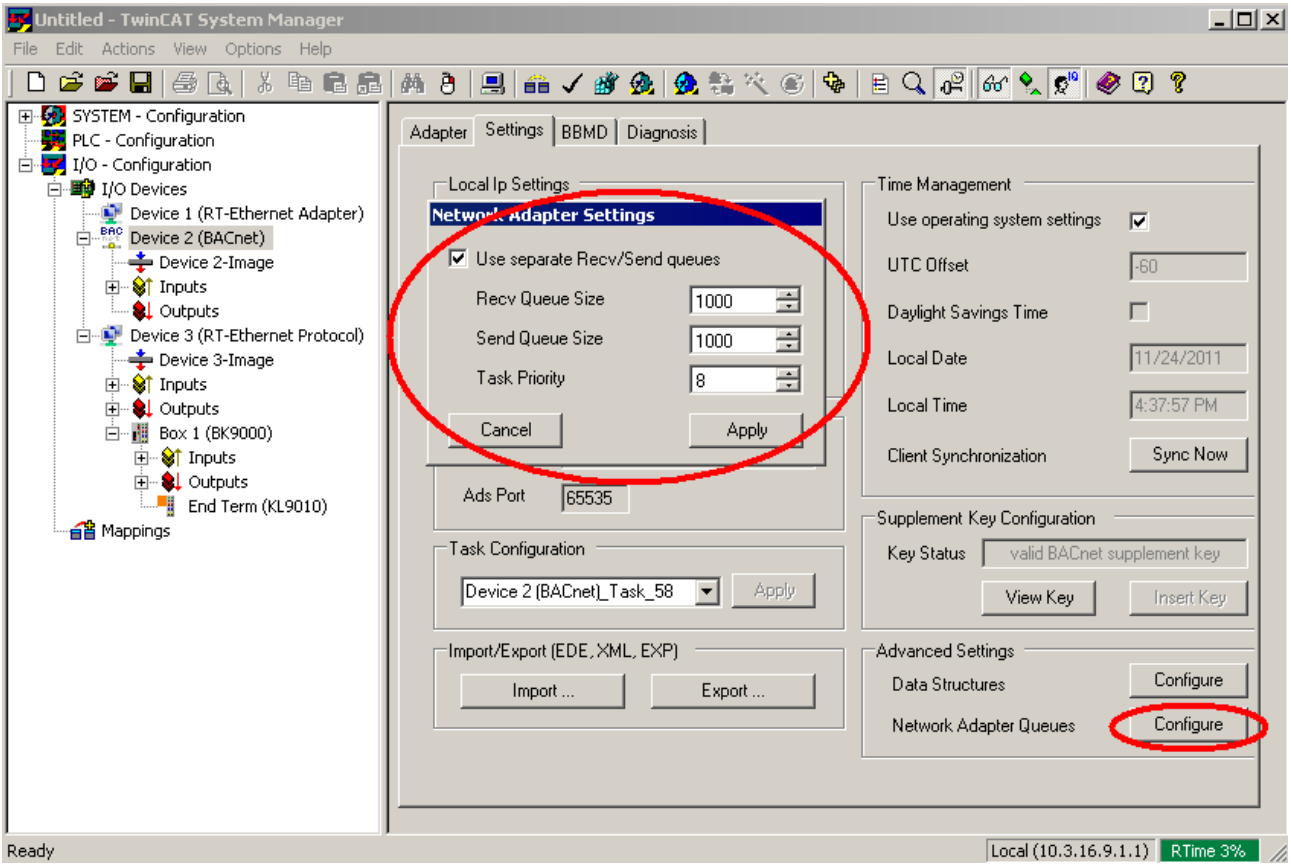
The procedure for the real-time Ethernet protocol is similar:



BACnet and real-time Ethernet can now be operated in parallel via the same Ethernet interface. (e.g. scanning in of devices etc.)

If the real-time Ethernet protocol device is not controlled via BACnet but through synchronous mapping from a PLC, for example, the network queues must be activated in the BACnet device. The reason is that internal BACnet functions can only be called up via the BACnet task. Synchronous mapping of the real-time Ethernet protocol device results in BACnet being called up in the context of the linked task. In this case the network queues decouple internal BACnet functions from this task.

The network queues can be activated in the "Settings" dialog of the BACnet device:



The procedure described here can also be used for configuring several BACnet servers (devices) on a network interface. For each BACnet server a BACnet device must be created and linked with the port of the multi-protocol handler. In this way several BACnet servers can be operated in parallel - the IP address then must be adapted for each associated BACnet device, since the BACnet servers are distinguished through the IP address.

3.8 Example: NotificationClass and NotificationSink

The following is an example of using the *NotificationClass* in conjunction with a *NotificationSink*. This "collects" the events of 3 objects (BV, AV, MV) and sends them to the local process 10 (*NotificationSink*) of the local BACnet device (event notification by intrinsic reporting).

In addition the *PresentValue* of the *AnalogValue* object should be sent to the *NotificationSink* if the value changes by 0.1 (COV notification).

The example https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749047307/.zip can be downloaded here https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749047307/.zip. The included TSM file can be used for testing purposes without loading the PLC.

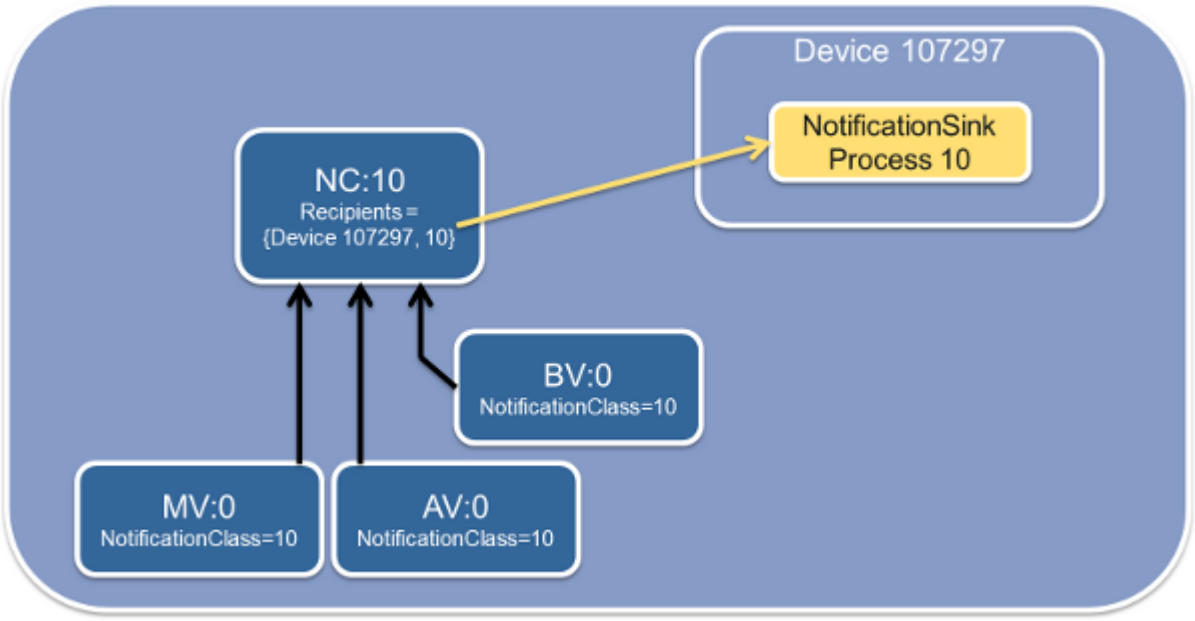


Fig. 1: Figure 1: Schematic diagram of an NotificationClass example configuration, its event-generating objects (AV, MV, BV) and the NotificationSink.

The objects are configured via the declaration in the PLC program:

```
VAR
 viewDummy : BOOL; (* ~( BACnet_StructuredViewPath : \/DemoNClass : ) *)

 fbDevice : FB_BACnet_Device;

 fbBV_0 : FB_BACnet_BinaryValue;
  (* ~(BACnet_ObjectIdentifier : 0 : nolink )
  (BACnet_EventEnable : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_ActiveText : EIN : nolink )
  (BACnet_InactiveText : AUS : nolink )
  (BACnet_NotificationClass : 10 : nolink ) *)

 fbAV_0 : FB_BACnet_AnalogValue;
  (* ~(BACnet_ObjectIdentifier : 0 : nolink )
  (BACnet_EventEnable : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_NotificationClass : 10 : nolink ) *)

 fbMV_0 : FB_BACnet_MultiStateValue;
  (* ~(BACnet_ObjectIdentifier : 0 : nolink )
  (BACnet_EventEnable : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_NotificationClass : 10 : nolink )
  (BACnet_NumberOfStates : 5 : nolink )
  (BACnet_StateText : ErrLow;WarnLow;Normal;WarnHigh;ErrHigh : nolink )
  (BACnet_FaultValues : 1;5 : nolink )
  (BACnet_AlarmValues : 2;4 : nolink )
  (BACnet_Description : MV DEMO Objekt. : nolink ) *)

 fbNC_10 : FB_BACnet_NotificationClass;
  (* ~(BACnet_ObjectIdentifier : 10 : nolink )
  (BACnet_Priority : {70;80;90} : nolink )
  (BACnet_AckRequired : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_RecipientList :
      <ArrayOfBACnetDestination>
    <BACnetDestination>
      <recipient>
        <choice>device</choice>
        <BACnetObjectIdentifier>
          <objId>33661729</objId>
        </BACnetObjectIdentifier>
      </recipient>
      <validDays>65025</validDays>
      <fromTime>
        <hour>0</hour>
        <minute>0</minute>
        <second>0</second>
        <hundredths>0</hundredths>
      </fromTime>
      <toTime>
```

```
        <hour>23</hour>
        <minute>59</minute>
        <second>59</second>
        <hundredths>99</hundredths>
      </toTime>
      <processIdentifier>10</processIdentifier>
      <transitions>57349</transitions>
      <issueConfirmedNotifications>false</issueConfirmedNotifications>
    </BACnetDestination>
      </ArrayOfBACnetDestination>
  : nolink ) *)END_VAR
```

The NotificationSink added for receiving the events is configured in the System Manager: *NotificationSink*

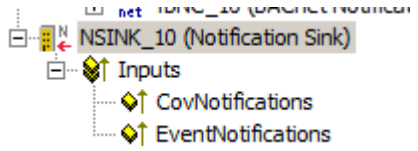


Fig. 2: Figure 2: NotificationSink in the System Manager tree view

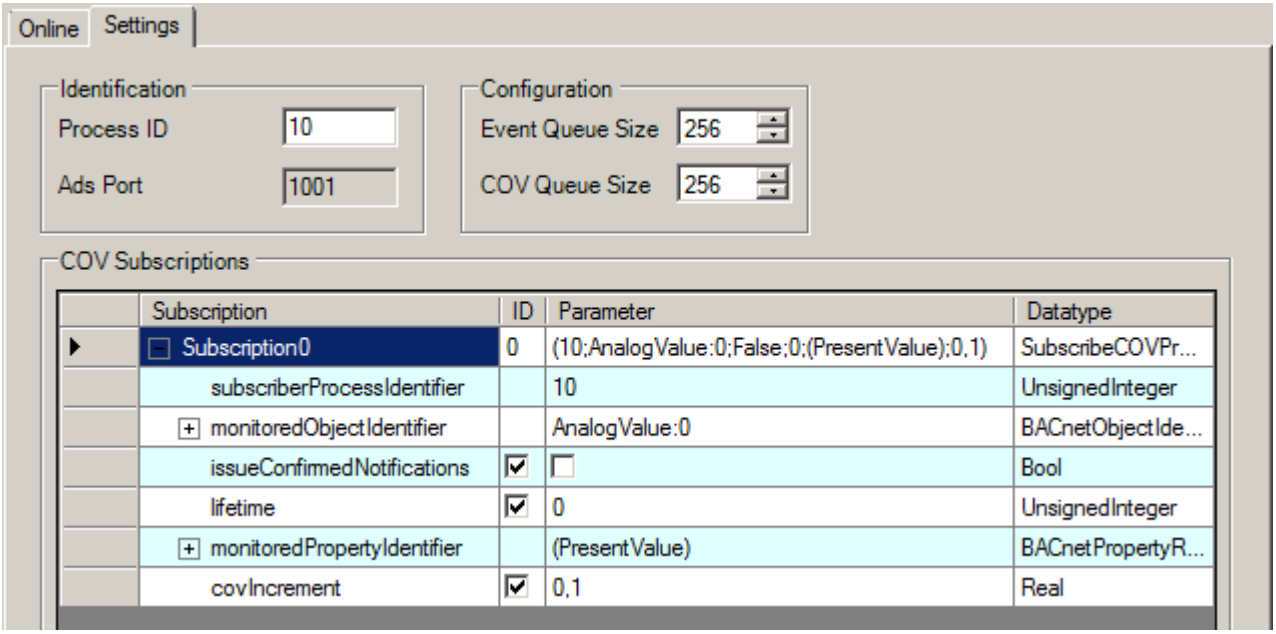


Fig. 3: Figure 3: NotificationSink configuration in the System Manager

Process ID: The process ID is set to 10 and reported to the *NotificationClass* as a receive process (via PLC automapping comment).

COV Subscriptions: As an example the *PresentValue* of the *AnalogValue* object (AV:0) is subscribed to. The parameter *covIncrement* specifies the change of value threshold at which a COV notification is sent.

The following entries are displayed in the Online tab, once the configuration has been loaded and the values of the respective objects have been modified: *NotificationSink PresentValue*

Notifications

Nr	Time	Source	Param
COVNotification_0	12.06.2012 14:20:53	(10;Device:107297;AnalogValue:0;0;{(Pres...	COV...
COVNotification_1	12.06.2012 14:21:19	(10;Device:107297;AnalogValue:0;0;{(Pres...	COV...
COVNotification_2	12.06.2012 14:21:24	(10;Device:107297;AnalogValue:0;0;{(Pres...	COV...
COVNotification_3	12.06.2012 14:21:27	(10;Device:107297;AnalogValue:0;0;{(Pres...	COV...
COVNotification_4	12.06.2012 14:21:30	(10;Device:107297;AnalogValue:0;0;{(Pres...	COV...
EventNotification_0	12.06.2012 14:20:53	(10;Device:107297;MultiStateValue:0;12.0...	Eve...
EventNotification_1	12.06.2012 14:20:53	(10;Device:107297;BinaryValue:0;12.06.20...	Eve...
EventNotification_2	12.06.2012 14:21:11	(10;Device:107297;BinaryValue:0;12.06.20...	Eve...
EventNotification_3	12.06.2012 14:21:13	(10;Device:107297;BinaryValue:0;12.06.20...	Eve...
EventNotification_4	12.06.2012 14:21:57	(10;Device:107297;MultiStateValue:0;12.0...	Eve...
EventNotification_5	12.06.2012 14:22:03	(10;Device:107297;MultiStateValue:0;12.0...	Eve...
EventNotification_6	12.06.2012 14:22:05	(10;Device:107297;MultiStateValue:0;12.0...	Eve...
EventNotification_7	12.06.2012 14:22:08	(10;Device:107297;MultiStateValue:0;12.0...	Eve...
processIdentifier		10	Unsi...
initiatingDeviceId...		Device:107297	BAC...
eventObjectIdent...		MultiStateValue:0	BAC...
timeStamp		12.06.2012 14:22:08	d...
notificationClass		10	Unsi...
priority		80	Unsi...
eventType		change_of_state	BAC...
messageText	☑	MV DEMO Objekt.	Char...
notifyType		alarm	BAC...
ackRequired	☑	☑	Bool
fromState	☑	offnormal	BAC...
toState		fault	BAC...
eventValues	☑	((49156;11;5))	c...

Fig. 4: Figure 4: NotificationSink online data after COV and event notifications have been received.

The text under messageText is formed from the property description of the event-generating object.

3.9 Frequently asked questions

BACnet client does not provide current process data

- During stopping/shutdown of the TwinCAT system the process data of some BACnet objects are reset [▶ 133]
  - How can a restart of remote servers be detected?
  - How can the status of remote servers be monitored?
  - How to increase the maximum number of properties subscriptions, which the server permits per object?
- Writing of data to a remote server in the event of change of value (Write On Change) should be temporarily disabled. [▶ 135]
- How can the sending of alarm events be prevented for BinaryOutput/Input or Value objects? [▶ 136]

- How can NULL (NaN) be written to the priority array of the commandable property PresentValue of an AnalogOutput by the PLC via the process data? [▶ 136]
- Why is the process data PresentValue of a binary* object of type WORD? [▶ 136]
- The K-bus of BACnet controller fails sporadically, without apparent reason. What could be the causes? [▶ 136]
  - How can the K-bus task be triggered independent of the BACnet task?
- Why does the property StatusFlags of an object have the value 0x0004 or the property EventEnable the value 0x0005, even though no status bits of the property are set? [▶ 138]
- A BACnet client whose process data are linked with the PLC is always shown as “Offline” in the System Manager. However, the client is shown in a scan. What could be the problem? [▶ 138]
  - How can the input process data (Read) of a client be changed from COV to Polling?
- The PLC auto-mapping is much slower than usual. What could be the problem? [▶ 139]

I. BACnet client does not provide current process data

A BACnet client does not provide current process data, which were subscribed to via COV, to the PLC (process data connected with the PLC on the client side via mapping; the object itself runs on a remote server). What might cause this?

1. The remote server which the client tries to access was restarted (reboot, TwinCAT restart or control voltage failure). Once the server is running again the client must re-register all COV subscriptions, or the time defined under Resubscription Interval has elapsed (Resubscription Interval, see also section “Process data [▶ 42]” under "Properties as process data").
   - **How can a restart of remote servers be detected?**

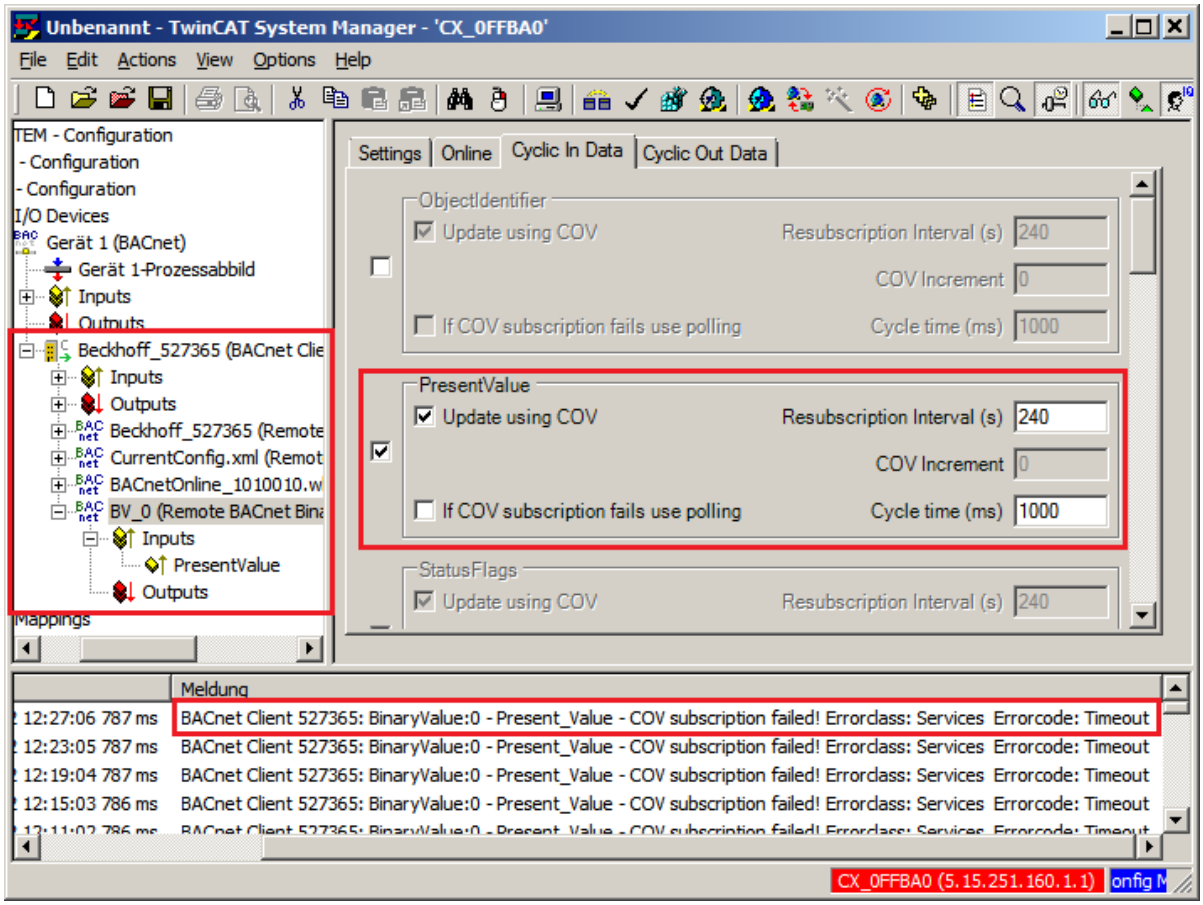
     Under a client the process data SystemStatus can be mapped. Mapping of SystemStatus should be set to Polling with a period that should be as short as possible (between 1..5 s). If the remote server is switched off, timeout of the polling query is triggered (server cannot be reached, client status bit 1 = TRUE → acknowledgement via ClientControl). Once the remote server commences a restart, the SystemStatus assumes state “4” (non-operational). Once the SystemStatus changes from “4” to “0” the remote server is ready for queries, so that COV subscriptions can be re-registered. The registration of the COV subscriptions can be controlled via ClientControl bit 3.

     For further details see section “Process data”, subsection “BACnet client”.
2. The network connection between client and server is interrupted.
   - **How can the status of remote servers be monitored?**

     See item 1. In addition, the Device Status of the local BACnet adapter can be used to monitor the “linked” state of the network adapter (see section “Process data”, subsection “BACnet device”).
3. The remote server does not support COV of Properties (COV-P).

   See also Section IX. [▶ 138]

4. The remote server is supporting COV-P, but the maximum number of subscriptions to the corresponding object exceeded on server side. In the logger window of the System Manager on the client side you will see:

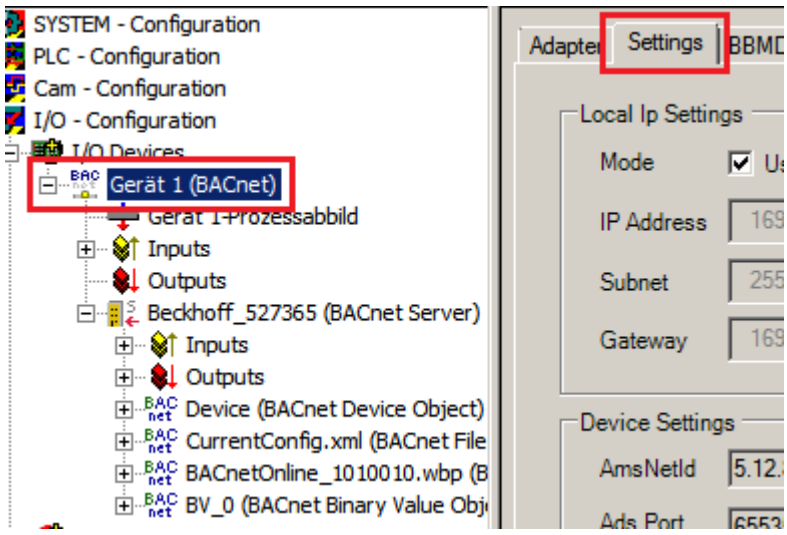


Example of COV-P error

- **How to increase the maximum number of properties subscriptions, which the server permits per object?**

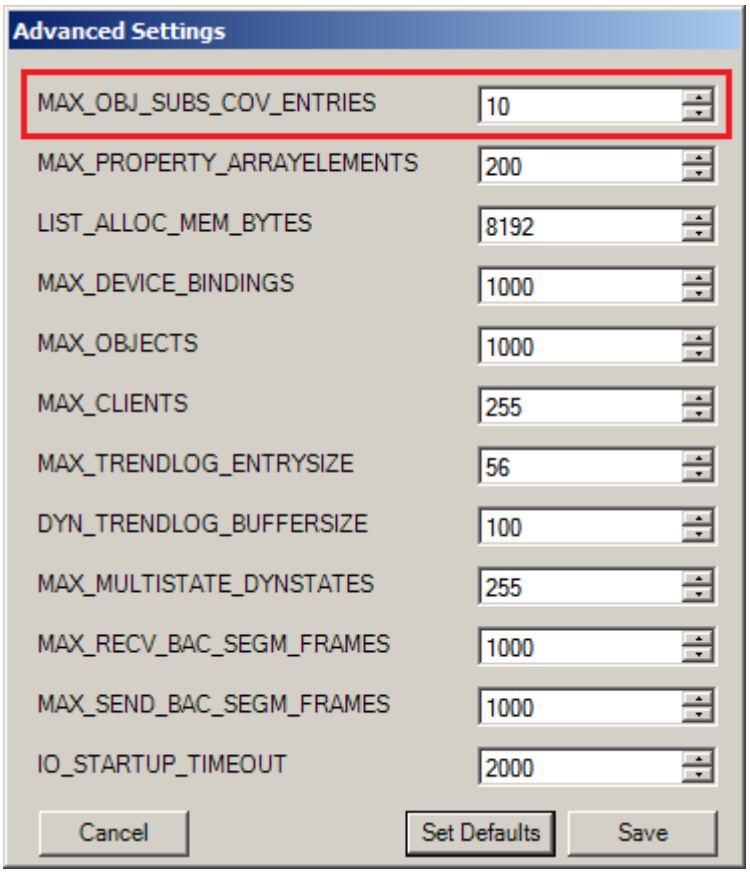
  On the server side the following parameter needs to be adjusted to increase the maximum number of subscriptions per object (see Advanced Settings under Memory-specific parameters [▶ 531]):

  a. Select the BACnet adapter of the appropriate server



b. Press button "Configure" under "Advanced Settings - Data Structures"

c. Increase parameter "MAX_OBJ_SUBS_COV_ENTRIES" accordingly



d. Press button "Save" to store changes / button "Cancel" to abort

e. Apply changes with symbol "Activate configuration" respectively with symbol "Reload I/O devices (F4)" in mode "CONFIG" from the tool bar.

II. During stopping/shutdown of the TwinCAT system the process data of some BACnet objects are reset

During stopping/shutdown of the TwinCAT system the process data of some BACnet objects are reset. This leads to unintended effects since some object states influence the persistent PLC data. How can the validity of the object states in the PLC be monitored?

1. The validity of the object states can be monitored with the aid of the Device Status (bits 8...15) of the local BACnet adapter, by querying the state of the device state machine (mapped in the PLC). If the device state machine reports Complete (8), the local server and its objects and properties were initialized and the communication with remote servers can commence. During shutdown of TwinCAT system the device state machine is reset (< 8). If the resetting (< 8) is detected in the PLC, the object states (mapped properties) should no longer be evaluated.
2. In general, the validity of client objects (of remote servers) cannot be verified since communication timeouts etc. prevent timely monitoring. Failure of BACnet services is indicated by the client status (bit 0 and 1, see section "Process data [▶ 42]", subsection "BACnet client"). Failed queries result in the respective bits being set in the client status, which then must be acknowledged. The local BACnet adapter can be monitored with the aid of the device status (see item 1.).

III. Writing of data to a remote server in the event of change of value (Write On Change) should be temporarily disabled.

Writing of data to a remote server in the event of change of value (Write On Change) should be temporarily disabled (e.g. for re-triggering of the PresentValue of a BinaryOutput object, without state INACTIVE being set). How can this be realised?

1. In the process data ClientControl writing of Write On Change can be disabled with the aid of bit 2 (see section "Process data", subsection "BACnet client [▶ 44]").
2. Re-triggering of writing to a remote object can alternatively be accomplished with an ADS command (see section "ADS interface [▶ 513]"). The PLC library "TcBACnet.lib" offers the convenient function block "FB_BACnetWriteProp" for this purpose.

IV. How can the sending of alarm events be prevented for BinaryOutput/Input or Value objects?

1. Sending of events can be prevented by deselecting the bits in the Settings or Online tabs, or by writing the property EventEnable no. 35 (value = 0x0005). However, this only suppresses sending of event notifications. The status bit in_alarm of the property StatusFlags no. 111 continues to be set.
2. To prevent generating any events for the respective binary object, support for intrinsic reporting (O4) can be disabled (see section "BACnet objects and properties [▶ 21]", subsection "Configuration"). This prevents setting of status bit in_alarm of property StatusFlags no. 111.

V. How can NULL (NaN) be written to the priority array of the commandable property PresentValue of an AnalogOutput by the PLC via the process data?

1. The PresentValue must be activated as a process data with the required priority.
2. The process data must be linked with the PLC (data type REAL)
3. In PLC program the library function "F_BACnet_NAN() [▶ 486]" can be used to write the state NULL (NaN) from the library "TcBACnet.lib" to the process data → This cancels the respective priority of the priority array.

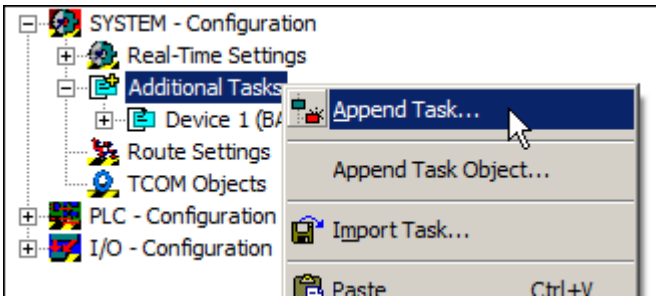
VI. Why is the process data PresentValue of a binary* object of type WORD?

Within BACnet the PresentValue of binary* object is stored as enumeration of type BACnetBinaryPV [▶ 518] (active, inactive). Enumerations are generally mapped with type WORD. The manufacturer-specific property PresentValueBool was introduced to simplify handling of binary* objects. It can be mapped with data type BOOL via the PLC mapped (see section "Process data [▶ 42]").

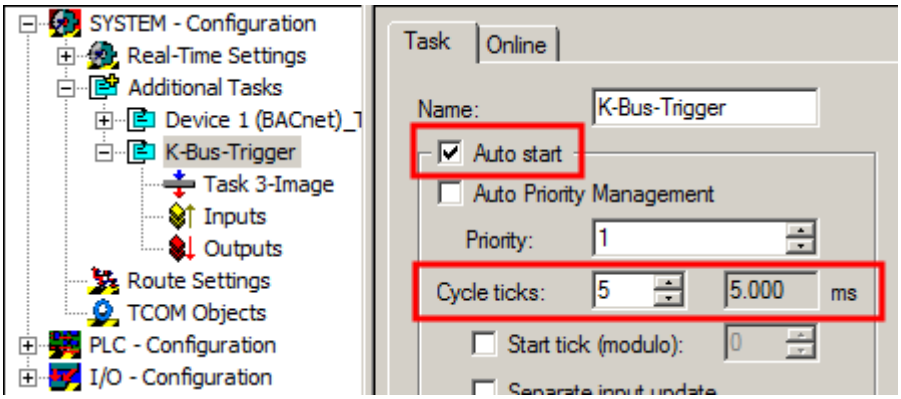
VII. The K-bus of BACnet controller fails sporadically, without apparent reason. What could be the causes?

1. The K-bus power supply was exceeded. The K-bus is switched off, if e.g. many relay output terminals are activated simultaneously during operation, resulting in the maximum permitted K-Bus current to be exceeded. Power requirements of the terminals should be checked (see power consumption in the data sheet for the terminal) and an appropriate K-Bus refresh (see description of the KL9400) must be provided.
2. If the K-bus task is triggered by the BACnet task (mapping only between BACnet and K-bus, no PLC etc.→ asynchronous mapping), under heavy load of the BACnet task the K-bus task may not be triggered often enough. The reason is task time exceedance of the BACnet task (frequently exceeds → see Exceed counter for the respective task) in the event of high BACnet communication traffic.
    - **How can the K-bus task be triggered independent of the BACnet task?**

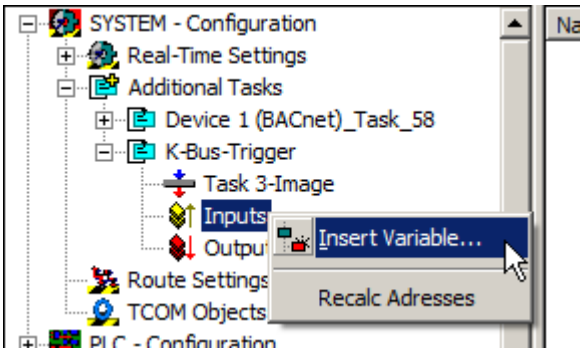
a. Create a new task in the TwinCAT System Manager



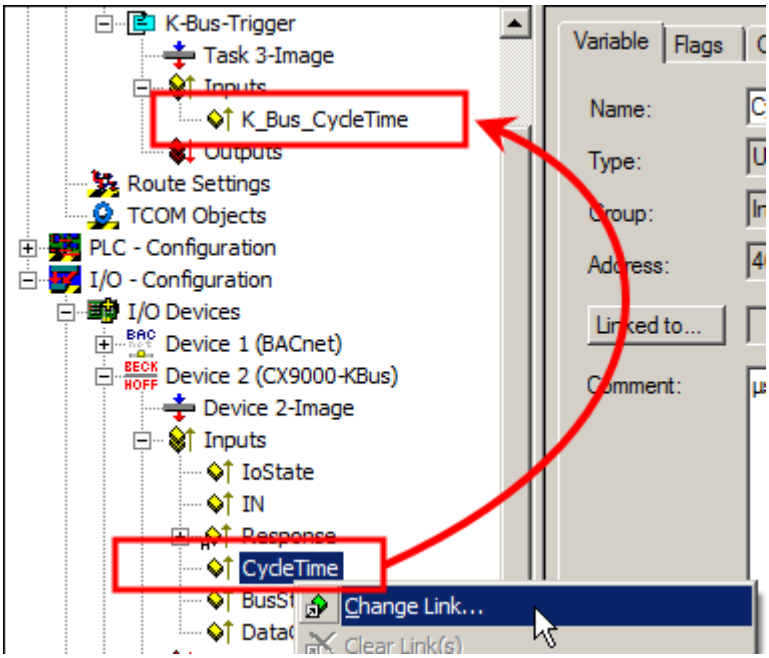
b. Configure the task; activate Auto start and set the required cycle time (Cycle ticks)



c. Add a variable for the mapping (e.g. CycleTime of type UINT)



d. Link the K-bus variable “CycleTime” and task variable “K_Bus_CycleTime”



e. Activate the modified configuration with the icon “Activate configuration” in the toolbar

VIII. Why does the property StatusFlags of an object have the value 0x0004 or the property EventEnable the value 0x0005, even though no status bits of the property are set?

Properties with bit flags (subtype BitFieldBits) are internally transferred as so-called bit string. A bit string always consists off a high byte and a low byte. The low byte indicates how many bits of high byte are not used (from the left). i.e. StatusFlags = 0x0004 means the 4 higher bits of the high byte are not used (i.e. only bits 8, 9, 10 and 11 of the WORD data type are used). For further details see section “Process data / bit string [▶ 42]”.

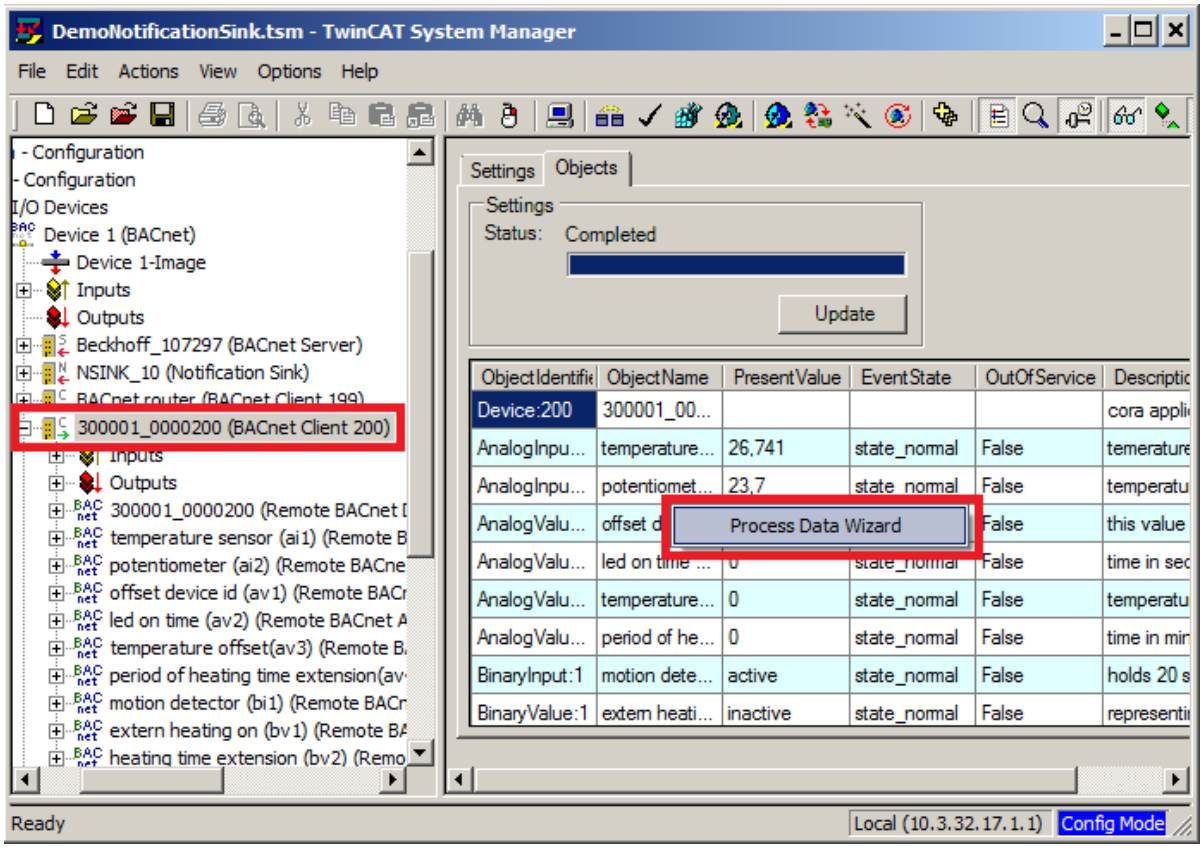
IX. A BACnet client whose process data are linked with the PLC is always shown as “Offline” in the System Manager. However, the client is shown in a scan. What could be the problem?

If process data are used that were linked by PLC automapping, the properties are usually obtained via COV subscription. This can lead to problems, if the client does not support COV (logger messages such as: “Error TCOM Server (10) [...] BACnet Client [...] - COV subscription failed! Reject Reason: **unrecognized service**”). Since the COV subscription fails, new requests keep being sent to the remote server. This leads to timeouts and eventually to the client be shown as “Offline” in the System Manager.

Solution: The input process data of the client properties should be switched to “Polling” mode in the System Manager.

How can the input process data (Read) of a client be changed from COV to Polling?

1. Select the respective client
2. Open the “Objects” tab
3. Open the Property wizard (see also BACnet wizards)



4. Select the client under “Select Objects” and...
   - a. Activate option “Optimize Process Data Access (Tick Modulo)”
   - b. Activate option “Change Process Data In Configuration (Read)”
   - c. Set mode “Process Data Read Mode” to “Polling”

- d. Set parameter "Cycle Time (ms)" to 10000 = 10 seconds read cycle, for example
  Notice This parameter must be optimized based on the application requirements and may have to be determined individually for the respective property

5. Select all entries under “Select Properties” (right-click, “Select All”)
6. Click Apply to confirm the changes.

X. The PLC auto-mapping is much slower than usual. What could be the problem?

Cause: The PLC auto-mapping should not be performed when the system manager logged on the target system. When the System Manager is connected to the target system, there will be delays when linking process data. The PLC auto-mapping creates potentially many links of process data, and this leads to a much longer PLC auto-mapping cycle.

Solution: Before the PLC auto-mapping is performed the System Manager should be connected to local runtime system. After the PLC auto-mapping was performed the System Manager can be reconnected to the target system.

Also see about this

- Frequently asked questions [▶ 136]
- Help functions and wizards [▶ 31]

4 PLC library: TcBACnetRev12.Lib

The functionality of the PLC library "TcBACnetRev12.lib" is described below. The library offers convenient access to objects of a BACnet configuration from a PLC program.

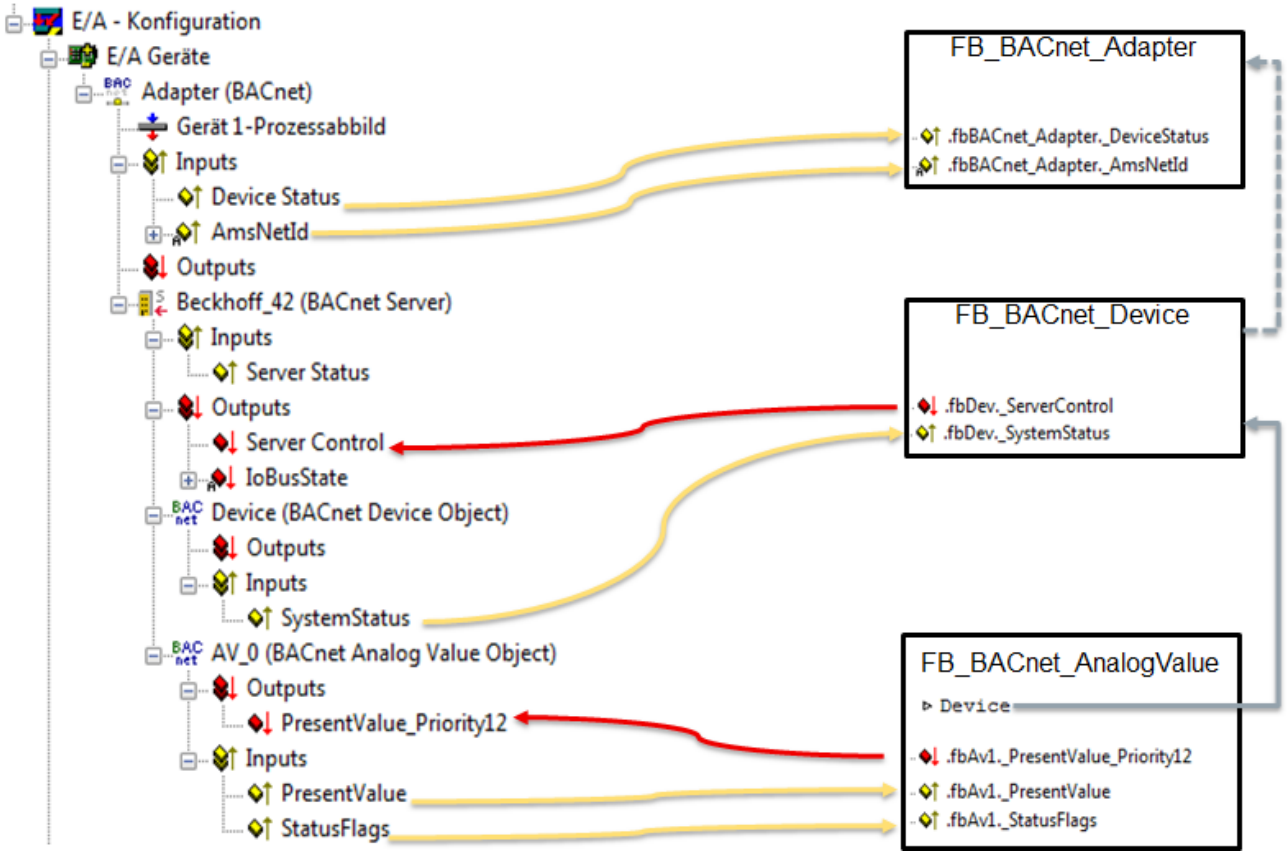
Overview

The PLC library "TcBACnetRev12.lib" is a collection of function blocks for programming a BACnet controller. For each BACnet object function blocks are available that are read and written via the selected *Properties*. In general a distinction is made between function blocks for server and client objects. Function blocks for client objects have the prefix "Remote", since they enable access to remote BACnet objects on other devices. The BACnet object function blocks ascertain whether process data are valid and the BACnet controller is ready for operation (bReady).

The function block FB_BACnet_Adapter [▶ 153] represents a BACnet device and therefore the access point to the BACnet network via a network card. This block enables the existence of a link to be verified, for example. To make the PLC program more transparent, the BACnet adapter is created as a global variable within the "TcBACnetRev12.lib", since in the majority of the projects a single BACnet adapter is used.

Depending on the client and server functionality, the function blocks FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241] can be used to link state and control information for the BACnet client and server. The operational readiness of the BACnet objects is determined via the client and server status and the property *System_Status* for the respective device or remote device object. Each BACnet object function block must therefore transfer a reference to the instance of the corresponding FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241], in order to enable the status evaluation (**bReady**).

The following figure shows an overview of the function blocks of "TcBACnetRev12.lib" and the corresponding links with the BACnet modules. The linking between a PLC program based on "TcBACnetRev12.lib" and the BACnet modules of a configuration can be automated via the function PLC automapping [▶ 62].



Compatibility and new features

From TwinCAT 2.11 Build 2042, the PLC library "TcBACnetRev12.lib" is included in the basic TwinCAT installation. The library is largely compatible with the previous version "TcBACnet.lib". A brief overview of new features is provided below:

- BACnet function block names are compatible (except Accumulator)
- Improved alignment of process data
- FBs for new (Revision 12) objects, consistently _Ex for all objects
- FB_BACnet_Device
  - Data type, name nAmsPort changed (no function with Rev. 12)
  - Automatic reading of ADS information
  - Triggering of persistent data writing
- Loop_Drv: Internal control active automatically (via BACnet stack)
- Loop_Ex: Auto-tuning (PID determination), ramp limitation (AccLimit) outputs
- New ADS functions:
  - String decoding according to Win1252 (Description, ObjectName,EventMessageTexts)
  - ExceptionSchedule (bool), WeeklySchedule (bool), LogBuffer (Real), RecipientList (Read,Write), ObjectList, RecpientList (Notification Class)
  - Access to NotificationSink functions (Acknowledge, Read message list, Delete)
  - Support for time synchronization
- Remote_blocks
  - ERR_OPERATIONAL error if client not present
  - bTriggerWOC
  - For write access, _EX or _WR versions must be used.

Older projects that were created based on the "TcBACnet.Lib" PLC library can be converted for use with the new library based to the following scheme:

1. Delete "TcBACnet.Lib" from the PLC project
2. Add "TcBACnetRev12.Lib"
3. Recompile
4. In the System Manager:
   - Re-load the PLC project
   - undo the old mapping, if necessary
   - create a BACnet device in the new Revision, if necessary.
   - Run PLC automapping

Persistent data cannot be preserved during the changeover to the new "TcBACnetRev12.lib".

As a rule, PLC projects based on the old "TcBACnet.Lib" should also work under "BACnet Revision 12", without the need for conversion.

Overview

An overview of the PLC library components is provided below. Detailed information can be called up via corresponding links.

BACnet adapter and Notification Sink

Blocks	Description
FB_BACnet_Adapter [▶ 153]	Function block for linking the PLC program with a local BACnet adapter (network card).

Blocks	Description
FB_BACnet_NotificationSink [▶ 258]	Function block for realizing an ADS connection with a BACnet NotificationSink.

Local BACnet objects (server)

Standard objects

The following function blocks establish the connection between the BACnet object in the TwinCAT System Manager and its use in the PLC program. The blocks contain all comments that are required for the PLC automapping.

The standard function blocks are available in two versions:

- **Minimum process data** - only the essential properties are linked, including *Present_Value* and *Status_Flags*.
- **Extended process data** - *Object_Identifier*, *Event_Flags* and *Reliability* are also linked. Function blocks with extended process data are identified with the extension *_EX*.

The documentation refers to the version with extended process data. Advanced process data are identified as such in the description.

Blocks	Symbol	BACnet object	Description
FB_BACnet_Accumulator_EX [▶ 156]		Accumulator	A BACnet Accumulator object represents a measured value determined through pulse count.
FB_BACnet_AnalogInput_EX [▶ 166]		Analog Input	A BACnet Analog Input object represents an analog input value.
FB_BACnet_AnalogOutput_EX [▶ 170]		Analog Output	A BACnet Analog Output object represents an analog output value.
FB_BACnet_AnalogValue_EX [▶ 176]		Analog Value	Represents an analog state value within a program.
FB_BACnet_Averaging_EX [▶ 179]		Averaging	Enables calculation of statistical data within a control system.
FB_BACnet_BinaryInput_EX [▶ 180]		Binary Input	A Binary Input object represents a binary input value.
FB_BACnet_BinaryOutput_EX [▶ 185]		Binary Output	A Binary Output object represents a binary output value.
FB_BACnet_BinaryValue_EX [▶ 191]		Binary Value	Represents a binary state value within a program.
FB_BACnet_Calendar_EX [▶ 194]		Calendar	Enables separate definition of exception days for schedules (schedule objects).
FB_BACnet_Command_EX [▶ 195]		Command	Enables control of complex sequences over chronologically staggered write commands for BACnet object properties.

Blocks	Symbol	BACnet object	Description
FB_BACnet_Device [▶ 196]		Device	Forms the logical entry point of a BACnet device. Includes the list of all BACnet objects for this device.
FB_BACnet_EventEnrollment_EX [▶ 198]		Event Enrollment	Enables configuration of rule-based event messages. The notification system integrated in many objects enables definition of comprehensive rules for triggering event messages. Example: additional or multiple limit value pairs for a *PresentValue*.
FB_BACnet_File_EX [▶ 199]		File	Represents properties of a file object.
FB_BACnet_Group_EX [▶ 200]		Group	Group objects enable consolidation of multiple *Properties* in a single data point.
FB_BACnet_Loop_EX [▶ 201]		Loop	Represents the properties of a PID controller.
FB_BACnet_MultiStateInput_EX [▶ 205]		Multi State Input	Represents an integer/ multi-stage input value.
FB_BACnet_MultiStateOutput_EX [▶ 209]		Multi State Output	Represents an integer/ multi-stage output value.
FB_BACnet_MultiStateValue_EX [▶ 215]		Multi State Value	Represents an integer/ multi-stage state value.
FB_BACnet_NotificationClass_EX [▶ 218]		Notification Class	The Notification Class object is used to configure the distribution of event messages (EventNotifications).
FB_BACnet_Program_EX [▶ 219]		Program	A BACnet Program object enables changing the states of a PLC program.
FB_BACnet_PulseConverter_EX [▶ 222]		Pulse Converter	A Pulse Converter object represents a measured value determined through pulse count.
FB_BACnet_Schedule_EX [▶ 225]		Schedule	Represents a schedule which is used to write values of other BACnet objects on the basis of time-based switching entries.

Blocks	Symbol	BACnet object	Description
FB_BACnet_TrendLog_EX [▶ 226]		Trend Log	Represents recorded historical data, which are recorded cyclically at fixed intervals or event-based.

Local BACnet objects (ADS)

Write access to the property *PresentValue* occurs priority-based via ADS. This function block can be used in cases where no priorities for write access have been specified in advance of the program runtime, or to manipulate several priority levels in the program at the same time.

Function blocks	Description
FB_BACnet_AnalogOutput_ADS [▶ 171]	A BACnet Analog Output object represents an analog output value.
FB_BACnet_AnalogValue_ADS [▶ 177]	Represents an analog state value within a program.
FB_BACnet_BinaryOutput_ADS [▶ 186]	A Binary Output object represents a binary output value.
FB_BACnet_BinaryValue_ADS [▶ 192]	Represents a binary state value within a program.
FB_BACnet_MultiStateOutput_ADS [▶ 210]	Represents an integer output value.
FB_BACnet_MultiStateValue_ADS [▶ 216]	Represents an integer output value.

Table 1: Local BACnet objects (drivers)

Function blocks	Description
FB_BACnet_Loop_DRV_EX [▶ 203]	This block is linked to a TwinCAT BACnet Loop object with enabled internal control. The control algorithm is implemented directly in the BACnet stack, not in the PLC. All other loop-FBs implement their control algorithms within the PLC. The advantage of this function block is a more efficient execution time, since fewer BACnet properties have to be linked via process data.

Local BACnet objects (raw versions)

RAW FBs enable specification of physical measured and control values from the PLC. This is in contrast to other FBs for physical measured and control values, which are linked directly to a hardware module. The corresponding RawIo* process record of the BACnet object is linked directly to a PLC variable via PLC automapping.

Function blocks	Description
FB_BACnet_Accumulator_RAW [▶ 159]	A BACnet Accumulator object represents a measured value determined through pulse count. In this case this measured value is formed in the PLC.
FB_BACnet_AnalogInput_RAW [▶ 167]	A BACnet Analog Input object represents an analog input value. This function block can be used for postprocessing of analog input values in the PLC, or for generating read-only analog values directly in the PLC.
FB_BACnet_AnalogOutput_RAW [▶ 173]	A BACnet Analog Output object represents an analog output value. This function block can be used in cases where an analog output signal is to be pre-processed prior the output, for example.
FB_BACnet_BinaryInput_RAW [▶ 182]	A Binary Input object represents a binary input value. This function block can used in cases where postprocessing of binary input values in the PLC is desirable (e.g. for debouncing)
FB_BACnet_BinaryOutput_RAW [▶ 188]	A Binary Output object represents a binary output value. This function block can be used in cases where a binary output signal is to be pre-processed prior the output, for example.

Function blocks	Description
FB_BACnet_MultiStateInput_RAW [▶ 206]	A multi-state input object represents an integer/multi-stage input value.
FB_BACnet_MultiStateOutput_RAW [▶ 212]	A multi-state output object represents an integer/multi-stage output value.
FB_BACnet_PulseConverter_RAW [▶ 224]	A Pulse Converter object represents a measured value determined through pulse count.

Table 2: Local BACnet objects (Revision 6 specific)

Function blocks	Description
FB_BACnet_Accumulator_R6 [▶ 162]	Function block that is compatible with Revision 6. It should only be used with TwinCAT BACnet Revision 6.
FB_BACnet_Accumulator_RAW_R6 [▶ 163]	Function block that is compatible with Revision 6. It should only be used with TwinCAT BACnet Revision 6.

Client BACnet objects (remote)

The function blocks for remote objects offer access to BACnet objects of other devices. A distinction is made between three versions of remote FBs. The basic FBs without extension offer read access to the properties *PresentValue* and *StatusFlags*. The "_EX" versions support a wider range of properties. The "_WR" versions can be used for additional write access with minimum read properties. In general, function blocks with few properties are preferable, to minimize the network load.

The function. exp export (in the BACnet device) can be used for efficient integration of BACnet devices. It facilitates generation of PLC variable declarations for the remote FBs of scanned client configurations or configurations that were read via an EDE file.

Function blocks	Description
FB_BACnet_RemoteAccumulator [▶ 227]	A BACnet Accumulator object represents a measured value determined through pulse count.
FB_BACnet_RemoteAnalogInput [▶ 229]	A BACnet Analog Input object represents an analog input value.
FB_BACnet_RemoteAnalogOutput [▶ 230]	A BACnet Analog Output object represents an analog output value.
FB_BACnet_RemoteAnalogValue [▶ 232]	Represents an analog state value within a program.
FB_BACnet_RemoteAveraging [▶ 233]	Enables calculation of statistical data within a control system.
FB_BACnet_RemoteBinaryInput [▶ 234]	A Binary Input object represents a binary input value.
FB_BACnet_RemoteBinaryOutput [▶ 235]	A Binary Output object represents a binary output value.
FB_BACnet_RemoteBinaryValue [▶ 237]	Represents a binary state value within a program.
FB_BACnet_RemoteCalendar [▶ 238]	Enables separate definition of exception days for schedules (schedule objects).
FB_BACnet_RemoteCommand [▶ 239]	Enables control of complex sequences over chronologically staggered write commands for BACnet object properties.
FB_BACnet_RemoteDevice [▶ 241]	Forms the logical entry point of a BACnet device. Includes the list of all BACnet objects for this device.
FB_BACnet_RemoteEventEnrollment [▶ 243]	Enables configuration of rule-based event messages. The notification system integrated in many objects enables definition of comprehensive rules for triggering event messages. Example: additional or multiple limit value pairs for a *PresentValue*.
FB_BACnet_RemoteFile [▶ 244]	Represents properties of a file object.
FB_BACnet_RemoteGroup [▶ 244]	Group objects enable consolidation of multiple *Properties* in a single data point.

Function blocks	Description
FB_BACnet_RemoteLoop [▶ 245]	Represents the properties of a PID controller.
FB_BACnet_RemoteMultiStateInput [▶ 247]	Represents an integer/multi-stage input value.
FB_BACnet_RemoteMultiStateOutput [▶ 248]	Represents an integer/multi-stage output value.
FB_BACnet_RemoteMultiStateValue [▶ 250]	Represents an integer/multi-stage state value.
FB_BACnet_RemoteNotificationClass [▶ 251]	The Notification Class object is used to configure the distribution of event messages (EventNotifications).
FB_BACnet_RemoteProgram [▶ 252]	A BACnet Program object enables changing the states of a PLC program.
FB_BACnet_RemotePulseConverter [▶ 255]	A Pulse Converter object represents a measured value determined through pulse count.
FB_BACnet_RemoteSchedule [▶ 256]	Represents a schedule which is used to write values of other BACnet objects on the basis of time-based switching entries.
FB_BACnet_RemoteTrendLog [▶ 257]	Represents recorded historical data, which are recorded cyclically at fixed intervals or event-based.

ADS function blocks for generic access to all online properties (raw data access)

Function blocks for access to BACnet properties via ADS. All BACnet properties of server and client objects can be read or written via ADS.

Function blocks	Description
FB_BACnet_ReadProp [▶ 273]	Read access to properties
FB_BACnet_WriteProp [▶ 276]	Write access to properties

ADS function blocks for access to specific properties

Function blocks for specific access to BACnet properties with data type conversion via ADS. All BACnet properties of server and client objects can be read or written via ADS. Based on the function blocks FB_BACnet_ReadProp and FB_BACnet_WriteProp, the following function blocks are used to convert data read via ADS into PLC data types, or to code PLC data as BACnet data during write access. The size limitation of approx. 8 kbytes for the following ADS access operations is related to the global ADS data buffer (see ST_BACnet_GlobalAdsBuffer [▶ 360]).

Function blocks	Description	Access
FB_BACnet_ObjectNameProperty [▶ 292]	ADS access to the property *Object_Name* of type CharacterString, including decoding of UTF-8, UCS-2 and UCS-4	Read
FB_BACnet_DescriptionProperty [▶ 280]	ADS access to the property *Description* of type CharacterString, including decoding of UTF-8, UCS-2 and UCS-4	Read
FB_BACnet_EventMessageTextsProperty [▶ 281]	ADS access to the property *Event_Message_Texts* of type CharacterStringExtList, including decoding of UTF-8, UCS-2 and UCS-4	Read
FB_BACnet_ObjectListProperty [▶ 290]	ADS access to the property *Object_List* of type BACnetObjectIdentifier[]	Read
FB_BACnet_ExceptionScheduleProperty [▶ 283]	ADS access to the property *Exception_Schedule* of type BACnetSpecialEventList; only for entries with data type **Bool**	Read, write
FB_BACnet_WeeklyScheduleProperty [▶ 297]	ADS access to the property *Weekly_Schedule* of type BACnetDailyScheduleList; only for entries with data type **Bool**	Read, write

Function blocks	Description	Access
FB_BACnet_LogBufferProperty [▶ 287]	ADS access to the property *Log_Buffer* of type BACnetLogRecordList; only for entries with data type **Real**	Read
FB_BACnet_RecipientListProperty [▶ 294]	ADS access to the property *Recipient_List* of type BACnetDestination[] (e.g. NotificationClass object)	Read, write

ADS blocks for access to service and diagnostic data

Function blocks	Description
FB_BACnet_GetDiagInfo [▶ 259]	ADS access to the diagnostic data of a BACnet adapter
FB_BACnet_NSinkReadEvent [▶ 264]	ADS access to the BACnet Notification Sink: reading a BACnet event
FB_BACnet_NSinkAcknEvent [▶ 261]	ADS access to the BACnet Notification Sink: service to acknowledge a BACnet event
FB_BACnet_NSinkRemoveEvent [▶ 268]	ADS access to the BACnet Notification Sink: deleting a BACnet event (replaces FB_BACnet_NotificationSinkDelEntry [▶ 489])
FB_BACnet_TimeSync [▶ 270]	ADS access to the BACnet adapter: service for time synchronization in the BACnet network (broadcast) or locally. **The function block provides the current system time cyclically as an output and should be used in each BACnet project as the time source in the PLC program.**

BACnet auxiliary blocks for date and time

Function blocks	Description
F_BACnet_CheckDay [▶ 307]	Function for checking the validity of a numerical date value (BYTE) for the day of the month.
F_BACnet_CheckDayOfWeek [▶ 308]	Function for checking the validity of a numerical date value (BYTE) for the day of the week.
F_BACnet_CheckHour [▶ 308]	Function for checking the validity of a numerical time value (BYTE) for the hour indication.
F_BACnet_CheckHundredths [▶ 308]	Function for checking the validity of a numerical time value (BYTE) for the hundredths of a second indication.
F_BACnet_CheckMinute [▶ 309]	Function for checking the validity of a numerical time value (BYTE) for the minute indication.
F_BACnet_CheckMonth [▶ 309]	Function for checking the validity of a numerical date value (BYTE) for the month.
F_BACnet_CheckSecond [▶ 309]	Function for checking the validity of a numerical time value (byte) for the second.
F_BACnet_CheckWeekOfMonth [▶ 310]	Function for checking the validity of a numerical date value (BYTE) for the week of the month.
F_BACnet_CheckYear [▶ 310]	Function for checking the validity a numerical date value (BYTE) for the year.
F_BACnet_DateHasPlaceholder [▶ 310]	Function for checking for placeholders (255 → undefined) in a date.
F_BACnet_DateMerge [▶ 311]	Function for merging of two time stamps.
F_BACnet_DateTime_TO_TimeStruct [▶ 306]	Function for converting a BACnet time stamp to data type *TIMESTRUCT*..
F_BACnet_DateTimeString [▶ 307]	Function for string representation of a BACnet time stamp.
F_BACnet_DateUnspecified [▶ 311]	Function for checking for placeholders (*255* → undefined) in a date.
F_BACnet_DaysInMonth [▶ 311]	Function for calculating the number of days in a given month of a year.
F_BACnet_Get100msDate [▶ 312]	Function for calculating the timestamp in 100 ms steps since 1900.

Function blocks	Description
F_BACnet_Get100msTime [▶ 312]	Function for calculating the time stamp in 100 ms steps since 00:00:00.0 hrs.
F_BACnet_TimeHasPlaceholder [▶ 312]	Function for checking for placeholders (*255* → undefined) in a time specification.
F_BACnet_TimeMerge [▶ 312]	Function for merging of two time stamps.
F_BACnet_TimeString [▶ 307]	Function for string representation of a BACnet time stamp.
F_BACnet_TimeStruct_TO_DateTime [▶ 306]	Function for converting a time stamp of data type *TIMESTRUCT* to a BACnet time stamp.
F_BACnet_TimeUnspecified [▶ 313]	Function for checking for placeholders (*255* → undefined) in a time specification.

BACnet auxiliary blocks for bit conversion (bit strings)

Function blocks	Description
F_BACnet_EventTransitionFlags [▶ 314]	Function to decode the process data of the property *Acked_Transitions* of a BACnet object.
F_BACnet_GetEventTransFlagsData [▶ 314]	Function for encoding the value of the property *Event_Enable* of a BACnet object.
F_BACnet_GetLimitEnFlagsData [▶ 315]	Function for encoding the value of the property *Limit_Enable* of a BACnet object.
F_BACnet_GetStatusFlagsData [▶ 313]	Function for encoding the value of the property *Status_Flags* of a BACnet object.
F_BACnet_LimitEnableFlags [▶ 315]	Function to decode the process data of the property *Limit_Enable* of a BACnet object.
F_BACnet_StatusFlags [▶ 313]	Function to decode the process data of the property *Status_Flags* of a BACnet object.

BACnet auxiliary blocks for multi-state objects

Function blocks	Description
F_BACnet_MultiStatePV [▶ 316]	Function for implementing a UDINT value of the PLC in the process data value of a BACnet MultiState* object property *Present_Value*.

BACnet auxiliary blocks for REAL values

Function blocks	Description
F_BACnet_RealPV [▶ 316] (obsolete: F_BACnet_AnalogPV [▶ 322])	Function for implementing a REAL value of the PLC in the process data value of a BACnet analog* object property *Present_Value*.
F_BACnet_IsFinite [▶ 318]	Function for checking a REAL value for finiteness.
F_BACnet_RealEQ [▶ 318]	Function for comparing two floating-point values considering the value range.
F_BACnet_RealGE [▶ 319]	Function for comparing two floating-point values considering the value range.
F_BACnet_RealGT [▶ 319]	Function for comparing two floating-point values considering the value range.
F_BACnet_RealLE [▶ 319]	Function for comparing two floating-point values considering the value range.
F_BACnet_RealLT [▶ 320]	Function for comparing two floating-point values considering the value range.
F_BACnet_RealNull [▶ 317] (obsolete: F_BACnet_NAN [▶ 486])	Function returns the floating-point value "**N**ot **a**Number", which, as value of the property *Present_Value* of an analog* object, corresponds to the value *Null* (no value).

Function blocks	Description
F_BACnet_RealNothing [▶ 317]	Function returns the floating-point value "**N**ot **aN**umber", which, as value of the property *Present_Value* of an analog* object, corresponds to the value *Nothing* (do not evaluate). Process data with the coded value *Nothing* are not processed by the BACnet stack from Revision 12.
F_BACnet_RealToStr [▶ 317]	Function for converting a floating-point number to a string, considering the value range.

BACnet auxiliary blocks for string processing

Function blocks	Description
F_BACnet_GetObjectIdString [▶ 320]	Function for outputting the object ID of a BACnet object as a short string.
FB_BACnet_StringExtDecode [▶ 321]	Function block for decoding BACnet strings.
FB_BACnet_StringExtEncode [▶ 321]	Function block for coding BACnet strings.

Signal conversion

Function blocks	Description
FB_BACnet_PWM [▶ 327]	Function block for generating a pulsating output signal, with defined switch-on and switch-off duration in percent of the cycle duration (**p**ulse **w**idth **m**odulation).

Auxiliary functions for BACnet BinaryPV data types

Function blocks	Description
F_BinPV_AND [▶ 324]	Function for logical linking of BACnet BinaryPV values.
F_BinPV_NOT [▶ 325]	Function for inverting a BACnet BinaryPV value.
F_BinPV_OR [▶ 325]	Function for logical linking of BACnet BinaryPV values.
F_BinPV_XOR [▶ 326]	Function for logical linking of BACnet BinaryPV values.
F_BinPV_To_Bool [▶ 323]	Function for converting a BACnet BinaryPV value to data type BOOL.
F_Bool_To_BinPV [▶ 323]	Function for implementing a value with data type BOOL of the PLC in the process data value of a BACnet binary* object / property *Present_Value*.

Other BACnet auxiliary functions

Function blocks	Description
F_BACnet_GetObjectListIndex [▶ 302]	Function for determining the position of a BACnet object ID in a list of IDs.
F_BACnet_GetObjId [▶ 301]	Function for coding the object type and the object instance in the BACnet *Object_Identifier*.
F_BACnet_GetObjInstance [▶ 301]	Function for decoding the BACnet *Object_Identifier* in the object instance (object number).
F_BACnet_GetObjType [▶ 301]	Function for decoding the BACnet *Object_Identifier* in the object type.
FB_BACnet_AccLimit [▶ 302]	Function block for limiting signal change per time (maximum acceleration).
FB_BACnet_AvgValue [▶ 303]	Function block for averaging an input variable **X** over **n** values.
FB_BACnet_PidControl [▶ 303]	PID controller block in parallel configuration or Ideal form.
FB_BACnet_PT1 [▶ 306]	Function block for emulating a 1st order deceleration.

BACnet constants

Constants	Description
BACnet_Globals [▶ 328]	Global constants for default values, error messages, data range limits etc.

BACnet diagnostic data

Data Types	Description
ST_BACnet_DiagEthStatistics [▶ 357]	Describes diagnostic data of the Ethernet adapter such as sent, received or faulty messages.
ST_BACnet_DiagnosisTiming [▶ 358]	Contains information on execution times within the BACnet stack.
ST_BACnet_FrameStatistics [▶ 360]	Contains information on sent and received BACnet network packets.
ST_BACnet_Info [▶ 361]	Information on memory usage.
ST_BACnet_ServerStatistics [▶ 366]	Contains information on persistent data and change statistics for the properties.
ST_BACnet_TcIoEthStatistic [▶ 367]	Describes diagnostic data of the Ethernet adapter.
ST_BACnet_TcIoEthTxRxErrorCount [▶ 368]	Describes diagnostic data of the Ethernet adapter.
ST_BACnet_UnConfirmedServiceDiag [▶ 369]	More detailed data for BACnet services

BACnet data types

Data Types	Description
ST_BACnet_AdsConnection [▶ 355]	Structure with connection information for an ADS server of the BACnet driver
ST_BACnet_CharacterStringExt [▶ 356]	BACnet string, including the encoding format used.
ST_BACnet_CharacterStringExtListEntry [▶ 356]	List of BACnet strings.
ST_BACnet_Date [▶ 356]	Date with day, day of the week, month, and year.
ST_BACnet_DateTime [▶ 357]	Describes date and time.
ST_BACnet_Diagnosis [▶ 357]	See BACnet diagnostic data.
ST_BACnet_EventTransitionBits [▶ 358]	PLC mapping of BACnet data type BACnetEventTransitionBits (properties *Event_Enable* and *Acked_Transitions*).
ST_BACnet_ExceptionScheduleBool [▶ 359]	Structure for data exchange of the property *exception_schedule* with the aid of function block FB_BACnet_ExceptionScheduleProperty [▶ 283]
ST_BACnet_ExceptionScheduleEntryBool [▶ 359]	Partial data of property ExceptionSchedule of the Schedule object
ST_BACnet_LimitEnable [▶ 362]	PLC mapping of BACnet data type BACnetLimitEnable. See BACnet specification DIN EN ISO 16484-5 for property *Limit_Enable*.
ST_BACnet_LogBufferEntryReal [▶ 362]	PLC mapping of BACnet data type BACnetLogRecord for *Log_Datum* of type *Real*. See BACnet specification DIN EN ISO 16484-5 for property *Log_Buffer*.
ST_BACnet_LogBufferReal [▶ 362]	Structure for data exchange of property *Log_Buffer* with the aid of function block FB_BACnet_LogBufferProperty [▶ 287]

Data Types	Description
ST_BACnet_NSinkEvent [▶ 363]	PLC mapping of the data of an event entry of the BACnet Notification Sink.
ST_BACnet_ObjectIdentifierList [▶ 363]	Structure for describing the object list of a device. See BACnet specification DIN EN ISO 16484-5 for property Object_List (device object).
ST_BACnet_ObjectTypesSupported [▶ 364]	PLC mapping of BACnet data type BACnetObjectTypesSupported. See BACnet specification DIN EN ISO 16484-5 for property *Protocol_Object_Types_Supported*.
ST_BACnet_ProgramHandshakeRequests [▶ 365]	Structure for operating the program object
ST_BACnet_ProgramHandshakeStates [▶ 365]	Structure for operating the program object
ST_BACnet_RecipientListDevice [▶ 365]	Structure for data exchange of property *Recipient_List* with the aid of function block FB_BACnet_RecipientListProperty [▶ 294]
ST_BACnet_RecipientListDeviceEntry [▶ 365]	Partial structure for data exchange of property *Recipient_List* with the aid of function block FB_BACnet_RecipientListProperty
ST_BACnet_ServicesSupported [▶ 366]	PLC mapping of BACnet data type BACnetServicesSupported. See BACnet specification DIN EN ISO 16484-5 for property *Protocol_Services_Supported.*
ST_BACnet_StatusFlags [▶ 367]	PLC mapping of BACnet data type BACnetStatusFlags. See BACnet specification DIN EN ISO 16484-5 for property *Status_Flags*.
ST_BACnet_Time [▶ 368]	PLC mapping of BACnet data type Time. See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.
ST_BACnet_TimeValue [▶ 368]	PLC mapping of BACnet data type BACnetTimeValue for entries of type Bool or Null. See BACnet specification DIN EN ISO 16484-5 for data type BACnetTimeValue.
ST_BACnet_TimeValueBool [▶ 368]	PLC mapping of BACnet data type BACnetTimeValue for entries of type Bool.
ST_BACnet_TimeValueList [▶ 369]	List of BACnet_TimeValue entries.
ST_BACnet_Value [▶ 369]	Partial structure of ST_BACnet_TimeValue
ST_BACnet_WeeklyScheduleBool [▶ 370]	Structure for data exchange of property *Weekly_Schedule* with the aid of function block FB_BACnet_WeeklyScheduleProperty [▶ 297]

BACnet Enumerationen

Data Types	Description
E_BACNETACTION [▶ 332]	PLC mapping of BACnet data type BACnetAction. See BACnet specification DIN EN ISO 16484-5 for property *Action*.
E_BACNETADAPTERSTATUS [▶ 332]	Status of the BACnet adapter.
E_BACNETBINARYPV [▶ 333]	PLC mapping of BACnet data type BACnetBinaryPV. See BACnet specification DIN EN ISO 16484-5 for property *Present_Value* of binary* objects.
E_BACNETDATATYPES [▶ 333]	List of possible BACnet data types (excerpt).
E_BACNETDAYSOFWEEKBITS [▶ 336]	Bit assignment of the days of the week
E_BACNETDEVICESTATUS [▶ 336]	Status of the BACnet server object (see BACnet specification DIN EN ISO 16484-5 for BACnet device object and property *System_Status*).
E_BACNETEVENTSTATE [▶ 336]	PLC mapping of BACnet data type BACnetEventState. See BACnet specification DIN EN ISO 16484-5 for property *Event_State*.
E_BACNETEVENTTRANSITIONBIT [▶ 337]	Bit assignment of the event transition flags [▶ 358] Event_Enable and *Acked_Transitions*).

Data Types	Description
E_BACNETEVENTTYPE [▶ 337]	PLC mapping of BACnet data type BACnetEventType. See BACnet specification DIN EN ISO 16484-5 for property *Event_Type*.
E_BACNETFILEACCESSMETHOD [▶ 338]	PLC mapping of BACnet data type BACnetFileAccessMethod. See BACnet specification DIN EN ISO 16484-5 for property *File_Access_Method*.
E_BACNETLIFESAFETYMODE [▶ 339]	PLC mapping of BACnet data type BACnetLifeSafetyMode. See BACnet specification DIN EN ISO 16484-5 for property *Mode* and *Accepted_Modes*.
E_BACNETLIFESAFETYOPERATION [▶ 339]	PLC mapping of BACnet data type BACnetLifeSafetyOperation. See BACnet specification DIN EN ISO 16484-5 for property *Operation_Expected*.
E_BACNETLIMITENABLEBITS [▶ 340]	Bit assignment of the Limit-Enable-Flags [▶ 362] (Limit_Enable).
E_BACNETLOGGINGTYPE [▶ 340]	PLC mapping of BACnet data type BACnetLoggingType. See BACnet specification DIN EN ISO 16484-5 for property *Logging_Type*.
E_BACNETLOOPMODE [▶ 340]	Operating modes of PLC LOOP object FB_BACnet_LOOP [▶ 201]
E_BACNETNOTIFYTYPE [▶ 341]	PLC mapping of BACnet data type BACnetNotifyType. See BACnet specification DIN EN ISO 16484-5 for property *Notify_Type*.
E_BACNETOBJECTSUPPORTEDBITS [▶ 341]	Bit assignment of the Object Types-Supported flags [▶ 364] (Protocol_Object_Types_Supported).
E_BACNETOBJECTTYPE [▶ 342]	List of possible BACnet objects (excerpt).
E_BACNETPERSISTENTDATASTATE [▶ 343]	PLC mapping of proprietary BACnet data type PersistentDataState (see BACnet device object
E_BACNETPIDTUNINGMODE	Tuning modes of block FB_BACnet_LOOP [▶ 201]
E_BACNETPOLARITY [▶ 343]	PLC mapping of BACnet data type BACnetPolarity. See BACnet specification DIN EN ISO 16484-5 for property *Polarity*.
E_BACNETPRIORITY [▶ 344]	List of possible BACnet priorities of a commandable property (e.g. *Present_Value*).
E_BACNETPROGRAMERROR [▶ 345]	PLC mapping of BACnet data type BACnetProgramError. See BACnet specification DIN EN ISO 16484-5 for property *Reason_For_Halt* and function block description FB_BACnet_Program [▶ 219]
E_BACNETPROGRAMREQUEST	PLC mapping of BACnet data type BACnetProgramRequest. See BACnet specification DIN EN ISO 16484-5 for property *Program_Change* and function block description FB_BACnet_Program
E_BACNETPROGRAMSTATE [▶ 345]	PLC mapping of BACnet data type BACnetProgramState. See BACnet specification DIN EN ISO 16484-5 for property *Program_State* and function block description FB_BACnet_Program
E_BACNETPROPERTYIDENTIFIER [▶ 346]	List of possible BACnet properties (excerpt).
E_BACNETRELIABILITY [▶ 349]	List of possible values of BACnet property *Reliability* (excerpt).
E_BACNETSEGMENTATION [▶ 350]	PLC mapping of BACnet data type BACnetSegmentation. See BACnet specification DIN EN ISO 16484-5 for property *Segmentation_Supported*.
E_BACNETSILENCEDSTATE [▶ 351]	PLC mapping of BACnet data type BACnetSilencedState. See BACnet specification DIN EN ISO 16484-5 for property *Silenced*.
E_BACNETSTATUSFLAGS [▶ 351]	Bit assignment of the Status flags [▶ 367] (property Status_Flags).

Data Types	Description
E_BACNETSTRINGENCODINGTYPES [▶ 351]	The enumeration contains a list of the string encodings supported by the BACnet driver.
E_BACNETUNIT [▶ 352]	PLC mapping of BACnet data type BACnetEngineeringUnits. See BACnet specification DIN EN ISO 16484-5 for property *Units*.

Also see about this

- E_BACNETPIDTUNINGMODE [▶ 343]
- E_BACNETPROGRAMREQUEST [▶ 345]

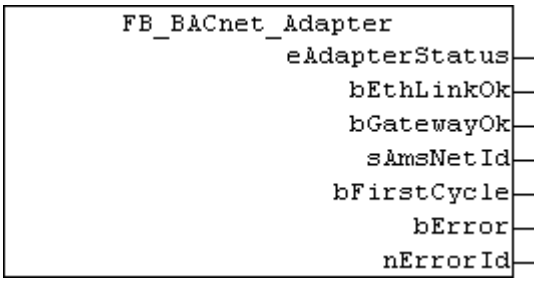
4.1 FB_BACnet_Adapter

Function block for linking the PLC program with a local BACnet adapter (network card). The linking of the function block takes place with the aid of process data.

The process data can be linked manually or automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.

Attention:

This block must not be instantiated in the PLC program! An instance of the block is already included in the global data of the PLC library (*fbBACnet_Adapter*), where it is automatically detected by the PLC automapping. Instantiation in the PLC program is only necessary if several BACnet adapters are used in a PLC program!



Application

The function block FB_BACnet_Adapter is used to read the state of the local BACnet adapter (process data *Device Status*) and output it in the PLC program. In addition, the process data*AmsNetId* is used to read the AMS NetID of the BACnet adapter and output it to **sAmsNetId**.

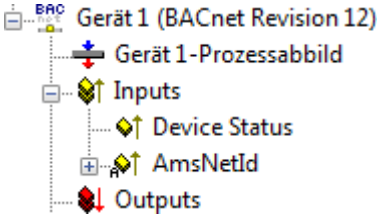


Fig. 5: Fig. 1: Process data of the BACnet adapter in the System Manager.

As described above, one instance of the block FB_BACnet_Adapter is already provided by the PLC library as a global instance and must therefore *not* be explicitly created. The global instance is recognized by the PLC automapping and automatically linked to the corresponding BACnet adapter in the System Manager.

The block instance is required by the BACnet function blocks FB_BACnet_Device [▶ 196] and FB_BACnet_RemoteDevice [▶ 241]. The hierarchy between FB_BACnet_Adapter, FB_BACnet_Device [▶ 196] and a BACnet object of type AnalogValue is shown below:

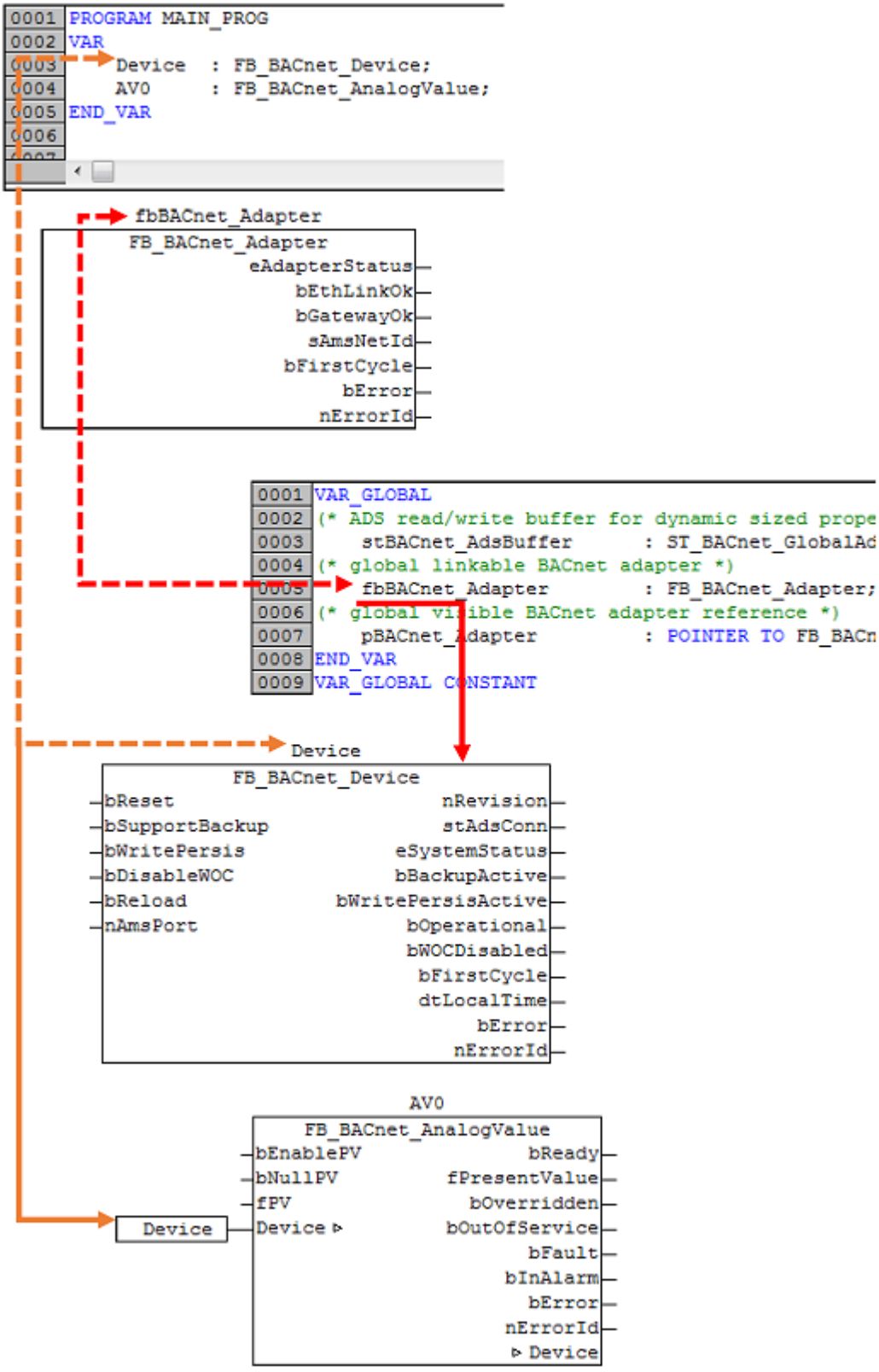


Fig. 6: Fig. 2: Relationship between FB_BACnet_Adapter, FB_BACnet_Device and a BACnet function block (e.g. FB_BACnet_AnalogValue).

Notes re. the figure:

- **Red dashed:** Function block call FB_BACnet_Adapter from the global data of the PLC Library. Each call updates the global pointer *pBACnet_Adapter*, which is used by the blocks FB_BACnet_Device [▶ 196] and FB_BACnet_RemoteDevice [▶ 241] as adapter reference.
- **Red:** Device blocks FB_BACnet_Device [▶ 196] and FB_BACnet_RemoteDevice [▶ 241] will use the global instance of the BACnet adapter internally to check the status of the BACnet connection and read the AMS NetID. The instance is determined via the global pointer *pBACnet_Adaper*. The global pointer can also be overwritten in the PLC program with its own adapter instance, before FB_BACnet_Device [▶ 196] and FB_BACnet_RemoteDevice [▶ 241] are called. In this way it is possible to realize PLC programs with multiple adapters, for example.
- **Orange dashed:** Function block call FB_BACnet_Device [▶ 196] with the instance from the PLC program. The user creates the instance locally or globally in the PLC program. It is linked to the corresponding device in the TwinCAT System Manager through PLC automapping.
- **Orange:** Transfer of the device instance (FB_BACnet_Device [▶ 196]) to the function block of the BACnet object. The function block of the BACnet object reads among other things the device status of the instance.

Tip:

The instance of the block FB_BACnet_Adapter does not have to called in the PLC program. If required, the adapter is called automatically by the blocks FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241].

VAR_OUPUT

```
eAdapterStatus  : E_BACNETADAPTERSTATUS;
bEthLinkOk      : BOOL;
bGatewayOk      : BOOL;
sAmsNetId       : T_AmsNetId;
bFirstCycle     : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

eAdapterStatus: Status of the BACnet adapter; see: E_BACNETADAPTERSTATUS [▶ 332].

bEthLinkOk: The local Ethernet connection is active (cable connected, adapter linked) if the output is set to *TRUE*.

bGatewayOk: The configured gateway can be reached if the output is set to *TRUE*.

sAmsNetId: Output of the AMS NetID of the local BACnet adapter (can be used for asynchronous access to BACnet objects via ADS).

bFirstCycle: Is set for one cycle when the block instance is first called after a PLC reset or restart.

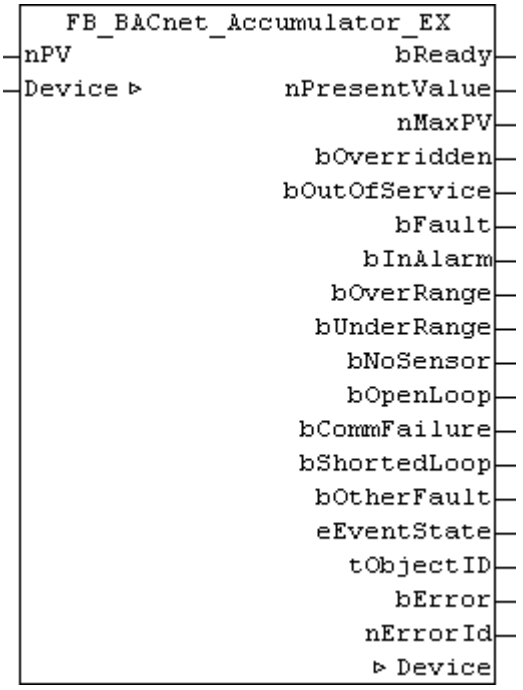
bError: An error is pending.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

If there is no network connection (**bEthLinkOk** = *FALSE*) or the gateway cannot be reached (**bGatewayOk** = *FALSE*), all remote objects (BACnet objects under one client) are blocked. Objects of the local BACnet server are not affected.

4.2 BACnet Server Objects

4.2.1 FB_BACnet_Accumulator_EX



Application

The function block FB_BACnet_Accumulator can be used for reading and write access to a BACnet object of type *Accumulator* (ACC).

An accumulator object is typically used for mapping meter readings to BACnet. The Example [▶ 157] illustrates the interfacing of an M-Bus counter with an accumulator object.

The basic and *_EX* versions of the block differ in terms of how the property *Present_Value* is written and the number of status outputs. In the basic version write access to the object is via the property *Value_Set*. Before the value acceptance in the property *Present_Value*, the previous value including time stamp is saved. In this way the property *Value_Change_Time* indicates when the property *Present_Value* was last updated, for example. In the case of an external counter (e.g. M-Bus), this feature can be used to check the up-to-dateness of the value.

VAR_INPUT

Variables shown in gray color are not included in the basic version of the function block. Variables shown in black are not included in the *_EX* version of the function block.

```
nPV         : UDINT;
nValueSet   : UDINT;
```

nPV: value to be written to the property *Present_Value*. The process data are written when there is a change.

nValueSet: value to be written to the property *Value_Set*. Write access also triggers update of the properties *Value_Change_Time* and *Value_Before_Change*. The process data are written when there is a change.

VAR_OUPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
nMaxPV          : UDINT;
bOverridden     : BOOL;
```

```
bOutOfService  : BOOL;
bFault         : BOOL;
bInAlarm       : BOOL;
bOverRange     : BOOL;
bUnderRange    : BOOL;
bNoSensor      : BOOL;
bOpenLoop      : BOOL;
bCommFailure   : BOOL;
bShortedLoop   : BOOL;
bOtherFault    : BOOL;
eEventState    : E_BACNETEVENTSTATE;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError         : BOOL;
nErrorId       : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_value*).

nMaxPV: Current value of the property *Max_Pres_Value* of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Max_Pres_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Event_State*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

```
Device         : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

The following example illustrates the implementation of an M-Bus counter on a BACnet object of type *Accumulator*.

```
0003 (* M-BUS *)
0004     MBUS         : FB_MBUSKL6781;
0005     MBUSCom      : ST_MBUS_Communication;
0006     MBUSRaw      : FB_MBUS_RawData;
0007
0008 (* BACnet *)
0009     Device       : FB_BACnet_Device;
0010     pAccValue    : POINTER TO UDINT;
0011     ACC          : FB_BACnet_Accumulator;
0012         (* ~(BACnet_ObjectName   : MAIN.ACC : NOLINK) *)
0013     ACCRel       AT%Q*:WORD;
0014         (* ~(BACnet_ObjectType   : ACC : NOLINK)
0015             (BACnet_ObjectName   : MAIN.ACC : OBJREF, NOLINK)
0016             (BACnet_Reliability : : LINK) *)
```

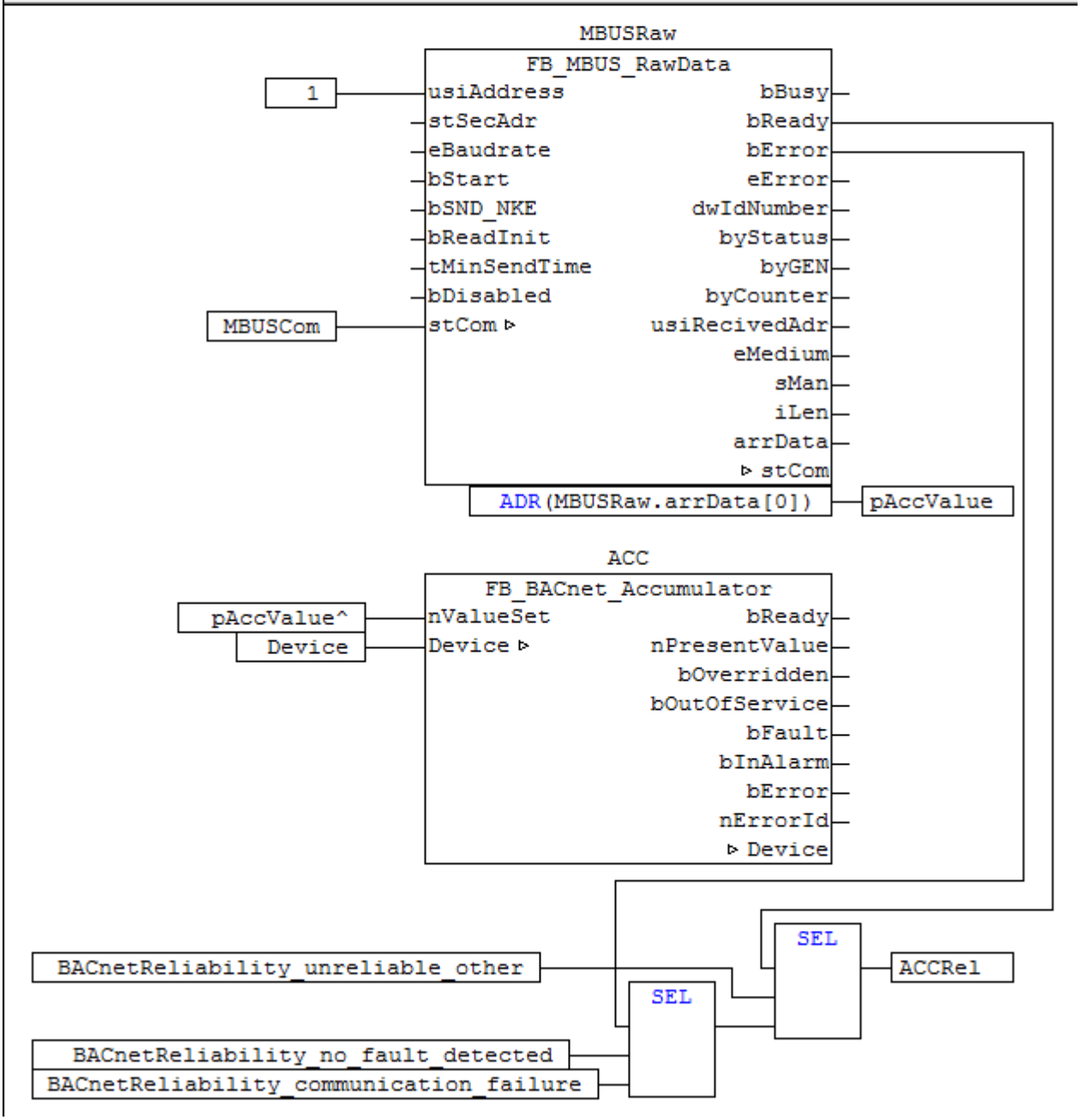
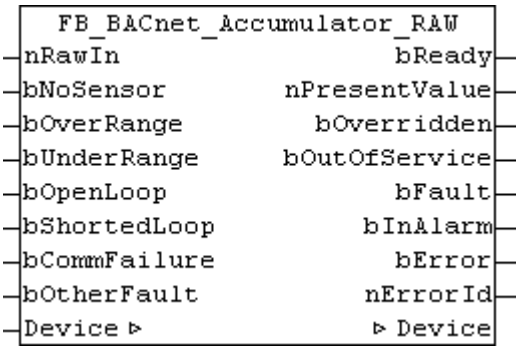


Fig. 7: Fig. 1: Example for the implementation of an M-Bus counter in the BACnet object Accumulator in the PLC program.

The BACnet object records the value of the input nValueSet and saves it as property Present_Value. Further information for setting the property Reliability of the BACnet object is provided by the bError and bReady states of the M-Bus block. This property indicates the state of the value of the BACnet object. If the M-Bus block reports bReady = TRUE, the BACnet object assumes state no_fault_detected → value valid.

4.2.2 FB_BACnet_Accumulator_RAW



Application

The function block FB_BACnet_Accumulator_RAW can be used for read and write access to a BACnet object of type *Accumulator*.

In contrast to the standard and *_EX* versions of the block, the raw value, and the state of the property *Reliablitity* are provided by function block inputs, not directly by the IO hardware. For example, the state of a counter input can be mapped from a sub-bus system in PLC code to a BACnet object (signal conversion from sub-bus systems or virtual data points to BACnet, see Example [▶ 160]).

VAR_INPUT

```
nRawIn          : UINT;
bNoSensor       : BOOL;
bOverRange      : BOOL;
bUnderRange     : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bCommFailure    : BOOL;
bOtherFault     : BOOL;
```

nRawIn: Raw value input of the object in the range -32768 to 32767. The input is linked to the process data "RawIoAccumulatorUnsignedValue" of the BACnet object. Changes in the value of **nRawIn** are offset with the property *Prescale* to calculate the value of the property *Present_Value* (provided the object state is not *out_of_service*).

bNoSensor, bOverRange, bUnderRange, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoSensor** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Reliability*.

VAR_OUPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

The following example illustrates the implementation of a digital pulse signal on a BACnet object of type *Accumulator*.

```
0003 (* Virtual value *)
0004     diPulseInput AT%I*:BOOL;
0005     diPulseOk    AT%I*:BOOL;
0006     rtrigPulse   : R_TRIG;
0007     nCounter     : UINT;
0008
0009 (* BACnet *)
0010     Device       : FB_BACnet_Device;
0011     ACCRawCount  : FB_BACnet_Accumulator_RAW;
```

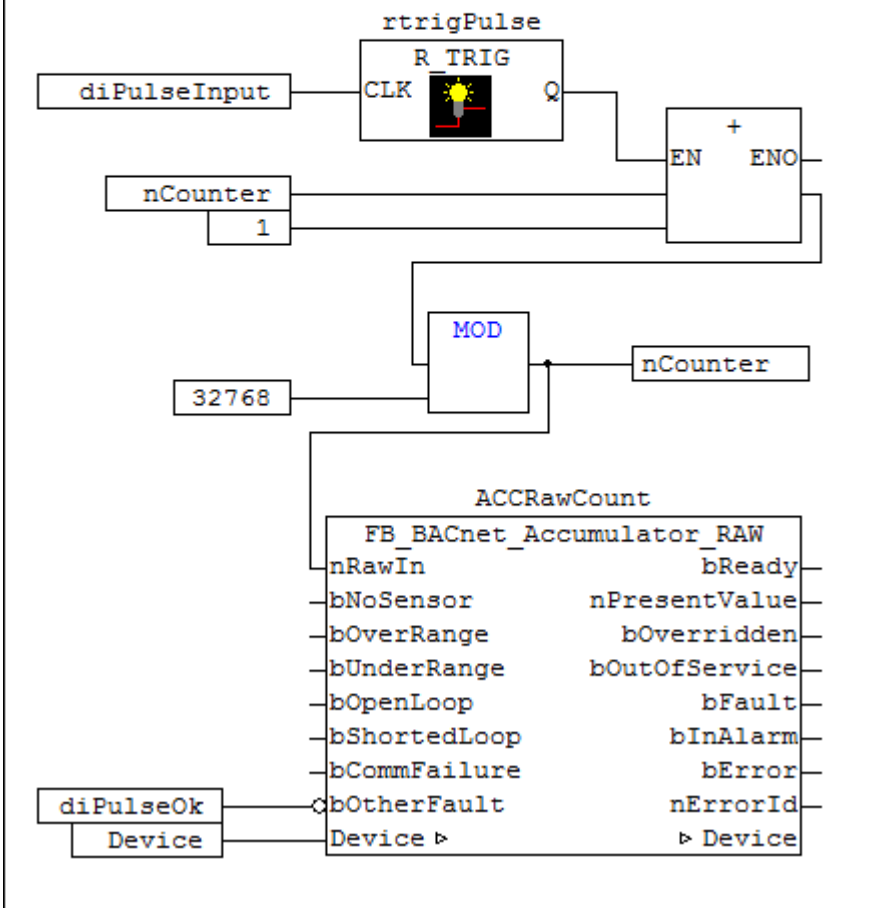
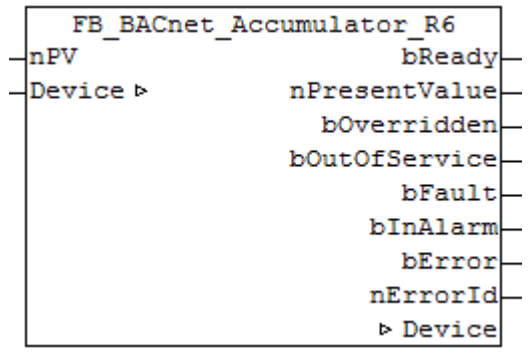


Fig. 8: Fig. 1: Example for the implementation of a pulse signal in the BACnet object Accumulator in the PLC program.

The BACnet object records the changes in the input nRawIn. An overflow of the UINT value is non-critical, since the BACnet object can respond accordingly. The difference of the input value is counted in module 32768. Further information is provided by the input diPulseOk. This input indicates that the pulse-generating side (e.g. an external counter) is ready.

4.2.3 FB_BACnet_Accumulator_R6



Application

The function block FB_BACnet_Accumulator_R6 can be used for reading and write access to a BACnet object of type *Accumulator*. Write access in this version (*_R6*) is the same as in the previous library version (Revision 6): The value of **nPV** is transferred directly into the property *Present_Value*.

VAR_INPUT

```
nPV         : UDINT;
```

nPV: Value that is written directly to the Property *Present_Value*.

VAR_OUPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bError: An error is pending.

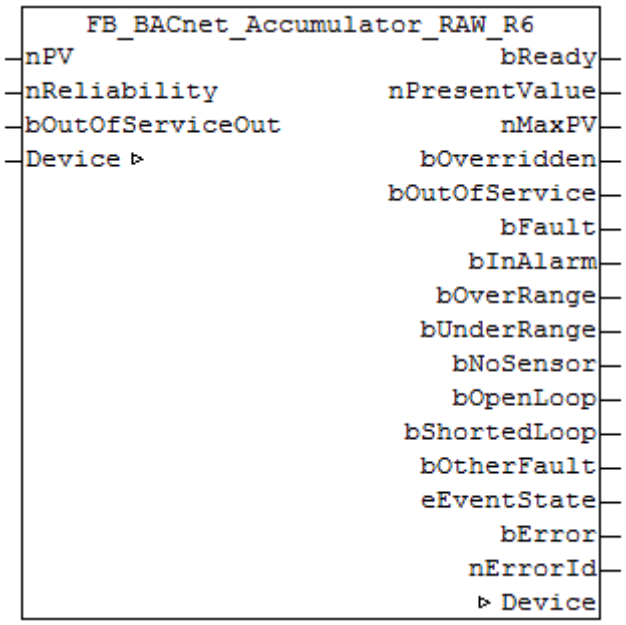
nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.4 FB_BACnet_Accumulator_RAW_R6



Application

The function block can be used for read and write access to a BACnet object of type *Accumulator* (ACC).

In contrast to the standard and *_EX* versions of the block, the current counter value (*Present_Value*) and the state of the property *Reliablitity* are provided by function block inputs, not directly by the IO hardware. For example, the state of a counter input can be mapped from a sub-bus system in PLC code to a BACnet object (signal conversion from sub-bus systems or virtual data points to BACnet, see Example [▶ 164]).

VAR_INPUT

```
nPV               : UDINT;
nReliability      : WORD;
bOutOfServiceOut  : BOOL;
```

nPV: Value that is written directly to the Property *Present_Value*.

nReliability: Input sets the appropriate state [▶ 349] of the property *Reliability*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Reliability*.

bOutOfServiceOut: *TRUE* sets the object state to *out_of_service*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

VAR_OUPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
nMaxPV          : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm       : BOOL;
bOverRange     : BOOL;
bUnderRange    : BOOL;
bNoSensor      : BOOL;
bOpenLoop      : BOOL;
bShortedLoop   : BOOL;
bOtherFault    : BOOL;
eEventState    : E_BACNETEVENTSTATE;
bError         : BOOL;
nErrorId       : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_value*).

nMaxPV: Current value of the property *Max_Pres_Value* of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Max_Pres_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Event_State*.

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device        : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

The following example illustrates the implementation of a digital pulse signal on a BACnet object of type *Accumulator*.

```
0002 VAR PERSISTENT
0003 (* Persistent counter value *)
0004     nCounter      : UDINT;
0005 END_VAR
0006 VAR
0007 (* Virtual value *)
0008     diPulseInput AT%I*:BOOL;
0009     diPulseOk    AT%I*:BOOL;
0010     rtrigPulse   : R_TRIG;
0011 (* BACnet *)
0012     Device       : FB_BACnet_Device;
0013     ACCRawCount  : FB_BACnet_Accumulator_RAW_R6;
0014 END_VAR
```

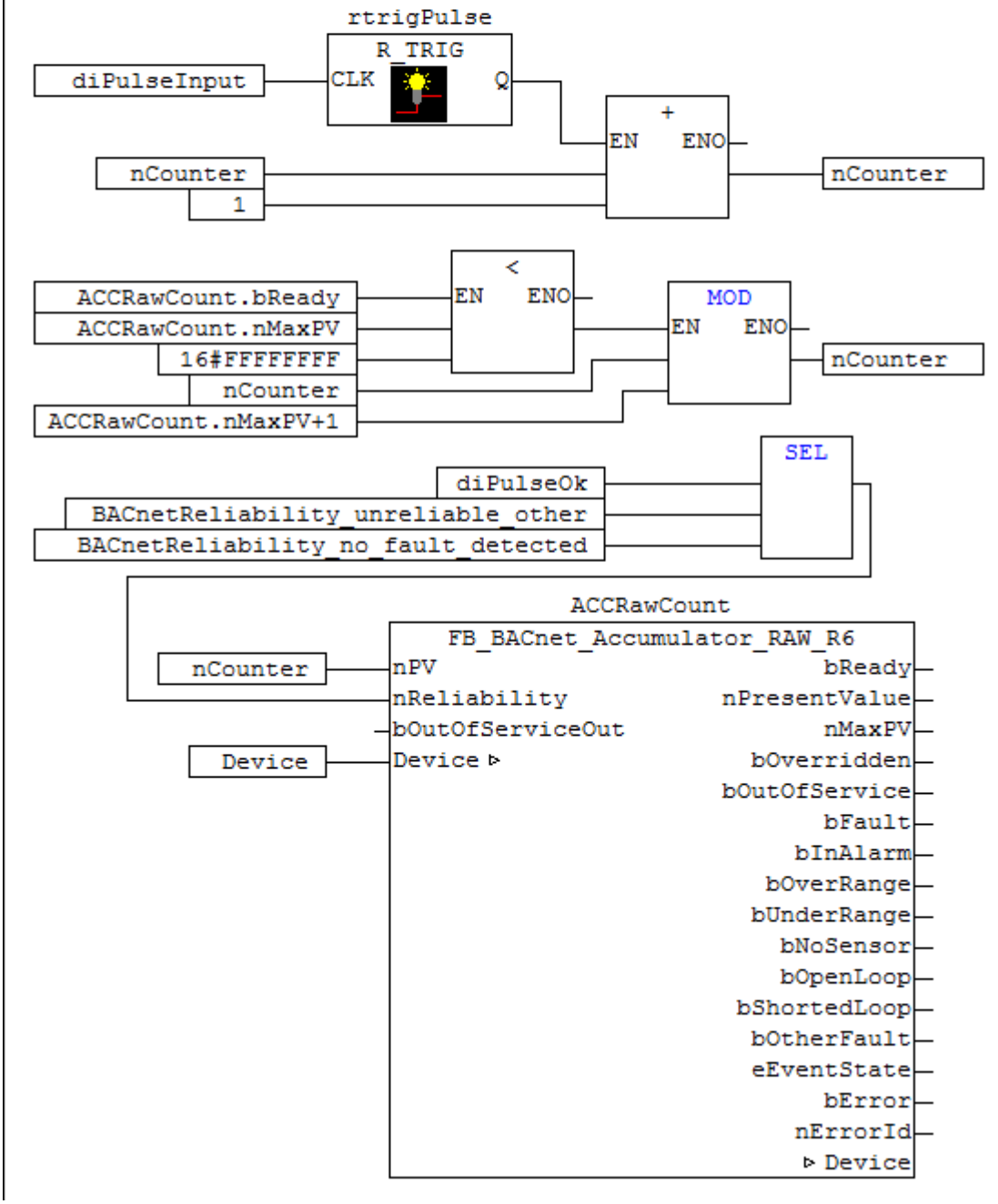
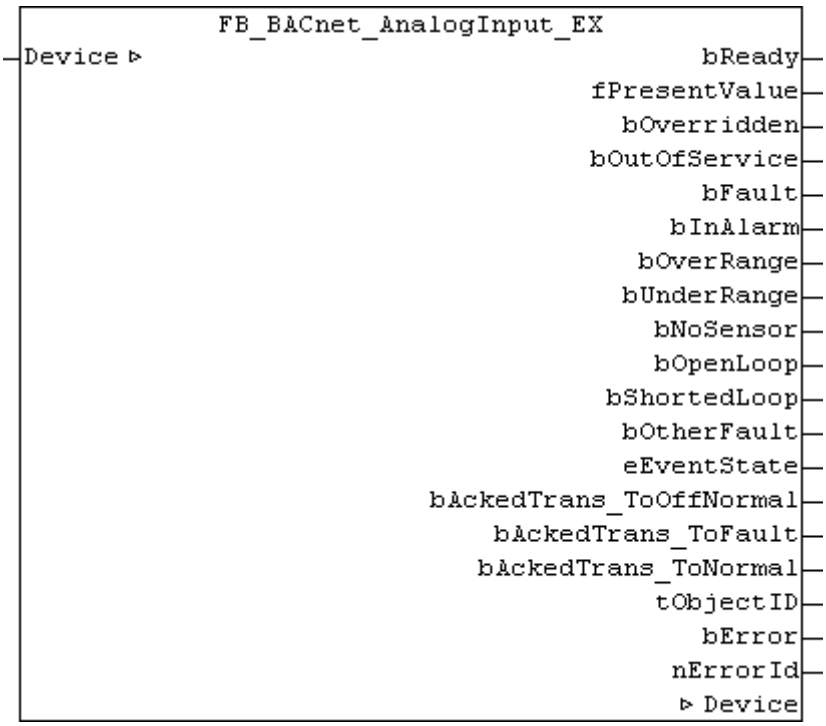


Fig. 9: Fig. 1: Example for the implementation of a pulse signal in the BACnet object Accumulator in the PLC program.

The BACnet object records the count value at the input **nPV**. An overflow of the UDINT value occurs when a maximum value is set or depending on data type. Further information is provided by the input **diPulseOk**. This input indicates that the pulse-generating side (e.g. an external counter) is ready.

4.2.5 FB_BACnet_AnalogInput_EX



Application

The function block FB_BACnet_AnalogInput can be used for read access to a BACnet object of type *AnalogInput* (AI).

VAR_OUPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bOverRange              : BOOL;
bUnderRange             : BOOL;
bNoSensor               : BOOL;
bOpenLoop               : BOOL;
bShortedLoop            : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Present_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

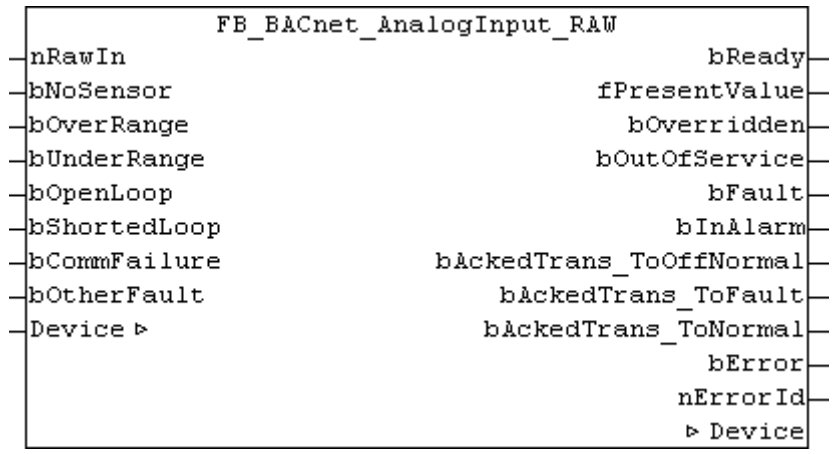
```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.6 FB_BACnet_AnalogInput_RAW



Application

The function block FB_BACnet_AnalogInput_RAW can be used for read and write access to a BACnet object of type *AnalogInput*.

In contrast to the standard and *_EX* versions of the block, the raw value and the state of the property *Reliablitity* are provided by function block inputs, not directly by the IO hardware. For example, the state of an analog value can be mapped from a sub-bus system in PLC code to a BACnet object (signal conversion from sub-bus systems or virtual data points to BACnet, see Example [▶ 168]).

VAR_INPUT

```
nRawIn                : INT;
bNoSensor             : BOOL;
bOverRange            : BOOL;
bUnderRange           : BOOL;
bOpenLoop             : BOOL;
```

```
bShortedLoop          : BOOL;
bCommFailure          : BOOL;
bOtherFault           : BOOL;
```

nRawIn: Raw value input of the object in the range -32768 to 32767. The input is linked to the process data "RawIoAnalogSignedValue" of the BACnet object. The value from **nRawIn** is offset with the property *Resolution* to calculate the value of the property *Present_Value* (provided the object state is not *out_of_service*).

bNoSensor, bOverRange, bUnderRange, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoSensor** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Reliability*.

VAR_OUPUT

```
bReady                    : BOOL;
fPresentValue             : REAL;
bOverridden               : BOOL;
bOutOfService             : BOOL;
bFault                    : BOOL;
bInAlarm                  : BOOL;
bAckedTrans_ToOffNormal   : BOOL;
bAckedTrans_ToFault       : BOOL;
bAckedTrans_ToNormal      : BOOL;
bError                    : BOOL;
nErrorId                  : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Present_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

Example

The following example shows the implementation of an integer value from EIB in the property *Present_Value* of an AnalogInput object:

```
0002 VAR
0003 (* EIB *)
0004     EIB          : KL6301;
0005     EIBAnIn      : EIB_2OCTET_SIGN_REC;
0006
0007 (* BACnet *)
0008     Device       : FB_BACnet_Device;
0009     AIRaw        : FB_BACnet_AnalogInput_RAW;
0010 END_VAR
```

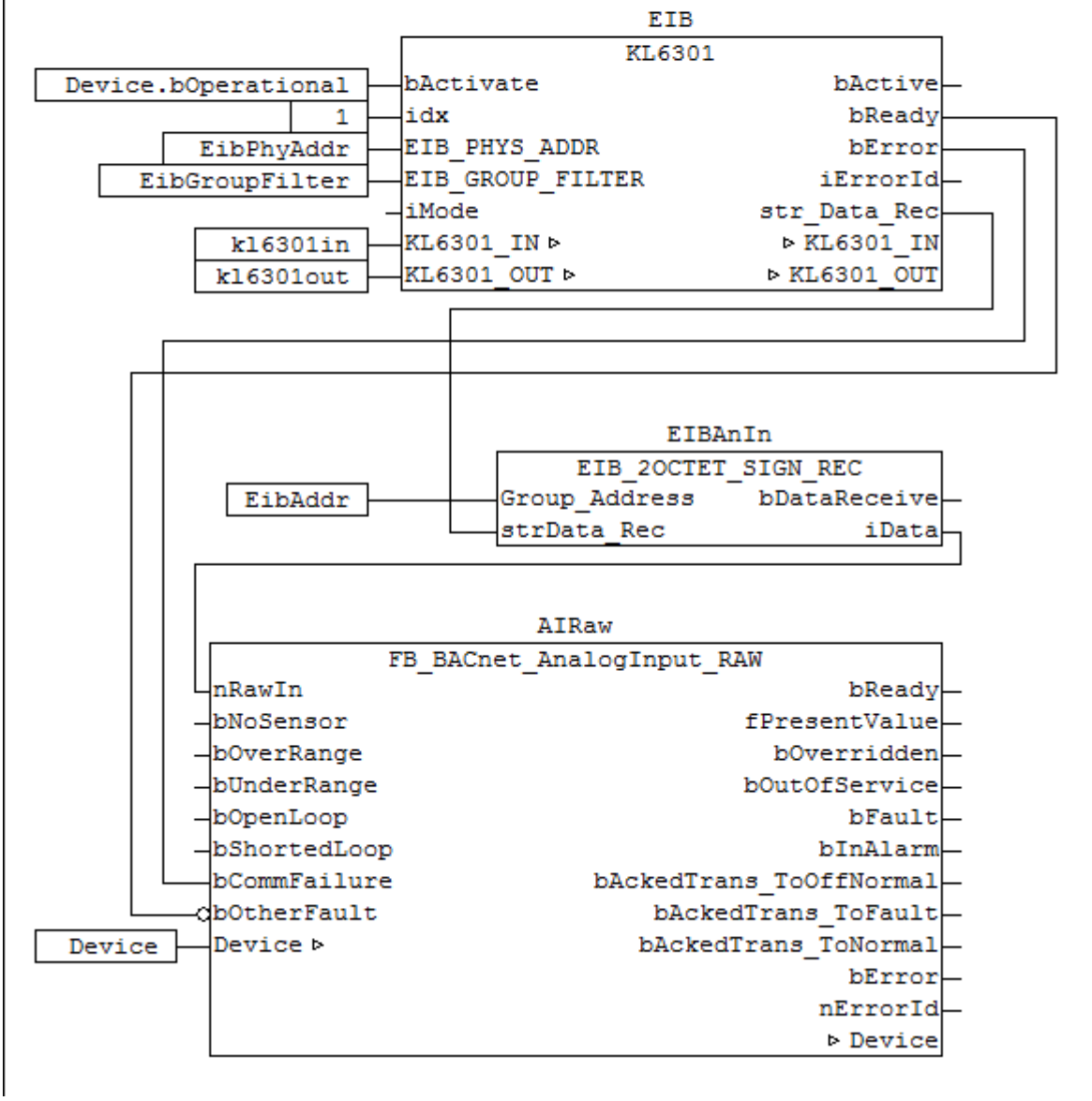
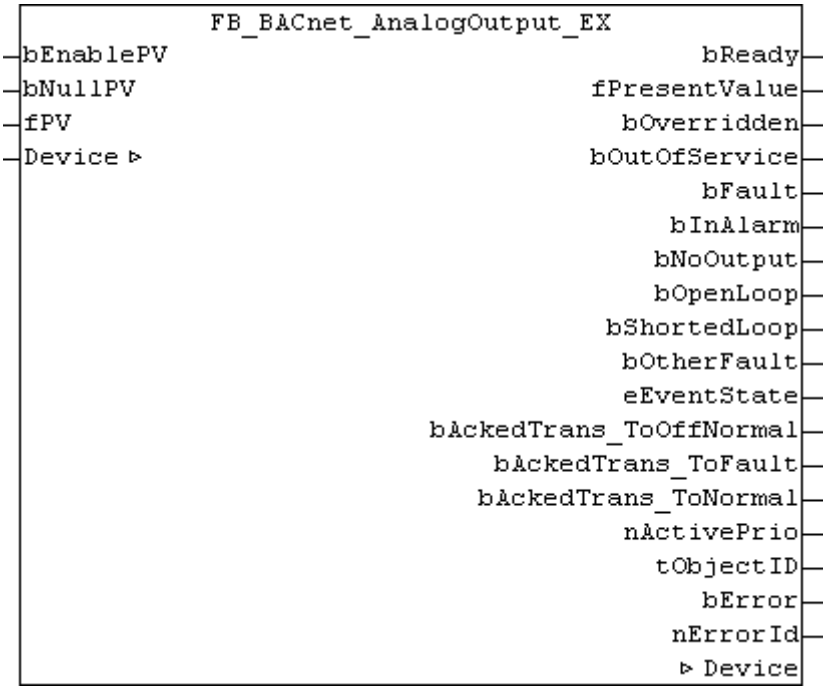


Fig. 10: Fig. 1: Example for the implementation of an integer value from EIB in the property Present_Value in the PLC program.

This example assumes the library "TcEIB.lib" is available. For further information on EIB please refer to the manual for the KL6301 terminal.

4.2.7 FB_BACnet_AnalogOutput_EX



Application

The function block FB_BACnet_AnalogOutput can be used for read access with priority 12 to the property *Present_Value* and for write access to a BACnet object of type AnalogOutput (AO).

VAR_INPUT

```
bEnablePV         : BOOL;
bNullPV           : BOOL;
fPV               : REAL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **fPV**; *FALSE* = write value of **fPV** as property value.

fPV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bNoOutput               : BOOL;
bOpenLoop               : BOOL;
bShortedLoop            : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nActivePrio             : UINT;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bNoOutput, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

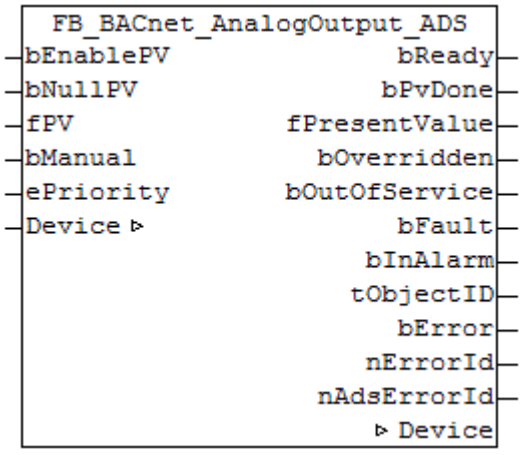
```
Device               : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.8 FB_BACnet_AnalogOutput_ADS



Application

The function block FB_BACnet_AnalogOutput_ADS can be used for read and write access to a BACnet object of type *AnalogOutput*.

In contrast to the standard and *_EX* versions, writing of the property *Present_Value* is realized with ADS-WRITE (acyclic).

VAR_INPUT

```
bEnablePV  : BOOL;
bNullPV    : BOOL;
fPV        : REAL;
bManual    : BOOL;
ePriority  : E_BACNETPRIORITY:=BACNETPRIORITY_12;
```

bEnablePV: enables the value of input **fPV**. As soon as the input is set to *TRUE*, write access to the commandable property *Present_Value* takes place with the value from **fPV** and the priority from **ePriority** (default: 12, if the input **ePriority** is not connected). If the input is set to *FALSE*, no further write access takes place (no change).

bNullPV: overwrites the entry of the commandable property *Present_Values* in priority **ePriority** with the value *NULL*, if **bEnablePV** is set to *TRUE*. The input **fPV** is ignored, as long as **bNullPV** is set to *TRUE*.

fPV: value to be written to the corresponding location of the property *Priority_Array* of the commandable property *Present_Value* if **bEnablePV** is set to *TRUE* and **bNullPV** is set to *FALSE*. The priority is determined with the input **ePriority** (default: 12, if the input **ePriority** is not connected). If the value changes, writing takes place acyclically (ADS WRITE).

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: the following evaluations are distinguished for the input:
- *FALSE*: write on value change
- Edge change *FALSE* → *TRUE*: write the property *Present_Value* on edge change.
- *TRUE*: write on value change *and* when the associated BACnet server changes to the "Operational" state (see FB_BACnet_Device [▶ 196]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 344]).

VAR_OUPUT

```
bReady         : BOOL;
bPvDone        : BOOL;
fPresentValue  : REAL;
bOverridden    : BOOL;
bOutOfService  : BOOL;
bFault         : BOOL;
bInAlarm       : BOOL;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError         : BOOL;
nErrorId       : UINT;
nAdsErrorId    : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPvDone: Positive edge if writing of property *Present_Value* via ADS was successful.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

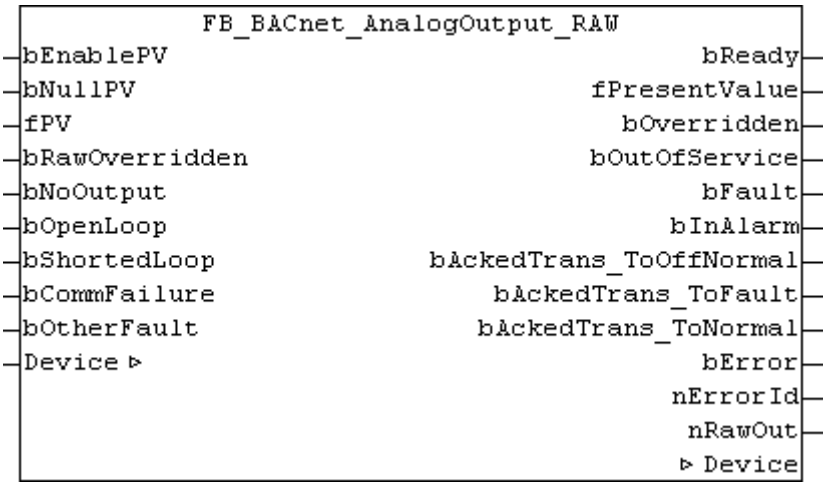
nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.9 FB_BACnet_AnalogOutput_RAW



Application

The function block FB_BACnet_AnalogOutput_RAW can be used for reading and write access to a BACnet object of type *AnalogOutput*.

In contrast to the standard or *_EX* version of the block, the flag of the property *Status_Flags* is *overridden*, and the value of the property *Reliablitity* is provided by function block inputs and mapped from the PLC program to the BACnet object. The PLC program maps the state of the property *Present_Value* to the hardware and the sub-bus system. Otherwise, this is dealt with directly by the IO hardware. In this way it is possible, for example, for a BACnet object to map the output value to a sub-bus system in PLC code (signal conversion to sub-bus systems or virtual points from BACnet, see Example [▶ 174]).

VAR_INPUT

```
bEnablePV       : BOOL;
bNullPV         : BOOL;
fPV             : REAL;
bRawOverridden  : BOOL;
bNoOutput       : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bCommFailure    : BOOL;
bOtherFault     : BOOL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **fPV**; *FALSE* = write value of **fPV** as property value.

fPV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

bRawOverridden: *TRUE* sets the flag *overridden* of the property *Status_Flags*. This indicates to BACnet that the value of the property *Present_Value* was decoupled from the hardware output (e.g. through manual local operation). See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bNoOutput, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoOutput** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Reliability*.

VAR_OUPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
nRawOut                 : INT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: see global constants BACnet Globals [▶ 328].

nRawOut: Raw value output of the object in the range -32768 to 32767. The output is linked to the process data "RawIoAnalogSignedValue" of the BACnet object. The value of **nRawOut** results from the property *Present_Value,* taking account of the property *Resolution*.

VAR_IN_OUT

```
Device              : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

The following example illustrates the implementation of the property *Present_Value* of an AnalogOutput object in EIB:

```
0002 VAR
0003 (* EIB *)
0004     EIB          : KL6301;
0005     EIBAnOut     : EIB_2OCTET_SIGN_SEND;
0006
0007 (* Manual override *)
0008     diManActive AT%I*:BOOL; (* key switch on cabinet *)
0009     aiManSlider AT%I*:INT; (* analog slider on cabinet *)
0010
0011 (* BACnet *)
0012     Device       : FB_BACnet_Device;
0013     AORaw        : FB_BACnet_AnalogOutput_RAW;
0014 END_VAR
```

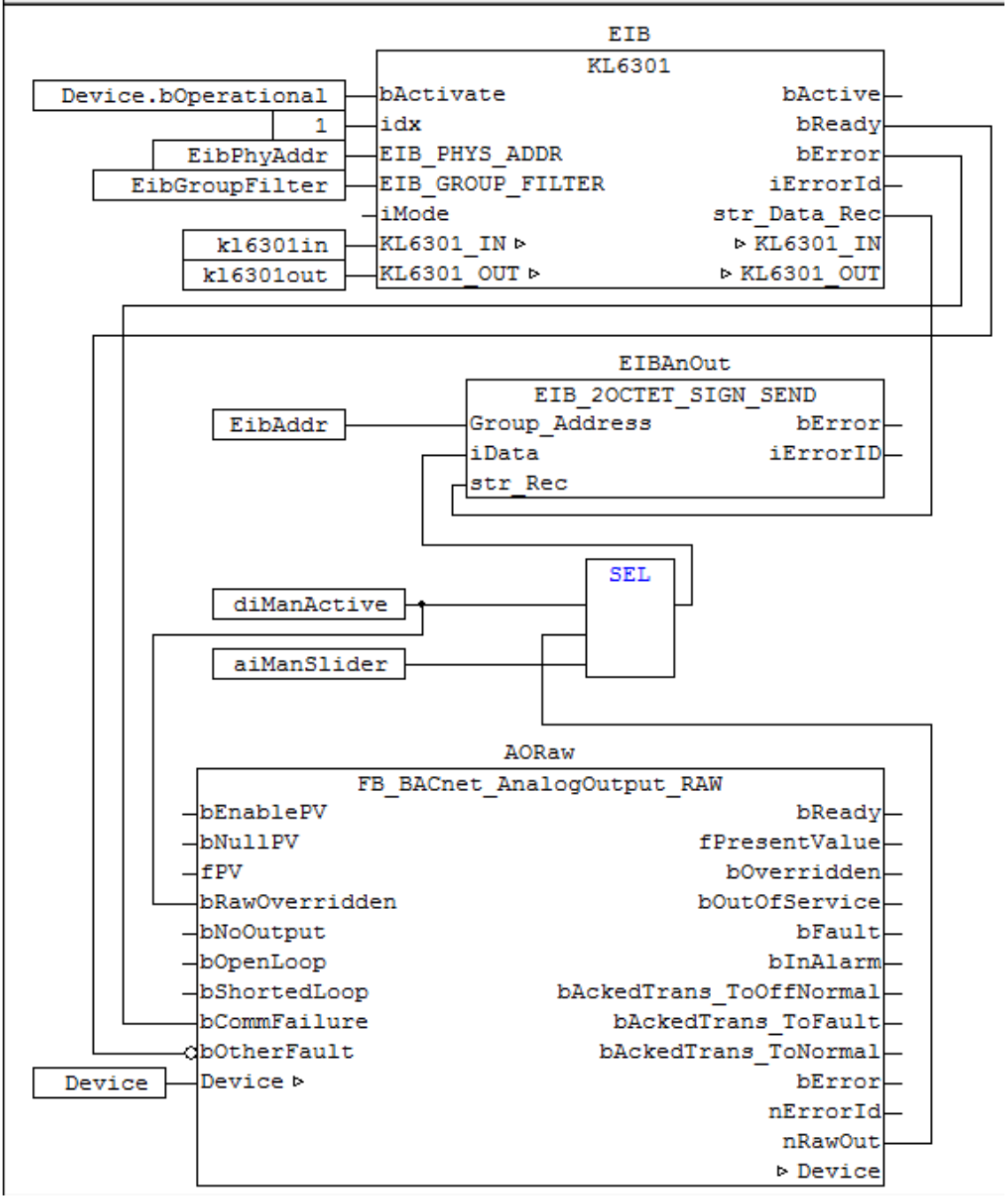
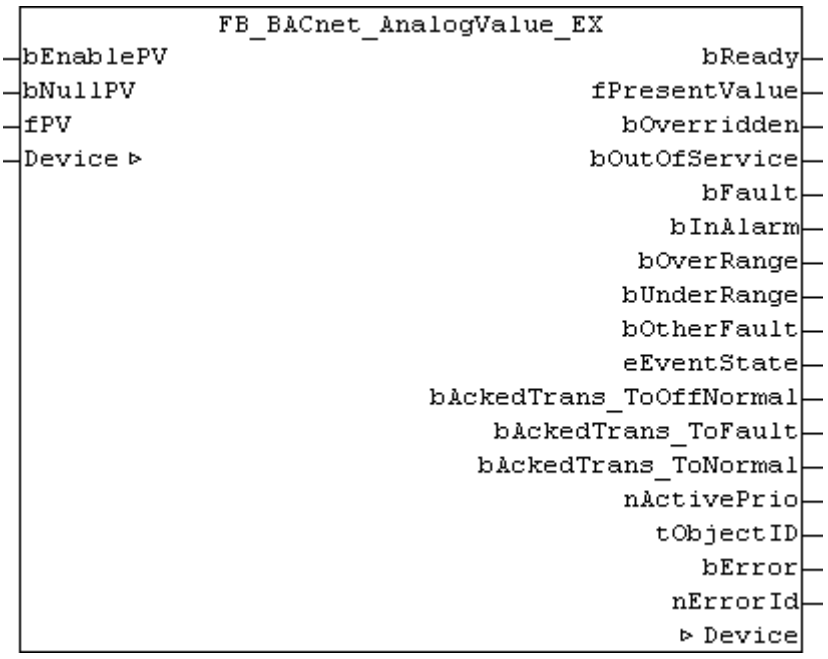


Fig. 11: Fig. 1: Example for the implementation of property Present_Value in EIB in the PLC program.

This example assumes the library "TcEIB.lib" is available. For further information on EIB please refer to the manual for the KL6301 terminal.

The implementation of local operation by means of a selector switch (digital input) and a control potentiometer (analog input) is also illustrated.

4.2.10 FB_BACnet_AnalogValue_EX



Application

The function block FB_BACnet_AnalogOutput can be used for read access with priority 12 to the property *Present_Value* and for write access to a BACnet object of type AnalogValue (AV).

VAR_INPUT

```
bEnablePV        : BOOL;
bNullPV          : BOOL;
fPV              : REAL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **fPV**; *FALSE* = write value of **fPV** as property value.

fPV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bOverRange              : BOOL;
bUnderRange             : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nActivePrio             : UINT;
```

```
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bOverRange, bUnderRange, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

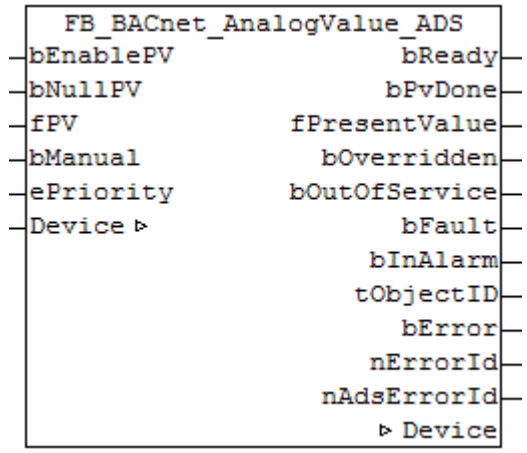
```
Device                  : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.11 FB_BACnet_AnalogValue_ADS



Application

The function block FB_BACnet_AnalogValue_ADS can be used for reading and write access to a BACnet object of type *AnalogValue*.

In contrast to the standard and *_EX* versions, writing of the property *Present_Value* is realized with ADS-WRITE (acyclic).

VAR_INPUT

```
bEnablePV   : BOOL;
bNullPV     : BOOL;
fPV         : REAL;
bManual     : BOOL;
ePriority   : E_BACNETPRIORITY:=BACNETPRIORITY_12;
```

bEnablePV: enables the value of input **fPV**. As soon as the input is set to *TRUE*, write access to the commandable property *Present_Value* takes place with the value from **fPV** and the priority from **ePriority** (default: 12, if the input **ePriority** is not connected). If the input is set to *FALSE*, no further write access takes place (no change).

bNullPV: overwrites the entry of the commandable property *Present_Values* in priority **ePriority** with the value *NULL*, if **bEnablePV** is set to *TRUE*. The input **fPV** is ignored, as long as **bNullPV** is set to *TRUE*.

fPV: value to be written to the corresponding location of the property *Priority_Array* of the commandable property *Present_Value* if **bEnablePV** is set to *TRUE* and **bNullPV** is set to *FALSE*. The priority is determined with the input **ePriority** (default: 12, if the input **ePriority** is not connected). If the value changes, writing takes place acyclically (ADS WRITE).

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: the following evaluations are distinguished for the input:
- *FALSE*: write on value change
- Edge change *FALSE* → *TRUE*: write the property *Present_Value* on edge change.
- *TRUE*: write on value change *and* when the associated BACnet server changes to the "Operational" state (see FB_BACnet_Device [▶ 196]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 344]).

VAR_OUPUT

```
bReady          : BOOL;
bPvDone         : BOOL;
fPresentValue   : REAL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
nAdsErrorId     : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPvDone: Positive edge if writing of property *Present_Value* via ADS was successful.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogValue* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

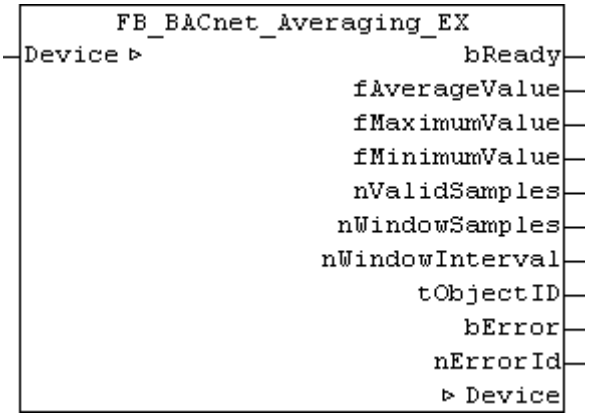
nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.12 FB_BACnet_Averaging_EX



Application

The function block FB_BACnet_Averaging can be used for read access to a BACnet object of type *Averaging* (AVG).

VAR_OUPUT

Variables are not included in the basic version of the function block.

```
bReady           : BOOL;
fAverageValue    : REAL;
fMaximumValue    : REAL;
fMinimumValue    : REAL;
nValidSamples    : UDINT;
nWindowSamples   : UDINT;
nWindowInterval  : UDINT;
tObjectID        : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;  (*siehe Info*)
bError           : BOOL;
nErrorId         : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the function block instance was not linked correctly in the System Manager.

fAverageValue: current average value (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Average_Value*).

fMaximumValue: largest sampled value (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Maximum_Value*).

fMinimumValue: smallest sampled value (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Minimum_Value*).

nValidSamples: number of valid values of the BACnet object (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Valid_Samples*).

nWindowSamples: number of values sampled over the period of *Window_Intervall* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Window_Samples*).

nWindowInterval: period over which the values are averaged in full seconds (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Window_Interval*).

tObjectID: object ID of the BACnet object (object type and object instance).

bError: an error is pending.

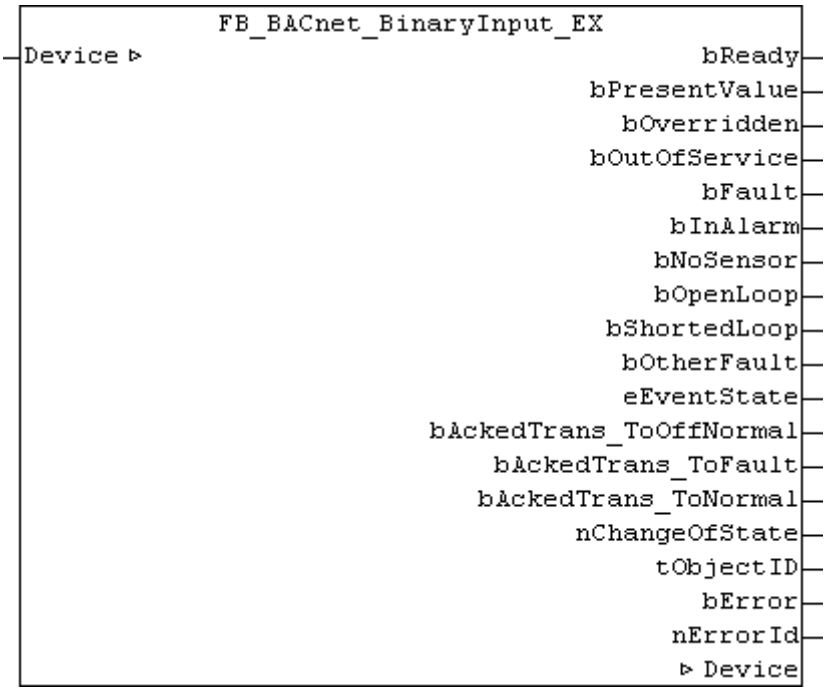
nErrorId: see global constants (BACnet Globals [▶ 328]).

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.13 FB_BACnet_BinaryInput_EX



Application

The function block FB_BACnet_AnalogInput can be used for read access to a BACnet object of type *BinaryInput* (BI).

VAR_OUPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bNoSensor               : BOOL;
```

```
bOpenLoop                : BOOL;
bShortedLoop             : BOOL;
bOtherFault              : BOOL;
eEventState              : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal  : BOOL;
bAckedTrans_ToFault      : BOOL;
bAckedTrans_ToNormal     : BOOL;
nChangeOfState           : UDINT;
tObjectID                : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                   : BOOL;
nErrorId                 : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Status_Flags*.

bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Acked_Transitions*).

nChangeOfState: Number of changes of state (*not* to be confused with changes in value) of the property *Present_Value* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Change_Of_State_Count*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet Globals [▶ 328]).

VAR_IN_OUT

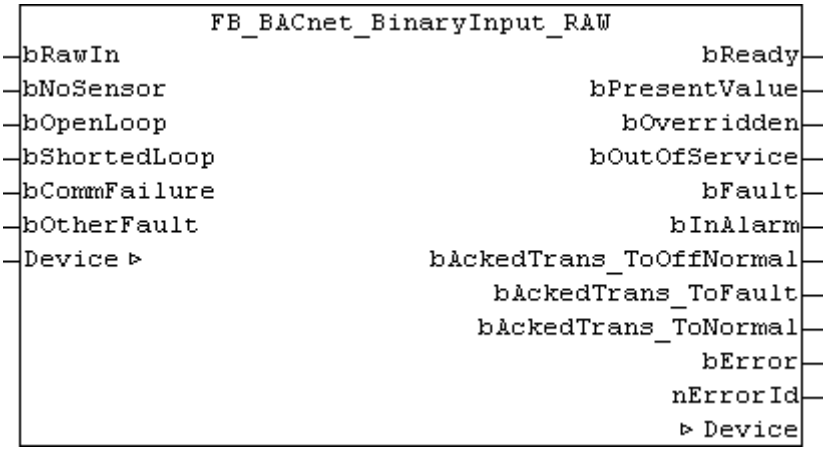
```
Device               : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.14 FB_BACnet_BinaryInput_RAW



Application

The function block FB_BACnet_BinaryInput_RAW can be used for reading and write access to a BACnet object of type *BinaryInput*.

In contrast to the standard and *_EX* versions of the block, the raw value, and the state of the property *Reliablitity* are provided by function block inputs, not directly by the IO hardware. For example, the state of a binary value can be mapped from a sub-bus system in PLC code to a BACnet object (signal conversion from sub-bus systems or virtual data points to BACnet, see Example [▶ 183]).

VAR_INPUT

```
bRawIn               : BOOL;
bNoSensor            : BOOL;
bOpenLoop            : BOOL;
bShortedLoop         : BOOL;
bCommFailure         : BOOL;
bOtherFault          : BOOL;
```

bRawIn: Raw value input of the object. The input is linked to the process data "RawIoBinaryBoolValue" of the BACnet object. The value from **bRawIn** is offset with the property *Polarity* to calculate the value of the property *Present_Value* (provided the object state is not *out_of_service*).

bNoSensor, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoSensor** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Reliability*.

VAR_OUPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

Example

The following example shows the implementation of a bit value from EIB in the property *Present_Value* of a BinaryInput object:

```
0002 VAR
0003 (* EIB *)
0004     EIB          : KL6301;
0005     EIBBinIn     : EIB_BIT_REC;
0006
0007 (* BACnet *)
0008     Device       : FB_BACnet_Device;
0009     BIRaw        : FB_BACnet_BinaryInput_RAW;
0010 END_VAR
```

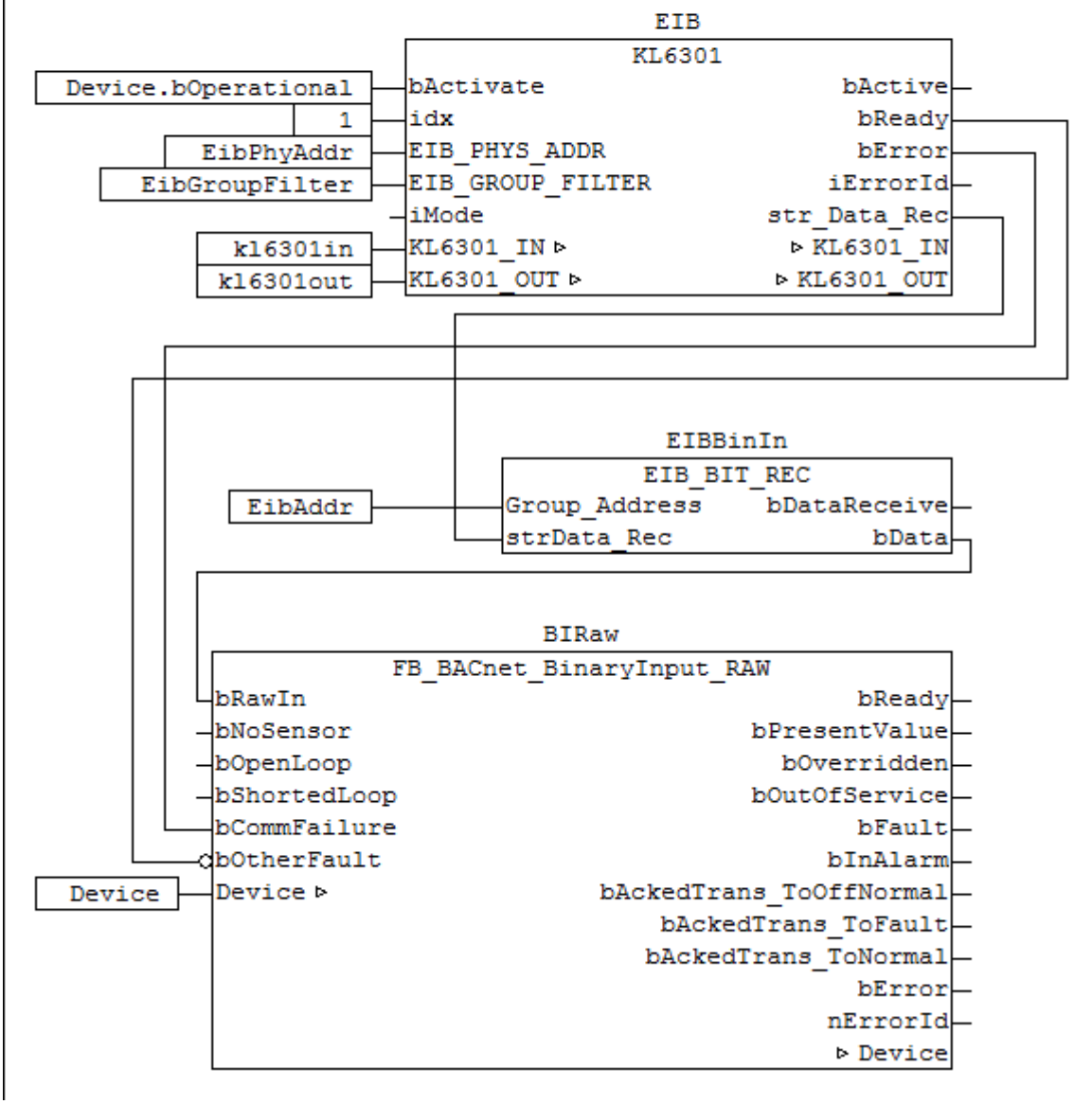
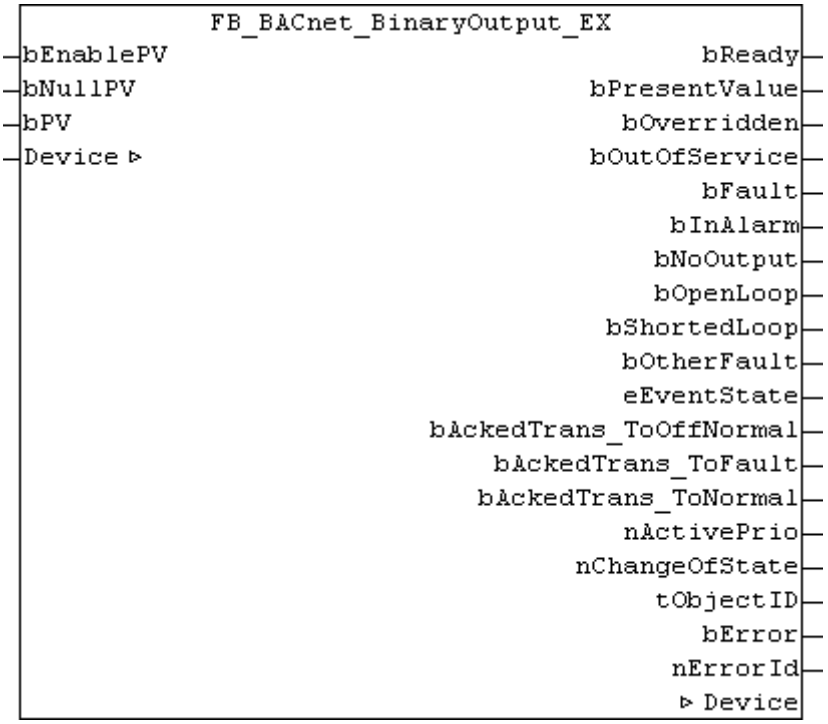


Fig. 12: Fig. 1: Example for the implementation of a bit value from EIB in the property Present_Value in the PLC program.

This example assumes the library "TcEIB.lib" is available. For further information on EIB please refer to the manual for the KL6301 terminal.

4.2.15 FB_BACnet_BinaryOutput_EX



Application

The function block FB_BACnet_BinaryOutput can be used for reading access with priority 12 to the property *Present_Value* and for write access to a BACnet object of type *BinaryOutput* (BO).

VAR_INPUT

```
bEnablePV        : BOOL;
bNullPV          : BOOL;
bPV              : BOOL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **bPV**; *FALSE* = write value of **bPV** as property value.

b PV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bNoOutput               : BOOL;
bOpenLoop               : BOOL;
bShortedLoop            : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nActivePrio             : UINT;
nChangeOfState          : UDINT;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

nChangeOfState: Number of changes of state (*not* to be confused with changes in value) of the property *Present_Value* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Change_Of_State_Count*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

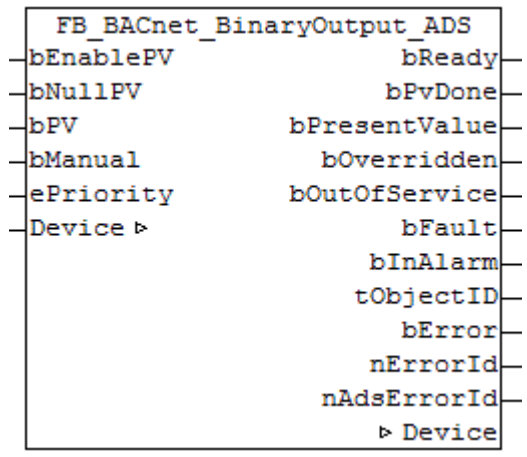
```
Device              : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.16 FB_BACnet_BinaryOutput_ADS



Application

The function block FB_BACnet_BinaryOutput_ADS can be used for read and write access to a BACnet object of type *BinaryOutput*.

In contrast to the standard and *_EX* versions, writing of the property *Present_Value* is realized with ADS-WRITE (acyclic).

VAR_INPUT

```
bEnablePV   : BOOL;
bNullPV     : BOOL;
bPV         : BOOL;
bManual     : BOOL;
ePriority   : E_BACNETPRIORITY:=BACNETPRIORITY_12;
```

bEnablePV: enables the value of input **bPV**. As soon as the input is set to *TRUE*, write access to the commandable property *Present_Value* takes place with the value from **bPV** and the priority from **ePriority** (default: 12, if the input **ePriority** is not connected). If the input is set to *FALSE*, no further write access takes place (no change).

bNullPV: overwrites the entry of the commandable property *Present_Values* in priority **ePriority** with the value *NULL*, if **bEnablePV** is set to *TRUE*. The input **bPV** is ignored, as long as **bNullPV** is set to *TRUE*.

bPV: value to be written to the corresponding location of the property *Priority_Array* of the commandable property *Present_Value* if **bEnablePV** is set to *TRUE* and **bNullPV** is set to *FALSE*. The priority is determined with the input **ePriority** (default: 12, if the input **ePriority** is not connected). If the value changes, writing takes place acyclically (ADS WRITE).

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* → *TRUE*: writing the property *Present_Value* on edge change.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 196]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 344]).

VAR_OUPUT

```
bReady          : BOOL;
bPvDone         : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
nAdsErrorId     : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPvDone: Positive edge if writing of property *Present_Value* via ADS was successful.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

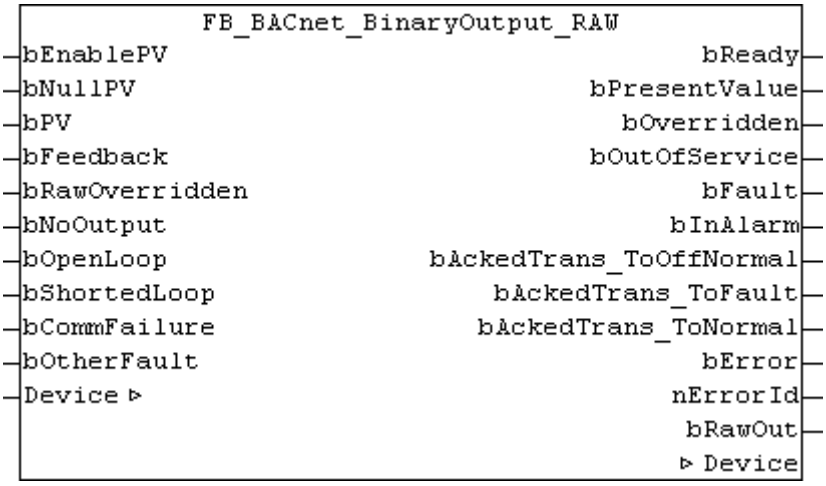
nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.17 FB_BACnet_BinaryOutput_RAW



Application

The function block FB_BACnet_BinaryOutput_RAW can be used for reading and write access to a BACnet object of type *BinaryOutput*.

In contrast to the standard or *_EX* version of the block, the flag of the property *Status_Flags* is *overridden*, the property *Feedback_Value* and the value of the property *Reliablitity* is provided by function block inputs and mapped from the PLC program to the BACnet object. The PLC program maps the state of the property *Present_Value* to the hardware and the sub-bus system. Otherwise, this is dealt with directly by the IO hardware. In this way it is possible, for example, for a BACnet object to map the output value to a sub-bus system in PLC code (signal conversion to sub-bus systems or virtual points from BACnet, see Example [▶ 189]).

VAR_INPUT

```
bEnablePV           : BOOL;
bNullPV             : BOOL;
bPV                 : BOOL;
bFeedback           : BOOL;
bRawOverridden      : BOOL;
bNoOutput           : BOOL;
bOpenLoop           : BOOL;
bShortedLoop        : BOOL;
bCommFailure        : BOOL;
bOtherFault         : BOOL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **bPV**; *FALSE* = write value of **bPV** as property value.

b PV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

bFeedback: Signal feedback to the BACnet object. See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Feedback_Value*.

bRawOverridden: *TRUE* sets the flag *overridden* of the property *Status_Flags*. This indicates to BACnet that the value of the property *Present_Value* was decoupled from the hardware output (e.g. through manual local operation). See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bNoOutput, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoOutput** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Reliability*.

VAR_OUPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
bRawOut                 : BOOL;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Reliability*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

bRawOut: Raw value output of the object. The output is linked to the process data "RawIoBinaryBoolValue" of the BACnet object. The value of **bRawOut** results from the property *Present_Value,* taking account of the property *Polarity*.

VAR_IN_OUT

```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

Example

The following example illustrates the implementation of the property *Present_Value* of a BinaryOutput object in EIB:

```
0002 VAR
0003 (* EIB *)
0004     EIB          : KL6301;
0005     EIBBinOut    : EIB_BIT_SEND;
0006
0007 (* Manual override *)
0008     diManActive AT%I*:BOOL; (* key switch on cabinet *)
0009     diManOnOff  AT%I*:BOOL; (* on/off switch on cabinet *)
0010
0011 (* BACnet *)
0012     Device       : FB_BACnet_Device;
0013     BORaw        : FB_BACnet_BinaryOutput_RAW;
0014 END_VAR
```

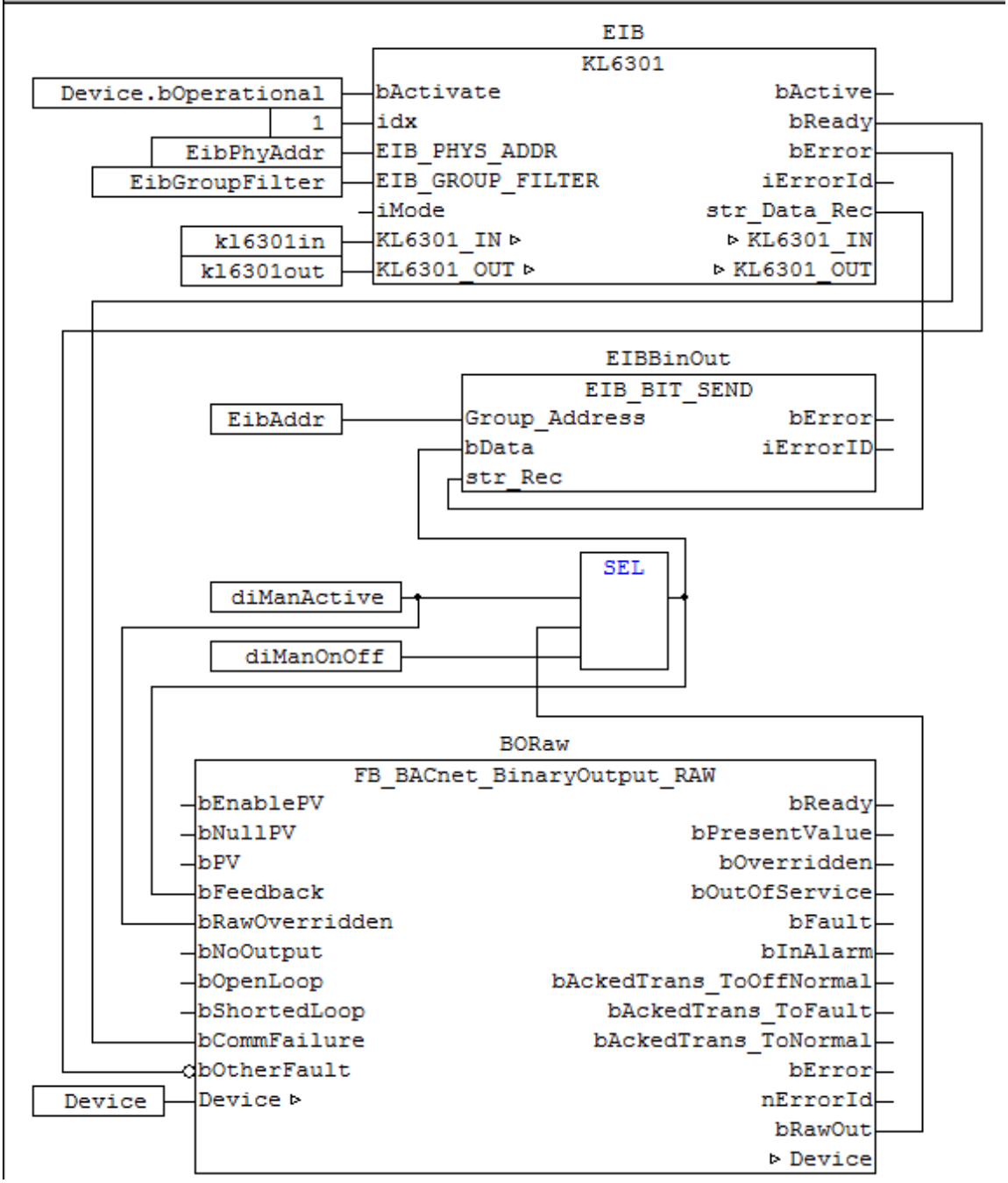
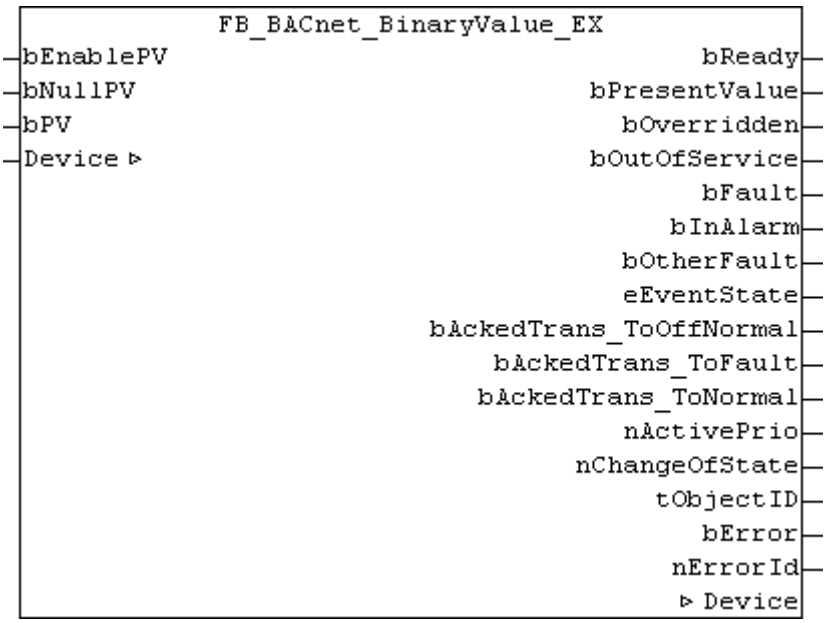


Fig. 13: Fig. 1: Example for the implementation of property Present_Value in EIB in the PLC program.

This example assumes the library "TcEIB.lib" is available. For further information on EIB please refer to the manual for the KL6301 terminal.

The implementation of local operation by means of a selector switch (digital input) and a control switch (digital input) is also illustrated.

4.2.18 FB_BACnet_BinaryValue_EX



Application

The function block FB_BACnet_BinaryValue can be used for read access with priority 12 to the property *Present_Value* and for write access to a BACnet object of type *BinaryValue* (BV).

VAR_INPUT

```
bEnablePV        : BOOL;
bNullPV          : BOOL;
bPV              : BOOL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **bPV**; *FALSE* = write value of **bPV** as property value.

b PV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nActivePrio             : UINT;
nChangeOfState          : UDINT;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

nChangeOfState: Number of changes of state (*not* to be confused with changes in value) of the property *Present_Value* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Change_Of_State_Count*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

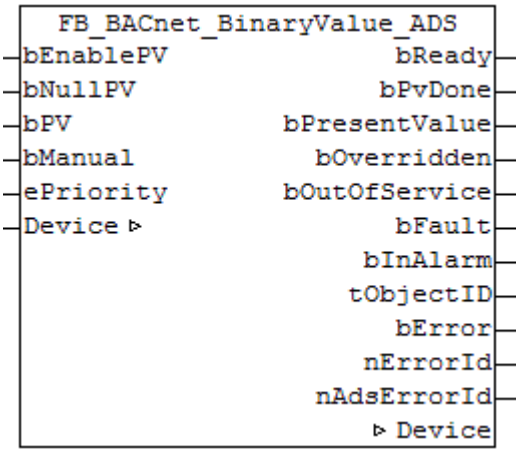
```
Device              : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.19 FB_BACnet_BinaryValue_ADS



Application

The function block FB_BACnet_BinaryValue_ADS can be used for read and write access to a BACnet object of type *BinaryValue*.

In contrast to the standard and *_EX* versions, writing of the property *Present_Value* is realized with ADS-WRITE (acyclic).

VAR_INPUT

```
bEnablePV       : BOOL;
bNullPV         : BOOL;
bPV             : BOOL;
bManual         : BOOL;
ePriority       : E_BACNETPRIORITY:=BACNETPRIORITY_12;
```

bEnablePV: enables the value of input **bPV**. As soon as the input is set to *TRUE*, write access to the commandable property *Present_Value* takes place with the value from **bPV** and the priority from **ePriority** (default: 12, if the input **ePriority** is not connected). If the input is set to *FALSE*, no further write access takes place (no change).

bNullPV: overwrites the entry of the commandable property *Present_Values* in priority **ePriority** with the value *NULL*, if **bEnablePV** is set to *TRUE*. The input **bPV** is ignored, as long as **bNullPV** is set to *TRUE*.

bPV: value to be written to the corresponding location of the property *Priority_Array* of the commandable property *Present_Value* if **bEnablePV** is set to *TRUE* and **bNullPV** is set to *FALSE*. The priority is determined with the input **ePriority** (default: 12, if the input **ePriority** is not connected). If the value changes, writing takes place acyclically (ADS WRITE).

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: the following evaluations are distinguished for the input:
- *FALSE*: write on value change
- Edge change *FALSE* → *TRUE*: write the property *Present_Value* on edge change.
- *TRUE*: write on value change *and* when the associated BACnet server changes to the "Operational" state (see FB_BACnet_Device [▶ 196]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 344]).

VAR_OUPUT

```
bReady          : BOOL;
bPvDone         : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
nAdsErrorId     : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPvDone: Positive edge if writing of property *Present_Value* via ADS was successful.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

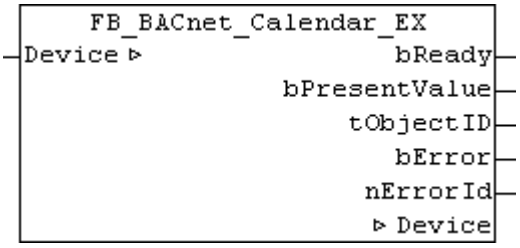
nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.20 FB_BACnet_Calendar_EX



Application

The function block FB_BACnet_Calendar can be used for reading access to a BACnet object of type *Calendar* (CAL).

VAR_OUPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Calendar* and property *Present_Value*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

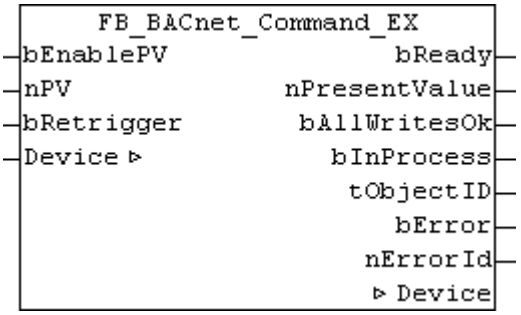
nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.21 FB_BACnet_Command_EX



Application

The function block "FB_BACnet_Command" can be used for reading and write access to a BACnet object of type *Command* (CMD).

VAR_INPUT

```
bEnablePV  : BOOL;
nPV        : UDINT;
bRetrigger : BOOL;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, writing takes place into the property *Present_Value* of the corresponding BACnet object with the value of input **nPV**.
If **bEnablePV** is set to *FALSE*, the process data of the mapped property *Present_Value* are written to *0* and thus deactivated.

nPV: Value of the property *Present_Value* to be written. If the value is outside the value range (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*), the corresponding process data is disabled, writing to the property is disabled.
Writing of a valid value triggers execution of the corresponding command list of the BACnet object. The value of the property *Present_Value* is written whenever the process data changes (i.e. change in the value of **nPV** with **bEnablePV** set, or signal change from *FALSE* to *TRUE* at input **bRetrigger** with **bEnablePV** set).

bRetrigger: A rising edge at this input triggers a repeat of the write process and therefore execution of the corresponding commands of the BACnet object *Command*. The signal change from *FALSE* to *TRUE* corresponds to a change of the process data of the property *Present_Value* from x to *0* to x.

VAR_OUPUT

```
bReady         : BOOL;
nPresentValue  : UDINT;
bAllWritesOk   : BOOL;
bInProcess     : BOOL;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError         : BOOL;
nErrorId       : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*).

bAllWritesOk: The last requested command list was successfully processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *All_Writes_Successful*).

bInProcess: The selected command list is processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *In_Process*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

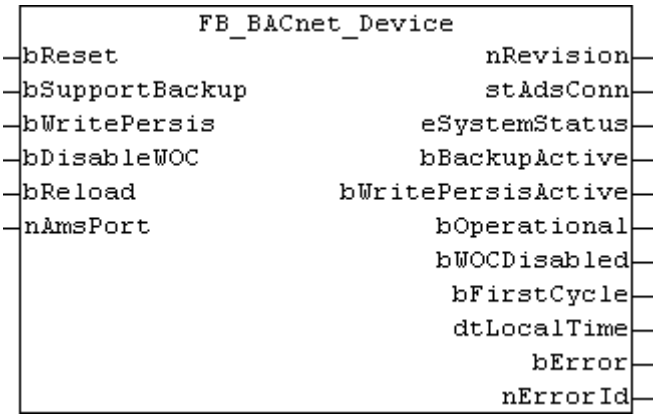
nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.22 FB_BACnet_Device



Application

Function block for interfacing of the PLC program with a local BACnet device object (server). The function block is linked with the aid of process data and ADS (evaluate AMS port, read object list, determine the stack revision and backup).

As a rule, the function block is backward compatible (revision 6). The BACnet revision in use can be read at output **nRevision**.
If revision 6 is used, some ADS services of the BACnet stack are limited; e.g. the AMS port of the BACnet device (server and client) can only be determined automatically from revision 12. If revision 6 is used, the AMS port (Ads port) shown in the TwinCAT System Manager must therefore be transferred at input **nAmsPort** (see Figure 1)!

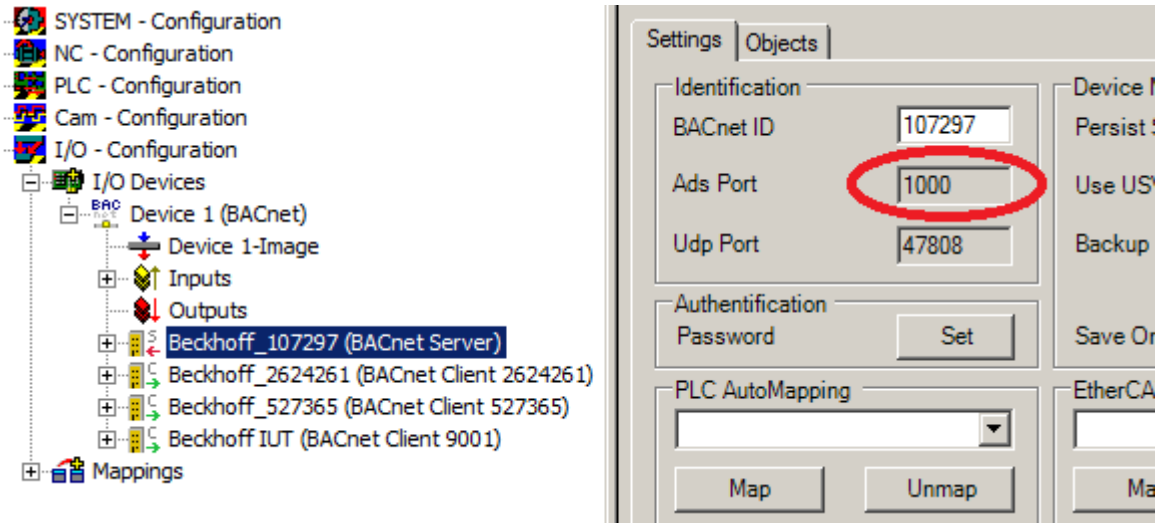


Fig. 14: Fig. 1: AMS port of the BACnet device in the TwinCAT System Manager

VAR_INPUT

Inputs are optional and do not have to be assigned if Revision 12 or higher is used.

```
bReset          : BOOL;
bSupportBackup  : BOOL;
bWritePersis    : BOOL;
bDisableWOC     : BOOL;
bReload         : BOOL;
nAmsPort        : T_AmsPort:=1000;  (*siehe Info*)
```

bReset: resets the error state in the event of a signal change *FALSE → TRUE*.

bSupportBackup: if the input is set to *TRUE*, saving of the persistent PLC runtime data via BACnet is supported. If this mode is activated, the persistent files of the PLC (by default under: "C:\TwinCAT\Boot\") must be created as BACnet file objects and provided with the value "TwinCAT Configuration File" of the property *File_Type* (thus the BACnet stack takes care of the backup and transfer of the files to the backing up remote system).
When using the BACnet-side backup of the persistent PLC data, no further instance of the function block *FB_WritePersistentData* has to be used in the PLC runtime (*FB_WritePersistentData* is executed by FB_BACnet_Device) - The input **bWritePersis** triggers the writing of the persistent data independently of the BACnet backup.

bWritePersis: signal change *FALSE → TRUE* triggers writing of the persistent PLC data, independent from the BACnet backup. Within the function block, the call is triggered by *FB_WritePersistentData*.

bDisableWOC: state TRUE at the input disables writing of process data to the BACnet objects of the respective BACnet device. All property write access is blocked.

bReload: the ADS connection is renewed. Subsequent function blocks, which use the ADS connection, are also triggered.

nAmsPort: *only if revision 6 is used, otherwise do not use:* AMS port of the local BACnet server. The default number is 1000 (provided the BACnet server is the first element under the BACnet adapter in the TwinCAT System Manager). The port number can be read in the System Manager (see Figure 1).

VAR_OUPUT

```
nRevision           : DINT:=-1;
stAdsConn           : ST_BACnet_AdsConnection;
eSystemStatus       : E_BACNETDEVICESTATUS;
bBackupActive       : BOOL;
bWritePersisActive  : BOOL;
bOperational        : BOOL;
bWOCDisabled        : BOOL;
bFirstCycle         : BOOL;
dtLocalTime         : ST_BACnet_DateTime;
bError              : BOOL;
nErrorId            : UINT;
```

nRevision: Output of the BACnet revision number in use.

stAdsConn: Structure with the ADS connection data (from Revision 12 the included AMS port is determined automatically).

eSystemStatus: Status of the BACnet server object (see BACnet specification DIN EN ISO 16484-5 for BACnet device object and property *System_Status*).

bBackupActive: Feedback to indicate that a backup via BACnet is executed if the output is set to *TRUE*.

bWritePersisActive: Writing of the persistent PLC data is executed. Writing of the persistent PLC data can be triggered by BACnet backup or by setting the input **bWritePersis**.

bOperational: Device adapter is no longer in *Init* state. If the output falls to *FALSE,* all linked BACnet object function blocks in the PLC program are blocked.

bWOCDisabled: Feedback to input **bDisableWOC**.

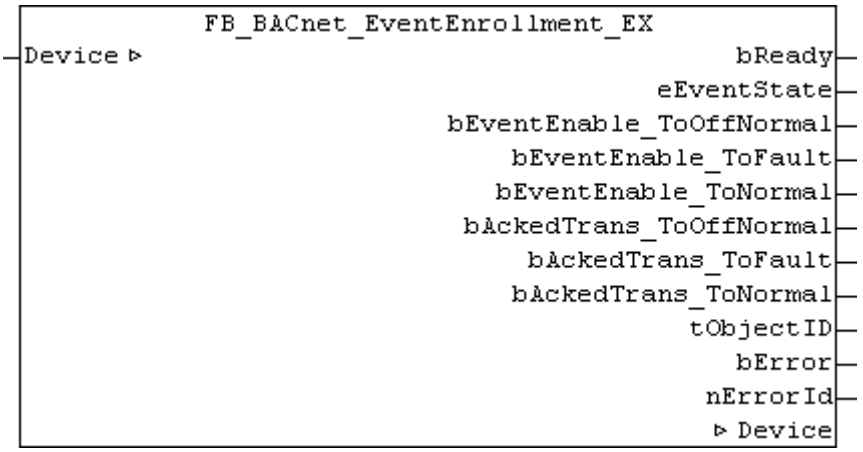
bFirstCycle: Output is set to *TRUE* when the function block instance is called for the first time (PLC start). Subsequently the output remains on *FALSE*.

dtLocalTime: Current BACnet time (see properties *Local_Date* and *Local_Time* of the BACnet device object). The function block FB_BACnet_TimeSync [▶ 270] is available for time synchronization between operating systems, BACnet stack and BACnet network.

bError: An error is pending.

nErrorId: see global constants (BACnet_Globals [▶ 328]).

4.2.23 FB_BACnet_EventEnrollment_EX



Application

The function block FB_BACnet_EventEnrollment can be used for reading access to a BACnet object of type *EventEnrollment* (EE).

VAR_OUPUT

```
bReady                    : BOOL;
eEventState               : E_BACNETEVENTSTATE;
bEventEnable_ToOffNormal  : BOOL;  (*siehe Anmerkung*)
bEventEnable_ToFault      : BOOL;  (*siehe Anmerkung *)
bEventEnable_ToNormal     : BOOL;  (*siehe Anmerkung *)
bAckedTrans_ToOffNormal   : BOOL;
bAckedTrans_ToFault       : BOOL;
bAckedTrans_ToNormal      : BOOL;
tObjectID                 : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                    : BOOL;
nErrorId                  : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the function block instance was not linked correctly in the TwinCAT System Manager.

eEventState: see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Event_State*.

bEventEnable_ToOffNormal, bEventEnable_ToFault, bEventEnable_ToNormal: Flags [▶ 358] of the property *Event_Enable* (see also BACnet specification DIN EN ISO 16484-5 for the BACnet object *BinaryValue* and property *Event_Enable*).

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags [▶ 358] of the property *Acked_Transitions* (see also BACnet specification DIN EN ISO 16484-5 for the BACnet object *BinaryValue* and property *Acked_Transitions*).

tObjectID: object ID of the BACnet object (object type and object instance).

bError: an error is pending.

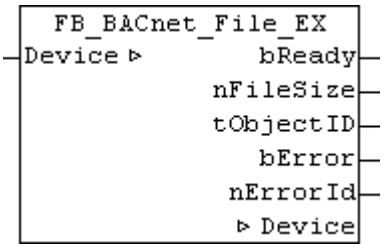
nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

```
Device            : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.24 FB_BACnet_File_EX



Application

The function block FB_BACnet_File can be used for reading access to a BACnet object of type *File* (FILE).

VAR_OUPUT

```
bReady          : BOOL;
nFileSize       : UDINT;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the remaining status outputs are valid (FileSize and ObjectID). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nFileSize: Current file size in bytes (see also BACnet specification DIN EN ISO 16484-5 re. BACnet object *File* and property *File_Size*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

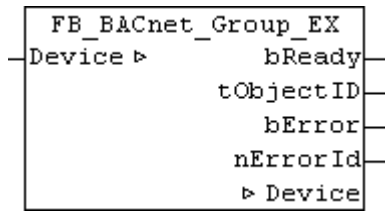
nErrorId: see global constants (BACnet_Globals [▶ 328]).

VAR_IN_OUT

```
Device      : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.25 FB_BACnet_Group_EX



Application

The function block FB_BACnet_Group_EX can be used for reading and write access to a BACnet object of type *Group*.

VAR_OUPUT

```
bReady     : BOOL;
tObjectID  : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError     : BOOL;
nErrorId   : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

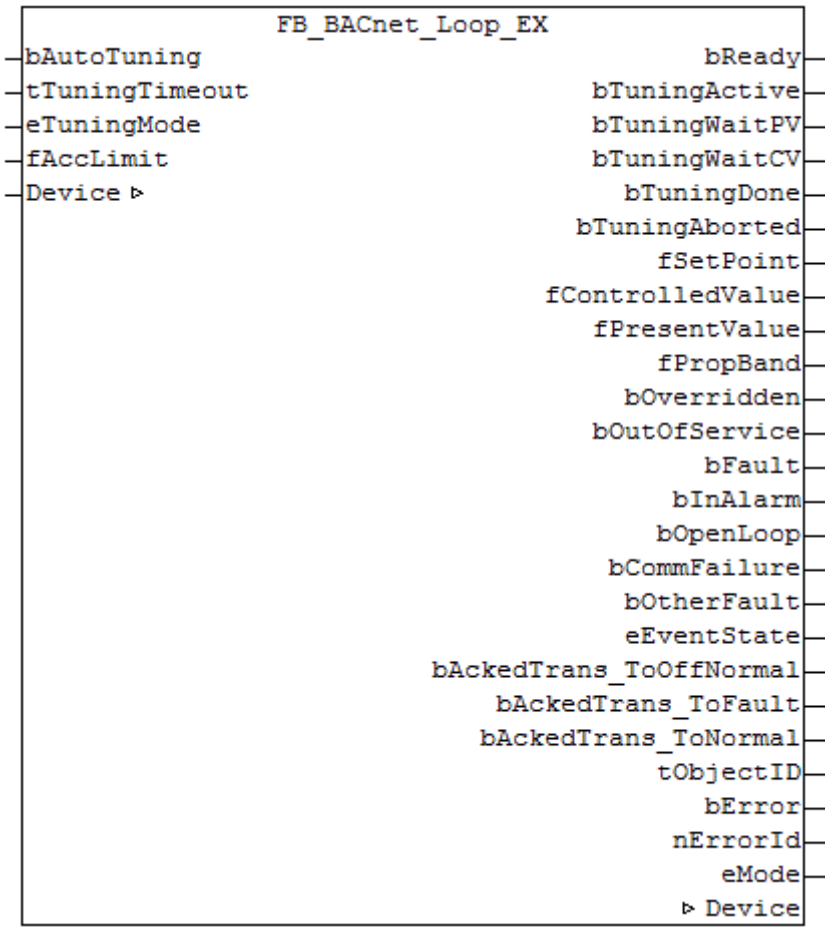
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device     : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.26 FB_BACnet_Loop_EX



Application

The function block FB_BACnet_Loop_EX can be used for reading and write access to a BACnet object of type *Loop*. The function block contains a standard PID controller (FB_BACnet_PidControl [▶ 303]) and therefore covers a wide range of control tasks.

VAR_INPUT

```
bAutoTuning      : BOOL;
tTuningTimeout   : TIME:=T#15m;
eTuningMode      : E_BACNETPIDTUNINGMODE:=BACnetPIDTuningMode_PI;
fAccLimit        : REAL:=0.0;
```

bAutoTuning: Starts internal auto-tuning.

tTuningTimeout: Maximum duration of auto-tuning in [ms].

eTuningMode: Preselection of the control parameters to be determined (P, PI, PD, PID).

fAccLimit: Parameters for limiting the control value acceleration (ramp) *0* = unlimited or [1/s] (positive and negative).

VAR_OUPUT

```
bReady                   : BOOL;
bTuningActive            : BOOL;
bTuningWaitPV            : BOOL;
bTuningWaitCV            : BOOL;
bTuningDone              : BOOL;
bTuningAborted           : BOOL;
fSetPoint                : REAL;
fControlledValue         : REAL;
fPresentValue            : REAL;
fPropBand                : REAL;
```

```
bOverridden              : BOOL;
bOutOfService            : BOOL;
bFault                   : BOOL;
bInAlarm                 : BOOL;
bOpenLoop                : BOOL;
bCommFailure             : BOOL;
bOtherFault              : BOOL;
eEventState              : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal  : BOOL;
bAckedTrans_ToFault      : BOOL;
bAckedTrans_ToNormal     : BOOL;
tObjectID                : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                   : BOOL;
nErrorId                 : UINT;;
eMode                    : E_BACNETLOOPMODE;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bTuningActive: Auto-tuning is active.

bTuningWaitPV: Auto-tuning expects the specified set value jump to be reached.

bTuningWaitCV: Auto-tuning expects the actual value response to be reached.

bTuningAborted: Auto-tuning was aborted (see **nErrorId**).

fSetPoint: Feedback of the controller specification (W, set value).

fControlledValue: Feedback of the current process parameter (X, actual value).

fPresentValue: Feedback of the current control output (Y, control value). Attention: *Present_Value* and *Controller_Variable_Value* can easily lead to confusion (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and properties *Present_Value*, *Controlled_Variable_Value* and *Controlled_Variable_Reference*).

fPropBand: Feedback of the current control output in percent (-100%...+100%) in relation to the minimum and maximum control output (properties *Minimum_Output* and *Maximum_Output*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Status_Flags*.

bOpenLoop, bCommFailure, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

eMode: Output of controller mode. A distinction is made between two modes: 1. Standard form and 2. Ideal form (see E_BACNETLOOPMODE [▶ 340]). The mode is detected automatically based on the parameter unit set for parameter I and/or D as "*Time_seconds*" → Standard form. If they are configured as factors "*Other_no_units* ", the mode "Ideal form" is detected. The Ideal form is a pure parallel connection of the 3 control variables (P, I and D). The Ideal form enables practice-oriented controller configuration since the parameters can be optimized without direct dependency. Control parameters can be deduced directly from the path behavior, for example.

VAR_IN_OUT

```
Device               : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Controller configuration

The controller is configured using the following BACnet properties: *Action*, *Proportional_Constant* (P-factor), *Integral_Constant* (I-factor), *Derivative_Constant* (D-factor), *Bias* (output offset), *Maximum_Output* (maximum control output) and *Minimum_Output* (minimum control output). See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop*.

Determining the control parameters in Ideal form mode, using temperature control as an example:

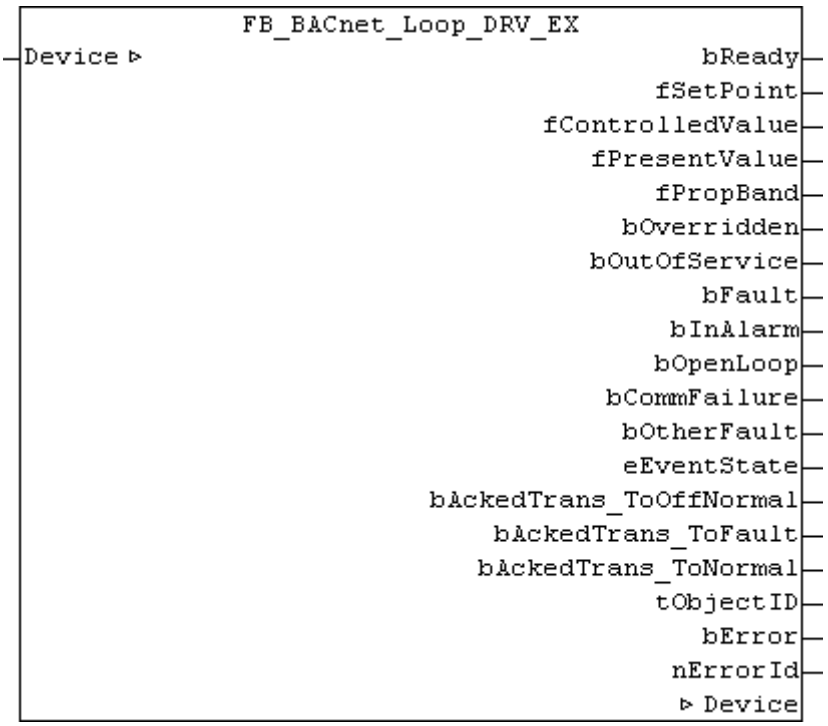
- Determination of the P-factor; assumption: At a temperature difference of 2 Kelvin, the heating output should be 10% → **Kp** = 5 = 10% / 2K;
- Determination of the I-factor; assumption: At a constant temperature difference of 1 Kelvin, the heat output should increase or decrease by 1% per minute → **Ki** = 0.17 = 1% / (60s * 1K).

Usually, the Standard form is used. To determine the control parameters in Standard form, the first step usually involves analyzing the step response of the path, followed in the second step by the control behavior for the purpose of optimization.

Also see about this

- E_BACNETPIDTUNINGMODE [▶ 343]
- E_BACNETEVENTSTATE [▶ 337]

4.2.27 FB_BACnet_Loop_DRV_EX



Application

The function block FB_BACnet_Loop_DRV_EX can be used for reading access to a BACnet object of type *Loop*. In contrast to the standard or *_EX* version of the function block, the control algorithm is implemented within the BACnet stack (see also FB_BACnet_Loop_EX [▶ 201]). The *Loop* object is therefore in control mode, even without an executable PLC program.

VAR_OUPUT

```
bReady                  : BOOL;
fSetPoint               : REAL;
fControlledValue        : REAL;
fPresentValue           : REAL;
fPropBand               : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bOpenLoop               : BOOL;
bCommFailure            : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fSetPoint: Feedback of the controller specification (W, set value).

fControlledValue: Feedback of the current process parameter (X, actual value).

fPresentValue: Feedback of the current control output (Y, control value). Attention: *Present_Value* and *Controller_Variable_Value* can easily lead to confusion (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and properties *Present_Value*, *Controlled_Variable_Value* and *Controlled_Variable_Reference*).

fPropBand: Feedback of the current control output in percent (-100%...+100%) in relation to the minimum and maximum control output (properties *Minimum_Output* and *Maximum_Output*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Status_Flags*.

bOpenLoop, bCommFailure, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet Globals [▶ 328].

VAR_IN_OUT

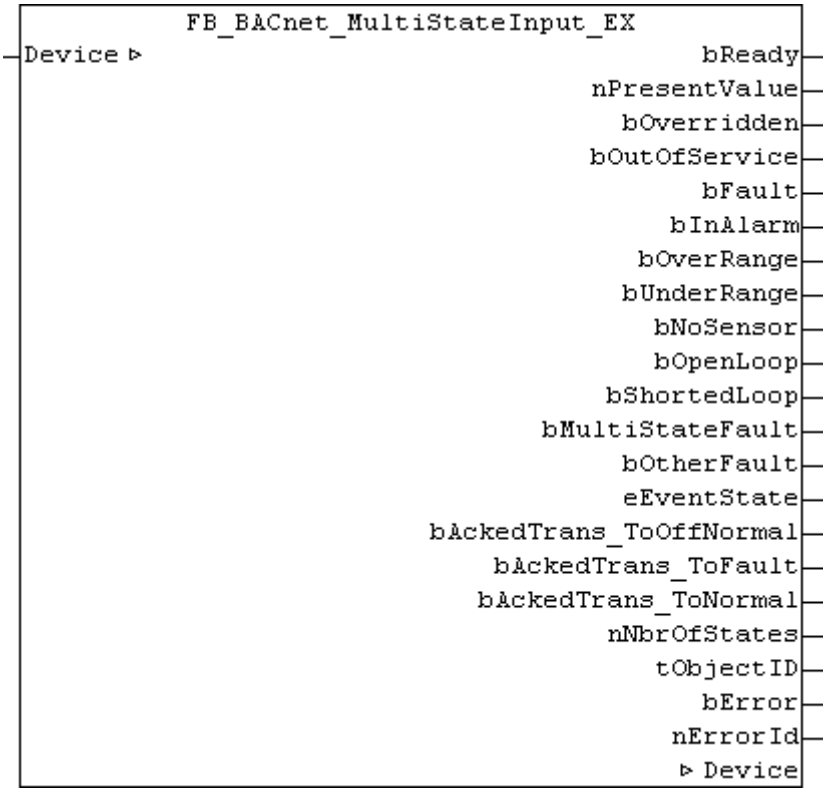
```
Device              : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.28 FB_BACnet_MultiStateInput_EX



Application

The function block FB_BACnet_MultiStateInput_EX can be used for read and write access to a BACnet object of type *MultiStateInput*.

VAR_OUPUT

```
bReady                  : BOOL;
nPresentValue           : UDINT;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bOverRange              : BOOL;
bUnderRange             : BOOL;
bNoSensor               : BOOL;
bOpenLoop               : BOOL;
bShortedLoop            : BOOL;
bMultiStateFault        : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nNbrOfStates            : UDINT;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bMultiStateFault, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Reliability*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Acked_Transitions*).

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

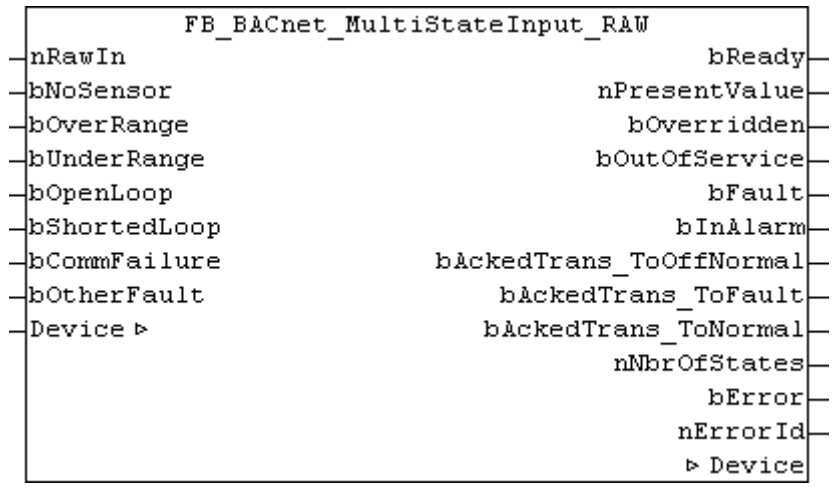
```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.29 FB_BACnet_MultiStateInput_RAW



Application

The function block FB_BACnet_MultiStateInput_RAW can be used for reading and write access to a BACnet object of type *MultiStateInput*.

In contrast to the standard and *_EX* versions of the block, the raw value, and the state of the property *Reliablitity* are provided by function block inputs, not directly by the IO hardware. For example, a multi-stage state can be mapped from a sub-bus system in PLC code to a BACnet object (signal conversion from sub-bus systems or virtual data points to BACnet, see Example [▶ 207]).

VAR_INPUT

```
nRawIn                : DWORD;
bNoSensor             : BOOL;
bOverRange            : BOOL;
```

```
bUnderRange          : BOOL;
bOpenLoop            : BOOL;
bShortedLoop         : BOOL;
bCommFailure         : BOOL;
bOtherFault          : BOOL;
```

nRawIn: Raw value input of the object in the range 1 to *Number_Of_States*. The input is linked to the process record "PresentValue" of the BACnet object. The value of **nRawIn** is written directly to the property *Present_Value* (provided the object state is not *out_of_service*).

bNoSensor, bOverRange, bUnderRange, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoSensor** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Reliability*.

VAR_OUPUT

```
bReady                   : BOOL;
nPresentValue            : UDINT;
bOverridden              : BOOL;
bOutOfService            : BOOL;
bFault                   : BOOL;
bInAlarm                 : BOOL;
bAckedTrans_ToOffNormal  : BOOL;
bAckedTrans_ToFault      : BOOL;
bAckedTrans_ToNormal     : BOOL;
nNbrOfStates             : UDINT;
bError                   : BOOL;
nErrorId                 : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Acked_Transitions*).

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device               : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

Example

The following example shows the implementation of an integer value from EIB in the property *Present_Value* of an MultiStateInput object:

```
0002 VAR
0003 (* EIB *)
0004     EIB          : KL6301;
0005     EIB8BitIn    : EIB_8BIT_UNSIGN_REC;
0006
0007 (* BACnet *)
0008     Device       : FB_BACnet_Device;
0009     MIRaw        : FB_BACnet_MultiStateInput_RAW;
0010 END_VAR
```

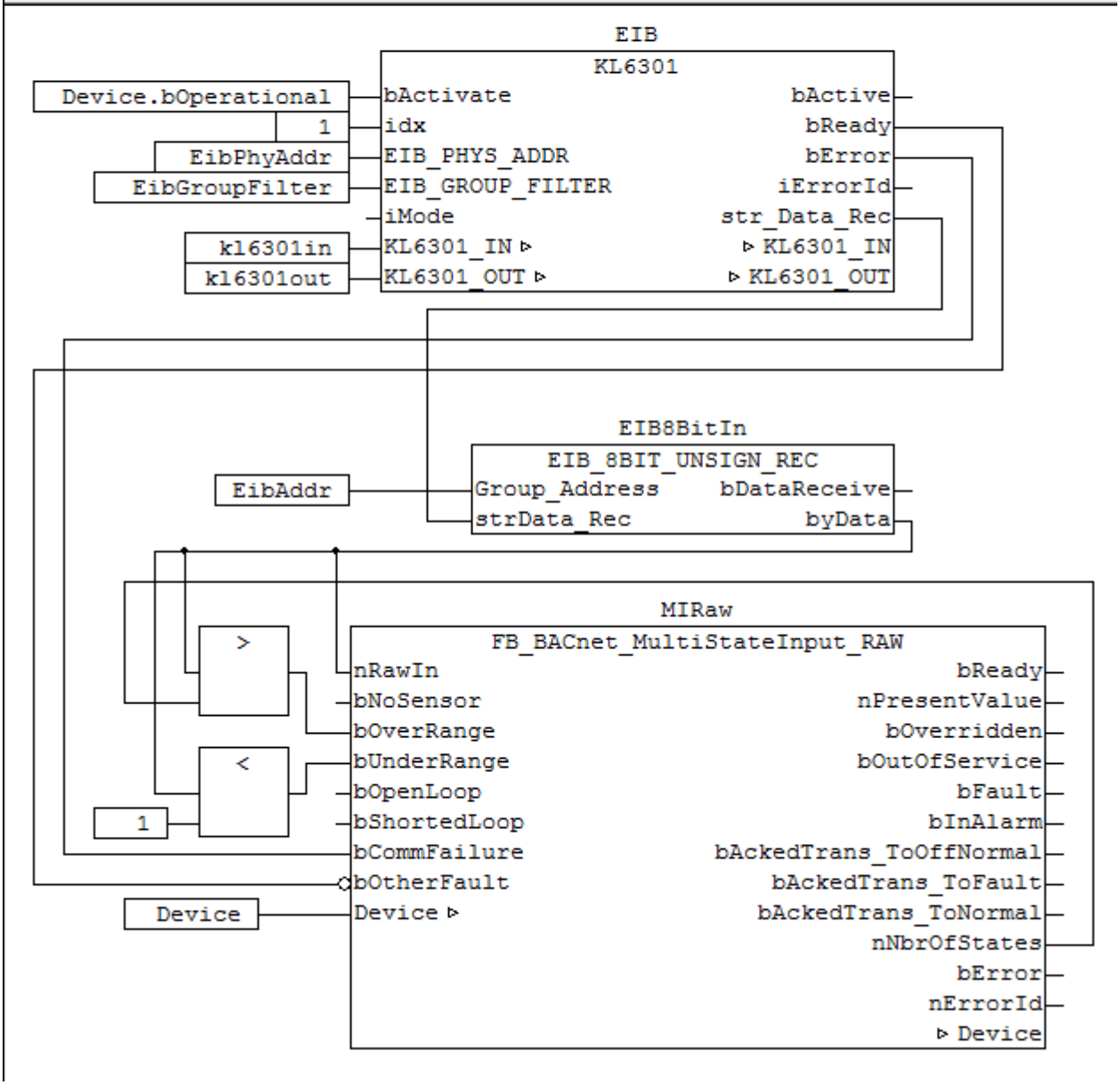
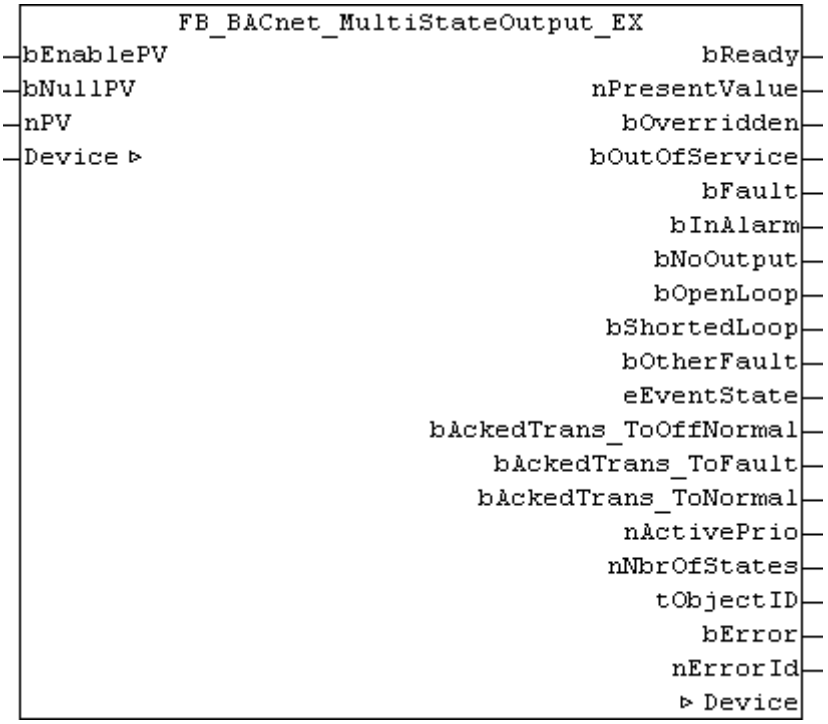


Fig. 15: Fig. 1: Example for the implementation of an integer value from EIB in the property Present_Value in the PLC program.

This example assumes the library "TcEIB.lib" is available. For further information on EIB please refer to the manual for the KL6301 terminal.

4.2.30 FB_BACnet_MultiStateOutput_EX



Application

The function block FB_BACnet_MultiStateOutput_EX can be used for reading and write access to a BACnet object of type *MultiStateOutput*.

VAR_INPUT

```
bEnablePV        : BOOL;
bNullPV          : BOOL;
nPV              : UDINT;
```

bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **nState** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g. for deleting a priority), *FALSE* = write value from **nState**

nPV: Value that is written to the BACnet object, if **bEnable** = *TRUE* and **bNull** = *FALSE*. Multi-State in the range [1 .. *Number_Of_States*].

VAR_OUPUT

```
bReady                  : BOOL;
nPresentValue           : UDINT;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bNoOutput               : BOOL;
bOpenLoop               : BOOL;
bShortedLoop            : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nActivePrio             : UINT;
nNbrOfStates            : UDINT;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

bNoOutput, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Reliability*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_Value*).

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If **nNbrOfStates** is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

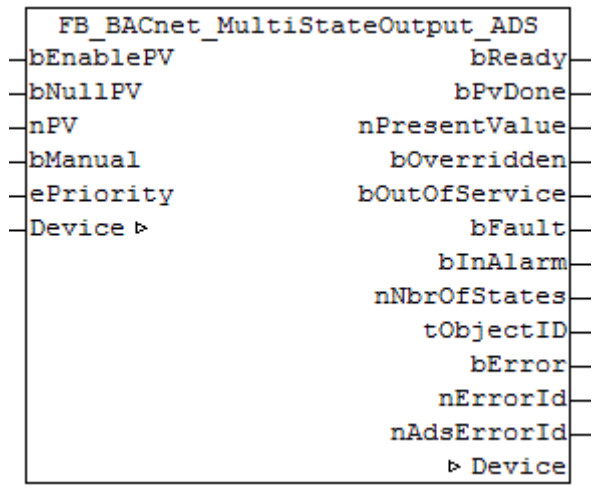
```
Device              : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.31 FB_BACnet_MultiStateOutput_ADS



Application

The function block FB_BACnet_MultiStateOutput_ADS can be used for read and write access to a BACnet object of type *MultiStateOutput*.

In contrast to the standard and *_EX* versions, writing of the property *Present_Value* is realized with ADS-WRITE (acyclic).

VAR_INPUT

```
bEnablePV   : BOOL;
bNullPV     : BOOL;
nPV         : UDINT;
bManual     : BOOL;
ePriority   : E_BACNETPRIORITY:=BACNETPRIORITY_12;
```

bEnablePV: enables the value of input **nPV**. As soon as the input is set to *TRUE*, write access to the commandable property *Present_Value* takes place with the value from **nPV** and the priority from **ePriority** (default: 12, if the input **ePriority** is not connected). If the input is set to *FALSE*, no further write access takes place (no change).

bNullPV: overwrites the entry of the commandable property *Present_Values* in priority **ePriority** with the value *NULL*, if **bEnablePV** is set to *TRUE*. The input **nPV** is ignored, as long as **bNullPV** is set to *TRUE*.

nPV: value to be written to the corresponding location of the property *Priority_Array* of the commandable property *Present_Value* if **bEnablePV** is set to *TRUE* and **bNullPV** is set to *FALSE*. The priority is determined with the input **ePriority** (default: 12, if the input **ePriority** is not connected). If the value changes, writing takes place acyclically (ADS WRITE).

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* → *TRUE*: writing the property *Present_Value* on edge change.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 196]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 344]).

VAR_OUPUT

```
bReady          : BOOL;
bPvDone         : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
nNbrOfStates    : UDINT;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
nAdsErrorId     : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPvDone: Positive edge if writing of property *Present_Value* via ADS was successful.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

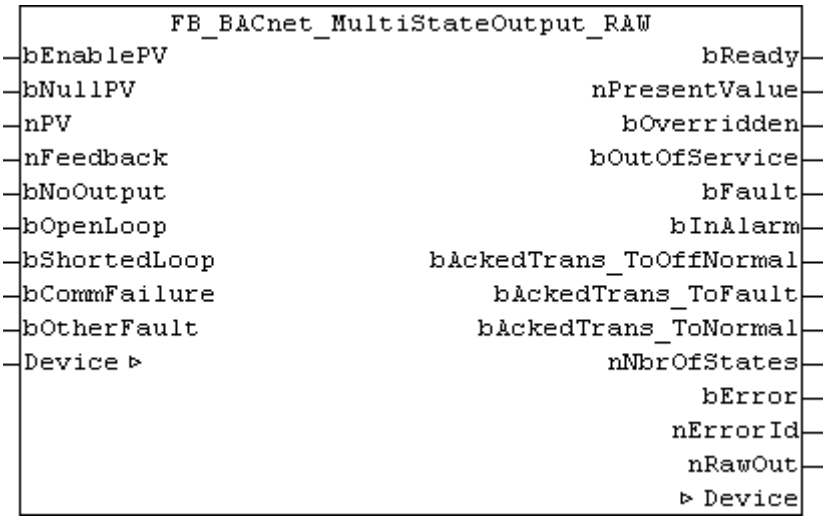
nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.32 FB_BACnet_MultiStateOutput_RAW



Application

The function block FB_BACnet_MultiStateOutput_RAW can be used for reading and write access to a BACnet object of type *MultiStateOutput*.

In contrast to the standard or *_EX* version of the block, the flag of the property *Status_Flags* is *overridden*, the property *Feedback_Value* and the value of the property *Reliablitity* is provided by function block inputs and mapped from the PLC program to the BACnet object. The PLC program maps the state of the property *Present_Value* to the hardware and the sub-bus system. Otherwise, this is dealt with directly by the IO hardware. In this way it is possible, for example, for a BACnet object to map the output value to a sub-bus system in PLC code (signal conversion to sub-bus systems or virtual points from BACnet, see Example [▶ 213]).

VAR_INPUT

```
bEnablePV           : BOOL;
bNullPV             : BOOL;
nPV                 : UDINT;
nFeedback           : DWORD;
bNoOutput           : BOOL;
bOpenLoop           : BOOL;
bShortedLoop        : BOOL;
bCommFailure        : BOOL;
bOtherFault         : BOOL;
```

bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **nState** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g., for deleting a priority), *FALSE* = write value from **nState**

nPV: Value that is written to the BACnet object, if **bEnable** = *TRUE* and **bNull** = *FALSE*. Multi-State in the range [1 .. *Number_Of_States*].

nFeedback: Signal feedback to the BACnet object. See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Feedback_Value*.

bNoOutput, bOpenLoop, bShortedLoop, bCommFailure, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoOutput** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Reliability*.

VAR_OUPUT

```
bReady                  : BOOL;
nPresentValue           : UDINT;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nNbrOfStates            : UDINT;
bError                  : BOOL;
nErrorId                : UINT;
nRawOut                 : DWORD;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Acked_Transitions*).

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If **nNbrOfStates** is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: see global constants BACnet Globals [▶ 328].

nRawOut: Raw value output of the object. The output is linked to the process data "PresentValue" of the BACnet object.

VAR_IN_OUT

```
Device              : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Example

The following example illustrates the implementation of the property *Present_Value* of an MultiStateOutput object in EIB:

```
0002 VAR
0003 (* EIB *)
0004     EIB          : KL6301;
0005     EIB8BitOut   : EIB_8BIT_UNSIGN_SEND;
0006
0007 (* BACnet *)
0008     Device       : FB_BACnet_Device;
0009     MORaw        : FB_BACnet_MultiStateOutput_RAW;
0010 END_VAR
```

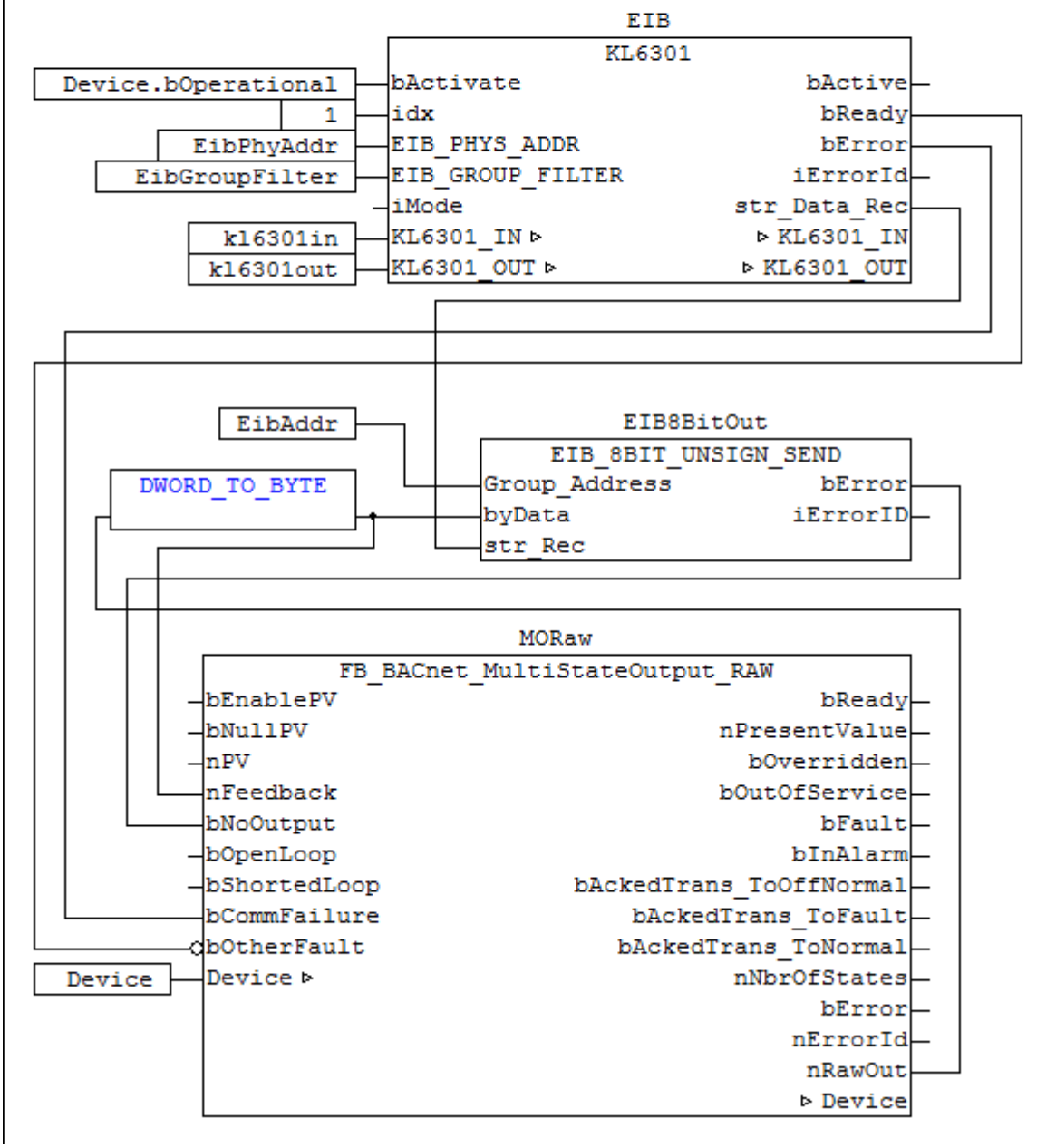
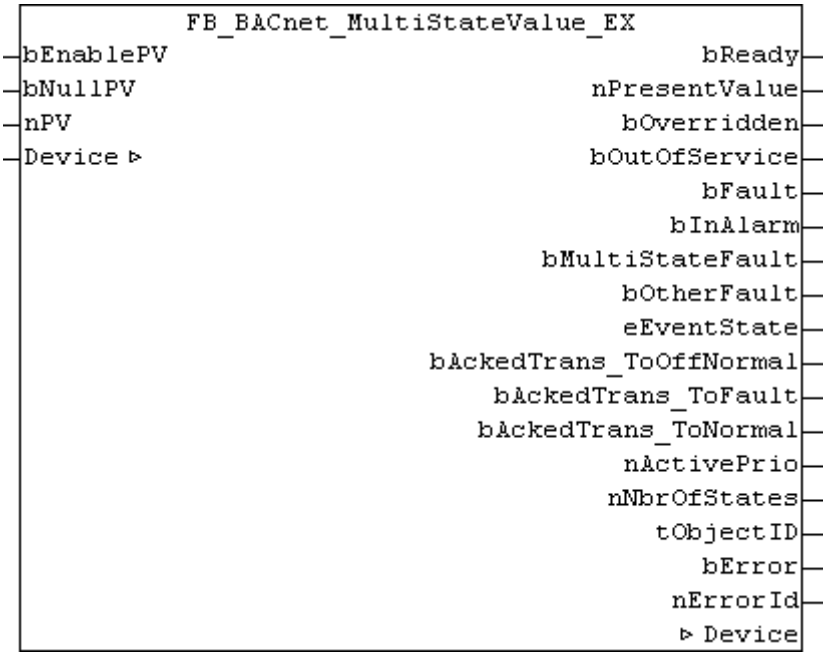


Fig. 16: Fig. 1: Example for the implementation of property Present_Value in EIB in the PLC program.

This example assumes the library "TcEIB.lib" is available. For further information on EIB please refer to the manual for the KL6301 terminal.

4.2.33 FB_BACnet_MultiStateValue_EX



Application

The function block FB_BACnet_MultiStateValue_EX can be used for reading and write access to a BACnet object of type *MultiStateValue*.

VAR_INPUT

```
bEnablePV       : BOOL;
bNullPV         : BOOL;
nPV             : UDINT;
```

bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **nState** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g. for deleting a priority), *FALSE* = write value from **nState**

nState: Value that is written to the BACnet object, if **bEnable** = *TRUE* and **bNull** = *FALSE*. Multi-State in the range [1 .. *Max_Number_Of_States*].

VAR_OUPUT

```
bReady                  : BOOL;
nPresentValue           : UDINT;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bMultiStateFault        : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
nActivePrio             : UINT;
nNbrOfStates            : UDINT;
tObjectID               : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Status_Flags*.

bMultiStateFault, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Reliability*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Acked_Transitions*).

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateValue object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

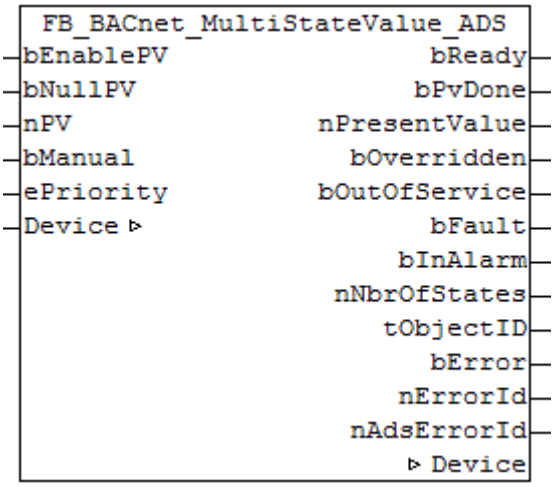
```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.34 FB_BACnet_MultiStateValue_ADS



Application

The function block FB_BACnet_MultiStateValue_ADS can be used for reading and write access to a BACnet object of type *MultiStateValue*.

In contrast to the standard and *_EX* versions, writing of the property *Present_Value* is realized with ADS-WRITE (acyclic).

VAR_INPUT

```
bEnablePV   : BOOL;
bNullPV     : BOOL;
nPV         : UDINT;
bManual     : BOOL;
ePriority   : E_BACNETPRIORITY:=BACNETPRIORITY_12;
```

bEnablePV: enables the value of input **nPV**. As soon as the input is set to *TRUE*, write access to the commandable property *Present_Value* takes place with the value from **nPV** and the priority from **ePriority** (default: 12, if the input **ePriority** is not connected). If the input is set to *FALSE*, no further write access takes place (no change).

bNullPV: overwrites the entry of the commandable property *Present_Values* in priority **ePriority** with the value *NULL*, if **bEnablePV** is set to *TRUE*. The input **nPV** is ignored, as long as **bNullPV** is set to *TRUE*.

nPV: value to be written to the corresponding location of the property *Priority_Array* of the commandable property *Present_Value* if **bEnablePV** is set to *TRUE* and **bNullPV** is set to *FALSE*. The priority is determined with the input **ePriority** (default: 12, if the input **ePriority** is not connected). If the value changes, writing takes place acyclically (ADS WRITE).

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* → *TRUE*: writing the property *Present_Value* on edge change.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 196]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 344]).

VAR_OUPUT

```
bReady          : BOOL;
bPvDone         : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
nNbrOfStates    : UDINT;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
nAdsErrorId     : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPvDone: Positive edge if writing of property *Present_Value* via ADS was successful.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

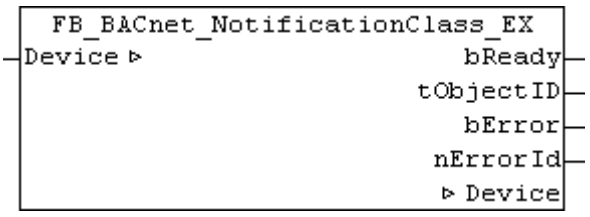
nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.35 FB_BACnet_NotificationClass_EX



Application

The function block FB_BACnet_NotificationClass_EX can be used for reading and write access to a BACnet object of type *NotificationClass*.

VAR_OUPUT

```
bReady      : BOOL;
tObjectID   : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError      : BOOL;
nErrorId    : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

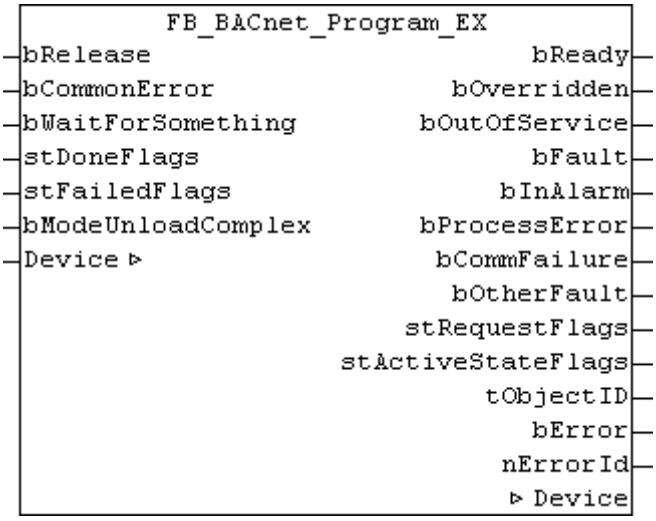
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device      : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.36 FB_BACnet_Program_EX



Application

The function block FB_BACnet_Program_EX can be used for reading and write access to a BACnet object of type *Program*.

The function block FB_BACnet_Program contains the *State Transitions* described in the BACnet specification and maps them to the FB interface using the structures: *stDoneFlags*, *stFailedFlags* and *stRequestFlags* (see Transition diagram [▶ 221] Fig. 1).

VAR_INPUT

```
bRelease            : BOOL;
bCommonError        : BOOL;
bWaitForSomething   : BOOL;
stDoneFlags         : ST_BACnet_ProgramHandshakeRequests;
stFailedFlags       : ST_BACnet_ProgramHandshakeRequests;
bModeUnloadComplex  : BOOL:=TRUE;
```

bRelease: Enables processing of requests. If **bRelease** = *FALSE* is set, the program waits in state *HALTED*.

bCommonError: Error during execution of a *Program_Change* requests from the structure **stRequestFlags**. Setting the input while a *Program_Change* requests is running leads to an abort condition (see Transition diagram [▶ 221] Fig. 1).

bWaitForSomething: If the BACnet object is in state *RUNNING*, setting the input to *TRUE* switches to state *WAITING*, so that the BACnet object is blocked for further requests except *Unload* and *Halt* (see Transition diagram [▶ 221] Fig. 1).
To return to state *RUNNING* set the input back to *FALSE*.

stDoneFlags: The flags within the input structure are used to acknowledge pending *Program_Change* requests from the output structure **stRequestFlags**. A signal change of one of the flags from *FALSE* to *TRUE* acknowledges execution of the corresponding *Program_Change* request.

stFailedFlags: The flags within the input structure are used to report cancellation of the pending *Program_Change* request from the output structure **stRequestFlags**. A signal change of one of the flags from *FALSE* to *TRUE* cancels execution of the corresponding *Program_Change* request.

bModeUnloadComplex: If the mode "Unload Complex" was activated via the input, the BACnet object status changes from *UNLOADING* to *IDLE* once the request *Unload* has been executed (see Transition diagram [▶ 221] Fig. 1). If the input is set to *FALSE*, the BACnet object remains in state *UNLOADING* after the request *Unload* has been executed.

VAR_OUPUT

```
bReady              : BOOL;
bOverridden         : BOOL;
bOutOfService       : BOOL;
bFault              : BOOL;
bInAlarm            : BOOL;
bProcessError       : BOOL;
bCommFailure        : BOOL;
bOtherFault         : BOOL;
stRequestFlags      : ST_BACnet_ProgramHandshakeRequests;
stActiveStateFlags  : ST_BACnet_ProgramHandshakeStates;
tObjectID           : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError              : BOOL;
nErrorId            : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Status_Flags*.

bProcessError, bCommFailure, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Reliability*.

stRequestFlags: Contains the flags for the corresponding *Program_Change* requests. If a flag is set to *TRUE*, a change in status of the program object is requested by the PLC program. If the change was successful, this is acknowledged with a signal change from *FALSE* to *TRUE* of the corresponding flags in the input structure **stDoneFlags**. If the corresponding PLC program is unable to execute the request or if the process is aborted, this must be reported via the corresponding flag in the input structure **stFailedFlags**. The termination is then shown in the status of the BACnet object.

stActiveStateFlags: Contains flags that reflect the current state of the BACnet program object (see Transition diagram [▶ 221] Fig. 1).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Transition diagram

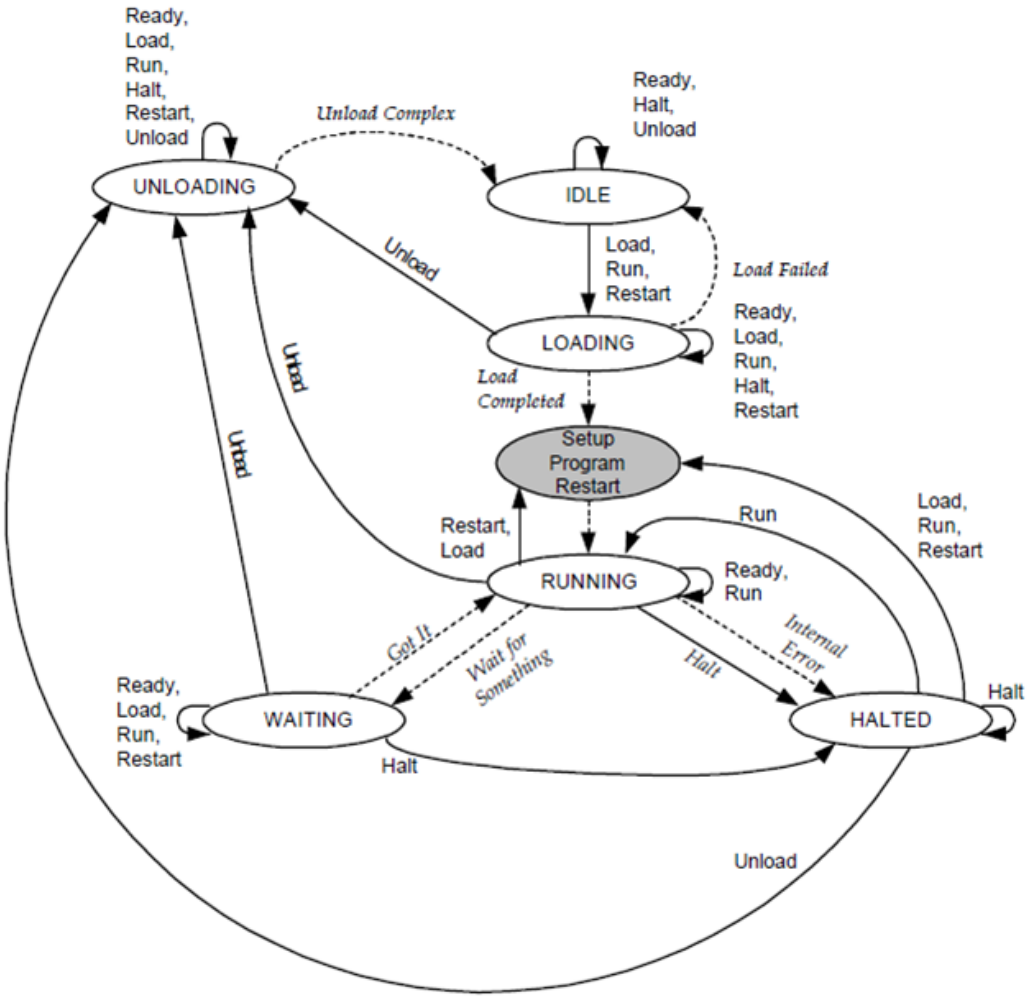


Fig. 17: Fig. 1: from BACnet specification DIN EN ISO 16484-5 for BACnet object Program, Fig. 12-3 "State Transitions for the program object"

4.2.37 FB_BACnet_PulseConverter_EX

Application

The function block FB_BACnet_PulseConverter_EX can be used for read access to a BACnet object of type *PulseConverter*.

VAR_OUPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bHighLimitEn            : BOOL;
fHighLimit              : REAL;
bLowLimitEn             : BOOL;
fLowLimit               : REAL;
fAdjustValue            : REAL;
nCount                  : UDINT;
nCountBeforeChg         : UDINT;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bOverRange              : BOOL;
bUnderRange             : BOOL;
bNoSensor               : BOOL;
bOpenLoop               : BOOL;
bShortedLoop            : BOOL;
bCommFailure            : BOOL;
bConfigError            : BOOL;
bOtherFault             : BOOL;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
```

```
bAckedTrans_ToNormal     : BOOL;
tObjectID                : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError                   : BOOL;
nErrorId                 : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Present_Value*).

bHighLimitEn, fHighLimit, bLowLimitEn, fLowLimit: Status and value of the *Present_Value* reporting limits (lower and upper limit values from which messages are generated). *FALSE* → limit value not active; *TRUE* → limit value active; see also BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and properties *High_Limit*, *Low_Limit* and *Limit_Enable*.

fAdjustValue: Value of property *Adjust_Value*. *Adjust_Value* can be used to correct the value of property *Present_Value*. The value of *Adjust_Value* is subtracted from *Present_Value*. The value of the property Count is also corrected, considering the property *Scale_Factor*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Adjust_Value*.

nCount: Value of the property *Count*. *Count* represents the sampled input pulses or input value changes. In addition, the value of *Count* may contain corrections from the property *Adjust_Value*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Count*.

nCountBeforeChg: Value of property *Count_Before_Change*. Contains the value of the property *Count* before the correction through property *Adjust_Value*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Count_Before_Change*.

bOverridden, bOutOfService, **bFault**, **bInAlarm:** See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Status_Flags*.

bOverRange, bUnderRange, **bNoSensor**, **bOpenLoop**, **bShortedLoop**, **bCommFailure**, **bConfigError**, **bOtherFault:** See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Status_Flags*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

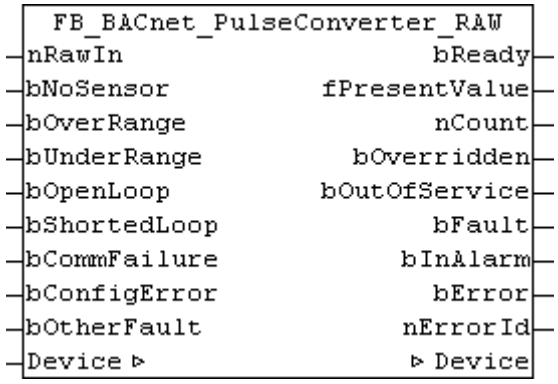
```
Device                : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.2.38 FB_BACnet_PulseConverter_RAW



Application

The function block FB_BACnet_PulseConverter_RAW can be used for read and write access to a BACnet object of type *PulseConverter*.

In contrast to the standard and *_EX* versions of the block, the raw value and the state of the property *Reliablitity* are provided by function block inputs, not directly by the IO hardware. For example, the state of a counter input can be mapped from a sub-bus system in PLC code to a BACnet object (signal conversion from sub-bus systems or virtual data points to BACnet, see Example).

VAR_INPUT

```
nRawIn          : UINT;
bNoSensor       : BOOL;
bOverRange      : BOOL;
bUnderRange     : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bCommFailure    : BOOL;
bConfigError    : BOOL;
bOtherFault     : BOOL;
```

nRawIn: Raw value input of the object in the range -32768 to 32767. The input is linked to the process data "RawIoPulseConverterUnsignedValue" of the BACnet object. Changes in the value of **nRawIn** are offset with the property *Scale_Factor* to calculate the value of the property *Present_Value* (provided the object state is not *out_of_service*).

bNoSensor, bOverRange, bUnderRange, bOpenLoop, bShortedLoop, bCommFailure, bConfigError, bOtherFault: *TRUE* at the input sets the appropriate state [▶ 349] of the property *Reliability*. The priority decreases with the order of the inputs (**bNoSensor** has highest priority, **bOtherFault** lowest). See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Reliability*.

VAR_OUPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
nCount          : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Present_Value*).

nCount: Value of the property *Count*. *Count* represents the sampled input pulses or input value changes. In addition, the value of *Count* may contain corrections from the property *Adjust_Value*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Count*.

bOverridden, bOutOfService, **bFault**, **bInAlarm:** See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Status_Flags*.

bError: An error is pending.

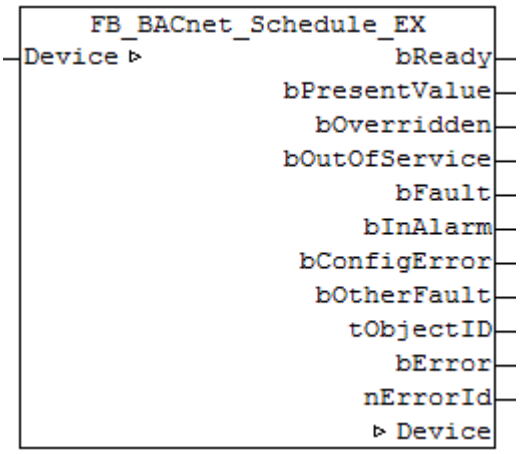
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.39 FB_BACnet_Schedule_EX



Application

The function block FB_BACnet_Schedule_EX can be used for read and write access to a BACnet object of type *Schedule*.

VAR_OUPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bConfigError    : BOOL;
bOtherFault     : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Status_Flags*.

bConfigError, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Reliability*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

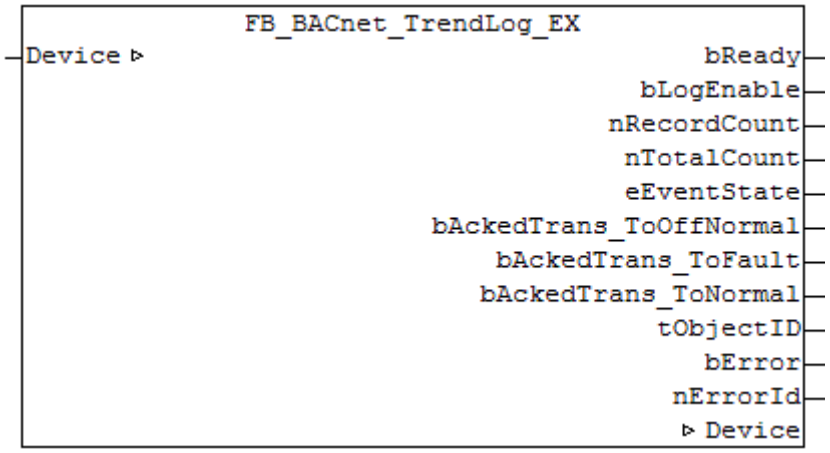
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
Device          : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.2.40 FB_BACnet_TrendLog_EX



Application

The function block FB_BACnet_TrendLog_EX can be used for read access to a BACnet object of type *TrendLog*.

VAR_OUPUT

```
bReady                  : BOOL;
bLogEnable              : BOOL;
nRecordCount            : UDINT;
nTotalCount             : UDINT;
eEventState             : E_BACNETEVENTSTATE;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
tObjectID               : DINT:=-1;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bLogEnable: Logging of values is enabled. See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Log_Enable*.

nRecordCount: Number of entries of the property *Log_Buffer*. If "0" is written to the property *Record_Count*, the *Log_Buffer* is reset. See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Record_Count*.

nTotalCount: Total number of entries of the property Log_Buffer. See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Total_Record_Count*.

eEventState: E_BACNETEVENTSTATE [▶ 336], see BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

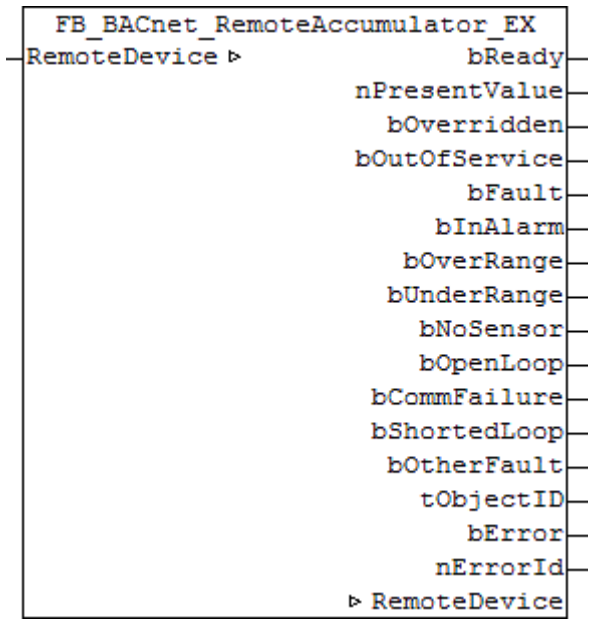
VAR_IN_OUT

```
Device               : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_Device [▶ 196] for further information.

4.3 BACnet Client Objects

4.3.1 FB_BACnet_RemoteAccumulator_EX



Application

The function block FB_BACnet_RemoteAccumulator_EX can be used for read access to a BACnet object of type *Accumulator*.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
```

```
bOverRange      : BOOL;
bUnderRange     : BOOL;
bNoSensor       : BOOL;
bOpenLoop       : BOOL;
bCommFailure    : BOOL;
bShortedLoop    : BOOL;
bOtherFault     : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_value*).

nMaxPV: Current value of the property *Max_Pres_Value* of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Max_Pres_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Event_State*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

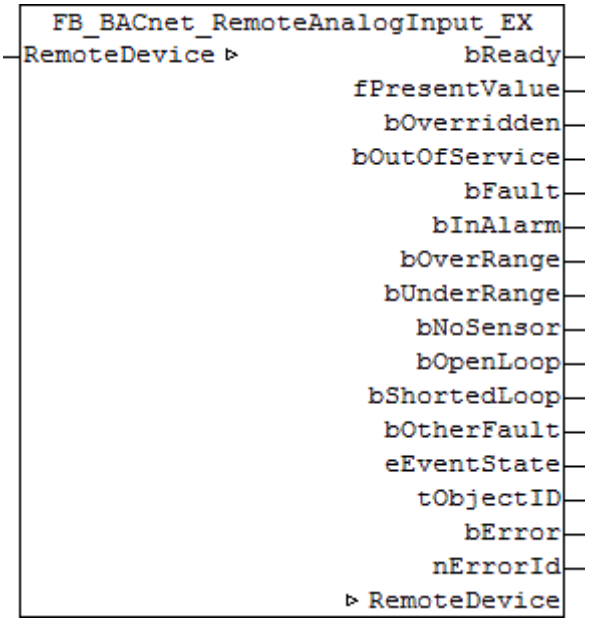
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice        : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.2 FB_BACnet_RemoteAnalogInput_EX



Application

The function block FB_BACnet_RemoteAnalogInput_EX can be used for reading access to a BACnet object of type *AnalogInput*.

VAR_OUPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bOverRange      : BOOL;
bUnderRange     : BOOL;
bNoSensor       : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bOtherFault     : BOOL;
eEventState     : E_BACNETEVENTSTATE;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Present_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

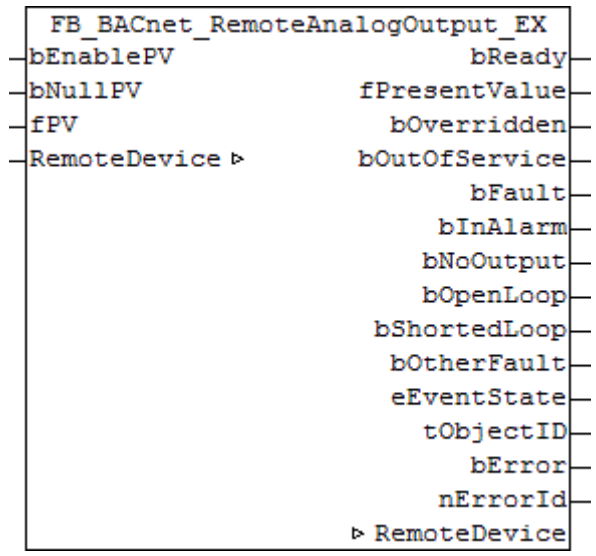
```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.3 FB_BACnet_RemoteAnalogOutput_EX



Application

The function block FB_BACnet_RemoteAnalogOutput_EX can be used for reading and write access to a BACnet object of type *AnalogOutput*.

VAR_INPUT

```
bEnablePV      : BOOL;
bNullPV        : BOOL;
fPV            : REAL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **fPV**; *FALSE* = write value of **fPV** as property value.

fPV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady         : BOOL;
fPresentValue  : REAL;
bOverridden    : BOOL;
```

```
bOutOfService  : BOOL;
bFault         : BOOL;
bInAlarm       : BOOL;
bNoOutput      : BOOL;
bOpenLoop      : BOOL;
bShortedLoop   : BOOL;
bOtherFault    : BOOL;
eEventState    : E_BACNETEVENTSTATE;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError         : BOOL;
nErrorId       : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bNoOutput, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

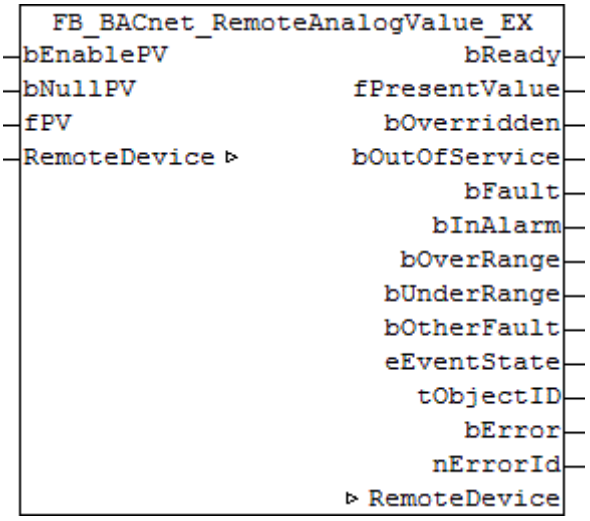
```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.4 FB_BACnet_RemoteAnalogValue_EX



Application

The function block FB_BACnet_RemoteAnalogValue_EX can be used for reading and write access to a BACnet object of type *AnalogValue*.

VAR_INPUT

Notice: Variables shown in red are not included in the basic version of the function block, but in the *_WR* version.

```
bEnablePV   : BOOL;
bNullPV     : BOOL;
fPV         : REAL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **fPV**; *FALSE* = write value of **fPV** as property value.

fPV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bOverRange      : BOOL;
bUnderRange     : BOOL;
bOtherFault     : BOOL;
eEventState     : E_BACNETEVENTSTATE;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bOverRange, bUnderRange, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Acked_Transitions*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

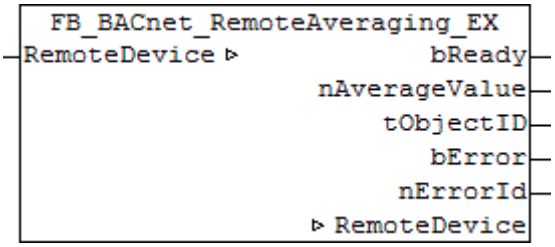
```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.5 FB_BACnet_RemoteAveraging_EX



Application

The function block FB_BACnet_RemoteAveraging_EX can be used for read access to a BACnet object of type *Averaging*.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady         : BOOL;
nAverageValue  : REAL;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError         : BOOL;
nErrorId       : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fAverageValue: Current average value (see BACnet specification DIN EN ISO 16484-5 for BACnet Object *Averaging* and property *Average_Value*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

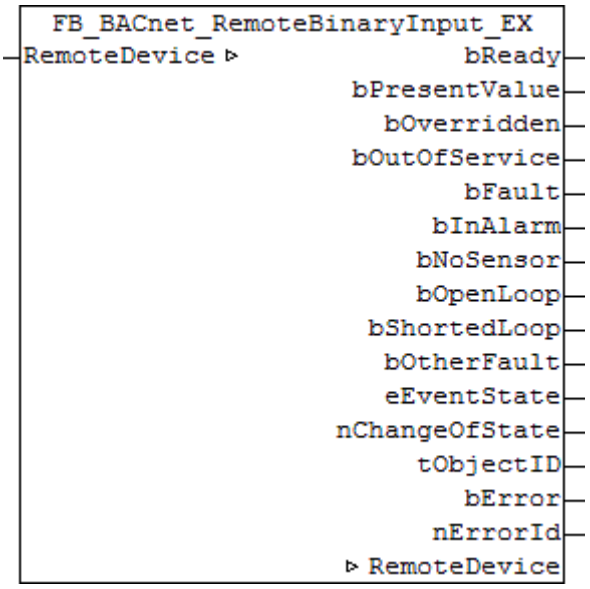
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.6 FB_BACnet_RemoteBinaryInput_EX



Application

The function block FB_BACnet_RemoteBinaryInput_EX can be used for reading access to a BACnet object of type *BinaryInput*.

VAR_OUPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bNoSensor       : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bOtherFault     : BOOL;
eEventState     : E_BACNETEVENTSTATE;
nChangeOfState  : UDINT;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Status_Flags*.

bNoSensor, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Acked_Transitions*).

nChangeOfState: Number of changes of state (*not* to be confused with changes in value) of the property *Present_Value* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Change_Of_State_Count*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

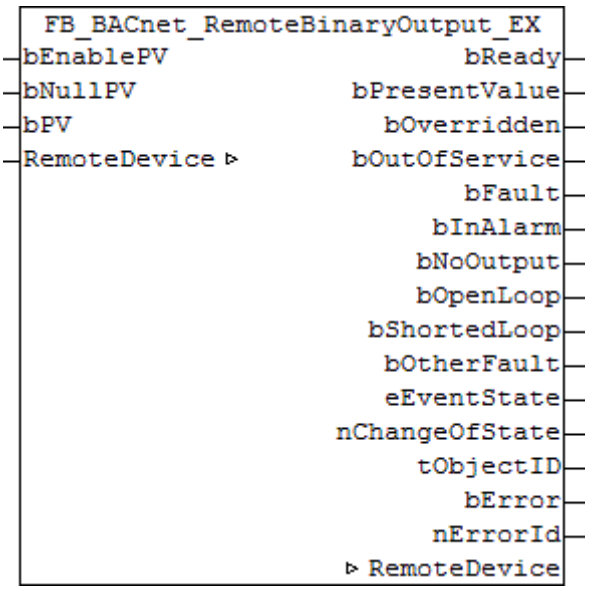
```
RemoteDevice    : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.7 FB_BACnet_RemoteBinaryOutput_EX



Application

The function block FB_BACnet_RemoteBinaryOutput_EX can be used for reading and write access to a BACnet object of type *BinaryOutput*.

VAR_INPUT

```
bEnablePV   : BOOL;
bNullPV     : BOOL;
bPV         : BOOL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **bPV**; *FALSE* = write value of **bPV** as property value.

b PV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bNoOutput       : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bOtherFault     : BOOL;
eEventState     : E_BACNETEVENTSTATE;
nChangeOfState  : UDINT;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

nChangeOfState: Number of changes of state (*not* to be confused with changes in value) of the property *Present_Value* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Change_Of_State_Count*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

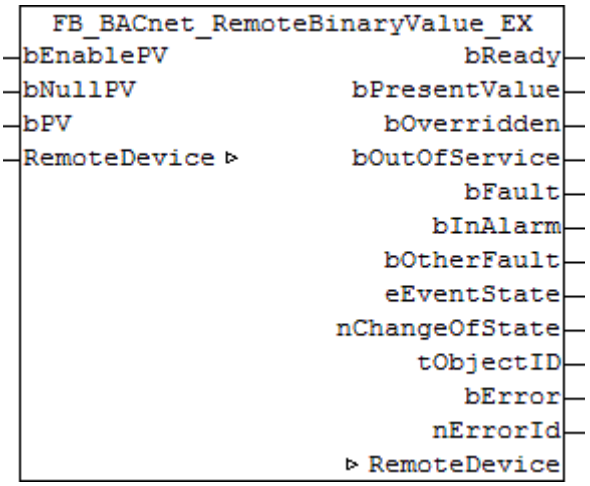
```
RemoteDevice    : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.8 FB_BACnet_RemoteBinaryValue_EX



Application

The function block FB_BACnet_RemoteBinaryValue_EX can be used for read and write access to a BACnet object of type *BinaryValue*.

VAR_INPUT

Notice: Variables shown in red are not included in the basic version of the function block, but in the *_WR* version.

```
bEnablePV    : BOOL;
bNullPV      : BOOL;
bPV          : BOOL;
```

bEnablePV: *TRUE* = write enable of the property value; *FALSE* = do not change the entry in the BACnet object (**bNullPV** and **fPV** have no effect).

bNullPV: *TRUE* = delete the property entry (*NULL*); instead of the value of **bPV**; *FALSE* = write value of **bPV** as property value.

b PV: Value to be written to the property *Present_Value,* if **bEnablePV** = *TRUE* and **bNullPV** = *FALSE*. The process data are written when there is a change.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bOtherFault     : BOOL;
eEventState     : E_BACNETEVENTSTATE;
nChangeOfState  : UDINT;
```

```
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

nChangeOfState: Number of changes of state (*not* to be confused with changes in value) of the property *Present_Value* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Change_Of_State_Count*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

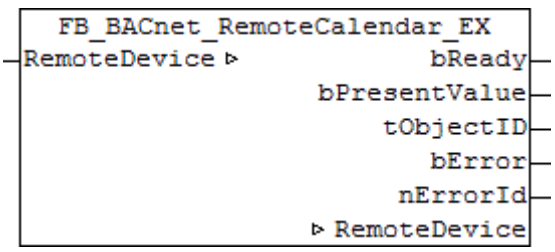
```
RemoteDevice    : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.9 FB_BACnet_RemoteCalendar_EX



Application

The function block FB_BACnet_RemoteCalendar_EX can be used for reading access to a BACnet object of type *Calendar*.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady          : BOOL;
bPresentValue   : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Calendar* and property *Present_Value*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

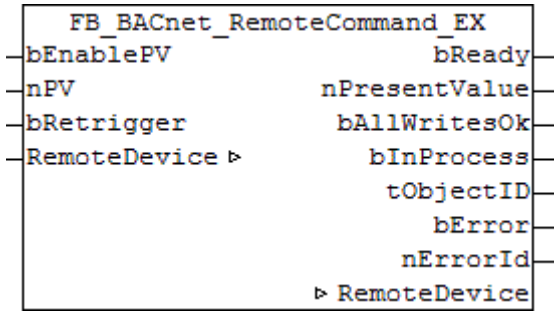
nErrorId: see global constants BACnet Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice        : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.10 FB_BACnet_RemoteCommand_EX



Application

The function block FB_BACnet_RemoteCommand_EX can be used for reading and write access to a BACnet object of type *Command*.

VAR_INPUT

```
bEnablePV  : BOOL;
nPV        : UDINT;
bRetrigger : BOOL;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, writing takes place into the property *Present_Value* of the corresponding BACnet object with the value of input **nPV**.
If **bEnablePV** is set to *FALSE*, the process data of the mapped property *Present_Value* are written to *0* and thus deactivated.

nPV: Value of the property *Present_Value* to be written. If the value is outside the value range (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*), the corresponding process data is disabled, writing to the property is disabled.
Writing of a valid value triggers execution of the corresponding command list of the BACnet object. The

value of the property *Present_Value* is written whenever the process data changes (i.e. change in the value of **nPV** with **bEnablePV** set, or signal change from *FALSE* to *TRUE* at input **bRetrigger** with **bEnablePV** set).

bRetrigger: A rising edge at this input triggers a repeat of the write process and therefore execution of the corresponding commands of the BACnet object *Command*. The signal change from *FALSE* to *TRUE* corresponds to a change of the process data of the property *Present_Value* from x to *0* to x.

VAR_OUPUT

Variables are not included in the basic version of the function block.

```
bReady          : BOOL;
nPresentValue   : UDINT;
bAllWritesOk    : BOOL;
bInProcess      : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the function block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*).

bAllWritesOk: the last requested command list was successfully processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *All_Writes_Successful*).

bInProcess: the selected command list is processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *In_Process*).

tObjectID: object ID of the BACnet object Object type and object instance.

bError: an error is pending.

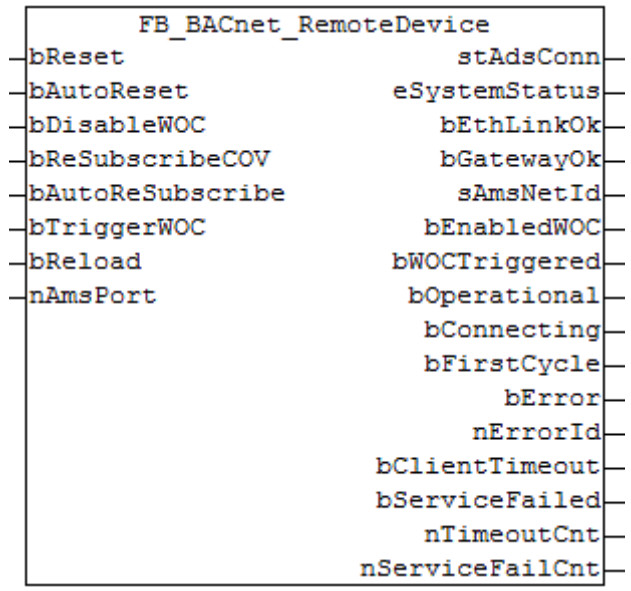
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice        : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.11 FB_BACnet_RemoteDevice



Application

Function block for interfacing of the PLC program with a remote object BACnet device object (client). The function block is linked with the aid of process data and ADS (evaluate AMS port, read object list and determine the stack revision).

An instance of the function block FB_BACnet_RemoteDevice is required whenever objects of a remote BACnet server are to be accessed from the PLC with the aid of a local BACnet client. Several instances of the function block FB_BACnet_RemoteDevice can be created and linked with the corresponding clients in the TwinCAT System Manager via PLC automapping.

As a rule, the function block is backward compatible (revision 6). If Revision 6 is used, some ADS services of the BACnet stack are limited; e.g. the AMS port of the BACnet device (server and client) can only be determined automatically from Revision 12. If revision 6 is used, the AMS port (Ads port) shown in the TwinCAT System Manager must therefore be transferred at input **nAmsPort** (see Figure 1)!

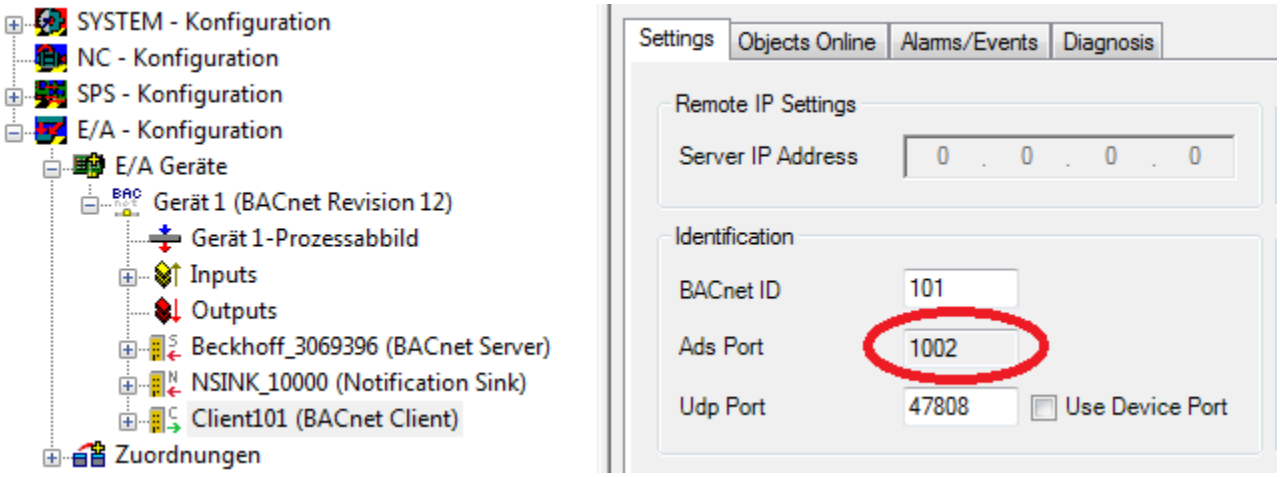


Fig. 18: Fig. 1: AMS port of the BACnet Client in the TwinCAT System Manager

VAR_INPUT

Variables are optional and do not have to be assigned if revision 12 or higher is used.

```
bReset            : BOOL;
bAutoReset        : BOOL;
bDisableWOC       : BOOL;
bReSubscribeCOV   : BOOL;
bAutoReSubscribe  : BOOL;
bTriggerWOC       : BOOL;
bReload           : BOOL;
nAmsPort          : UINT:=0;  (*siehe Info*)
```

bReset: resets the error state in the event of a signal change *FALSE --> TRUE*.

bAutoReset: if the input is set to *TRUE*, errors with ID 22 and 23 are acknowledged automatically. Linked remote function blocks are not blocked (**bReady** at the object FBs remains TRUE). Instead of the error an error counter is incremented (see **nTimeoutCnt** and **nServiceFailCnt**).

bDisableWOC: if the input is set to *TRUE*, WriteOnChange (WOC) is suppressed for all objects of this client. This means that changes in the process data do not automatically trigger writing of the properties to the remote objects (only applies to properties that are configured for WriteOnChange under the client).

bReSubscribeCOV: a signal change at the input from *FALSE --> TRUE* triggers resubscription of all properties that are configured for COV. This function may affect many objects and should therefore only be executed when necessary. Normally all subscribed properties are automatically re-subscribed once a timeout time has elapsed (resubscription interval).

bAutoReSubscribe: setting the input to *TRUE* triggers the automatic resubscription of all properties configured for COV on demand. Not to be confused with the resubscription interval! Automatic resubscription is triggered if a remote server was no longer accessible and the connection has been restored, for example. If the input is set to *FALSE*, resubscription is only triggered once the resubscription interval has elapsed.

bTriggerWOC: from revision 12: change from *FALSE → TRUE* writes all process data again.

bReload: the ADS connection is renewed. Subsequent function blocks, which use the ADS connection, are also triggered.

nAmsPort: *only if revision 6 is used, otherwise do not use:* AMS port of the local BACnet client. In the standard case, this is not defined and must be read in the TwinCAT System Manager (see Figure 1).

VAR_OUPUT

```
stAdsConn         : ST_BACnet_AdsConnection;
eSystemStatus     : E_BACNETDEVICESTATUS;
bEthLinkOk        : BOOL;
bGatewayOk        : BOOL;
sAmsNetId         : T_AmsNetId;
bEnabledWOC       : BOOL;
bWOCTriggered     : BOOL;
bOperational      : BOOL;
bConnecting       : BOOL;
bFirstCycle       : BOOL;
bError            : BOOL;
nErrorId          : UINT;
bClientTimeout    : BOOL;
bServiceFailed    : BOOL;
nTimeoutCnt       : UDINT;
nServiceFailCnt   : UDINT;
```

eSystemStatus: Status of the remote BACnet server object (see BACnet specification DIN EN ISO 16484-5 for BACnet device object and property *System_Status*).

bEthLinkOk: The local Ethernet connection is active (cable connected, adapter linked) if the output is set to *TRUE*.

bGatewayOk: The configured gateway can be reached if the output is set to *TRUE*.

sAmsNetId: Output of the AMS NetID of the local BACnet adapter (can be used for asynchronous access to BACnet objects via ADS).

bEnabledWOC: WriteOnChange (WOC) is activated for properties that are configured for COV.

bOperational: BACnet device object of the remote server reports "operational" (System_Status = 0), the device adapter reports status **bEthLinkOk**, status **bConnecting** is *FALSE* and bit 0 of the process data "Client Status" is not set (timeout error [▶ 328]). If the output becomes *FALSE*, all linked objects are blocked (block instances).

bConnecting: The connection to the remote server is established (process data "Client Status" is less than 0x04xx).

bFirstCycle: Is set for one cycle when the block instance is first called after a PLC reset or restart.

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

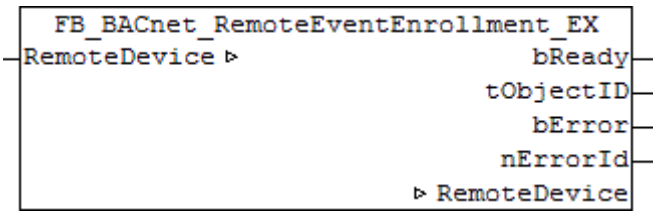
bClientTimeout: A timeout error [▶ 328] has occurred if the output is set. The output is only reset once no more timeout errors have occurred for 10 seconds.

bServiceFailed: A service error [▶ 328] has occurred if the output is set. The output is only reset once no more errors have occurred for 10 seconds.

nTimeoutCnt: Number of timeout errors [▶ 328] that have occurred. The positive edge of the corresponding status flag of the client is counted.

nServiceFailCnt: Number of service errors [▶ 328] that have occurred. The positive edge of the corresponding status flag of the client is counted.

4.3.12 FB_BACnet_RemoteEventEnrollment_EX



Application

The function block FB_RemoteEventEnrollment_EX can be used for reading access to a BACnet object of type *EventEnrollment*.

VAR_OUPUT

Variables are not included in the basic version of the function block.

```
bReady         : BOOL;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;  (*siehe Info*)
bError         : BOOL;
nErrorId       : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the function block instance was not linked correctly in the TwinCAT System Manager.

tObjectID: object ID of the BACnet object Object type and object instance.

bError: an error is pending.

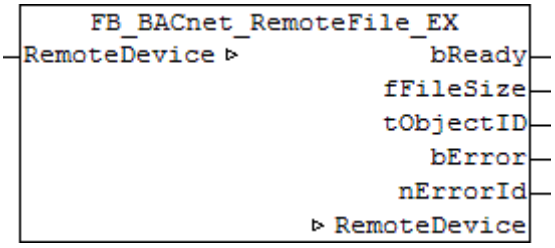
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice      : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.13 FB_BACnet_RemoteFile_EX



Application

The function block FB_BACnet_RemoteFile_EX can be used for reading access to a BACnet object of type *File*.

VAR_OUPUT

```
bReady        : BOOL;
fFileSize     : UDINT;
tObjectID     : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError        : BOOL;
nErrorId      : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fFileSize: Current file size in bytes (see also BACnet specification DIN EN ISO 16484-5 re. BACnet object *File* and property *File_Size*).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

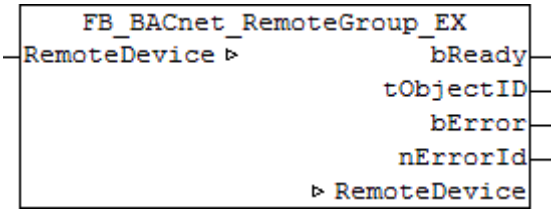
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice      : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.14 FB_BACnet_RemoteGroup_EX



Application

The function block FB_BACnet_RemoteGroup_EX can be used for reading access to a BACnet object of type *Group*.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady          : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

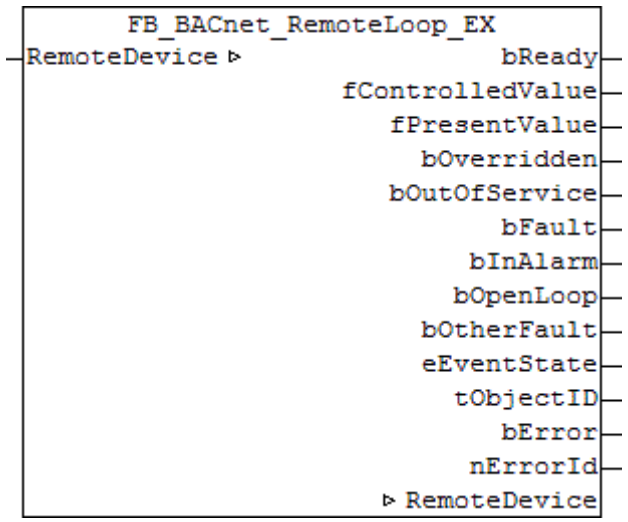
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.15 FB_BACnet_RemoteLoop_EX



Application

The function block FB_BACnet_RemoteLoop_EX can be used for reading access to a BACnet object of type *Loop*.

VAR_OUPUT

```
bReady             : BOOL;
fControlledValue   : REAL;
fPresentValue      : REAL;
bOverridden        : BOOL;
bOutOfService      : BOOL;
bFault             : BOOL;
bInAlarm           : BOOL;
```

```
bOpenLoop          : BOOL;  (*siehe Beschreibung*)
bOtherFault        : BOOL;  (*siehe Beschreibung*)
eEventState        : E_BACNETEVENTSTATE;  (*siehe Beschreibung*)
tObjectID          : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;  (*siehe Beschreibung*)
bError             : BOOL;
nErrorId           : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the function block instance was not linked correctly in the TwinCAT System Manager.

fControlledValue: feedback of the current process parameter (X, actual value).

fPresentValue: feedback of the current control output (X, control value). Attention: *Present_Value* and *Controlled_Variable_Value* can easily lead to confusion (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and properties *Present_Value*, *Controlled_Variable_Value* and *Controlled_Variable_Reference*).

fPropBand: feedback of the current control output in percent (-100%...+100%) in relation to the minimum and maximum control output (properties *Minimum_Output* and *Maximum_Output*).

bOverridden, bOutOfService, bFault, bInAlarm: see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Status_Flags*.

bOpenLoop, bCommFailure, bOtherFault: see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Reliability*.

eEventState: see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property Event_State [▶ 337].

tObjectID: object ID of the BACnet object Object type and object instance.

bError: an error is pending.

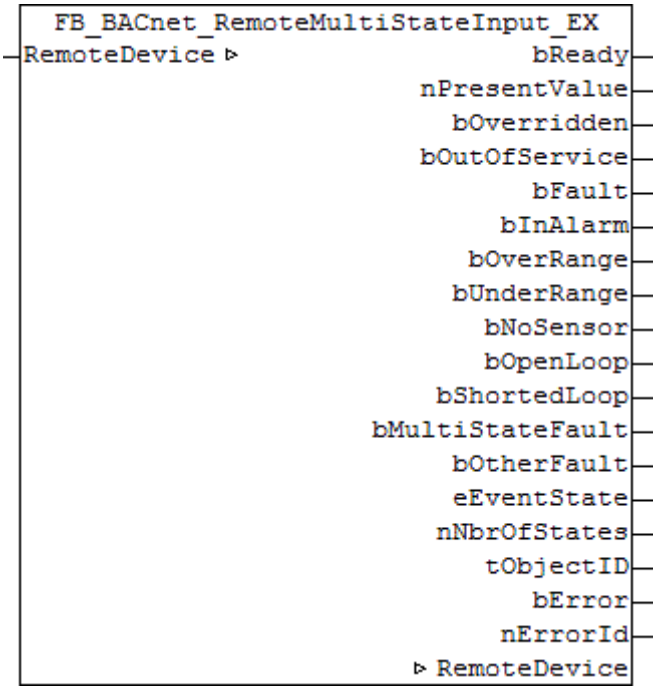
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.16 FB_BACnet_RemoteMultiStateInput_EX



Application

The function block FB_BACnet_RemoteMultiStateInput_EX can be used for read access to a BACnet object of type *MultiStateInput*.

VAR_OUPUT

```
bReady            : BOOL;
nPresentValue     : UDINT;
bOverridden       : BOOL;
bOutOfService     : BOOL;
bFault            : BOOL;
bInAlarm          : BOOL;
bOverRange        : BOOL;
bUnderRange       : BOOL;
bNoSensor         : BOOL;
bOpenLoop         : BOOL;
bShortedLoop      : BOOL;
bMultiStateFault  : BOOL;
bOtherFault       : BOOL;
eEventState       : E_BACNETEVENTSTATE;
nNbrOfStates      : UDINT;
tObjectID         : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError            : BOOL;
nErrorId          : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Status_Flags*.

bOverRange, bUnderRange, bNoSensor, bOpenLoop, bShortedLoop, bMultiStateFault, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Reliability*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Event_State*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

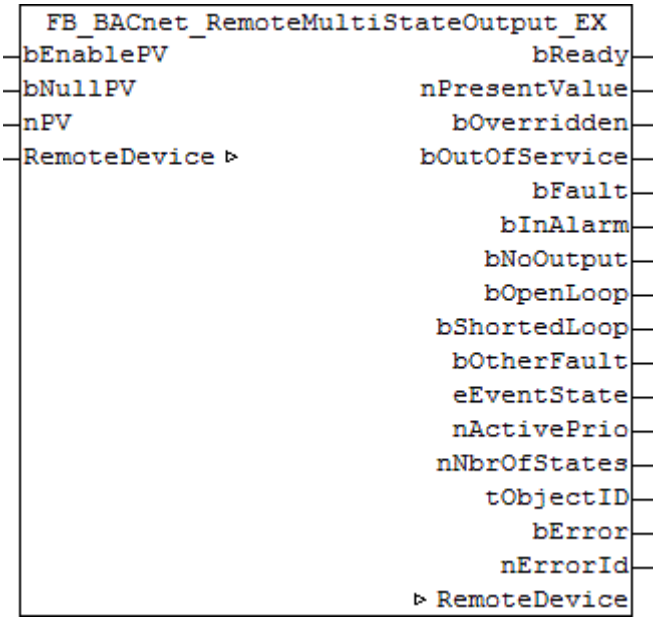
```
RemoteDevice      : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.17 FB_BACnet_RemoteMultiStateOutput_EX



Application

The function block FB_BACnet_RemoteMultiStateOutput_EX can be used for read and write access to a BACnet object of type *MultiStateOutput*.

VAR_INPUT

Notice: Variables shown in red are not included in the basic version of the function block, but in the *_WR* version.

```
bEnablePV      : BOOL;
bNullPV        : BOOL;
nPV            : UDINT;
```

bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **nState** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g. for deleting a priority), *FALSE* = write value from **nState**

nPV: Value that is written to the BACnet object, if **bEnable** = *TRUE* and **bNull** = *FALSE*. Multi-State in the range [1 .. *Number_Of_States*].

VAR_OUPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bNoOutput       : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bOtherFault     : BOOL;
eEventState     : E_BACNETEVENTSTATE;
nActivePrio     : UINT;
nNbrOfStates    : UDINT;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

bNoOutput, bOpenLoop, bShortedLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Reliability*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Event_State*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Acked_Transitions*).

nActivePrio: Indicates the currently active priority of the priority array, which acts on the *Present_Value* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_Value*).

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If **nNbrOfStates** is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

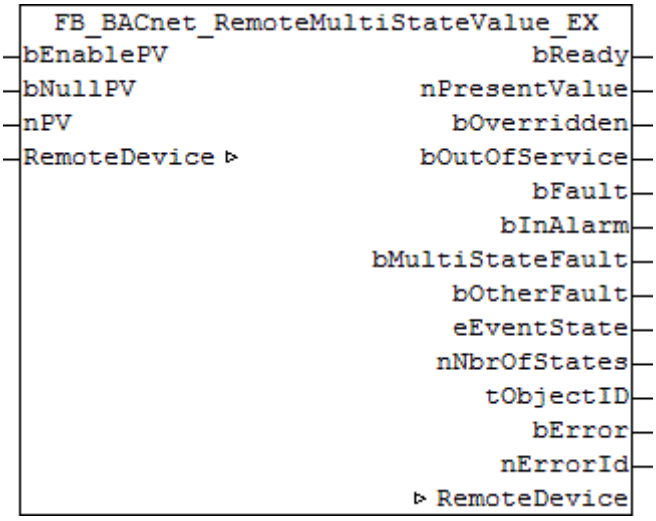
```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.3.18 FB_BACnet_RemoteMultiStateValue_EX



Application

The function block FB_BACnet_RemoteMultiStateValue_EX can be used for reading and write access to a BACnet object of type *MultiStateValue*.

VAR_INPUT

```
bEnablePV      : BOOL;
bNullPV        : BOOL;
nPV            : UDINT;
```

bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **nState** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g. for deleting a priority), *FALSE* = write value from **nState**

nState: Value that is written to the BACnet object, if **bEnable** = *TRUE* and **bNull** = *FALSE*. Multi-State in the range [1 .. *Max_Number_Of_States*].

VAR_OUPUT

```
bReady             : BOOL;
nPresentValue      : UDINT;
bOverridden        : BOOL;
bOutOfService      : BOOL;
bFault             : BOOL;
bInAlarm           : BOOL;
bMultiStateFault   : BOOL;
bOtherFault        : BOOL;
eEventState        : E_BACNETEVENTSTATE;
nNbrOfStates       : UDINT;
tObjectID          : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError             : BOOL;
nErrorId           : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Status_Flags*.

bMultiStateFault, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Reliability*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Event_State*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateValue object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

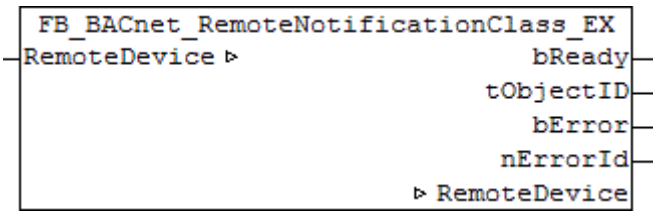
```
RemoteDevice      : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 336]

4.3.19 FB_BACnet_RemoteNotificationClass_EX



Application

The function block FB_BACnet_RemoteNotificationClass_EX can be used for reading access to a BACnet object of type *NotificationClass*.

VAR_OUPUT

```
bReady        : BOOL;
tObjectID     : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError        : BOOL;
nErrorId      : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

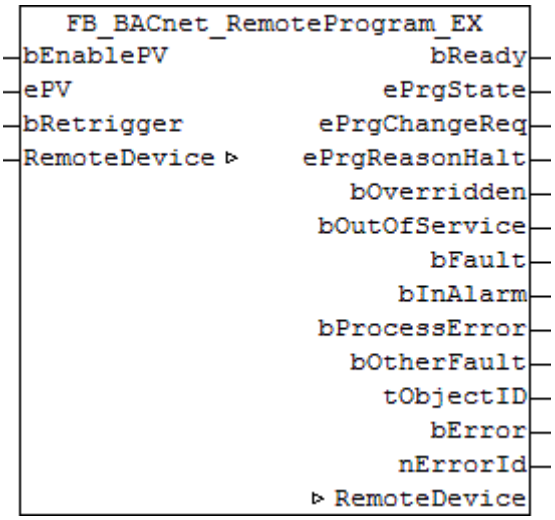
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice      : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.20 FB_BACnet_RemoteProgram_EX



Application

The function block FB_BACnet_RemoteProgram_EX can be used for reading and write access to a BACnet object of type *Program*.

VAR_INPUT

Notice: Variables shown in red are not included in the basic version of the function block, but in the *_WR* version.

```
bEnablePV       : BOOL;
ePV             : E_BACNETPROGRAMREQUEST;
bRetrigger      : BOOL;
```

bEnablePV: Enables the value of input **ePV**. If the input is set to *TRUE*, the value of **ePV** is written to the property *Program_Change* of the object. If the input is set to *FALSE*, *0xFFFF* is written to the process data of the property *Program_Change* of the object. The value *0xFFFF* prevents writing to the remote server and therefore writing to the remote object.

ePV: Request to the remote program object. If **bEnablePV** was set to *TRUE*, the value of the input is written to the property *Program_Change* (see also Transition diagram [▶ 254]).

bRetrigger: A change from *FALSE* to *TRUE* triggers writing of the input **ePV** to the property *Program_Change*, if input **bEnablePV** is set to *TRUE*.

VAR_OUPUT

```
bReady              : BOOL;
ePrgState           : E_BACNETPROGRAMSTATE;
ePrgChangeReq       : E_BACNETPROGRAMREQUEST;
ePrgReasonHalt      : E_BACNETPROGRAMERROR;
bOverridden         : BOOL;
bOutOfService       : BOOL;
bFault              : BOOL;
bInAlarm            : BOOL;
bProcessError       : BOOL;
bOtherFault         : BOOL;
tObjectID           : DINT:=-1;
bError              : BOOL;
nErrorId            : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

ePrgState: Feedback of the current program state (see also Transition diagram [▶ 254]).

ePrgChangeReq: Feedback on the state of the current program request (see also Transition diagram [▶ 254]).

ePrgReasonHalt: Error feedback on program cancellation.

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Status_Flags*.

bProcessError, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Reliability*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice    : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Transition diagram

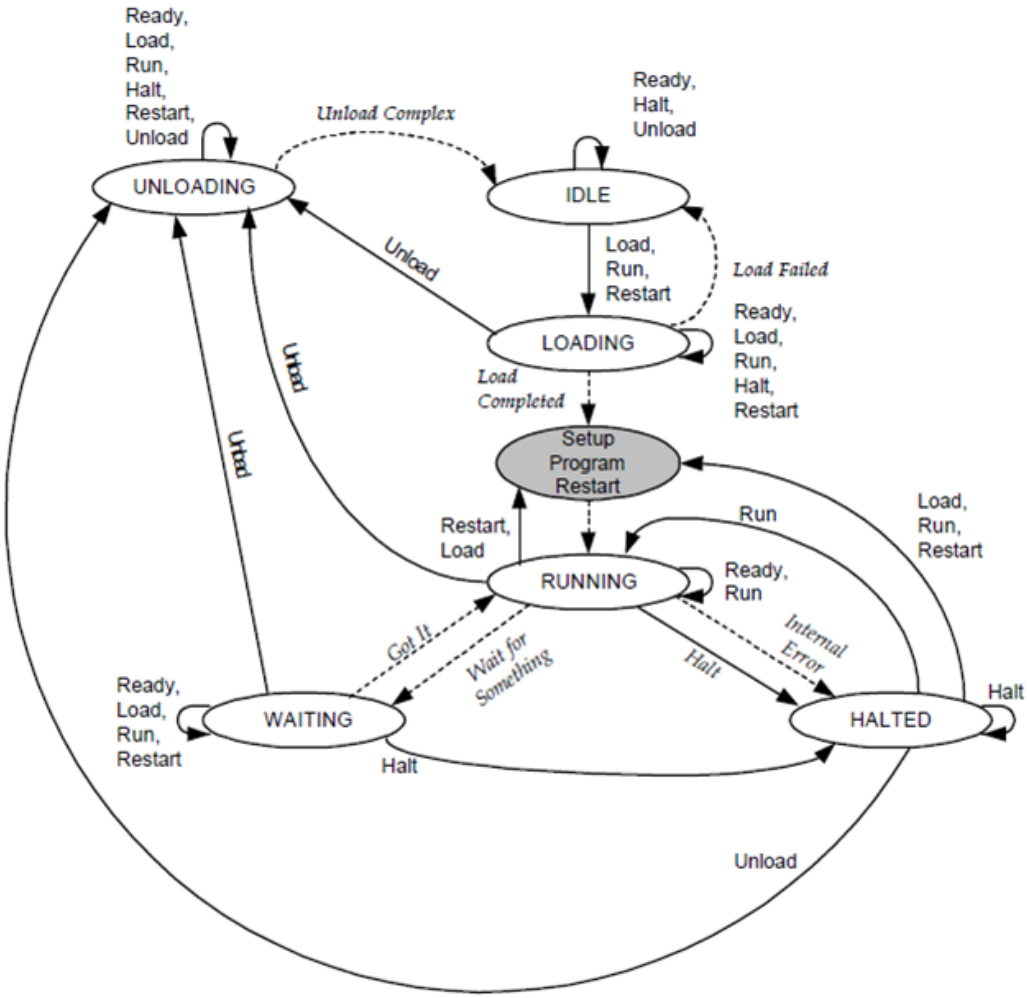
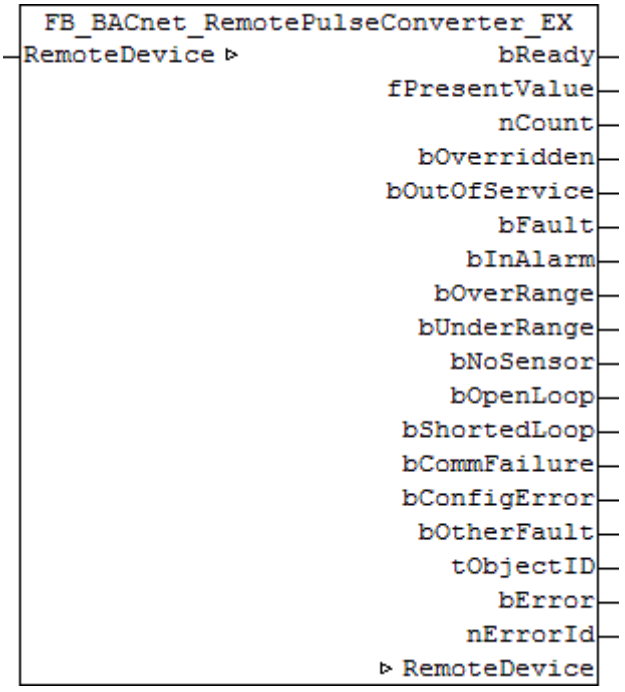


Fig. 19: Fig. 1: from BACnet specification DIN EN ISO 16484-5 for BACnet object Program, Fig. 12-3 "State Transitions for the program object"

Also see about this

- E_BACNETPROGRAMREQUEST [▶ 345]

4.3.21 FB_BACnet_RemotePulseConverter_EX



Application

The function block FB_BACnet_RemotePulseConverter_EX can be used for read access to a BACnet object of type *PulseConverter*.

VAR_OUPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
nCount          : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bOverRange      : BOOL;
bUnderRange     : BOOL;
bNoSensor       : BOOL;
bOpenLoop       : BOOL;
bShortedLoop    : BOOL;
bCommFailure    : BOOL;
bConfigError    : BOOL;
bOtherFault     : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

fPresentValue: Current value of the BACnet object (see BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Present_Value*).

nCount: Value of the property *Count*. *Count* represents the sampled input pulses or input value changes. In addition, the value of *Count* may contain corrections from the property *Adjust_Value*. See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Count*.

bOverridden, bOutOfService, **bFault**, **bInAlarm:** See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Status_Flags*.

bOverRange, bUnderRange, **bNoSensor**, **bOpenLoop**, **bShortedLoop**, **bCommFailure**, **bConfigError**, **bOtherFault:** See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Status_Flags*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

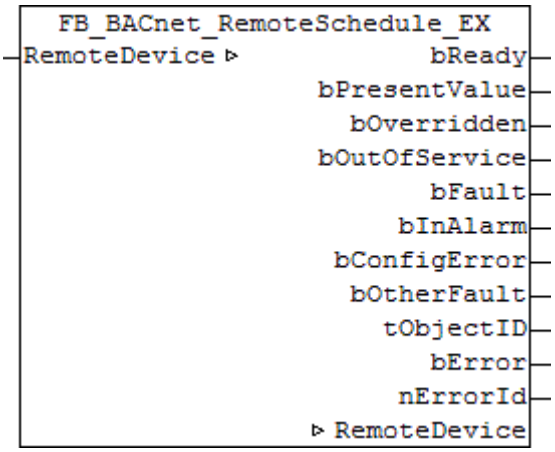
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.22 FB_BACnet_RemoteSchedule_EX



Application

The function block FB_BACnet_RemoteSchedule_EX can be used for reading access to a BACnet object of type *Schedule*.

VAR_OUPUT

Notice: Variables shown in grey color are not included in the basic version of the block.

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bConfigError    : BOOL;
bOtherFault     : BOOL;
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Status_Flags*.

bConfigError, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Reliability*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

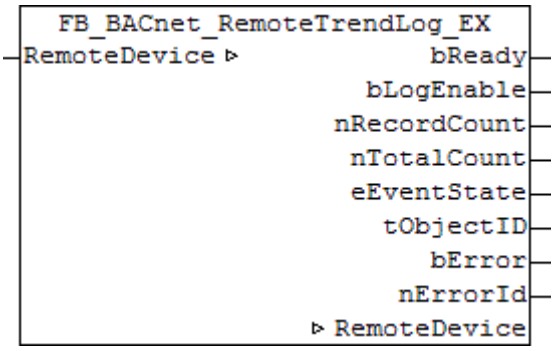
nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

```
RemoteDevice       : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

4.3.23 FB_BACnet_RemoteTrendLog_EX



Application

The function block FB_BACnet_RemoteTrendLog_EX can be used for read access to a BACnet object of type *TrendLog*.

VAR_OUPUT

```
bReady         : BOOL;
bLogEnable     : BOOL;
nRecordCount   : UDINT;
nTotalCount    : UDINT;
eEventState    : E_BACNETEVENTSTATE;
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bError         : BOOL;
nErrorId       : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_Device does not report "Operational", or the block instance was not linked correctly in the TwinCAT System Manager.

bLogEnable: Logging of values is enabled. See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Log_Enable*.

nRecordCount: Number of entries of the property *Log_Buffer*. If "0" is written to the property *Record_Count*, the *Log_Buffer* is reset. See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Record_Count*.

nTotalCount: Total number of entries of the property Log_Buffer. See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Total_Record_Count*.

eEventState: E_BACNETEVENTSTATE, see BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Event_State*.

tObjectID: Object ID of the BACnet object (object type and object instance).

bError: An error is pending.

nErrorId: see global constants BACnet_Globals [▶ 328].

VAR_IN_OUT

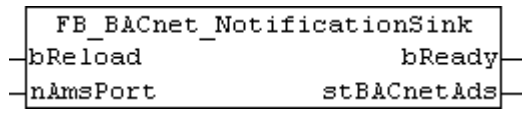
```
RemoteDevice      : FB_BACnet_RemoteDevice;
```

RemoteDevice: Specification of the instance of the corresponding remote BACnet server block (client). A BACnet adapter can be used for several BACnet clients. See FB_BACnet_Adapter [▶ 153] and FB_BACnet_RemoteDevice [▶ 241] for further information.

Also see about this

- E_BACNETEVENTSTATE [▶ 337]

4.4 FB_BACnet_NotificationSink



Application

Function block for realizing an ADS connection with a BACnet NotificationSink that was created in the TwinCAT System Manager:

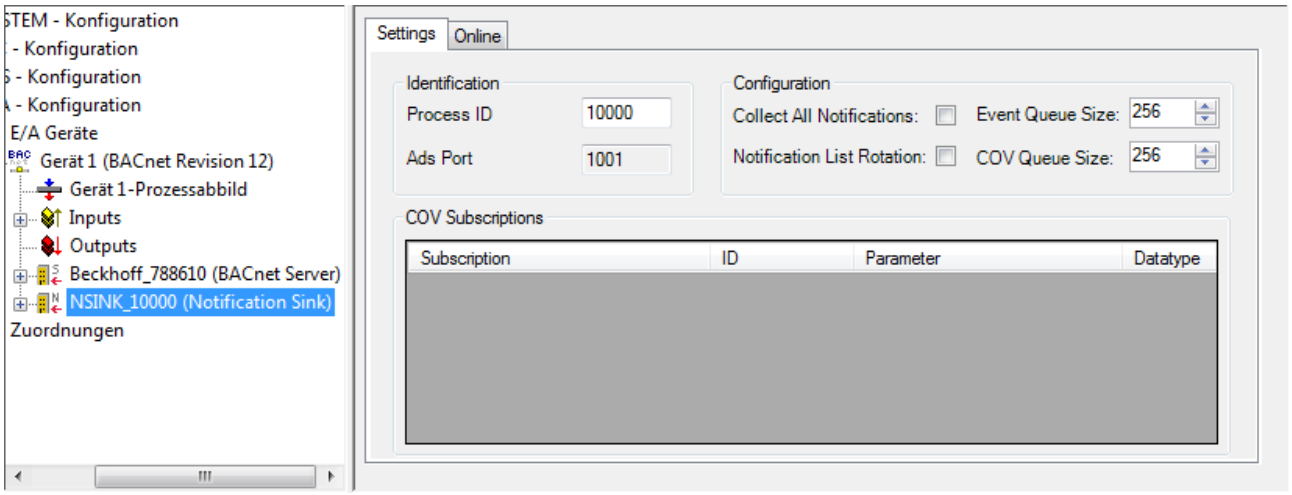


Fig. 20: Fig. 1: Notification sink in TwinCAT System Manager with open Settings dialog. "ADS Port" reflects the AMS port of the ADS connection.

The following blocks are available for accessing the NotificationSink:

Blocks	Description
FB_BACnet_NSinkReadEvent [▶ 264]	ADS access to the BACnet notification sink: reading a BACnet event
FB_BACnet_NSinkAcknEvent [▶ 261]	ADS access to the BACnet notification sink: service for acknowledging a BACnet event
FB_BACnet_NSinkRemoveEvent [▶ 268]	ADS access to the BACnet notification sink: Deleting a BACnet event (replaces FB_BACnet_NotificationSinkDelEntry [▶ 489])

VAR_INPUT

```
bReload      : BOOL;
nAmsPort     : T_AmsPort:=0;
```

bReload: The ADS connection is refreshed. Subsequent blocks, which use the ADS connection, are also triggered.

nAmsPort: AMS port (Ads Port) through which the NotificationSink can be accessed (see TwinCAT System Manager → Settings dialog of the corresponding NotificationSink; see Figure 1). Values between 1000 and 65534 are valid. No check takes place. If the specified port is wrong or no assigned, the behavior may be unexpected, or ADS errors may occur in function blocks that use this connection.

VAR_OUPUT

```
bReady       : BOOL;
stBACnetAds  : ST_BACnet_AdsConnection;
```

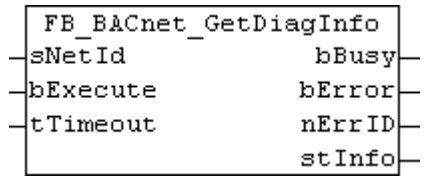
bReady: The specified AMS port is in the valid range (1000 ≤ **nAmsPort** < 65535).

stBACnetAds: Structure with the connection data.

4.5 BACnet ADS

4.5.1 FB_BACnet_GetDiagInfo

Function block for access to BACnet diagnosis via ADS. In contrast to the Read/Write property, the ADS access takes place on the BACnet adapter. The BACnet adapter is instantiated by the global variables of the PLC Library and is linked to the BACnet adapter process data in the TwinCAT System Manager. Diagnostic data of a remote BACnet server can be queried via its remote AMS NetID.



Application

The function block instance is created by the PLC program and called cyclically. The input **sNetId** must be assigned the corresponding AMS NetID of the BACnet adapter. The AMS NetID can be queried via the global BACnet adapter instance.

The AMS NetID does not match the local AMS NetID and always has to be specified - empty strings cannot be used!

The current AMS NetID is output by the associated FB_BACnet_Adapter [▶ 153]:

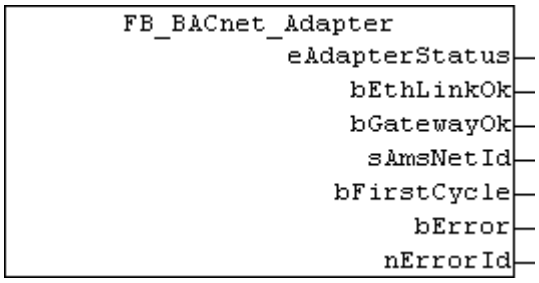


Fig. 21: **Figure 1**: FB_BACnet_Adapter [▶ 153] → **sAmsNetId** outputs the local AMS NetID of the BACnet adapter

The AMS NetID can also be displayed in the TwinCAT System Manager (see Figure 2). The AMS port of the BACnet adapter is 0xFFFF (65535).

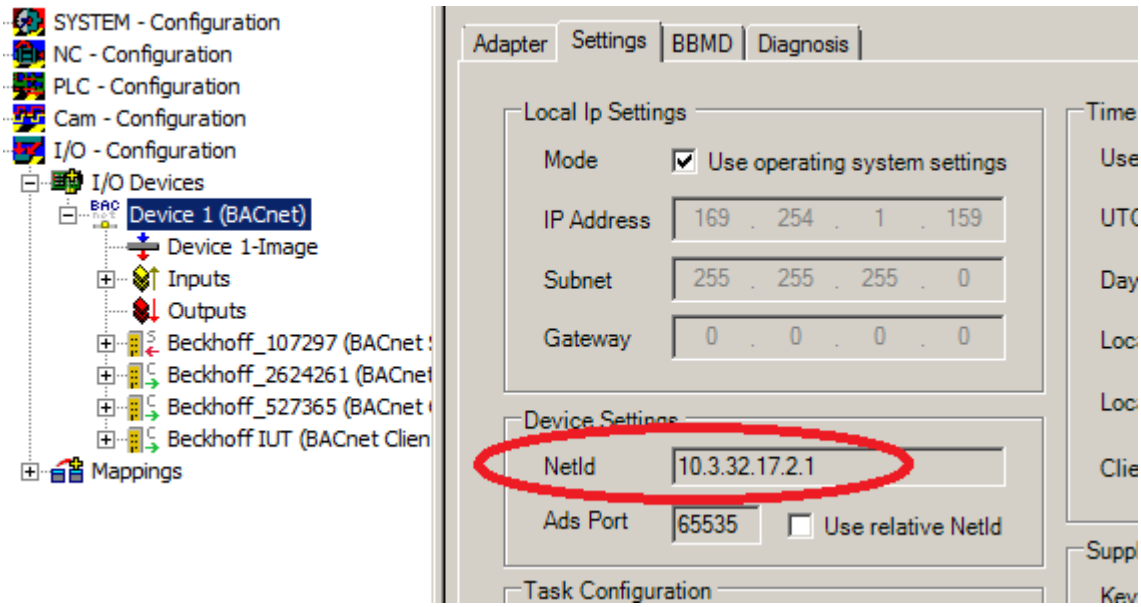


Fig. 22: **Figure 2**: AMS NetID of the BACnet adapter

VAR_INPUT

```
sNetId    : T_AmsNetId;
bExecute  : BOOL;
tTimeout  : TIME:=BACnet_ADSTimeOut;
```

sNetId: AMS NetId of the BACnet adapter.

bExecute: Rising edge at the input starts the read process.

tTimeout: Optional input, monitoring time for ADS access (default: see BACnet_ADSTimeOut [▶ 328]).

VAR_OUPUT

```
bBusy     : BOOL;
bError    : BOOL;
nErrID    : UDINT;
stInfo    : ST_BACnet_Diagnosis;
```

bBusy: The block is busy.

bError: Error during processing.

nErrID: ADS error code.

stInfo: Structure with diagnostic information on BACnet.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     fbDiag       : FB_BACnet_GetDiagInfo;
0004     tonReadDiag : TON;
0005 END_VAR
```

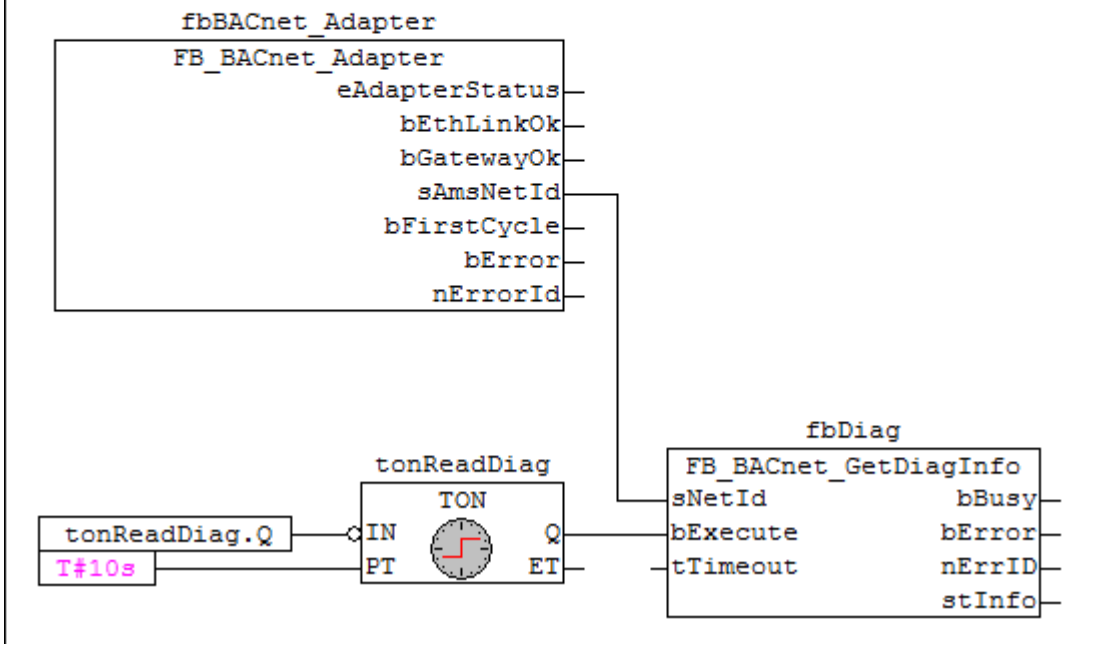
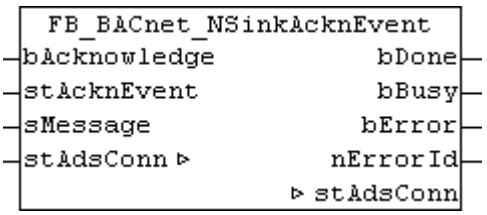


Fig. 23: **Figure 3:** Example for cyclic reading of the diagnostic information

The instance fbBACnet_Adapter is a global variable of the PLC Library of type FB_BACnet_Adapter [▶ 153]. If should not be instantiated more than once.

4.5.2 FB_BACnet_NSinkAcknEvent



Application

The function block can be used to acknowledge an event from the BACnet Notification Sink. The Example shows a possible configuration.

The event to be acknowledged is transferred at the input **stAcknEvent**. Events from the BACnet notification sink can be read with the aid of FB_BACnet_NSinkReadEvent [▶ 264]. It is irrelevant whether the event is still present in the Notification Sink at the time of the acknowledgement, or whether it was already removed or overwritten.

When an event is acknowledged, BACnet triggers an acknowledgement, based on the event parameters. The acknowledgement is only successful, if the event time stamp of the corresponding transition of the object to be acknowledged (see Figure 4) matches the time stamp from the event parameters (see Figure 3). The acknowledgement is aborted with an error if the time stamps do not match exactly (see Figure 5).

The input **sMessage** can be used to append a message to the Acknowledge event. The block expects a *Windows-1252*-coded string at the input. Internally, the block codes an UTF-8 string from it (see FB_BACnet_StringExtEncode [▶ 321]). If an error occurs during coding, this is indicated at the **nErrorId** output.

The block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stBACnetAds** of the corresponding NotificationSink function block (FB_BACnet_NotificationSink [▶ 258]).

	Time	Notification
✓ ℹ	12.11.2014 10:15:28	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:31	Beckhoff_927609 - BI_0 : BI_0: Eingang ist in Fehler
✓ ⚠	12.11.2014 10:15:32	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:34	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:37	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:40	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:43	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ⚠	12.11.2014 10:15:46	Beckhoff_927609 - BI_0 : BI_0: Eingang ist in Fehler
✓ ℹ	12.11.2014 10:15:48	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:49	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:52	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:55	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS

Fig. 24: Figure 1: Online view of the BACnet Notification Sink.

✓ ℹ	12.11.2014 14:03:22	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ●	12.11.2014 14:03:36	Beckhoff_927609 - BI_0 : PLC User Reset

Fig. 25: Fig. 2: Online view of the BACnet notification sink with acknowledgement as example.

```
stEvent
    .nProcessID = 10000
    .nDeviceID = 34482041
    .tObjectID = 12582912
    .stDateTime
        .stDate
            .nYear = 114
            .nMonth = 11
            .nDay = 12
            .nDayOfWeek = 3
        .stTime
            .nHour = 14
            .nMinute = 3
            .nSecond = 36
            .nHundredths = 85
    .nNC = 0
    .nPriotity = 127
    .eEventType = BACnetEventType_change_of_state
```

Fig. 26: Fig. 3: Time stamp from event parameters in the PLC.

⊟ EventTimeStamps	130	{12.11.2014 14:03:17;12.11.2014 14:03:20;12.11....
⊞ [1] to_offnormal		12.11.2014 14:03:17
⊞ [2] to_fault		12.11.2014 14:03:20
⊞ [3] to_normal		12.11.2014 14:03:36

Fig. 27: Fig. 4: Time stamp in the object triggering the event; property Event_Time_Stamps

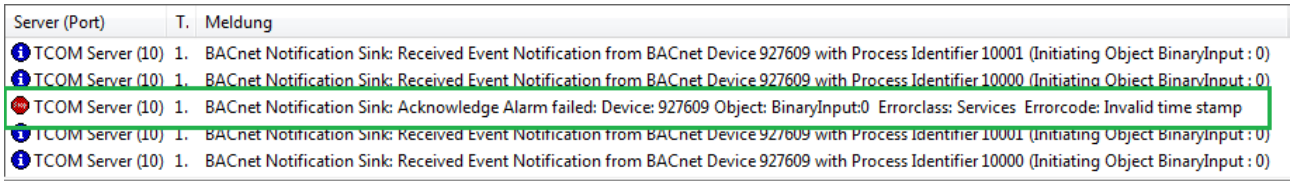


Fig. 28: Fig. 5: Error message in the TwinCAT System Manager logger. The time stamps of the property Event_Time_Stamps and the event parameters of the acknowledgment do not match.

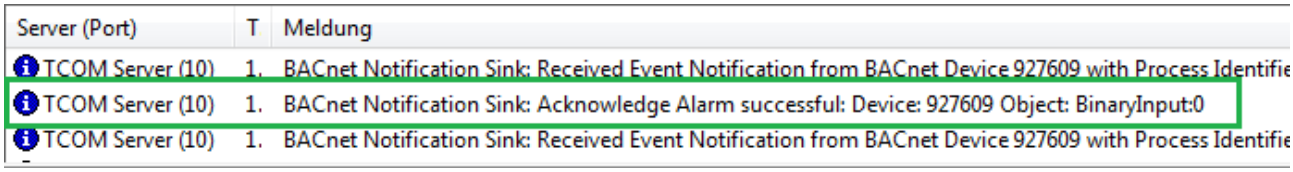


Fig. 29: Fig. 6: Message in the TwinCAT System Manager logger. Acknowledgement was successful. In the object generating the event (BI:0), the flag of the corresponding transition is reset to TRUE (property Acked_Transitions).

VAR_INPUT

```
bAcknowledge    : BOOL;
stAcknEvent     : ST_BACnet_NSinkEvent;
sMessage        : T_MaxString;
```

bAcknowledge: Edge *FALSE* → *TRUE* triggers sending if the acknowledgement.

stAcknEvent: Event parameters of the event to be acknowledged. The event parameters can be read from a BACnet Notification Sink with the aid of the function block FB_BACnet_NSinkReadEvent [▶ 264]. At the time of the acknowledgement, the read event no longer must be listed in the BACnet Notification Sink. In this way it is possible to buffer and display event parameters in the PLC and acknowledge them later.

sMessage: *Windows-1252*-coded message sent with the acknowledgement.

VAR_OUPUT

```
bDone           : BOOL;
bBusy           : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bDone: Read of the data completed successfully. **bDone** remains set until **bAcknowledge** is reset. If **bAcknowledge** is reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

VAR_IN_OUT

```
stAdsConn      : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stBACnetAds** of the corresponding NotificationSink function block.

Example: Function block call

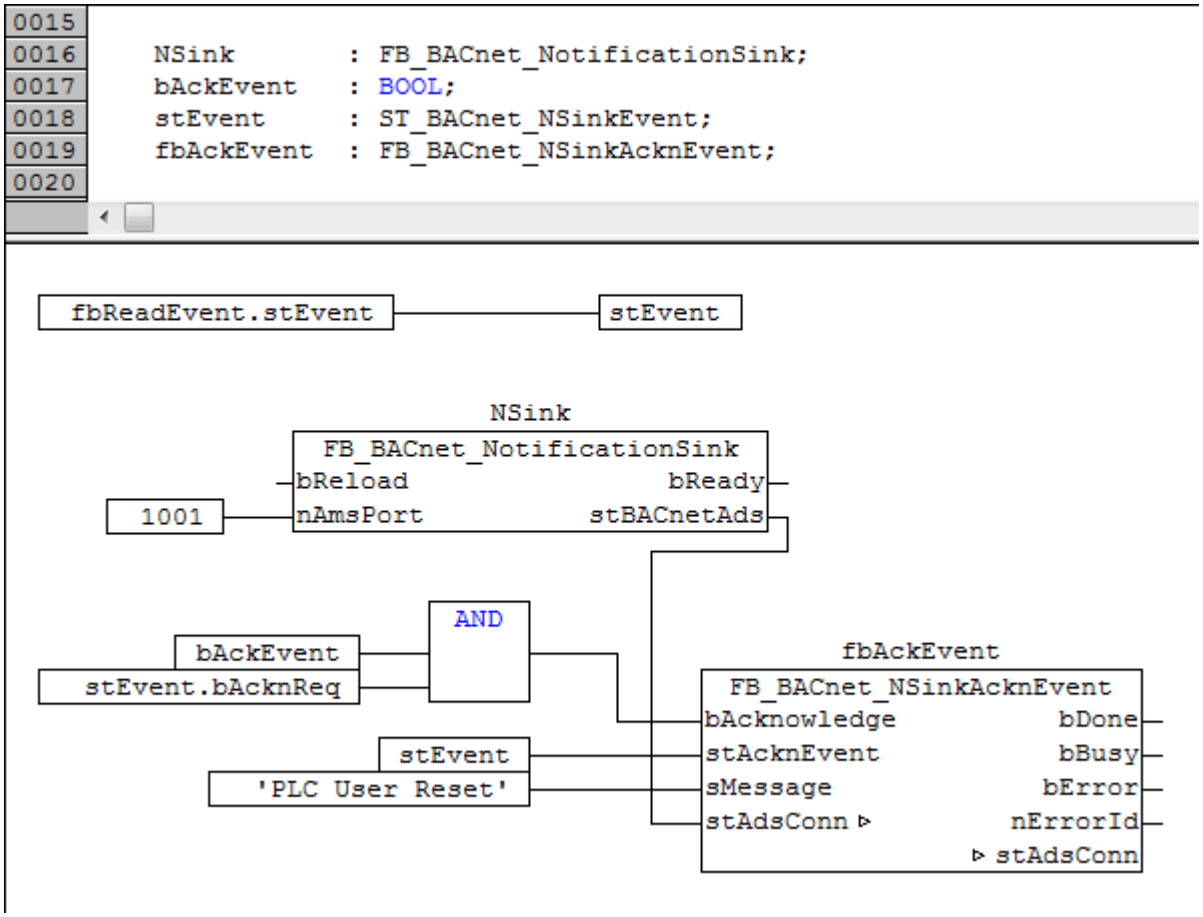
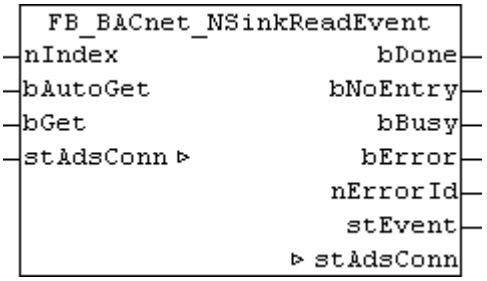


Fig. 30: Figure 7: Application example. Acknowledgment of a previously read event entry

(see FB_BACnet_NSinkReadEvent [▶ 264]) of a BACnet Notification Sink with AMS Port 1001.

The AMS port depends on the TwinCAT System Manager configuration of the corresponding BACnet device (see FB_BACnet_NotificationSink [▶ 258]).

4.5.3 FB_BACnet_NSinkReadEvent



Application

The function block can be used to read any entry from the BACnet Notification Sink. The Example [▶ 266] shows a possible configuration.

The input to be read is selected at input **nIndex**. The numbering matches the order in the buffer of the Notification Sink. In a ring buffer the numbering of incoming events starts at 0, in the event of a buffer overflow.

Within the block, the message text is decoded based on the string coding. The following encodings are supported: *UTF-8*, *UCS2*, *UCS4* and *ISO8859-1*. The string **sMessage** within the output structure **stEvent** is Windows-1252-coded (see also FB_BACnet_StringExtDecode [▶ 321]). If an error occurs during decoding, this is indicated at the **nErrorId** output.

The block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stBACnetAds** of the corresponding NotificationSink function block (FB_BACnet_NotificationSink [▶ 258]).

	Time	Notification
✓ ℹ	12.11.2014 10:15:28	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⛔	12.11.2014 10:15:31	Beckhoff_927609 - BI_0 : BI_0: Eingang ist in Fehler
✓ ⚠	12.11.2014 10:15:32	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:34	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:37	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:40	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:43	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ⛔	12.11.2014 10:15:46	Beckhoff_927609 - BI_0 : BI_0: Eingang ist in Fehler
✓ ℹ	12.11.2014 10:15:48	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:49	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:52	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:55	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS

Fig. 31: Figure 1: Online view of the BACnet Notification Sink.

VAR_INPUT

```
nIndex      : UDINT;
bAutoGet    : BOOL:=TRUE;
bGet        : BOOL;
```

nIndex: Index of the event entry to be read (0 to n).

bAutoGet: *TRUE* = the event entry is read automatically when the ADS connection or the input **nIndex** have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the event entry itself changes!

bGet: *FALSE* → *TRUE* = event entry is read once, irrespective of input **bAutoGet**.

VAR_OUPUT

```
bDone       : BOOL;
bNoEntry    : BOOL;
bBusy       : BOOL;
bError      : BOOL;
nErrorId    : UINT;
stEvent     : ST_BACnet_NSinkEvent;
```

bDone: Read of the data completed successfully. **bDone** remains set until to **bGet** and **bAutoGet** are reset or re-reading begins. If **bGet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle. **bDone** is only set if the event entry to be read is not empty.

bNoEntry: Reading completed, but without data - event entry under **nIndex** is empty. **bNoEntry** remains set until a new read operation starts. The output is used to detect the end of the event list.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet Globals [▶ 328] for an overview.

stEvent: Output structure with data of an event entry. See example for a comparison of the PLC data and the online event entry in the TwinCAT System Manager.

VAR_IN_OUT

```
stAdsConn       : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stBACnetAds** of the corresponding NotificationSink function block.

Example 1: Comparison of PLC and System Manager

```
.stEvent
    .nProcessID = 10000
    .nDeviceID = 34482041
    .tObjectID = 12582912
    .stDateTime
        .stDate
            .nYear = 114
            .nMonth = 11
            .nDay = 12
            .nDayOfWeek = 3
        .stTime
            .nHour = 10
            .nMinute = 16
            .nSecond = 57
            .nHundredths = 51
    .nNC = 0
    .nPriotity = 127
    .eEventType = BACnetEventType_change_of_state
    .eNotifyType = BACnetNotifyType_alarm
    .eFromState = BACnetEventState_state_normal
    .eToState = BACnetEventState_offnormal
    .sMessage = 'BI_0: Eingang ist AUS'
    .bAcknReq = TRUE
```

Fig. 32: Fig. 2: Example of an event entry in the PLC

12.11.2014 10:16:57	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
processIdentifier	10000
initiatingDeviceIdentifier	Device:927609
eventObjectIdentifier	BinaryInput:0
timeStamp	12.11.2014 10:16:57
notificationClass	0
priority	127
eventType	change_of_state
messageText	BI_0: Eingang ist AUS
notifyType	alarm
ackRequired	☑
fromState	state_normal
toState	offnormal
eventValues	((32772;inactive))
sequenceId	1277

Fig. 33: Fig. 3: Example of an event entry in the TwinCAT System Manager

Example 2: Function block call

```
0001 PROGRAM MAIN
0002 VAR
0003     bReadRel     : BOOL := TRUE;
0004     NSink        : FB_BACnet_NotificationSink;
0005     fbReadEvent  : FB_BACnet_NSinkReadEvent;
0006     tonReadEvent: TON;
```

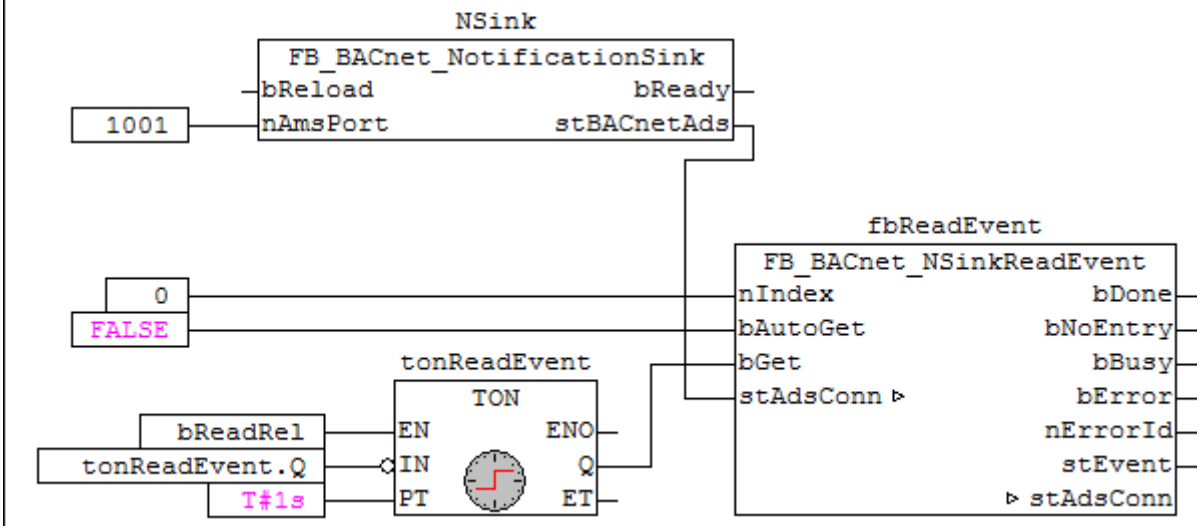
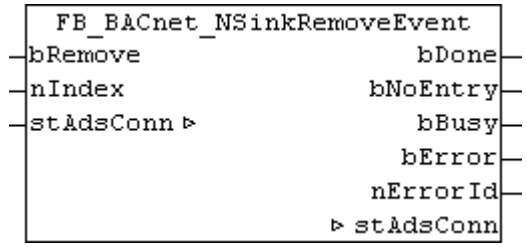


Fig. 34: Figure 4: Application example. Cyclic reading of the first event entry of a BACnet Notification Sink with AMS port 1001.

The AMS port depends on the TwinCAT System Manager configuration of the corresponding BACnet device (see FB_BACnet_NotificationSink [▶ 258]).

4.5.4 FB_BACnet_NSinkRemoveEvent



Application

The function block can be used to delete any entry from the BACnet Notification Sink. The Example shows a possible configuration.

The input to be deleted is selected at input **nIndex**. The numbering matches the order in the buffer of the Notification Sink. In a ring buffer the numbering of incoming events starts at 0, in the event of a buffer overflow.

The block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stBACnetAds** of the corresponding NotificationSink function block (FB_BACnet_NotificationSink [▶ 258]).

	Time	Notification
✓ ℹ	12.11.2014 10:15:28	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⛔	12.11.2014 10:15:31	Beckhoff_927609 - BI_0 : BI_0: Eingang ist in Fehler
✓ ⚠	12.11.2014 10:15:32	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:34	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:37	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:40	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:43	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ⛔	12.11.2014 10:15:46	Beckhoff_927609 - BI_0 : BI_0: Eingang ist in Fehler
✓ ℹ	12.11.2014 10:15:48	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:49	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS
✓ ℹ	12.11.2014 10:15:52	Beckhoff_927609 - BI_0 : BI_0: Eingang ist EIN
✓ ⚠	12.11.2014 10:15:55	Beckhoff_927609 - BI_0 : BI_0: Eingang ist AUS

Fig. 35: Figure 1: Online view of the BACnet Notification Sink.

VAR_INPUT

```
bRemove     : BOOL;
nIndex      : UDINT;
```

bRemove: *FALSE* → *TRUE* = event entry is deleted.

nIndex: Index of the event entry to be deleted (0 to n).

VAR_OUPUT

```
bDone       : BOOL;
bNoEntry    : BOOL;
bBusy       : BOOL;
bError      : BOOL;
nErrorId    : UINT;
```

bDone: Read of the data completed successfully. **bDone** remains set until **bRemove** is reset. If **bRemove** is reset before **bDone** is active, **bDone** is set for one cycle.

bNoEntry: No entry for deletion exists - event entry under **nIndex** is empty. **bNoEntry** remains set until a new delete operation starts. The output is used to detect the end of the event list.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

VAR_IN_OUT

```
stAdsConn       : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stBACnetAds** of the corresponding NotificationSink function block.

Example: Function block call

```
0015
0016    NSink        : FB_BACnet_NotificationSink;
0017    fbDelEvent   : FB_BACnet_NSinkRemoveEvent;
0018    bDelEvent    : BOOL;
0019
```

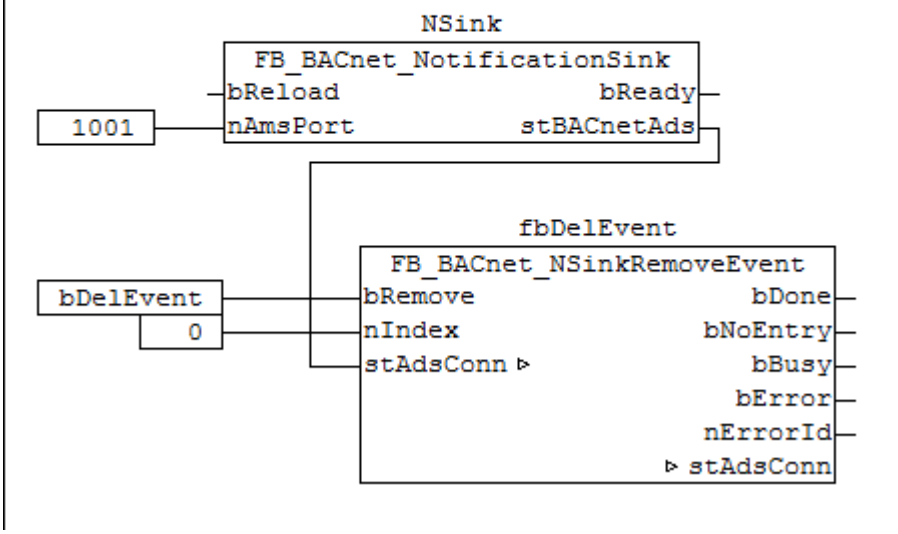
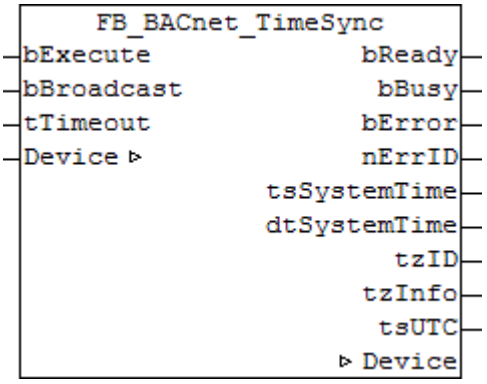


Fig. 36: Figure 7: Application example. Deleting of an event entry of a BACnet Notification Sink with AMS port 1001.

The AMS port depends on the TwinCAT System Manager configuration of the corresponding BACnet device (see FB_BACnet_NotificationSink [▶ 258]).

4.5.5 FB_BACnet_TimeSync



Application

The function block is called cyclically and outputs the current local time and the current UTC time. When an edge is applied to the input **bExecute**, the function block sends the time to the BACnet stack and the BACnet network as a broadcast message (time master) - provided the input **bBroadcast** is set; otherwise, only the local BACnet stack is synchronized.

The function includes the function block FB_LocalSystemTime for reading the local time.

This function block can be used with BACnet stack Revision 6. However, under Revision 6 a broadcast is always sent (input **bBroadcast** is disabled under Revision 6).

A time synchronization overview is shown below:

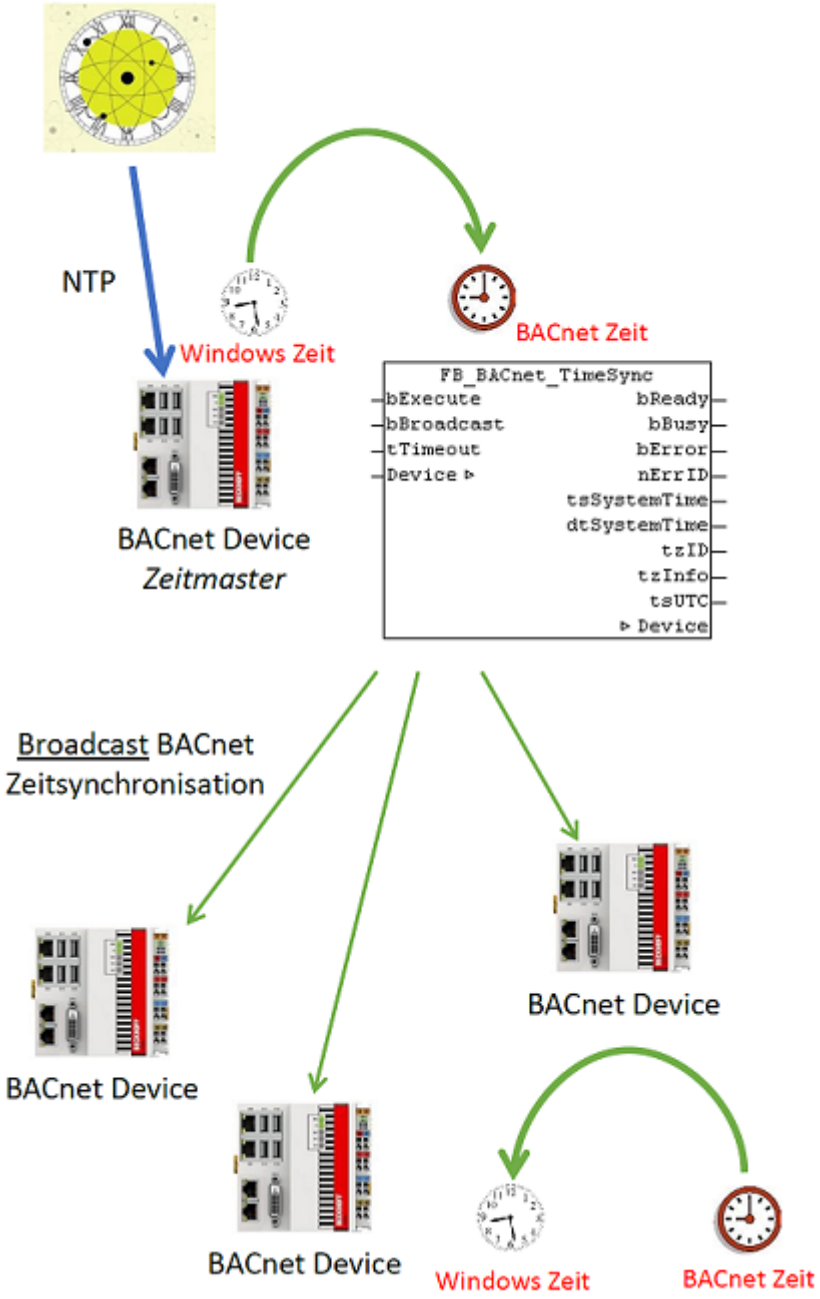


Fig. 37: Figure 1: Example for time synchronization sequence in the BACnet network:

1. An external time source (e.g. NTP) synchronizes the operating system time of the *time master* system (NTP → operating system time)
2. If **bExecute** = *TRUE*, the function block FB_BACnet_TimeSync of the *time master* system synchronizes the BACnet stack with the current operating system time (operating system time → BACnet time)
3. If the input **bBroadcast** is set at the function block, a BACnet broadcast time synchronization message is sent to the network
4. BACnet devices in network adjust the time accordingly (BACnet time → operating system time)

Time synchronization typically takes place at intervals of several hours or daily. Times between 23:00 and 03:00 should be avoided, in order to avoid potential overlap with automatic summer/winter time changeover of the devices to be synchronized.

Application information:

A distinction is made between two modes for time synchronization:

1. Each BACnet device is connected to an external time source (e.g. NTP). FB_BACnet_TimeSync must be executed on each BACnet controller. **bBroadcast** always has the value *FALSE* and **bExecute** is triggered at regular intervals.
2. A device is connected to an external time source (e.g. NTP). Then FB_BACnet_TimeSync must be executed on this device with **bBroadcast** = *TRUE* and **bExecute** must be triggered regularly.
   All other BACnet devices execute FB_BACnet_TimeSync with **bBroadcast** = *FALSE* and **bExecute** = *FALSE*, are synchronized via BACnet and automatically adjust their local operating system time.

VAR_INPUT

```
bExecute   : BOOL;
bBroadcast : BOOL:=TRUE;
tTimeout   : TIME:=BACnet_ADSTimeOut;
```

bExecute: The synchronization starts with a rising edge.

bBroadcast: *TRUE* → additionally sends a broadcast time synchronization to the BACnet network; *FALSE* → only local synchronization (Revision 12 and higher)

tTimeout: Optional input, monitoring time for ADS access (default: see BACnet_ADSTimeOut [▶ 328]).

VAR_OUPUT

```
bReady       : BOOL;
bBusy        : BOOL;
bError       : BOOL;
nErrID       : UDINT;
tsSystemTime : TIMESTRUCT;
dtSystemTime : ST_BACnet_DateTime;
tzID         : E_TimeZoneID;
tzInfo       : ST_TimeZoneInformation;
tsUTC        : TIMESTRUCT;
```

bReady: Time output (**tsSystemTime**, **dtSystemTime**, **tzID**, **tzInfo** and **tsUTC**) is valid

bBusy: Block is busy (time synchronization is active)

bError: Error during processing.

nErrID: ADS error code.

tsSystemTime: Local operating system time (Windows time) as typical PLC data structure

dtSystemTime: Local operating system time (Windows time) as typical BACnet data structure

tzID: ID of the current time zone

tzInfo : Information regarding the current time zone

tsUTC: Operating system time as UTC timestamp (GMT or coordinated world time) as typical PLC data structure

VAR_IN_OUT

```
Device : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block.

Example

```
0001 PROGRAM MAIN_PROG
0002 VAR
0003     Device         : FB_BACnet_Device;
0004     fbTimeSync     : FB_BACnet_TimeSync;
0005     bTimeSyncExec  : SR;
0006 END_VAR
```

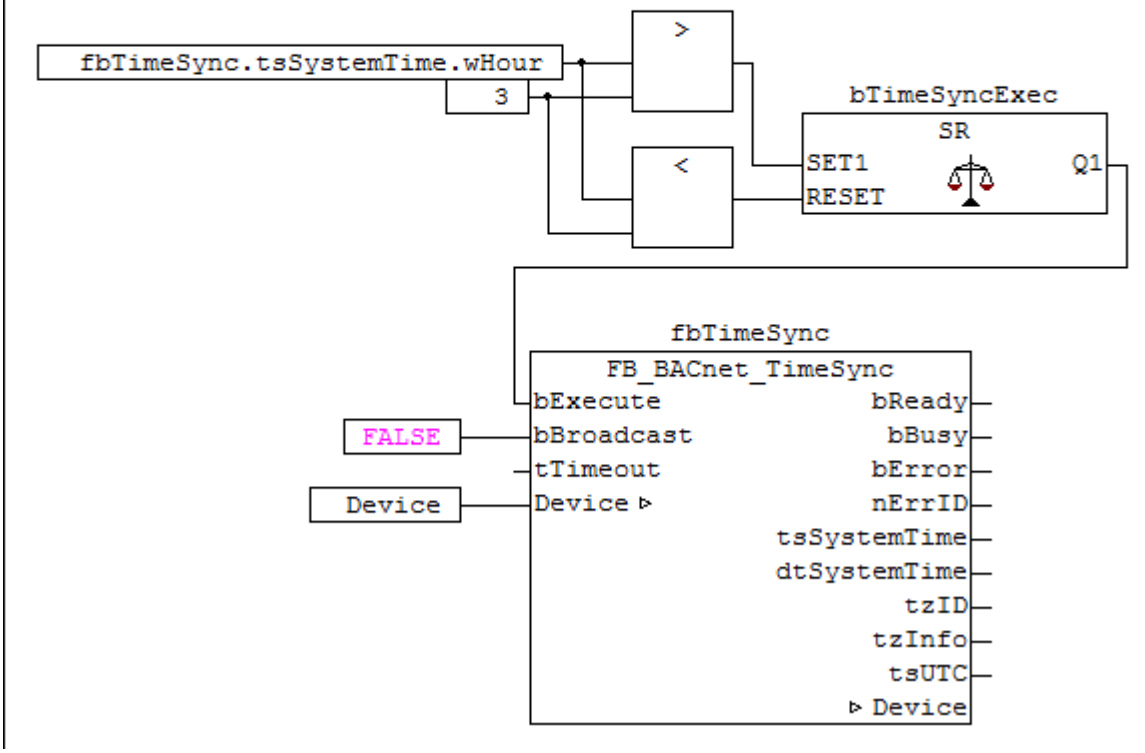
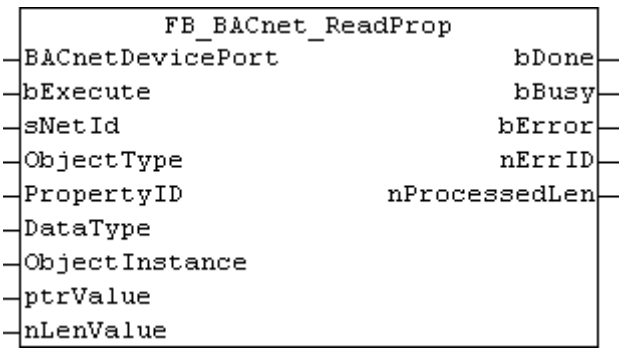


Fig. 38: Fig. 2: Example for cyclic synchronization of the local time (no broadcast).

bExecute is set if 03:00 is exceeded and remains active until the time reaches 00:00 → daily synchronization.

4.5.6 FB_BACnet_ReadProp

Function block for access to BACnet properties via ADS. All BACnet properties of server and client objects that are shown in the Online tab of the respective BACnet object can be read or written via ADS. If a client (remote BACnet server) is accessed via ADS, the ADS accesses are converted to BACnet queries and sent to the remote server.



Application

The function block instance is created by the PLC program and called cyclically. The input **BACnetDevicePort** and **sNetId** must be assigned the appropriate AMS port and NetID. The AMS port and NetID for each BACnet server and client can be queried via the corresponding BACnet device function block.

The AMS NetID does not match the local AMS NetID and always has to be specified - empty strings cannot be used!

The current AMS NetID and AMS port are output by the corresponding FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]:

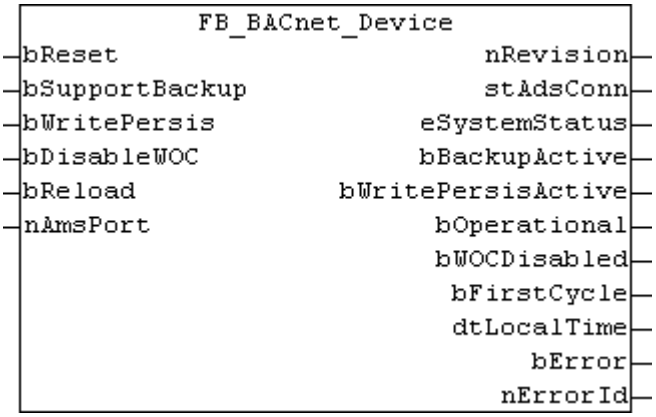


Fig. 39: Figure 1: FB_BACnet_Device → stAdsConn outputs the current ADS connection data.

AMS NetID and AMS port in the output structure **stAdsConn** of the device function block.

```
TYPE ST_BACnet_AdsConnection:
  STRUCT
    bValid     : BOOL;
    nReload    : USINT;
    sAmsNetId  : T_AmsNetId;
    nAmsPort   : T_AmsPort;
    bServer    : BOOL;
    bClient    : BOOL;
    bNSink     : BOOL;
    nDeviceId  : UDINT;
  END_STRUCT
END_TYPE
```

In addition, the AMS port and the AMS NetID can be displayed in the TwinCAT System Manager (see Figures 3 and 4). Ports for servers and clients may vary, depending on the configuration.

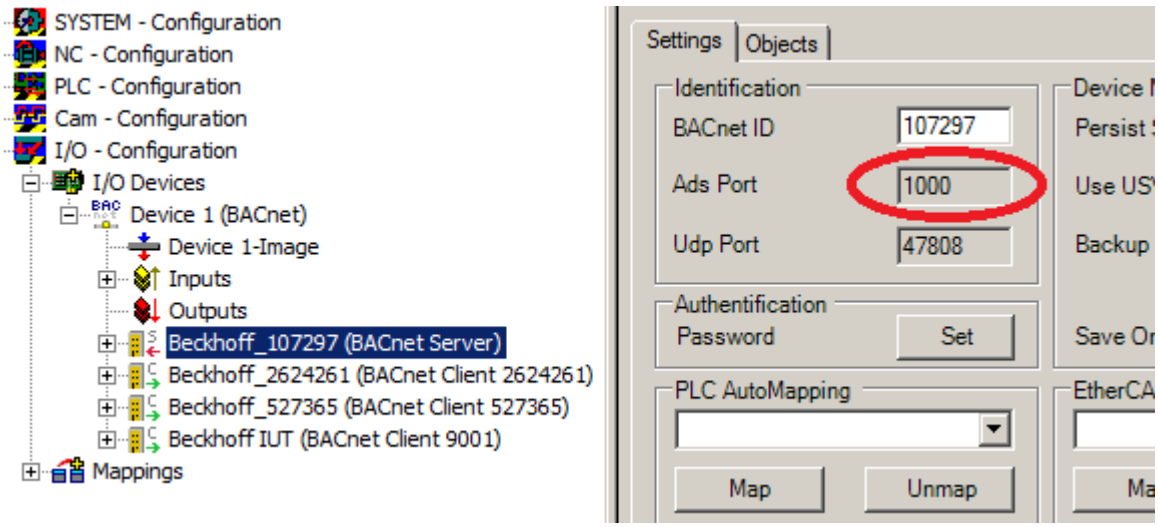


Fig. 40: Figure 3: AMS port of the local BACnet server in the System Manager

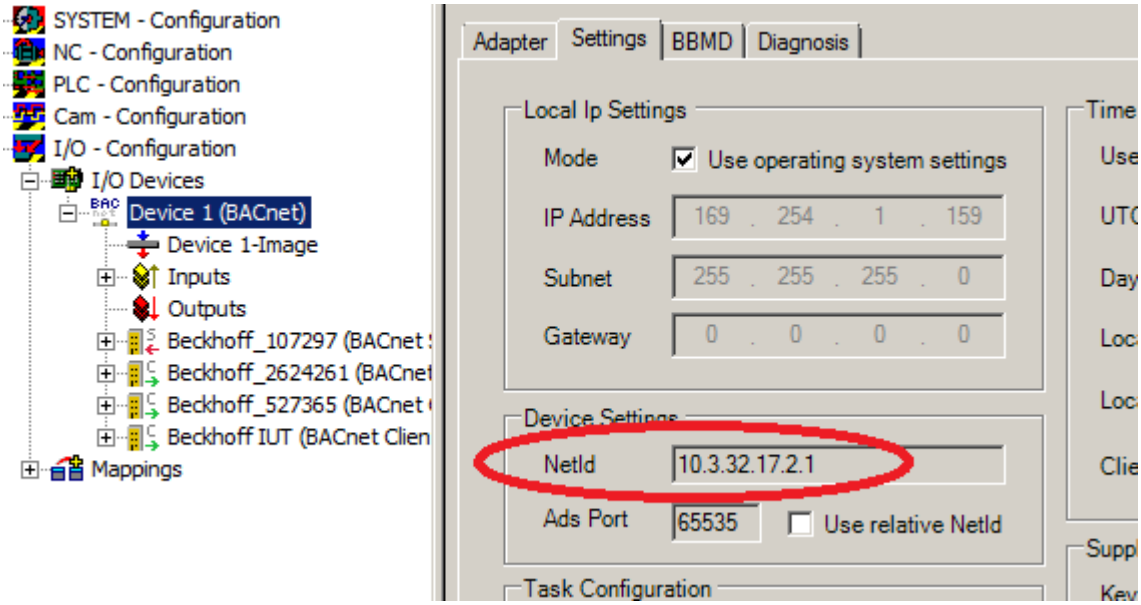


Fig. 41: Figure 4: AMS NetID of the BACnet device

VAR_INPUT

```
BACnetDevicePort  : UINT:=16#FFFF;
bExecute          : BOOL;
sNetId            : T_AmsNetId;
ObjectType        : E_BACNETOBJECTTYPE;
PropertyID        : E_BACNETPROPERTYIDENTIFIER;
DataType          : E_BACNETDATATYPES;
ObjectInstance    : DINT;
ptrValue          : POINTER TO BYTE;
nLenValue         : UDINT;
```

BACnetDevicePort: AMS port under which the desired object was created.

bExecute: Rising edge at the input starts the read process. Falling edge deletes the output **bDone**.

sNetId: AMS NetID of the BACnet device (adapter) under which the server or client was configured.

ObjectType: Enumeration of the object type (AnalogValue, BinaryInput...).

PropertyID: Enumeration of the property ID (*Present_Value*, *Status_Flags*...).

DataType: Data type of the property (Bool, BinaryPV, Real, Unsigned Integer...).

ObjectInstance: Object number of the BACnet object (the lower 22 bits of the *Object_Identifier*).

ptrValue: Pointer to the variable in which the read data are to be stored (can be determined with ADR()).

nLenValue: Byte length of the variable in which the read data are to be stored (can be determined with SIZEOF()).

VAR_OUPUT

```
bDone          : BOOL;
bBusy          : BOOL;
bError         : BOOL;
nErrID         : UDINT;
nProcessedLen  : UDINT;
```

bDone: Read of the data completed successfully. **bDone** remains set until **bExecute** is reset. If **bExecute** is reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrID: ADS error code.

nProcessedLen: Byte length of the read data. The length may differ from the target length **nLenValue** (smaller/equal). The gap between the input data length **nLenValue** and the actual read data length **nProcessedLen** is filled with zeros in the transferred data range.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbReadProp   : FB_BACnet_ReadProp;
0005     ePvBI0       : E_BACNETBINARYPV;
0006 END_VAR
0007
```

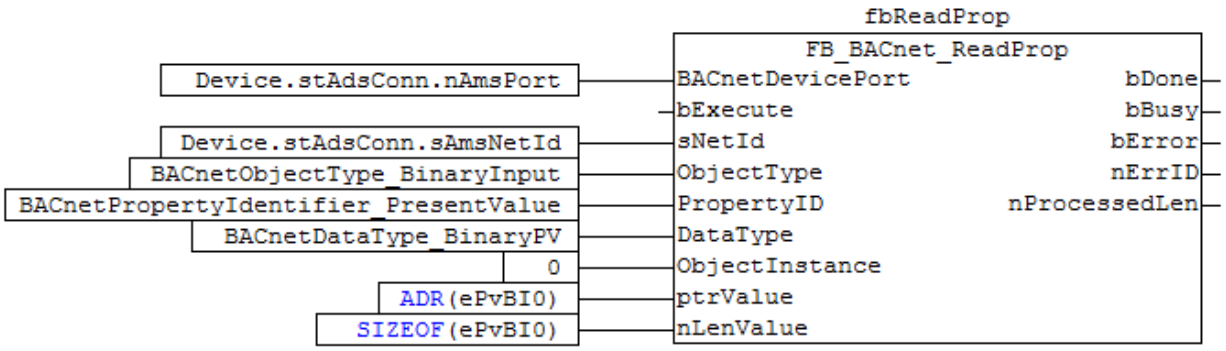
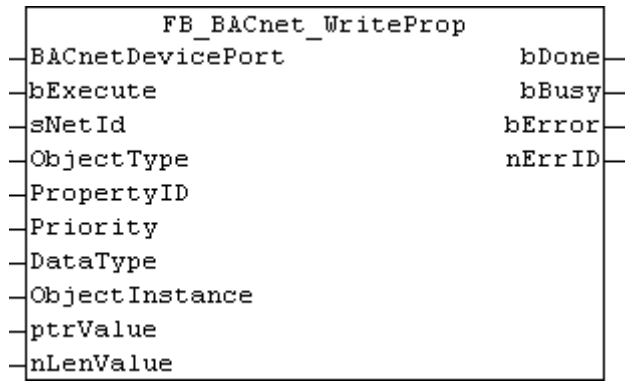


Fig. 42: Fig. 5: Application example. Reading of the property Present_Value of BACnet binary input object.

4.5.7 FB_BACnet_WriteProp

Function block for access to BACnet properties via ADS. All BACnet properties of server and client objects that are shown in the Online tab of the respective BACnet object can be read or written via ADS. If a client (remote BACnet server) is accessed via ADS, the ADS accesses are converted to BACnet queries and sent to the remote server.



Application

The function block instance is created by the PLC program and called cyclically. The input **BACnetDevicePort** and **sNetId** must be assigned the appropriate AMS port and AMS NetID. The AMS port and AMS NetID for each BACnet server and client can be queried via the corresponding BACnet device function block.

The AMS NetID does not match the local AMS NetID and always has to be specified - empty strings cannot be used!

The current AMS NetID and AMS port are output by the corresponding FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]:

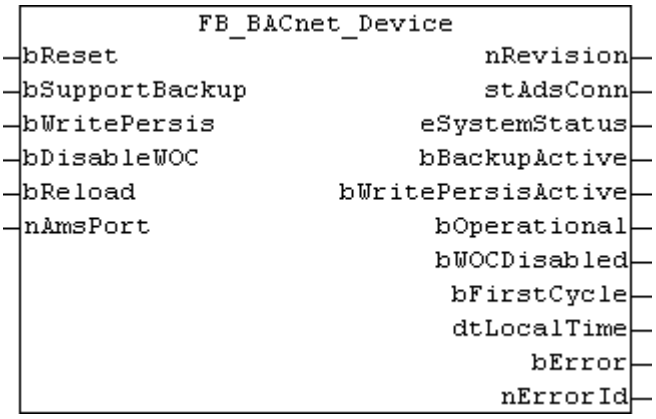


Fig. 43: Figure 1: FB_BACnet_Device → stAdsConn outputs the current ADS connection data.

AMS NetID and AMS port in the output structure stAdsConn of the device function block.

```
TYPE ST_BACnet_AdsConnection :
  STRUCT
    bValid    : BOOL;
    nReload   : USINT;
    sAmsNetId : T_AmsNetId;
    nAmsPort  : T_AmsPort;
    bServer   : BOOL;
    bClient   : BOOL;
    bNSink    : BOOL;
    nDeviceId : UDINT;
  END_STRUCT
END_TYPE
```

In addition, the AMS port and the AMS NetID can be displayed in the TwinCAT System Manager (see Figures 3 and 4). Ports for servers and clients may vary, depending on the configuration.

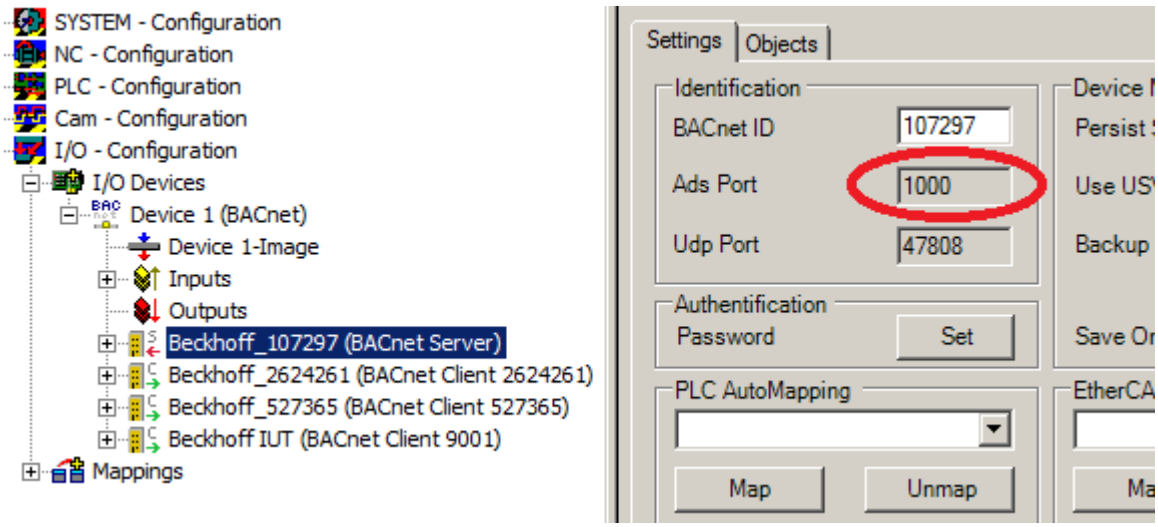


Fig. 44: Figure 3: AMS port of the local BACnet server in the System Manager

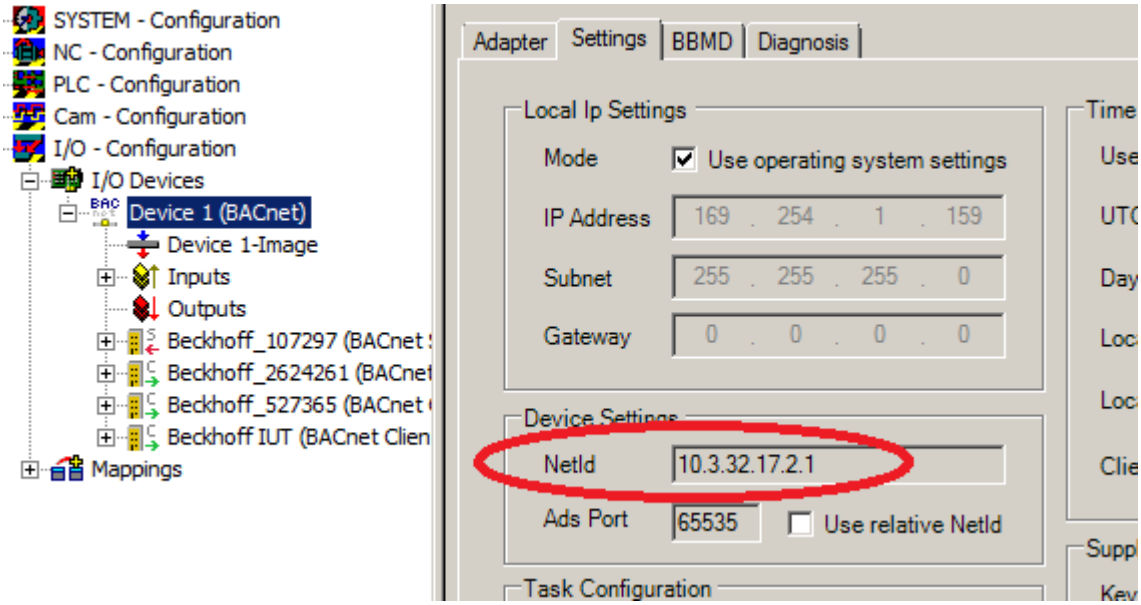


Fig. 45: Figure 4: AMS NetID of the BACnet device

VAR_INPUT

```
BACnetDevicePort  : UINT:=16#FFFF;
bExecute          : BOOL;
sNetId            : T_AmsNetId;
ObjectType        : E_BACNETOBJECTTYPE;
PropertyID        : E_BACNETPROPERTYIDENTIFIER;
Priority          : E_BACNETPRIORITY;
DataType          : E_BACNETDATATYPES;
ObjectInstance    : UDINT;
ptrValue          : POINTER TO BYTE;
nLenValue         : UDINT;
```

BACnetDevicePort: AMS port under which the desired object was created.

bExecute: Rising edge at the input start the write process. Falling edge deletes the output **bDone**.

sNetId: AMS NetID of the BACnet device (adapter) under which the server or client was configured.

ObjectType: Enumeration of the object type (AnalogValue, BinaryInput...).

PropertyID: Enumeration of the property ID (*Present_Value*, *Status_Flags*...).

Priority: Enumeration of the write access priority. The priority is required for write accesses to prioritisable properties (e.g. *Present_Value*). Access with priority x then generates an entry at location x in the corresponding *Priority_Array*.

DataType: Data type of the property (Bool, BinaryPV, Real, Unsigned Integer...).

ObjectInstance: Object number of the BACnet object (the lower 22 bits of the *Object_Identifier*).

ptrValue: Pointer to the variable in which the data are to be stored (can be determined with ADR()).

nLenValue: Byte length of the variable in which the data are to be stored (can be determined with SIZEOF()).

VAR_OUPUT

```
bDone          : BOOL;
bBusy          : BOOL;
bError         : BOOL;
nErrID         : UDINT;
```

bDone: Writing of the data completed successfully. **bDone** remains set until **bExecute** is reset. If **bExecute** is reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrID: ADS error code.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbWriteProp  : FB_BACnet_WriteProp;
0005     ePvBO0       : E_BACNETBINARYPV;
0006 END_VAR
0007
```

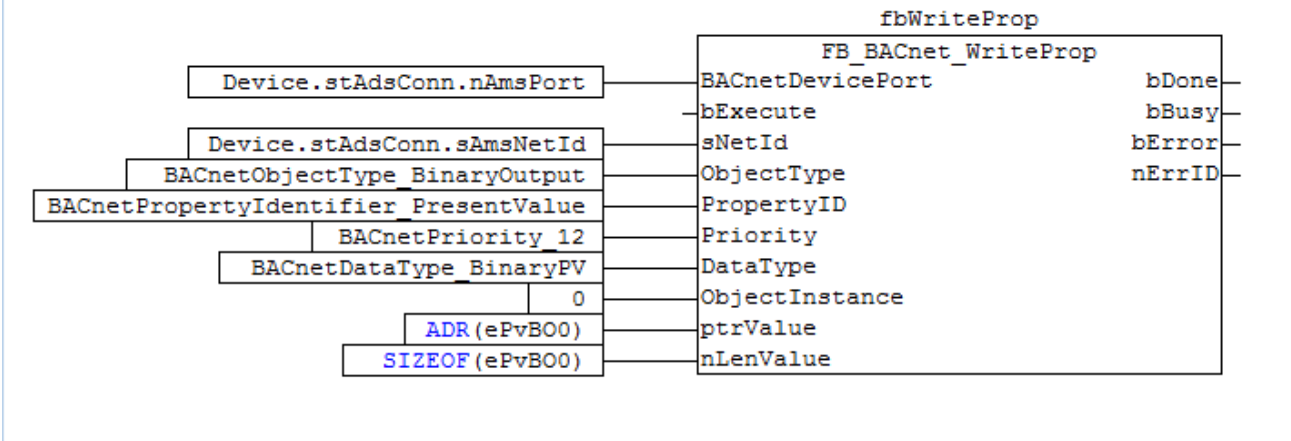
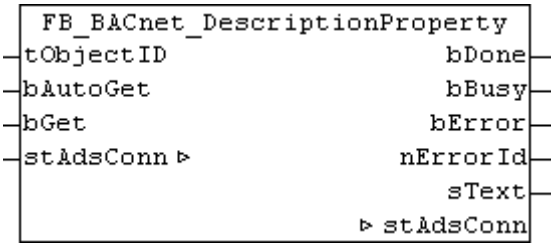


Fig. 46: Fig. 5: Application example. Writing of the property Present_Value of BACnet binary output object.

4.5.8 FB_BACnet_DescriptionProperty



Application

The function block can be used to read the property *Description* of any BACnet object. The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 281] shows a possible configuration.

Within the block, the property is decoded based on the string coding. The following property encodings are supported: *UTF-8*, *UCS2*, *UCS4* and *ISO8859-1*. The output string **sText** is Windows-1252-coded (see also FB_BACnet_StringExtDecode [▶ 321]). If an error occurs during decoding of the property, this is indicated at the **nErrorId** output.

The block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

VAR_INPUT

```
tObjectID  : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet   : BOOL:=TRUE;
bGet       : BOOL;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = *Property* is read once, irrespective of input **bAutoGet**.

VAR_OUPUT

```
bDone      : BOOL;
bBusy      : BOOL;
bError     : BOOL;
nErrorId   : UINT;
sText      : T_MaxString;
```

bDone: Read of the data completed successfully. **bDone** remains set until to **bGet** and **bAutoGet** are reset or re-reading begins. If **bGet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

sText: *Description* as Windows-1252-coded string.

VAR_IN_OUT

```
stAdsConn       : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbDesc       : FB_BACnet_DescriptionProperty;
0005     tonPropRead : TON;
0006 END_VAR
```

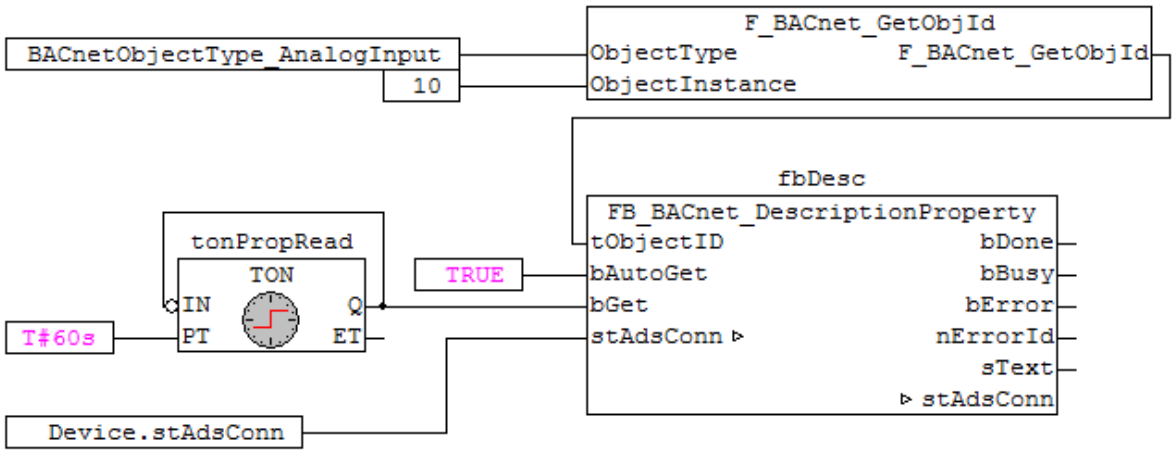
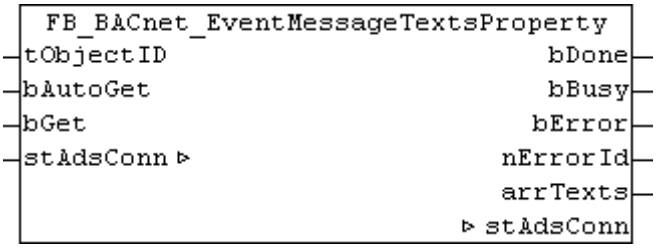


Fig. 47: Figure 1: Application example. Cyclic reading of the property Description of a BACnet analog input object (AnalogInput: 10).

4.5.9 FB_BACnet_EventMessageTextsProperty



Application

The function block can be used to read the property *Event_Message_Texts* of any BACnet object. The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 283] shows a possible configuration.

Within the block, the property is decoded based on the string coding. The following property encodings are supported: *UTF-8*, *UCS2*, *UCS4* and *ISO8859-1*. The output strings in the array **arrTexts** are Windows-1252-coded (see also FB_BACnet_StringExtDecode [▶ 321]). If an error occurs during decoding of the property, this is indicated at the **nErrorId** output.

The block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

VAR_INPUT

```
tObjectID  : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet   : BOOL:=TRUE;
bGet       : BOOL;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = *Property* is read once, irrespective of input **bAutoGet**.

VAR_OUPUT

```
bDone      : BOOL;
bBusy      : BOOL;
bError     : BOOL;
nErrorId   : UINT;
arrTexts   : ARRAY [0..2] OF T_MaxString;
```

bDone: Read of the data completed successfully. **bDone** remains set until to **bGet** and **bAutoGet** are reset or re-reading begins. If **bGet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

arrTexts: Message texts as Windows-1252-coded strings.

VAR_IN_OUT

```
stAdsConn      : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbMsgTexts   : FB_BACnet_EventMessageTextsProperty;
0005     tonPropRead : TON;
0006 END_VAR
```

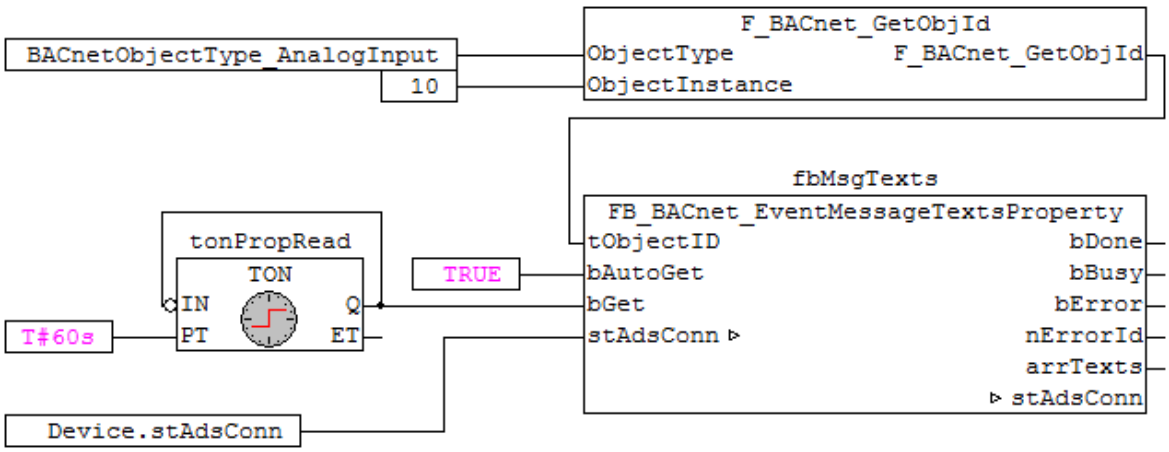
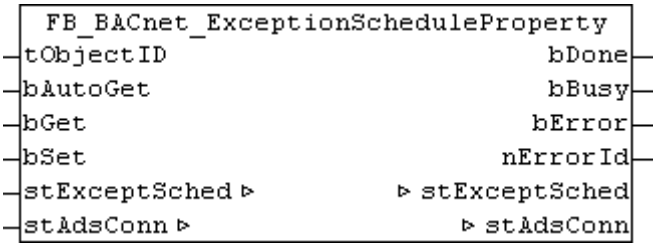


Fig. 48: Figure 1: Application example. Cyclic reading of the property Event_Message_Texts of a BACnet analog input object (AnalogInput: 10).

4.5.10 FB_BACnet_ExceptionScheduleProperty



Application

The function block can be used to read the property *Exception_Schedule* of any BACnet object (typically BACnet schedule object). The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 285] shows a possible configuration.

The data to be read or written is stored in the same input/output data structure (see also ST_BACnet_ExceptionScheduleBool [▶ 359]).

The only data types supported by the PLC function block for entries in the property *Exception_Schedule* are *Bool* and *Null*! Entries with other data types are ignored or deleted on write access to the property.

ExceptionSchedule	38	4 Elements	BACnetSpecialEventList
[1]		(Calendar:0;1;{(18:00:00;True);(06:00:00;False)})	calendarReference
calendarReference		Calendar:0	BACnetObjectIdentifier
eventPriority		1	UnsignedInteger
listOfTimeValues		{(18:00:00;True);(06:00:00;False)}	BACnetTimeValueList
[1]		(18:00:00;True)	BACnetTimeValue
time		18:00:00	BACnetTime
value		True	BoolValue
[2]		(06:00:00;False)	BACnetTimeValue
time		06:00:00	BACnetTime
value		False	BoolValue

Fig. 49: Figure 1: Example entry of the property Exception_Schedule with data type Bool.

The function block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device function block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

The ADS buffer from the global variables is used for buffering the ADS data (see ST_BACnet_GlobalAdsBuffer [▶ 360]).

VAR_INPUT

```
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet        : BOOL:=TRUE;
bGet            : BOOL;
bSet            : BOOL;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = Property is read once, irrespective of input **bAutoGet**.

bSet : *FALSE* --> *TRUE* = property is written.

VAR_OUPUT

```
bDone           : BOOL;
bBusy           : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bDone: Reading or writing of the data completed successfully. **bDone** set remains set until **bGet**, **bSet** and **bAutoGet** are reset, or until a new read or write operation starts. If **bGet**, **bSet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

VAR_IN_OUT

```
stExceptSched  : ST_BACnet_ExceptionScheduleBool;
stAdsConn      : ST_BACnet_AdsConnection;
```

stExceptSched: Input/output data structure for the entries of the property *Exception_Schedule* with data type *Bool* or *Null*. The following section compares an entry in the PLC and the TwinCAT System Manager (see Example [▶ 285]).

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example Comparison of PLC and System Manager

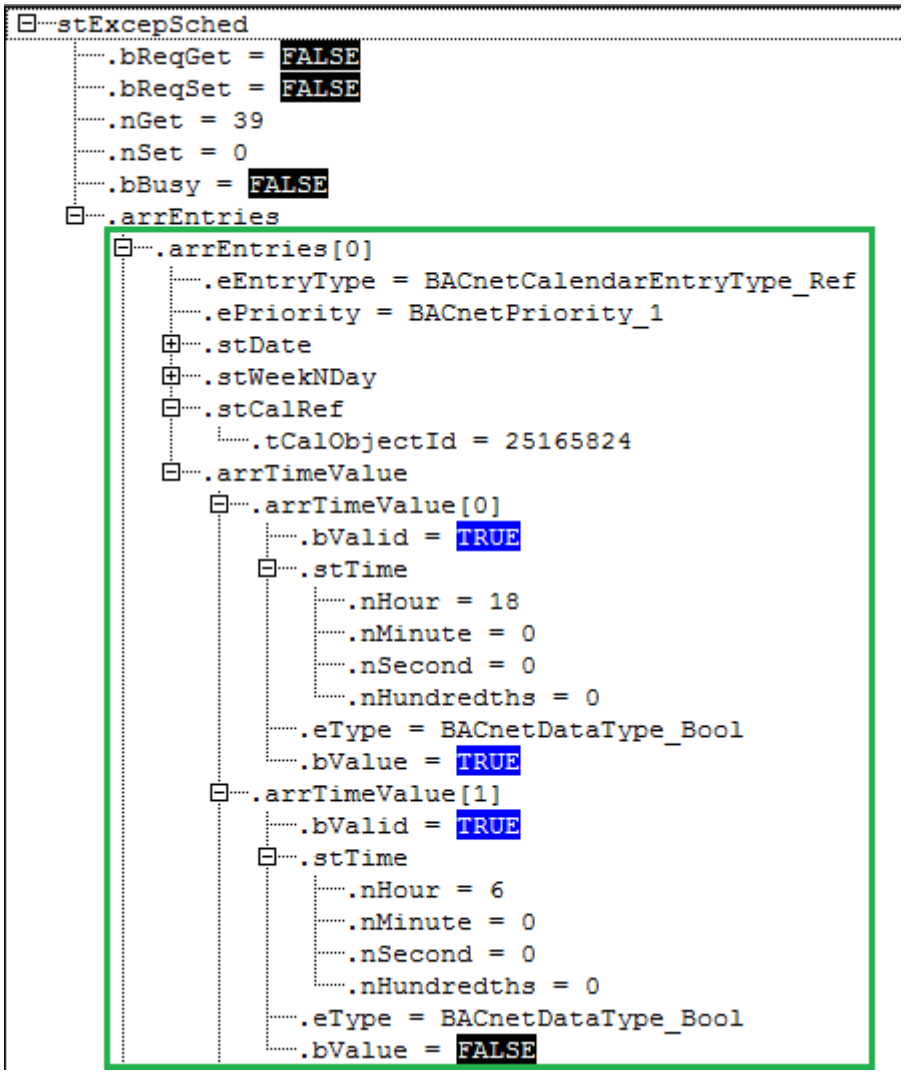


Fig. 50: Fig. 2: Example of a property entry in the PLC

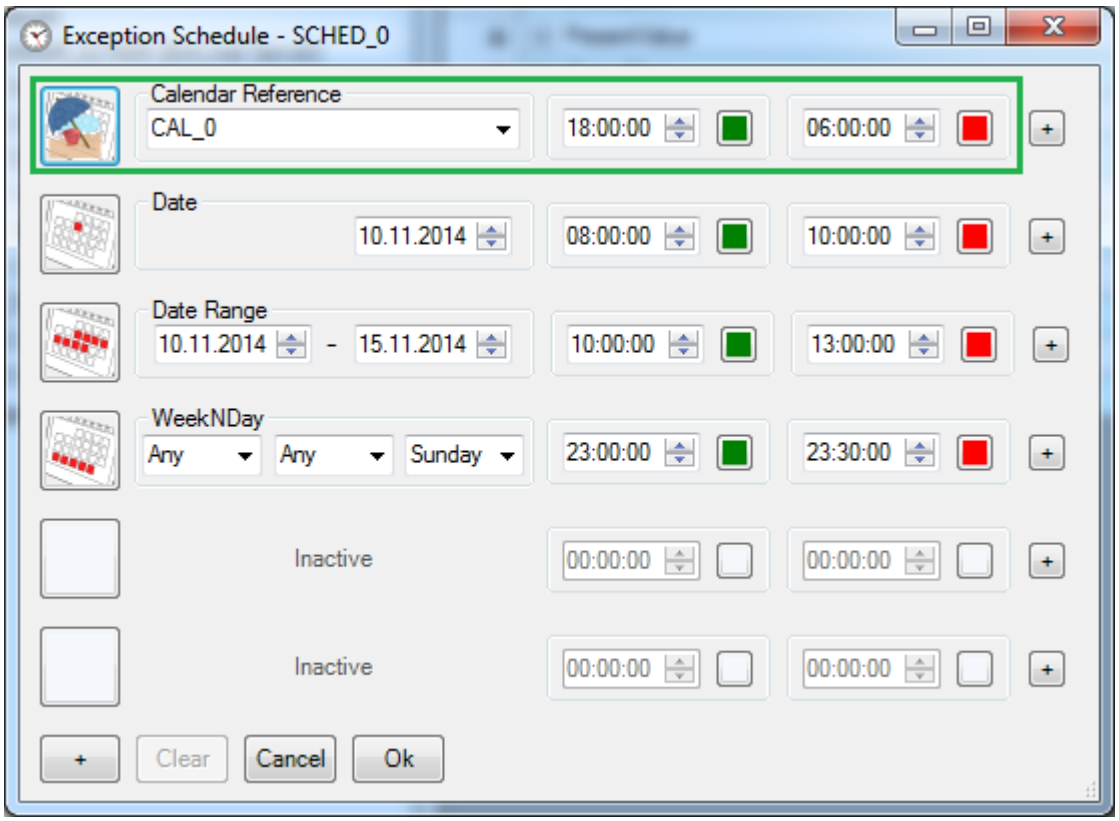


Fig. 51: Fig. 3: Example of a property entry in the TwinCAT System Manager

Example 2: Function block call

```
0001 PROGRAM MAIN
0002 VAR
0003     Device         : FB_BACnet_Device;
0004     fbExceptSched  : FB_BACnet_ExceptionScheduleProperty;
0005     stExcepSched   : ST_BACnet_ExceptionScheduleBool;
0006
0007     bWriteScheduleData: BOOL;
0008     bReadScheduleData: BOOL;
0009 END_VAR
```

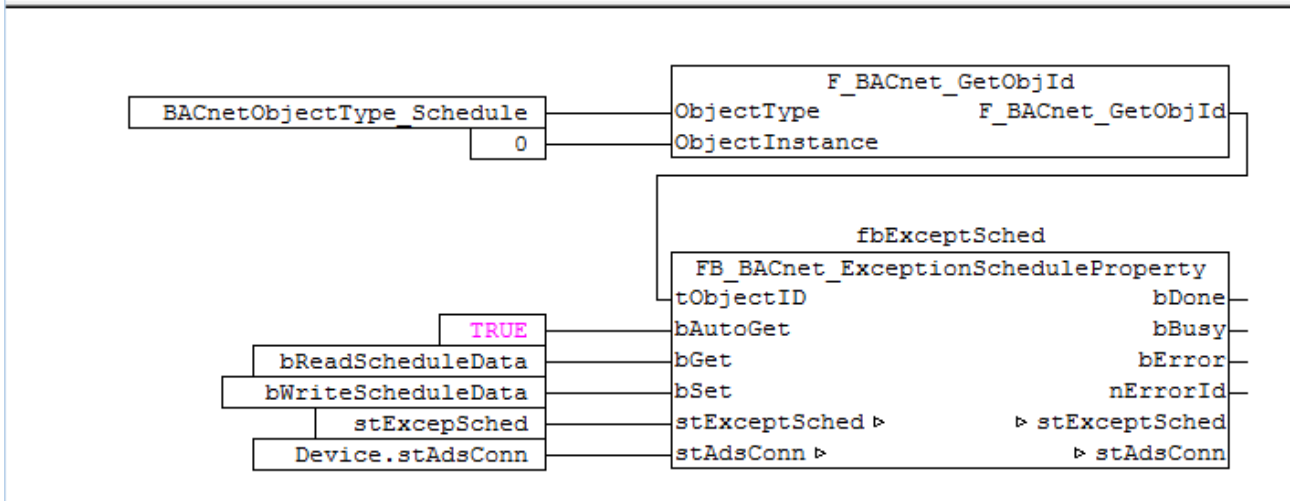
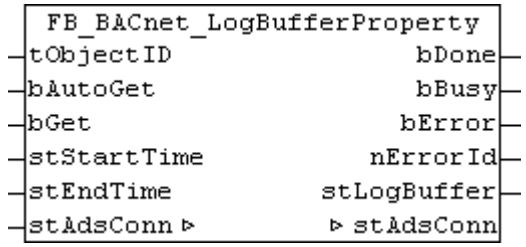


Fig. 52: Figure 4: Application example. Reading and writing of the property Exception_Schedule of a BACnet schedule object.

4.5.11 FB_BACnet_LogBufferProperty



Application

The function block can be used to read the property *Log_Buffer* of any BACnet object (typically BACnet TrendLog object). The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 289] shows a possible configuration.

The data to be read are stored in the output data structure **stLogBuffer** (see also ST_BACnet_LogBufferReal [▶ 362]).

The only data type supported by the PLC function block for entries in the property *Log_Buffer* is *Real*! Entries with other data types are ignored.

⊟ LogBuffer	131	100 Elements	BACnetLogRecordList
⊞ [1]		10.11.2014 12:19:44: {(log_disabled)}	BACnetLogRecord
⊞ [2]		10.11.2014 12:20:51: {(log_disabled)}	BACnetLogRecord
⊞ [3]		10.11.2014 12:21:02: {(log_enabled)}	BACnetLogRecord
⊟ [4]		10.11.2014 12:21:02: {28}	BACnetLogRecord
⊞ timestamp		10.11.2014 12:21:02	BACnetDateTime
⊟ logDatum		{28}	BACnetLogDatumList
⊟ [1]		28	realValue
realValue		28	Real
⊞ statusFlags	☑	0x00000004	BACnetStatusFlags
⊞ [5]		10.11.2014 12:21:03: {28,5}	BACnetLogRecord
⊞ [6]		10.11.2014 12:21:04: {29}	BACnetLogRecord

Fig. 53: Figure 1: Example entry of property Log_Buffer with data type Real.

The function block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device function block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

The ADS buffer from the global variables is used for buffering the ADS data (see ST_BACnet_GlobalAdsBuffer [▶ 360]).

VAR_INPUT

```
tObjectID    : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet     : BOOL:=TRUE;
bGet         : BOOL;
stStartTime  : ST_BACnet_DateTime := (
                  stDate := (nYear       := 16#FF,
                  nMonth      := 16#FF,
                  nDay    := 16#FF),
                  stTime := (nHour       := 16#FF,
                    nMinute     := 16#FF,
                    nSecond     := 16#FF,
```

```
                    nHundredths := 16#FF));
stEndTime    : ST_BACnet_DateTime := (
                  stDate := (nYear       := 16#FF,
                  nMonth      := 16#FF,
                  nDay    := 16#FF),
                  stTime := (nHour       := 16#FF,
                    nMinute     := 16#FF,
                    nSecond     := 16#FF,
                    nHundredths := 16#FF));
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = Property is read once, irrespective of input **bAutoGet**.

stStartTime, **:stEndTime** Only entries from the *Log_Buffer* with time stamps between the start and end times are output. The filter is disabled by default (placeholder 255 → undefined).

VAR_OUPUT

```
bDone        : BOOL;
bBusy        : BOOL;
bError       : BOOL;
nErrorId     : UINT;
stLogBuffer  : ST_BACnet_LogBufferReal;
```

bDone: Reading or writing of the data completed successfully. **bDone** set remains set until **bGet**, **bSet** and **bAutoGet** are reset, or until a new read or write operation starts. If **bGet**, **bSet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals for an overview.

stLogBuffer: Output data structure for entries of property *Log_Buffer* with data type *Real*. The following section compares two entries in the PLC and the TwinCAT System Manager (see Example).

VAR_IN_OUT

```
stAdsConn    : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example 1: Comparison of PLC and System Manager

```
.stLogBuffer
    .nGet = 1
    .sObjectDesc = 'TLOG_0'
    .nCount = 97
  .arrEntries
    .arrEntries[0]
        .nTimeStamp = 1863042261
      .stDateTime
        .stDate
        .stTime
            .nHour = 12
            .nMinute = 21
            .nSecond = 2
            .nHundredths = 91
        .fValue = 28
    .arrEntries[1]
        .nTimeStamp = 1863042271
      .stDateTime
        .stDate
        .stTime
            .nHour = 12
            .nMinute = 21
            .nSecond = 3
            .nHundredths = 91
        .fValue = 28.5
```

Fig. 54: Fig. 2: Example of 2 log entries in the PLC

LogBuffer	131	100 Elements	BACnetLogRecordList
[1]		10.11.2014 12:19:44: {(log_dis...	BACnetLogRecord
[2]		10.11.2014 12:20:51: {(log_dis...	BACnetLogRecord
[3]		10.11.2014 12:21:02: {(log_en...	BACnetLogRecord
[4]		10.11.2014 12:21:02: {28}	BACnetLogRecord
timestamp		10.11.2014 12:21:02	BACnetDateTime
date		10.11.2014	BACnetDate
time		12:21:02	BACnetTime
logDatum		{28}	BACnetLogDatumList
[1]		28	realValue
statusFlags	☑	0x00000004	BACnetStatusFlags
[5]		10.11.2014 12:21:03: {28,5}	BACnetLogRecord
timestamp		10.11.2014 12:21:03	BACnetDateTime
date		10.11.2014	BACnetDate
time		12:21:03	BACnetTime
logDatum		{28,5}	BACnetLogDatumList
[1]		28,5	realValue
statusFlags	☑	0x00000004	BACnetStatusFlags

Fig. 55: Fig. 3: Example of 2 log entries in the TwinCAT System Manager

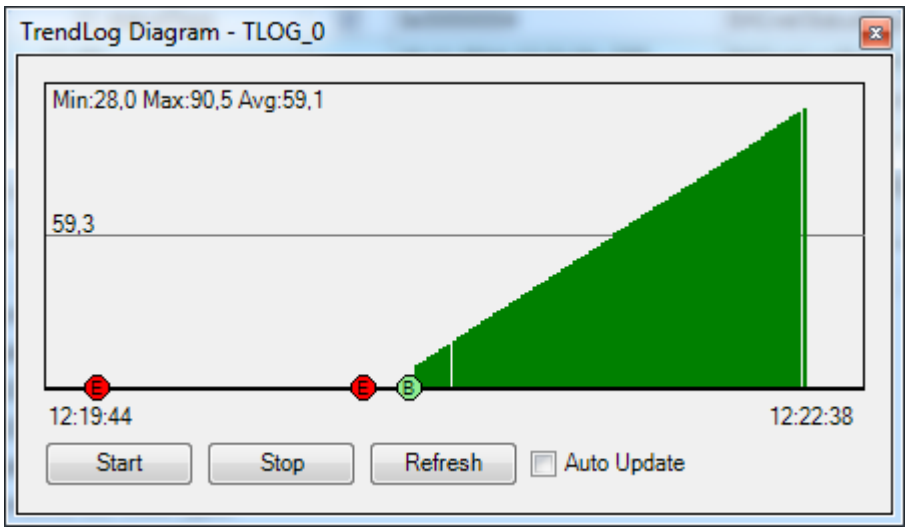


Fig. 56: Fig. 4: Exemplary representation of the property Log_Buffer in the TwinCAT System Manager.

Example 2: Function block call

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbLogBuf     : FB_BACnet_LogBufferProperty;
0005     tonReadProp : TON;
0006 END_VAR
```

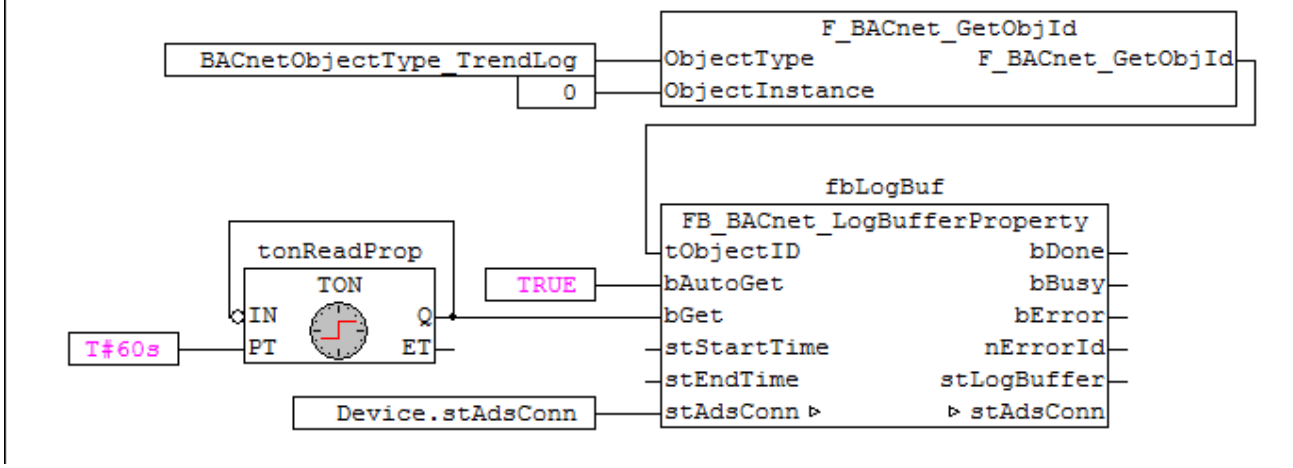
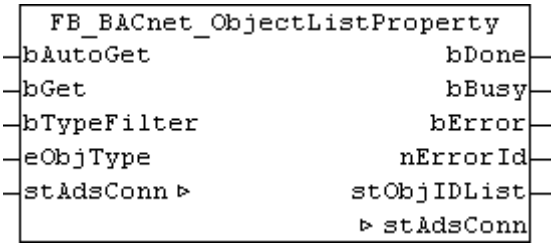


Fig. 57: Fig. 5: Application example. Cyclic reading of the property Log_Buffer of a BACnet TrendLog object.

4.5.12 FB_BACnet_ObjectListProperty



Application

The function block can be used to read the property *Object_List* of any BACnet device object (local or remote). The Example [▶ 292] shows a possible configuration.

An object type filter can be used to limit the **stObjIDList** output.

The function block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device function block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

The ADS buffer from the global variables is used for buffering the ADS data (see ST_BACnet_GlobalAdsBuffer [▶ 360]).

ObjectList	76	{Device:927609;File:0;File:1...	BACnetObjectIdentifier[]
[1]		Device:927609	BACnetObjectIdentifier
[2]		File:0	BACnetObjectIdentifier
[3]		File:1	BACnetObjectIdentifier
[4]		StructuredView:0	BACnetObjectIdentifier
[5]		Loop:0	BACnetObjectIdentifier
[6]		Schedule:0	BACnetObjectIdentifier

Fig. 58: Figure 1: Online view of the property Object_List

VAR_INPUT

```
bAutoGet     : BOOL:=TRUE;
bGet         : BOOL;
bTypeFilter  : BOOL;
eObjType     : E_BACnetObjectType;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bTypeFilter: *TRUE* = output **stObjIDList** is limited to one BACnet object type; *FALSE* = all BACnet objects of the BACnet server (or client) are output.

eObjType: Filter of the BACnet object type to be output. If **bTypeFitler** = *FALSE*, the input is ignored.

VAR_OUPUT

```
bDone        : BOOL;
bBusy        : BOOL;
bError       : BOOL;
nErrorId     : UINT;
stObjIDList  : ST_BACnet_ObjectIdentifierList;
```

bDone: Read of the data completed successfully. **bDone** remains set until to **bGet** and **bAutoGet** are reset or re-reading begins. If **bGet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals for an overview.

stObjIDList: Structure with the number and list of the read BACnet object IDs (object type and object instance).

VAR_IN_OUT

```
stAdsConn    : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     Device      : FB_BACnet_Device;
0004     fbObjList   : FB_BACnet_ObjectListProperty;
0005     tonPropRead : TON;
0006 END_VAR
```

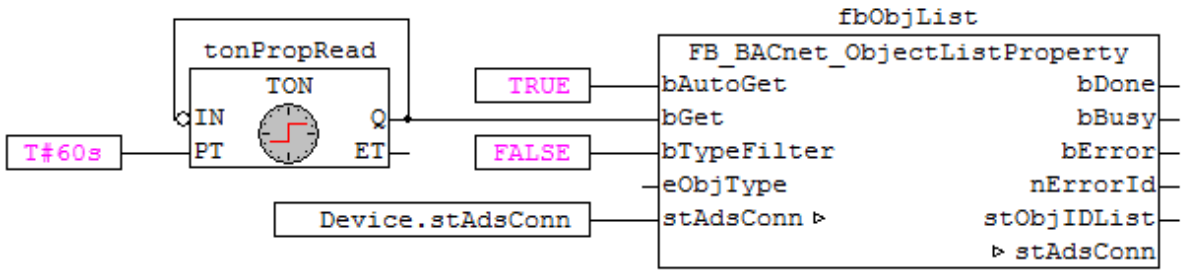
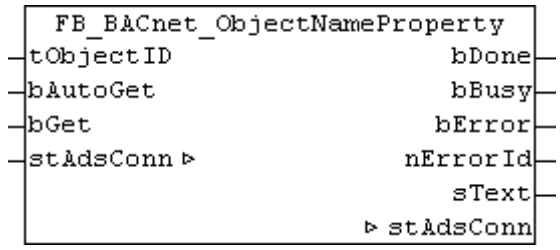


Fig. 59: Fig. 2: Application example. Cyclic reading of property Object_List of a BACnet server (from the BACnet device object).

4.5.13 FB_BACnet_ObjectNameProperty



Application

The function block can be used to read the property *Object_Name* of any BACnet object. The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 294] shows a possible configuration.

Within the block, the property is decoded based on the string coding. The following property encodings are supported: *UTF-8*, *UCS2*, *UCS4* and *ISO8859-1*. The output string **sText** is Windows-1252-coded (see also FB_BACnet_StringExtDecode [▶ 321]). If an error occurs during decoding of the property, this is indicated at the **nErrorId** output.

The block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

VAR_INPUT

```
tObjectID  : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet   : BOOL:=TRUE;
bGet       : BOOL;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = *Property* is read once, irrespective of input **bAutoGet**.

VAR_OUPUT

```
bDone      : BOOL;
bBusy      : BOOL;
bError     : BOOL;
nErrorId   : UINT;
sText      : T_MaxString;
```

bDone: Read of the data completed successfully. **bDone** remains set until to **bGet** and **bAutoGet** are reset or re-reading begins. If **bGet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

sText: Object name as Windows-1252-coded string.

VAR_IN_OUT

```
stAdsConn      : ST_BACnet_AdsConnection;
```

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbObjName    : FB_BACnet_ObjectNameProperty;
0005     tonPropRead : TON;
0006 END_VAR
```

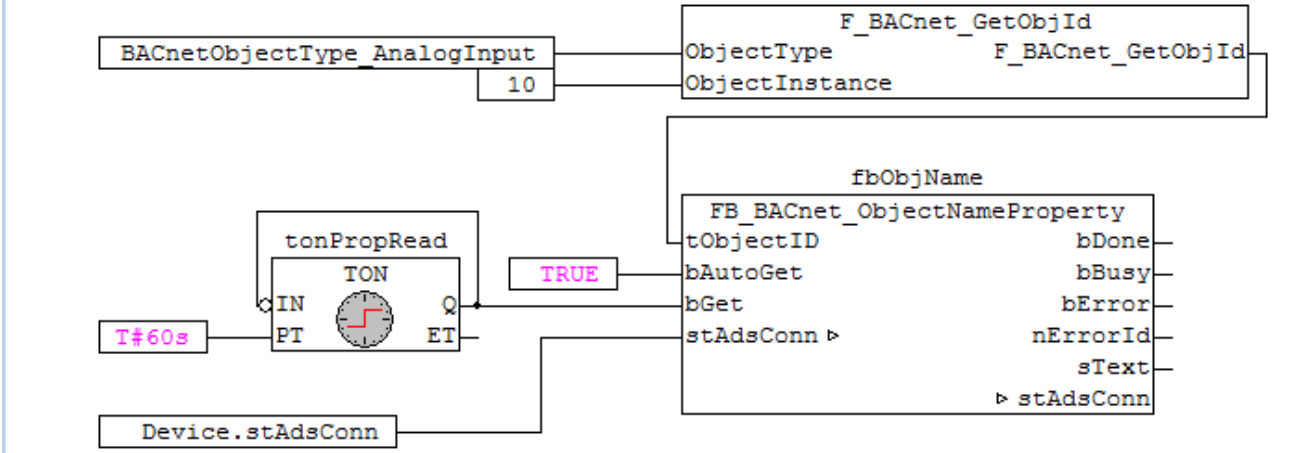
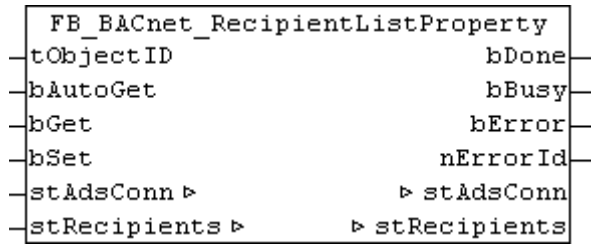


Fig. 60: Figure 1: Application example. Cyclic reading of the property Object_Name of a BACnet analog input object (AnalogInput: 10).

4.5.14 FB_BACnet_RecipientListProperty



Application

The function block can be used to read the property *Recipient_List* of any BACnet object (typically BACnet Notification Class Object). The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 296] shows a possible configuration.

The data to be read or written is stored in the same input/output data structure (see also ST_BACnet_RecipientListDevice [▶ 365]).

The function block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device function block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

The ADS buffer from the global variables is used for buffering the ADS data (see ST_BACnet_GlobalAdsBuffer [▶ 360]).

VAR_INPUT

```
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet       : BOOL:=TRUE;
bGet           : BOOL;
bSet           : BOOL;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = **Property** is read once, irrespective of input **bAutoGet**.

bSet : *FALSE* --> *TRUE* = property is written.

VAR_OUPUT

```
bDone          : BOOL;
bBusy          : BOOL;
bError         : BOOL;
nErrorId       : UINT;
```

bDone: Reading or writing of the data completed successfully. **bDone** set remains set until **bGet**, **bSet** and **bAutoGet** are reset, or until a new read or write operation starts. If **bGet**, **bSet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet Globals [▶ 328] for an overview.

VAR_IN_OUT

```
stAdsConn      : ST_BACnet_AdsConnection;
stRecipients   : ST_BACnet_RecipientListDevice;
```

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

stRecipients: Input/output data structure for the entries of the property *Recipient_List*. The following section compares an entry in the PLC and the TwinCAT System Manager (see Example [▶ 296]).

Example 1: Comparison of PLC and System Manager

```
stRecipList
    .nEntries = 1
    .arrDestList
        .arrDestList[0]
            .bMonday = TRUE
            .bTuesday = TRUE
            .bWednesday = TRUE
            .bThursday = TRUE
            .bFriday = TRUE
            .bSaturday = TRUE
            .bSunday = TRUE
            .stFromTime
                .nHour = 0
                .nMinute = 0
                .nSecond = 0
                .nHundredths = 0
            .stToTime
                .nHour = 23
                .nMinute = 59
                .nSecond = 59
                .nHundredths = 99
            .tDeviceId = 34482041
            .nProcessId = 10000
            .bToOffNormal = TRUE
            .bToFault = TRUE
            .bToNormal = TRUE
            .bIssueConfirmedNotification = FALSE
```

Fig. 61: Figure 1: Example of a property entry in the PLC

RecipientList	102	{(65025;00:00:00;23:59:59;(De...	BACnetDestination[]
[1]		(65025;00:00:00;23:59:59;(De...	BACnetDestination
validDays		0x0000FE01 (Monday;Tuesday...	BACnetDaysOfWeek
Monday		☑	BitFieldBit
Tuesday		☑	BitFieldBit
Wednesday		☑	BitFieldBit
Thursday		☑	BitFieldBit
Friday		☑	BitFieldBit
Saturday		☑	BitFieldBit
Sunday		☑	BitFieldBit
fromTime		00:00:00	BACnetTime
toTime		23:59:59	BACnetTime
recipient		(Device:927609)	device
processIdentifier		10000	UnsignedInteger
transitions		0x0000E005 (to_offnormal;to_f...	BACnetEventTransitionBits
to_offnormal		☑	BitFieldBit
to_fault		☑	BitFieldBit
to_normal		☑	BitFieldBit
issueConfirmedNo...		☐	Bool

Fig. 62: Fig. 2: Example of a property entry in the TwinCAT System Manager

Example 2: Function block call

```
0001 PROGRAM MAIN
0002 VAR
0003     Device       : FB_BACnet_Device;
0004     fbRecipList  : FB_BACnet_RecipientListProperty;
0005     stRecipList  : ST_BACnet_RecipientListDevice;
0006
0007     bReadRecipData  : BOOL;
0008     bWriteRecipData : BOOL;
0009 END_VAR
```

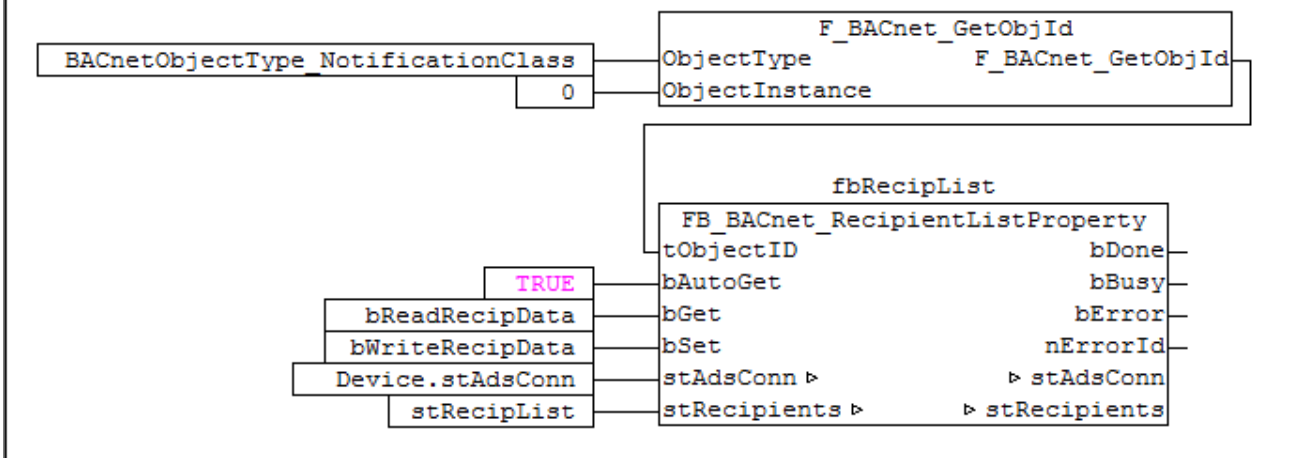
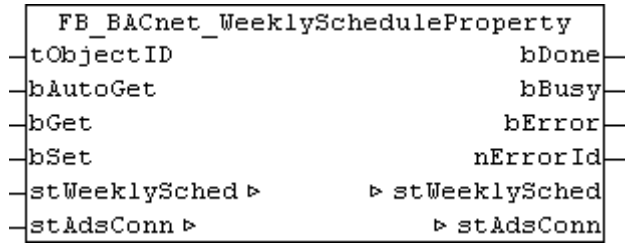


Fig. 63: Fig. 3: Application example. Reading and writing of the property Recipient_List of a BACnet notification class object.

4.5.15 FB_BACnet_WeeklyScheduleProperty



Application

The function block can be used to read the property *Weekly_Schedule* of any BACnet object (typically BACnet schedule object). The input **tObjectID** specifies the object (*Object_Identifier*: object type and object instance) to be accessed. The Example [▶ 299] shows a possible configuration.

The data to be read or written is stored in the same input/output data structure (see also ST_BACnet_WeeklyScheduleBool [▶ 370]).

The only data types supported by the PLC function block for entries in the property *Weekly_Schedule* are *Bool* and *Null*! Entries with other data types are ignored or deleted on write access to the property.

WeeklySchedule	123	7 Elements	BACnetDailyScheduleList
[1] Monday		{(07:00:00;True);(18:00:00;False);(20:...	BACnetTimeValueList
[1]		(07:00:00;True)	BACnetTimeValue
time		07:00:00	BACnetTime
value		True	BoolValue
[2]		(18:00:00;False)	BACnetTimeValue
time		18:00:00	BACnetTime
value		False	BoolValue
[3]		(20:00:00;True)	BACnetTimeValue
[4]		(05:00:00;False)	BACnetTimeValue

Fig. 64: Figure 1: Example entry of the property Weekly_Schedule with data type Bool.

The function block instance is created by the PLC program and called cyclically. The input/output **stAdsConn** must be linked to the output **stAdsConn** of the respective device function block (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]).

The ADS buffer from the global variables is used for buffering the ADS data (see ST_BACnet_GlobalAdsBuffer [▶ 360]).

VAR_INPUT

```
tObjectID       : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
bAutoGet        : BOOL:=TRUE;
bGet            : BOOL;
bSet            : BOOL;
```

tObjectID: Specifies the object (*Object_Identifier*: object type and object instance) to be accessed.

bAutoGet: *TRUE* = the property is read automatically when the ADS connection or the *Object_Identifier* have changed. An ADS connection change occurs when the connection is restored after an interruption, or if the AMS NetID or port has changed. Automatic reading does not occur cyclically, and not if the property itself changes!

bGet: *FALSE* → *TRUE* = Property is read once, irrespective of input **bAutoGet**.

bSet : *FALSE* --> *TRUE* = property is written.

VAR_OUPUT

```
bDone           : BOOL;
bBusy           : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bDone: Reading or writing of the data completed successfully. **bDone** set remains set until **bGet**, **bSet** and **bAutoGet** are reset, or until a new read or write operation starts. If **bGet**, **bSet** and **bAutoGet** are reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrorId: Error code, see BACnet_Globals [▶ 328] for an overview.

VAR_IN_OUT

```
stWeeklySched  : ST_BACnet_WeeklyScheduleBool;
stAdsConn      : ST_BACnet_AdsConnection;
```

stWeeklySched : Input/output data structure for the entries of the property *Weekly_Schedule* with data type *Bool* or *Null*. The following section compares an entry in the PLC and the TwinCAT System Manager (see Example [▶ 299]).

stAdsConn: Linking to the output **stAdsConn** of the corresponding device function block.

Example 1: Comparison of PLC and System Manager

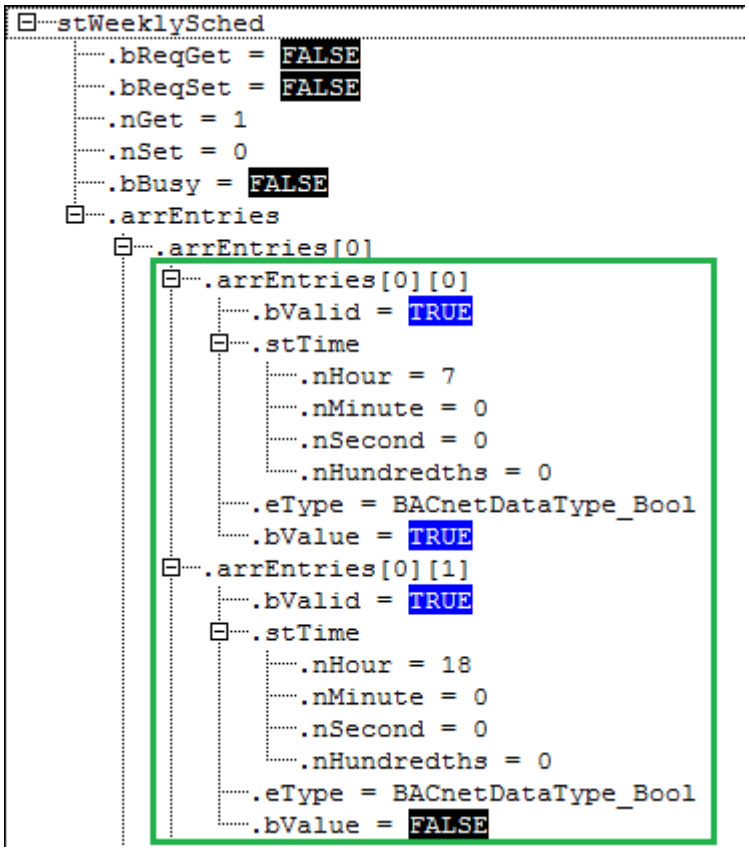


Fig. 65: Fig. 2: Example of a property entry in the PLC

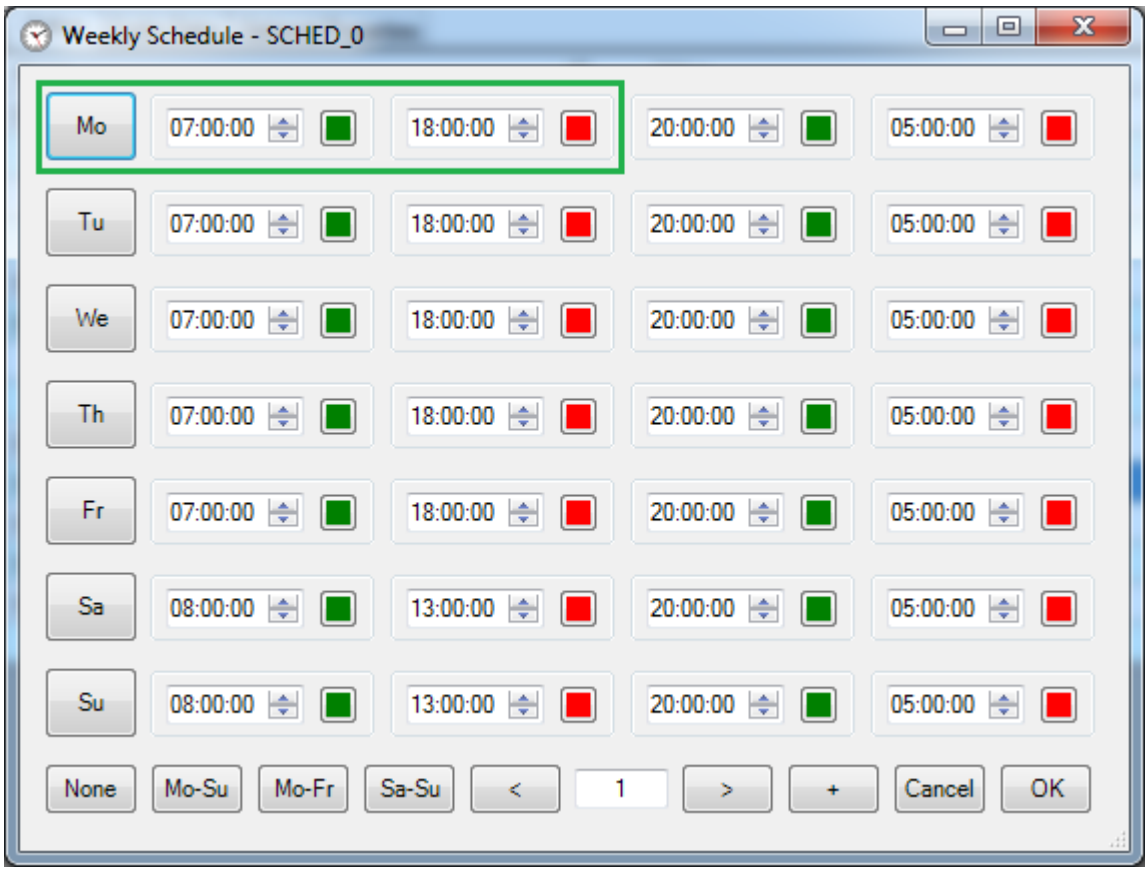


Fig. 66: Fig. 3: Example of a property entry in the TwinCAT System Manager

Example 2: Function block call

```
0001 PROGRAM MAIN
0002 VAR
0003     Device         : FB_BACnet_Device;
0004     fbWeeklySched  : FB_BACnet_WeeklyScheduleProperty;
0005     stWeeklySched  : ST_BACnet_WeeklyScheduleBool;
0006
0007     bWriteScheduleData: BOOL;
0008     bReadScheduleData: BOOL;
0009 END_VAR
0010
```

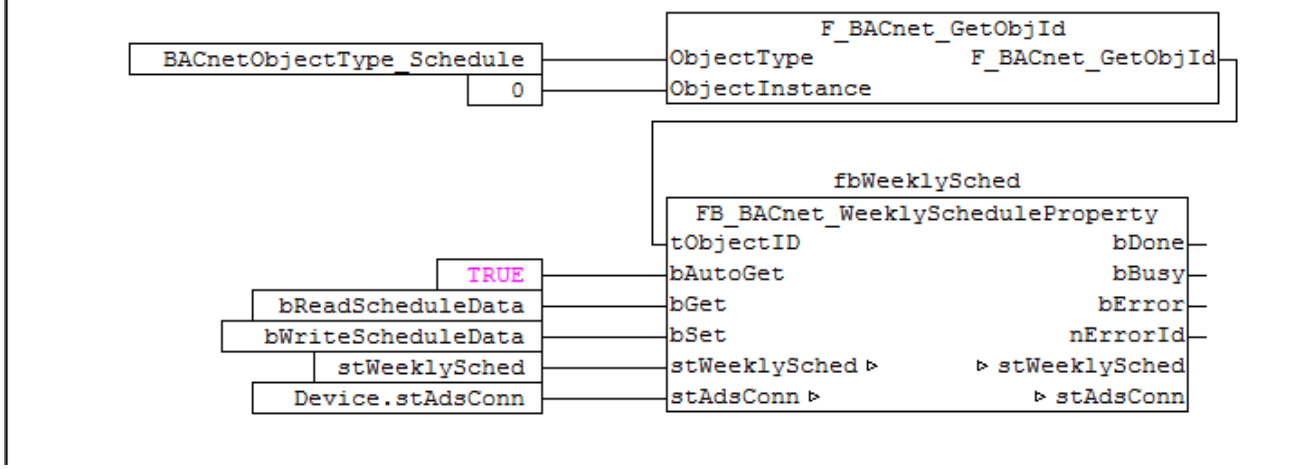
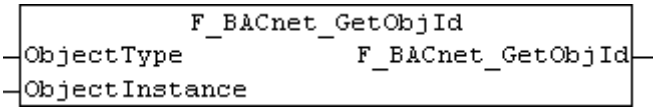


Fig. 67: Fig. 4: Application example. Reading and writing of the property Weekly_Schedule of a BACnet schedule object.

4.6 BACnet Helper

4.6.1 F_BACnet_GetObjId : DWORD



Application

Function for coding the object type and the object instance in the BACnet *Object_Identifier*.

VAR_INPUT

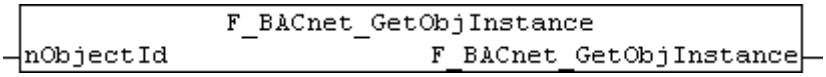
```
ObjectType      : E_BACNETOBJECTTYPE;
ObjectInstance  : UDINT;
```

ObjectType: BACnet object type.

ObjectInstance: BACnet object number (instance, 0...4'194'303).

Return value: The resulting BACnet *Object_Identifier*.

4.6.2 F_BACnet_GetObjInstance : UDINT



Application

Function for decoding the BACnet *Object_Identifier* in the object instance (object number).

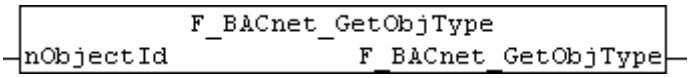
VAR_INPUT

```
nObjectId      : DWORD;
```

nObjectId: BACnet *Object_Identifier*.

Return value: BACnet object number (instance, 0...4'194'303).

4.6.3 F_BACnet_GetObjType : E_BACNETOBJECTTYPE



Application

Function for decoding the BACnet *Object_Identifier* in the object type.

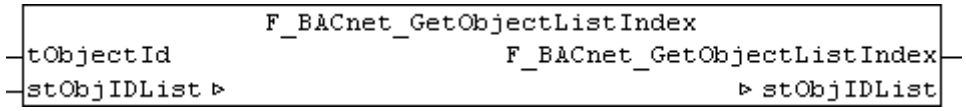
VAR_INPUT

```
nObjectId      : DWORD;
```

nObjectId: BACnet *Object_Identifier*.

Return value: The resulting BACnet object type (see E_BACnetObjectType [▶ 342]).

4.6.4 F_BACnet_GetObjectListIndex : DINT



Application

Function for determining the position of a BACnet object ID in a list of IDs. Returns the index (position) of the input object ID in a list, or *-1*. A BACnet object list provides the function block FB_BACnet_ObjectListProperty [▶ 290] for the property *Object_List*.

VAR_INPUT

```
tObjectId    : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
```

tObjectId: Specifies the object (*Object_Identifier*: object type and object instance) whose position is to be determined in the list.

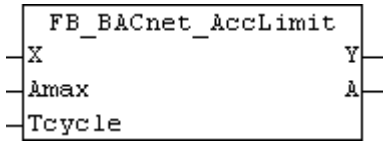
VAR_IN_OUT

```
stObjIDList      : ST_BACnet_ObjectIdentifierList;
```

stObjIDList: List of BACnet object IDs.

Return value: Position within the list (from *0* to *length -1*), or *-1* if not present.

4.6.5 FB_BACnet_AccLimit



Application

Function block for limiting signal change per time (maximum acceleration). The output variable **Y** [1] follows the input variable **X** [1] with a maximum acceleration (positive and negative) in accordance with the value at input **Amax** [1/s]. The input value of **Tcycle** indicates the current PLC task cycle time in seconds. If **Tcycle** is set to *0*, the current cycle time is determined internally.

The current acceleration is output at output **A** [1/s]. If **Amax** is set to *0*, there is no limitation → **Y** = **X**.

VAR_INPUT

```
X       : REAL;
Amax    : REAL;
Tcycle  : REAL;
```

X: Input value [1].

Amax: Maximum acceleration [1/s]; *0* = no limitation.

Tcycle: PLC cycle time in seconds; *0* = determine internally.

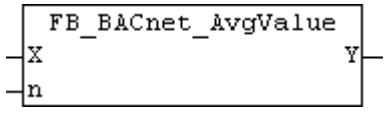
VAR_OUPUT

```
Y       : REAL;
A       : REAL;
```

Y: Limited output value [1].

A: Currently determined acceleration [1/s] of the input value **X** relative to the output value **Y**.

4.6.6 FB_BACnet_AvgValue



Application

Function block for averaging an input variable **X** over **n** values. Output **Y** is used for the output. Averaging takes place with each block call. The average is calculated internally using an 8-byte real value (LREAL). With increasing number of averaging operations (n >>), the accuracy of the resulting average value decreases accordingly.

VAR_INPUT

```
X       : REAL;
n       : UINT;
```

X: Input value to be averaged.

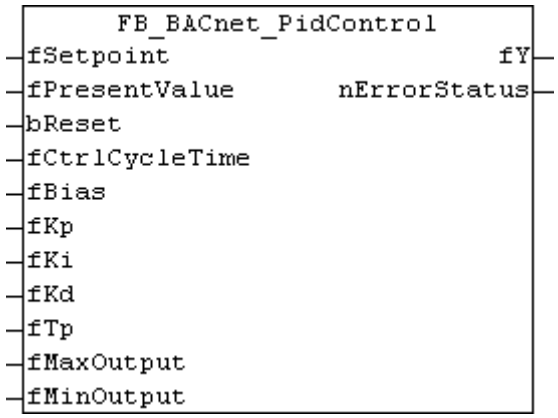
n: Number of values to be averaged (*0* = no averaging **Y** = **X**).

VAR_OUPUT

```
Y       : REAL;
```

Y: Result of averaging.

4.6.7 FB_BACnet_PidControl



Application

PID controller block in parallel configuration or Ideal form. The controller block serves as a basis for the BACnet loop object (FB_BACnet_Loop [▶ 201]).

Block diagram

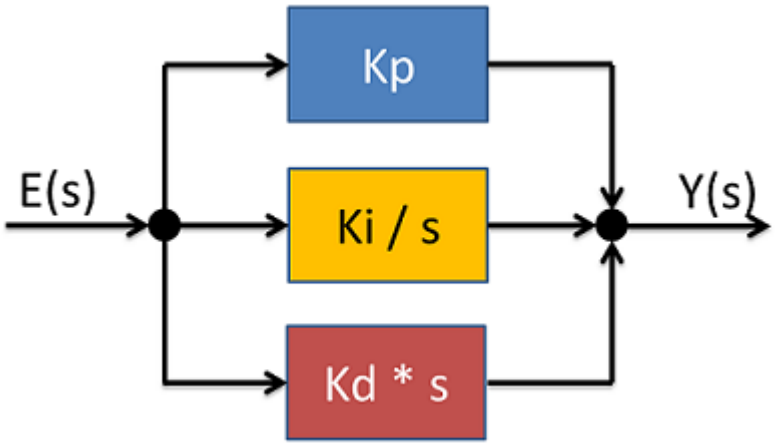


Fig. 68: Fig. 1: Block diagram of the transfer function (frequency range).

Use

The controller block must be called cyclically in the PLC program. Please ensure that the cycle time in the moment of the call at input **fCtrlCycleTime** is transferred as precisely as possible. The cycle time can be determined from the global structure "SystemTaskInfoArr", for example. A possible controller configuration is presented under Example [▶ 305].

VAR_INPUT

```
fSetpoint       : REAL;
fPresentValue   : REAL;
bReset          : BOOL;
fCtrlCycleTime  : REAL;
fBias           : REAL;
fKp             : REAL;
fKi             : REAL;
fKd             : REAL;
fTp             : REAL:=-1.0;
fMaxOutput      : REAL;
fMinOutput      : REAL;
```

fSetpoint: setpoint value input (e.g. temperature specification for a room in [°C])

fPresentValue: actual value (e.g. actual room temperature in [°C])

bReset: resets the controller, the output is set to **fBias** for the duration and limited through **fMaxOutput** and **fMinOutput**.

If **fMinOutput** is set to 5% and **fBias** to 0%, for example, the output **fY** is set to 5% for the duration of the reset.

fCtrlCycleTime: cycle time with which the controller is called in seconds [s].

fBias: output offset. This value is offset with the output. The unit must match the unit of the control value (e.g. heating capacity in [%]).

fKp: gain factor P. The control deviation is multiplied with P and added to the output (output behaves proportional to the input deviation).

fKi: integral gain I. The control deviation is multiplied with I (taking into account the cycle time), added to the previous result and added to the output (high control deviation = fast increasing output).

fKd: differential gain D. The control deviation is offset with the previous controller difference, the result multiplied with D (taking into account the cycle time) and added to the output (strong controller variation = strong response).

fTp: damping time for D part (PT1) [s] (if **fTp***0*, then: *Tp* = **fKd** / *10*).

fMaxOutput: upper limit of the output value **fY**. The unit must match the unit of the control value (e.g. heating capacity maximum 90%).

fMinOutput: lower limit of the output value **fY**. The unit must match the unit of the control value (e.g. heating capacity minimum 5%).

VAR_OUPUT

```
fY            : REAL;
nErrorStatus  : UINT;
```

fY: Control value (e.g. heating capacity in [%]).

nErrorStatus: Error code, see BACnet_Globals [▶ 328] for an overview.

Example

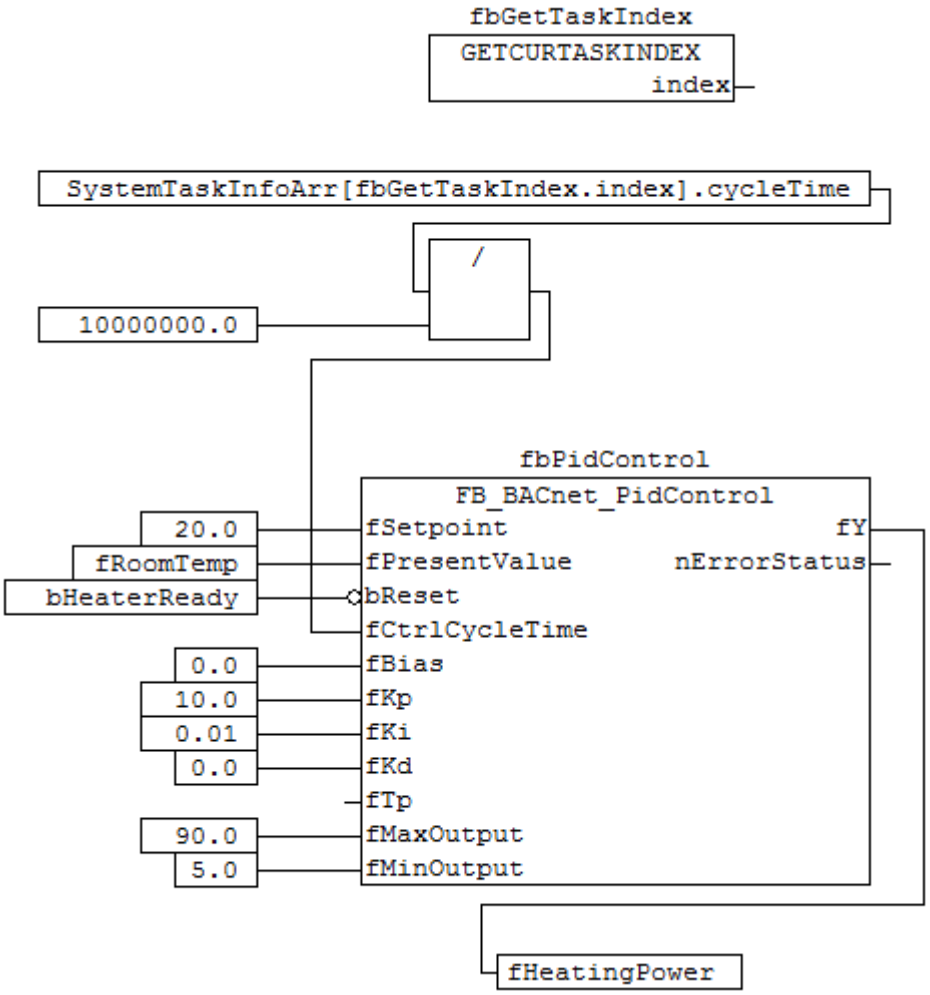


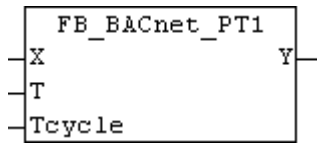
Fig. 69: Figure 2: Application example

Note re. P = 10 --> a temperature difference of 1 K results in 10% heating capacity, for example

Note re. I = 0.01 --> a temperature difference of 1 K results in 0.01% heating output increase per second, for example

Note re. D = 0 --> no D part.

4.6.8 FB_BACnet_PT1



Application

Function block for emulating a 1st order deceleration. Higher-order decelerations can be achieved by linking several PT1 blocks. The output value **Y** [1] represents the function result of the deceleration of input value **X** [1]. Input value **T** specifies the damping time [s] (also known as τ). If **T** is set to *0*, there is no deceleration → **Y** = **X**.

The input value of **Tcycle** indicates the current PLC task cycle time in seconds. If **Tcycle** is set to *0*, the current cycle time is determined internally.

VAR_INPUT

```
X       : REAL;
T       : REAL;
Tcycle  : REAL;
```

X: Input value [1].

T: Damping time [s]; *0* = no deceleration.

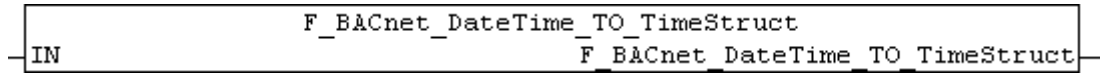
Tcycle: PLC cycle time in seconds; *0* = determine internally.

VAR_OUPUT

```
Y       : REAL;
```

Y: Decelerated output value [1].

4.6.9 F_BACnet_DateTime_TO_TimeStruct : TIMESTRUCT



Application

Function for converting a BACnet time stamp to data type *TIMESTRUCT*..

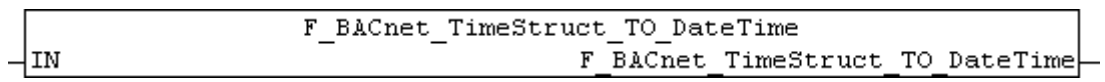
Warning: No placeholders (*255* → *) and special values ("*all even months*" or "*last day of the month*" etc.) may be used!

VAR_INPUT

```
IN      : ST_BACnet_DateTime;
```

IN: BACnet time stamp as input value.

4.6.10 F_BACnet_TimeStruct_TO_DateTime : ST_BACnet_DateTime



Application

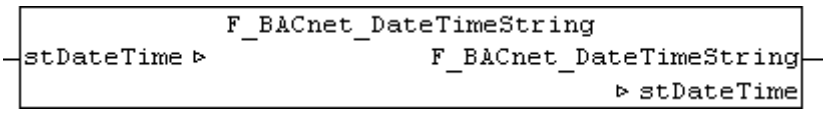
Function for converting a time stamp of data type *TIMESTRUCT* to a BACnet time stamp.

Warning: In this case, the time stamp accuracy is reduced from 1/1000 ms to 1/100 ms. Only year values between 1900 and 2154 are supported!

VAR_INPUT

```
IN      : TIMESTRUCT;
```

4.6.11 F_BACnet_DateTimeString : STRING(19)



Application

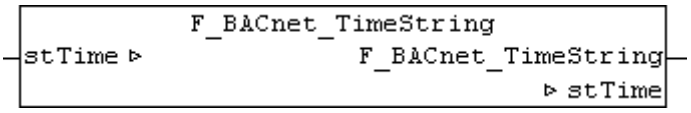
Function for string representation of a BACnet time stamp. Example: "*07.01.2015 11:12:20*" or with placeholders: "*07.01.**** 10:**:30*".

VAR_IN_OUT

```
stDateTime      : ST_BACnet_DateTime;
```

stDateTime: BACnet time stamp for conversion to a string.

4.6.12 F_BACnet_TimeString : STRING(12)



Application

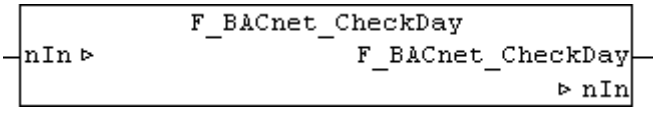
Function for string representation of a BACnet time stamp. Example: "*11:12:20.00*" or with placeholders: "*10:**:30.70*".

VAR_IN_OUT

```
stTime      : ST_BACnet_Time;
```

stTime: BACnet time stamp for conversion to a string.

4.6.13 F_BACnet_CheckDay : USINT



Application

Function for checking the validity of a numerical date value (BYTE) for the day of the month. The return value matches a valid value for the day of the month, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime [▶ 357].

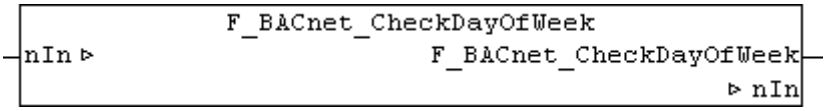
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical day value to be corrected (1 to 31 → day of the month; 32 → last day of the month; otherwise converted to 255 → undefined)

4.6.14 F_BACnet_CheckDayOfWeek : USINT



Application

Function for checking the validity of a numerical date value (BYTE) for the day of the week. The return value matches a valid value for the day of the week, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime [▶ 357].

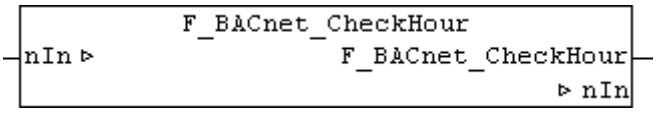
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical day of the week value to be checked (1 to 7 valid; otherwise converted to 255 → undefined)

4.6.15 F_BACnet_CheckHour : USINT



Application

Function for checking the validity of a numerical time value (BYTE) for the hour indication. The return value matches a valid value for the hour, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

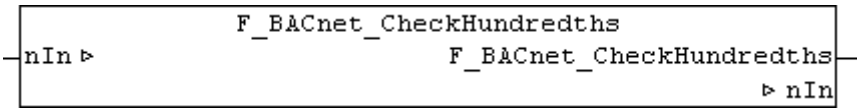
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical hour value to be checked (0 to 23 valid; otherwise converted to 255 → undefined)

4.6.16 F_BACnet_CheckHundredths : USINT



Application

Function for checking the validity of a numerical time value (BYTE) for the hundredths of a second indication. The return value matches a valid value for hundredths of a second, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

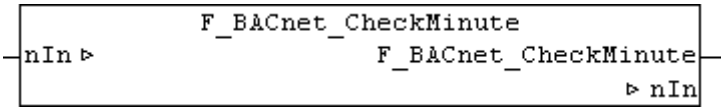
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical hundredths of a second value to be checked (0 to 99 valid; otherwise converted to 255 → undefined)

4.6.17 F_BACnet_CheckMinute : USINT



Application

Function for checking the validity of a numerical time value (BYTE) for the minute indication. The return value matches a valid value for the minute, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

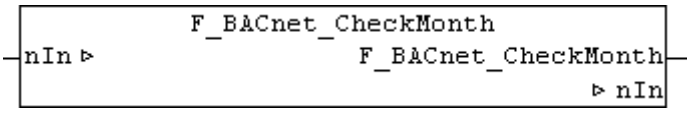
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical minute value be checked (0 to 59 valid; otherwise converted to 255 → undefined)

4.6.18 F_BACnet_CheckMonth : USINT



Application

Function for checking the validity of a numerical date value (BYTE) for the month. The return value matches a valid value for the month, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime [▶ 357].

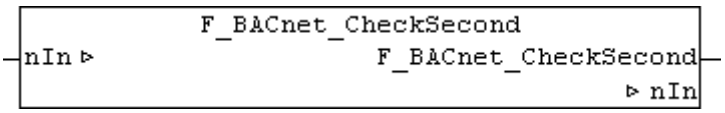
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical month value to be corrected (1 to 12 → month 1 = January; 13 → odd months; 14 → even months, otherwise converted to 255 → undefined)

4.6.19 F_BACnet_CheckSecond : USINT



Application

Function for checking the validity of a numerical time value (byte) for the second. The return value matches a valid value for the second, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

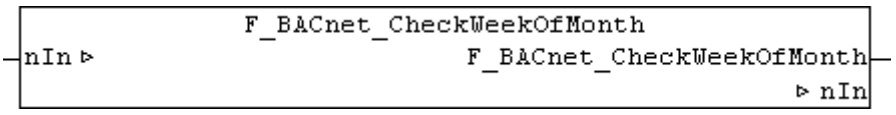
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical second value to be checked (valid 0 to 59; otherwise converted to 255 → undefined)

4.6.20 F_BACnet_CheckWeekOfMonth : USINT



Application

Function for checking the validity of a numerical date value (BYTE) for the week of the month. The return value matches a valid value for the week of the month, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetWeekNDay.

Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical week of the month value to be corrected; possible values:

1 → days 1 to 7 of the month

2 → days 8 to 14 of the month

3 → days 15 to 21 of the month

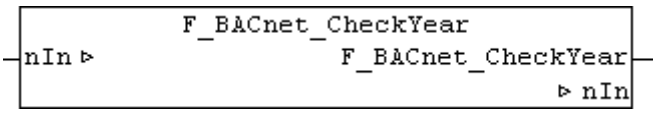
4 → days 22 to 28 of the month

5 → days 29 to 31 of the month

6 → the last 7 days of the month

otherwise converted to 255 → undefined

4.6.21 F_BACnet_CheckYear : USINT



Application

Function for checking the validity a numerical date value (BYTE) for the year. The return value matches a valid value for a year, or undefined as a numerical code (*). See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime [▶ 357].

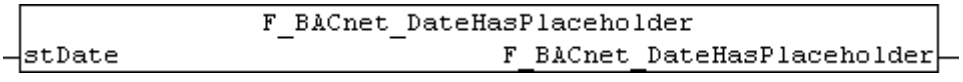
Attention: The input value is also corrected (**nIn**).

VAR_IN_OUT

```
nIn      : BYTE;
```

nIn: Numerical year value to be corrected (year minus 1900 → 0 to 254 corresponds to the years 1900 to 2154; 255 → undefined)

4.6.22 F_BACnet_DateHasPlaceholder : BOOL



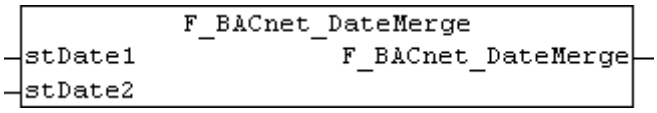
Application

Function for checking for placeholders (255 → undefined) in a date. Return value *TRUE* → at least one placeholder found / *FALSE* → no placeholders.

VAR_INPUT

```
stDate      : ST_BACnet_Date;
```

4.6.23 F_BACnet_DateMerge : ST_BACnet_Date



Application

Function for merging of two time stamps. The placeholders of one time stamp are replaced by the respective value of the second time stamp. If neither of the two time stamps contains a placeholder, the value of the time stamp from **stDate1** is preferred.

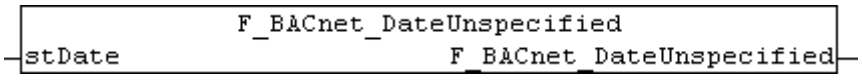
VAR_INPUT

```
stDate1      : ST_BACnet_Date;
stDate2      : ST_BACnet_Date;
```

stDate1: Time stamp 1 (preferred if no placeholder is present)

stDate2: Time stamp 2.

4.6.24 F_BACnet_DateUnspecified : BOOL



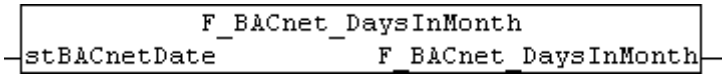
Application

Function for checking for placeholders (*255* → undefined) in a date. Return value *TRUE* = all values are undefined / *FALSE* = at least one value has been specified.

VAR_INPUT

```
stDate      : ST_BACnet_Date;
```

4.6.25 F_BACnet_DaysInMonth : UDINT



Application

Function for calculating the number of days in a given month of a year. Return value *16#FFFFFFFF* means month and/or year is not specified. Leap years are taken into account.

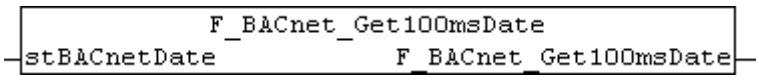
Warning: No placeholders (*255* → *) or special values ("*all even months*") may be used for years and months!

VAR_INPUT

```
stBACnetDate      : ST_BACnet_Date;
```

stBACnetDate: BACnet time stamp as input value. Year and month must be valid.

4.6.26 F_BACnet_Get100msDate : UDINT



Application

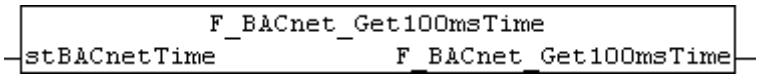
Function for calculating the timestamp in 100 ms steps since 1900.

VAR_INPUT

```
stBACnetDate      : ST_BACnet_Date;
```

stBACnetDate: BACnet time stamp as input value.

4.6.27 F_BACnet_Get100msTime : UDINT



Application

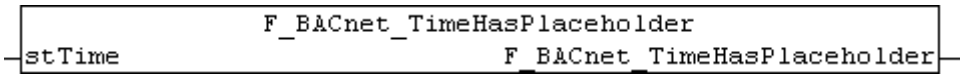
Function for calculating the time stamp in 100 ms steps since 00:00:00.0 hrs.

VAR_INPUT

```
stBACnetTime      : ST_BACnet_Time;
```

stBACnetTime: BACnet time stamp as input value.

4.6.28 F_BACnet_TimeHasPlaceholder : BOOL



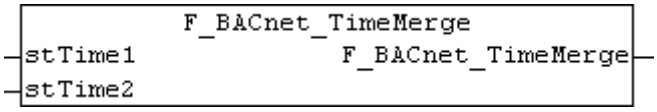
Application

Function for checking for placeholders (*255* → undefined) in a time specification. Return value *TRUE* → at least one placeholder found / *FALSE* → no placeholders.

VAR_INPUT

```
stTime      : ST_BACnet_Time;
```

4.6.29 F_BACnet_TimeMerge : ST_BACnet_Time



Application

Function for merging of two-time stamps. The placeholders of one-time stamp are replaced by the respective value of the second time stamp. If neither of the two-time stamps contains a placeholder, the value of the time stamp from **stTime1** is preferred.

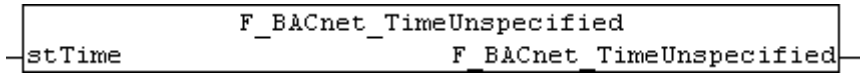
VAR_INPUT

```
stTime1      : ST_BACnet_Time;
stTime2      : ST_BACnet_Time;
```

stTime1: Time stamp 1 (preferred if no placeholder is present)

stTime2: Time stamp 2.

4.6.30 F_BACnet_TimeUnspecified : BOOL



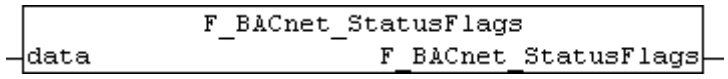
Application

Function for checking for placeholders (*255* → undefined) in a time specification. Return value *TRUE* = all values are undefined / *FALSE* = at least one value has been specified.

VAR_INPUT

```
stTime      : ST_BACnet_Time;
```

4.6.31 F_BACnet_StatusFlags : ST_BACnet_StatusFlags



Application

Function to decode the process data of the property *Status_Flags* of a BACnet object. The return value contains the set flags; see ST_BACnet_StatusFlags [▶ 367].

VAR_INPUT

```
data      : WORD;
```

data: Value from the process data of the property with the BACnet data type BACnetStatusFlags.

Example

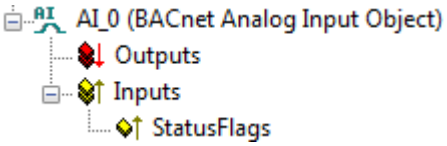
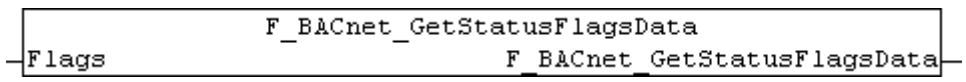


Fig. 70: Figure 1: Example of the process data of property Status_Flags of a BACnet analog input object in the TwinCAT System Manager.

4.6.32 F_BACnet_GetStatusFlagsData : WORD



Application

Function for encoding the value of the property *Status_Flags* of a BACnet object. The return value contains the value for writing the process data.

VAR_INPUT

```
Flags       : ST_BACnet_StatusFlags;
```

Example

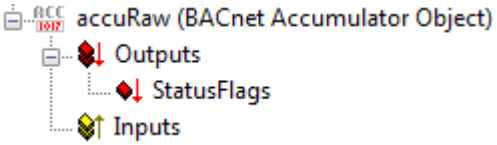
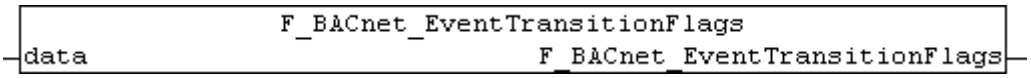


Fig. 71: Figure 1: Example of the process data of property Status_Flags of a BACnet Accumulator object in the TwinCAT System Manager.

4.6.33 F_BACnet_EventTransitionFlags : ST_BACnet_EventTransitionBits



Application

Function to decode the process data of the property *Acked_Transitions* of a BACnet object. The return value contains the set flags; see ST_BACnet_EventTransitionBits.htm [▶ 358].

VAR_INPUT

```
data        : WORD;
```

data: Value from the process data of the property with the BACnet data type BACnetEventTransitionBits.

Example

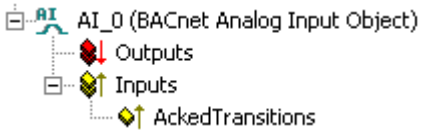
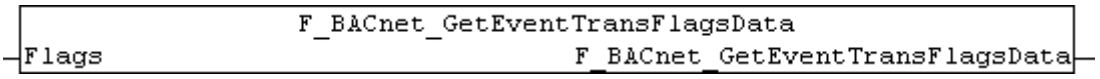


Fig. 72: Figure 1: Example of the process data of property Acked_Transitions of a BACnet analog input object in the TwinCAT System Manager.

4.6.34 F_BACnet_GetEventTransFlagsData : WORD



Application

Function for encoding the value of the property *Event_Enable* of a BACnet object. The return value contains the value for writing the process data.

VAR_INPUT

```
Flags       : ST_BACnet_EventTransitionBits;
```

Example

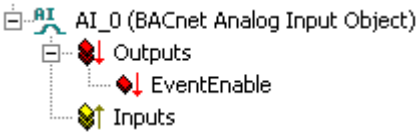
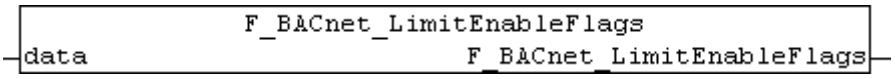


Fig. 73: Figure 2: Example of the process data of property Event_Enable of a BACnet analog input object in the TwinCAT System Manager.

4.6.35 F_BACnet_LimitEnableFlags : ST_BACnet_LimitEnable



Application

Function to decode the process data of the property *Limit_Enable* of a BACnet object. The return value contains the set flags; see ST_BACnet_LimitEnable [▶ 362].

VAR_INPUT

```
data      : WORD;
```

data: Value from the process data of the property with the BACnet data type BACnetLimitEnable.

Example

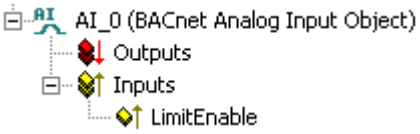
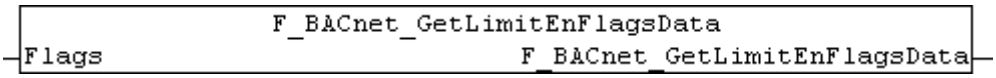


Fig. 74: Figure 1: Example of the process data of property Limit_Enable of a BACnet analog input object in the TwinCAT System Manager.

4.6.36 F_BACnet_GetLimitEnFlagsData : WORD



Application

Function for encoding the value of the property *Limit_Enable* of a BACnet object. The return value contains the value for writing the process data.

VAR_INPUT

```
Flags      : ST_BACnet_LimitEnable;
```

Example

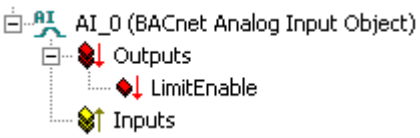
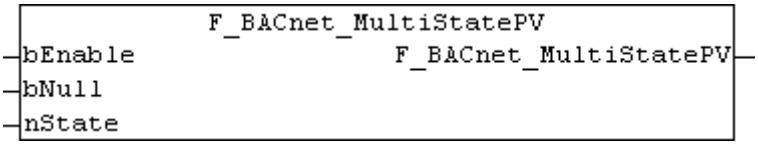


Fig. 75: Figure 1: Example of the process data of property Limit_Enable of a BACnet analog input object in the TwinCAT System Manager.

4.6.37 F_BACnet_MultiStatePV : UDINT



Application

Function for implementing a UDINT value of the PLC in the process data value of a BACnet MultiState* object property *Present_Value*. Using this function it is possible to write BACnet MultiState* objects, for example, which are only linked to a BACnet object via a primitive PLC variable (e.g.

```
nMV0 AT%Q* : UDINT; (* ~(BACnet_ObjectType : MV : NOLINK)(BACnet_ObjectIdentifier : 0 : NOLINK)
(BACnet_PresentValue_Priority12 : : LINK) *)
```

).

VAR_INPUT

```
bEnable  : BOOL;
bNull    : BOOL;
nState   : UDINT;
```

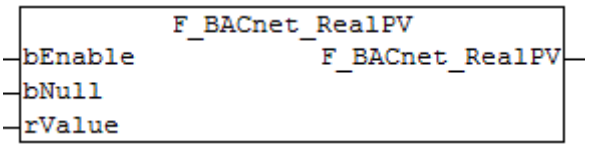
bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **nState** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g. for deleting a priority), *FALSE* = write value from **nState**

nState: Value that is written to the BACnet object, if **bEnable** = *TRUE* and **bNull** = *FALSE*. Multi-State in the range [1 .. *Number_Of_States*].

Return value: Function result of type UDINT.

4.6.38 F_BACnet_RealPV : DWORD



Application

Function for implementing a REAL value of the PLC in the process data value of a BACnet analog* object property *Present_Value*. Using this function it is possible to write BACnet analog* objects, for example, which are only linked to a BACnet object via a primitive PLC variable (e.g.

```
rAV0 AT%Q* : DWORD; (* ~(BACnet_ObjectType : AV : NOLINK)(BACnet_ObjectIdentifier : 0 : NOLINK)
(BACnet_PresentValue_Priority12 : : LINK) *)
```

).

The process data type DWORD must be used in the case of BACnet analog* objects, since special formats of the REAL data type are also used for encoding the property *Present_Value*. Depending on the target platform, process data of type REAL may not be treated properly, if special formats of the floating-point unit are used. The mapping of the process data between DWORD and REAL type is not critical, since the memory requirement is identical (4 bytes each).

VAR_INPUT

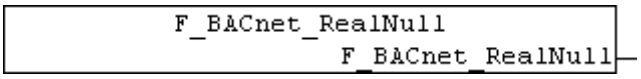
```
bEnable   : BOOL;
bNull     : BOOL;
rValue    : REAL;
```

bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **rValue** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: *TRUE* = zero writing of the BACnet object (e.g. for deleting a priority), *FALSE* = write value from **rValue**

rValue: Value that is written to the BACnet Object**.**

4.6.39 F_BACnet_RealNull : DWORD



Application

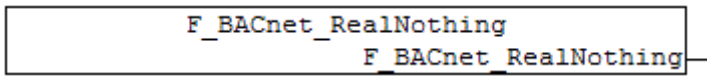
Function returns the floating-point value "**N**ot **aN**umber", encoded as DWORD. This can be used for deleting priority entries in the *Priority_Array* of an *Analog_Value* or *Analog_Output* object.

Return value: "Not a number" or **#QNAN** (0x7FFF0000).

4.6.40 F_BACnet_RealNothing : DWORD

Fig. 76: expand

Fig. 77: collapse

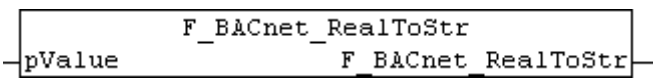


Application

Function returns the floating-point value "**N**ot **aN**umber", encoded as DWORD. Process data of property *Present_Value* of *AnalogValue* and *output* objects, which are described with the encoded value *Nothing*, are not processed by the BACnet stack from Revision 12 (*Nothing* → do-nothing state).

Return value: "Not a number" or **#QNAN** (0x7FFFFFFF).

4.6.41 F_BACnet_RealToStr : STRING



Application

Function for converting a floating-point number to a string, considering the value range. The REAL value is handed over by means of a pointer.

VAR_INPUT

```
pValue      : POINTER TO REAL;
```

pValue: Pointer to the floating-point number to be converted (ADR()).

Return value: For a finite value range the string from the floating-point number is output in the same format as after conversion with "REAL_TO_STRING()". If the value is not in the finite range, "#QNAN" (positive not-a-number) or "-#QNAN" (negative not-a-number) is output, depending on the sign.

Example

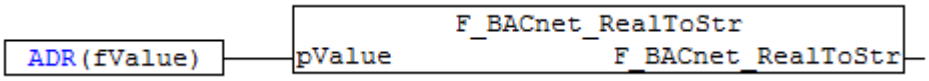
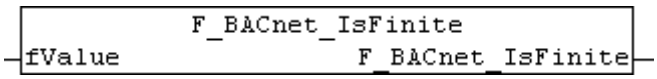


Fig. 78: Figure 1: Example for pointer determination with "ADR()". "fValue" represents a variable of type REAL.

4.6.42 F_BACnet_IsFinite : BOOL



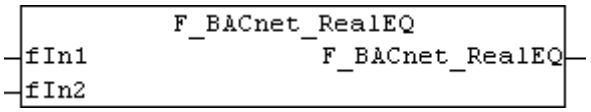
Application

Function for checking a REAL value for finiteness.

VAR_INPUT

```
fValue      : REAL;
```

4.6.43 F_BACnet_RealEQ : BOOL



Application

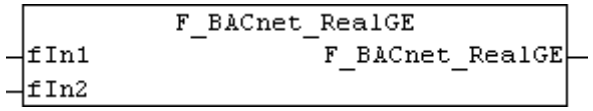
Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("=" or "EQ") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range (e.g. not-a-number values).

VAR_INPUT

```
fIn1        : REAL;
fIn2        : REAL;
```

Return value: *TRUE* = **fIn1** and **fIn2** are identical or both are not in the finite value range. *FALSE* = **fIn1** and **fIn2** are different.

4.6.44 F_BACnet_RealGE : BOOL



Application

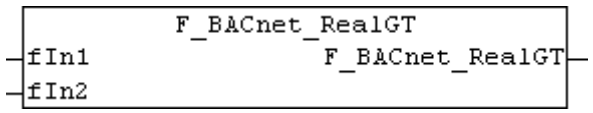
Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("=" or "GE") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range (e.g. not-a-number values).

VAR_INPUT

```
fIn1      : REAL;
fIn2      : REAL;
```

Return value: *TRUE* = **fIn1** is greater than or equal **fIn2**. *FALSE* = **fIn2** is less than **fIn1**, or one of the two values is not in the finite range of values.

4.6.45 F_BACnet_RealGT : BOOL



Application

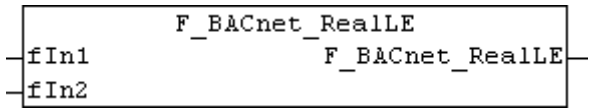
Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators (">" or "GT") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range (e.g. not-a-number values).

VAR_INPUT

```
fIn1      : REAL;
fIn2      : REAL;
```

Return value: *TRUE* = **fIn1** is greater than **fIn2**. *FALSE* = **fIn2** is less than or equal to **fIn1**, or one of the two values is not in the finite range of values.

4.6.46 F_BACnet_RealLE : BOOL



Application

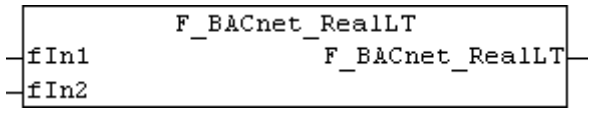
Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("<=" or "LE") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range (e.g. not-a-number values).

VAR_INPUT

```
fIn1      : REAL;
fIn2      : REAL;
```

Return value: *TRUE* = **fIn1** is less than or equal **fIn2**. *FALSE* = **fIn2** is greater than **fIn1**, or one of the two values is not in the finite range of values.

4.6.47 F_BACnet_RealLT : BOOL



Application

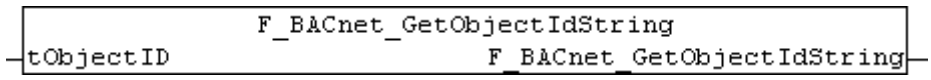
Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("<" or "LT") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range (e.g. not-a-number values).

VAR_INPUT

```
fIn1      : REAL;
fIn2      : REAL;
```

Return value: *TRUE* = **fIn1** is less than **fIn2**. *FALSE* = **fIn2** is greater than or equal to **fIn1**, or one of the two values is not in the finite range of values.

4.6.48 F_BACnet_GetObjectIdString : STRING



Application

Function for outputting the object ID of a BACnet object as a short string. The output includes the object type (short form) and the instance number of the object. The following short object names are supported:

AI	AnalogInput	AO	AnalogOutput	AV	AnalogValue	BI	BinaryInput
BO	BinaryOutput	BV	BinaryValue	CAL	Calendar	CMD	Command
DEV	Device	EE	EventEnrollment	FILE	File	GROUP	Group
LOOP	Loop	MI	MultiStateInput	MO	MultiStateOutput	NC	NotificationClass
PROG	Program	SCHED	Schedule	AVG	Averaging	MV	MultiStateValue
TLOG	TrendLog	LP	LifeSafetyPoint	LZ	LifeSafetyZone	ACC	Accumulator
PC	PulseConverter						

For unknown object type the placeholder "???" is set instead of the type ID.

VAR_INPUT

```
tObjectID      : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
```

tObjectID: Object ID (*Object_Identifier*: object type and object instance) to be represented as a string.

Example

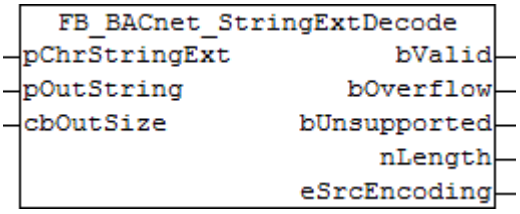
AnalogInput object number 0 → "AI:0"

MultiStateValue object number 10 → "MV:10"

Accumulator object number 5 → "ACC:5"

Unknown object number 40 → "???:40"

4.6.49 FB_BACnet_StringExtDecode



Application

Function block for decoding BACnet strings. The following codings are supported: *ASCII/UTF-8*, *UCS2*, *UCS4* and *ISO8859-1*. The output is in *Windows-1252*.

VAR_INPUT

```
pChrStringExt  : POINTER TO ST_BACnet_CharacterStringExt;
pOutString     : POINTER TO T_MaxString;
cbOutSize      : UDINT;
```

pChrStringExt: BACnet character string data structure (property value) with header and string.

pOutString: Pointer to the output STRING (Windows-1252-coded output).

cbOutSize: Byte length of the output string (maximum writeable length).

VAR_OUPUT

```
bValid         : BOOL;
bOverflow      : BOOL;
bUnsupported   : BOOL;
nLength        : UDINT;
eSrcEncoding   : E_BACNETSTRINGENCODINGTYPES;
```

bValid: Decoding was successful.

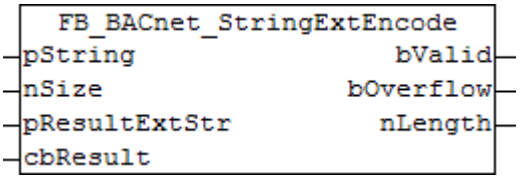
bOverflow: Input string is too long to be written to the output STRING.

bUnsupported: The input string contains a character set that is not supported.

nLength: Character length of the output STRING. The number of actually written characters.

eSrcEncoding: Detected input character set.

4.6.50 FB_BACnet_StringExtEncode



Application

Function block for coding BACnet strings. The following coding is supported: *ASCII* and *UTF-8*. The input takes place in in *Windows-1252*.

VAR_INPUT

```
pString         : POINTER TO T_MaxString;
nSize           : INT;
pResultExtStr   : POINTER TO ST_BACnet_CharacterStringExt;
cbResult        : UDINT;
```

pString: Pointer to the input STRING (Windows-1252-coded text).

nSize: Character length of the input string (LEN or SIZEOF).

pResultExtStr: Pointer to the output structure (BACnet string).

cbResult: Byte length of the output structure (maximum writeable length in bytes, SIZEOF).

VAR_OUPUT

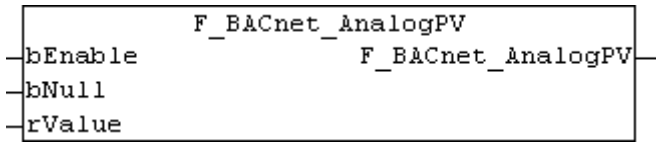
```
bValid          : BOOL;
bOverflow       : BOOL;
nLength         : UDINT;
```

bValid: Coding was successful.

bOverflow: Input string is too long to be coded to the output structure (for conversion of *Windows-1252* [1 byte per character] to *UTF-8* [1 to 3 bytes per character], the memory requirement of the output may be significantly higher than that of the input STRINGs → space requirement varies between 1 and 3 bytes for *UTF-8* coding).

nLength: Actually written number of bytes of the output structure (does not necessarily corresponds to the character length).

4.6.51 F_BACnet_AnalogPV : REAL



Application

Function for implementing a REAL value of the PLC in the process data value of a BACnet analog* object property *Present_Value*. Using this function, it is possible to write BACnet analog* objects, for example, which are only linked to a BACnet object via a primitive PLC variable (e.g.

```
rAV0 AT%Q* : REAL; (* ~(BACnet_ObjectType : AV : NOLINK)(BACnet_ObjectIdentifier : 0 : NOLINK)
(BACnet_PresentValue_Priority12 : : LINK) *)
```

).

VAR_INPUT

```
bEnable  : BOOL;
bNull    : BOOL;
rValue   : REAL;
```

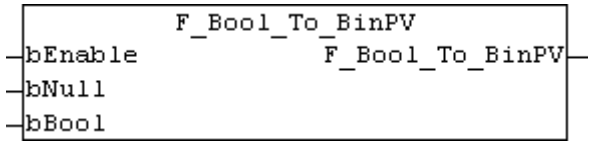
bEnable: TRUE = The process data is enabled; the value resulting from **bNull** or **rValue** is written to the corresponding BACnet object, FALSE = process data is disabled

bNull: TRUE = zero writing of the BACnet object (e.g. for deleting a priority), FALSE = write value from **rValue**

rValue: Value that is written to the BACnet object, if **bEnable** = TRUE and **bNull** = FALSE.

4.7 BACnet Logic

4.7.1 F_Bool_To_BinPV : E_BACNETBINARYPV



Application

Function for implementing a value with data type BOOL of the PLC in the process data value of a BACnet binary* object / property *Present_Value*. This function can be used to write BACnet binary* objects, for example, which are only linked to a BACnet object via a primitive PLC variable (e.g.

```
bBV0 AT%Q* : E_BACnetBinaryPV; (* ~(BACnet_ObjectType : BV : NOLINK)
(BACnet_ObjectIdentifier : 0 : NOLINK)(BACnet_PresentValue_Priority12 : : LINK) *)
```

).

The following table shows the logic:

bEnable	bNull	bBool		Return value
FALSE	-	-	→	**NOTHING**
TRUE	TRUE	-	→	**NULL**
TRUE	FALSE	FALSE	→	**INACTIVE**
TRUE	FALSE	TRUE	→	**ACTIVE**

VAR_INPUT

```
bEnable  : BOOL;
bNull    : BOOL;
bBool    : BOOL;
```

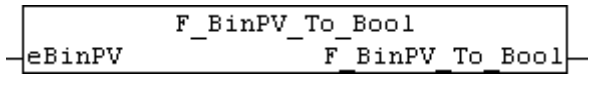
bEnable: *TRUE* = The process data is enabled; the value resulting from **bNull** or **bValue** is written to the corresponding BACnet object, *FALSE* = process data is disabled

bNull: TRUE = zero writing of the BACnet object (e.g. for deleting a priority), FALSE = write value from **bValue**

bBool: Value that is written to the BACnet object, if **bEnable** = TRUE and **bNull** = FALSE.

Return value: Function result of type E_BACNETBINARYPV [▶ 333].

4.7.2 F_BinPV_To_Bool : BOOL



Application

Function for converting a BACnet BinaryPV value to data type BOOL. The following table shows the logic:

eBinPV		Return value
INACTIVE	→	**FALSE**
ACTIVE	→	**TRUE**
NULL	→	**FALSE**
NOTHING	→	**FALSE**

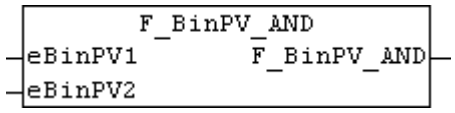
VAR_INPUT

```
eBinPV      : E_BACNETBINARYPV;
```

eBinPV: Input value.

Return value: Function result of type BOOL.

4.7.3 F_BinPV_AND : E_BACNETBINARYPV



Application

Function for logical linking of BACnet BinaryPV values. The following table shows the logic:

eBinPV1	eBinPV2	Return value
INACTIVE	INACTIVE	**INACTIVE**
INACTIVE	ACTIVE	**NOTHING**
INACTIVE	NULL	**NOTHING**
INACTIVE	NOTHING	**NOTHING**
ACTIVE	INACTIVE	**NOTHING**
ACTIVE	ACTIVE	**ACTIVE**
ACTIVE	NULL	**NOTHING**
ACTIVE	NOTHING	**NOTHING**
NULL	INACTIVE	**NOTHING**
NULL	ACTIVE	**NOTHING**
NULL	NULL	**NULL**
NULL	NOTHING	**NOTHING**
NOTHING	INACTIVE	**NOTHING**
NOTHING	ACTIVE	**NOTHING**
NOTHING	NULL	**NOTHING**
NOTHING	NOTHING	**NOTHING**

The basic rule: If both values are identical, the result is also identical; otherwise, the function returns *NOTHING*.

VAR_INPUT

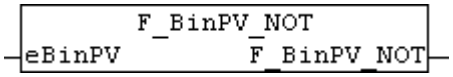
```
eBinPV1      : E_BACNETBINARYPV;
eBinPV2      : E_BACNETBINARYPV;
```

eBinPV1: Input value 1.

eBinPV2: Input value 2.

Return value: Function result of type E_BACNETBINARYPV [▶ 333].

4.7.4 F_BinPV_NOT : E_BACNETBINARYPV



Application

Function for inverting a BACnet BinaryPV value. The following table shows the logic:

eBinPV	Return value
INACTIVE	**ACTIVE**
ACTIVE	**INACTIVE**
NULL	**NULL**
NOTHING	**NOTHING**

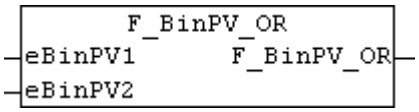
VAR_INPUT

```
eBinPV       : E_BACNETBINARYPV;
```

eBinPV: Input value.

Return value: Function result of type E_BACNETBINARYPV [▶ 333].

4.7.5 F_BinPV_OR : E_BACNETBINARYPV



Application

Function for logical linking of BACnet BinaryPV values. The following table shows the logic:

eBinPV1	eBinPV2	Return value
INACTIVE	INACTIVE	**INACTIVE**
INACTIVE	ACTIVE	**ACTIVE**
INACTIVE	NULL	**INACTIVE**
INACTIVE	NOTHING	**INACTIVE**
ACTIVE	INACTIVE	**ACTIVE**

eBinPV1	eBinPV2	Return value
ACTIVE	ACTIVE	**ACTIVE**
ACTIVE	NULL	**ACTIVE**
ACTIVE	NOTHING	**ACTIVE**
NULL	INACTIVE	**INACTIVE**
NULL	ACTIVE	**ACTIVE**
NULL	NULL	**NULL**
NULL	NOTHING	**NULL**
NOTHING	INACTIVE	**INACTIVE**
NOTHING	ACTIVE	**ACTIVE**
NOTHING	NULL	**NULL**
NOTHING	NOTHING	**NOTHING**

The basic rule: If one of the two values is *ACTIVE*, the result is *ACTIVE*. If one of the two values is *INACTIVE* and the other is not *ACTIVE*, the result is *INACTIVE*. If one of the two values is *NULL* and the other is not *ACTIVE* and not *INACTIVE*, the result is *NULL*. Otherwise, the result is *NOTHING*.

VAR_INPUT

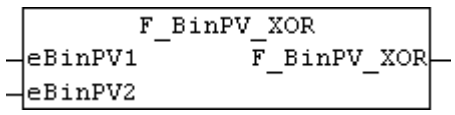
```
eBinPV1      : E_BACNETBINARYPV;
eBinPV2      : E_BACNETBINARYPV;
```

eBinPV1: Input value 1.

eBinPV2: Input value 2.

Return value: Function result of type E_BACNETBINARYPV [▶ 333]

4.7.6 F_BinPV_XOR : E_BACNETBINARYPV



Application

Function for logical linking of BACnet BinaryPV values. The following table shows the logic:

eBinPV1	eBinPV2	Return value
INACTIVE	INACTIVE	**NOTHING**
INACTIVE	ACTIVE	**ACTIVE**
INACTIVE	NULL	**INACTIVE**
INACTIVE	NOTHING	**INACTIVE**
ACTIVE	INACTIVE	**ACTIVE**
ACTIVE	ACTIVE	**NOTHING**
ACTIVE	NULL	**ACTIVE**

eBinPV1	eBinPV2	Return value
ACTIVE	NOTHING	**ACTIVE**
NULL	INACTIVE	**INACTIVE**
NULL	ACTIVE	**ACTIVE**
NULL	NULL	**NOTHING**
NULL	NOTHING	**NULL**
NOTHING	INACTIVE	**INACTIVE**
NOTHING	ACTIVE	**ACTIVE**
NOTHING	NULL	**NULL**
NOTHING	NOTHING	**NOTHING**

The basic rule: If only one of the two values is *ACTIVE*, the result is *ACTIVE*. If only one of the two values is *INACTIVE* and the other is not *ACTIVE*, the result is *INACTIVE*. If only one of the two values is *NULL* and the other is not *ACTIVE* and not *INACTIVE*, the result is *NULL*. Otherwise, the result is *NOTHING*.

VAR_INPUT

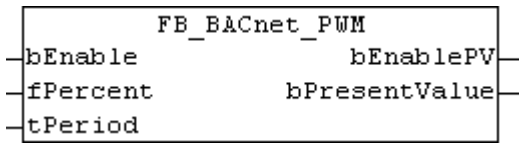
```
eBinPV1      : E_BACNETBINARYPV;
eBinPV2      : E_BACNETBINARYPV;
```

eBinPV1: Input value 1.

eBinPV2: Input value 2.

Return value: Function result of type E_BACNETBINARYPV [▶ 333]

4.8 FB_BACnet_PWM



Application

The function block FB_BACnet_PWM can be used to generate a pulsating output signal with defined switch-on and switch-off duration in percent of the cycle duration.

VAR_INPUT

```
bEnable        : BOOL;
fPercent       : REAL;
tPeriod        : TIME;
```

bEnable:

fPercent: 0...100%

tPeriod: PWM period time

VAR_OUPUT

```
bEnablePV      : BOOL;
bPresentValue      : BOOL;
```

bEnablePV:

bPresentValue:

4.9 Data types

4.9.1 BACnet_Globals

```
VAR_GLOBAL CONSTANT
  BACnet_MaxDayEntry                              := 19,
  BACnet_MaxExSchEntries                          := 9,
  BACnet_MaxTimeValues                            := 9,
  BACnet_MaxObjIdList                             := 1999,
  BACnet_MaxLogBufferEntries                      := 499,
  BACnet_MaxDestinationArray                      := 19,

  BACnet_ERR_NO_ERROR                             := 0,
  BACnet_ERR_NO_VALID_NET_ID                      := 1,
  BACnet_ERR_WRONG_MAPPING                        := 2,
  BACnet_ERR_OPERATIONAL                          := 3,
  BACnet_ERR_NO_ETH_LINK                          := 4,
  BACnet_ERR_NO_GATEWAY                           := 5,
  BACnet_ERR_WOC_TRIGGER_UNSUP                    := 6,
  BACnet_ERR_WOC_DISABLED                         := 7,
  BACnet_ERR_INVALID_OBJECTID                     := 8,
  BACnet_ERR_INVALID_OBJECTTYPE                   := 10,
  BACnet_ERR_INVALID_CYCLETIME                    := 11,
  BACnet_ERR_INVALID_PARAMETER                    := 12,

  BACnet_ERR_PERSISTENT_WRITE                     := 20,
  BACnet_ERR_INVALID_ADS_CONN                     := 21,
  BACnet_ERR_CLIENT_TIMEOUT                       := 22,
  BACnet_ERR_CLIENT_SERVICE                       := 23,
  BACnet_ERR_WOC_TRIGGER_ACTIVE                   := 24,
  BACnet_ERR_REVISION_READ                        := 25,
  BACnet_ERR_PERSISTENT_PLC_WRITE                 := 26,

  BACnet_ERR_ADS_NO_VALID_NET_ID                  := 30,
  BACnet_ERR_ADS_Data                             := 31,
  BACnet_ERR_ADS_Read                             := 32,
  BACnet_ERR_ADS_Write                            := 33,
  BACnet_ERR_ADS_ObjType                          := 34,
  BACnet_ERR_ADS_ObjListEmpty                     := 35,
  BACnet_ERR_ADS_DeviceOP                         := 36,
  BACnet_ERR_ADS_NoDevice                         := 37,
  BACnet_ERR_ADS_ObjInstance                      := 38,
  BACnet_ERR_ADS_AmsPort                          := 39,
  BACnet_ERR_ADS_ObjectId                         := 40,
  BACnet_ERR_ADS_LegacyStack                      := 41,
  BACnet_ERR_ADS_ObjIdMissmatch                   := 42,
  BACnet_ERR_ADS_BufferSize                       := 43,
  BACnet_ERR_ADS_SchedData                        := 44,
  BACnet_ERR_UnknownEventChoice                   := 45,
  BACnet_ERR_UnknownEntryChoice                   := 46,
  BACnet_ERR_ADS_TooManyObjects                   := 47,
  BACnet_ERR_ADS_WrongEntrySize                   := 48,
  BACnet_ERR_ADS_RecipientData                    := 49,
  BACnet_ERR_ADS_TooManyEntries                   := 50,
  BACnet_ERR_ADS_NotEnoughEntries                 := 51,
  BACnet_ERR_ADS_Encoding                         := 52,
  BACnet_ERR_ADS_NoServerClient                   := 53,
  BACnet_ERR_ADS_NoNotificationSink               := 54,

  BACnet_ADSTimeOut                               := t#15s,
  BACnet_PlcPersisTimeOut                         := t#2m,
  BACnet_STRING_MAXLENGTH                         := 256,

  BACnet_ADSIGRP_GET_OBJECT_LIST                  := 16#82400000,
  BACnet_ADSIGRP_CLIENT_COM_PARAMETER             := 16#84000000,
  BACnet_ADSIGRP_DEV_R_GETADSPORTBYDEVID          := 16#80000047,
  BACnet_ADSIGRP_NTFSINK_EVENTNTF_CNT             := 16#00000001,
```

```
  BACnet_ADSIGRP_NTFSINK_COVNTF_CNT                       := 16#00000002,
  BACnet_ADSIGRP_NTFSINK_EVENTNTF_READ                    := 16#00000003,
  BACnet_ADSIGRP_NTFSINK_COVNTF_READ                      := 16#00000004,
  BACnet_ADSIGRP_NTFSINK_EVENTNTF_DELETE                  := 16#00000005,
  BACnet_ADSIGRP_NTFSINK_COVNTF_DELETE                    := 16#00000006,
  BACnet_ADSIGRP_NTFSINK_EVENTNTF_INFO                    := 16#00000007,
  BACnet_ADSIGRP_NTFSINK_COVNTF_INFO                      := 16#00000008,
  BACnet_ADSIGRP_WRITE_NTFSINK_EVENTNTF_ACK               := 16#00000009,
  BACnet_ADSIGRP_NTFSINK_W_EVENTNTF_ACK_BY_SEQUENCE       := 16#0000000A,
  BACnet_ADSIGRP_NTFSINK_W_EVENTNTF_ACK_BY_PARAMETER      := 16#0000000B,
  BACnet_ADSIGRP_NTFSINK_R_EVENTNTF_READ_BY_SEQUENCE      := 16#0000000C,
  BACnet_ADSIGRP_NTFSINK_R_COVNTF_READ_BY_SEQUENCE        := 16#0000000D,
  BACnet_ADSIGRP_NTFSINK_R_EVENTNTF_DELETE_BY_SEQUENCE    := 16#0000000E,
  BACnet_ADSIGRP_NTFSINK_R_COVNTF_DELETE_BY_SEQUENCE      := 16#0000000F
END_VAR
```

`BACnet_MaxDayEntry`: Limiting the maximum number of entries per day of the week in the structure ST_BACnet_WeeklyScheduleBool [▶ 370].

`BACnet_MaxExSchEntries`: Limiting the maximum number of exception entries in the structure ST_BACnet_ExceptionScheduleBool [▶ 359].

`BACnet_MaxTimeValues`: Limiting the maximum number of time entries in the structure ST_BACnet_ExceptionScheduleEntryBool [▶ 359].

`BACnet_MaxObjIdList`: Limiting the maximum readable object IDs with the function block FB_BACnet_ObjectListProperty [▶ 290].

`BACnet_MaxLogBufferEntries`: Limiting the maximum readable log buffer entries with the function block FB_BACnet_LogBufferProperty [▶ 287].

`BACnet_MaxDestinationArray`: Limiting the maximum writeable/readable entries of a recipient list with the function block FB_BACnet_RecipientListProperty [▶ 294].

`BACnet_ERR_NO_ERROR`: Error code: No error.

`BACnet_ERR_NO_VALID_NET_ID`: Error code: No valid AMS NetID available (possible cause: the function block FB_BACnet_Adapter [▶ 153] was not fully linked in the TwinCAT System Manager; solution: check the mapping of the process data in the TwinCAT System Manager, re-link or re-run PLC automapping, if necessary).

`BACnet_ERR_WRONG_MAPPING`: Error code: Faulty process data mapping (possible cause: the corresponding block was not fully linked in the TwinCAT System Manager; solution: check the mapping of the process data in the TwinCAT System Manager, re-link or re-run PLC automapping, if necessary).

`BACnet_ERR_OPERATIONAL`: Error code: The associated BACnet Server / Client (FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241]) is not in "Operational" state (possible cause local/remote: The block FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241] is not called cyclically in the PLC program; solution: call the instance of FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241] at least once per PLC program cycle; possible cause remote: the connection to the remote server is interrupted; measures: check the connection; it may be possible to determine the cause of the interruption by means of the diagnostic [▶ 75] data).

`BACnet_ERR_NO_ETH_LINK`: Error code: The network adapters report a missing network connection (possible cause: the network cable was removed, or the remote terminal is not connected or has no power supply; measures: check the network connector & connection and the remote terminal).

`BACnet_ERR_NO_GATEWAY`: Error code: The configured gateway is not available (possible causes: the gateway configuration is incorrect; the gateway cannot be reached due to lack of network connection; the gateway is faulty/switched off; measures: check the configuration, network connection and accessibility of the gateway).

`BACnet_ERR_WOC_TRIGGER_UNSUP`: Error code: Write-On-Change trigger is not supported by the BACnet driver used.

`BACnet_ERR_WOC_DISABLED`: Error code: Write-On-Change could not be triggered, since Write-On-Change is disabled.

BACnet_ERR_INVALID_OBJECTID: Error code: Invalid object ID detected (possible causes: the object_identifier property was not properly linked in the TwinCAT System Manager; the object type used does not match the object type that is encoded in the object ID; measures: check/renew the mapping in the TwinCAT System Manager or re-run PLC automapping; compare the type of the linked object in the TwinCAT System Manager and the function block type used in the PLC program).

BACnet_ERR_INVALID_OBJECTTYPE: Error code: see **BACnet_ERR_INVALID_OBJECTID**.

BACnet_ERR_INVALID_CYCLETIME: Error code: Invalid PLC cycle time detected. The cycle time must not be less than/equal to zero.

BACnet_ERR_INVALID_PARAMETER: Error code: The transferred parameters are not correct.

BACnet_ERR_PERSISTENT_WRITE: Error code: Persistent data of the BACnet driver could not be written (a possible cause could be a lack of memory or inadequate router memory).

BACnet_ERR_INVALID_ADS_CONN: Error code: The ADS connection to the server/client could not be established or is incorrectly configured.

BACnet_ERR_CLIENT_TIMEOUT: Error code: The connection to a remote server (client) was interrupted (see diagnostic [▶ 75] data).

BACnet_ERR_CLIENT_SERVICE: Error code: A service of a remote server (client) could not be executed (possible causes: an object configured under a client object does not exist on the remote server; COV-P was configured under the client but is not supported by a remote server, for example; see diagnostic [▶ 75] data).

BACnet_ERR_WOC_TRIGGER_ACTIVE: Error code: WOC trigger is active, while another edge was detected at the input.

BACnet_ERR_REVISION_READ: Error code: The revision number of the BACnet driver could not be read.

BACnet_ERR_PERSISTENT_PLC_WRITE: Error code: Persistent PLC data could not be written (a possible cause could be a lack of memory or inadequate router memory).

BACnet_ERR_ADS_NO_VALID_NET_ID: Error code: No valid AMS NetID. See also **BACnet_ERR_NO_VALID_NET_ID**.

BACnet_ERR_ADS_Data: Error code: The received data length does not match the expected data length.

BACnet_ERR_ADS_Read: Error code: Error at the internal ADSREAD block (see error output of the subordinate ADSREAD or FW_AdsRead instance for an error description).

BACnet_ERR_ADS_Write: Error code: Error at the internal ADSWRITE block (see error output of the lower-level ADSWRITE or FW_AdsWrite instance for an error description).

BACnet_ERR_ADS_ObjType: Error code: not used.

BACnet_ERR_ADS_Data: Error code: The received object list contains no entry. However, at least one property per BACnet server is expected (device object).

BACnet_ERR_ADS_DeviceOP: Error code: The ADS connection to the server/client cannot be established, because the associated device object does not report "operational" (see FB_BACnet_Device [▶ 196] or FB_BACnet_RemoteDevice [▶ 241] status).

BACnet_ERR_ADS_NoDevice: Error code: No device object was found while reading the object list of the device object.

BACnet_ERR_ADS_ObjInstance: Error code: An incorrectly coded object instance was detected while reading the object list of the device object.

BACnet_ERR_ADS_AmsPort: Error code: The configured AMS port is not in the valid range (valid range: >= 1000).

BACnet_ERR_ADS_ObjectId: Error code: see BACnet_ERR_INVALID_OBJECTID [▶ 328].

`BACnet_ERR_ADS_LegacyStack`: Error code: The BACnet driver used does not support the service used (BACnet driver revision 6 was detected).

`BACnet_ERR_ADS_ObjIdMissmatch`: Error code: The object instance of the object list of the device object does not match the associated device ID (possible cause: incorrect mapping in the TwinCAT System Manager; solution: check the mapping of the device object and the BACnet adapter in the TwinCAT System Manager; re-run the PLC automapping, if necessary).

`BACnet_ERR_ADS_BufferSize`: Error code: The data to be read exceed the size of the global ADS buffer in the PLC (default: approx. 8 kbyte). The size of the global ADS buffer in the PLC cam be adapted in the structure ST_BACnet_GlobalAdsBuffer [▶ 360].

`BACnet_ERR_ADS_SchedData`: Error code: Incorrectly coded data detected while reading the property weekly_schedule.

`BACnet_ERR_UnknownEventChoice`: Error code: Unknown event type detected while reading the property exception_schedule (known: calendar reference and calendar entry).

`BACnet_ERR_UnknownEntryChoice`: Error code: Unknown entry type detected while reading the property exception_schedule (known: date, date range and day of the week).

`BACnet_ERR_ADS_TooManyObjects`: Error code: Too many objects in the object list. The object list could only be read partially.

`BACnet_ERR_ADS_WrongEntrySize`: Error code: The received entry (text or data) is longer/shorter than expected.

`BACnet_ERR_ADS_RecipientData`: Error code: The data of the property recipient_list to be written is incorrect (possible cause: the receiver object ID does not match the device object type).

`BACnet_ERR_ADS_TooManyEntries`: Error code: The number of read entries exceeds the maximum number of expected entries (possible cause: when reading the property event_message_texts more than 3 entries were read - the maximum permitted number is 3).

`BACnet_ERR_ADS_NotEnoughEntries`: Error code: The number of read entries is below the minimum number of expected entries (possible cause: when reading the property event_message_texts less than 3 entries were read - the minimum number is 3).

`BACnet_ERR_ADS_Encoding`: Error code: The string encoding used is not supported, or an error was detected in the encoding.

BACnet_ERR_ADS_NoServerClient: Error code: The ADS connection used neither represents a local server or a client, although client or server is expected (see ST_BACnet_AdsConnection [▶ 355]).

`BACnet_ERR_ADS_NoNotificationSink`: Error code: The ADS connection used does not represent a notification sink, although a notification sink is expected (see ST_BACnet_AdsConnection [▶ 355]).

`BACnet_ADSTimeOut`: Time constant: Maximum waiting time for an ADS link in milliseconds (`TIME`, standard `T#15s`).

`BACnet_PlcPersisTimeOut`: Time constant: Maximum waiting time during writing of the persistent PLC data in milliseconds (`TIME`, standard `T#2m`).

`BACnet_STRING_MAXLENGTH`: Character length: Maximum length of a BACnet string in bytes (INT, default 256).

`BACnet_ADSIGRP_GET_OBJECT_LIST`: Index group constant.

`BACnet_ADSIGRP_CLIENT_COM_PARAMETER`: Index group constant.

`BACnet_ADSIGRP_DEV_R_GETADSPORTBYDEVID`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_EVENTNTF_CNT`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_COVNTF_CNT`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_EVENTNTF_READ`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_COVNTF_READ`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_EVENTNTF_DELETE`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_COVNTF_DELETE`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_EVENTNTF_INFO`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_COVNTF_INFO`: Index group constant.

`BACnet_ADSIGRP_WRITE_NTFSINK_EVENTNTF_ACK`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_W_EVENTNTF_ACK_BY_SEQUENCE`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_W_EVENTNTF_ACK_BY_PARAMETER`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_R_EVENTNTF_READ_BY_SEQUENCE`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_R_COVNTF_READ_BY_SEQUENCE`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_R_EVENTNTF_DELETE_BY_SEQUENCE`: Index group constant.

`BACnet_ADSIGRP_NTFSINK_R_COVNTF_DELETE_BY_SEQUENCE`: Index group constant.

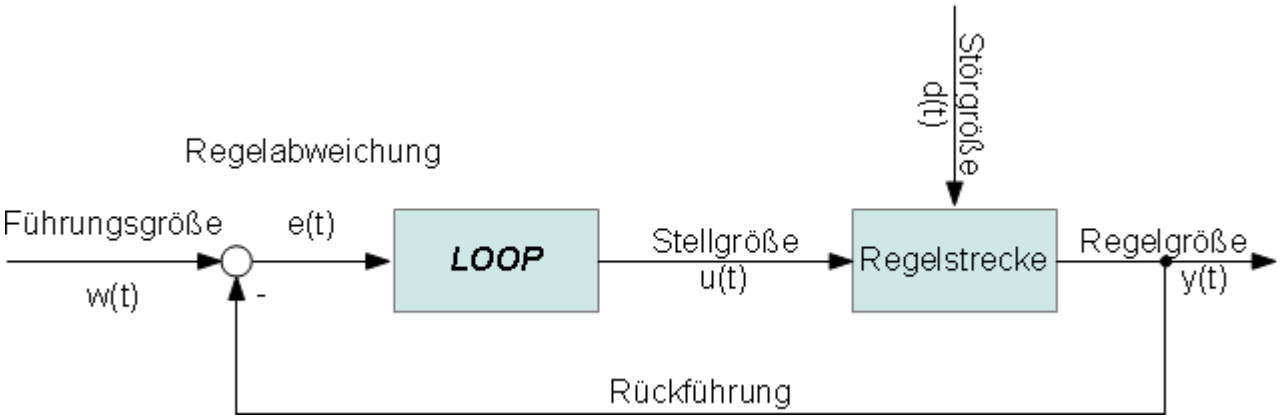
4.9.2 E_BACNETACTION

PLC mapping of BACnet data type BACnetAction. See BACnet specification DIN EN ISO 16484-5 for property *Action*.

```
TYPE E_BACNETACTION :
(
  BACnetAction_direct   := 0,
  BACnetAction_reverse  := 1,
)
END_TYPE
```

`BACnetAction_direct`: The controller output acts directly (positive input error e results in increasing output [e = w - y])

`BACnetAction_reverse`: The controller output acts inversely (the input error e is inverted [e = y - w])



4.9.3 E_BACNETADAPTERSTATUS

Status of the BACnet adapter.

```
TYPE E_BACNETADAPTERSTATUS :
(
  BACnetAdapterStatus_Init            := 16#0,
  BACnetAdapterStatus_CheckIpAddr     := 16#1,
  BACnetAdapterStatus_CheckParam      := 16#2,
  BACnetAdapterStatus_GetGatewayMAC   := 16#3,
  BACnetAdapterStatus_WaitGatewayMAC  := 16#4,
```

```
  BACnetAdapterStatus_Complete        := 16#8,
)
END_TYPE
```

`BACnetAdapterStatus_Init`: Initialization has started.

`BACnetAdapterStatus_CheckIpAddr`: Checking the IP address.

`BACnetAdapterStatus_CheckParam`: Checking the parameters.

`BACnetAdapterStatus_GetGatewayMAC`: Check the MAC address of the gateway.

`BACnetAdapterStatus_WaitGatewayMAC`: Wait while the MAC address of the gateway is determined (gateway reachable?).

`BACnetAdapterStatus_Complete`: Initialisation complete and ready.

4.9.4 E_BACNETBINARYPV

PLC mapping of BACnet data type BACnetBinaryPV. See BACnet specification DIN EN ISO 16484-5 for property *Present_Value* of binary* objects. For the internal processing of the commandable execution of the property of type BACnetBinaryPV, two further states were introduced for the property (NULL and Nothing). These additional states are not visible for BACnet and are only required for processing of the cyclic process data.

```
TYPE E_BACNETBINARYPV :
(
  BACnetBinaryPV_inactive  := 0,
  BACnetBinaryPV_active    := 1,
  BACnetBinaryPV_NULL      := 2,
  BACnetBinaryPV_Nothing   := 16#FFFF
)
END_TYPE
```

`BACnetBinaryPV_inactive`: Value of Property *Present_Value* is INACTIVE (FALSE)

`BACnetBinaryPV_active`: Value of Property *Present_Value* is ACTIVE (TRUE)

`BACnetBinaryPV_NULL`: Value of property is *Present_Value* is written as NULL (no value; for commandable properties entry in the corresponding property *Priority_Array* is deleted)

`BACnetBinaryPV_Nothing`: Value of property *Present_Value* remains unchanged (no action; process data is not evaluated)

4.9.5 E_BACNETDATATYPES

List of possible BACnet data types (excerpt).

```
TYPE E_BACNETDATATYPES :
(
  BACnetDataType_Null                              := 0,
  BACnetDataType_Bool                              := 1,
  BACnetDataType_UnsignedInteger                   := 2,
  BACnetDataType_SignedInteger                     := 3,
  BACnetDataType_Real                              := 4,
  BACnetDataType_Double                            := 5,
  BACnetDataType_OctetString                       := 6,
  BACnetDataType_CharacterStringExt                := 7,
  BACnetDataType_BitString                         := 8,
  BACnetDataType_Enumerated                        := 9,
  BACnetDataType_Date                              := 10,
  BACnetDataType_Time                              := 11,
  BACnetDataType_ObjectIdentifier                  := 12,
  BACnetDataType_DateTime                          := 14,
  BACnetDataType_ArrayIndex                        := 15,
  BACnetDataType_CalendarEntry                     := 16,
  BACnetDataType_ObjectType                        := 17,
  BACnetDataType_EventTransitionBits               := 18,
  BACnetDataType_ActionList                        := 19,
  BACnetDataType_StatusFlags                       := 20,
  BACnetDataType_EventState                        := 21,
  BACnetDataType_Reliability                       := 22,
```

```
BACnetDataType_Polarity                          := 23,
BACnetDataType_EngineeringUnits                  := 25,
BACnetDataType_PriorityValue                     := 26,
BACnetDataType_LimitEnable                       := 27,
BACnetDataType_NotifyType                        := 29,
BACnetDataType_TimeStamp                         := 30,
BACnetDataType_DeviceObjectPropertyReference     := 31,
BACnetDataType_DeviceStatus                      := 32,
BACnetDataType_ServicesSupported                 := 33,
BACnetDataType_ObjectTypesSupported              := 34,
BACnetDataType_Segmentation                      := 35,
BACnetDataType_VTClass                           := 36,
BACnetDataType_VTSession                         := 37,
BACnetDataType_SessionKey                        := 38,
BACnetDataType_Recipient                         := 39,
BACnetDataType_AddressBinding                    := 40,
BACnetDataType_COVSubscription                   := 41,
BACnetDataType_EventType                         := 42,
BACnetDataType_EventParameter                    := 43,
BACnetDataType_FileAccessMethod                  := 44,
BACnetDataType_ReadAccessSpecification           := 45,
BACnetDataType_ReadAccessResult                  := 46,
BACnetDataType_ObjectPropertyReference           := 47,
BACnetDataType_SetpointReference                 := 48,
BACnetDataType_Destination                       := 49,
BACnetDataType_ProgramState                      := 50,
BACnetDataType_ProgramRequest                    := 51,
BACnetDataType_ProgramError                      := 52,
BACnetDataType_DateRange                         := 53,
BACnetDataType_DailySchedule                     := 54,
BACnetDataType_SpecialEvent                      := 55,
BACnetDataType_ClientCOV                         := 56,
BACnetDataType_LogRecord                         := 57,
BACnetDataType_LifeSafetyState                   := 58,
BACnetDataType_LifeSafetyMode                    := 59,
BACnetDataType_SilencedState                     := 60,
BACnetDataType_LifeSafetyOperation               := 61,
BACnetDataType_BinaryPV                          := 62,
BACnetDataType_DaysOfWeek                        := 63,
BACnetDataType_WeekNDay                          := 64,
BACnetDataType_TimeValue                         := 65,
BACnetDataType_Value                             := 66,
BACnetDataType_Address                           := 67,
BACnetDataType_PropertyIdentifier                := 68,
BACnetDataType_EnumerationValueType              := 69,
BACnetDataType_BitFieldBit                       := 70,
BACnetDataType_LogDatum                          := 71,
BACnetDataType_AnyType                           := 72,
BACnetDataType_PresentValue                      := 73,
BACnetDataType_ContextTag                        := 74,
BACnetDataType_ValueChoice                       := 75,
BACnetDataType_LogStatus                         := 76,
BACnetDataType_RecipientProcess                  := 77,
BACnetDataType_SubscribeCOVPropertyRequest       := 78,
BACnetDataType_PropertyReference                 := 79,
BACnetDataType_PropertyResult                    := 80,
BACnetDataType_DeviceObjectPropertyValue         := 81,
BACnetDataType_COVNotificationRequest            := 82,
BACnetDataType_EventNotificationRequest          := 83,
BACnetDataType_NotificationParameters            := 84,
BACnetDataType_Change_of_Bitstring               := 85,
BACnetDataType_Change_of_State                   := 86,
BACnetDataType_Change_of_Value                   := 87,
BACnetDataType_Command_failure                   := 88,
BACnetDataType_Floating_limit                    := 89,
BACnetDataType_Out_of_Range                      := 90,
BACnetDataType_PropertyValue                     := 91,
BACnetDataType_Complex_Tag                       := 92,
BACnetDataType_Change_of_life_safety             := 93,
BACnetDataType_Extended                          := 94,
BACnetDataType_Buffer_ready                      := 95,
BACnetDataType_Unsigned_range                    := 96,
BACnetDataType_PropertyResultValue               := 97,
BACnetDataType_ServiceCOVPropSubscription        := 98,
BACnetDataType_ServiceCOVSubscription            := 99,
BACnetDataType_COVNotification                   := 100,
BACnetDataType_AcknowledgeAlarmRequest           := 101,
BACnetDataType_VTOpenRequest                     := 102,
BACnetDataType_Prescale                          := 103,
BACnetDataType_Scale                             := 104,
```

```
BACnetDataType_Action                                := 105,
BACnetDataType_Error                                 := 106,
BACnetDataType_ErrorType                             := 107,
BACnetDataType_ErrorClass                            := 108,
BACnetDataType_ErrorCode                             := 109,
BACnetDataType_RejectReason                          := 110,
BACnetDataType_AbortReason                           := 111,
BACnetDataType_BDTEntry                              := 112,
BACnetDataType_IpEntry                               := 113,
BACnetDataType_MaskEntry                             := 114,
BACnetDataType_EthernetAddress                       := 115,
BACnetDataType_LonTalkAddress                        := 116,
BACnetDataType_LonTalkNeuronId                       := 117,
BACnetDataType_NodeType                              := 118,
BACnetDataType_DeviceObjectReference                 := 119,
BACnetDataType_ActionCommand                         := 120,
BACnetDataType_GetEventInformation                   := 121,
BACnetDataType_GetEnrollmentSummaryCriteria          := 122,
BACnetDataType_GetEnrollmentSummaryCriteriaResponse  := 123,
BACnetDataType_DeviceCommunicationControl            := 124,
BACnetDataType_Int16                                 := 125,
BACnetDataType_UInt16                                := 126,
BACnetDataType_UInt8                                 := 127,
BACnetDataType_TypeInt8                              := 128,
BACnetDataType_Byte                                  := 129,
BACnetDataType_FDTEntry                              := 130,
BACnetDataType_Address_3                             := 131,
BACnetDataType_Address_4                             := 132,
BACnetDataType_Address_5                             := 133,
BACnetDataType_PersistentDataState                   := 134,
BACnetDataType_FallbackRealValue                     := 135,
BACnetDataType_FallbackBinaryValue                   := 136,
BACnetDataType_RealNull                              := 137,
BACnetDataType_EnumeratedNull                        := 138,
BACnetDataType_UnsignedIntegerNull                   := 139,
BACnetDataType_Unknown                               := 140,
BACnetDataType_DiagnosisPerformance                  := 200,
BACnetDataType_DiagnosisEthStatistics                := 201,
BACnetDataType_FrameStatistics                       := 202,
BACnetDataType_Info                                  := 203,
BACnetDataType_Diagnosis                             := 204,
BACnetDataType_TcIoEthStatistic                      := 205,
BACnetDataType_TcIoEthTxRxErrorCount                 := 206,
BACnetDataType_TcIoEthPortStatistic                  := 207,
BACnetDataType_ServerStatistics                      := 208,
BACnetDataType_FrameServicesDiag                     := 209,
BACnetDataType_UnsignedIntegerArr                    := 258,
BACnetDataType_CharacterStringExtArr                 := 263,
BACnetDataType_ObjectIdentifierArr                   := 268,
BACnetDataType_CalendarEntryArr                      := 272,
BACnetDataType_ActionListArr                         := 275,
BACnetDataType_PriorityValueArr                      := 282,
BACnetDataType_TimeStampArr                          := 286,
BACnetDataType_DeviceObjectPropertyReferenceArr      := 287,
BACnetDataType_VTClassArr                            := 292,
BACnetDataType_VTSessionArr                          := 293,
BACnetDataType_SessionKeyArr                         := 294,
BACnetDataType_RecipientArr                          := 295,
BACnetDataType_AddressBindingArr                     := 296,
BACnetDataType_COVSubscriptionArr                    := 297,
BACnetDataType_ReadAccessSpecificationArr            := 301,
BACnetDataType_ReadAccessResultArr                   := 302,
BACnetDataType_DestinationArr                        := 305,
BACnetDataType_SpecialEventArr                       := 311,
BACnetDataType_LogRecordArr                          := 313,
BACnetDataType_LifeSafetyStateArr                    := 314,
BACnetDataType_TimeValueArr                          := 321,
BACnetDataType_UnsignedIntegerList                   := 514,
BACnetDataType_CharacterStringExtList                := 519,
BACnetDataType_ReadAccessSpecificationList           := 557,
BACnetDataType_ReadAccessResultList                  := 558,
BACnetDataType_SpecialEventList                      := 567,
BACnetDataType_LogRecordList                         := 569,
BACnetDataType_TimeValueList                         := 577,
BACnetDataType_ValueList                             := 578,
BACnetDataType_LogDatumList                          := 583,
BACnetDataType_PropertyReferenceList                 := 591,
BACnetDataType_PropertyResultList                    := 592,
BACnetDataType_COVNotificationRequestList            := 594,
BACnetDataType_EventNotificationRequestList          := 595,
```

```
  BACnetDataType_PropertyValueList                    := 603,
  BACnetDataType_DailyScheduleList                    := 1089,
)
END_TYPE
```

4.9.6 E_BACNETDAYSOFWEEKBITS

Bit assignment of the days of the week

```
TYPE E_BACNETDAYSOFWEEKBITS :
(
  BACnetDaysOfWeekBits_Monday     := 0,
  BACnetDaysOfWeekBits_Tuesday    := 1,
  BACnetDaysOfWeekBits_Wednesday  := 2,
  BACnetDaysOfWeekBits_Thursday   := 3,
  BACnetDaysOfWeekBits_Friday     := 4,
  BACnetDaysOfWeekBits_Saturday   := 5,
  BACnetDaysOfWeekBits_Sunday     := 6,
)
END_TYPE
```

`BACnetDaysOfWeekBits_Monday`: Monday.

`BACnetDaysOfWeekBits_Tuesday`: Tuesday.

`BACnetDaysOfWeekBits_Wednesday`: Wednesday.

`BACnetDaysOfWeekBits_Thursday`: Thursday.

`BACnetDaysOfWeekBits_Friday`: Friday.

`BACnetDaysOfWeekBits_Saturday`: Saturday.

`BACnetDaysOfWeekBits_Sunday`: Sunday.

4.9.7 E_BACNETDEVICESTATUS

Status of the BACnet server object (see BACnet specification DIN EN ISO 16484-5 for BACnet device object and property *System_Status*).

```
TYPE E_BACNETDEVICESTATUS :
(
  BACnetDeviceStatus_Operational          := 0,
  BACnetDeviceStatus_OperationalReadOnly  := 1,
  BACnetDeviceStatus_DownloadRequired     := 2,
  BACnetDeviceStatus_DownloadInProgress   := 3,
  BACnetDeviceStatus_NonOperational       := 4,
  BACnetDeviceStatus_BackupInProgress     := 5,
)
END_TYPE
```

`BACnetDeviceStatus_Operational`: Ready to operate.

`BACnetDeviceStatus_OperationalReadOnly`: Read-only access to properties.

`BACnetDeviceStatus_DownloadRequired`: Configuration loading required.

`BACnetDeviceStatus_DownloadInProgress`: Configuration loading in progress.

`BACnetDeviceStatus_NonOperational`: Not ready to operate.

`BACnetDeviceStatus_BackupInProgress`: Data backup in progress.

4.9.8 E_BACNETEVENTSTATE

PLC mapping of BACnet data type BACnetEventState. See BACnet specification DIN EN ISO 16484-5 for property *Event_State*.

```
TYPE E_BACNETEVENTSTATE :
(
  BACnetEventState_state_normal       := 0,
```

```
  BACnetEventState_fault              := 1,
  BACnetEventState_offnormal          := 2,
  BACnetEventState_high_limit         := 3,
  BACnetEventState_low_limit          := 4,
  BACnetEventState_life_safety_alarm  := 5
)
END_TYPE
```

`BACnetEventState_state_normal`: Object state normal (*Reliability* property equal NO_FAULT_DETECTED).

`BACnetEventState_fault`: Object state in error (*Reliability* property not equal NO_FAULT_DETECTED).

`BACnetEventState_offnormal`: Object state off-normal (*Reliability* property equal NO_FAULT_DETECTED).

`BACnetEventState_high_limit`: Object state upper limit reached/exceeded (property *High_Limit*).

`BACnetEventState_low_limit`: Object state lower limit reached/below (property *Low_Limit*).

`BACnetEventState_life_safety_alarm`: Object state life-safety-alarm.

4.9.9 E_BACNETEVENTSTATE

PLC mapping of BACnet data type BACnetEventState. See BACnet specification DIN EN ISO 16484-5 for property *Event_State*.

```
TYPE E_BACNETEVENTSTATE :
(
  BACnetEventState_state_normal       := 0,
  BACnetEventState_fault              := 1,
  BACnetEventState_offnormal          := 2,
  BACnetEventState_high_limit         := 3,
  BACnetEventState_low_limit          := 4,
  BACnetEventState_life_safety_alarm  := 5
)
END_TYPE
```

`BACnetEventState_state_normal`: Object state normal (*Reliability* property equal NO_FAULT_DETECTED).

`BACnetEventState_fault`: Object state in error (*Reliability* property not equal NO_FAULT_DETECTED).

`BACnetEventState_offnormal`: Object state off-normal (*Reliability* property equal NO_FAULT_DETECTED).

`BACnetEventState_high_limit`: Object state upper limit reached/exceeded (property *High_Limit*).

`BACnetEventState_low_limit`: Object state lower limit reached/below (property *Low_Limit*).

`BACnetEventState_life_safety_alarm`: Object state life-safety-alarm.

4.9.10 E_BACNETEVENTTRANSITIONBIT

Bit assignment of the event transition flags [▶ 358] (properties *Event_Enable* and *Acked_Transitions*).

```
TYPE E_BACNETEVENTTRANSITIONBIT :
(
  BACnetEventTransitionBit_to_offnormal  := 0,
  BACnetEventTransitionBit_to_fault      := 1,
  BACnetEventTransitionBit_to_normal     := 2
)
END_TYPE
```

4.9.11 E_BACNETEVENTTYPE

PLC mapping of BACnet data type BACnetEventType. See BACnet specification DIN EN ISO 16484-5 for property *Event_Type*.

```
TYPE E_BACNETEVENTTYPE :
(
  BACnetEventType_change_of_bitstring        := 0,
  BACnetEventType_change_of_state            := 1,
  BACnetEventType_change_of_value            := 2,
  BACnetEventType_command_failure            := 3,
  BACnetEventType_floating_limit             := 4,
  BACnetEventType_out_of_range               := 5,
  BACnetEventType_complex_event_type         := 6,
  BACnetEventType_deprecated7                := 7,
  BACnetEventType_change_of_life_safety      := 8,
  BACnetEventType_enumeration                := 9,
  BACnetEventType_buffer_ready               := 10,
  BACnetEventType_unsigned_range             := 11,
  BACnetEventType_reserved12                 := 12,
  BACnetEventType_access_event               := 13,
  BACnetEventType_double_out_of_range        := 14,
  BACnetEventType_signed_out_of_range        := 15,
  BACnetEventType_unsigned_out_of_range      := 16,
  BACnetEventType_change_of_characterstring  := 17,
  BACnetEventType_change_of_state_flags      := 18,
)
END_TYPE
```

`BACnetEventType_change_of_bitstring`: CHANGE_OF_BITSTRING

`BACnetEventType_change_of_state`: CHANGE_OF_STATE **(COS)**

`BACnetEventType_change_of_value`: CHANGE_OF_VALUE **(COV)**

`BACnetEventType_command_failure`: COMMAND_FAILURE

`BACnetEventType_floating_limit`: FLOATING_LIMIT

`BACnetEventType_out_of_range`: OUT_OF_RANGE

`BACnetEventType_complex_event_type`: COMPLEX_EVENT_TYPE

`BACnetEventType_deprecated7`: *no longer used*

`BACnetEventType_change_of_life_safety`: CHANGE_OF_LIFE_SAFETY

`BACnetEventType_enumeration`: ENUMERATION

`BACnetEventType_buffer_ready`: BUFFER_READY

`BACnetEventType_unsigned_range`: UNSIGNED_RANGE

`BACnetEventType_reserved12`: *reserved for future use*

`BACnetEventType_access_event`: ACCESS_EVENT

`BACnetEventType_double_out_of_range`: DOUBLE_OUT_OF_RANGE

`BACnetEventType_signed_out_of_range`: SIGNED_OUT_OF_RANGE

`BACnetEventType_unsigned_out_of_range`: UNSIGNED_OUT_OF_RANGE

`BACnetEventType_change_of_characterstring`: CHANGE_OF_CHARACTERSTRING

`BACnetEventType_change_of_state_flags`: CHANGE_OF_STATE_FLAGS

4.9.12 E_BACNETFILEACCESSMETHOD

PLC mapping of BACnet data type BACnetFileAccessMethod. See BACnet specification DIN EN ISO 16484-5 for property *File_Access_Method*.

```
TYPE E_BACNETFILEACCESSMETHOD :
(
  BACnetFileAccessMethod_record_access  := 0,
  BACnetFileAccessMethod_stream_access  := 1,
)
END_TYPE
```

`BACnetFileAccessMethod_record_access`: File access using fixed data blocks.

`BACnetFileAccessMethod_stream_access`: File access using data stream (default method).

4.9.13 E_BACNETLIFESAFETYMODE

PLC mapping of BACnet data type BACnetLifeSafetyMode. See BACnet specification DIN EN ISO 16484-5 for property *Mode* and *Accepted_Modes*.

```
TYPE E_BACNETLIFESAFETYMODE :
(
  BACnetLifeSafetyMode_off                          := 0,
  BACnetLifeSafetyMode_on                           := 1,
  BACnetLifeSafetyMode_test                         := 2,
  BACnetLifeSafetyMode_manned                       := 3,
  BACnetLifeSafetyMode_unmanned                     := 4,
  BACnetLifeSafetyMode_armed                        := 5,
  BACnetLifeSafetyMode_disarmed                     := 6,
  BACnetLifeSafetyMode_prearmed                     := 7,
  BACnetLifeSafetyMode_slow                         := 8,
  BACnetLifeSafetyMode_fast                         := 9,
  BACnetLifeSafetyMode_disconnected                 := 10,
  BACnetLifeSafetyMode_enabled                      := 11,
  BACnetLifeSafetyMode_disabled                     := 12,
  BACnetLifeSafetyMode_automatic_release_disabled   := 13,
  BACnetLifeSafetyMode_mode_default                 := 14,
)
END_TYPE
```

`BACnetLifeSafetyMode_off`: OFF

`BACnetLifeSafetyMode_on`: ON

`BACnetLifeSafetyMode_test`: TEST

`BACnetLifeSafetyMode_manned`: MANNED

`BACnetLifeSafetyMode_unmanned`: UNMANNED

`BACnetLifeSafetyMode_armed`: ARMED

`BACnetLifeSafetyMode_disarmed`: DISARMED

`BACnetLifeSafetyMode_prearmed`: PREARMED

`BACnetLifeSafetyMode_slow`: SLOW

`BACnetLifeSafetyMode_fast`: FAST

`BACnetLifeSafetyMode_disconnected`: DISCONNECTED

`BACnetLifeSafetyMode_enabled`: ENABLED

`BACnetLifeSafetyMode_disabled`: DISABLED

`BACnetLifeSafetyMode_automatic_release_disabled`: AUTOMATIC_RELEASE_DISABLED

`BACnetLifeSafetyMode_mode_default`: MODE_DEFAULT

4.9.14 E_BACNETLIFESAFETYOPERATION

PLC mapping of BACnet data type BACnetLifeSafetyOperation. See BACnet specification DIN EN ISO 16484-5 for property *Operation_Expected*.

```
TYPE E_BACNETLIFESAFETYOPERATION :
(
  BACnetLifeSafetyOperation_none              := 0,
  BACnetLifeSafetyOperation_silence           := 1,
  BACnetLifeSafetyOperation_silence_audible   := 2,
  BACnetLifeSafetyOperation_silence_visual    := 3,
  BACnetLifeSafetyOperation_reset             := 4,
```

```
  BACnetLifeSafetyOperation_reset_alarm        := 5,
  BACnetLifeSafetyOperation_reset_fault        := 6,
  BACnetLifeSafetyOperation_unsilence          := 7,
  BACnetLifeSafetyOperation_unsilence_audible  := 8,
  BACnetLifeSafetyOperation_unsilence_visual   := 9,
)
END_TYPE
```

`BACnetLifeSafetyOperation_none`: NONE

`BACnetLifeSafetyOperation_silence`: SILENCE

`BACnetLifeSafetyOperation_silence_audible`: SILENCE_AUDIBLE

`BACnetLifeSafetyOperation_silence_visual`: SILENCE_VISUAL

`BACnetLifeSafetyOperation_reset`: RESET

`BACnetLifeSafetyOperation_reset_alarm`: RESET_ALARM

`BACnetLifeSafetyOperation_reset_fault`: RESET_FAULT

`BACnetLifeSafetyOperation_unsilence`: UNSILENCE

`BACnetLifeSafetyOperation_unsilence_audible`: UNSILENCE_AUDIBLE

`BACnetLifeSafetyOperation_unsilence_visual`: UNSILENCE_VISUAL

4.9.15 E_BACNETLIMITENABLEBITS

Bit assignment of the Limit-Enable-Flags [▶ 362] (property *Limit_Enable*).

```
TYPE E_BACNETLIMITENABLEBITS :
(
  BACnetLimitEnableBits_lowLimit   := 0,
  BACnetLimitEnableBits_highLimit  := 1
)
END_TYPE
```

4.9.16 E_BACNETLOGGINGTYPE

PLC mapping of BACnet data type BACnetLoggingType. See BACnet specification DIN EN ISO 16484-5 for property *Logging_Type*.

```
TYPE E_BACNETLOGGINGTYPE :
(
  BACnetLoggingType_cov,
  BACnetLoggingType_polled,
  BACnetLoggingType_triggered
)
END_TYPE
```

`BACnetLoggingType_cov`: COV: Change on value: Changes in value are logged (see properties *Client_Cov_Increment* and *Cov_Resubscription_Interval*).

`BACnetLoggingType_polled`: POLLED: Values are logged with a fixed distance (see properties *Log_Interval*, *Interval_Offset* and *Align_Intervals*).

`BACnetLoggingType_triggered`: TRIGGERED: Values are logged with the aid of a Trigger property (see online property *Trigger*)

4.9.17 E_BACNETLOOPMODE

Operating modes of PLC LOOP object FB_BACnet_LOOP [▶ 201].

```
TYPE E_BACNETLOOPMODE :
(
  BACnetLoopMode_None,
  BACnetLoopMode_Ideal,
```

```
  BACnetLoopMode_Standard
)
END_TYPE
```

`BACnetLoopMode_None`: None: Invalid LOOP parameters. The mode cannot be identified unambiguously.

`BACnetLoopMode_Ideal`: Ideal: The controller is operated in ideal form: The P, I and D components act in parallel; the parameters for P, I and D are factors and have the unit [1] *Other_no_units* (Kp, Ki and Kd).

`BACnetLoopMode_Standard`: Standard: The controller is operated in standard form: The P and I / P and D components act in series; the parameter for P is a factor and has the unit [1] *Other_no_units* (Kp), the parameters for I and D are time-based and have the unit [s] *Time_seconds* (Tn and Tv).

4.9.18 E_BACNETNOTIFYTYPE

PLC mapping of BACnet data type BACnetNotifyType. See BACnet specification DIN EN ISO 16484-5 for property *Notify_Type*.

```
TYPE E_BACNETNOTIFYTYPE :
(
  BACnetNotifyType_alarm             := 0,
  BACnetNotifyType_notify_event      := 1,
  BACnetNotifyType_ack_notification  := 2
)
END_TYPE
```

`BACnetNotifyType_alarm`: ALARM

`BACnetNotifyType_notify_event`: NOTIFY_EVENT

`BACnetNotifyType_ack_notification`: ACK_NOTIFICATION

4.9.19 E_BACNETOBJECTSUPPORTEDBITS

Bit assignment of the Object Types-Supported flags [▶ 364] (property *Protocol_Object_Types_Supported*).

```
TYPE E_BACNETOBJECTSUPPORTEDBITS :
(
  BACnetObjectSupportedBits_AnalogInput           := 0,
  BACnetObjectSupportedBits_AnalogOutput          := 1,
  BACnetObjectSupportedBits_AnalogValue           := 2,
  BACnetObjectSupportedBits_BinaryInput           := 3,
  BACnetObjectSupportedBits_BinaryOutput          := 4,
  BACnetObjectSupportedBits_BinaryValue           := 5,
  BACnetObjectSupportedBits_Calendar              := 6,
  BACnetObjectSupportedBits_Command               := 7,
  BACnetObjectSupportedBits_Device                := 8,
  BACnetObjectSupportedBits_EventEnrollment       := 9,
  BACnetObjectSupportedBits_File                  := 10,
  BACnetObjectSupportedBits_Group                 := 11,
  BACnetObjectSupportedBits_Loop                  := 12,
  BACnetObjectSupportedBits_MultiStateInput       := 13,
  BACnetObjectSupportedBits_MultiStateOutput      := 14,
  BACnetObjectSupportedBits_NotificationClass     := 15,
  BACnetObjectSupportedBits_Program               := 16,
  BACnetObjectSupportedBits_Schedule              := 17,
  BACnetObjectSupportedBits_Averaging             := 18,
  BACnetObjectSupportedBits_MultiStateValue       := 19,
  BACnetObjectSupportedBits_TrendLog              := 20,
  BACnetObjectSupportedBits_LifeSafetyPoint       := 21,
  BACnetObjectSupportedBits_LifeSafetyZone        := 22,
  BACnetObjectSupportedBits_Accumulator           := 23,
  BACnetObjectSupportedBits_PulseConverter        := 24,
  BACnetObjectSupportedBits_EventLog              := 25,
  BACnetObjectSupportedBits_GlobalGroup           := 26,
  BACnetObjectSupportedBits_TrendLogMultiple      := 27,
  BACnetObjectSupportedBits_LoadControl           := 28,
  BACnetObjectSupportedBits_StructuredView        := 29,
  BACnetObjectSupportedBits_AccessDoor            := 30,
  BACnetObjectSupportedBits_unassigend31          := 31,
  BACnetObjectSupportedBits_AccessCredential      := 32,
  BACnetObjectSupportedBits_AccessPoint           := 33,
  BACnetObjectSupportedBits_AccessRights          := 34,
  BACnetObjectSupportedBits_AccessUser            := 35,
```

```
  BACnetObjectSupportedBits_AccessZone            := 36,
  BACnetObjectSupportedBits_CredentialDataInput   := 37,
  BACnetObjectSupportedBits_NetworkSecurity       := 38,
  BACnetObjectSupportedBits_BitStringValue        := 39,
  BACnetObjectSupportedBits_CharacterStringValue  := 40,
  BACnetObjectSupportedBits_DatePatternValue      := 41,
  BACnetObjectSupportedBits_DateValue             := 42,
  BACnetObjectSupportedBits_DateTimePatternValue  := 43,
  BACnetObjectSupportedBits_DateTimeValue         := 44,
  BACnetObjectSupportedBits_IntegerValue          := 45,
  BACnetObjectSupportedBits_LargeAnalogValue      := 46,
  BACnetObjectSupportedBits_OctetStringValue      := 47,
  BACnetObjectSupportedBits_PositiveIntegerValue  := 48,
  BACnetObjectSupportedBits_TimePatternValue      := 49,
  BACnetObjectSupportedBits_TimeValue             := 50,
  BACnetObjectSupportedBits_NetworkPort           := 51,
  BACnetObjectSupportedBits_AlertEnrollment       := 52,
  BACnetObjectSupportedBits_Channel               := 53,
)
END_TYPE
```

4.9.20 E_BACNETOBJECTTYPE

List of possible BACnet objects (excerpt).

```
TYPE E_BACNETOBJECTTYPE :
(
  BACnetObjectType_AnalogInput           := 0,
  BACnetObjectType_AnalogOutput          := 1,
  BACnetObjectType_AnalogValue           := 2,
  BACnetObjectType_BinaryInput           := 3,
  BACnetObjectType_BinaryOutput          := 4,
  BACnetObjectType_BinaryValue           := 5,
  BACnetObjectType_Calendar              := 6,
  BACnetObjectType_Command               := 7,
  BACnetObjectType_Device                := 8,
  BACnetObjectType_EventEnrollment       := 9,
  BACnetObjectType_File                  := 10,
  BACnetObjectType_Group                 := 11,
  BACnetObjectType_Loop                  := 12,
  BACnetObjectType_MultiStateInput       := 13,
  BACnetObjectType_MultiStateOutput      := 14,
  BACnetObjectType_NotificationClass     := 15,
  BACnetObjectType_Program               := 16,
  BACnetObjectType_Schedule              := 17,
  BACnetObjectType_Averaging             := 18,
  BACnetObjectType_MultiStateValue       := 19,
  BACnetObjectType_TrendLog              := 20,
  BACnetObjectType_LifeSafetyPoint       := 21,
  BACnetObjectType_LifeSafetyZone        := 22,
  BACnetObjectType_Accumulator           := 23,
  BACnetObjectType_PulseConverter        := 24,
  BACnetObjectType_EventLog              := 25,
  BACnetObjectType_GlobalGroup           := 26,
  BACnetObjectType_TrendLogMultiple      := 27,
  BACnetObjectType_LoadControl           := 28,
  BACnetObjectType_StructuredView        := 29,
  BACnetObjectType_AccessDoor            := 30,
  BACnetObjectType_unassigend31          := 31,
  BACnetObjectType_AccessCredential      := 32,
  BACnetObjectType_AccessPoint           := 33,
  BACnetObjectType_AccessRights          := 34,
  BACnetObjectType_AccessUser            := 35,
  BACnetObjectType_AccessZone            := 36,
  BACnetObjectType_CredentialDataInput   := 37,
  BACnetObjectType_NetworkSecurity       := 38,
  BACnetObjectType_BitStringValue        := 39,
  BACnetObjectType_CharacterStringValue  := 40,
  BACnetObjectType_DatePatternValue      := 41,
  BACnetObjectType_DateValue             := 42,
  BACnetObjectType_DateTimePatternValue  := 43,
  BACnetObjectType_DateTimeValue         := 44,
  BACnetObjectType_IntegerValue          := 45,
  BACnetObjectType_LargeAnalogValue      := 46,
  BACnetObjectType_OctetStringValue      := 47,
  BACnetObjectType_PositiveIntegerValue  := 48,
  BACnetObjectType_TimePatternValue      := 49,
  BACnetObjectType_TimeValue             := 50,
```

```
  BACnetObjectType_NetworkPort           := 51,
  BACnetObjectType_AlertEnrollment       := 52,
  BACnetObjectType_Channel               := 53
  BACnetObjectType_ZCount,
)
END_TYPE
```

4.9.21 E_BACNETPERSISTENTDATASTATE

PLC mapping of proprietary BACnet data type PersistentDataState (see BACnet device object [▶ 196]).

```
TYPE E_BACNETPERSISTENTDATASTATE :
(
  BACnetPersistentData_Disabled,
  BACnetPersistentData_Initialize,
  BACnetPersistentData_NoChange,
  BACnetPersistentData_Ready,
  BACnetPersistentData_Write,
  BACnetPersistentData_Writing
)
END_TYPE
```

`BACnetPersistentData_Disabled`: Persistent data are disabled.

`BACnetPersistentData_Initialize`: Persistent data are initialized (loaded).

`BACnetPersistentData_NoChange`: No change to online data detected.

`BACnetPersistentData_Ready`: Persistent data can be written (changes to online data detected).

`BACnetPersistentData_Write`: Request to write persistent data: If the property *Persistent_Data* is set to this value, it will set the internal request to write the persistent data in the BACnet driver.

`BACnetPersistentData_Writing`: Persistent data are written.

4.9.22 E_BACNETPIDTUNINGMODE

Tuning modes of block FB_BACnet_LOOP [▶ 201].

```
TYPE E_BACNETPIDTUNINGMODE :
(
  BACnetPIDTuningMode_P,
  BACnetPIDTuningMode_PI,
  BACnetPIDTuningMode_PD,
  BACnetPIDTuningMode_PID
)
END_TYPE
```

`BACnetPIDTuningMode_P`: Optimize controller to P behavior (I and D disabled; typically for lighting control).

`BACnetPIDTuningMode_PI`: Optimize controller to PI behavior (D disabled; typically for temperature control).

`BACnetPIDTuningMode_PD`: Optimize controller to PD behavior (I disabled).

`BACnetPIDTuningMode_PID`: Optimize controller to PID behavior.

4.9.23 E_BACNETPOLARITY

PLC mapping of BACnet data type BACnetPolarity. See BACnet specification DIN EN ISO 16484-5 for property *Polarity*.

```
TYPE E_BACNETPOLARITY :
(
  BACnetPolarity_normal   := 0,
  BACnetPolarity_reverse  := 1
)
END_TYPE
```

`BACnetPolarity_normal`: Normal: e.g. BACnet binary input physically active (24V level/TRUE/ACTIVE) → property Present_Value = ACTIVE (if property Reliability = NO_FAULT_DETECTED and Out_Of_Service = FALSE).

`BACnetPolarity_reverse`: Reverse: e.g. BACnet binary input physically active (24V level/TRUE/ACTIVE) → property Present_Value = INACTIVE (if property Reliability = NO_FAULT_DETECTED and Out_Of_Service = FALSE).

Attention: The property *Polarity* does not affect the evaluation of the property *Feedback_Value* (e.g. BACnet binary output).

4.9.24 E_BACNETPRIORITY

List of possible BACnet priorities of a commandable property (e.g. *Present_Value*).

```
TYPE E_BACNETPRIORITY :
(
  BACnetPriority_None  := 0,
  BACnetPriority_1     := 1,
  BACnetPriority_2     := 2,
  BACnetPriority_3     := 3,
  BACnetPriority_4     := 4,
  BACnetPriority_5     := 5,
  BACnetPriority_6     := 6,
  BACnetPriority_7     := 7,
  BACnetPriority_8     := 8,
  BACnetPriority_9     := 9,
  BACnetPriority_10    := 10,
  BACnetPriority_11    := 11,
  BACnetPriority_12    := 12,
  BACnetPriority_13    := 13,
  BACnetPriority_14    := 14,
  BACnetPriority_15    := 15,
  BACnetPriority_16    := 16
)
END_TYPE
```

BACnetPriority_None: No priority.

`BACnetPriority_1`: Priority 1: Manual-Life Safety

`BACnetPriority_2`: Priority 2: Automatic-Life Safety

`BACnetPriority_3`: Priority 3: Free

`BACnetPriority_4`: Priority 4: Free

`BACnetPriority_5`: Priority 5: Critical Equipment Control

`BACnetPriority_6`: Priority 6: Minimum On/Off

`BACnetPriority_7`: Priority 7: Free

`BACnetPriority_8`: Priority 8: Manual Operator (e.g. on-site operation)

`BACnetPriority_9`: Priority 9: Free

`BACnetPriority_10`: Priority 10: Free

`BACnetPriority_11`: Priority 11: Free

`BACnetPriority_12`: Priority 12: PLC blocks (standard for automapping of library FBs)

`BACnetPriority_13`: Priority 13: Free

`BACnetPriority_14`: Priority 14: Free

`BACnetPriority_15`: Priority 15: Free

`BACnetPriority_16`: Priority 16: Free

4.9.25 E_BACNETPROGRAMERROR

PLC mapping of BACnet data type BACnetProgramError. See BACnet specification DIN EN ISO 16484-5 for property *Reason_For_Halt* and function block description FB_BACnet_Program [▶ 219].

```
TYPE E_BACNETPROGRAMERROR :
(
  BACnetProgramError_program_normal      := 0,
  BACnetProgramError_load_failed         := 1,
  BACnetProgramError_progerror_internal  := 2,
  BACnetProgramError_program             := 3,
  BACnetProgramError_other               := 4
)
END_TYPE
```

`BACnetProgramError_program_normal`: PROGRAM_NORMAL

`BACnetProgramError_load_failed`: LOAD_FAILED

`BACnetProgramError_progerror_internal`: PROGERROR_INTERNAL

`BACnetProgramError_program`: PROGRAM

`BACnetProgramError_other`: OTHER

4.9.26 E_BACNETPROGRAMREQUEST

PLC mapping of BACnet data type BACnetProgramRequest. See BACnet specification DIN EN ISO 16484-5 for property *Program_Change* and function block description FB_BACnet_Program [▶ 219].

```
TYPE E_BACNETPROGRAMREQUEST :
(
  BACnetProgramRequest_ready    := 0,
  BACnetProgramRequest_load     := 1,
  BACnetProgramRequest_run      := 2,
  BACnetProgramRequest_halt     := 3,
  BACnetProgramRequest_restart  := 4,
  BACnetProgramRequest_unload   := 5
)
END_TYPE
```

`BACnetProgramRequest_ready`: READY

`BACnetProgramRequest_load`: LOAD

`BACnetProgramRequest_run`: RUN

`BACnetProgramRequest_halt`: HALT

`BACnetProgramRequest_restart`: RESTART

`BACnetProgramRequest_unload`: UNLOAD

4.9.27 E_BACNETPROGRAMSTATE

PLC mapping of BACnet data type BACnetProgramState. See BACnet specification DIN EN ISO 16484-5 for property *Program_State* and function block description FB_BACnet_Program [▶ 219].

```
TYPE E_BACNETPROGRAMSTATE :
(
  BACnetProgramState_idle       := 0,
  BACnetProgramState_loading    := 1,
  BACnetProgramState_running    := 2,
  BACnetProgramState_waiting    := 3,
  BACnetProgramState_halted     := 4,
  BACnetProgramState_unloading  := 5
)
END_TYPE
```

`BACnetProgramState_idle`: IDLE

`BACnetProgramState_loading`: LOADING

`BACnetProgramState_running`: RUNNING

`BACnetProgramState_waiting`: WAITING

`BACnetProgramState_halted`: HALTED

`BACnetProgramState_unloading`: UNLOADING

4.9.28 E_BACNETPROPERTYIDENTIFIER

List of possible BACnet properties (excerpt).

```
TYPE E_BACNETPROPERTYIDENTIFIER :
(
  BACnetPropertyIdentifier_AckedTransitions                 := 0,
  BACnetPropertyIdentifier_AckRequired                      := 1,
  BACnetPropertyIdentifier_Action                           := 2,
  BACnetPropertyIdentifier_ActionText                       := 3,
  BACnetPropertyIdentifier_ActiveText                       := 4,
  BACnetPropertyIdentifier_ActiveVTSessions                 := 5,
  BACnetPropertyIdentifier_AlarmValue                       := 6,
  BACnetPropertyIdentifier_AlarmValues                      := 7,
  BACnetPropertyIdentifier_All                              := 8,
  BACnetPropertyIdentifier_AllWritesSuccessful              := 9,
  BACnetPropertyIdentifier_ApduSegmentTimeout               := 10,
  BACnetPropertyIdentifier_ApduTimeout                      := 11,
  BACnetPropertyIdentifier_ApplicationSoftwareVersion       := 12,
  BACnetPropertyIdentifier_Archive                          := 13,
  BACnetPropertyIdentifier_Bias                             := 14,
  BACnetPropertyIdentifier_ChangeOfStateCount               := 15,
  BACnetPropertyIdentifier_ChangeOfStateTime                := 16,
  BACnetPropertyIdentifier_NotificationClass                := 17,
  BACnetPropertyIdentifier_Reserved18                       := 18,
  BACnetPropertyIdentifier_ControlledVariableReference      := 19,
  BACnetPropertyIdentifier_ControlledVariableUnits          := 20,
  BACnetPropertyIdentifier_ControlledVariableValue          := 21,
  BACnetPropertyIdentifier_CovIncrement                     := 22,
  BACnetPropertyIdentifier_DateList                         := 23,
  BACnetPropertyIdentifier_DaylightSavingsStatus            := 24,
  BACnetPropertyIdentifier_Deadband                         := 25,
  BACnetPropertyIdentifier_DerivativeConstant               := 26,
  BACnetPropertyIdentifier_DerivativeConstantUnits          := 27,
  BACnetPropertyIdentifier_Description                      := 28,
  BACnetPropertyIdentifier_DescriptionOfHalt                := 29,
  BACnetPropertyIdentifier_DeviceAddressBinding             := 30,
  BACnetPropertyIdentifier_DeviceType                       := 31,
  BACnetPropertyIdentifier_EffectivePeriod                  := 32,
  BACnetPropertyIdentifier_ElapsedActiveTime                := 33,
  BACnetPropertyIdentifier_ErrorLimit                       := 34,
  BACnetPropertyIdentifier_EventEnable                      := 35,
  BACnetPropertyIdentifier_EventState                       := 36,
  BACnetPropertyIdentifier_EventType                        := 37,
  BACnetPropertyIdentifier_ExceptionSchedule                := 38,
  BACnetPropertyIdentifier_FaultValues                      := 39,
  BACnetPropertyIdentifier_FeedbackValue                    := 40,
  BACnetPropertyIdentifier_FileAccessMethod                 := 41,
  BACnetPropertyIdentifier_FileSize                         := 42,
  BACnetPropertyIdentifier_FileType                         := 43,
  BACnetPropertyIdentifier_FirmwareRevision                 := 44,
  BACnetPropertyIdentifier_HighLimit                        := 45,
  BACnetPropertyIdentifier_InactiveText                     := 46,
  BACnetPropertyIdentifier_InProcess                        := 47,
  BACnetPropertyIdentifier_InstanceOf                       := 48,
  BACnetPropertyIdentifier_IntegralConstant                 := 49,
  BACnetPropertyIdentifier_IntegralConstantUnits            := 50,
  BACnetPropertyIdentifier_IssueConfirmedNotifications      := 51,
  BACnetPropertyIdentifier_LimitEnable                      := 52,
  BACnetPropertyIdentifier_ListOfGroupMembers               := 53,
  BACnetPropertyIdentifier_ListOfObjectPropertyReferences   := 54,
  BACnetPropertyIdentifier_ListOfSessionKeys                := 55,
  BACnetPropertyIdentifier_LocalDate                        := 56,
  BACnetPropertyIdentifier_LocalTime                        := 57,
```

```
BACnetPropertyIdentifier_Location                       := 58,
BACnetPropertyIdentifier_LowLimit                       := 59,
BACnetPropertyIdentifier_ManipulatedVariableReference   := 60,
BACnetPropertyIdentifier_MaximumOutput                  := 61,
BACnetPropertyIdentifier_MaxApduLengthAccepted          := 62,
BACnetPropertyIdentifier_MaxInfoFrames                  := 63,
BACnetPropertyIdentifier_MaxMaster                      := 64,
BACnetPropertyIdentifier_MaxPresValue                   := 65,
BACnetPropertyIdentifier_MinimumOffTime                 := 66,
BACnetPropertyIdentifier_MinimumOnTime                  := 67,
BACnetPropertyIdentifier_MinimumOutput                  := 68,
BACnetPropertyIdentifier_MinPresValue                   := 69,
BACnetPropertyIdentifier_ModelName                      := 70,
BACnetPropertyIdentifier_ModificationDate               := 71,
BACnetPropertyIdentifier_NotifyType                     := 72,
BACnetPropertyIdentifier_NumberOfAPDURetries            := 73,
BACnetPropertyIdentifier_NumberOfStates                 := 74,
BACnetPropertyIdentifier_ObjectIdentifier               := 75,
BACnetPropertyIdentifier_ObjectList                     := 76,
BACnetPropertyIdentifier_ObjectName                     := 77,
BACnetPropertyIdentifier_ObjectPropertyReference        := 78,
BACnetPropertyIdentifier_ObjectType                     := 79,
BACnetPropertyIdentifier_Optional                       := 80,
BACnetPropertyIdentifier_OutOfService                   := 81,
BACnetPropertyIdentifier_OutputUnits                    := 82,
BACnetPropertyIdentifier_EventParameters                := 83,
BACnetPropertyIdentifier_Polarity                       := 84,
BACnetPropertyIdentifier_PresentValue                   := 85,
BACnetPropertyIdentifier_Priority                       := 86,
BACnetPropertyIdentifier_PriorityArray                  := 87,
BACnetPropertyIdentifier_PriorityForWriting             := 88,
BACnetPropertyIdentifier_ProcessIdentifier              := 89,
BACnetPropertyIdentifier_ProgramChange                  := 90,
BACnetPropertyIdentifier_ProgramLocation                := 91,
BACnetPropertyIdentifier_ProgramState                   := 92,
BACnetPropertyIdentifier_ProportionalConstant           := 93,
BACnetPropertyIdentifier_ProportionalConstantUnits      := 94,
BACnetPropertyIdentifier_ProtocolConformanceClass       := 95,
BACnetPropertyIdentifier_ProtocolObjectTypesSupported   := 96,
BACnetPropertyIdentifier_ProtocolServicesSupported      := 97,
BACnetPropertyIdentifier_ProtocolVersion                := 98,
BACnetPropertyIdentifier_ReadOnly                       := 99,
BACnetPropertyIdentifier_ReasonForHalt                  := 100,
BACnetPropertyIdentifier_Recipient                      := 101,
BACnetPropertyIdentifier_RecipientList                  := 102,
BACnetPropertyIdentifier_Reliability                    := 103,
BACnetPropertyIdentifier_RelinquishDefault              := 104,
BACnetPropertyIdentifier_Required                       := 105,
BACnetPropertyIdentifier_Resolution                     := 106,
BACnetPropertyIdentifier_SegmentationSupported          := 107,
BACnetPropertyIdentifier_Setpoint                       := 108,
BACnetPropertyIdentifier_SetpointReference              := 109,
BACnetPropertyIdentifier_StateText                      := 110,
BACnetPropertyIdentifier_StatusFlags                    := 111,
BACnetPropertyIdentifier_SystemStatus                   := 112,
BACnetPropertyIdentifier_TimeDelay                      := 113,
BACnetPropertyIdentifier_TimeOfActiveTimeReset          := 114,
BACnetPropertyIdentifier_TimeOfStateCountReset          := 115,
BACnetPropertyIdentifier_TimeSynchronizationRecipients  := 116,
BACnetPropertyIdentifier_Units                          := 117,
BACnetPropertyIdentifier_UpdateInterval                 := 118,
BACnetPropertyIdentifier_UtcOffset                      := 119,
BACnetPropertyIdentifier_VendorIdentifier               := 120,
BACnetPropertyIdentifier_VendorName                     := 121,
BACnetPropertyIdentifier_VtClassesSupported             := 122,
BACnetPropertyIdentifier_WeeklySchedule                 := 123,
BACnetPropertyIdentifier_AttemptedSamples               := 124,
BACnetPropertyIdentifier_AverageValue                   := 125,
BACnetPropertyIdentifier_BufferSize                     := 126,
BACnetPropertyIdentifier_ClientCovIncrement             := 127,
BACnetPropertyIdentifier_CovResubscriptionInterval      := 128,
BACnetPropertyIdentifier_CurrentNotifyTime              := 129,
BACnetPropertyIdentifier_EventTimeStamps                := 130,
BACnetPropertyIdentifier_LogBuffer                      := 131,
BACnetPropertyIdentifier_LogDeviceObjectProperty        := 132,
BACnetPropertyIdentifier_Enable                         := 133,
BACnetPropertyIdentifier_LogInterval                    := 134,
BACnetPropertyIdentifier_MaximumValue                   := 135,
BACnetPropertyIdentifier_MinimumValue                   := 136,
BACnetPropertyIdentifier_NotificationThreshold          := 137,
```

```
BACnetPropertyIdentifier_PreviousNotifyTime                := 138,
BACnetPropertyIdentifier_ProtocolRevision                  := 139,
BACnetPropertyIdentifier_RecordsSinceNotification          := 140,
BACnetPropertyIdentifier_RecordCount                       := 141,
BACnetPropertyIdentifier_StartTime                         := 142,
BACnetPropertyIdentifier_StopTime                          := 143,
BACnetPropertyIdentifier_StopWhenFull                      := 144,
BACnetPropertyIdentifier_TotalRecordCount                  := 145,
BACnetPropertyIdentifier_ValidSamples                      := 146,
BACnetPropertyIdentifier_WindowInterval                    := 147,
BACnetPropertyIdentifier_WindowSamples                     := 148,
BACnetPropertyIdentifier_MaximumValueTimestamp             := 149,
BACnetPropertyIdentifier_MinimumValueTimestamp             := 150,
BACnetPropertyIdentifier_VarianceValue                     := 151,
BACnetPropertyIdentifier_ActiveCovSubscriptions            := 152,
BACnetPropertyIdentifier_BackupFailureTimeout              := 153,
BACnetPropertyIdentifier_ConfigurationFiles                := 154,
BACnetPropertyIdentifier_DatabaseRevision                  := 155,
BACnetPropertyIdentifier_DirectReading                     := 156,
BACnetPropertyIdentifier_LastRestoreTime                   := 157,
BACnetPropertyIdentifier_MaintenanceRequired               := 158,
BACnetPropertyIdentifier_MemberOf                          := 159,
BACnetPropertyIdentifier_Mode                              := 160,
BACnetPropertyIdentifier_OperationExpected                 := 161,
BACnetPropertyIdentifier_Setting                           := 162,
BACnetPropertyIdentifier_Silenced                          := 163,
BACnetPropertyIdentifier_TrackingValue                     := 164,
BACnetPropertyIdentifier_ZoneMembers                       := 165,
BACnetPropertyIdentifier_LifeSafetyAlarmValues             := 166,
BACnetPropertyIdentifier_MaxSegmentsAccepted               := 167,
BACnetPropertyIdentifier_ProfileName                       := 168,
BACnetPropertyIdentifier_AutoSlaveDiscovery                := 169,
BACnetPropertyIdentifier_ManualSlaveAddressBinding         := 170,
BACnetPropertyIdentifier_SlaveAddressBinding               := 171,
BACnetPropertyIdentifier_SlaveProxyEnable                  := 172,
BACnetPropertyIdentifier_LastNotifyRecord                  := 173,
BACnetPropertyIdentifier_ScheduleDefault                   := 174,
BACnetPropertyIdentifier_AcceptedModes                     := 175,
BACnetPropertyIdentifier_AdjustValue                       := 176,
BACnetPropertyIdentifier_Count                             := 177,
BACnetPropertyIdentifier_CountBeforeChange                 := 178,
BACnetPropertyIdentifier_CountChangeTime                   := 179,
BACnetPropertyIdentifier_CovPeriod                         := 180,
BACnetPropertyIdentifier_InputReference                    := 181,
BACnetPropertyIdentifier_LimitMonitoringInverval           := 182,
BACnetPropertyIdentifier_LoggingObject                     := 183,
BACnetPropertyIdentifier_LoggingRecord                     := 184,
BACnetPropertyIdentifier_Prescale                          := 185,
BACnetPropertyIdentifier_PulseRate                         := 186,
BACnetPropertyIdentifier_Scale                             := 187,
BACnetPropertyIdentifier_ScaleFactor                       := 188,
BACnetPropertyIdentifier_UpdateTime                        := 189,
BACnetPropertyIdentifier_ValueBeforeChange                 := 190,
BACnetPropertyIdentifier_ValueSet                          := 191,
BACnetPropertyIdentifier_ValueChangeTime                   := 192,
BACnetPropertyIdentifier_AlignIntervals                    := 193,
BACnetPropertyIdentifier_GroupMemberNames                  := 194,
BACnetPropertyIdentifier_IntervalOffset                    := 195,
BACnetPropertyIdentifier_LastRestartReason                 := 196,
BACnetPropertyIdentifier_LoggingType                       := 197,
BACnetPropertyIdentifier_MemberStatusFlags                 := 198,
BACnetPropertyIdentifier_NotificationPeriod                := 199,
BACnetPropertyIdentifier_PreviousNotifyRecord              := 200,
BACnetPropertyIdentifier_RequestedUpdateInterval           := 201,
BACnetPropertyIdentifier_RestartNotificationRecipients     := 202,
BACnetPropertyIdentifier_TimeOfDeviceRestart               := 203,
BACnetPropertyIdentifier_TimeSynchronizationInterval       := 204,
BACnetPropertyIdentifier_Trigger                           := 205,
BACnetPropertyIdentifier_UTCTimeSynchronizationRecipients  := 206,
BACnetPropertyIdentifier_NodeSubtype                       := 207,
BACnetPropertyIdentifier_NodeType                          := 208,
BACnetPropertyIdentifier_StructuredObjectList              := 209,
BACnetPropertyIdentifier_SubordinateAnnotations            := 210,
BACnetPropertyIdentifier_SubordinateList                   := 211,
BACnetPropertyIdentifier_ActualShedLevel                   := 212,
BACnetPropertyIdentifier_DutyWindow                        := 213,
BACnetPropertyIdentifier_ExpectedShedLevel                 := 214,
BACnetPropertyIdentifier_FullDutyBaseline                  := 215,
BACnetPropertyIdentifier_NodeSubtype216                    := 216,
BACnetPropertyIdentifier_NodeType217                       := 217,
```

```
  BACnetPropertyIdentifier_RequestedShedLevel                := 218,
  BACnetPropertyIdentifier_ShedDuration                      := 219,
  BACnetPropertyIdentifier_ShedLevelDescriptions             := 220,
  BACnetPropertyIdentifier_ShedLevels                        := 221,
  BACnetPropertyIdentifier_StateDescription                  := 222,
  BACnetPropertyIdentifier_DoorAlarmState                    := 226,
  BACnetPropertyIdentifier_DoorExtendedPulseTime             := 227,
  BACnetPropertyIdentifier_DoorMembers                       := 228,
  BACnetPropertyIdentifier_DoorOpenTooLongTime               := 229,
  BACnetPropertyIdentifier_DoorPulseTime                     := 230,
  BACnetPropertyIdentifier_DoorStatus                        := 231,
  BACnetPropertyIdentifier_DoorUnlockDelayTime               := 232,
  BACnetPropertyIdentifier_LockStatus                        := 233,
  BACnetPropertyIdentifier_MaskedAlarmValues                 := 234,
  BACnetPropertyIdentifier_SecuredStatus                     := 235,
  BACnetPropertyIdentifier_BackupAndRestoreState             := 338,
  BACnetPropertyIdentifier_BackupPreparationTime             := 339,
  BACnetPropertyIdentifier_RestoreCompletionTime             := 340,
  BACnetPropertyIdentifier_RestorePreparationTime            := 341,
  BACnetPropertyIdentifier_EventMessageTexts                 := 351,
  BACnetPropertyIdentifier_EventMessageTextsConfig           := 352,
  BACnetPropertyIdentifier_ScaleOffset                       := 512,
  BACnetPropertyIdentifier_IoBusNr                           := 513,
  BACnetPropertyIdentifier_IoModuleNr                        := 514,
  BACnetPropertyIdentifier_IoModuleChn                       := 515,
  BACnetPropertyIdentifier_PersistentData                    := 516,
  BACnetPropertyIdentifier_CurrentlyActivePriority           := 517,
  BACnetPropertyIdentifier_LastConfirmedServiceAccess        := 518,
  BACnetPropertyIdentifier_DataRequestMode                   := 519,
  BACnetPropertyIdentifier_DataRequestCycle                  := 520,
  BACnetPropertyIdentifier_FileName                          := 521
)
END_TYPE
```

4.9.29 E_BACNETRELIABILITY

List of possible values of BACnet property *Reliability* (excerpt).

```
TYPE E_BACNETRELIABILITY :
(
  BACnetReliability_no_fault_detected      := 0,
  BACnetReliability_no_sensor              := 1,
  BACnetReliability_over_range             := 2,
  BACnetReliability_under_range            := 3,
  BACnetReliability_open_loop              := 4,
  BACnetReliability_shorted_loop           := 5,
  BACnetReliability_no_output              := 6,
  BACnetReliability_unreliable_other       := 7,
  BACnetReliability_process_error          := 8,
  BACnetReliability_multi_state_fault      := 9,
  BACnetReliability_configuration_error    := 10,
  BACnetReliability_reserved11             := 11,
  BACnetReliability_communication_failure  := 12,
  BACnetReliability_member_fault           := 13
)
END_TYPE
```

`BACnetReliability_no_fault_detected`: NO_FAULT_DETECTED

`BACnetReliability_no_sensor`: NO_SENSOR

`BACnetReliability_over_range`: OVER_RANGE

`BACnetReliability_under_range`: UNDER_RANGE

`BACnetReliability_open_loop`: OPEN_LOOP

`BACnetReliability_shorted_loop`: SHORTED_LOOP

`BACnetReliability_no_output`: NO_OUTPUT

`BACnetReliability_unreliable_other`: UNRELIABLE_OTHER

`BACnetReliability_process_error`: PROCESS_ERROR

`BACnetReliability_multi_state_fault`: MULTI_STATE_FAULT

`BACnetReliability_configuration_error`: CONFIGURATION_ERROR

`BACnetReliability_reserved11`: *Reserved for future use.*

`BACnetReliability_communication_failure`: COMMUNICATION_FAILURE

`BACnetReliability_member_fault`: MEMBER_FAULT

4.9.30 E_BACNETSEGMENTATION

PLC mapping of BACnet data type BACnetSegmentation. See BACnet specification DIN EN ISO 16484-5 for property *Segmentation_Supported*.

```
TYPE E_BACNETSEGMENTATION :
(
  BACnetSegmentation_both      := 0,
  BACnetSegmentation_transmit  := 1,
  BACnetSegmentation_receive   := 2,
  BACnetSegmentation_no        := 3
)
END_TYPE
```

`BACnetSegmentation_both`: BOTH

`BACnetSegmentation_transmit`: TRANSMIT

`BACnetSegmentation_receive`: RECEIVE

`BACnetSegmentation_no`: NO

4.9.31 E_BACNETSERVICESSUPPORTEDBITS

Bit assignment of the Protocol Services Supported flags [▶ 366] (property *Protocol_Services_Supported*).

```
TYPE E_BACNETSERVICESSUPPORTEDBITS :
(
  BACnetServicesSupportedBits_AcknowledgeAlarm              := 0,
  BACnetServicesSupportedBits_ConfirmedCOVNotification      := 1,
  BACnetServicesSupportedBits_ConfirmedEventNotification    := 2,
  BACnetServicesSupportedBits_GetAlarmSummary               := 3,
  BACnetServicesSupportedBits_GetEnrollmentSummary          := 4,
  BACnetServicesSupportedBits_SubscribeCOV                  := 5,
  BACnetServicesSupportedBits_AtomicReadFile                := 6,
  BACnetServicesSupportedBits_AtomicWriteFile               := 7,
  BACnetServicesSupportedBits_AddListElement                := 8,
  BACnetServicesSupportedBits_RemoveListElement             := 9,
  BACnetServicesSupportedBits_CreateObject                  := 10,
  BACnetServicesSupportedBits_DeleteObject                  := 11,
  BACnetServicesSupportedBits_ReadProperty                  := 12,
  BACnetServicesSupportedBits_ReadPropertyConditional       := 13,
  BACnetServicesSupportedBits_ReadPropertyMultiple          := 14,
  BACnetServicesSupportedBits_WriteProperty                 := 15,
  BACnetServicesSupportedBits_WritePropertyMultiple         := 16,
  BACnetServicesSupportedBits_DeviceCommunicationControl    := 17,
  BACnetServicesSupportedBits_ConfirmedPrivateTransfer      := 18,
  BACnetServicesSupportedBits_ConfirmedTextMessage          := 19,
  BACnetServicesSupportedBits_ReinitializeDevice            := 20,
  BACnetServicesSupportedBits_VtOpen                        := 21,
  BACnetServicesSupportedBits_VtClose                       := 22,
  BACnetServicesSupportedBits_VtData                        := 23,
  BACnetServicesSupportedBits_Authenticate                  := 24,
  BACnetServicesSupportedBits_RequestKey                    := 25,
  BACnetServicesSupportedBits_IAm                           := 26,
  BACnetServicesSupportedBits_IHave                         := 27,
  BACnetServicesSupportedBits_UnconfirmedCOVNotification    := 28,
  BACnetServicesSupportedBits_UnconfirmedEventNotification  := 29,
  BACnetServicesSupportedBits_UnconfirmedPrivateTransfer    := 30,
  BACnetServicesSupportedBits_UnconfirmedTextMessage        := 31,
  BACnetServicesSupportedBits_TimeSynchronization           := 32,
  BACnetServicesSupportedBits_WhoHas                        := 33,
  BACnetServicesSupportedBits_WhoIs                         := 34,
```

```
  BACnetServicesSupportedBits_ReadRange                    := 35,
  BACnetServicesSupportedBits_UtcTimeSynchronization       := 36,
  BACnetServicesSupportedBits_LifeSafetyOperation          := 37,
  BACnetServicesSupportedBits_SubscribeCOVProperty         := 38,
  BACnetServicesSupportedBits_GetEventInformation          := 39
)
END_TYPE
```

4.9.32 E_BACNETSILENCEDSTATE

PLC mapping of BACnet data type BACnetSilencedState. See BACnet specification DIN EN ISO 16484-5 for property *Silenced*.

```
TYPE E_BACNETSILENCEDSTATE :
(
  BACnetSilencedState_unsilenced        := 0,
  BACnetSilencedState_audible_silenced  := 1,
  BACnetSilencedState_visible_silenced  := 2,
  BACnetSilencedState_all_silenced      := 3
)
END_TYPE
```

`BACnetSilencedState_unsilenced`: UNSILENCED

`BACnetSilencedState_audible_silenced`: AUDIBLE_SILENCED

`BACnetSilencedState_visible_silenced`: VISIBLE_SILENCED

`BACnetSilencedState_all_silenced`: ALL_SILENCED

4.9.33 E_BACNETSTATUSFLAGS

Bit assignment of the Status flags [▶ 367] (property *Status_Flags*).

```
TYPE E_BACNETSTATUSFLAGS :
(
  BACnetStatusFlags_in_alarm        := 0,
  BACnetStatusFlags_fault           := 1,
  BACnetStatusFlags_overridden      := 2,
  BACnetStatusFlags_out_of_service  := 3
)
END_TYPE
```

`BACnetStatusFlags_in_alarm`: IN_ALARM

`BACnetStatusFlags_fault`: FAULT

`BACnetStatusFlags_overridden`: OVERRIDDEN

`BACnetStatusFlags_out_of_service`: OUT_OF_SERVICE

4.9.34 E_BACNETSTRINGENCODINGTYPES

The enumeration contains a list of the string encodings supported by the BACnet driver. Under TwinCAT BACnet/IP the default character set is ANSI/UTF-8. Mixing of character sets may occur, particularly with external systems (clients). Therefore, the following character sets are decoded in the PLC with the block FB_BACnet_StringExtDecode [▶ 321]: ANSI/UTF-8, UCS-2, UCS-4 and ISO8859-1.

The decoding should be generally applied to each read string. If the PLC detects an unsupported character set, the decoding block returns an error code. The decoded string is stored in the PLC in Windows-1252 encoding (also CP 1252) (1-byte character according to ISO 8859-1 and supplements).

```
TYPE E_BACNETSTRINGENCODINGTYPES :
(
  BACnetStringEncodingTypes_AnsiUtf8  := 0,
  BACnetStringEncodingTypes_DBCS      := 1,
  BACnetStringEncodingTypes_JIS       := 2,
  BACnetStringEncodingTypes_UCS4      := 3,
  BACnetStringEncodingTypes_UCS2      := 4,
```

```
  BACnetStringEncodingTypes_ISO8859   := 5,
)
END_TYPE
```

`BACnetStringEncodingTypes_AnsiUtf8`: ANSI X3.4 and UTF-8 encoding according to ISO 10646 (using the PLC block FB_BACnet_StringExtDecode [▶ 321] decodable to Windows-1252)

`BACnetStringEncodingTypes_DBCS`: IBM™/Microsoft™ DBCS: double-byte code with specific (possibly proprietary) codepage

`BACnetStringEncodingTypes_JIS`: JIS C 6226 (Japanese characters)

`BACnetStringEncodingTypes_UCS4`: UCS-4 according to ISO 10646: 4-byte Unicode (using the PLC block FB_BACnet_StringExtDecode [▶ 321], decodable to Windows-1252)

`BACnetStringEncodingTypes_UCS2`: UCS-2 according to ISO 10646: 2-byte Unicode (using the PLC block FB_BACnet_StringExtDecode [▶ 321], decodable to Windows-1252)

`BACnetStringEncodingTypes_ISO8859`: Latin-1 according to ISO 8859-1 (0x00 to 0x7F corresponding to US-ASCII; 0xA0 to 0xFF ISO 8859-1-specific). The printable characters are compatible with Windows-1252.

4.9.35 E_BACNETUNIT

PLC mapping of BACnet data type BACnetEngineeringUnits. See BACnet specification DIN EN ISO 16484-5 for property *Units*.

```
TYPE E_BACNETUNIT :
(
  BACnetUnit_Area_square_meters                              := 0,
  BACnetUnit_Area_square_feet                                := 1,
  BACnetUnit_Electrical_milliamperes                         := 2,
  BACnetUnit_Electrical_amperes                              := 3,
  BACnetUnit_Electrical_ohms                                 := 4,
  BACnetUnit_Electrical_volts                                := 5,
  BACnetUnit_Electrical_kilovolts                            := 6,
  BACnetUnit_Electrical_megavolts                            := 7,
  BACnetUnit_Electrical_volt_amperes                         := 8,
  BACnetUnit_Electrical_kilovolt_amperes                     := 9,
  BACnetUnit_Electrical_megavolt_amperes                     := 10,
  BACnetUnit_Electrical_volt_amperes_reactive                := 11,
  BACnetUnit_Electrical_kilovolt_amperes_reactive            := 12,
  BACnetUnit_Electrical_megavolt_amperes_reactive            := 13,
  BACnetUnit_Electrical_degrees_phase                        := 14,
  BACnetUnit_Electrical_power_factor                         := 15,
  BACnetUnit_Energy_joules                                   := 16,
  BACnetUnit_Energy_kilojoules                               := 17,
  BACnetUnit_Energy_watt_hours                               := 18,
  BACnetUnit_Energy_kilowatt_hours                           := 19,
  BACnetUnit_Energy_btus                                     := 20,
  BACnetUnit_Energy_therms                                   := 21,
  BACnetUnit_Energy_ton_hours                                := 22,
  BACnetUnit_Enthalpy_joules_per_kilogram_dry_air            := 23,
  BACnetUnit_Enthalpy_btus_per_pound_dry_air                 := 24,
  BACnetUnit_Frequency_cycles_per_hour                       := 25,
  BACnetUnit_Frequency_cycles_per_minute                     := 26,
  BACnetUnit_Frequency_hertz                                 := 27,
  BACnetUnit_Humidity_grams_of_water_per_kilogram_dry_air    := 28,
  BACnetUnit_Humidity_percent_relative_humidity              := 29,
  BACnetUnit_Length_millimeters                              := 30,
  BACnetUnit_Length_meters                                   := 31,
  BACnetUnit_Length_inches                                   := 32,
  BACnetUnit_Length_feet                                     := 33,
  BACnetUnit_Light_watts_per_square_foot                     := 34,
  BACnetUnit_Light_watts_per_square_meter                    := 35,
  BACnetUnit_Light_lumens                                    := 36,
  BACnetUnit_Light_luxes                                     := 37,
  BACnetUnit_Light_foot_candles                              := 38,
  BACnetUnit_Mass_kilograms                                  := 39,
  BACnetUnit_Mass_pounds_mass                                := 40,
  BACnetUnit_Mass_tons                                       := 41,
  BACnetUnit_MassFlow_kilograms_per_second                   := 42,
  BACnetUnit_MassFlow_kilograms_per_minute                   := 43,
  BACnetUnit_MassFlow_kilograms_per_hour                     := 44,
```

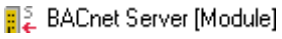
```
BACnetUnit_MassFlow_pounds_mass_per_minute              := 45,
BACnetUnit_MassFlow_pounds_mass_per_hour                := 46,
BACnetUnit_Power_watts                                  := 47,
BACnetUnit_Power_kilowatts                              := 48,
BACnetUnit_Power_megawatts                              := 49,
BACnetUnit_Power_btus_per_hour                          := 50,
BACnetUnit_Power_horsepower                             := 51,
BACnetUnit_Power_tons_refrigeration                     := 52,
BACnetUnit_Pressure_pascals                             := 53,
BACnetUnit_Pressure_kilopascals                         := 54,
BACnetUnit_Pressure_bars                                := 55,
BACnetUnit_Pressure_pounds_force_per_square_inch        := 56,
BACnetUnit_Pressure_centimeters_of_water                := 57,
BACnetUnit_Pressure_inches_of_water                     := 58,
BACnetUnit_Pressure_millimeters_of_mercury              := 59,
BACnetUnit_Pressure_centimeters_of_mercury              := 60,
BACnetUnit_Pressure_inches_of_mercury                   := 61,
BACnetUnit_Temperature_degrees_Celsius                  := 62,
BACnetUnit_Temperature_degrees_Kelvin                   := 63,
BACnetUnit_Temperature_degrees_Fahrenheit               := 64,
BACnetUnit_Temperature_degree_days_Celsius              := 65,
BACnetUnit_Temperature_degree_days_Fahrenheit           := 66,
BACnetUnit_Time_years                                   := 67,
BACnetUnit_Time_months                                  := 68,
BACnetUnit_Time_weeks                                   := 69,
BACnetUnit_Time_days                                    := 70,
BACnetUnit_Time_hours                                   := 71,
BACnetUnit_Time_minutes                                 := 72,
BACnetUnit_Time_seconds                                 := 73,
BACnetUnit_Velocity_meters_per_second                   := 74,
BACnetUnit_Velocity_kilometers_per_hour                 := 75,
BACnetUnit_Velocity_feet_per_second                     := 76,
BACnetUnit_Velocity_feet_per_minute                     := 77,
BACnetUnit_Velocity_miles_per_hour                      := 78,
BACnetUnit_Volume_cubic_feet                            := 79,
BACnetUnit_Volume_cubic_meters                          := 80,
BACnetUnit_Volume_imperial_gallons                      := 81,
BACnetUnit_Volume_liters                                := 82,
BACnetUnit_Volume_us_gallons                            := 83,
BACnetUnit_VolumetricFlow_cubic_feet_per_minute         := 84,
BACnetUnit_VolumetricFlow_cubic_meters_per_second       := 85,
BACnetUnit_VolumetricFlow_imperial_gallons_per_minute   := 86,
BACnetUnit_VolumetricFlow_liters_per_second             := 87,
BACnetUnit_VolumetricFlow_liters_per_minute             := 88,
BACnetUnit_VolumetricFlow_us_gallons_per_minute         := 89,
BACnetUnit_Other_degrees_angular                        := 90,
BACnetUnit_Other_degrees_Celsius_per_hour               := 91,
BACnetUnit_Other_degrees_Celsius_per_minute             := 92,
BACnetUnit_Other_degrees_Fahrenheit_per_hour            := 93,
BACnetUnit_Other_degrees_Fahrenheit_per_minute          := 94,
BACnetUnit_Other_no_units                               := 95,
BACnetUnit_Other_parts_per_million                      := 96,
BACnetUnit_Other_parts_per_billion                      := 97,
BACnetUnit_Other_percent                                := 98,
BACnetUnit_Other_percent_per_second                     := 99,
BACnetUnit_Other_per_minute                             := 100,
BACnetUnit_Other_per_second                             := 101,
BACnetUnit_Other_psi_per_degree_Fahrenheit              := 102,
BACnetUnit_Other_radians                                := 103,
BACnetUnit_Other_revolutions_per_minute                 := 104,
BACnetUnit_Currency_currency1                           := 105,
BACnetUnit_Currency_currency2                           := 106,
BACnetUnit_Currency_currency3                           := 107,
BACnetUnit_Currency_currency4                           := 108,
BACnetUnit_Currency_currency5                           := 109,
BACnetUnit_Currency_currency6                           := 110,
BACnetUnit_Currency_currency7                           := 111,
BACnetUnit_Currency_currency8                           := 112,
BACnetUnit_Currency_currency9                           := 113,
BACnetUnit_Currency_currency10                          := 114,
BACnetUnit_Area_square_inches                           := 115,
BACnetUnit_Area_square_centimeters                      := 116,
BACnetUnit_Enthalpy_btus_per_pound                      := 117,
BACnetUnit_Length_centimeters                           := 118,
BACnetUnit_MassFlow_pounds_mass_per_second              := 119,
BACnetUnit_Temperature_delta_degrees_Fahrenheit         := 120,
BACnetUnit_Temperature_delta_degrees_Kelvin             := 121,
BACnetUnit_Electrical_kilohms                           := 122,
BACnetUnit_Electrical_megohms                           := 123,
BACnetUnit_Electrical_millivolts                        := 124,
```

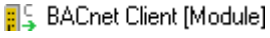
```
BACnetUnit_Energy_kilojoules_per_kilogram               := 125,
BACnetUnit_Energy_megajoules                            := 126,
BACnetUnit_Entropy_joules_per_degree_Kelvin             := 127,
BACnetUnit_Entropy_joules_per_kilogram_degree_Kelvin    := 128,
BACnetUnit_Frequency_kilohertz                          := 129,
BACnetUnit_Frequency_megahertz                          := 130,
BACnetUnit_Frequency_per_hour                           := 131,
BACnetUnit_Power_milliwatts                             := 132,
BACnetUnit_Pressure_hectopascals                        := 133,
BACnetUnit_Pressure_millibars                           := 134,
BACnetUnit_VolumetricFlow_cubic_meters_per_hour         := 135,
BACnetUnit_VolumetricFlow_liters_per_hour               := 136,
BACnetUnit_Other_kilowatt_hours_per_square_meter        := 137,
BACnetUnit_Other_kilowatt_hours_per_square_foot         := 138,
BACnetUnit_Other_megajoules_per_square_meter            := 139,
BACnetUnit_Other_megajoules_per_square_foot             := 140,
BACnetUnit_Other_watts_per_square_meter_degree_kelvin   := 141,
BACnetUnit_VolumetricFlow_cubic_feet_per_second         := 142,
BACnetUnit_Other_percent_obscuration_per_foot           := 143,
BACnetUnit_Other_percent_obscuration_per_meter          := 144,
BACnetUnit_Electrical_milliohms                         := 145,
BACnetUnit_Energy_megawatt_hours                        := 146,
BACnetUnit_Energy_kilo_btus                             := 147,
BACnetUnit_Energy_mega_btus                             := 148,
BACnetUnit_Enthalpy_kilojoules_per_kilogram_dry_air     := 149,
BACnetUnit_Enthalpy_megajoules_per_kilogram_dry_air     := 150,
BACnetUnit_Entropy_kilojoules_per_degree_Kelvin         := 151,
BACnetUnit_Entropy_megajoules_per_degree_Kelvin         := 152,
BACnetUnit_Force_newton                                 := 153,
BACnetUnit_MassFlow_grams_per_second                    := 154,
BACnetUnit_MassFlow_grams_per_minute                    := 155,
BACnetUnit_MassFlow_tons_per_hour                       := 156,
BACnetUnit_Power_kilo_btus_per_hour                     := 157,
BACnetUnit_Time_hundredths_seconds                      := 158,
BACnetUnit_Time_milliseconds                            := 159,
BACnetUnit_Torque_newton_meters                         := 160,
BACnetUnit_Velocity_millimeters_per_second              := 161,
BACnetUnit_Velocity_millimeters_per_minute              := 162,
BACnetUnit_Velocity_meters_per_minute                   := 163,
BACnetUnit_Velocity_meters_per_hour                     := 164,
BACnetUnit_VolumetricFlow_cubic_meters_per_minute       := 165,
BACnetUnit_Acceleration_meters_per_second_per_second    := 166,
BACnetUnit_Electrical_amperes_per_meter                 := 167,
BACnetUnit_Electrical_amperes_per_square_meter          := 168,
BACnetUnit_Electrical_ampere_square_meters              := 169,
BACnetUnit_Electrical_farads                            := 170,
BACnetUnit_Electrical_henrys                            := 171,
BACnetUnit_Electrical_ohm_meters                        := 172,
BACnetUnit_Electrical_siemens                           := 173,
BACnetUnit_Electrical_siemens_per_meter                 := 174,
BACnetUnit_Electrical_teslas                            := 175,
BACnetUnit_Electrical_volts_per_degree_Kelvin           := 176,
BACnetUnit_Electrical_volts_per_meter                   := 177,
BACnetUnit_Electrical_webers                            := 178,
BACnetUnit_Light_candelas                               := 179,
BACnetUnit_Light_candelas_per_square_meter              := 180,
BACnetUnit_Temperature_degrees_Kelvin_per_hour          := 181,
BACnetUnit_Temperature_degrees_Kelvin_per_minute        := 182,
BACnetUnit_Other_joule_seconds                          := 183,
BACnetUnit_Other_radians_per_second                     := 184,
BACnetUnit_Other_square_meters_per_Newton               := 185,
BACnetUnit_Other_kilograms_per_cubic_meter              := 186,
BACnetUnit_Other_newton_seconds                         := 187,
BACnetUnit_Other_newtons_per_meter                      := 188,
BACnetUnit_Other_watts_per_meter_per_degree_Kelvin      := 189,
BACnetUnit_micro_siemens                                := 190,
BACnetUnit_cubic_feet_per_hour                          := 191,
BACnetUnit_us_gallons_per_hour                          := 192,
BACnetUnit_kilometers                                   := 193,
BACnetUnit_micrometers                                  := 194,
BACnetUnit_grams                                        := 195,
BACnetUnit_milligrams                                   := 196,
BACnetUnit_milliliters                                  := 197,
BACnetUnit_milliliters_per_second                       := 198,
BACnetUnit_decibels                                     := 199,
BACnetUnit_decibels_millivolt                           := 200,
BACnetUnit_decibels_volt                                := 201,
BACnetUnit_millisiemens                                 := 202,
BACnetUnit_watt_hours_reactive                          := 203,
BACnetUnit_kilowatt_hours_reactive                      := 204,
```

```
  BACnetUnit_megawatt_hours_reactive                    := 205,
  BACnetUnit_millimeters_of_water                       := 206,
  BACnetUnit_per_mille                                  := 207,
  BACnetUnit_grams_per_gram                             := 208,
  BACnetUnit_kilograms_per_kilogram                     := 209,
  BACnetUnit_grams_per_kilogram                         := 210,
  BACnetUnit_milligrams_per_gram                        := 211,
  BACnetUnit_milligrams_per_kilogram                    := 212,
  BACnetUnit_grams_per_milliliter                       := 213,
  BACnetUnit_grams_per_liter                            := 214,
  BACnetUnit_milligrams_per_liter                       := 215,
  BACnetUnit_micrograms_per_liter                       := 216,
  BACnetUnit_grams_per_cubic_meter                      := 217,
  BACnetUnit_milligrams_per_cubic_meter                 := 218,
  BACnetUnit_micrograms_per_cubic_meter                 := 219,
  BACnetUnit_nanograms_per_cubic_meter                  := 220,
  BACnetUnit_grams_per_cubic_centimeter                 := 221,
  BACnetUnit_becquerels                                 := 222,
  BACnetUnit_kilobecquerels                             := 223,
  BACnetUnit_megabecquerels                             := 224,
  BACnetUnit_gray                                       := 225,
  BACnetUnit_milligray                                  := 226,
  BACnetUnit_microgray                                  := 227,
  BACnetUnit_sieverts                                   := 228,
  BACnetUnit_millisieverts                              := 229,
  BACnetUnit_microsieverts                              := 230,
  BACnetUnit_microsieverts_per_hour                     := 231,
  BACnetUnit_decibels_a                                 := 232,
  BACnetUnit_nephelometric_turbidity_unit               := 233,
  BACnetUnit_pH                                         := 234,
  BACnetUnit_grams_per_square_meter                     := 235,
  BACnetUnit_minutes_per_degree_kelvin                  := 236
)
END_TYPE
```

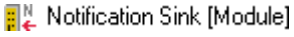
4.9.36 ST_BACnet_AdsConnection

Structure with connection information for an ADS server of the BACnet driver. Three types of ADS server are supported under BACnet:

Server → a BACnet server 

Client → a BACnet client representing a remote BACnet server 

Notification Sink → a BACnet messages "sink" (receiving of and subscribing to BACnet messages)




```
TYPE ST_BACnet_AdsConnection :
STRUCT
  bValid     : BOOL;
  nReload    : USINT;
  sAmsNetId  : T_AmsNetId;
  nAmsPort   : T_AmsPort:=1000;
  bServer    : BOOL;
  bClient    : BOOL;
  bNSink     : BOOL;
  nDeviceId  : UDINT;
END_STRUCT
END_TYPE
```

`bValid`**:** Included connection data are valid

`nReload`**:** Trigger for automatic reloading of data. If **nReload** changes, the connection to the ADS server is re-established and/or connection data are updated. Function blocks for evaluating the *ST_BACnet_AdsConnection* reload the data (e.g. of a property) from the corresponding ADS server, if the input **bAutoReload** at this function block is set to *TRUE*.

`sAmsNetId`**:** AMS NetID of the BACnet adapter.

`nAmsPort`: AMS port of the BACnet server, BACnet client or BACnet Notification Sink (typically ≥ 1000).

`bServer`: ADS server of type BACnet server

`bClient`: ADS server of type BACnet client

`bNSink`: ADS server of type BACnet Notification Sink

`nDeviceId`: BACnet ID of the BACnet server or BACnet instance number of the BACnet Device Object (only for BACnet server or BACnet client)

4.9.37 ST_BACnet_CharacterStringExt

Raw data structure of the property with BACnet data type CharacterString. The function blocks FB_BACnet_StringExtDecode [▶ 321] and FB_BACnet_StringExtEncode [▶ 321] decode and code data of this structure to/from strings that can be displayed and processed in the PLC, based on *Windows-1252* or *CP1252* encoding (Western European Windows character set).

```
TYPE ST_BACnet_CharacterStringExt :
STRUCT
  cookie      : BYTE;
  encoding    : BYTE;
  strLen      : UINT;
  stringData  : ARRAY[1..BACnet_STRING_MAXLENGTH] OF BYTE;
END_STRUCT
END_TYPE
```

`cookie`: 0 and **strLen** > 0 → no ANSI/UTF-8 string encoding; otherwise: From address of cookie ANSI/UTF-8 string is expected.

`encoding`: If no ANSI/UTF-8 string, then is the string encoding is included here (see E_BACNETSTRINGENCODINGTYPES [▶ 351])

`strLen`: If no ANSI/UTF-8 string, then is the byte data length is included in **stringData**

`stringData`: If no ANSI/UTF-8 string: String data

4.9.38 ST_BACnet_CharacterStringExtListEntry

Raw data of a property entry with the BACnet type List of CharacterString.

```
TYPE ST_BACnet_CharacterStringExtListEntry :
STRUCT
  nEntrySize  : DWORD;
  hdrInfo     : DWORD;
  extString   : ST_BACnet_CharacterStringExt;
END_STRUCT
END_TYPE
```

`nEntrySize`: Byte size of the data

`hdrInfo`: Info Flags

4.9.39 ST_BACnet_Date

PLC mapping of BACnet data type Date. See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

```
TYPE ST_BACnet_Date :
STRUCT
  nYear       : BYTE:=16#FF;
  nMonth      : BYTE:=16#FF;
  nDay        : BYTE:=16#FF;
  nDayOfWeek  : BYTE:=16#FF;
END_STRUCT
END_TYPE
```

`nYear`: Year minus 1900; 255 stands for undefined (or each year)

`nMonth`: Possible values for month(s): 1 to 14; 1 → January etc.; 13 → all odd months; 14 → all even months; 255 stands for undefined (or each month)

`nDay`: Possible values for day(s) of the month: 1 to 32; 1 → first day etc.; 32 → last day of the month; 255 stands for undefined (or each day)

`nDayOfWeek`: Possible values for day(s) of the week: 1 to 7; 1 → Monday etc.; 7 → Sunday; 255 stands for undefined (or each day)

4.9.40 ST_BACnet_DateTime

PLC mapping of BACnet data type BACnetDateTime. See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

```
TYPE ST_BACnet_DateTime :
STRUCT
  stDate  : ST_BACnet_Date;
  stTime  : ST_BACnet_Time;
END_STRUCT
END_TYPE
```

Warning:

If BACnetDateTime values are used for switching in schedules, for example, the following exemplary switching times should be avoided:

Mo. 0:00.00 - ACTIVE → Mo. 23:59.59 - INACTIVE

Reason: Switching to state INACTIVE may not be detected. The switching time is immediately prior to the day switching. In this example, if the day changes from Monday to Tuesday before the time change from 23:59.58 to 23:59.59 is detected, the entry for Monday 23:59.59 to INACTIVE is no longer executed. This effect may occur in controllers with high load, or if moderate BACnet cycle times are used (>> 100 ms).

To set switching periods covering a whole day (24 hrs), the following switching times should be used instead:

Mo. 0:00.00 - ACTIVE → **Tu.** 0:00.00 - INACTIVE

or (e.g. 5 minutes to midnight):

Mo. 0:00.00 - ACTIVE → Mo. **23:55.00** - INACTIVE

4.9.41 ST_BACnet_DiagEthStatistics

Structure with information on the Ethernet adapter.

```
TYPE ST_BACnet_DiagEthStatistics :
STRUCT
  ethStat     : ST_BACnet_TcIoEthStatistic;
  txRxErrCnt  : ST_BACnet_TcIoEthTxRxErrorCount;
END_STRUCT
END_TYPE
```

4.9.42 ST_BACnet_Diagnosis

Structure with information from the BACnet device adapter. Includes dynamic response, statistics for server and client and diagnostic information on network traffic.

```
TYPE ST_BACnet_Diagnosis :
STRUCT
  ethDevice        : ST_BACnet_DiagEthStatistics;
  execTiming       : ST_BACnet_DiagnosisTiming;
  frameStatistics  : ST_BACnet_FrameStatistics;
  nRes1            : DWORD;
  info             : ST_BACnet_Info;
  nRes2            : DWORD;
  serverInfo       : ST_BACnet_ServerStatistics;
  lastUpdate       : ST_BACnet_DateTime;
```

```
  nRes3            : DWORD;
END_STRUCT
END_TYPE
```

lastUpdate: Time stamp of the diagnostic data (time at which the data were read).

4.9.43 ST_BACnet_DiagnosisTiming

Structure with information on current execution times of the BACnet driver.

```
TYPE ST_BACnet_DiagnosisTiming :
STRUCT
  msIoInCurExecutionTime   : DWORD;
  msIoInMinExecutionTime   : DWORD;
  msIoInMaxExecutionTime   : DWORD;
  msIoOutCurExecutionTime  : DWORD;
  msIoOutMinExecutionTime  : DWORD;
  msIoOutMaxExecutionTime  : DWORD;
  msCycleCurExecutionTime  : DWORD;
  msCycleMinExecutionTime  : DWORD;
  msCycleMaxExecutionTime  : DWORD;
  msInputCurDistanceTime   : DWORD;
  msInputMinDistanceTime   : DWORD;
  msInputMaxDistanceTime   : DWORD;
END_STRUCT
END_TYPE
```

`msIoInCurExecutionTime`: Current processing time of the input data in the BACnet task (ms).

`msIoInMinExecutionTime`: Minimum processing time of the input data in the BACnet task (ms).

`msIoInMaxExecutionTime`: Maximum processing time of the input data in the BACnet task (ms).

`msIoOutCurExecutionTime`: Current processing time of the output data in the BACnet task (ms).

`msIoOutMinExecutionTime`: Minimum processing time of the output data in the BACnet task (ms).

`msIoOutMaxExecutionTime`: Maximum processing time of the output data in the BACnet task (ms).

`msCycleCurExecutionTime`: Current processing time of the BACnet task (ms).

`msCycleMinExecutionTime`: Minimum processing time of the BACnet task (ms).

`msCycleMaxExecutionTime`: Maximum processing time of the BACnet task (ms).

`msInputCurDistanceTime`: Current time duration between the start of the input data processing (ms).

`msInputMinDistanceTime`: Minimum time duration between the start of the input data processing (ms).

`msInputMaxDistanceTime`: Maximum time duration between the start of the input data processing (ms).

4.9.44 ST_BACnet_EventTransitionBits

PLC mapping of BACnet data type BACnetEventTransitionBits (properties *Event_Enable* and *Acked_Transitions*).

```
TYPE ST_BACnet_EventTransitionBits :
STRUCT
  to_offnormal  : BOOL;
  to_fault      : BOOL;
  to_normal     : BOOL;
END_STRUCT
END_TYPE
```

`to_offnormal`: Transition from X to OFF_NORMAL.

`to_fault`: Transition from X to FAULT.

`to_normal`: Transition from X to NORMAL.

4.9.45 ST_BACnet_ExceptionScheduleBool

Structure for data exchange of the property *exception_schedule* with the aid of function block FB_BACnet_ExceptionScheduleProperty [▶ 283].

```
TYPE ST_BACnet_ExceptionScheduleBool :
STRUCT
  bReqGet     : BOOL;
  bReqSet     : BOOL;
  nGet        : USINT;
  nSet        : USINT;
  bBusy       : BOOL;
  arrEntries  : ARRAY[0..BACnet_MaxExSchEntries] OF ST_BACnet_ExceptionScheduleEntryBool;
END_STRUCT
END_TYPE
```

`bReqGet`: Request read property data *FALSE* → *TRUE* for the corresponding property function block (see FB_BACnet_ExceptionScheduleProperty [▶ 283]). The edge change *TRUE* → *FALSE* indicates acknowledgement of the request by corresponding property function block. Current data will be available shortly. Successful loading of the property data is indicated by `nGet`.

`bReqSet`: Request write property data *FALSE* → *TRUE* for the corresponding property function block (see FB_BACnet_ExceptionScheduleProperty [▶ 283]). The edge change *TRUE* → *FALSE* indicates acknowledgement of the request by corresponding property function block. Current data will be available shortly. Successful writing of the property data is indicated by `nSet`.

`nGet`: Counter is incremented by 1 when property data were read successfully. Ring counter.

`nSet`: Counter is incremented by 1 when property data were written successfully. Ring counter.

`bBusy`: Corresponding property function block loads/writes data from/to the property. The data are invalid while **bBusy** is set to *TRUE*.

`arrEntries`: Array with data of property *Exception_Schedule*. 0 to BACnet_MaxExSchEntries entries are available. The entries are coded as BACnetTimeValue and are limited to data types Boolean and Null. Entries with other data types are ignored. In addition, all types of calendar entries are supported.

4.9.46 ST_BACnet_ExceptionScheduleEntryBool

```
TYPE ST_BACnet_ExceptionScheduleEntryBool :
STRUCT
  eEntryType    : E_BACnetCalendarEntryType;
  ePriority     : E_BACnetPriority;
  stDate        : ST_BACnet_ExceptionScheduleCalDate;
  stWeekNDay    : ST_BACnet_ExceptionScheduleCalWnD;
  stCalRef      : ST_BACnet_ExceptionScheduleCalRef;
  arrTimeValue  : ARRAY[0..BACnet_MaxTimeValues] OF ST_BACnet_TimeValueBool;
END_STRUCT
END_TYPE
```

eEntryType: Type of calendar entry. Possible values:

- `BACnetCalendarEntryType_None` → entry empty.
- `BACnetCalendarEntryType_Ref` → exception: time switching data are read from a calendar object (reference is in **stCalRef**).
- `BACnetCalendarEntryType_Date` → exception: time switching record is read from **stDate.stStart**.
- `BACnetCalendarEntryType_DateRange` → exception: time switching date range is read from **stDate.stStart** and **stDate.stEnd**.
- `BACnetCalendarEntryType_WeekNDay` → exception: time switching day is read from **stWeekNDay**.

`ePriority`: Specification of the priority with which the time switching value is to be written to a commandable property.

`stDate` : Date or date range:

```
TYPE ST_BACnet_ExceptionScheduleCalDate :
STRUCT
  stStart : ST_BACnet_Date;
  stEnd   : ST_BACnet_Date;
END_STRUCT
END_TYPE
```

`stWeekNDay`: Week and day period.

`stCalRef`: Periods are read from a BACnet object of type *Calendar*.

`arrTimeValue`: For each exception time, 0 to BACnet_MaxTimeValues entries are available. The entries are coded as BACnetTimeValue and are limited to data types Boolean and Null. Entries with other data types are ignored.

4.9.47 ST_BACnet_FrameStatistics

Structure with information on BACnet frames.

```
TYPE ST_BACnet_FrameStatistics :
STRUCT
  confirmed              : ARRAY[0..29] OF ST_BACnet_ConfirmedServiceDiag;
  unConfirmed            : ARRAY[0..9] OF ST_BACnet_UnConfirmedServiceDiag;
  nSegmSendTimeouts      : DWORD;
  nFrameAllocFailCnt     : DWORD;
  nFrameSendCnt          : DWORD;
  nFrameSendFailCnt      : DWORD;
  nFrameRecvCnt          : DWORD;
  nFrameRecvFailCnt      : DWORD;
  nLinkStatusChangedCnt  : DWORD;
END_STRUCT
END_TYPE
```

`nSegmSendTimeouts`: Timeout during receipt of acknowledgement of segmented packages (remote station does not acknowledge a segmented response, segmented ACK)

`nFrameAllocFailCnt`: Number of lost frames due to problems with memory allocation (no free memory).

`nFrameSendCnt`: Number of sent frames.

`nFrameSendFailCnt`: Number of failed frames.

`nFrameRecvCnt`: Number of received frames.

`nFrameRecvFailCnt`: Number of faulty received frames.

`nLinkStatusChangedCnt`: Frequency of network status changes (cable pulled/link lost, cable plugged/link OK).

4.9.48 ST_BACnet_GlobalAdsBuffer

Data structure of the global PLC ADS buffer. This buffer is used by the following function blocks for buffering the ADS data stream for reading and writing of complex properties:

- FB_BACnet_EventMessageTextsProperty [▶ 281]
- FB_BACnet_ExceptionScheduleProperty [▶ 283]
- FB_BACnet_LogBufferProperty [▶ 287]
- FB_BACnet_ObjectListProperty [▶ 290]
- FB_BACnet_RecipientListProperty [▶ 294]
- FB_BACnet_WeeklyScheduleProperty [▶ 297]

The buffer size is 8 kbyte. Function blocks that use the buffer prevent simultaneous access using the function *TestAndSet* from the library of the same name. However, this means that multiple read or write access operations with above function blocks are not processed in parallel, but sequentially. The access control is handled internally in the function block.

```
TYPE ST_BACnet_GlobalAdsBuffer :
STRUCT
  bTestSet : BOOL;
  iLocked  : DINT;
  arrData  : ARRAY[0..8191] OF BYTE;
END_STRUCT
END_TYPE
```

`bTestSet`: Lock bit of the function TestAndSet.

`iLocked`: Current blocked data size in the buffer.

`arrData`: Data buffer (8192 byte).

4.9.49 ST_BACnet_Info

Structure with general information on the BACnet task.

```
TYPE ST_BACnet_Info :
STRUCT
  nAmsAllocCnt                : DINT;
  nAmsAllocCntMax             : DWORD;
  nAmsAllocFailCnt            : DWORD;
  nAvailPhysMemBytes          : DWORD;
  nAvailFlashMemBytes         : ARRAY[0..1] OF DWORD;
  nRecvFrameFifoSize          : DWORD;
  nEmptyFrameFifoSize         : DWORD;
  nRecvSegmUDPFramesListSize  : DWORD;
  nRecvSegmFramesListSize     : DWORD;
  nSendSegmFramesListSize     : DWORD;
  nClientScanListSize         : DWORD;
  nAmsRecvMissCount           : DWORD;
END_STRUCT
END_TYPE
```

`nAmsAllocCnt`: Number of reserved router memory blocks (1 block= 1024 bytes). The number should not increase continuously (a maximum of 2000 blocks=2 MB are available by default). Otherwise increase the router memory in the System Manager (see Fig. 1).

`nAmsAllocCntMax`: Peak value of the simultaneously allocated router memory blocks.

`nAmsAllocFailCnt`: Number of failed router memory reservations (allocation fails if no more blocks are available).

`nAvailPhysMemBytes`: Free physical memory of the operating system in bytes.

`nAvailFlashMemBytes`: Free hard disk space of the operating system in bytes.

`nRecvFrameFifoSize`: Number of frames currently in the receive frame queue (for receive queue see "Network adapter settings")

`nEmptyFrameFifoSize`: Available memory for buffering received frames (frame is copied and then copied to RecvFrameFifo)

`nRecvSegmUDPFramesListSize`: Number of currently received buffered IP segments (name is wrong, not UDP but IP)

`nRecvSegmFramesListSize`: Number of currently received buffered BACnet frame segments

`nSendSegmFramesListSize`: Number of current BACnet frame segments that were buffered before sending

`nClientScanListSize`: Number of currently known BACnet devices in the network (DeviceAddressBinding in the device object is formed from this)

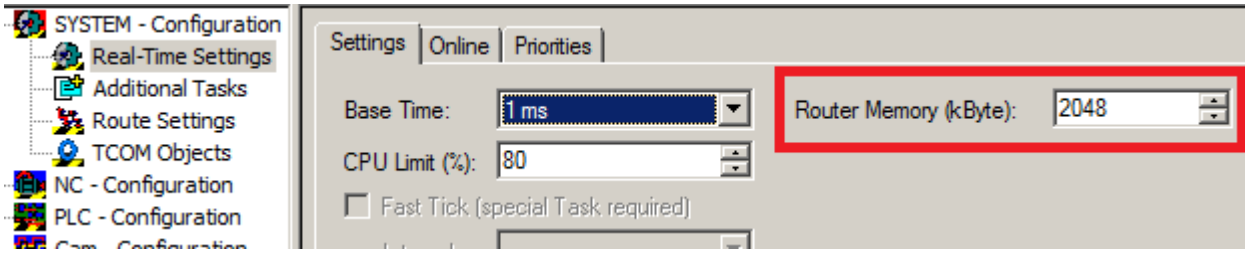


Fig. 79: Figure 1: AMS router memory setting

4.9.50 ST_BACnet_LimitEnable

PLC mapping of BACnet data type BACnetLimitEnable. See BACnet specification DIN EN ISO 16484-5 for property *Limit_Enable*.

```
TYPE ST_BACnet_LimitEnable :
STRUCT
  lowLimitEnable   : BOOL;
  highLimitEnable  : BOOL;
END_STRUCT
END_TYPE
```

`lowLimitEnable`: Lower limit value monitoring active (see property *Low_Limit*)

`highLimitEnable`: Upper limit value monitoring active (see property *High_Limit*)

If limits are exceeded, the property *Event_State* is set accordingly (see E_BACNETEVENTSTATE [▶ 336]).

4.9.51 ST_BACnet_LogBufferEntryReal

PLC mapping of BACnet data type BACnetLogRecord for *Log_Datum* of type *Real*. See BACnet specification DIN EN ISO 16484-5 for property *Log_Buffer*.

```
TYPE ST_BACnet_LogBufferEntryReal :
STRUCT
  nTimeStamp   : DWORD;
  stDateTime   : ST_BACnet_DateTime;
  fValue       : REAL;
END_STRUCT
END_TYPE
```

`nTimeStamp`: Time stamp in 100 ms since the year 1900

`fValue`: Value of the log entry

4.9.52 ST_BACnet_LogBufferReal

Structure for data exchange of property *Log_Buffer* with the aid of function block FB_BACnet_LogBufferProperty.

```
TYPE ST_BACnet_LogBufferReal :
STRUCT
  nGet          : USINT;
  sObjectDesc   : T_MAXSTRING;
  nCount        : UDINT;
  arrEntries    : ARRAY[0..BACnet_MaxLogBufferEntries] OF ST_BACnet_LogBufferEntryReal;
END_STRUCT
END_TYPE
```

`nGet`: Counter is incremented by 1 when property data were read successfully. Ring counter.

`sObjectDesc`: Value of the property *Description* of the corresponding BACnet object. If the property *Description* is empty, the content of the property *Object_Name* is read and entered.

`nCount`: Number of entries that were read. 0 to **nCount**-1 in **arrEntries** contain valid values.

`arrEntries`: Array of data from the property *Log_Buffer*. 0 to **nCount**-1 entries are available. Entries are limited to type Real of property *Log_Buffer*.

4.9.53 ST_BACnet_NSinkEvent

PLC mapping of the data of an event entry of the BACnet Notification Sink.

```
TYPE ST_BACnet_NSinkEvent :
STRUCT
  nProcessID    : UDINT;
  nDeviceID     : UDINT;
  tObjectID     : T_BACnet_ObjectIdentifier;
  stDateTime    : ST_BACnet_DateTime;
  nNC           : UDINT;
  nPriotity     : UDINT;
  eEventType    : E_BACNETEVENTTYPE;
  eNotifyType   : E_BACNETNOTIFYTYPE;
  eFromState    : E_BACNETEVENTSTATE;
  eToState      : E_BACNETEVENTSTATE;
  sMessage      : T_MaxString;
  bAcknReq      : BOOL;
END_STRUCT
END_TYPE
```

`nProcessID`: Identification number of the recipient process of the event.

`nDeviceID`: BACnet ID of the BACnet server that has sent the event.

`tObjectID`: Oject ID of the object that generated the event.

`stDateTime`: Time of the event.

`nNC`: BACnet instance of the corresponding BACnet Notification Class object.

`nPriotity`: Event priority.

`eEventType`: Event type.

`eNotifyType`: Message type.

`eFromState`: Object state prior to the event.

`eToState`: Oject state after the event.

`sMessage`: Message text (*Windows-1252* coded).

`bAcknReq`: *TRUE* → message must be acknowledged.

4.9.54 ST_BACnet_ObjectIdentifierList

```
TYPE ST_BACnet_ObjectIdentifierList :
STRUCT
  nEntries    : UDINT;
  arrObjList  : ARRAY[0..BACnet_MaxObjIdList] OF T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
END_STRUCT
END_TYPE
```

nEntries: Number of valid entries in **arrObjList** (index **0** ... **nEntries** - 1 contain valid values).

arrObjList: List with BACnet object IDs (object type and object instance).

4.9.55 ST_BACnet_ObjectTypesSupported

PLC mapping of BACnet data type BACnetObjectTypesSupported. See BACnet specification DIN EN ISO 16484-5 for property *Protocol_Object_Types_Supported*.

```
TYPE ST_BACnet_ObjectTypesSupported :
STRUCT
  SuppAnalogInput        : BOOL;
  SuppAnalogOutput       : BOOL;
  SuppAnalogValue        : BOOL;
  SuppBinaryInput        : BOOL;
  SuppBinaryOutput       : BOOL;
  SuppBinaryValue        : BOOL;
  SuppCalendar           : BOOL;
  SuppCommand            : BOOL;
  SuppDevice             : BOOL;
  SuppEventEnrollment    : BOOL;
  SuppFile               : BOOL;
  SuppGroup              : BOOL;
  SuppLoop               : BOOL;
  SuppMultiStateInput    : BOOL;
  SuppMultiStateOutput   : BOOL;
  SuppNotificationClass  : BOOL;
  SuppProgram            : BOOL;
  SuppSchedule           : BOOL;
  SuppAveraging          : BOOL;
  SuppMultiStateValue    : BOOL;
  SuppTrendLog           : BOOL;
  SuppLifeSafetyPoint    : BOOL;
  SuppLifeSafetyZone     : BOOL;
END_STRUCT
END_TYPE
```

`SuppAnalogInput`: FB_BACnet_AnalogInput [▶ 166]

`SuppAnalogOutput`: **FB_BACnet_AnalogOutput**

`SuppAnalogValue`: FB_BACnet_AnalogValue

`SuppBinaryInput`: **FB_BACnet_BinaryInput**

`SuppBinaryOutput`: **FB_BACnet_BinaryOutput**

`SuppBinaryValue`: **FB_BACnet_BinaryValue**

`SuppCalendar`: **FB_BACnet_Calendar**

`SuppCommand`: **FB_BACnet_Command**

`SuppDevice`: **FB_BACnet_Device**

`SuppEventEnrollment`: **FB_BACnet_EventEnrollment**

`SuppFile`: **FB_BACnet_File**

`SuppGroup`: **FB_BACnet_Group**

`SuppLoop`: **FB_BACnet_Loop**

`SuppMultiStateInput`: **FB_BACnet_MultiStateInput**

`SuppMultiStateOutput`: **FB_BACnet_MultiStateOutput**

`SuppNotificationClass`: **FB_BACnet_NotificationClass**

`SuppProgram`: **FB_BACnet_Program**

`SuppSchedule`: **FB_BACnet_Schedule**

`SuppAveraging`: **FB_BACnet_Averaging**

`SuppMultiStateValue`: **FB_BACnet_MultiStateValue**

`SuppTrendLog`: **FB_BACnet_TrendLog**

`SuppLifeSafetyPoint`: **SuppLifeSafetyZone:**

4.9.56 ST_BACnet_ProgramHandshakeRequests

```
TYPE ST_BACnet_ProgramHandshakeRequests :
STRUCT
  bLoad    : BOOL;
  bSetup   : BOOL;
  bRun     : BOOL;
  bHalt    : BOOL;
  bUnload  : BOOL;
END_STRUCT
END_TYPE
```

4.9.57 ST_BACnet_ProgramHandshakeStates

```
TYPE ST_BACnet_ProgramHandshakeStates :
STRUCT
  bIdle       : BOOL;
  bLoading    : BOOL;
  bRunning    : BOOL;
  bWaiting    : BOOL;
  bHalted     : BOOL;
  bUnloading  : BOOL;
END_STRUCT
END_TYPE
```

4.9.58 ST_BACnet_RecipientListDevice

Structure for data exchange of property *Recipient_List* with the aid of function block FB_BACnet_RecipientListProperty.

```
TYPE ST_BACnet_RecipientListDevice :
STRUCT
  nEntries     : INT;
  arrDestList  : ARRAY[0..BACnet_MaxDestinationArray] OF ST_BACnet_RecipientListDeviceEntry;
END_STRUCT
END_TYPE
```

`nEntries`: Number of entries that were read or written. 0 to **nEntries**-1 in **arrDestList** contain valid values.

`arrDestList`: Array with data from the property *Recipient_List*. 0 to **nCount**-1 entries are available.

4.9.59 ST_BACnet_RecipientListDeviceEntry

```
TYPE ST_BACnet_RecipientListDeviceEntry :
STRUCT
  bMonday              : BOOL:=TRUE;
  bTuesday             : BOOL:=TRUE;
  bWednesday           : BOOL:=TRUE;
  bThursday            : BOOL:=TRUE;
  bFriday              : BOOL:=TRUE;
  bSaturday            : BOOL:=TRUE;
  bSunday              : BOOL:=TRUE;
  stFromTime           : ST_BACnet_Time := (nHour       := 0,
                             nMinute     := 0,
                             nSecond     := 0,
                             nHundredths := 0);
  stToTime             : ST_BACnet_Time := (nHour       := 23,
                             nMinute     := 59,
                             nSecond     := 59,
                             nHundredths := 99);
  tDeviceId            : T_BACnet_ObjectIdentifier:=16#FFFFFFFF;
  nProcessId           : UDINT;
```

```
  bToOffNormal          : BOOL:=TRUE;
  bToFault              : BOOL:=TRUE;
  bToNormal             : BOOL:=TRUE;
  bIssueConfirmedNotification  : BOOL;
END_STRUCT
END_TYPE
```

`bMonday`, `bTuesday`, `bWednesday`, `bThursday`, `bFriday`, `bSaturday` and `bSunday`: Flags for the days of the week: *TRUE* for the respective day of the week enables the recipient for the corresponding day; *FALSE* = no transfer to the recipient.

`stFromTime` and `stToTime`: range from/to which the transfer of messages to the recipient is enabled (e.g. activate alarm for motion sensors or window contacts only during the night).

`tDeviceId`: BACnet ID of the recipient BACnet server.

`nProcessId`: BACnet process ID of the recipient.

`bToOffNormal`: Transition to *OFF-NORMAL* are transferred.

`bToFault`: Transition to *FAULT* are transferred.

`bToNormal`: Transitions to *NORMAL* are transferred.

`bIssueConfirmedNotification`: The message should be marked as to be acknowledged.

4.9.60 ST_BACnet_ServerStatistics

```
TYPE ST_BACnet_ServerStatistics :
STRUCT
  nOpenRequests         : DWORD;
  nRequestRetries       : DWORD;
  nRequestTimeOuts      : DWORD;
  nPropChanges          : DWORD;
  nPersistWrites        : DWORD;
  persistChangeSetId    : DWORD;
  secpersistNextWrite   : DWORD;
  persistLastWrite      : ST_BACnet_DateTime;
END_STRUCT
END_TYPE
```

4.9.61 ST_BACnet_ServicesSupported

PLC mapping of BACnet data type BACnetServicesSupported. See BACnet specification DIN EN ISO 16484-5 for property *Protocol_Services_Supported.*

```
TYPE ST_BACnet_ServicesSupported :
STRUCT
  SuppAcknowledgeAlarm              : BOOL;
  SuppConfirmedCOVNotification      : BOOL;
  SuppConfirmedEventNotification    : BOOL;
  SuppGetAlarmSummary               : BOOL;
  SuppGetEnrollmentSummary          : BOOL;
  SuppSubscribeCOV                  : BOOL;
  SuppAtomicReadFile                : BOOL;
  SuppAtomicWriteFile               : BOOL;
  SuppAddListElement                : BOOL;
  SuppRemoveListElement             : BOOL;
  SuppCreateObject                  : BOOL;
  SuppDeleteObject                  : BOOL;
  SuppReadProperty                  : BOOL;
  SuppReadPropertyConditional       : BOOL;
  SuppReadPropertyMultiple          : BOOL;
  SuppWriteProperty                 : BOOL;
  SuppWritePropertyMultiple         : BOOL;
  SuppDeviceCommunicationControl    : BOOL;
  SuppConfirmedPrivateTransfer      : BOOL;
  SuppConfirmedTextMessage          : BOOL;
  SuppReinitializeDevice            : BOOL;
  SuppVtOpen                        : BOOL;
  SuppVtClose                       : BOOL;
```

```
  SuppVtData                          : BOOL;
  SuppAuthenticate                    : BOOL;
  SuppRequestKey                      : BOOL;
  SuppIAm                             : BOOL;
  SuppIHave                           : BOOL;
  SuppUnconfirmedCOVNotification      : BOOL;
  SuppUnconfirmedEventNotification    : BOOL;
  SuppUnconfirmedPrivateTransfer      : BOOL;
  SuppUnconfirmedTextMessage          : BOOL;
  SuppTimeSynchronization             : BOOL;
  SuppWhoHas                          : BOOL;
  SuppWhoIs                           : BOOL;
  SuppReadRange                       : BOOL;
  SuppUtcTimeSynchronization          : BOOL;
  SuppLifeSafetyOperation             : BOOL;
  SuppSubscribeCOVProperty            : BOOL;
  SuppGetEventInformation             : BOOL;
END_STRUCT
END_TYPE
```

4.9.62 ST_BACnet_StatusFlags

PLC mapping of BACnet data type BACnetStatusFlags. See BACnet specification DIN EN ISO 16484-5 for property *Status_Flags*.

```
TYPE ST_BACnet_StatusFlags :
STRUCT
  in_alarm        : BOOL;
  fault           : BOOL;
  overridden      : BOOL;
  out_of_service  : BOOL;
END_STRUCT
END_TYPE
```

`in_alarm`: *FALSE*, if property *Event_State* = *NORMAL*, otherwise *TRUE*.

`fault`: *TRUE*, if property *Reliablility* ≠ *NO_FAULT_DETECTED*, otherwise *FALSE*.

`overridden`: *TRUE*, if overwriting of the output of the corresponding object is indicated via process data RawIoStateOverride = *TRUE* and RawIoStateOverrideInverted = *FALSE* (typically in manual operating modules).

`out_of_service`: *TRUE*, if property *Out_Of_Service* = *TRUE*, otherwise *FALSE*.

4.9.63 ST_BACnet_TcIoEthStatistic

Structure with information for sending and receiving of network frames from the network driver.

```
TYPE ST_BACnet_TcIoEthStatistic :
STRUCT
  dcTimeStamp    : ARRAY[0..1] OF DWORD;
  sendFrames     : DWORD;
  recvFrames     : DWORD;
  sendTimeRTime  : DWORD;
  recvTimeRTime  : DWORD;
  sendTimeNdis   : DWORD;
  recvTimeNdis   : DWORD;
END_STRUCT
END_TYPE
```

`dcTimeStamp`: Real-time timestamp (DC time).

`sendFrames`: Sent Ethernet frames of the BACnet adapter.

`recvFrames`: Processed Ethernet frames of the BACnet adapter.

`sendTimeRTime`: Non-BACnet specific information from the network driver (internal diagnosis).

`recvTimeRTime`: Non-BACnet specific information from the network driver (internal diagnosis).

`sendTimeNdis`: Non-BACnet specific information from the network driver (internal diagnosis).

`recvTimeNdis`: Non-BACnet specific information from the network driver (internal diagnosis).

4.9.64 ST_BACnet_TcIoEthTxRxErrorCount

Structure with information on faulty network frames from the network driver.

```
TYPE ST_BACnet_TcIoEthTxRxErrorCount :
STRUCT
  txCnt  : DWORD;
  rxCnt  : DWORD;
END_STRUCT
END_TYPE
```

`txCnt`: Number of faulty sent frames (lost sent packets).

`rxCnt`: Number of faulty received frames (discarded received packets).

4.9.65 ST_BACnet_Time

PLC mapping of BACnet data type Time. See BACnet specification DIN EN ISO 16484-5 for data type BACnetDateTime.

```
TYPE ST_BACnet_Time :
STRUCT
  nHour        : BYTE:=16#FF;
  nMinute      : BYTE:=16#FF;
  nSecond      : BYTE:=16#FF;
  nHundredths  : BYTE:=16#FF;
END_STRUCT
END_TYPE
```

`nHour`: Possible values for hour: 0 to 23; 255 corresponds to undefined (or each hour)

`nMinute`: Possible values for minute: 0 to 59; 255 corresponds to undefined (or each minute)

`nSecond`: Possible values for second: 0 to 59; 255 corresponds to undefined (or each second)

`nHundredths`: Possible values for hundredths of a second: 0 to 99; 255 corresponds to undefined (or each hundredth of a second)

4.9.66 ST_BACnet_TimeValue

```
TYPE ST_BACnet_TimeValue :
STRUCT
  time              : BACnetTime ;
  reserved          : DWORD ;
  value             : BACnetValue ;
  BACnetTimeValue   : *PBACnetTimeValue;
  stTime            : ST_BACnet_Time;
  nRes              : DWORD;
  stValue           : ST_BACnet_Value;
END_STRUCT
END_TYPE
```

4.9.67 ST_BACnet_TimeValueBool

PLC mapping of BACnet data type BACnetTimeValue for entries of type Bool or Null. See BACnet specification DIN EN ISO 16484-5 for data type BACnetTimeValue.

```
TYPE ST_BACnet_TimeValueBool :
STRUCT
  bValid  : BOOL;
  stTime  : ST_BACnet_Time;
  eType   : E_BACnetDataTypes;
```

```
  bValue  : BOOL;
END_STRUCT
END_TYPE
```

bValid: Value is valid.

eType: Possible values are BACnetDataType_Null or BACnetDataType_Boolean.

bValue: Value *FALSE* or *TRUE*, if **eType** = BACnetDataType_Boolean.

4.9.68 ST_BACnet_TimeValueList

```
TYPE ST_BACnet_TimeValueList :
STRUCT
  nEntrySize           : DWORD ;
  reserved             : DWORD ;
  listEntry            : BACnetTimeValue ;
  BACnetTimeValueList  : *PBACnetTimeValueList;
  nEntrySize           : DWORD;
  nRes                 : DWORD;
  stListEntry          : ST_BACnet_TimeValue;
END_STRUCT
END_TYPE
```

4.9.69 ST_BACnet_UnConfirmedServiceDiag

Structure with information on unconfirmed BACnet services (unconfirmed).

```
TYPE ST_BACnet_UnConfirmedServiceDiag :
STRUCT
  nReqSend      : DWORD;
  nReqSendFail  : DWORD;
  nReqRecv      : DWORD;
  nReqRecvFail  : DWORD;
END_STRUCT
END_TYPE
```

`nReqSend`: Number of sent requests (unconfirmed requests).

`nReqSendFail`: Number of failed requests.

`nReqRecv`: Number of received requests (unconfirmed requests).

`nReqRecvFail`: Number of requests that could not be processed (no free memory, wrong coding etc.).

4.9.70 ST_BACnet_Value

```
TYPE ST_BACnet_Value :
STRUCT
  BACnetValueChoice      : choice;
  length                 : DWORD;
  boolValue              : BOOL;
  realValue              : FLOAT;
  doubleValue            : DOUBLE;
  signedValue            : INT;
  unsignedValue          : UINT;
  pOctedStringValue      : BYTE*;
  pCharStringValue       : CHAR*;
  objectIdentifierValue  : BACnetObjectIdentifier;
  enumerationValue       : EnumerationValueType ;
  BACnetValue            : *PBACnetValue;
  eValueChoice           : E_BACnetDataTypes;
  nRes1                  : WORD;
  cbData                 : DWORD;
  nData                  : BYTE;
END_STRUCT
END_TYPE
```

4.9.71 ST_BACnet_WeeklyScheduleBool

Structure for data exchange of property *Weekly_Schedule* with the aid of function block FB_BACnet_WeeklyScheduleProperty.

```
TYPE ST_BACnet_WeeklyScheduleBool :
STRUCT
  bReqGet     : BOOL;
  bReqSet     : BOOL;
  nGet        : USINT;
  nSet        : USINT;
  bBusy       : BOOL;
  arrEntries  : ARRAY[0..6] OF ARRAY[0..BACnet_MaxDayEntry] OF ST_BACnet_TimeValueBool;
END_STRUCT
END_TYPE
```

`bReqGet`**:** Request read property data *FALSE* → *TRUE* for the corresponding property function block (see FB_BACnet_WeeklyScheduleProperty). The edge change *TRUE* → *FALSE* indicates acknowledgement of the request by corresponding property function block. Current data will be available shortly. Successful loading of the property data is indicated by `nGet`.

`bReqSet`**:** Request write property data *FALSE* → *TRUE* for the corresponding property function block (see FB_BACnet_WeeklyScheduleProperty). The edge change *TRUE* → *FALSE* indicates acknowledgement of the request by corresponding property function block. Current data will be available shortly. Successful writing of the property data is indicated by `nSet`.

`nGet`**:** Counter is incremented by 1 when property data were read successfully. Ring counter.

`nSet`**:** Counter is incremented by 1 when property data were written successfully. Ring counter.

`bBusy`**:** Corresponding property function block loads/writes data from/to the property. The data are invalid while **bBusy** is set to *TRUE*.

`arrEntries`**:** Array with data of property *Weekly_Schedule*. Entry 0 → Monday to 6 → Sunday; for each day of the week 0 to BACnet_MaxDayEntry entries are available. The entries are coded as BACnetTimeValue and are limited to data types of Boolean and Null. Entries with other data types are ignored.

5 PLC library: TcBACnet.Lib

The functionality of the PLC library "TcBACnet.lib" is described below. The library offers convenient access to objects of a BACnet configuration from a PLC program.

The application of the library is **not** compulsory.

The data (properties) of the BACnet objects can be accessed in two ways:

1. The function blocks or self-declared variables are linked with the aid of "synchronous" process data mapping (linking of the PLC with the hardware configuration in the System Manager). For this purpose the function blocks of the PLC include the allocation as input or output data ("bValue AT%I* : BOOL" or "bValue AT%Q* : BOOL"). The advantage is that the data are updated cyclically. All values are written or read synchronous with the PLC cycle.
2. Object data are read or written asynchronously with the aid of ADS. Data to be written/read are sent to the corresponding address with the aid of ADS blocks (NetID, port, index group and offset --> object type, instances and property ID) (FB_BACnet_WriteProp [▶ 497]) or read by it (FB_BACnet_ReadProp [▶ 494]). This method has the advantage that it is possible to write data only if an actual change has occurred. However, in cases with frequent changes unforeseeable delays may occur. The maximum delay is 5 seconds ("tBACnet_ADSTimeOut : TIME :=t#5s"). After this time the block returns an error code.

 . The library (https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749057547/.zip) can be downloaded from here.

Installation: https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749057547/.zipThe contents of the ZIP archive must be copied into the TwinCAT folder (default: "C:\TwinCAT") under "\PLC\Lib".

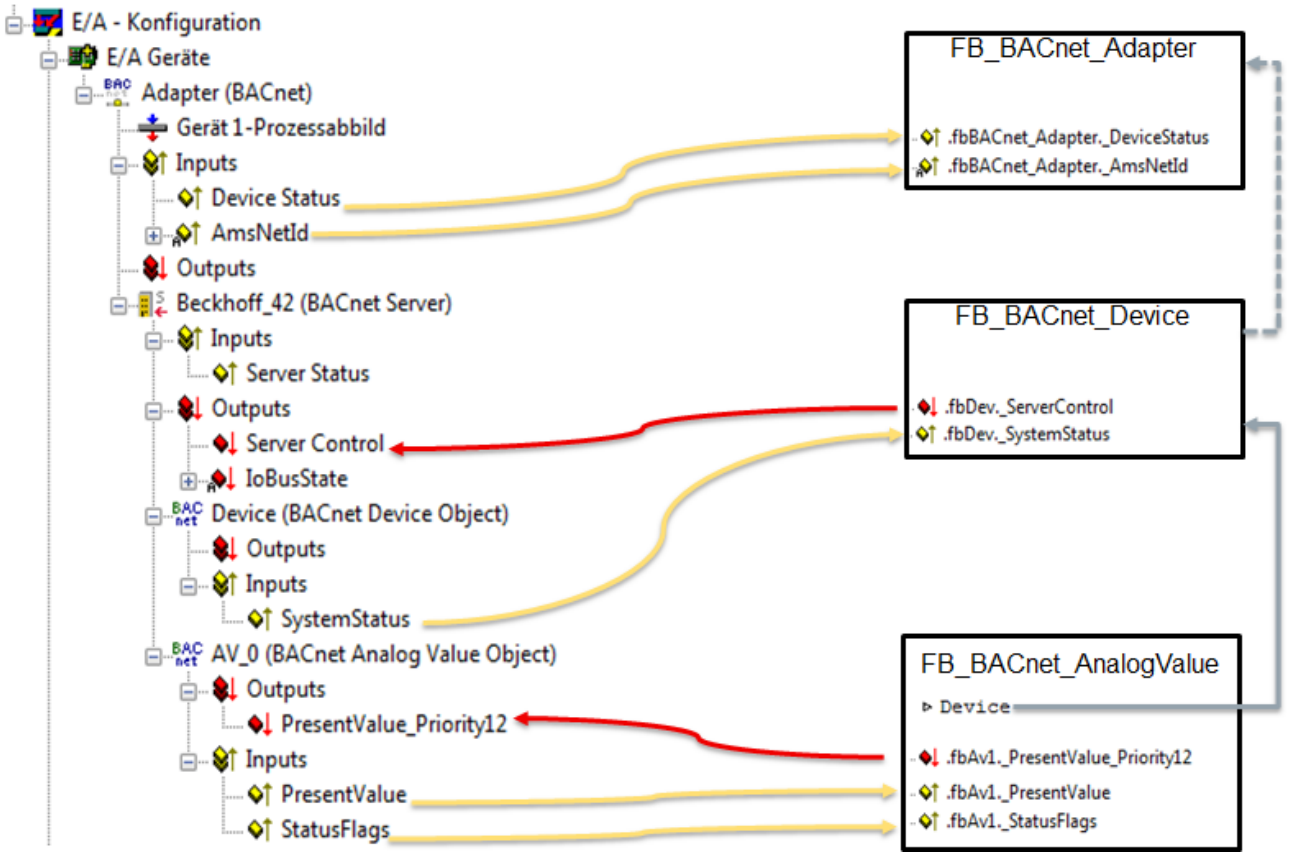
Overview

The PLC library TcBACnet-Lib is a collection of function blocks for programming a BACnet controller. For each BACnet object function blocks are available that are read and written via the selected properties. In general, a distinction is made between function blocks for server and client objects. Function blocks for client objects have the prefix "Remote“ since they enable access to remote BACnet objects on other devices. The BACnet object function blocks ascertain whether process data are valid and the BACnet controller is ready for operation (bReady).

The function block FB_BACnet_Adapter [▶ 375] represents a BACnet device and therefore the access point to the BACnet network via a network card. This block enables the existence of a link to be verified, for example. To make the PLC program more transparent, the BACnet adapter is created as a global variable within the TcBACnet-Lib, since in the majority of the projects a single BACnet adapter is used.

Depending on the client and server functionality, the function blocks FB_BACnet_Device [▶ 414] or FB_BACnet_RemoteDevice [▶ 459] can be used to link state and control information for the BACnet client and server. The operational readiness of the BACnet objects is determined via the client and server status and the property System_Status for the respective device or remote device object. Each BACnet object function block must therefore transfer a reference to the instance of the corresponding FB_BACnet_Device [▶ 414] or FB_BACnet-RemoteDevice [▶ 459], in order to enable the status evaluation (bReady).

The following figure shows an overview of the function blocks of TcBACnet-Lib and the corresponding links with the BACnet modules. The linking between a PLC program based on TcBACnet-Lib and the BACnet modules of a configuration can be automated via the function "PLC automapping [▶ 62]".



Block names

The block names refer to the block versions with maximum process data (extension "_EX"). The standard blocks contain a minimum set of process data, to avoid performance impairment in large configurations. In addition there are function blocks for objects with commandable properties Present_Value which realise write access via ADS (extension "_ADS"). This has the advantage that the process data are not copied cyclically but are only written to the object in the event of changes. This also enables repeated write access to local objects (writing with identical priority). Function block variants with extension "_RAW" provide access to the raw value of the corresponding object. Using this module variant, the property Present_Value is not mapped to the terminal hardware but in the PLC program. This allows for example to connect BACnet values with values of sub bus systems. An example is shown in the block description.

5.1 Example: NotificationClass and NotificationSink

The following is an example of using the *NotificationClass* in conjunction with a *NotificationSink*. This "collects" the events of 3 objects (BV, AV, MV) and sends them to the local process 10 (*NotificationSink*) of the local BACnet device (event notification by intrinsic reporting).

In addition the *PresentValue* of the *AnalogValue* object should be sent to the *NotificationSink* if the value changes by 0.1 (COV notification).

The example https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749047307/.zip can be downloaded here https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749047307/.zip. The included TSM file can be used for testing purposes without loading the PLC.

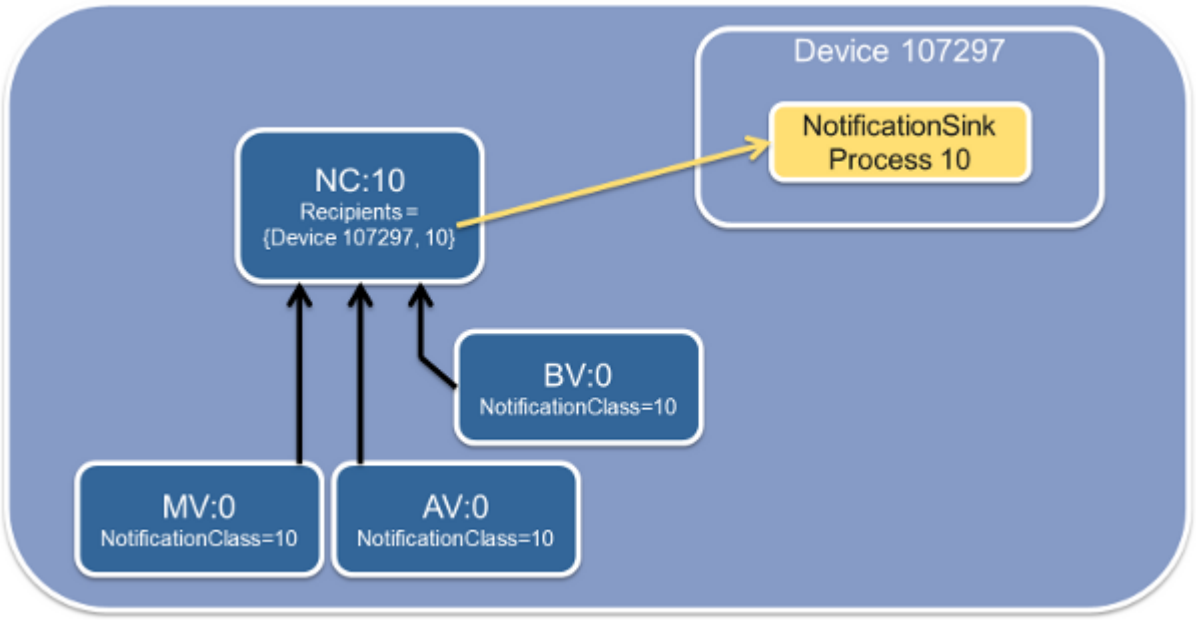


Fig. 80: Figure 1: Schematic diagram of an NotificationClass example configuration, its event-generating objects (AV, MV, BV) and the NotificationSink.

The objects are configured via the declaration in the PLC program:

```
VAR
 viewDummy : BOOL; (* ~( BACnet_StructuredViewPath : \/DemoNClass : ) *)

 fbDevice : FB_BACnet_Device;

 fbBV_0 : FB_BACnet_BinaryValue;
  (* ~(BACnet_ObjectIdentifier : 0 : nolink )
  (BACnet_EventEnable : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_ActiveText : EIN : nolink )
  (BACnet_InactiveText : AUS : nolink )
  (BACnet_NotificationClass : 10 : nolink ) *)

 fbAV_0 : FB_BACnet_AnalogValue;
  (* ~(BACnet_ObjectIdentifier : 0 : nolink )
  (BACnet_EventEnable : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_NotificationClass : 10 : nolink ) *)

 fbMV_0 : FB_BACnet_MultiStateValue;
  (* ~(BACnet_ObjectIdentifier : 0 : nolink )
  (BACnet_EventEnable : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_NotificationClass : 10 : nolink )
  (BACnet_NumberOfStates : 5 : nolink )
  (BACnet_StateText : ErrLow;WarnLow;Normal;WarnHigh;ErrHigh : nolink )
  (BACnet_FaultValues : 1;5 : nolink )
  (BACnet_AlarmValues : 2;4 : nolink )
  (BACnet_Description : MV DEMO Objekt. : nolink ) *)

 fbNC_10 : FB_BACnet_NotificationClass;
  (* ~(BACnet_ObjectIdentifier : 10 : nolink )
  (BACnet_Priority : 70;80;90 : nolink )
  (BACnet_AckRequired : {to_offnormal;to_fault;to_normal} : nolink )
  (BACnet_RecipientList :
      <ArrayOfBACnetDestination>
    <BACnetDestination>
      <recipient>
        <choice>device</choice>
        <BACnetObjectIdentifier>
          <objId>33661729</objId>
        </BACnetObjectIdentifier>
      </recipient>
      <validDays>65025</validDays>
      <fromTime>
        <hour>0</hour>
        <minute>0</minute>
        <second>0</second>
        <hundredths>0</hundredths>
      </fromTime>
      <toTime>
```

```
        <hour>23</hour>
        <minute>59</minute>
        <second>59</second>
        <hundredths>99</hundredths>
      </toTime>
      <processIdentifier>10</processIdentifier>
      <transitions>57349</transitions>
      <issueConfirmedNotifications>false</issueConfirmedNotifications>
    </BACnetDestination>
      </ArrayOfBACnetDestination>
  : nolink ) *)END_VAR
```

The NotificationSink added for receiving the events is configured in the System Manager: *NotificationSink*

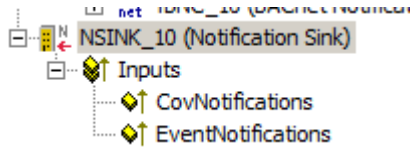


Fig. 81: Figure 2: NotificationSink in the System Manager tree view

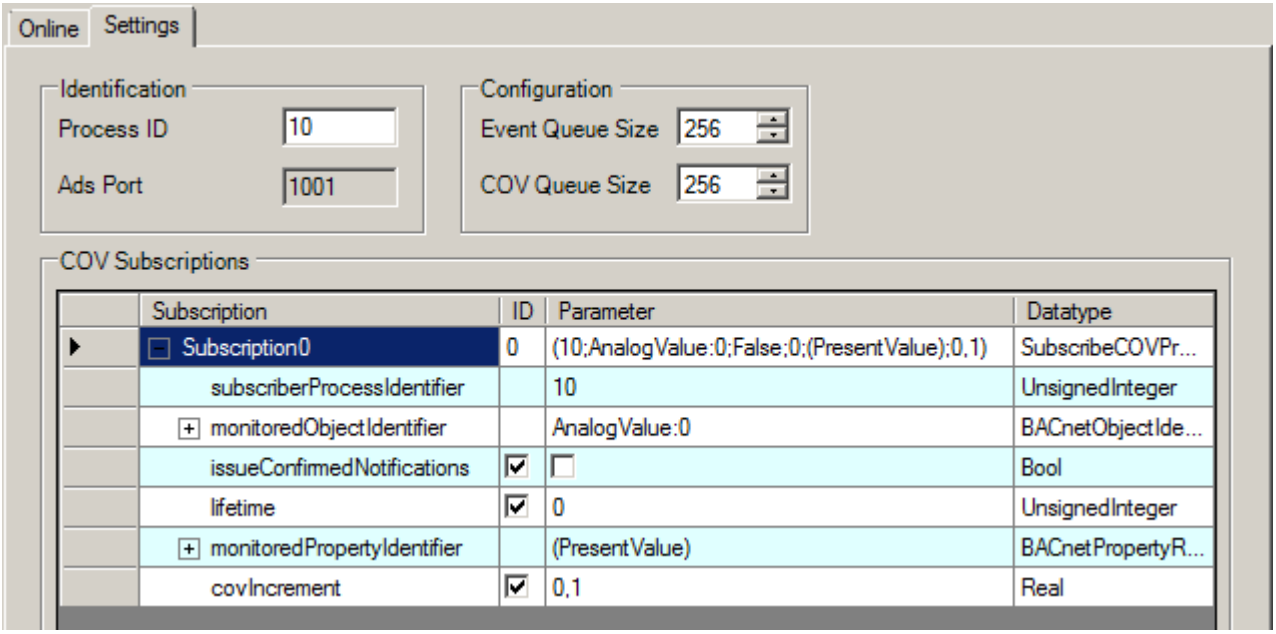


Fig. 82: Figure 3: NotificationSink configuration in the System Manager

Process ID: The process ID is set to 10 and reported to the *NotificationClass* as a receive process (via PLC automapping comment).

COV Subscriptions: As an example the *PresentValue* of the *AnalogValue* object (AV:0) is subscribed to. The parameter *covIncrement* specifies the change of value threshold at which a COV notification is sent.

The following entries are displayed in the Online tab, once the configuration has been loaded and the values of the respective objects have been modified: *NotificationSink PresentValue*

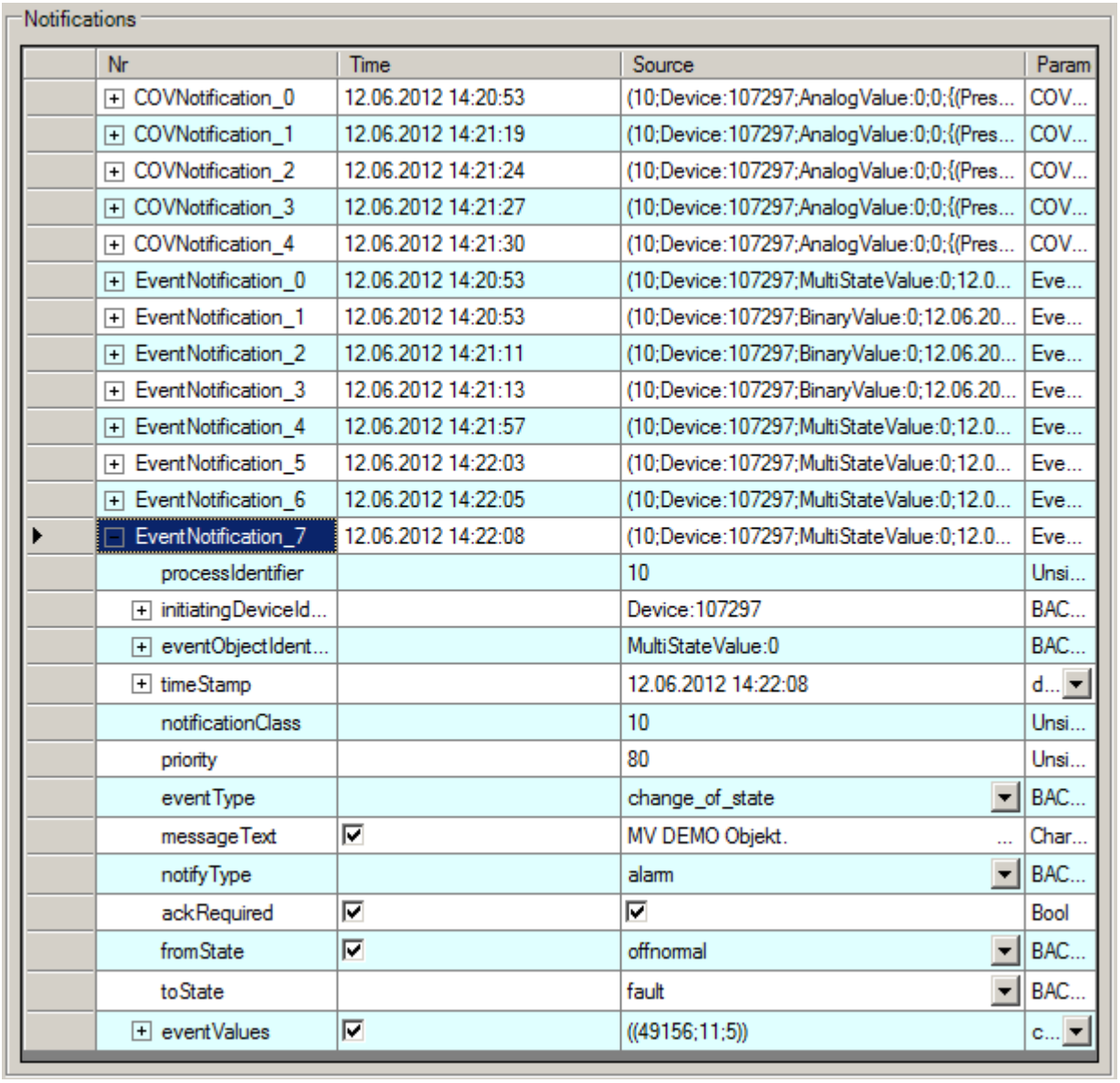


Fig. 83: Figure 4: NotificationSink online data after COV and event notifications have been received.

The text under messageText is formed from the property description of the event-generating object.

5.2 FB_BACnet_Adapter

The following function block is used for linking the PLC program with a local BACnet adapter (network card). The linking of the function block takes place with the aid of process data.

The process data can be linked manually or automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.

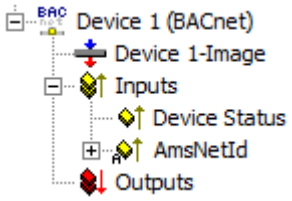


Fig. 84: Fig. 1: Process data of the BACnet adapter in the System Manager.

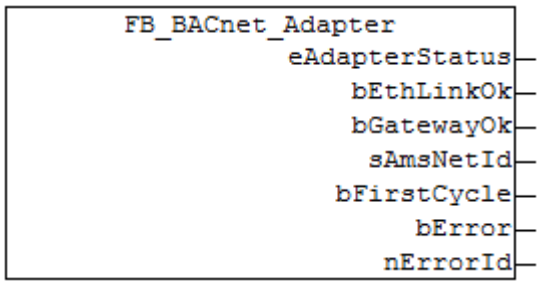


Fig. 85: Fig. 2: Function block of the BACnet adapter in the PLC program.

Use

The function block FB_BACnet_Adapter is used to read the state of the local BACnet adapter (process data "Device Status") and output it in the PLC program. In addition, the process data "AmsNetID" is used to read the AMS NetID of the BACnet adapter (ARRAY OF BYTE --> string) and to output it as a string.

An instance of the function block FB_BACnet_Adapter is provided through the PLC library as a global instance and does therefore *not* have to be created explicitly. The global instance is detected during PLC automapping and automatically linked with the BACnet adapter in the System Manager. The function block instance is required by the function blocks FB_BACnet_Device and FB_BACnet_RemoteDevice. The hierarchy between FB_BACnet_Adapter, FB_BACnet_Device and FB_BACnet_RemoteDevice and an object of type AnalogValue is shown below:

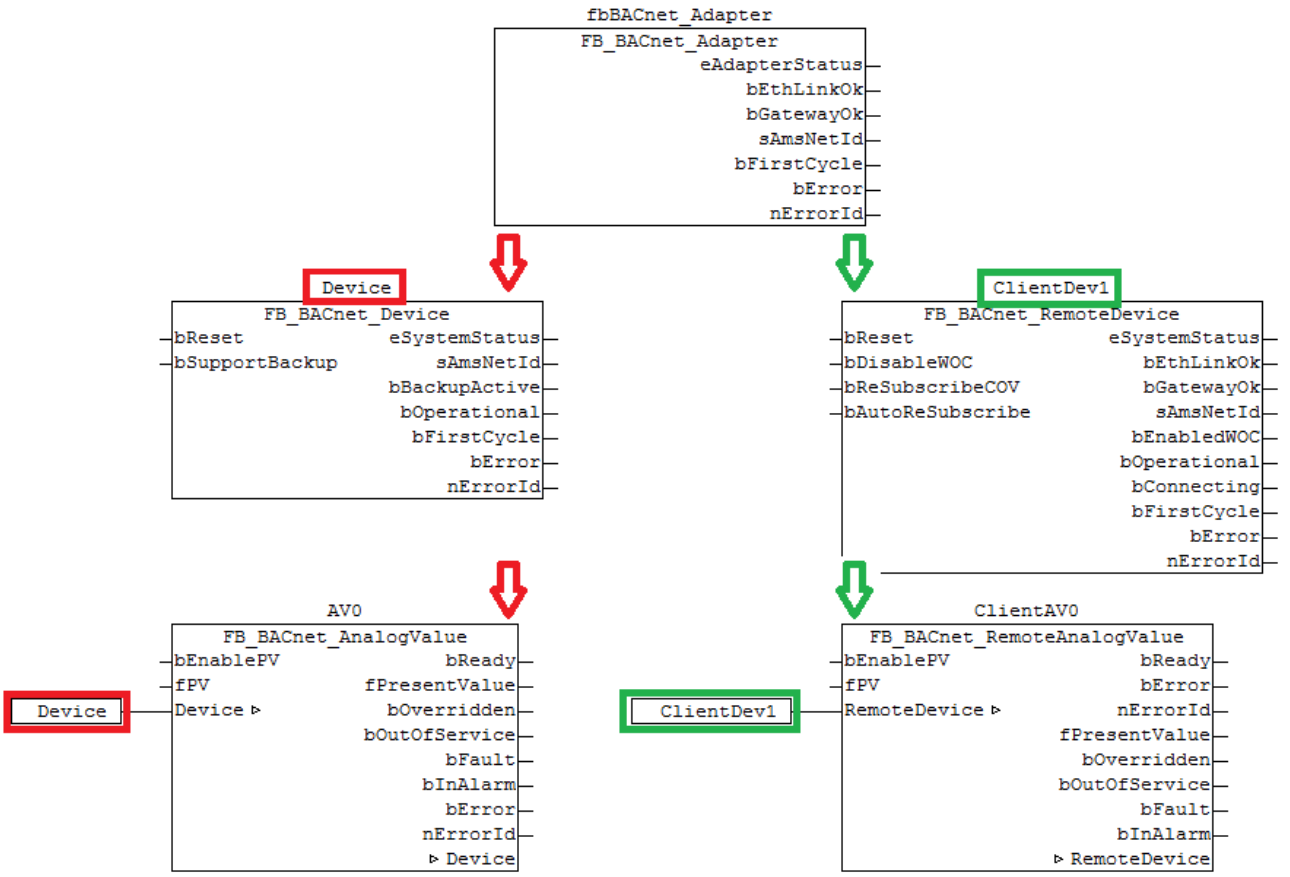


Fig. 86: Figure 3: Example for linking the FB instances in the PLC.

For information on the use of the function blocks FB_BACnet_Device and _RemoteDevice, see FB_BACnet_Device or FB_BACnet_RemoteDevice. Calling of the instance of the function block FB_BACnet_Adapter is not compulsory. If required, the adapter function block is called automatically by the instances of the function blocks FB_BACnet_Device or FB_BACnet_RemoteDevice.

VAR_OUTPUT

```
eAdapterStatus    : E_BACnetAdapterStatus;
bEthLinkOk        : BOOL;
bGatewayOk        : BOOL;
sAmsNetId         : T_AmsNetId;
bFirstCycle       : BOOL;
bError            : BOOL;
nErrorId          : UINT;
```

eAdapterStatus: Status of the BACnet adapter. The following values are possible:
0: BACnetAdapterStatus_Init (initialisation has commenced)
1: BACnetAdapterStatus_CheckIpAddr (checking the IP address)
2: BACnetAdapterStatus_CheckParam (checking the parameter)
3: BACnetAdapterStatus_GetGatewayMAC (determining the MAC of the gateway)
4: BACnetAdapterStatus_WaitGatewayMAC (waiting for MAC address of the gateway)
5: BACnetAdapterStatus_Complete (initialised and ready)

bEthLinkOk: The local Ethernet connection is active (cable connected, adapter linked) if the output is set to *TRUE*.

bGatewayOk: The configured gateway can be reached if the output is set to *TRUE*.

sAmsNetId: Output of the AMS NetID of the local BACnet adapter (can be used for asynchronous access to BACnet objects via ADS).

bFirstCycle: Is set for one cycle when the block instance is first called after a PLC reset or restart.

bError: An error is pending.

nErrorId: Error number (0 = no error; 1 = no valid AMS NetID). The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Adapter*.**nERR_xxx**).

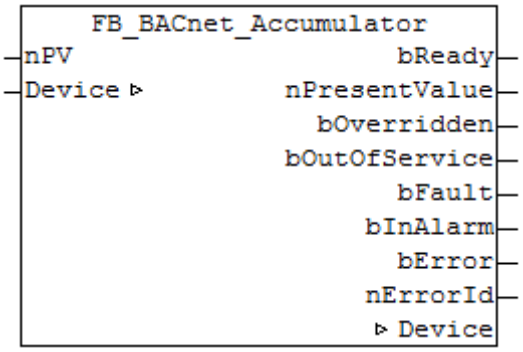
If there is no network connection (**bEthLinkOk** = *FALSE*) or the gateway cannot be reached (**bGatewayOk** = *FALSE*), all connected remote objects (clients) are blocked. Local objects are not affected.

5.3 BACnet Server Objects

5.3.1 FB_BACnet_Accumulator

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Accumulator" can be used for read and write access to a BACnet object of type *Accumulator* (ACC). To this end the BACnet object was created under a local BACnet server.

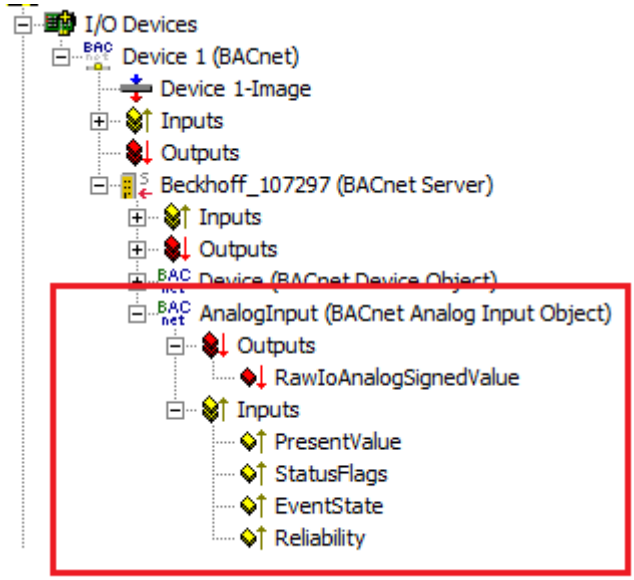


Fig. 87: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
nPV          : UDINT;
```

nPV: Value to be written to the property *Present_Value*. Process data are always written cyclically.

VAR_OUTPUT

```
bReady           : BOOL;
nPresentValue    : UDINT;
nMaxPV           : UDINT;  (*siehe Beschreibung*)
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is FALSE, the corresponding function block "FB_BACnet_Device [▶ 414]" does not report "Operational", or the function block instance was not linked correctly in the System Manager.

nPresentValue: current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_Value*).

bOverriden, bOutOfService, bFault, bInAlarm: see BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bError: an error is pending.

nErrorId: error number
0 = no error
2 = faulty process data mapping detected (check mapping in the System Manager; possibly compile PLC project completely and reload)
3 = the associated BACnet server is not ready (**bOperational** = *FALSE* at instance of FB_BACnet_Device [▶ 414])
The error numbers can be queried as function block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

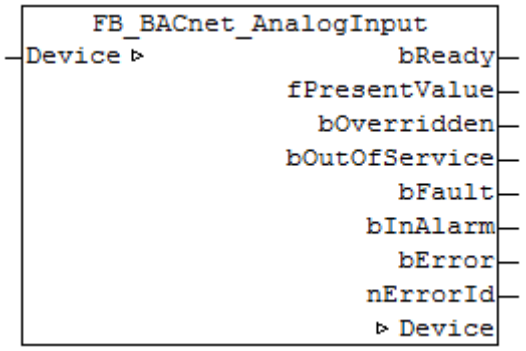
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.2 FB_BACnet_AnalogInput

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_AnalogInput" can be used for read access to a BACnet object of type *AnalogInput* (AI). To this end the BACnet object was created under a local BACnet server.

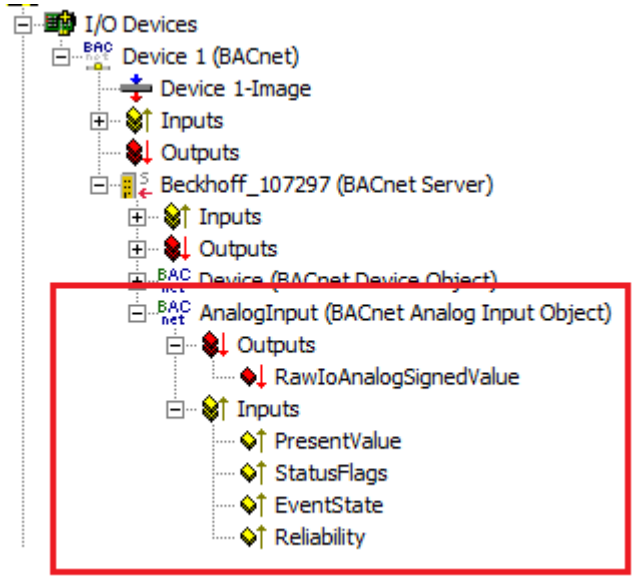


Fig. 88: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Present_Value*).

bOverriden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx).**

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

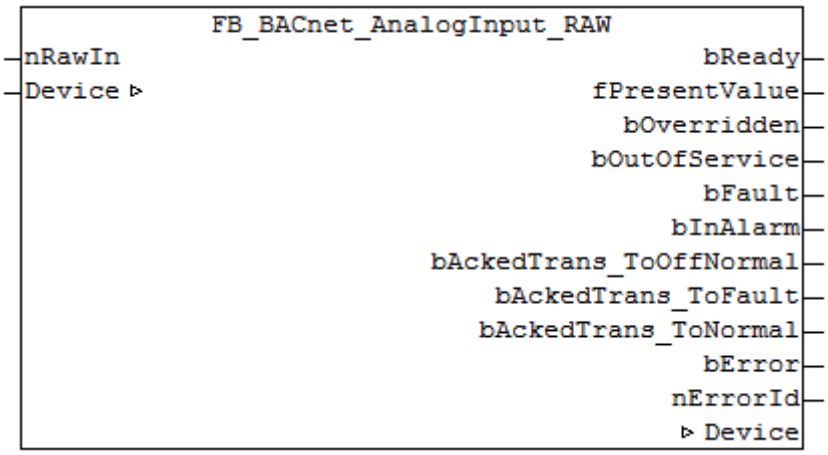
Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and and FB_BACnet_Device for further information.

5.3.3 FB_BACnet_AnalogInput_RAW

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.

Contrary to the standard variant of the function block, the raw value is provided by a function block input and not by the terminal hardware. This allows e.g. Analog input information that are generated in the PLC code displayed as an analog input object to BACnet (signal mapping for sub bus systems or virtual data points, etc.).



Use

The function block "FB_BACnet_AnalogInput_RAW" can be used for read access to a BACnet object of type *AnalogInput* (AI). To this end the BACnet object was created under a local BACnet server.

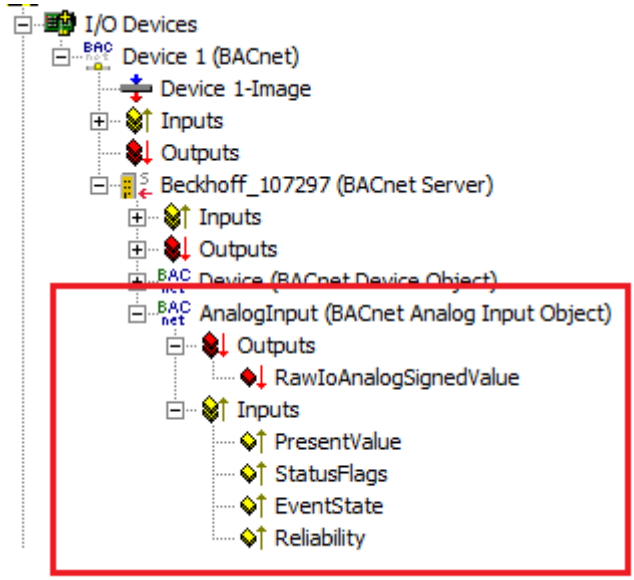


Fig. 89: BACnetObject_SysMan

Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
nRawIn       : INT; (* ~(BACnet_RawIoAnalogSignedValue : : ) *)
```

nRawIn: Raw value input of the object in the range -32768 ... 32767. The input is linked to the process data "RawIoAnalogSignedValue" of the BACnet object. The value of "nRawIn" results to property *Present_Value* according to property *Resolution* (assuming the object state is not *out_of_service*).

VAR_OUTPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Present_Value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx).**

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

Example

The following example shows the mapping of an EIB integer value into the Property *Present_Value* of an analog input object:

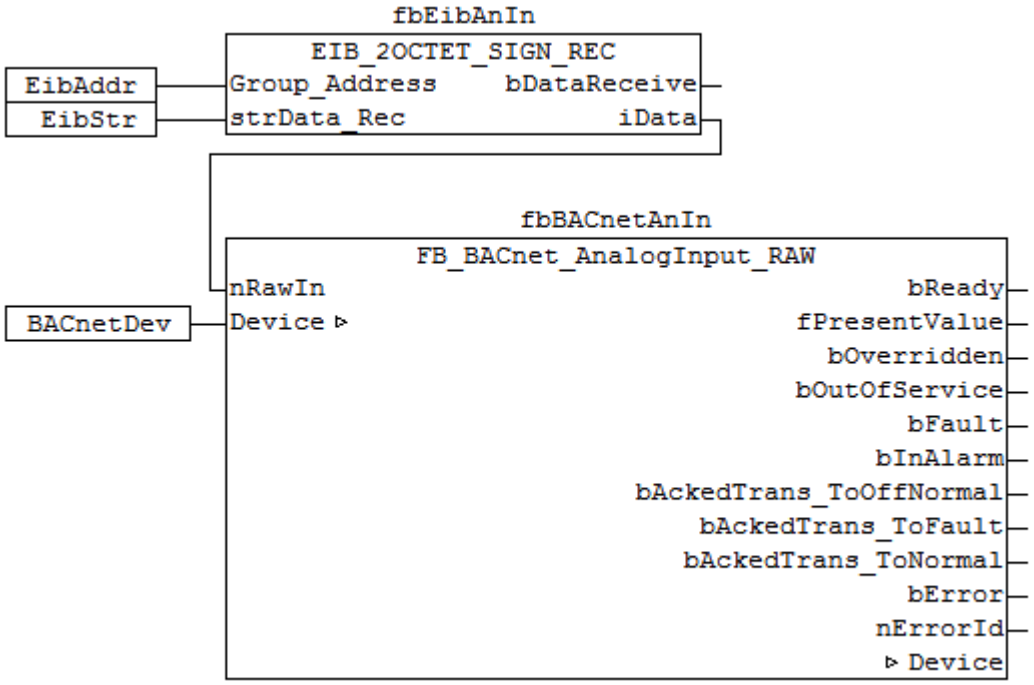


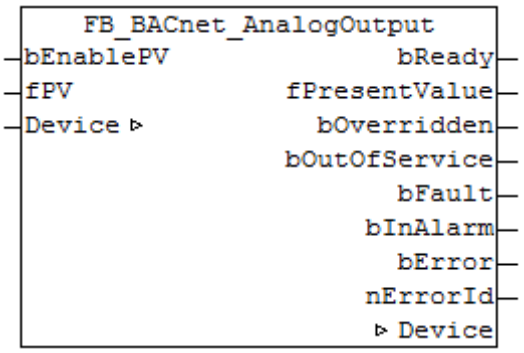
Fig. 90: Fig. 2: Example for the mapping from EIB to the Property Present_Value in PLC code.

This example requires the library "TcKL6301.lib". Refer to the manual of terminal KL6301 for more information about EIB.

5.3.4 FB_BACnet_AnalogOutput

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_AnalogOutput" can be used for reading and write access to a BACnet object of type *AnalogOutput* (AO) To this end the BACnet object was created under a local BACnet server.

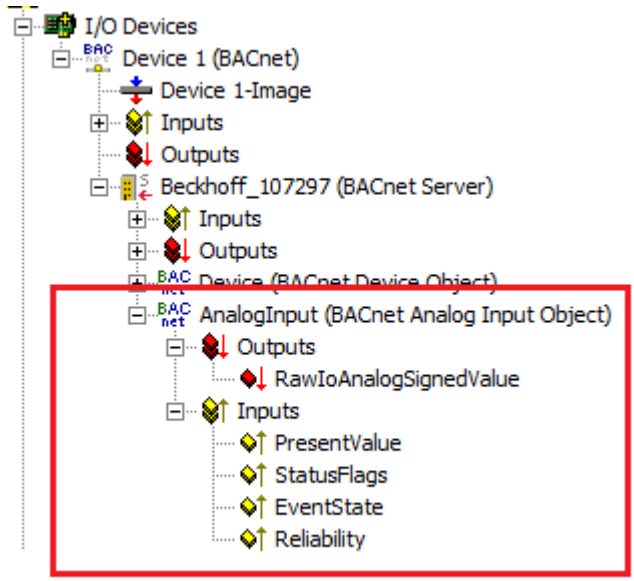


Fig. 91: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
fPV          : REAL;
```

bEnablePV: Enables the value of input "fPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "fPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).

fPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Process data are always written cyclically once enabled.

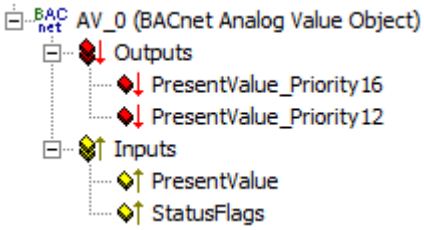


Fig. 92: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

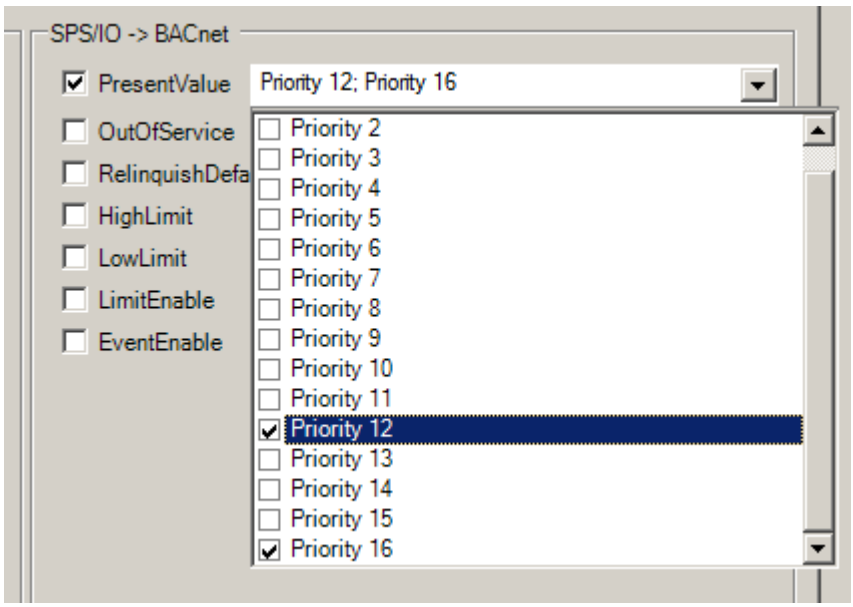


Fig. 93: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

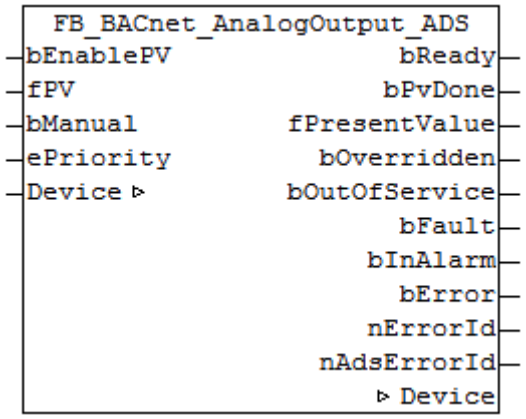
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.5 FB_BACnet_AnalogOutput_ADS

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_AnalogOutput_ADS" can be used for read and asynchronous write access to a BACnet object of type *AnalogOutput* (AO). To this end the BACnet object was created under a local BACnet server.

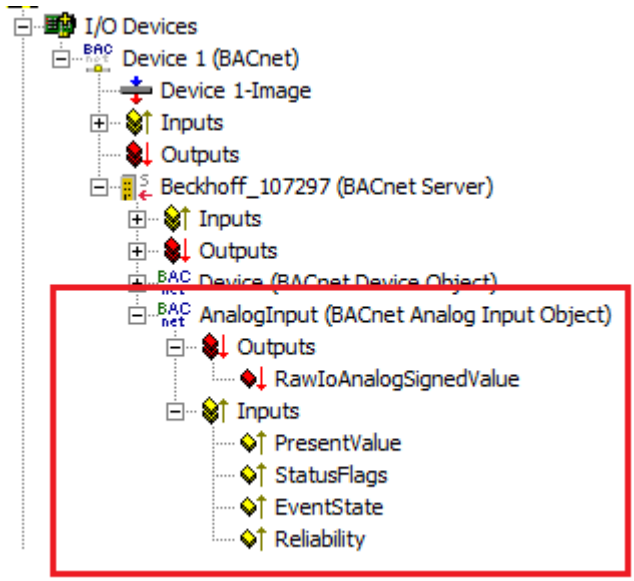


Fig. 94: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV           : BOOL;
fPV                 : REAL; (* decimal number for priority x, when input bEnablePV is TRUE *)
bManual             : BOOL; (* FALSE --> TRUE: write present_value immediately,
                FALSE: only write present_value on-change,
                TRUE: write present_value on-change and when device turns operational *)
ePriority           : E_BACNETPRIORITY := BACNETPRIORITY_12;
```

bEnablePV: enables the value of input "fPV". As soon as the input is set to *TRUE* there is a write access with the value from "fPV" to the commandable property *Present_Value* with priority from "ePriority" (default: 12, if the input "ePriority" is not connected).
If the input is set to *FALSE*, write access is made with the value *NULL* to the commandable property *Present_Value* with priority from "ePriority" (NULL write).

fPV: value to be written to the *Priority_Array* of the commandable property *Present_Value* when "bEnablePV" is set to *TRUE*. The priority is determined with the input "ePriority" (default: 12, if the input "ePriority" is not connected). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing to the object is executed via ADS when the value changes.

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* --> *TRUE*: triggers the writing of the property *Present_Value* immediately.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 414]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 508]).

VAR_OUTPUT

```
bReady       : BOOL;
bPvDone      : BOOL; (* present_value written over ADS *)
fPresentValue: REAL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
nAdsErrorId  : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPvDone: Positive edge when write access to property *Present_Value* was successfull.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the "FB_BACnet_Device")
4 = the object type of the BACnet object dosn't match the function block type (the object type will determined over the process data. --> check the process data mapping in the System Manager!)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

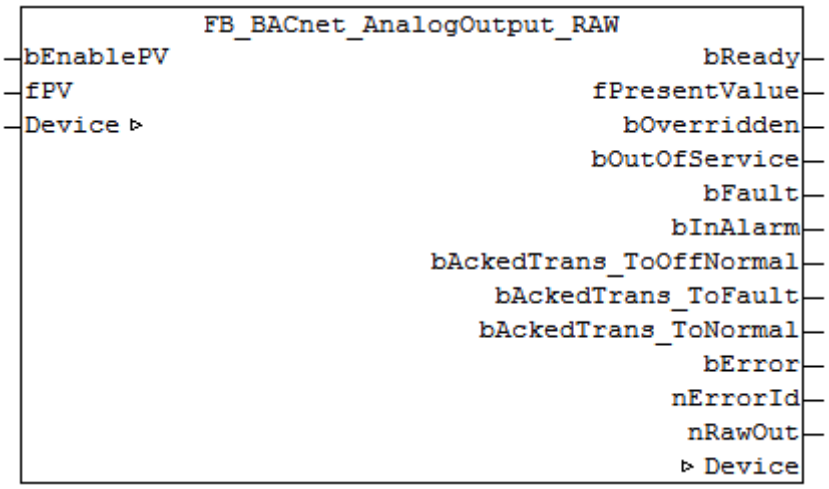
Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.6 FB_BACnet_AnalogOutput_RAW

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.

Contrary to the standard variant of the function block, the raw value is provided by a function block output and not by the terminal hardware. This allows e.g. Analog output information that is used in the PLC code displayed as an analog output object to BACnet (signal mapping for sub bus systems or virtual data points, etc.).



Use

The function block "FB_BACnet_AnalogOutput_RAW" can be used for read and write access to a BACnet object of type *AnalogOutput* (AO). To this end the BACnet object was created under a local BACnet server.

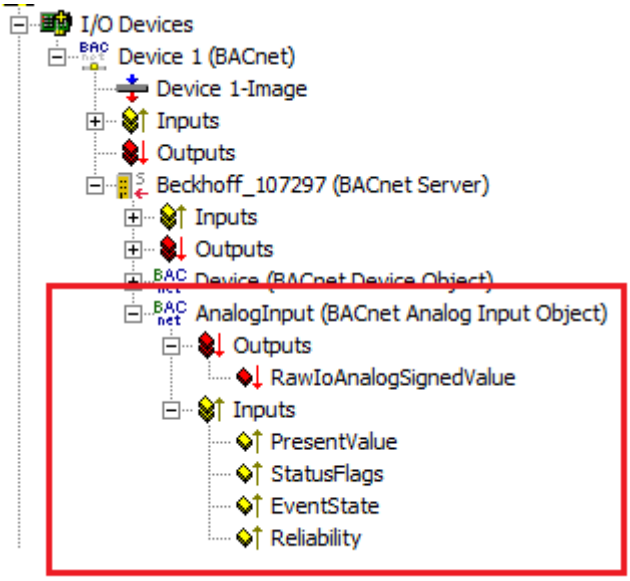


Fig. 95: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
fPV          : REAL;
```

bEnablePV: Enables the value of input "fPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "fPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).

fPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Process data are always written cyclically once enabled.

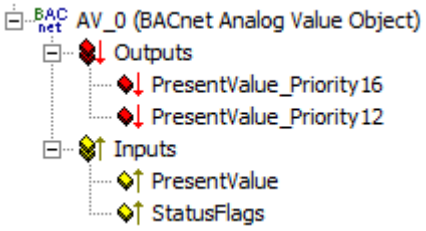


Fig. 96: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

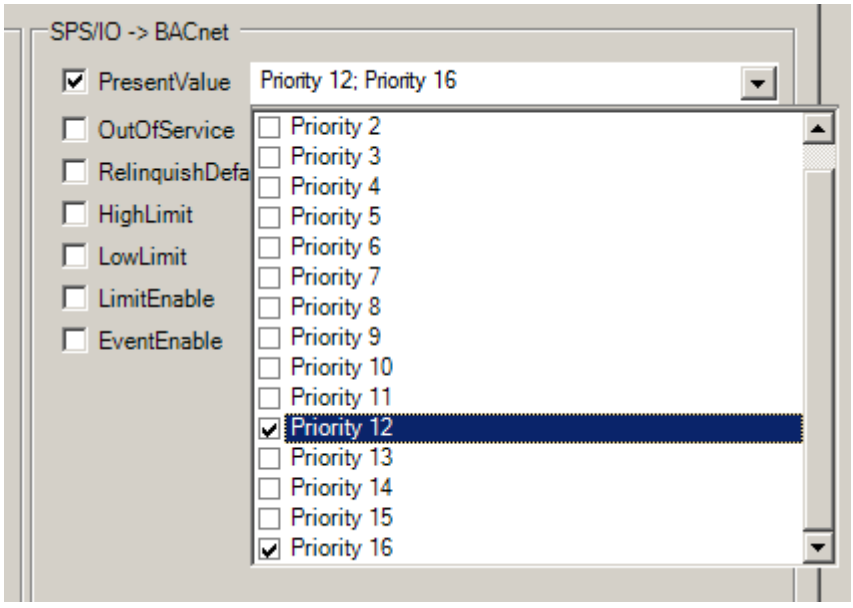


Fig. 97: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady                  : BOOL;
fPresentValue           : REAL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
nRawOut                 : INT; (* ~(BACnet_RawIoAnalogSignedValue : : ) *)
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nRawOut: Raw value output of the object in the range -32768 ... 32767. The output is linked to the process data "RawIoAnalogSignedValue" of the BACnet object. The value of "nRawOut" is the calculation result of property *Present_Value* and property *Resolution* (assuming the object state is not *out_of_service*).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

Example

The following example shows the mapping of the Property *Present_Value* of an analog output object to an EIB integer value:

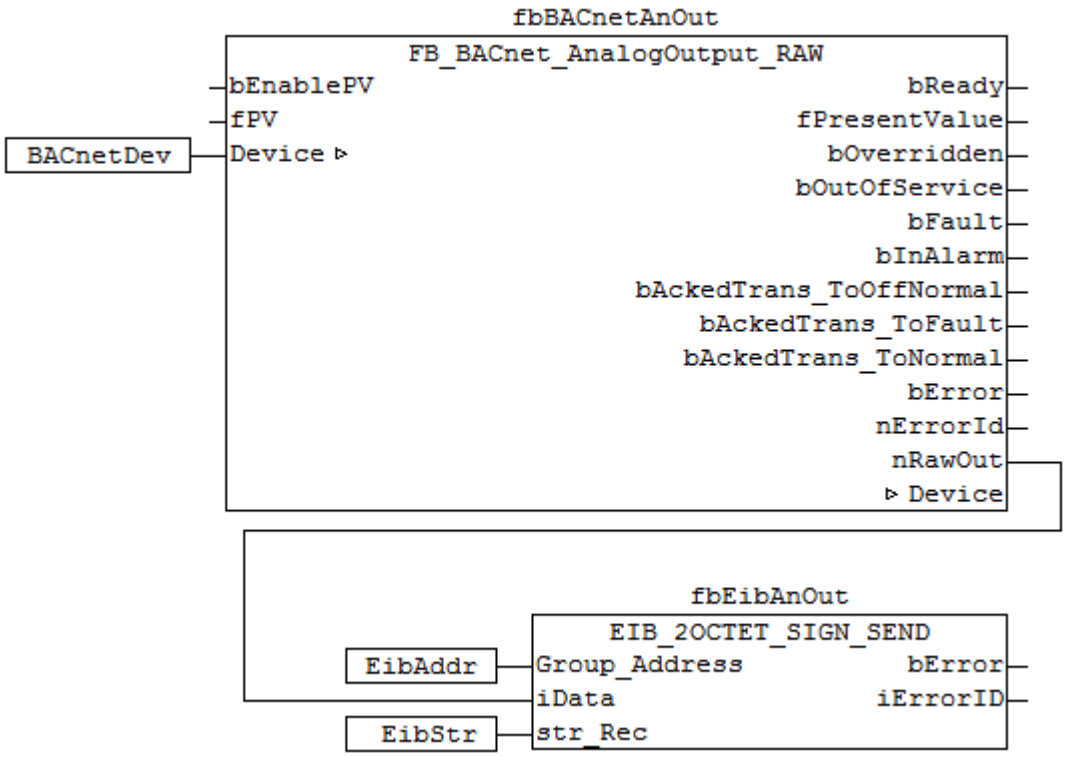


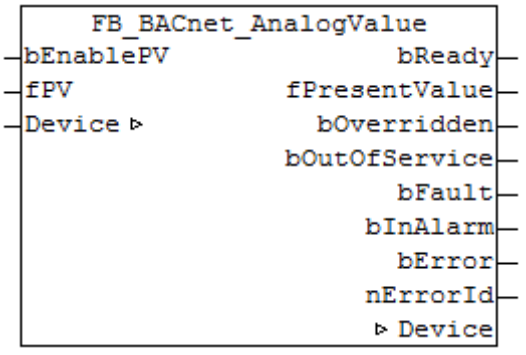
Fig. 98: Fig. 4: Example for the mapping of the property Present_Value to EIB in PLC code.

This example requires the library "TcKL6301.lib". Refer to the manual of terminal KL6301 for more information about EIB.

5.3.7 FB_BACnet_AnalogValue

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_AnalogValue" can be used for reading and write access to a BACnet object of type *AnalogValue* (AV). To this end the BACnet object was created under a local BACnet server.

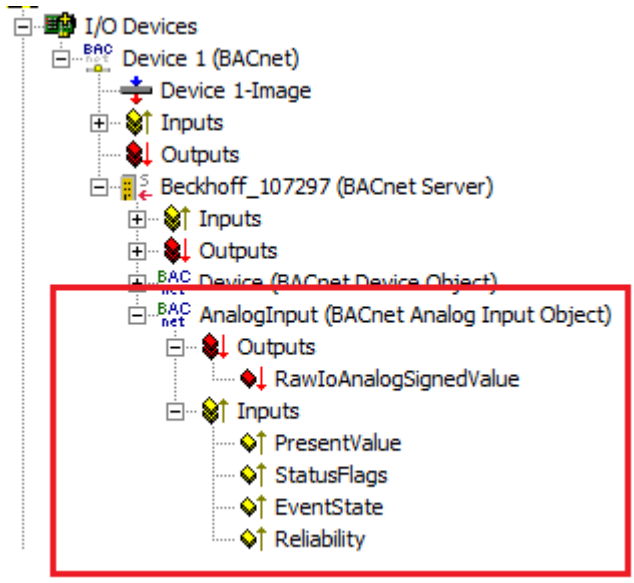


Fig. 99: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
fPV          : REAL;
```

bEnablePV: Enables the value of input "fPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "fPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

fPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input **bEnablePV** to *TRUE*. Process data are always written cyclically once enabled.

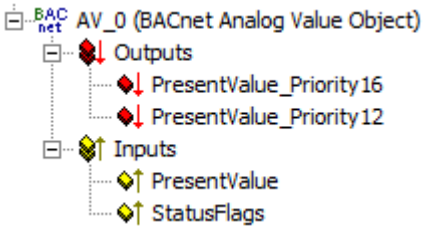


Fig. 100: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

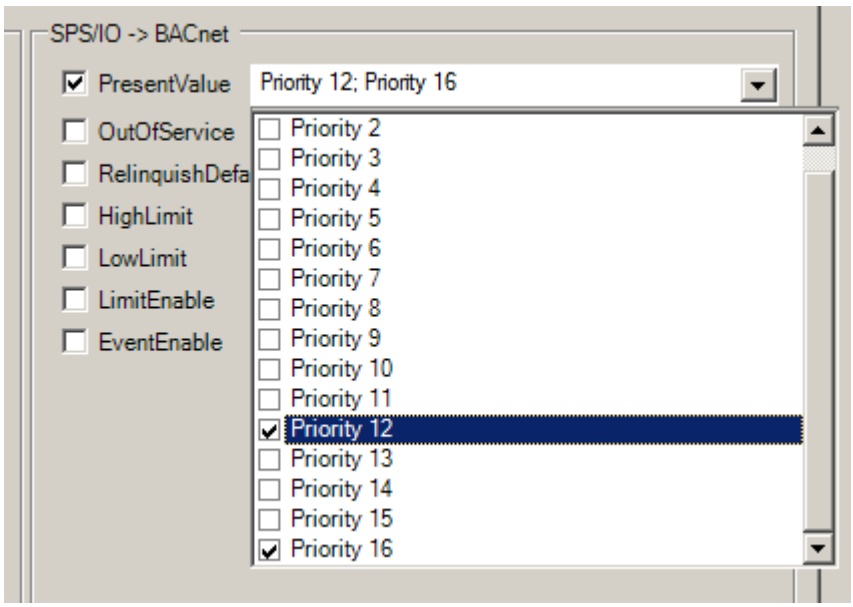


Fig. 101: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
fPresentValue   : REAL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bOverRange      : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogValue* and property *Present_Value*).

bOverriden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogValue* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

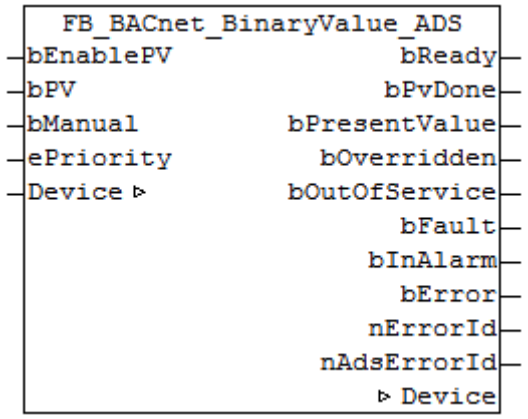
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.8 FB_BACnet_BinaryValue_ADS

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_BinaryValue_ADS" can be used for read and write access to a BACnet object of type *BinaryValue* (BV). To this end the BACnet object was created under a local BACnet server.

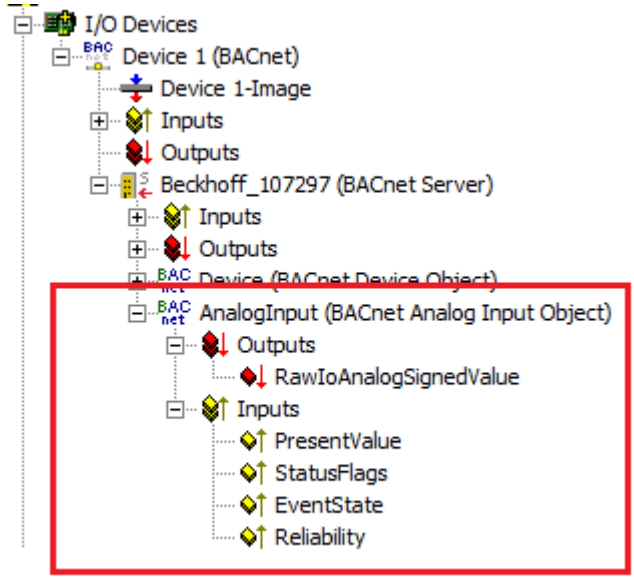


Fig. 102: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV           : BOOL;
bPV                 : BOOL; (* present value bool type *)
bManual             : BOOL; (* FALSE --> TRUE: write present_value immediately,
                FALSE: only write present_value on-change,
                TRUE: write present_value on-change and when device turns operational *)
ePriority           : E_BACNETPRIORITY := BACNETPRIORITY_12;
```

bEnablePV: enables the value of input "bPV". As soon as the input is set to *TRUE* there is a write access with the value from "bPV" to the commandable property *Present_Value* with priority from "ePriority" (default: 12, if the input "ePriority" is not connected).
If the input is set to *FALSE*, write access is made with the value *NULL* to the commandable property *Present_Value* with priority from "ePriority" (NULL write).

bPV: value to be written to the *Priority_Array* of the commandable property *Present_Value* when "bEnablePV" is set to *TRUE*. The priority is determined with the input "ePriority" (default: 12, if the input "ePriority" is not connected). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing to the object is executed via ADS when the value changes.

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* --> *TRUE*: triggers the writing of the property *Present_Value* immediately.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 414]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 508]).

VAR_OUTPUT

```
bReady       : BOOL;
bPvDone      : BOOL; (* present_value written over ADS *)
bPresentValue: BOOL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
nAdsErrorId  : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPvDone: Positive edge when write access to property *Present_Value* was successfull.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
4 = the object type of the BACnet object dosn't match the function block type (the object type will determined over the process data. --> check the process data mapping in the System Manager!)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nAdsErrorId: ADS error code.

VAR_IN_OUT

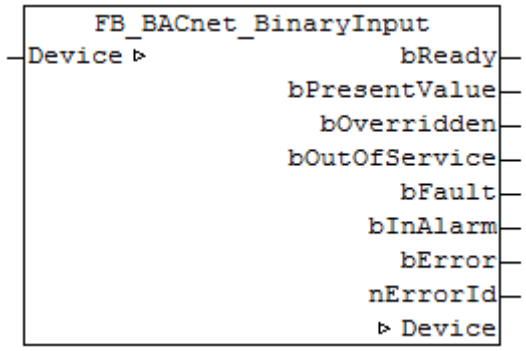
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.9 FB_BACnet_BinaryInput

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block FB_BACnet_BinaryInput can be used for reading access to a BACnet object of type *BinaryInput* (BI). To this end the BACnet object was created under a local BACnet server.

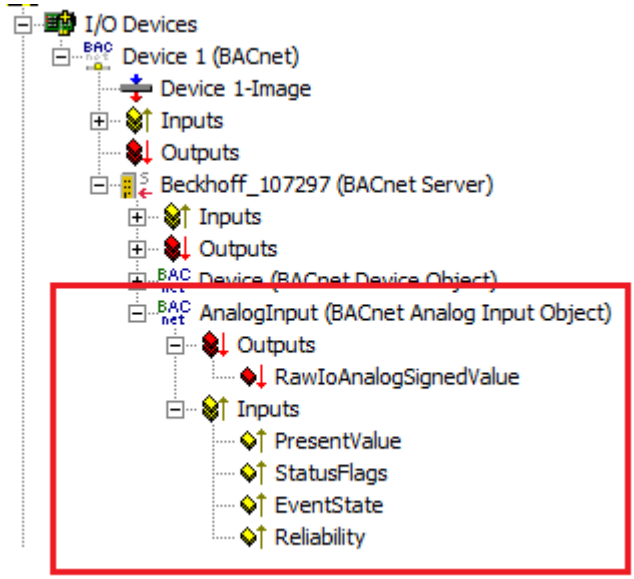


Fig. 103: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

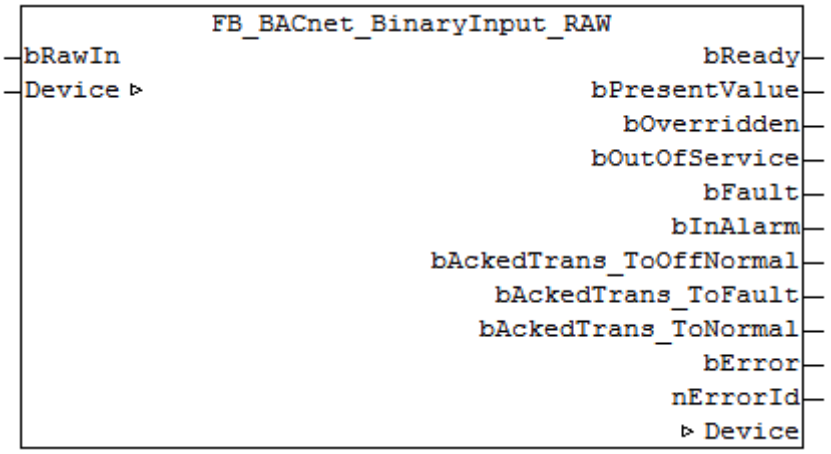
Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.10 FB_BACnet_BinaryInput_RAW

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.

Contrary to the standard variant of the function block, the raw value is provided by a function block input and not by the terminal hardware. This allows e.g. Digital input information that are generated in the PLC code displayed as an binary input object to BACnet (signal mapping for sub bus systems or virtual data points, etc.).



Use

The function block "FB_BACnet_BinaryInput_RAW" can be used for read access to a BACnet object of type *BinaryInput* (BI). To this end the BACnet object was created under a local BACnet server.

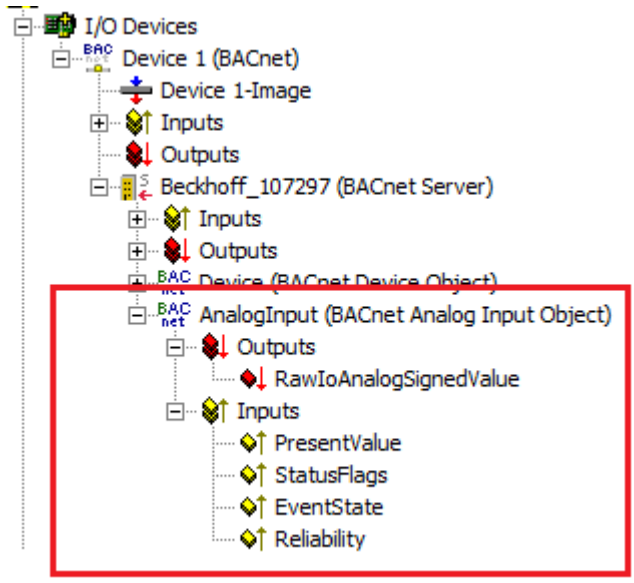


Fig. 104: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bRawIn       : BOOL; (* ~(BACnet_RawIoBinaryBoolValue : : ) *)
```

bRawIn: Raw value input of the object. The input is linked to the process data "RawIoBinaryBoolValue" of the BACnet object. The value of "bRawIn" is mapped to the property *Present_Value* (assuming the object state is not *out_of_service*).

VAR_OUTPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

Example

The following example shows the mapping of an EIB bit value to the property *Present_Value* of an binary input object:

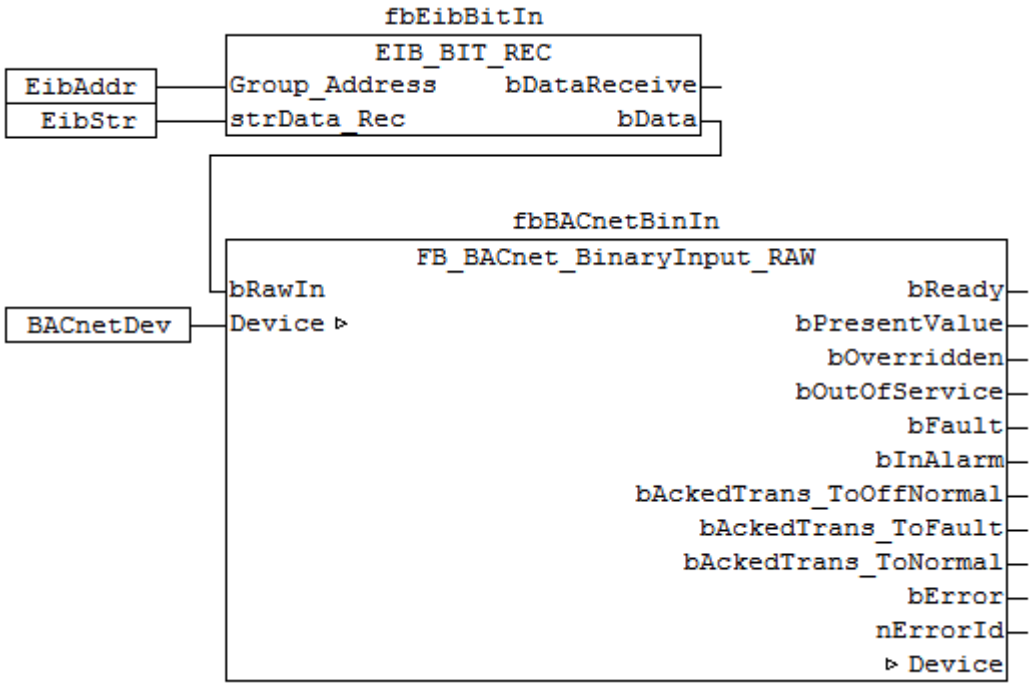


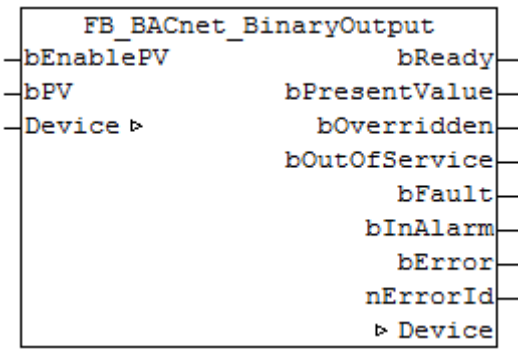
Fig. 105: Fig. 2: Example for the mapping of an EIB binary value to the Property Present_Value in PLC code.

This example requires the library "TcKL6301.lib". Refer to the manual of terminal KL6301 for more information about EIB.

5.3.11 FB_BACnet_BinaryOutput

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_BinaryOutput" can be used for read and write access to a BACnet object of type *BinaryOutput* (BO). To this end the BACnet object was created under a local BACnet server.

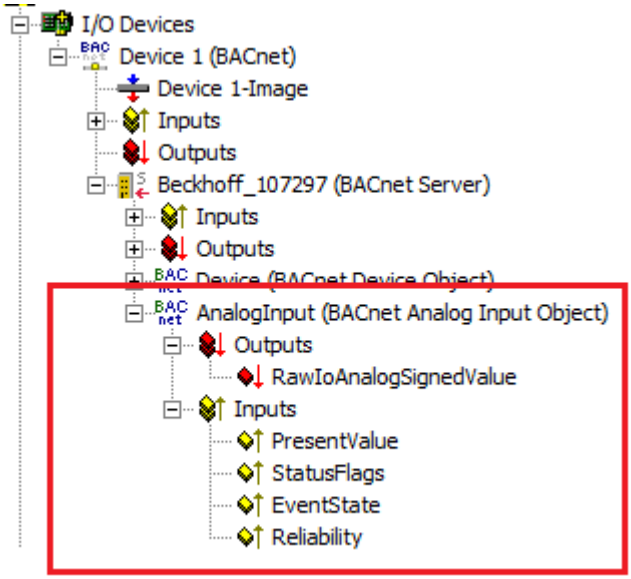


Fig. 106: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV     : BOOL;
bPV           : BOOL;
```

bEnablePV: Enables the value of input "bPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "bPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

bPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Process data are always written cyclically once enabled.

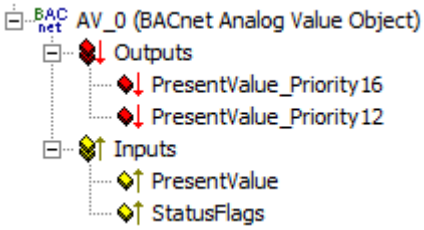


Fig. 107: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

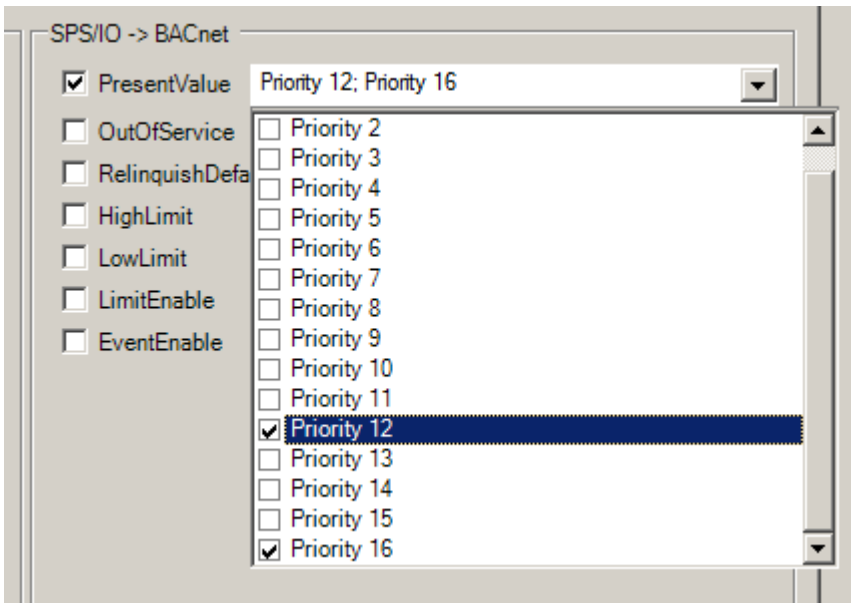


Fig. 108: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

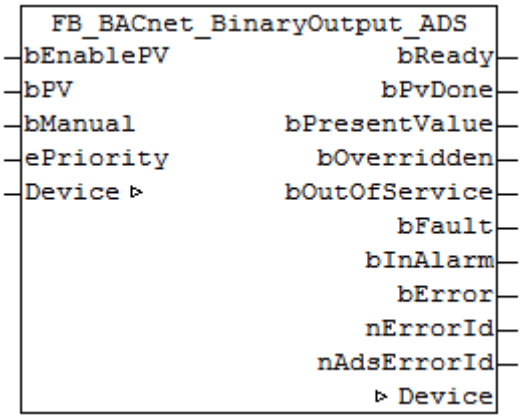
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.12 FB_BACnet_BinaryOutput_ADS

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_BinaryOutput_ADS" can be used for read and write access to a BACnet object of type *BinaryOutput* (BO). To this end the BACnet object was created under a local BACnet server.

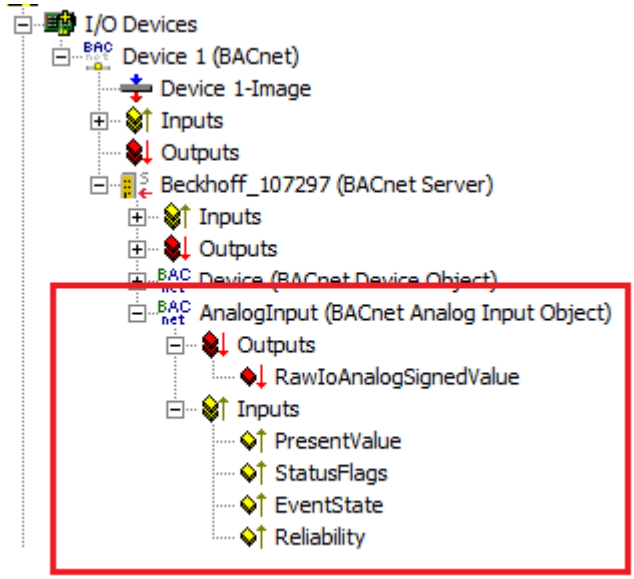


Fig. 109: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV           : BOOL;
bPV                 : BOOL; (* present value bool type *)
bManual             : BOOL; (* FALSE --> TRUE: write present_value immediately,
                FALSE: only write present_value on-change,
                TRUE: write present_value on-change and when device turns operational *)
ePriority           : E_BACNETPRIORITY := BACNETPRIORITY_12;
```

bEnablePV: enables the value of input "bPV". As soon as the input is set to *TRUE* there is a write access with the value from "bPV" to the commandable property *Present_Value* with priority from "ePriority" (default: 12, if the input "ePriority" is not connected).
If the input is set to *FALSE*, write access is made with the value *NULL* to the commandable property *Present_Value* with priority from "ePriority" (NULL write).

bPV: value to be written to the *Priority_Array* of the commandable property *Present_Value* when "bEnablePV" is set to *TRUE*. The priority is determined with the input "ePriority" (default: 12, if the input "ePriority" is not connected). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing to the object is executed via ADS when the value changes.

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* --> *TRUE*: triggers the writing of the property *Present_Value* immediately.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 414]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 508]).

VAR_OUTPUT

```
bReady           : BOOL;
bPvDone          : BOOL; (* present_value written over ADS *)
bPresentValue    : BOOL;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
nAdsErrorId      : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPvDone: Positive edge when write access to property *Present_Value* was successfull.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
4 = the object type of the BACnet object dosn't match the function block type (the object type will determined over the process data. --> check the process data mapping in the System Manager!)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nAdsErrorId: ADS error code.

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

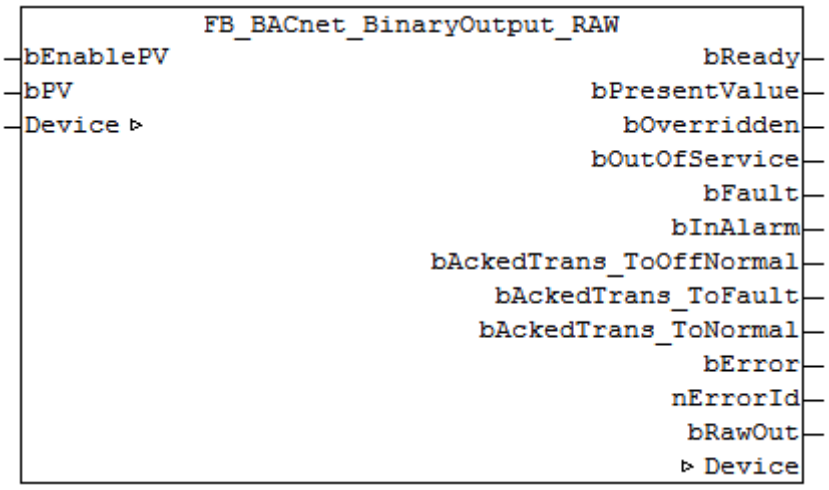
Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.13 FB_BACnet_BinaryOutput_RAW

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.

Contrary to the standard variant of the function block, the raw value is provided by a function block output and not by the terminal hardware. This allows e.g. Binary output information that is used in the PLC code displayed as an analog output object to BACnet (signal mapping for sub bus systems or virtual data points, etc.).



Use

The function block "FB_BACnet_BinaryOutput_RAW" can be used for read and write access to a BACnet object of type *BinaryOutput* (BO). To this end the BACnet object was created under a local BACnet server.

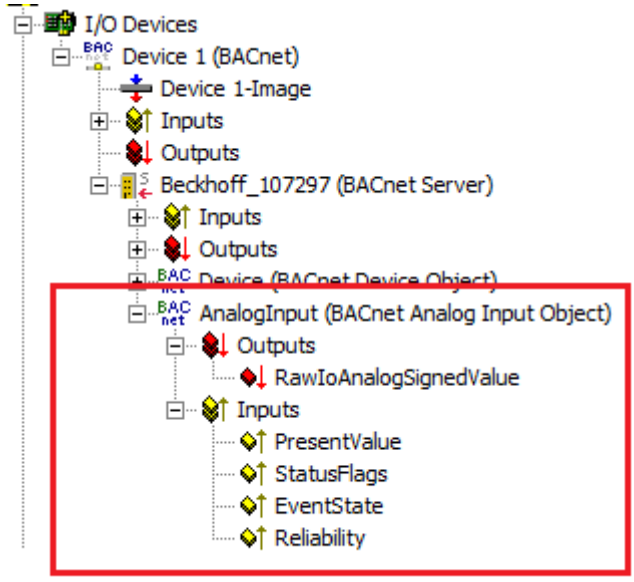


Fig. 110: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV     : BOOL;
bPV           : BOOL;
```

bEnablePV: Enables the value of input "bPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "bPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

bPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Process data are always written cyclically once enabled.

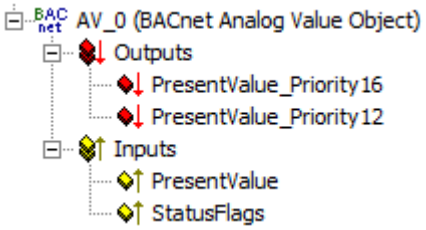


Fig. 111: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

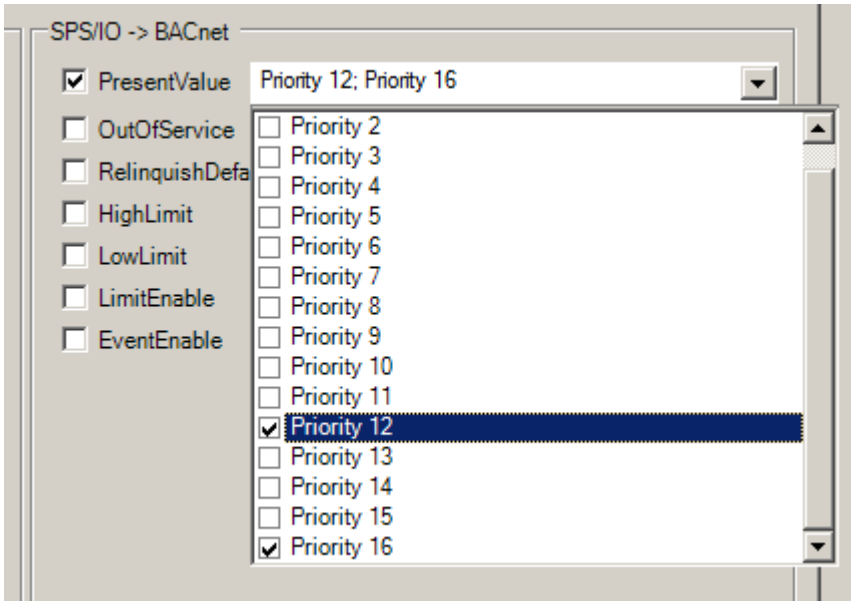


Fig. 112: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady                  : BOOL;
bPresentValue           : BOOL;
bOverridden             : BOOL;
bOutOfService           : BOOL;
bFault                  : BOOL;
bInAlarm                : BOOL;
bAckedTrans_ToOffNormal : BOOL;
bAckedTrans_ToFault     : BOOL;
bAckedTrans_ToNormal    : BOOL;
bError                  : BOOL;
nErrorId                : UINT;
bRawOut                 : BOOL; (* ~(BACnet_RawIoBinaryBoolValue : : ) *)
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bAckedTrans_ToOffNormal, bAckedTrans_ToFault, bAckedTrans_ToNormal: Flags of property *Acked_Transitions* (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Acked_Transitions*).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

bRawOut: Raw value output of the object. The output is linked to the process data "RawIoAnalogSignedValue" of the BACnet object. The value of "bRawOut" is the value of property *Present_Value* (assuming the object state is not *out_of_service*).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

Example

The following example shows the mapping of the Property *Present_Value* of an binary output object to an EIB bit value:

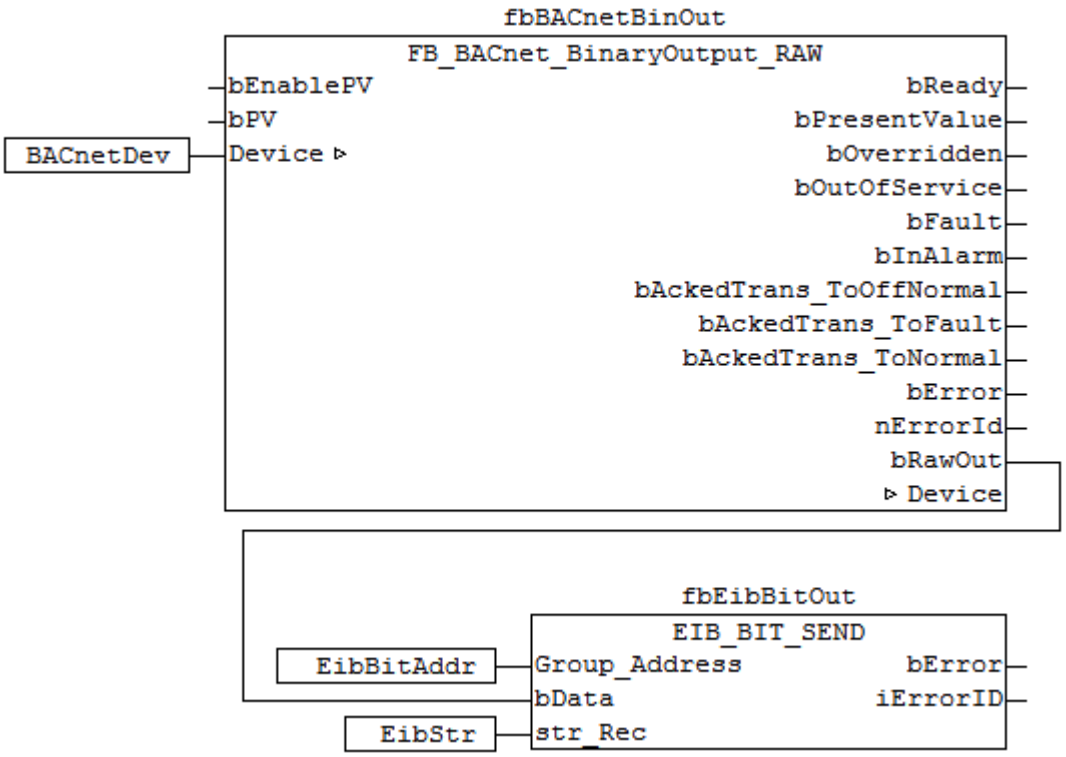


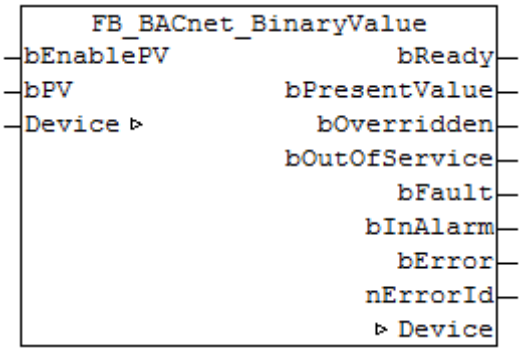
Fig. 113: Fig. 4: Example for the mapping of the property Present_Value to EIB in PLC code

This example requires the library "TcKL6301.lib". Refer to the manual of terminal KL6301 for more information about EIB.

5.3.14 FB_BACnet_BinaryValue

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_BinaryValue" can be used for read and write access to a BACnet object of type *BinaryValue* (BV). To this end the BACnet object was created under a local BACnet server.

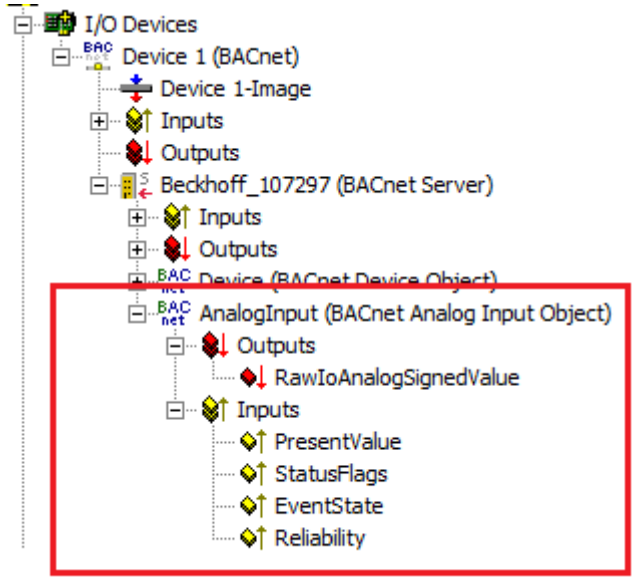


Fig. 114: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
bPV          : BOOL;
```

bEnablePV: Enables the value of input "bPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "bPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

bPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Process data are always written cyclically once enabled.

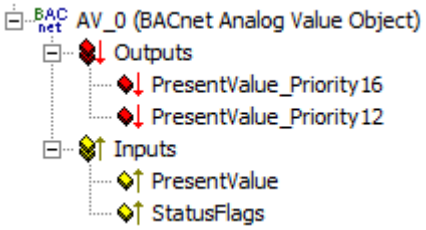


Fig. 115: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

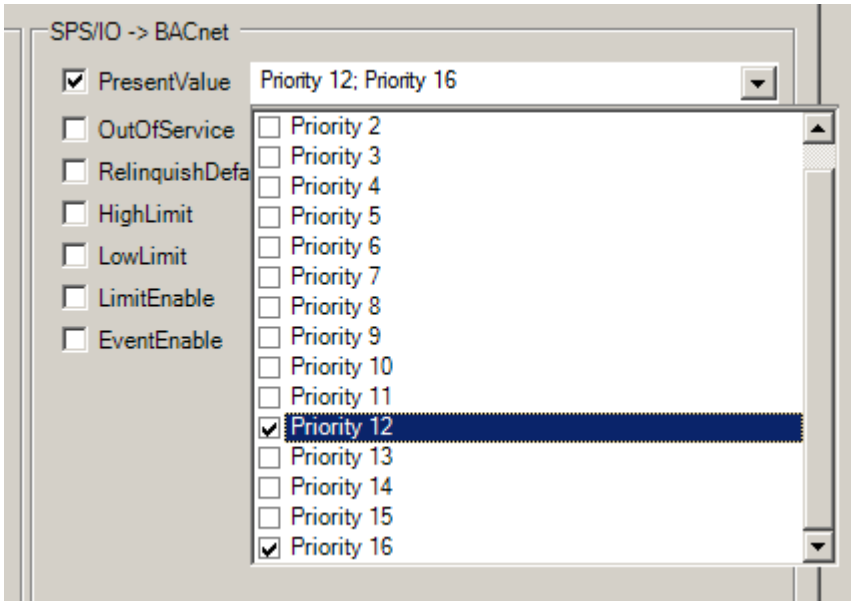


Fig. 116: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile

complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

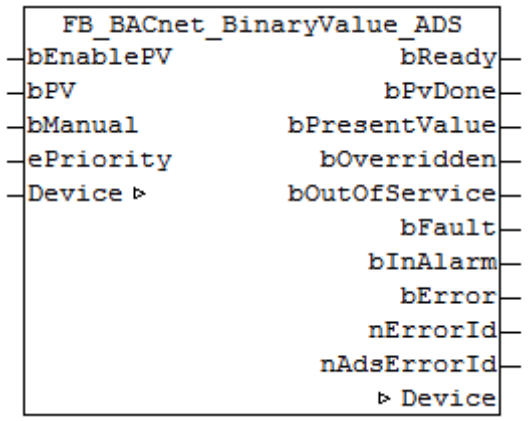
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.15 FB_BACnet_BinaryValue_ADS

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_BinaryValue_ADS" can be used for read and write access to a BACnet object of type *BinaryValue* (BV). To this end the BACnet object was created under a local BACnet server.

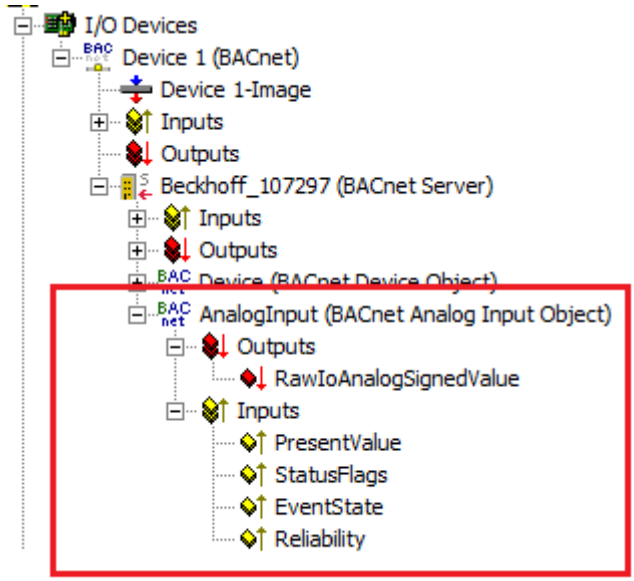


Fig. 117: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV            : BOOL;
bPV                  : BOOL; (* present value bool type *)
bManual              : BOOL; (* FALSE --> TRUE: write present_value immediately,
                FALSE: only write present_value on-change,
                TRUE: write present_value on-change and when device turns operational *)
ePriority            : E_BACNETPRIORITY := BACNETPRIORITY_12;
```

bEnablePV: enables the value of input "bPV". As soon as the input is set to *TRUE* there is a write access with the value from "bPV" to the commandable property *Present_Value* with priority from "ePriority" (default: 12, if the input "ePriority" is not connected).
If the input is set to *FALSE*, write access is made with the value *NULL* to the commandable property *Present_Value* with priority from "ePriority" (NULL write).

bPV: value to be written to the *Priority_Array* of the commandable property *Present_Value* when "bEnablePV" is set to *TRUE*. The priority is determined with the input "ePriority" (default: 12, if the input "ePriority" is not connected). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing to the object is executed via ADS when the value changes.

Since writing of the property *Present_Value* takes place acyclically, and only when the value changes, it is potentially possible that the property *Present_Value* is also overwritten with the same priority by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE* --> *TRUE*: triggers the writing of the property *Present_Value* immediately.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 414]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 508]).

VAR_OUTPUT

```
bReady           : BOOL;
bPvDone          : BOOL; (* present_value written over ADS *)
bPresentValue    : BOOL;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
nAdsErrorId      : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPvDone: Positive edge when write access to property *Present_Value* was successfull.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)

4 = the object type of the BACnet object dosn't match the function block type (the object type will determined over the process data. --> check the process data mapping in the System Manager!)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nAdsErrorId: ADS error code.

VAR_IN_OUT

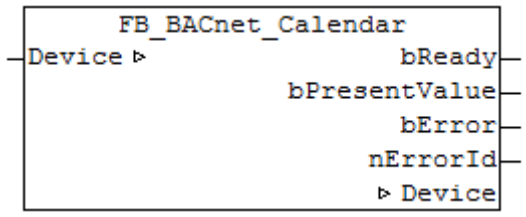
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.16 FB_BACnet_Calendar

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Calendar" can be used for read access to a BACnet object of type *Calendar* (CAL). To this end the BACnet object was created under a local BACnet server.

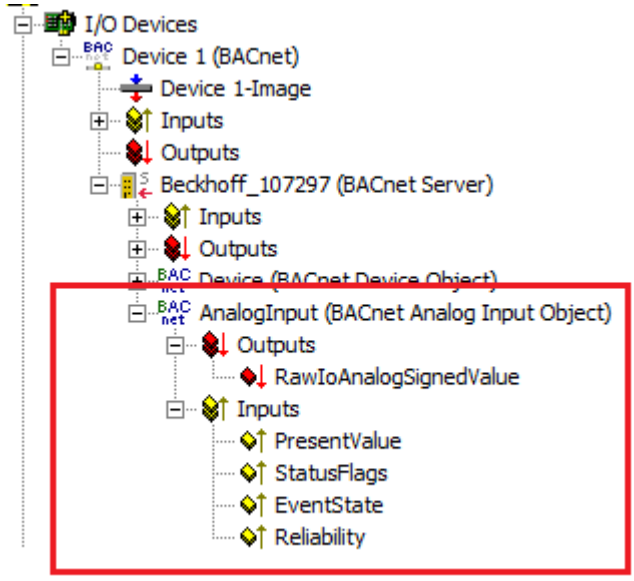


Fig. 118: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue). If the output is *FALSE*, the associated function block "FB_BACnet_Device" reports "not operational".

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Calendar* and property *Present_Value*).

bError: An error is pending.

nErrorId: Error number
0 = no error
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

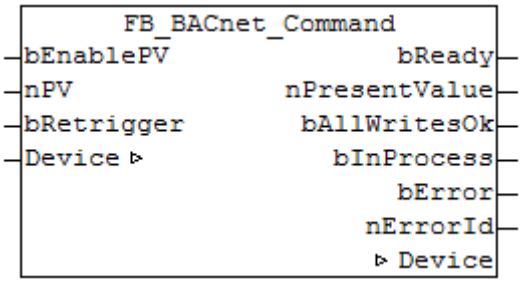
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.17 FB_BACnet_Command

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Command" can be used for read and write access to a BACnet object of type *Command* (CMD). To this end the BACnet object was created under a local BACnet server.

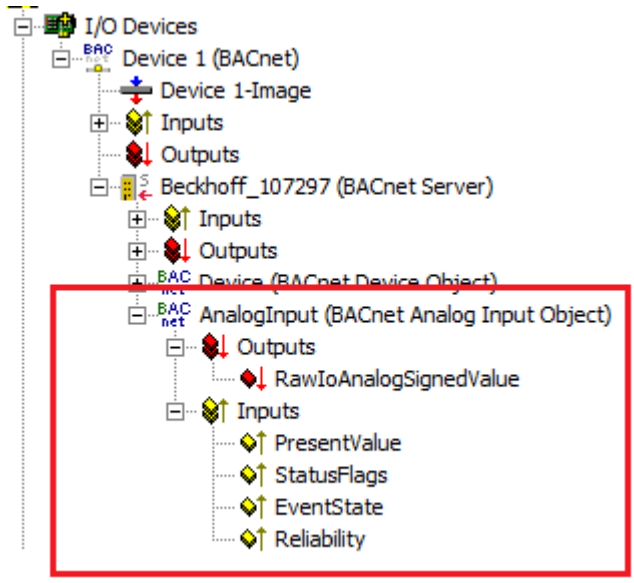


Fig. 119: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
nPV          : UDINT;
bRetrigger   : BOOL;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, writing takes place into the property *Present_Value* of the corresponding BACnet object with the value of input **nPV**.
If **bEnablePV** is set to *FALSE*, the process data of the mapped property *Present_Value* are written to *16#FFFFFFFF* and thus deactivated.

nPV: Value of the property *Present_Value* to be written. If the value is outside the value range (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*), the corresponding process data is disabled, i.e. writing to the property is disabled and other BACnet services have write access to the property *Present_Value*.
Writing of a valid value triggers execution of the corresponding command list of the BACnet object. The value of the property Present_Value is written whenever the process data changes (i.e. change in the value of **nPV** with **bEnablePV** set or signal change from *FALSE* to *TRUE* at input **bRetrigger** with **bEnablePV** set).

bRetrigger: A rising edge at this input triggers a repeat of the write process and therefore execution of the corresponding commands of the BACnet object *Command*. The signal change from *FALSE* to *TRUE* corresponds to a change of the process data of the property *Present_Value* from x to 0 to x.

VAR_OUTPUT

```
bReady       : BOOL;
nPresentValue: UDINT;
bAllWritesOk : BOOL;
bInProcess   : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, AllWritesOk, InProcess). If the output is *FALSE*, the associated function block "FB_BACnet_Device" reports "not operational".

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*).

bAllWritesOk: The last requested command list was successfully processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *All_Writes_Successful*).

bInProcess: The selected command list is processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *In_Process*).

bError: An error is pending.

nErrorId: Error number
0 = no error
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.18 FB_BACnet_Device

The following function block is used for linking the PLC program to a local BACnet device object (server). The linking of the function block takes place with the aid of process data.

The process data can be linked manually or automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block. The process data for the BACnet device object also include the required process data for linking the BACnet server.

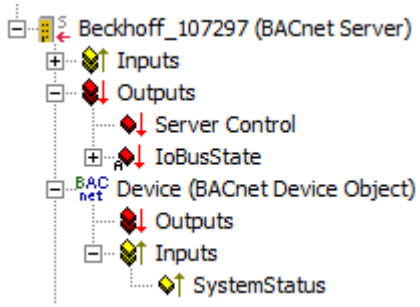


Fig. 120: Fig. 1: Process data of the BACnet device object and server in the System Manager.

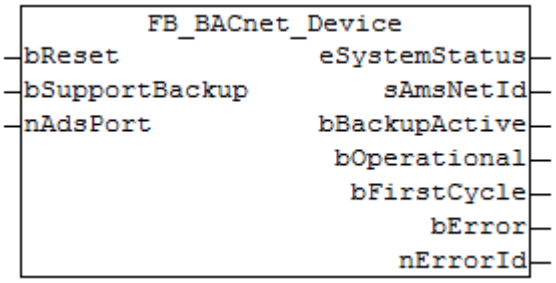


Fig. 121: Fig. 2: Function block of the BACnet device object and server in the PLC program.

Use

The function block "FB_BACnet_Device" is used to read the state of the local BACnet device object (System_Status) and output it in the PLC program. In addition, the process data "Server Control" is used to control the local BACnet server (activation and handling of the PLC backup).

The function block "FB_BACnet_Device" also requires a global instance of the function block FB_BACnet_Adapter [▶ 375] for each PLC project. The function block FB_BACnet_Adapter [▶ 375] establishes the connection between PLC and BACnet adapter in the System Manager. This global instance is already provided by the PLC library. The hierarchy between FB_BACnet_Adapter [▶ 375], "FB_BACnet_Device" and FB_BACnet_RemoteDevice [▶ 459] and an object of type *AnalogValue* (AV) is shown below:

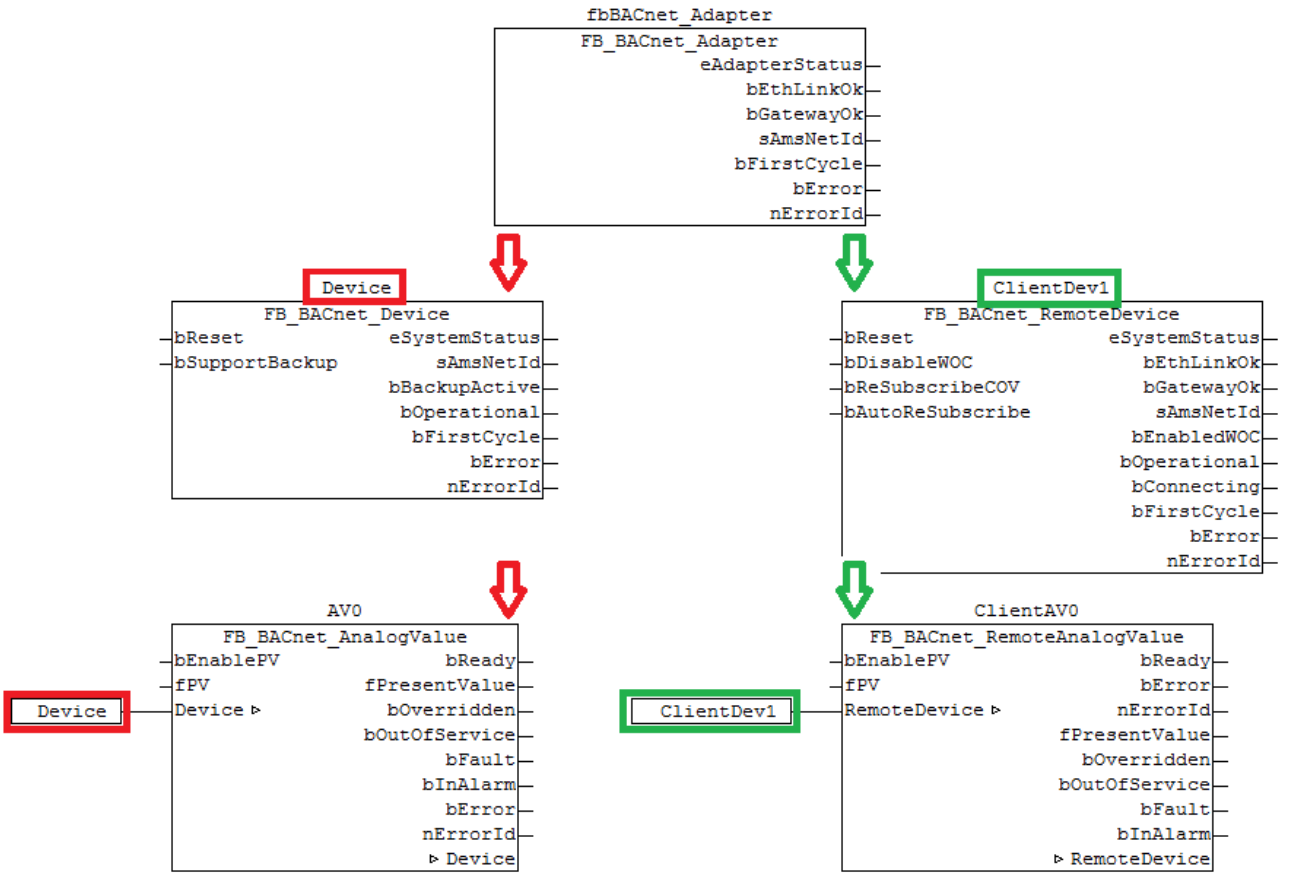


Fig. 122: Figure 3: Example for linking the FB instances in the PLC.

For information on the use of the function blocks FB_BACnet_Adapter [▶ 375] and _RemoteDevice [▶ 459], see FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] respectively.

VAR_INPUT

```
bReset           : BOOL;
bSupportBackup   : BOOL;
nAdsPort         : UINT := 1000; (* Optional setting: Ads port of local BACnet-
Server, default = 1000 *)
```

bReset: Resets the error state in the event of a signal change *FALSE* --> *TRUE*.

bSupportBackup: If the input is set to *TRUE*, saving of the persistent PLC runtime data via BACnet is supported. If this mode is activated, the persistent PLC data (by default under: "C:\TwinCAT\Boot\") have to be created as BACnet file objects and associated with the value "TwinCAT Configuration File" of the property *File_Type* (in this way the BACnet stack ensures that the files are saved).
If BACnet-based backup of the persistent PLC data is used, no further instance of the block *FB_WritePersistentData* should be used in the PLC runtime (*FB_WritePersistentData* is executed by *FB_BACnet_Device*)

nAdsPort: ADS port of the local BACnet server. The default number is 1000 (provided the BACnet server is the first element under the BACnet adapter in the System Manager). The port number can be read in the System Manager (see Fig. 4):

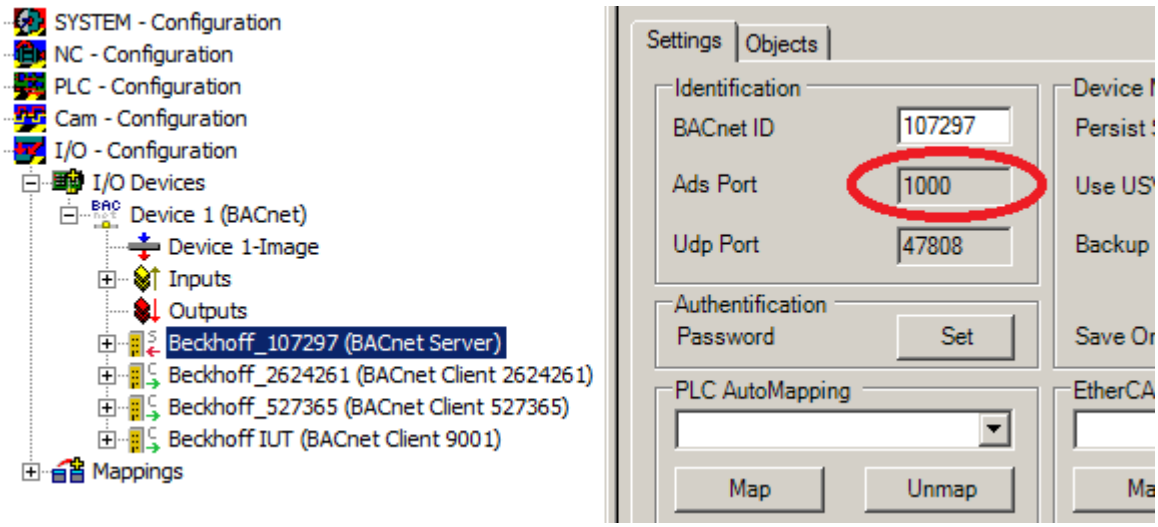


Fig. 123: Fig. 4: ADS port in the System Manager

VAR_OUTPUT

```
eSystemStatus    : E_BACnetDeviceStatus;
sAmsNetId        : T_AmsNetId;
bBackupActive    : BOOL;
bOperational     : BOOL;
bFirstCycle      : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

eSystemStatus: Status of the BACnet server object (see BACnet specification DIN EN ISO 16484-5 for BACnet device object and property *System_Status*).
0: BACnetDeviceStatus_Operational
1: BACnetDeviceStatus_OperationalReadOnly (only read access to properties)
2: BACnetDeviceStatus_DownloadRequired (loading of the configuration required)
3: BACnetDeviceStatus_DownloadInProgress (configuration download in progress)
4: BACnetDeviceStatus_NonOperational
5: BACnetDeviceStatus_BackupInProgress

sAmsNetId: Output of the AMS NetID of the local BACnet adapter (can be used for asynchronous access to BACnet objects via ADS).

bBackupActive: Feedback to indicate that a backup via BACnet is executed if the output is set to *TRUE*.

bOperational: BACnet device object (of the local server) reports "operational" and the device adapter reports the status "GetGatewayMAC" (0x03xx) or "WaitGatewayMAC" (0x04xx) or "Complete" (0x08xx). If the output becomes *FALSE*, all linked objects are blocked (block instances).

bFirstCycle: Is set for one cycle when the block instance is first called after a PLC reset or restart.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = no valid AMS NetID
20 = error during writing of the persistent data
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Device*.**nERR_xxx**). All errors with error number equal or greater than 20 have to be acknowledged with **bReset**.

An instance of function block FB_BACnet_Device is required whenever access to objects of a local BACnet server from the PLC is required. In principle, several instances of the function block FB_BACnet_Device can be created, although this would result in an error message in the System Manager when PLC automapping is used. This is due to the BACnet architecture. Under a BACnet adapter (network device) only one BACnet server can be created at a time. If PLC automapping is not used, any number of BACnet adapters and servers can be used in a PLC project. In this case the instances are linked manually.

The function block FB_BACnet_Device accesses the global variable "pBACnet_Adapter" internally. It contains a reference to the global instance of the function block FB_BACnet_Adapter. If several instances of the function block FB_BACnet_Device are to be created and manually linked in the System Manager with the corresponding BACnet servers, several instances of the function block FB_BACnet_Adapter have to be created. Before the instance of the BACnet server block is called, it is essential to call the corresponding instance of the BACnet adapter function block. A CFC example is provided below (Please note the order in which the function blocks are called!):

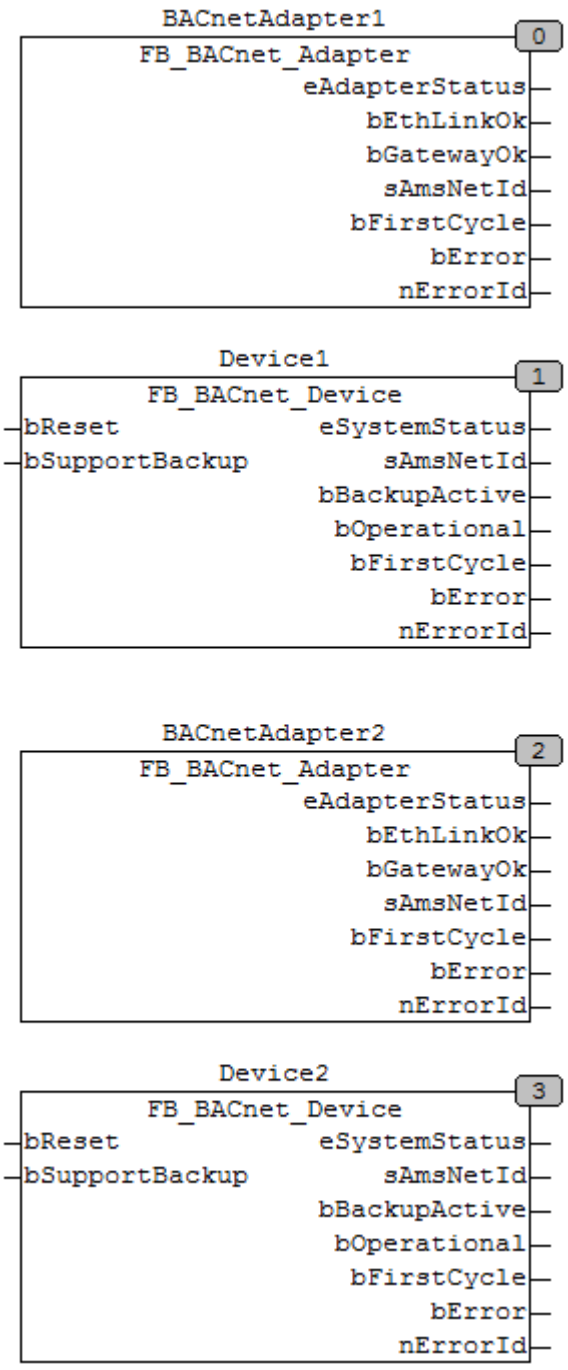


Fig. 124: Figure 5: Example for using several adapter and server instances in a PLC project.

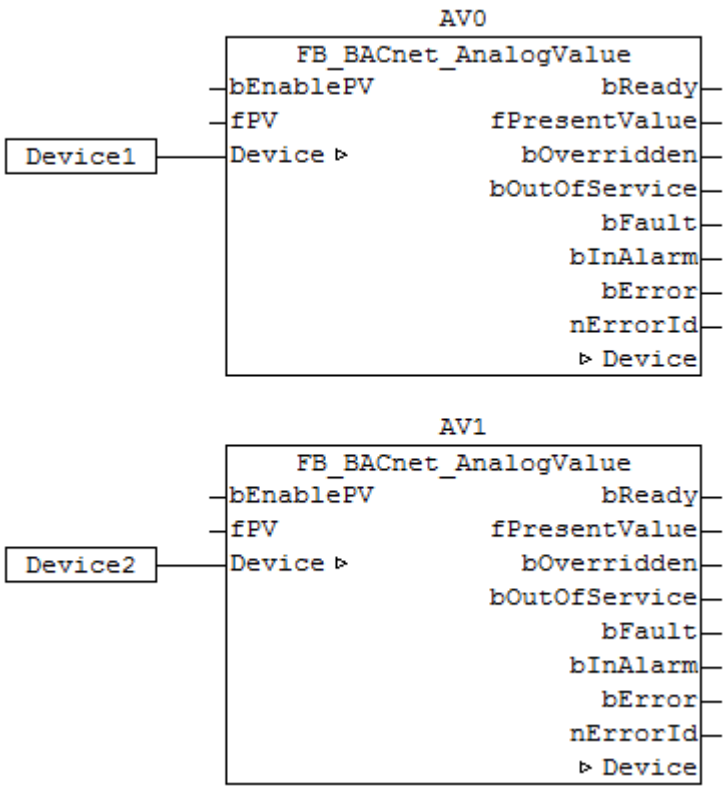


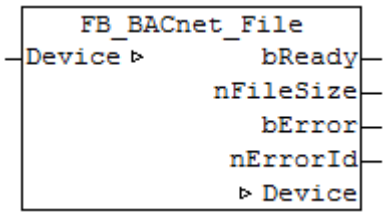
Fig. 125: Figure 6: Example for calling function blocks of type FB_BACnet_AnalogValue with different local BACnet servers in a PLC project.

The use of several local BACnet servers (adapters) in a PLC project is a special case and is not recommended. It could make sense (e.g. in cases where several network cards and therefore separate BACnet networks are used) to use several PLC projects and therefore several PLC runtimes. Data exchange between the individual runtimes can be realized via process data mapping or ADS communication. In this case PLC automapping could be used for BACnet objects.

5.3.19 FB_BACnet_File

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_File" can be used for read access to a BACnet object of type *File* (FILE). To this end the BACnet object was created under a local BACnet server.

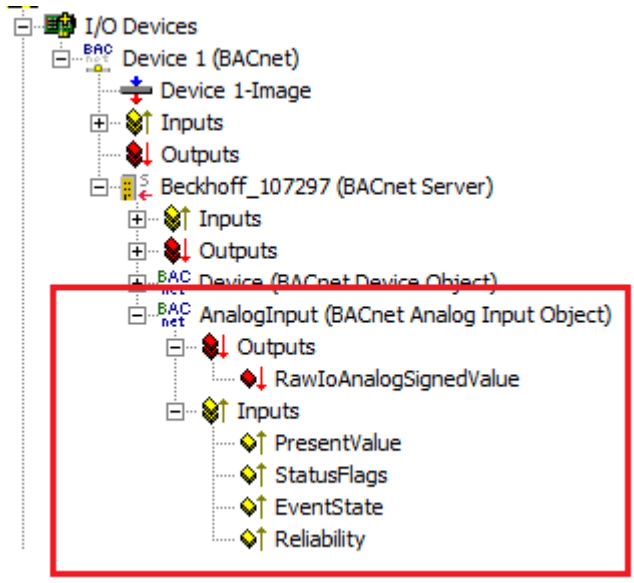


Fig. 126: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady          : BOOL;
bPresentValue   : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue). If the output is *FALSE*, the associated function block "FB_BACnet_Device" reports "not operational".

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *File* and property *Present_Value*).

bError: An error is pending.

nErrorId: Error number
0 = no error
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

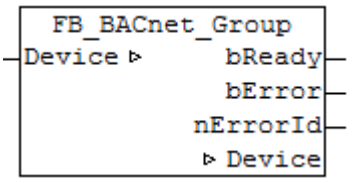
```
Device        : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.20 FB_BACnet_Group

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Group" serves as placeholder for future functions.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If the output is *FALSE*, the associated function block FB_BACnet_Device [▶ 414] reports "not operational".

bError: An error is pending.

nErrorId: Error number
0 = no error
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device [▶ 414])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

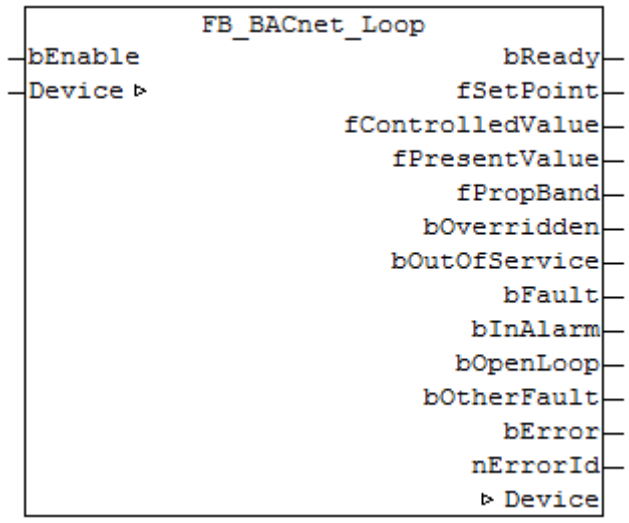
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.21 FB_BACnet_Loop

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Loop" provides the object functionality for the BACnet stack. To this end the BACnet object was created under a local BACnet server. The BACnet object on the BACnet stack side (System Manager) serves as interface to the BACnet world. The control and therefore functionality of the BACnet object is provided by the PLC instance of the block "FB_BACnet_Loop".

This approach has the advantage that existing, customer-specific controllers from a PLC project can easily be linked with the BACnet world. Only the process data of the BACnet-Loop object have to be provided and processed.

The function block "FB_BACnet_Loop" of the TcBACnet-Lib contains a standard PID controller (FB_BACnet_PidControl [▶ 492]) and therefore covers a wide range of control tasks.

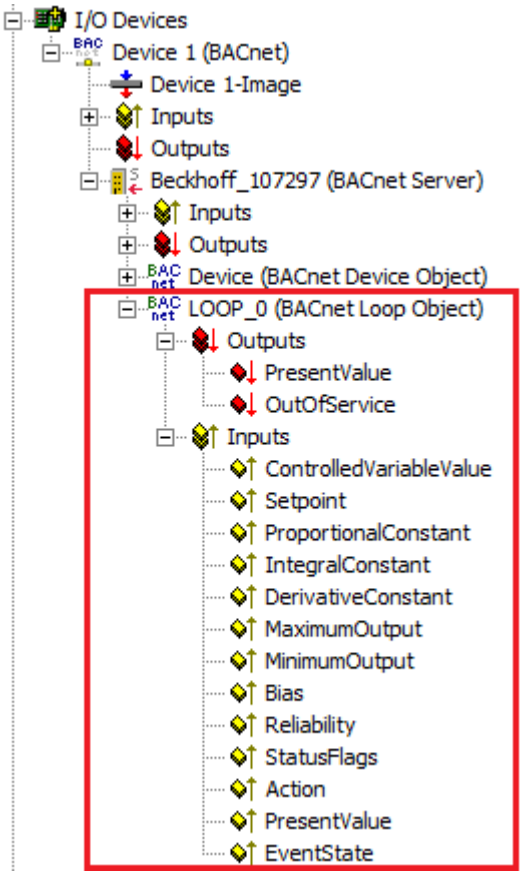


Fig. 127: Fig. 1: Example of a BACnet-LOOP object under a local BACnet server.

VAR_INPUT

```
bEnable      : BOOL;
```

bEnable: Enables the control. *FALSE* at the input sets the state of BACnet object to *Out_Of_Service* (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Out_Of_Service*).

VAR_OUTPUT

```
bReady           : BOOL;
fSetPoint        : REAL;
fControlledValue : REAL;
fPresentValue    : REAL;
fPropBand        : REAL;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bOpenLoop        : BOOL;
bOtherFault      : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

fSetPoint: Feedback of the controller specification (W, set value).

fControlledValue: Feedback of the current process parameter (X, actual value).

fPresentValue: Feedback of the current control output (Y, control value). Attention: *Present_Value* and *Controller_Variable_Value* can easily lead to confusion (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and properties *Present_Value*, *Controlled_Variable_Value* and *Controlled_Variable_Reference*).

fPropBand: Feedback of the current control output in percent (-100%...+100%) in relation to the minimum and maximum control output (properties *Minimum_Output* and *Maximum_Output*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Status_Flags*.

bOpenLoop, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Reliability*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device [▶ 414])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

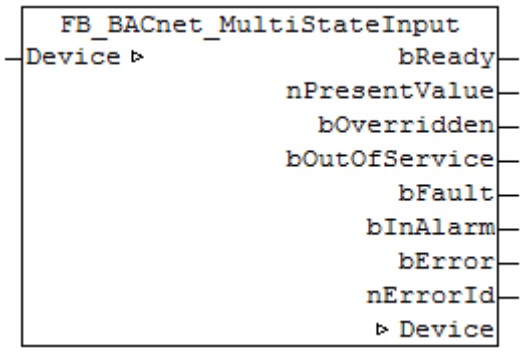
Controller configuration

The controller is configured using the following BACnet properties: *Action*, *Proportional_Constant* (P-factor), *Integral_Constant* (I-factor), *Derivative_Constant* (D-factor), Bias (output offset), *Maximum_Output* (maximum control output) and *Minimum_Output* (minimum control output). See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop*.

5.3.22 FB_BACnet_MultiStateInput

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_MultiStateInput" can be used for read access to a BACnet object of type *MultiStateInput* (MI). To this end the BACnet object was created under a local BACnet server.

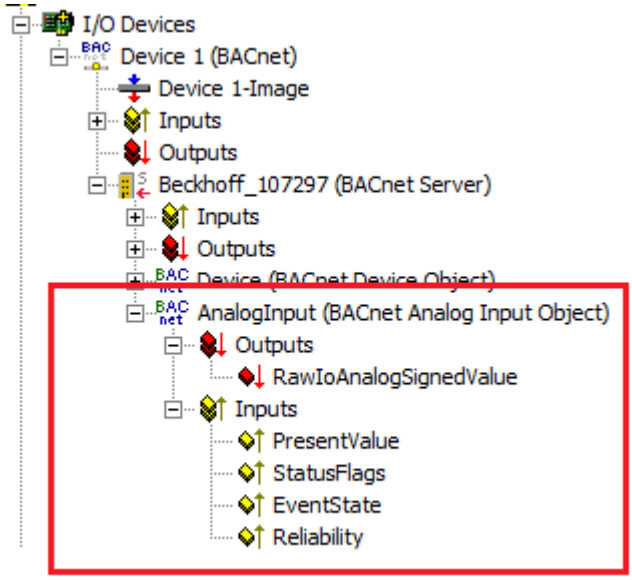


Fig. 128: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

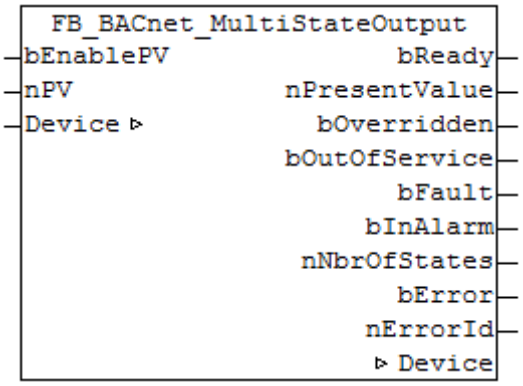
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.23 FB_BACnet_MultiStateOutput

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_MultiStateOutput" can be used for reading and write access to a BACnet object of type *MultiStateOutput* (MO). To this end the BACnet object was created under a local BACnet server.

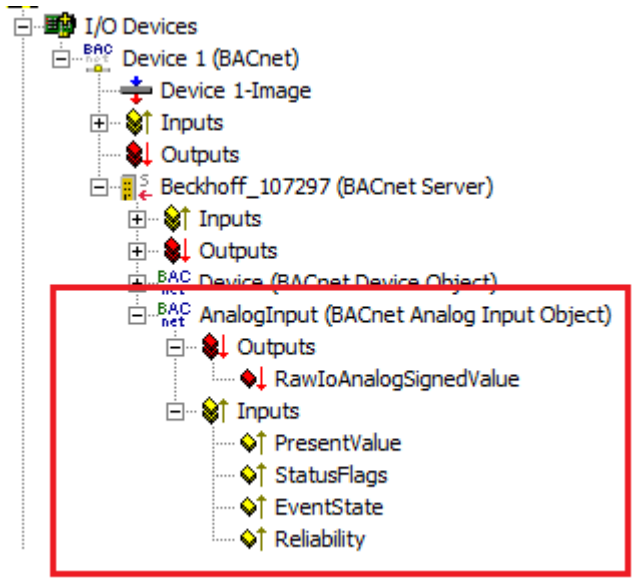


Fig. 129: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
nPV          : UDINT;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of the input **nPV**. This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

nPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and -3). Writing is enabled by setting the input **bEnablePV** to *TRUE*. Process data are always written cyclically once enabled.

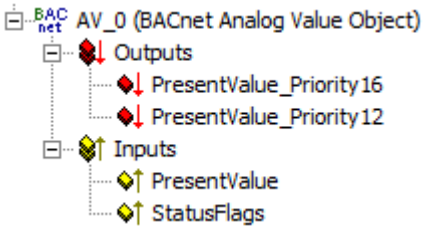


Fig. 130: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

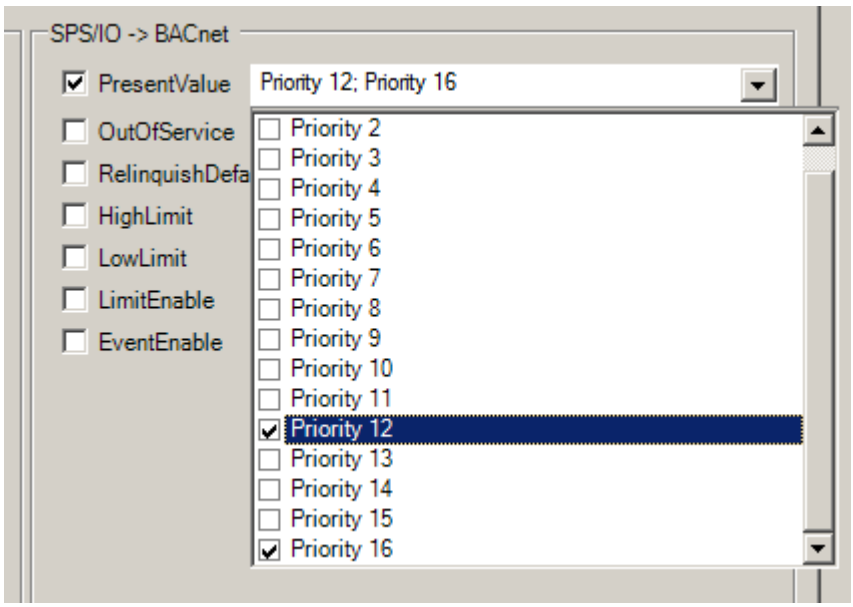


Fig. 131: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
nNbrOfStates    : UDINT;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the *MultiStateOutput* object can assume (1...*nNbrOfStates*). If **nNbrOfStates** is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

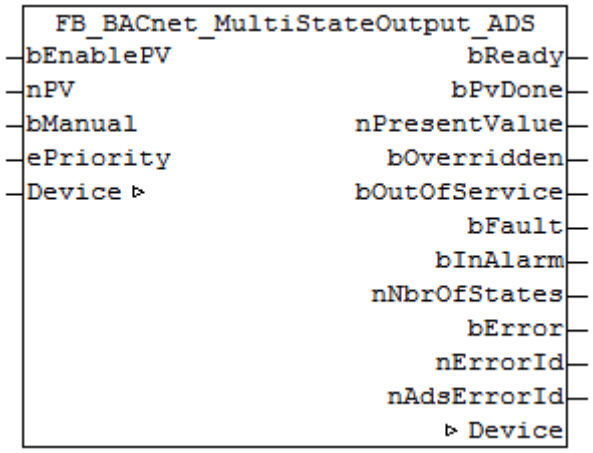
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.24 FB_BACnet_MultiStateOutput_ADS

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_MultiStateOutput_ADS" can be used for read and asynchronous write access to a BACnet object of type *MultiStateOutput* (MO). To this end the BACnet object was created under a local BACnet server.

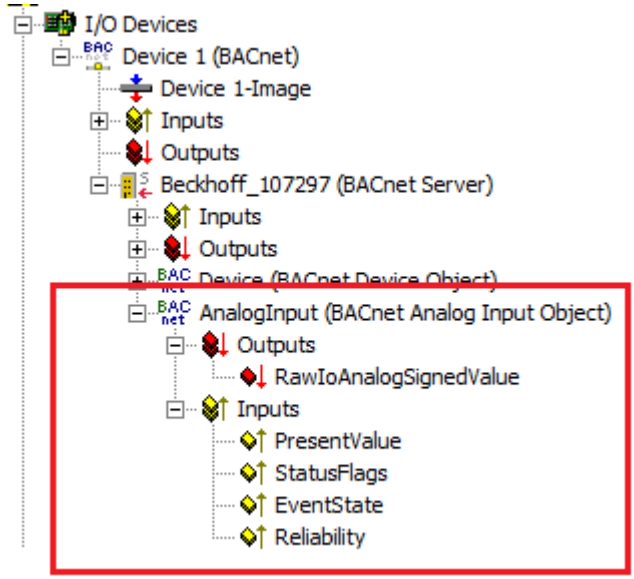


Fig. 132: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV           : BOOL;
nPV                 : UDINT; (* number for priority x, when input bEnablePV is TRUE *)
bManual             : BOOL; (* FALSE --> TRUE: write present_value immediately,
                FALSE: only write present_value on-change,
                TRUE: write present_value on-change and when device turns operational *)
ePriority           : E_BACNETPRIORITY := BACNETPRIORITY_12;
```

bEnablePV: enables the value of input "nPV". As soon as the input is set to *TRUE* there is a write access with the value from "nPV" to the commandable property *Present_Value* with priority from "ePriority" (default: 12, if the input "ePriority" is not connected).
If the input is set to *FALSE*, a write access with the value *NULL* is made to the commandable property *Present_Value* with priority from "ePriority" (NULL write).

nPV: value to be written to the *Priority_Array* of the commandable property *Present_Value* when "bEnablePV" is set to *TRUE*. The priority is determined with the input "ePriority" (default: 12, if the input "ePriority" is not connected). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing to the object is executed via ADS when the value changes.

Since the writing of the property *Present_Value* is performed acyclically and only when the value changes, it is potentially possible that the property *Present_Value* with the same priority is also overwritten by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: a distinction is made between the following options for the input:

- *FALSE*: writing when the value changes
- Edge change *FALSE --> TRUE*: triggers the writing of the property *Present_Value* immediately.
- *TRUE*: writing when the value changes *and* the associated BACnet server changes to "Operational" state (see FB_BACnet_Device [▶ 414]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 508]).

VAR_OUTPUT

```
bReady       : BOOL;
bPvDone      : BOOL; (* present_value written over ADS *)
nPresentValue: UDINT;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
nNbrOfStates : UDINT;
bError       : BOOL;
nErrorId     : UINT;
nAdsErrorId  : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPvDone: Positive edge when write access to property *Present_Value* was successfull.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the *MultiStateOutput* object can assume (1...*nNbrOfStates*). If **nNbrOfStates** is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
4 = the object type of the BACnet object dosn't match the function block type (the object type will determined over the process data. --> check the process data mapping in the System Manager!)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nAdsErrorId: ADS error code.

VAR_IN_OUT

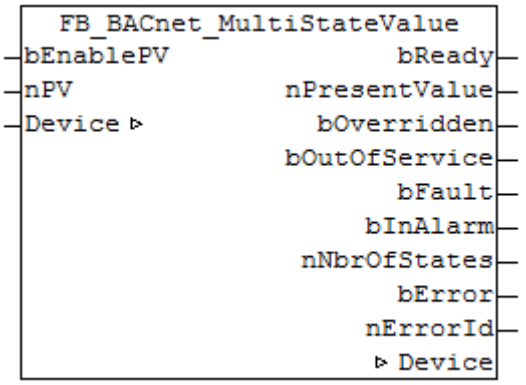
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.25 FB_BACnet_MultiStateValue

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_MultiStateValue" can be used for read and write access to a BACnet object of type *MultiStateValue* (MV). To this end the BACnet object was created under a local BACnet server.

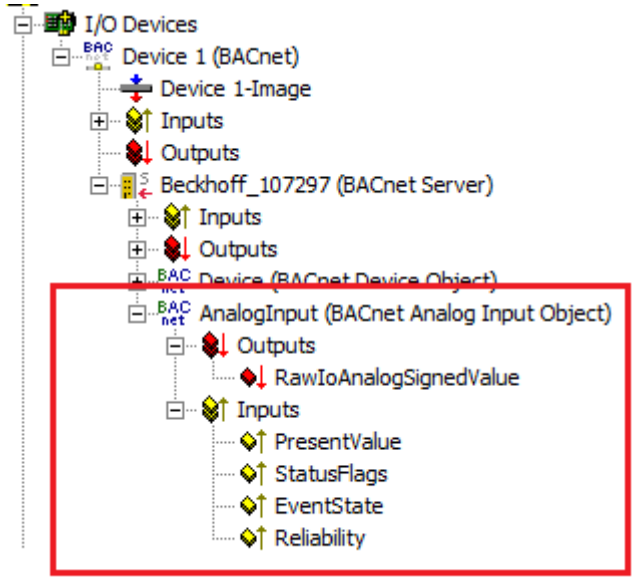


Fig. 133: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV    : BOOL;
nPV          : UDINT;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of the input **nPV**. This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

nPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and -3). Writing is enabled by setting the input **bEnablePV** to *TRUE*. Process data are always written cyclically once enabled.

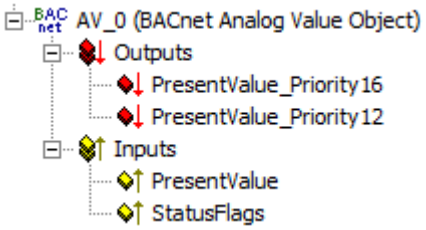


Fig. 134: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

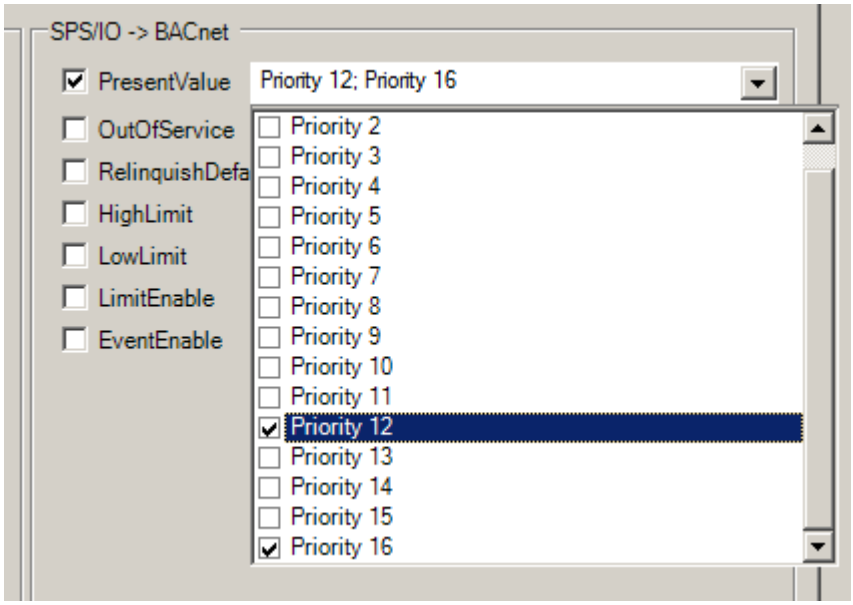


Fig. 135: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 and 16 are created as mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
nNbrOfStates    : UDINT;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Status_Flags*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the *MultiStateValue* object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

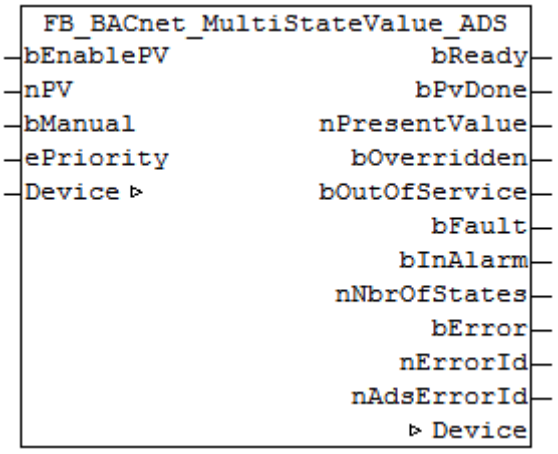
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.3.26 FB_BACnet_MultiStateValue_ADS

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_MultiStateValue_ADS" can be used for read and asynchronous write access to a BACnet object of type *MultiStateValue* (MV). To this end the BACnet object was created under a local BACnet server.

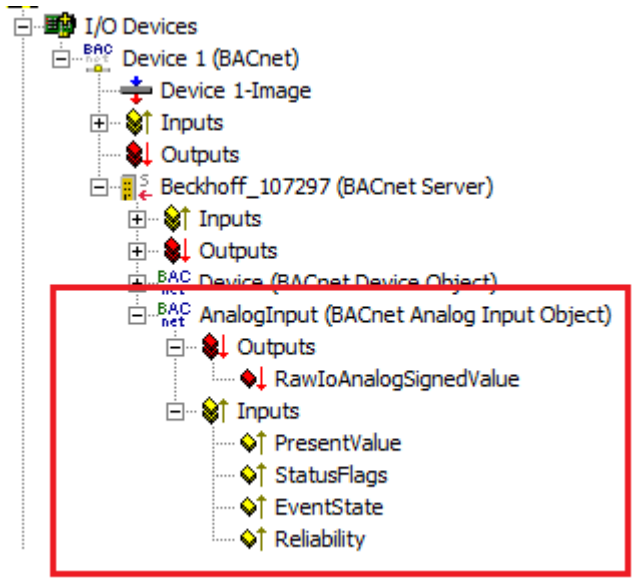


Fig. 136: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_INPUT

```
bEnablePV            : BOOL;
nPV                  : UDINT; (* number for priority x, when input bEnablePV is TRUE *)
bManual              : BOOL; (* FALSE --> TRUE: write present_value immediately,
                FALSE: only write present_value on-change,
                TRUE: write present_value on-change and when device turns operational *)
ePriority            : E_BACNETPRIORITY := BACNETPRIORITY_12;
```

bEnablePV: enables the value of input "nPV". As soon as the input is set to *TRUE* there is a write access with the value from "nPV" to the commandable property *Present_Value* with priority from "ePriority" (default: 12, if the input "ePriority" is not connected).
If the input is set to *FALSE*, a write access with the value *NULL* is made to the commandable property *Present_Value* with priority from "ePriority" (NULL write).

nPV: value to be written to the *Priority_Array* of the commandable property *Present_Value* when "bEnablePV" is set to *TRUE*. The priority is determined with the input "ePriority" (default: 12, if the input "ePriority" is not connected). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing to the object is executed via ADS when the value changes.

Since the writing of the property *Present_Value* is performed acyclically and only when the value changes, it is potentially possible that the property *Present_Value* with the same priority is also overwritten by other BACnet devices. In this case (i.e. same priority) the last write access always "wins".

bManual: the following evaluations are distinguished for the input:
- *FALSE*: write on value change
- Edge change *FALSE* --> *TRUE*: triggers the writing of the property *Present_Value* directly.
- *TRUE*: write on value change *and* when the associated BACnet server changes to the "Operational" state (see FB_BACnet_Device [▶ 414]).

ePriority: specification of the priority for write access to the property *Present_Value* (default: 12, see also E_BACNETPRIORITY [▶ 508]).

VAR_OUTPUT

```
bReady       : BOOL;
bPvDone      : BOOL; (* present_value written over ADS *)
nPresentValue    : UDINT;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
nNbrOfStates     : UDINT;
```

```
bError       : BOOL;
nErrorId     : UINT;
nAdsErrorId      : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPvDone: Positive edge when write access to property *Present_Value* was successfull.

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Status_Flags*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the *MultiStateValue* object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
4 = the object type of the BACnet object dosn't match the function block type (the object type will determined over the process data. --> check the process data mapping in the System Manager!)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

nAdsErrorId: ADS error code.

VAR_IN_OUT

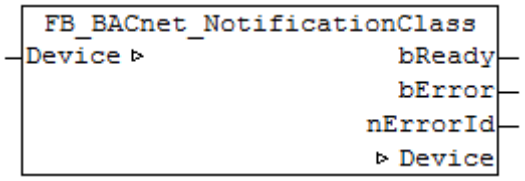
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.27 FB_BACnet_NotificationClass

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_NotificationClass" serves as placeholder for future functions.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If the output is *FALSE*, the associated function block "FB_BACnet_Device" reports "not operational".

bError: An error is pending.

nErrorId: Error number
0 = no error
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

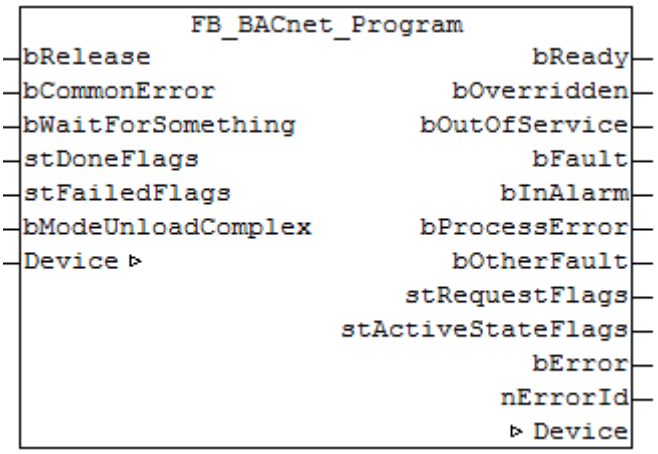
```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.28 FB_BACnet_Program

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Program" provides the object functionality for the BACnet stack. To this end the BACnet object was created under a local BACnet server. The BACnet object on the BACnet stack side serves as interface to the BACnet world. The processing of the BACnet object is provided by the PLC instance of the block FB_BACnet_Program.

This approach has the advantage that existing, customer-specific PLC functions can easily be linked with the BACnet world. In this way, procedures that are implemented in PLC code can be triggered with the aid of the handshake interface (*stDoneFlags*, *stFailedFlags* and *stRequestFlags*) of the FB "FB_BACnet_Program".

The function block "FB_BACnet_Program" contains the *state transitions* described in the BACnet specification and maps them to the FB interface using the structures: *stDoneFlags*, *stFailedFlags* and *stRequestFlags* (see below Transition diagram [▶ 437] Fig. 2).

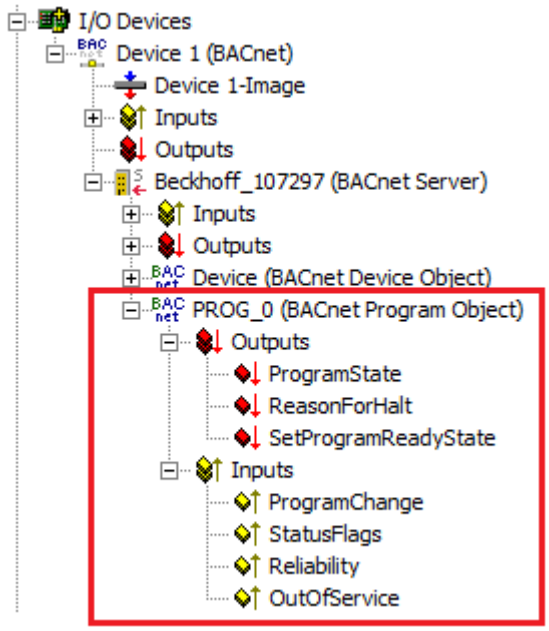


Fig. 137: Fig. 1: Example of a BACnet program object under a local BACnet server.

VAR_INPUT

```
bRelease            : BOOL;
bCommonError        : BOOL;
bWaitForSomething   : BOOL;
stDoneFlags         : ST_BACnet_ProgramHandshakeRequests;
stFailedFlags       : ST_BACnet_ProgramHandshakeRequests;
bModeUnloadComplex : BOOL := TRUE;
```

bRelease: Enables processing of requests. If **bRelease** = *FALSE* is set, the program waits in state *HALTED*.

bCommonError: Error during execution of a*Program_Change* requests from the structure **stRequestFlags**. Setting the input while a *Program_Change* requests is running leads to an abort condition (see Transition diagram Fig. 2).

bWaitForSomething: If the BACnet object is in state *RUNNING*, setting the input to *TRUE* switches to state *WAITING*, so that the BACnet object is blocked for further requests except *Unload* and *Halt* (see Transition diagram Fig. 2).
To return to state *RUNNING* set the input back to *FALSE*.

stDoneFlags: The flags within the input structure are used to acknowledge pending *Program_Change* requests from the output structure **stRequestFlags**. A signal change of one of the flags from *FALSE* to *TRUE* acknowledges execution of the corresponding *Program_Change* request.

stFailedFlags: The flags within the input structure are used to report cancellation of the pending *Program_Change* request from the output structure **stRequestFlags**. A signal change of one of the flags from *FALSE* to *TRUE* cancels execution of the corresponding *Program_Change* request.

bModeUnloadComplex: If the mode "Unload Complex" was activated via the input, the BACnet object status changes from *UNLOADING* to *IDLE* once the request *Unload* has been executed (see Transition diagram Fig. 2). If the input is set to *FALSE*, the BACnet object remains in state *UNLOADING* after the request *Unload* has been executed.

VAR_OUTPUT

```
bReady              : BOOL;
bOverridden         : BOOL;
bOutOfService       : BOOL;
bFault              : BOOL;
bInAlarm            : BOOL;
bProcessError       : BOOL;
bOtherFault         : BOOL;
stRequestFlags      : ST_BACnet_ProgramHandshakeRequests;
```

```
stActiveStateFlags : ST_BACnet_ProgramHandshakeStates;
bError             : BOOL;
nErrorId           : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (ActiveStateFlags, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Status_Flags*.

bProcessError, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Reliability*.

stRequestFlags: Contains the flags for the corresponding *Program_Change* requests. If a flag is set to *TRUE*, a change in status of the program object is requested by the PLC program. If the change was successful, this is acknowledged with a signal change from *FALSE* to *TRUE* of the corresponding flags in the input structure **stDoneFlags**. If the corresponding PLC program is unable to execute the request or if the process is aborted, this must be reported via the corresponding flag in the input structure **stFailedFlags**. The termination is then shown in the status of the BACnet object.

stActiveStateFlags: Contains flags that reflect the current state of the BACnet program object (see Transition diagram Fig. 2).

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

Transition diagram

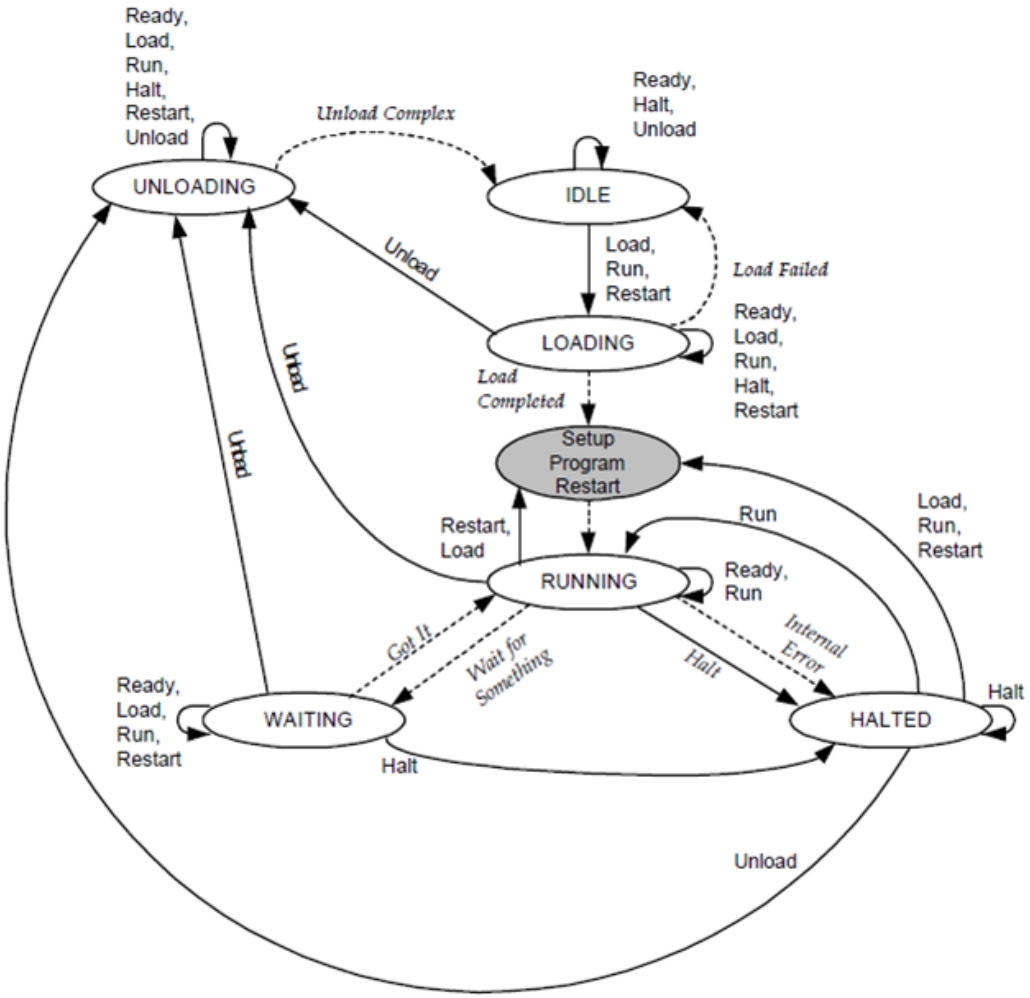
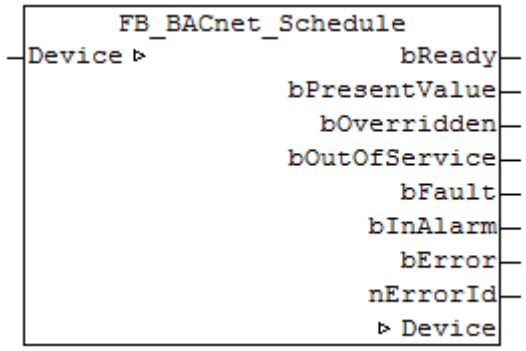


Fig. 138: Fig. 2: from BACnet specification DIN EN ISO 16484-5 for BACnet object Program, Fig. 12-3 "State Transitions for the program object"

5.3.29 FB_BACnet_Schedule

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_Schedule" can be used for reading access to a BACnet object of type *Schedule* (SCHED). To this end the BACnet object was created under a local BACnet server.

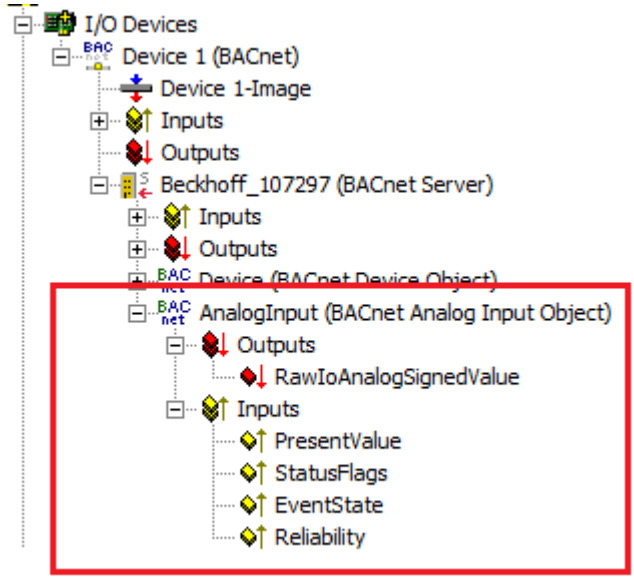


Fig. 139: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady       : BOOL;
bPresentValue: BOOL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device" does not report "Operational", or the block instance was not linked correctly in the System Manager.

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Present_Value*). The state is specified in 3 stages (enumeration) (0 = INACTIVE, 1 = ACTIVE and 2 = ZERO).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Status_Flags*.

bConfigError, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Reliability*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device [▶ 414])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

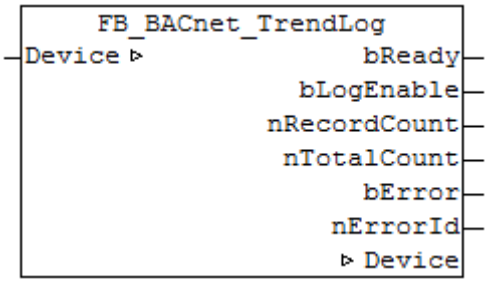
```
Device     : FB_BACnet_Device;
```

Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_Device [▶ 414] for further information.

5.3.30 FB_BACnet_TrendLog

The following function block is used for linking a BACnet object of the local BACnet server. The function block for the corresponding BACnet object is linked with the aid of process data.

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_TrendLog" can be used for read access to a BACnet object of type *TrendLog* (TREND). To this end the BACnet object was created under a local BACnet server.

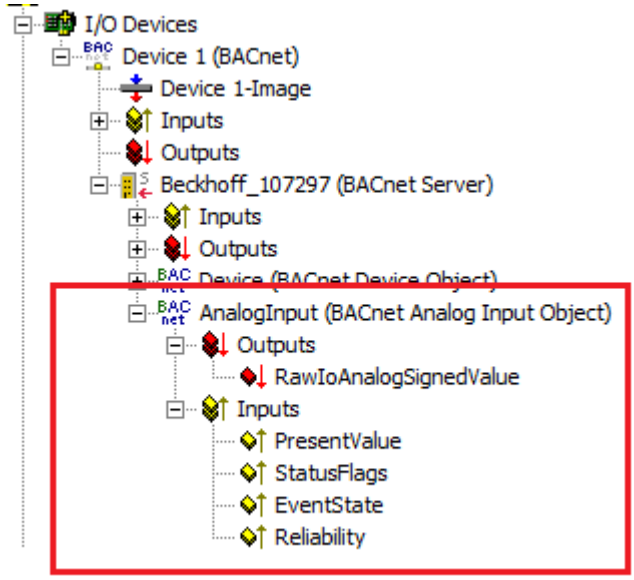


Fig. 140: Fig. 1: Example of a BACnet object under a local BACnet server.

VAR_OUTPUT

```
bReady          : BOOL;
bLogEnable      : BOOL;
nRecordCount    : UDINT;
nTotalCount     : UDINT;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (LogEnable, RecordCount...). If the output is *FALSE*, the corresponding function block "FB_BACnet_Device " does not report "Operational", or the block instance was not linked correctly in the System Manager.

bLogEnable: See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Log_Enable*.

nRecordCount: See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Record_Count*.

nTotalCount: See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Total_Record_Count*.

bError: An error is pending.

nErrorId: Error number
0 = no error
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet server is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_Device)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

```
Device       : FB_BACnet_Device;
```

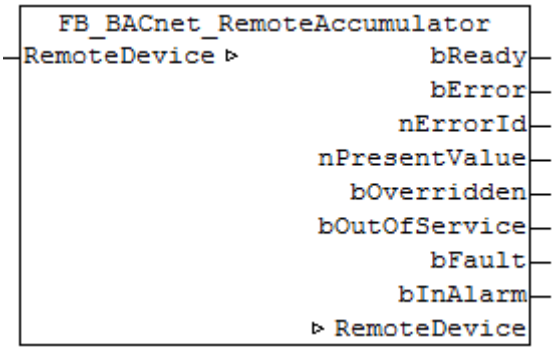
Device: Specification of the instance of the local corresponding BACnet server block. For each BACnet adapter one BACnet server is possible. See FB_BACnet_Adapter and FB_BACnet_Device for further information.

5.4 BACnet Client Objects

5.4.1 FB_BACnet_RemoteAccumulator

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteAccumulator" can be used for read access to a remote BACnet object of type *Accumulator* (AC). To this end the remote BACnet object was added to a local BACnet client.

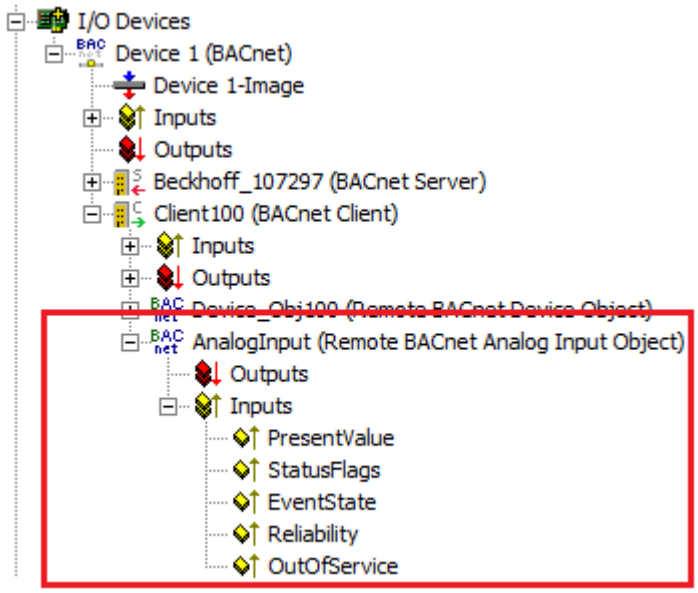


Fig. 141: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
nPresentValue: UDINT;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Present_value*).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Accumulator* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

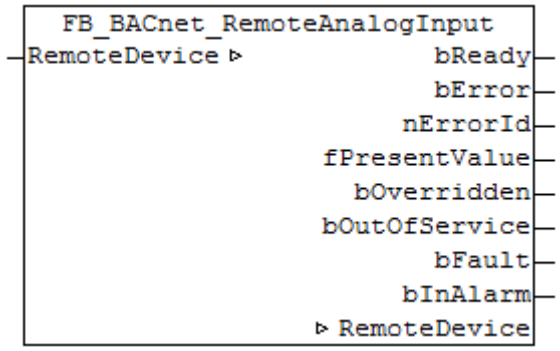
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.2 FB_BACnet_RemoteAnalogInput

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteAnalogInput" can be used for read access to a remote BACnet object of type *AnalogInput* (AI). To this end the remote BACnet object was added to a local BACnet client.

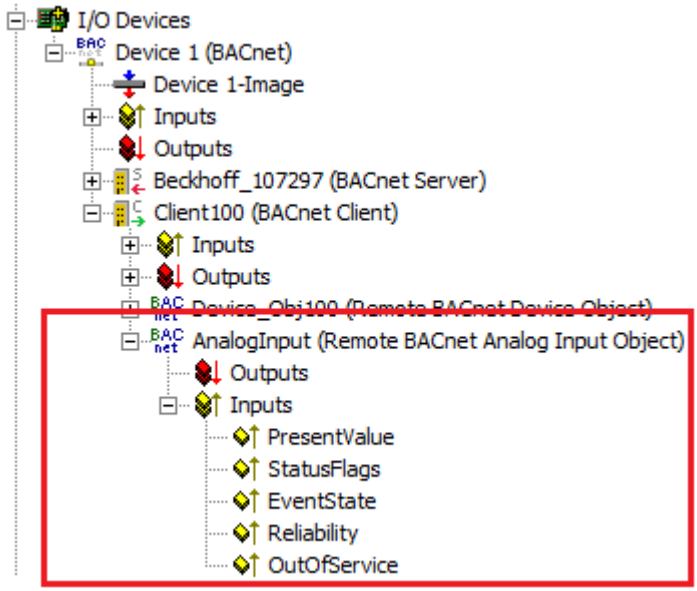


Fig. 142: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
fPresentValue: REAL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Present_Value*).

bOverriden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogInput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_???*.**nERR_xxx**).

VAR_IN_OUT

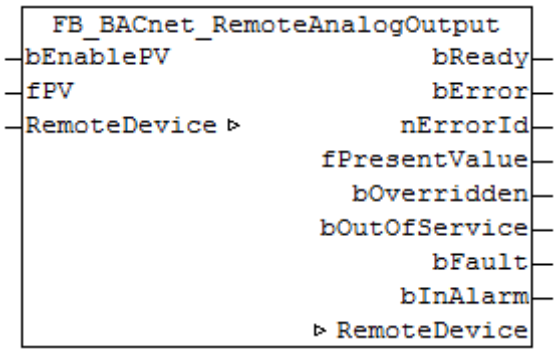
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.3 FB_BACnet_RemoteAnalogOutput

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteAnalogOutput" can be used for read and write access to a BACnet object of type *AnalogOutput* (AO) To this end the remote BACnet object was added to a local BACnet client.

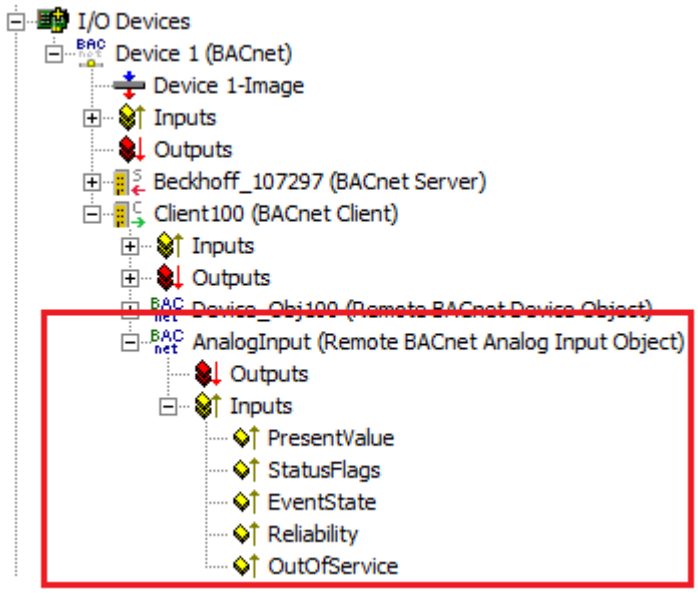


Fig. 143: Fig. 1: Example for a remote BACnet object under a local BACnet client

VAR_INPUT

```
bEnablePV    : BOOL;
fPV          : REAL;
```

bEnablePV: Enables the value of input "fPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "fPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).

fPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing of the process data to the local client always takes place cyclically. After enabling via bEnablePV the data are sent via BACnet to the remote server either cyclically (Periodic Write) or when changes occur (Write On Change - recommended).

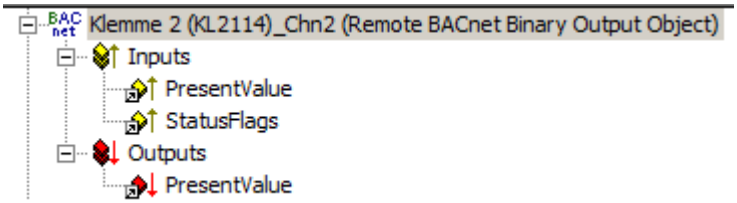


Fig. 144: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

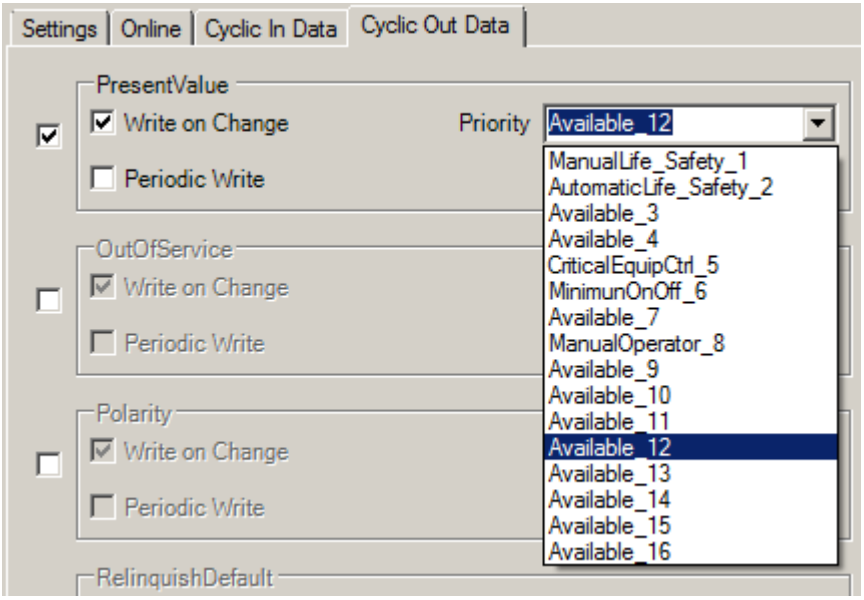


Fig. 145: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 is created as a mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady       : BOOL;
fPresentValue: REAL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_RemoteDevice reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogOutput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

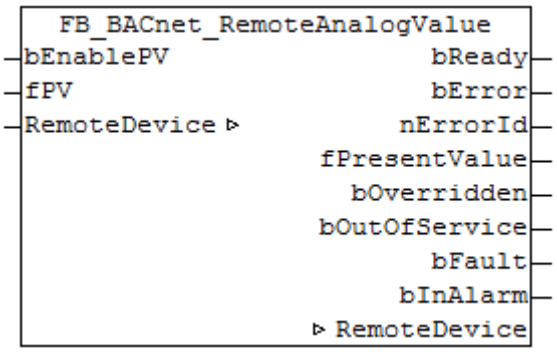
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.4 FB_BACnet_RemoteAnalogValue

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteAnalogValue" can be used for read and write access to a remote BACnet object of type *AnalogValue* (AV). To this end the remote BACnet object was added to a local BACnet client.

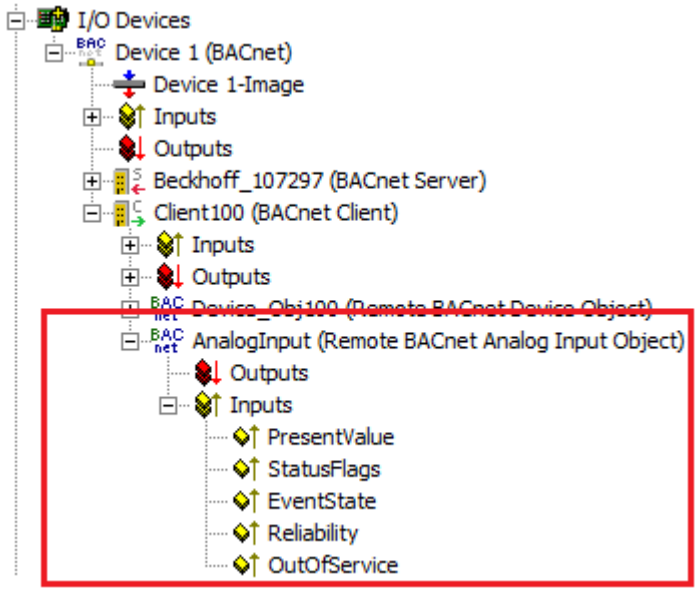


Fig. 146: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV    : BOOL;
fPV          : REAL;
```

bEnablePV: Enables the value of input **fPV**. When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input **fPV**. This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

fPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input **bEnablePV** to *TRUE*. Writing of the process data to the local client always takes place cyclically. After enabling via bEnablePV the data are sent via BACnet to the remote server either cyclically (Periodic Write) or when changes occur (Write On Change - recommended).

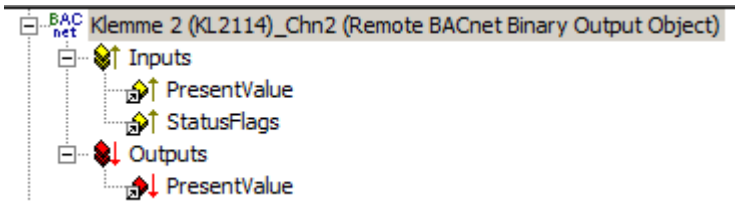


Fig. 147: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

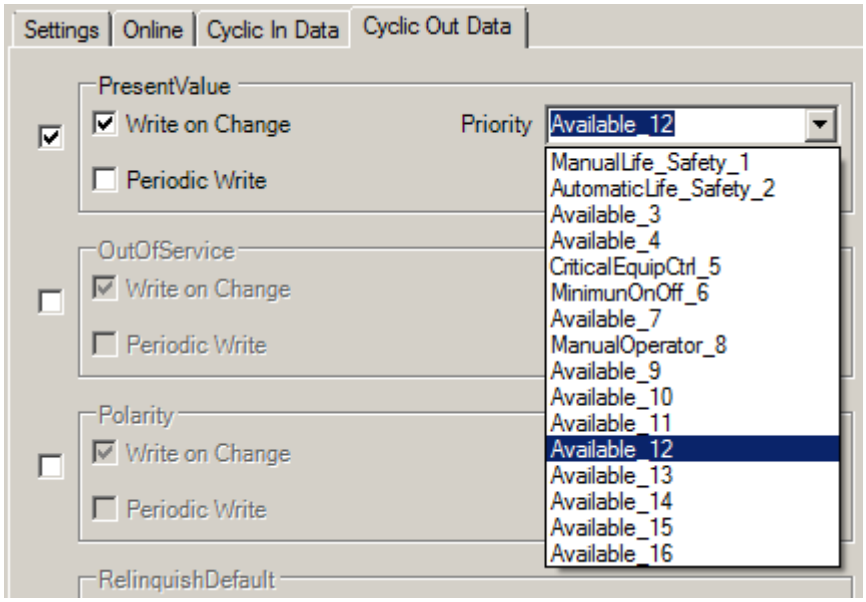


Fig. 148: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 is created as a mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady       : BOOL;
fPresentValue: REAL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

fPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogValue* and property *Present_Value*).

bOverriden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *AnalogValue* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

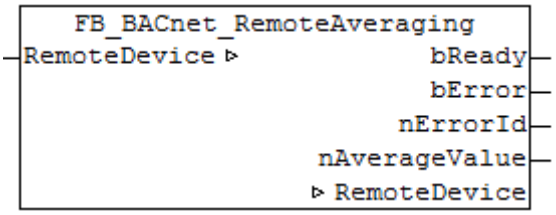
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.5 FB_BACnet_RemoteAveraging

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteAveraging" can be used for read access to a remote BACnet object of type *Averaging* (AVG). To this end the remote BACnet object was added to a local BACnet client.

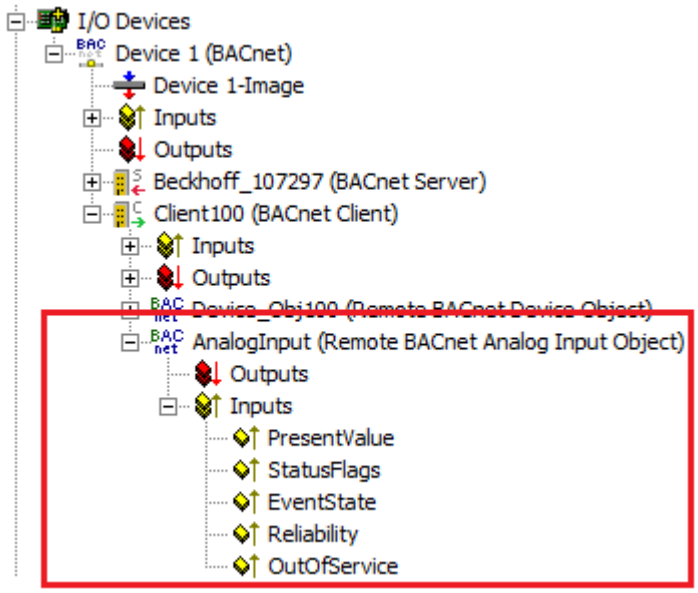


Fig. 149: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady           : BOOL;
nAverageValue    : UDINT;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

nAverageValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging*.

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Averaging* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

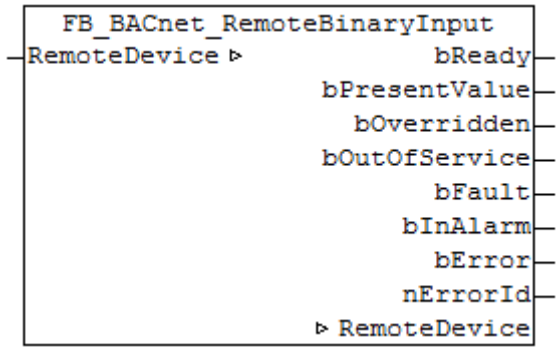
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.6 FB_BACnet_RemoteBinaryInput

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteBinaryInput" can be used for read access to a remote BACnet object of type *BinaryInput* (BI). To this end the remote BACnet object was added to a local BACnet client.

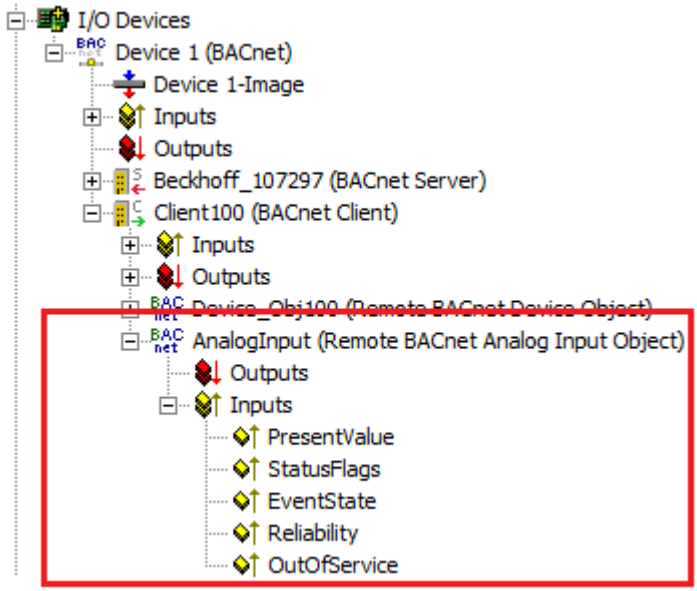


Fig. 150: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
bPresentValue: BOOL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryInput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

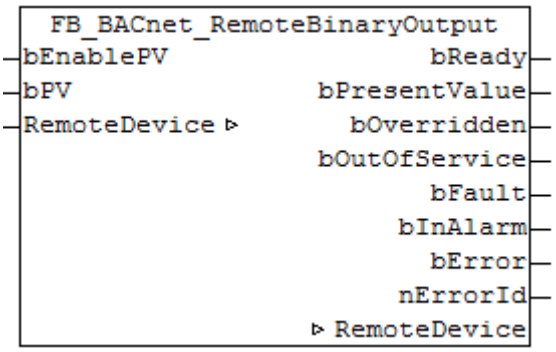
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.7 FB_BACnet_RemoteBinaryOutput

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteBinaryOutput" can be used for read and write access to a remote BACnet object of type *BinaryOutput* (BO). To this end the remote BACnet object was added to a local BACnet client.

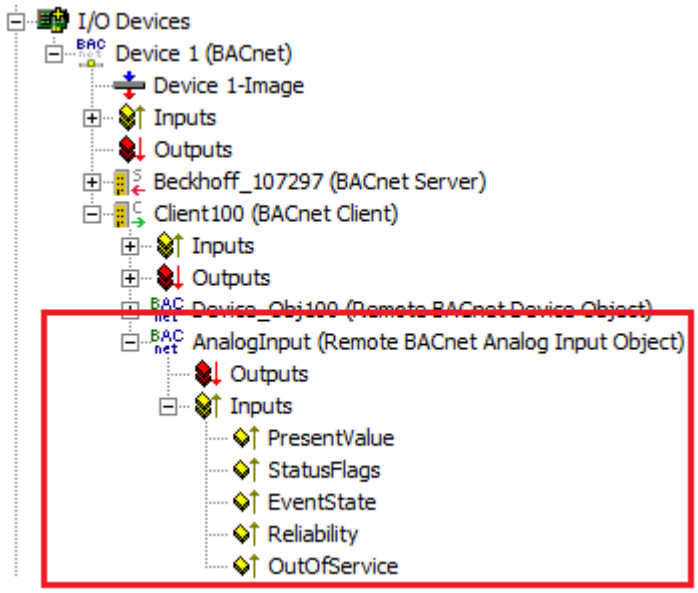


Fig. 151: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV    : BOOL;
bPV          : E_BACnetBinaryPV;
```

bEnablePV: Enables the value of input "bPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "bPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

bPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing of the process data to the local client always takes place cyclically. After enabling via bEnablePV the data are sent via BACnet to the remote server either cyclically (Periodic Write) or when changes occur (Write On Change - recommended).

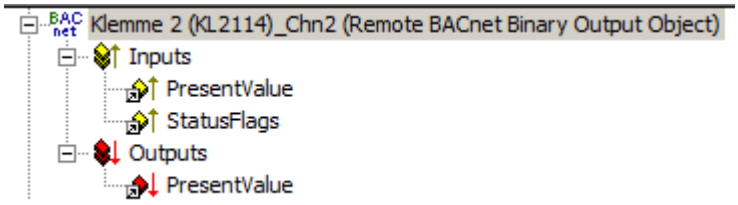


Fig. 152: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

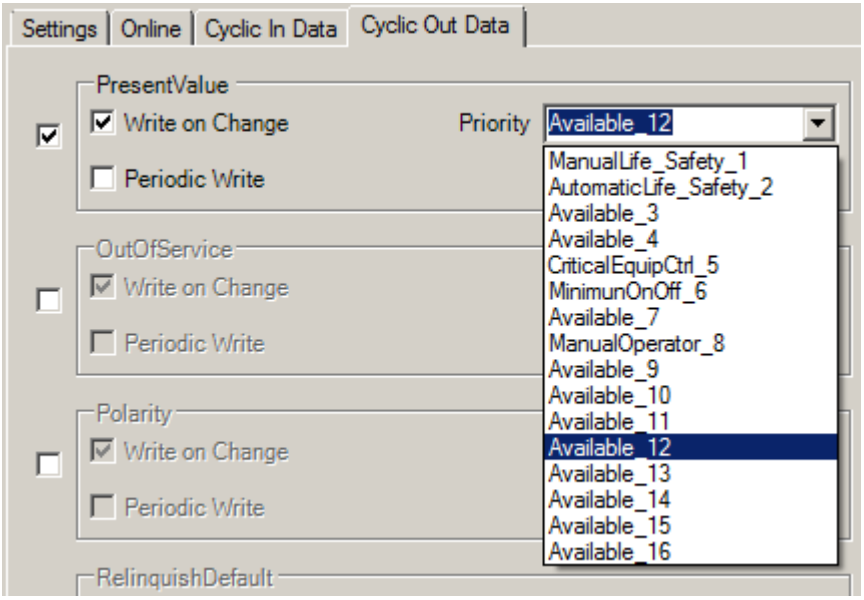


Fig. 153: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 is created as a mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady       : BOOL;
bPresentValue: BOOL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

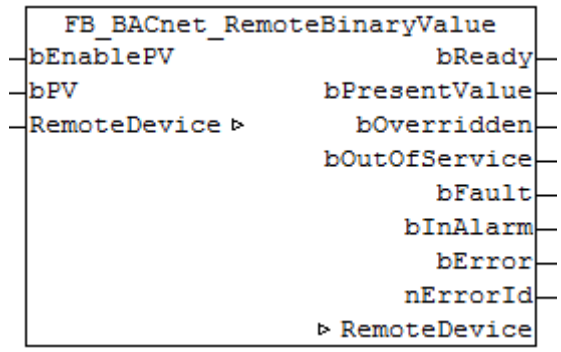
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.8 FB_BACnet_RemoteBinaryValue

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteBinaryValue" can be used for read and write access to a remote BACnet object of type *BinaryValue* (BV). To this end the remote BACnet object was added to a local BACnet client.

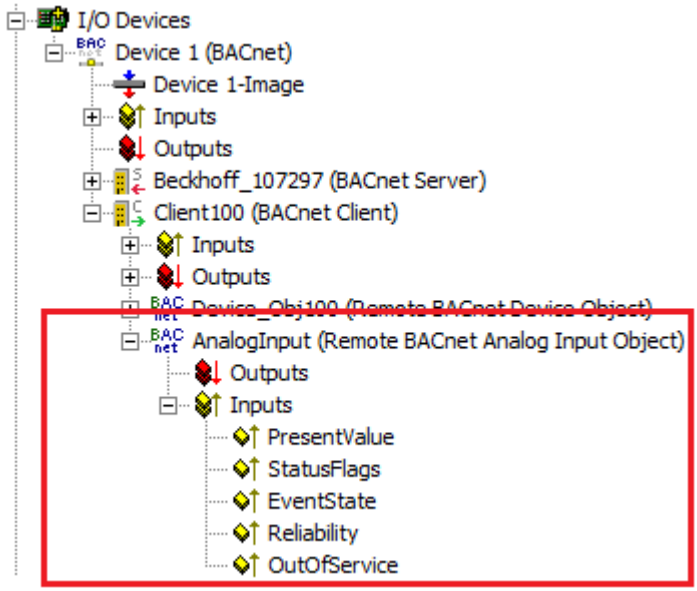


Fig. 154: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV    : BOOL;
bPV          : E_BACnetBinaryPV;
```

bEnablePV: Enables the value of input "bPV". When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of input "bPV". This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

bPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and 3). Writing is enabled by setting the input "bEnablePV" to *TRUE*. Writing of the process data to the local client always takes place cyclically. After enabling via bEnablePV the data are sent via BACnet to the remote server either cyclically (Periodic Write) or when changes occur (Write On Change - recommended).

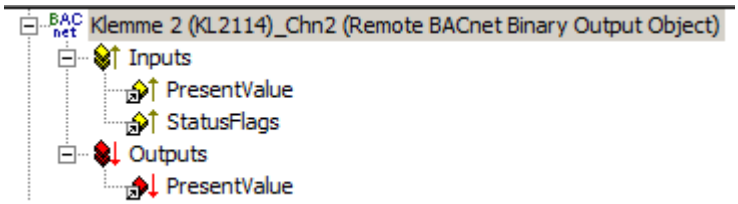


Fig. 155: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

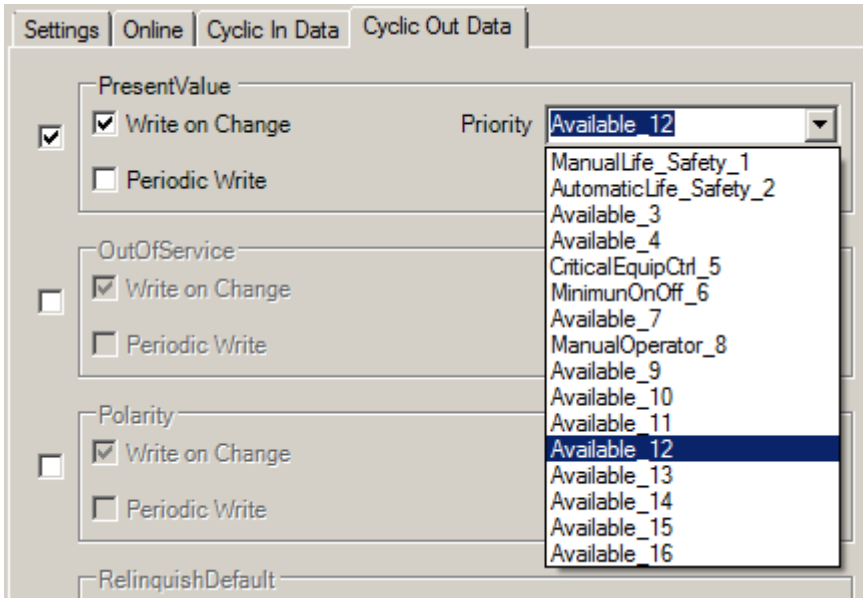


Fig. 156: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 is created as a mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady       : BOOL;
bPresentValue: BOOL;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryOutput* and property *Present_Value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *BinaryValue* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

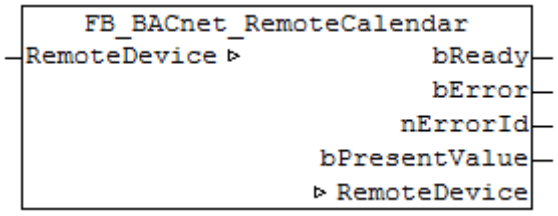
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.9 FB_BACnet_RemoteCalendar

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteCalendar" can be used for read access to a remote BACnet object of type *Calendar* (CAL). To this end the remote BACnet object was added to a local BACnet client.

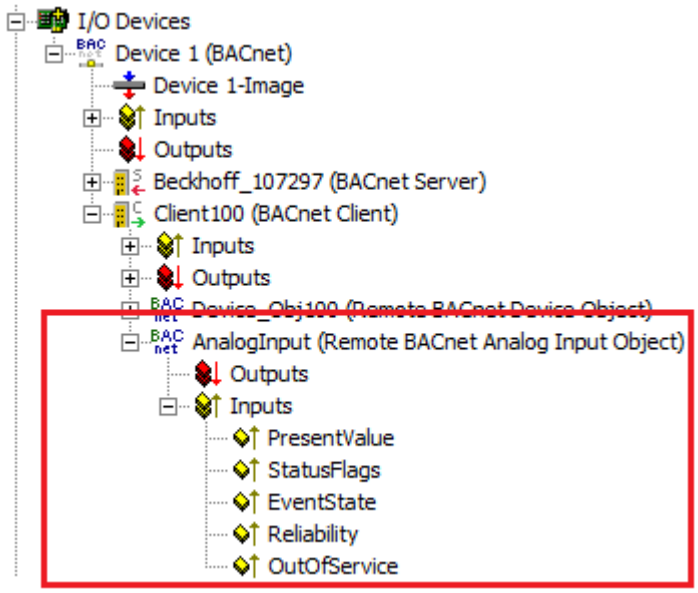


Fig. 157: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
bPresentValue: BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue). If the output is *FALSE*, the associated function block "FB_BACnet_RemoteDevice" reports "not operational".

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Calendar* and property *Present_Value*).

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

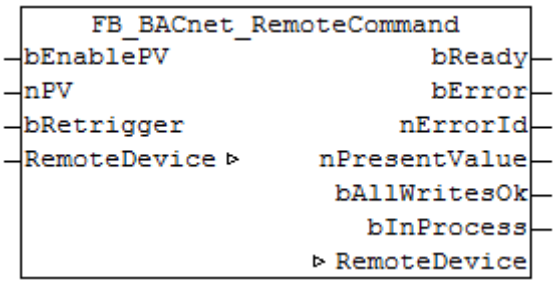
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.10 FB_BACnet_RemoteCommand

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteCommand" can be used for read and write access to a remote BACnet object of type *Command* (CMD). To this end the remote BACnet object was added to a local BACnet client.

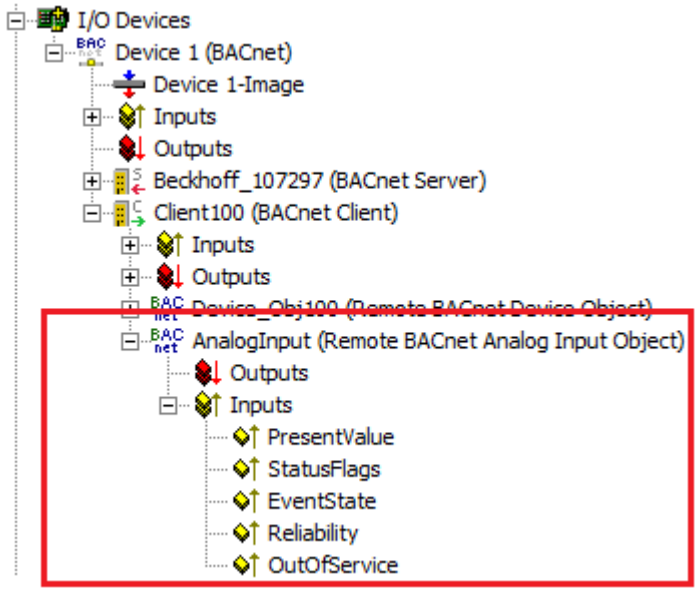


Fig. 158: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV    : BOOL;
nPV          : UDINT;
bRetrigger   : BOOL;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, writing takes place into the property *Present_Value* of the corresponding BACnet object with the value of input **nPV**.
If **bEnablePV** is set to *FALSE*, the process data of the mapped property *Present_Value* are written to *16#FFFFFFFF* and thus deactivated.

nPV: Value of the property *Present_Value* to be written. If the value is outside the value range (see BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*), the corresponding process data is disabled, i.e. writing to the property is disabled and other BACnet services have write access to the property *Present_Value*.
Writing of a valid value triggers execution of the corresponding command list of the BACnet object. The value of the property Present_Value is written whenever the process data changes (i.e. change in the value of **nPV** with **bEnablePV** set or signal change from *FALSE* to *TRUE* at input **bRetrigger** with **bEnablePV** set).

bRetrigger: A rising edge at this input triggers a repeat of the write process and therefore execution of the corresponding commands of the BACnet object *Command*. The signal change from *FALSE* to *TRUE* corresponds to a change of the process data of the property *Present_Value* from x to 0 to x.

VAR_OUTPUT

```
bReady          : BOOL;
nPresentValue   : UDINT;
bAllWritesOk    : BOOL;
bInProcess      : BOOL;
bError          : BOOL;
nErrorId        : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, AllWritesOk, InProcess). If the output is *FALSE*, the associated function block "FB_BACnet_RemoteDevice" reports "not operational".

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *Present_Value*).

bAllWritesOk: The last requested command list was successfully processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *All_Writes_Successful*).

bInProcess: The selected command list is processed (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Command* and property *In_Process*).

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

```
RemoteDevice    : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.11 FB_BACnet_RemoteDevice

The following function block is used for linking the PLC program to a remote BACnet device object (local client). The linking of the function block takes place with the aid of process data.

The process data can be linked manually or automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block. The process data for the BACnet device object also include the required process data for linking the BACnet client.

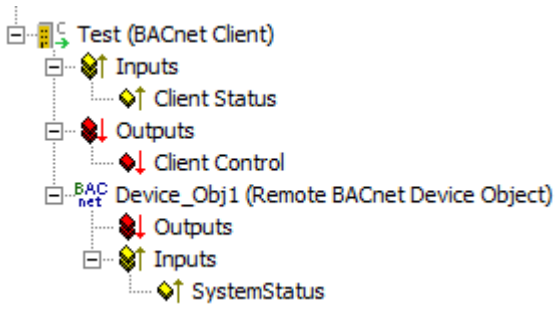


Fig. 159: Fig. 1: Process data of the remote BACnet device object and client in the System Manager.

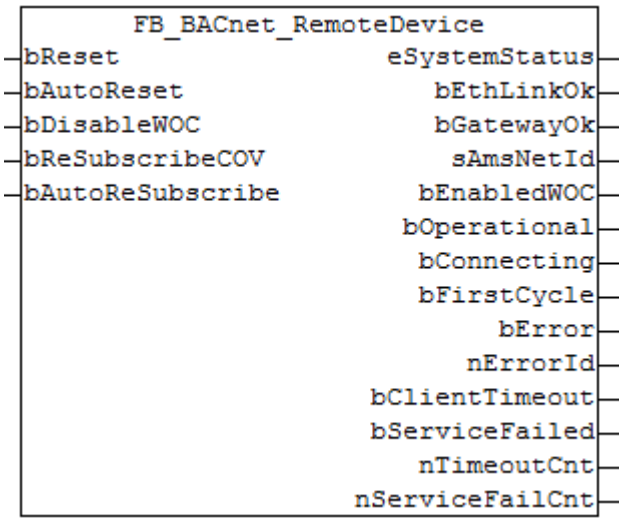


Fig. 160: Fig. 2: Function block of the remote BACnet device object and client in the PLC program.

Use

The function block FB_BACnet_RemoteDevice is used to read the status of the remote BACnet device object (*System_Status*) and output it in the PLC program. The process data "Client Control" and "Client Status" are also used to control the local BACnet client (disable/enable WriteOnChange, resubscription to properties, automatic subscription of properties after connection faults and error acknowledgement).

The function block FB_BACnet_RemoteDevice also requires a global instance of the function block FB_BACnet_Adapter [▶ 375] for each PLC project. The function block FB_BACnet_Adapter [▶ 375] establishes the connection between PLC and BACnet adapter in the System Manager. This global instance is already provided by the PLC library. The hierarchy between FB_BACnet_Adapter [▶ 375], FB_BACnet_Device [▶ 414] and FB_BACnet_RemoteDevice and an object of type *AnalogValue* is shown below:

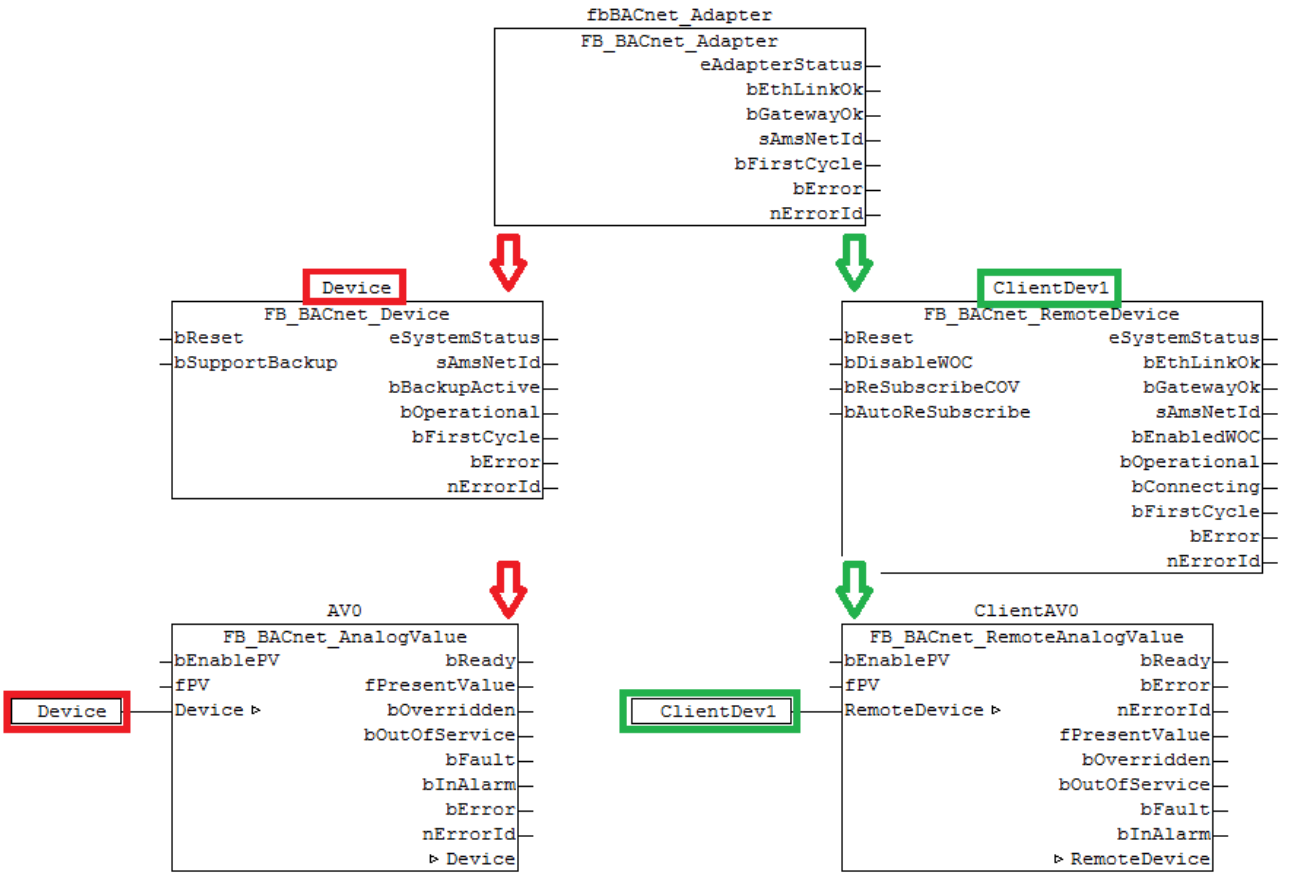


Fig. 161: Figure 3: Example for linking the FB instances in the PLC.

Information for use: see also function blocks FB_BACnet_Adapter [▶ 375] or FB_BACnet_Device [▶ 414].

VAR_INPUT

```
bReset           : BOOL;
bAutoReset       : BOOL;
bDisableWOC      : BOOL;
bReSubscribeCOV  : BOOL;
bAutoReSubscribe : BOOL;
```

bReset: Resets the error state in the event of a signal change *FALSE* --> *TRUE*.

bAutoReset: When input is *TRUE*, errors with ID 20 and 21 are automatically acknowledged. Related remote FBs will not be blocked (**bReady** of the FB remains *TRUE*). Instead of the error ID, an error counter is incremented (see **nTimeoutCnt** and **nServiceFailCnt**).

bDisableWOC: If the input is set to *TRUE*, WriteOnChange (WOC) is suppressed for all objects of this client. This means that changes in the process data do not automatically trigger writing of the properties to the remote objects (only applies to properties that are configured for WriteOnChange under the client).

bReSubscribeCOV: A signal change at the input from *FALSE to TRUE* triggers resubscription of all properties that are configured for COV. This function may affect many objects and should therefore only be executed when necessary. Normally all subscribed properties are automatically re-subscribed once a timeout time has elapsed (resubscription interval).

bAutoReSubscribe: If the input is set to *TRUE*, automatic resubscription of all properties that are configured for COV is triggered as required. Not to be confused with the resubscription interval! Automatic resubscription is triggered if a remote server was no longer accessible and the connection has been restored, for example. If the input is set to *FALSE*, resubscription is only triggered once the resubscription interval has elapsed.

VAR_OUTPUT

```
eSystemStatus    : E_BACnetDeviceStatus;
bEthLinkOk       : BOOL;
bGatewayOk       : BOOL;
sAmsNetId        : T_AmsNetId;
bEnabledWOC      : BOOL;
bOperational     : BOOL;
bConnecting      : BOOL;
bFirstCycle      : BOOL;
bError           : BOOL;
nErrorId         : UINT;
bClientTimeout   : BOOL;
bServiceFailed   : BOOL;
nTimeoutCnt      : UDINT;
nServiceFailCnt  : UDINT;
```

eSystemStatus: Current status of the remote BACnet server object (see BACnet specification DIN EN ISO 16484-5 for BACnet device object and property *System_Status*).
0: BACnetDeviceStatus_Operational
1: BACnetDeviceStatus_OperationalReadOnly (only read access to properties)
2: BACnetDeviceStatus_DownloadRequired (loading of the configuration required)
3: BACnetDeviceStatus_DownloadInProgress (configuration download in progress)
4: BACnetDeviceStatus_NonOperational
5: BACnetDeviceStatus_BackupInProgress

bEthLinkOk: The local Ethernet connection is active (cable connected, adapter linked) if the output is set to *TRUE*.

bGatewayOk: The configured gateway can be reached if the output is set to *TRUE*.

sAmsNetId: Output of the AMS NetID of the local BACnet adapter (can be used for asynchronous access to BACnet objects via ADS).

bEnabledWOC: WriteOnChange (WOC) is activated for properties that are configured for COV.

bOperational: BACnet device object of the remote server reports "operational" (System_Status = 0), the device adapter reports status **bEthLinkOk**, status **bConnecting** is *FALSE* and bit 0 of the process data "Client Status" is not set (nERR_CLIENT_TIMEOUT). If the output becomes *FALSE*, all linked objects are blocked (block instances).

bConnecting: The connection to the remote server is established (process data "Client Status" is less than 0x04xx).

bFirstCycle: Is set for one cycle when the block instance is first called after a PLC reset or restart.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
1 = no valid AMS NetID
3 = no network connection (bEthLinkOk = *FALSE*)
4 = gateway cannot be reached (bGatewayOk = *FALSE*)
20 = timeout during communication with remote server
21 = error during execution of a service between the remote server (Subscribe-COV, Property-Write etc.)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_RemoteDevice*.**nERR_xxx**). All errors with error number equal or greater than 20 must be acknowledged with **bReset**.

bClientTimeout: When output is set to *TRUE*, a timeout error (ID 20) has occurred. The output would reset to *FALSE* if no error occurred for 10 seconds.

bServiceFailed: When output is set to *TRUE*, a service error (ID 21) has occurred. The output would reset to *FALSE* if no error occurred for 10 seconds.

nTimeoutCnt: Number of timeout errors that occurred (ID 20). The positive edge of the corresponding status flag of client will count.

nServiceFailCnt: Number of errors that occurred Service (ID 21). The positive edge of the corresponding status flag of client will count.

An instance of the function block "FB_BACnet_RemoteDevice" is required whenever objects of a remote BACnet server are to be accessed from the PLC with the aid of a local BACnet client. Several instances of the function block "FB_BACnet_RemoteDevice" can be created and linked with the corresponding clients in the System Manager via PLC automapping. All instances of the function block "FB_BACnet_RemoteDevice" use the same global instance of function block FB_BACnet_Adapter.

If several instances of the function block FB_BACnet_Adapter are to be used, the corresponding instance of the BACnet adapter function block must be called before the instance of the BACnet client function block is called. A CFC example is provided below (Please note the order in which the function blocks are called!):

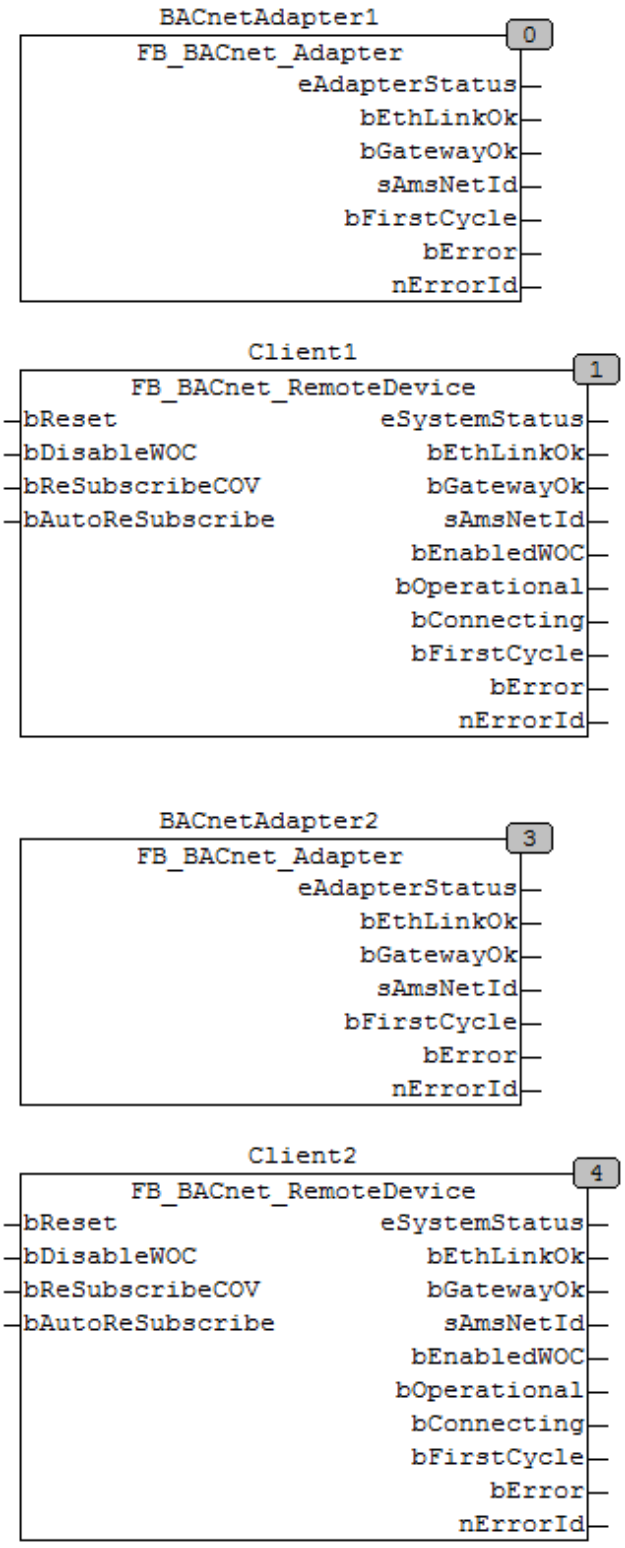


Fig. 162: Figure 4: Example for using several adapter and client instances in a PLC project.

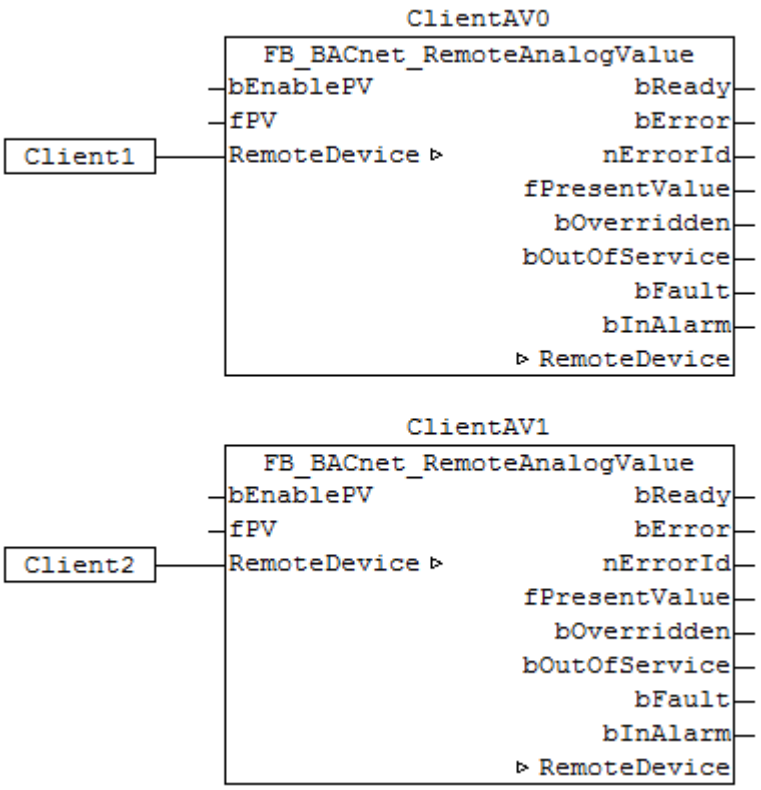


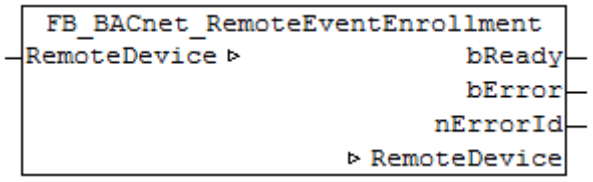
Fig. 163: Figure 5: Example for calling function blocks of type FB_BACnet_RemoteAnalogValue with different remote BACnet servers in a PLC project.

The use of several BACnet adapters in a PLC project is a special case and is not recommended. It could make sense (e.g. in cases where several network cards and therefore separate BACnet networks are used) to use several PLC projects and therefore several PLC runtimes. Data exchange between the individual runtimes can be realized via process data mapping or ADS communication. In this case PLC automapping could be used for BACnet objects.

5.4.12 FB_BACnet_RemoteEventEnrollment

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteEventEnrollment" serves as placeholder for future functions.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If the output is *FALSE*, the associated function block FB_BACnet_RemoteDevice [▶ 459] reports "not operational".

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

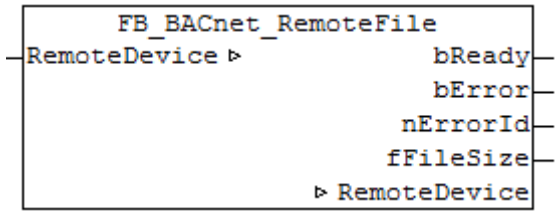
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.13 FB_BACnet_RemoteFile

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteFile" can be used for read access to a remote BACnet object of type *File* (FILE). To this end the remote BACnet object was added to a local BACnet client.

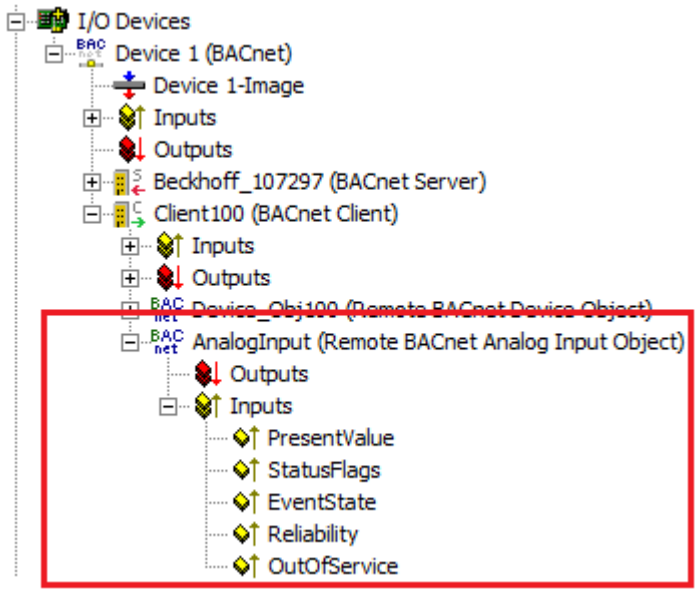


Fig. 164: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
bPresentValue: BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue). If the output is *FALSE*, the associated function block "FB_BACnet_RemoteDevice" reports "not operational".

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *File* and property *Present_Value*).

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

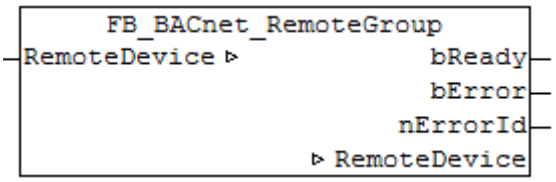
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.14 FB_BACnet_RemoteGroup

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteGroup" serves as placeholder for future functions.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If the output is *FALSE*, the associated function block FB_BACnet_RemoteDevice [▶ 459] reports "not operational".

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

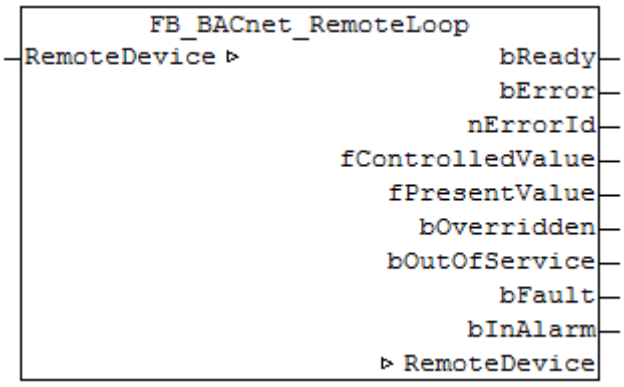
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.15 FB_BACnet_RemoteLoop

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteLoop" can be used for read access to a remote BACnet object of type *Loop* (LOOP). To this end the remote BACnet object was added to a local BACnet client.

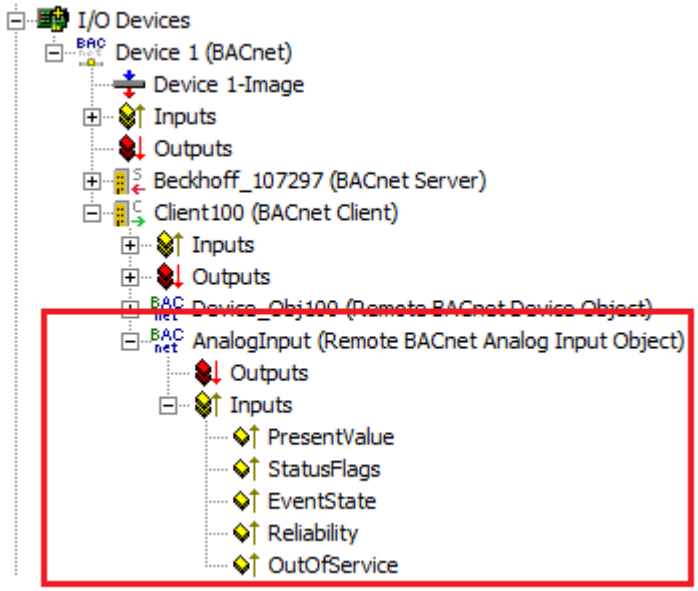


Fig. 165: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

Variables are only included in the extended version of the function block. The extended versions of the function blocks end with "_EX". They offer substantially more process data, although in projects with a large number of objects this may have performance implications.

```
bReady           : BOOL;
fControlledValue : REAL;
fPresentValue    : REAL;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

bReady: notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the function block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

fControlledValue: feedback of the current process parameter (X, actual value).

fPresentValue: feedback of the current control output (X, control value). Attention: *Present_Value* and *Controlled_Variable_Value* can easily lead to confusion (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and properties *Present_Value*, *Controlled_Variable_Value* and *Controlled_Variable_Reference*).

fPropBand: feedback of the current control output in percent (-100%...+100%) in relation to the minimum and maximum control output (properties *Minimum_Output* and *Maximum_Output*).

bOverridden, bOutOfService, bFault, bInAlarm: see BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop* and property *Status_Flags*.

bError: an error is pending.

nErrorId: error number
0 = no error
1 = function block of the associated client (RemoteDevice) is not called at all or too irregularly in the PLC program.
2 = faulty process data mapping detected (check mapping in the System Manager; if necessary compile PLC project completely and reload)
3 = the associated BACnet client is not ready (**bOperational** = *FALSE* on instance of FB_BACnet_RemoteDevice)
The error numbers can be queried as function block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

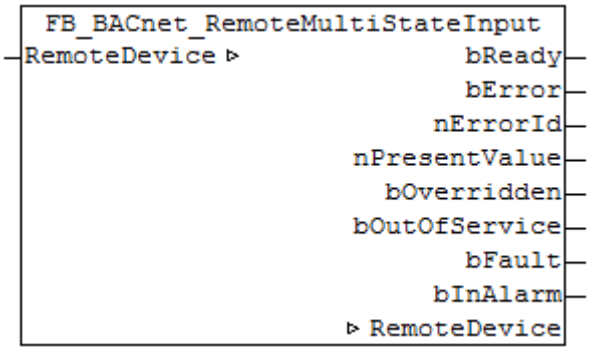
Controller configuration

The controller is configured using the following BACnet properties: *Action*, *Proportional_Constant* (P-factor), *Integral_Constant* (I-factor), *Derivative_Constant* (D-factor), Bias (output offset), *Maximum_Output* (maximum control output) and *Minimum_Output* (minimum control output). See BACnet specification DIN EN ISO 16484-5 for BACnet object *Loop*.

5.4.16 FB_BACnet_RemoteMultiStateInput

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteMultiStateInput" can be used for read access to a remote BACnet object of type *MultiStateInput* (MI). To this end the remote BACnet object was added to a local BACnet client.

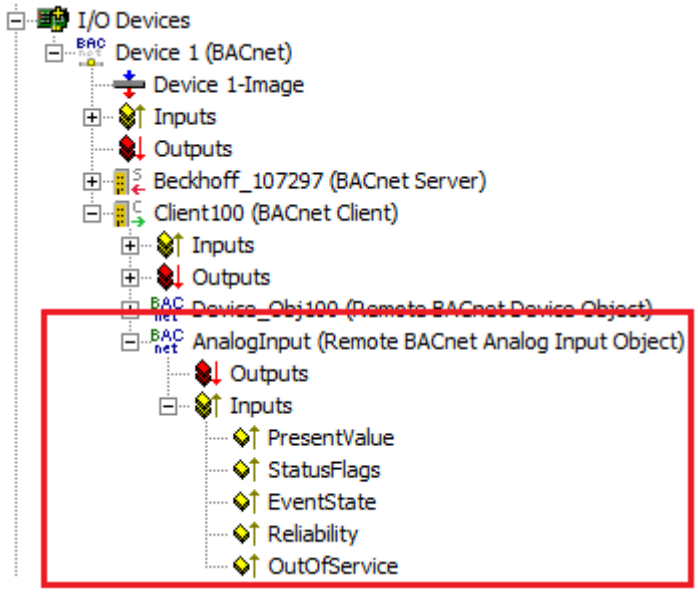


Fig. 166: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
nPresentValue: UDINT;
bOverridden  : BOOL;
bOutOfService: BOOL;
bFault       : BOOL;
bInAlarm     : BOOL;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateInput* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

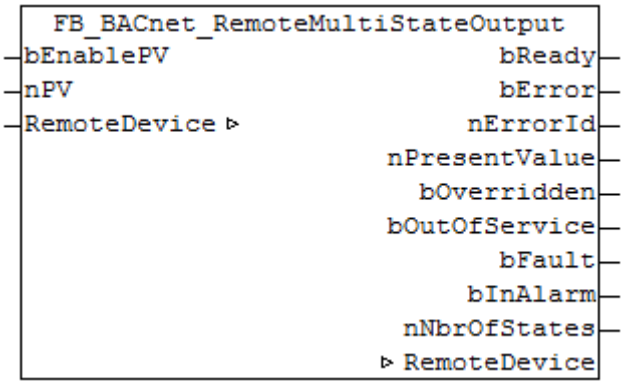
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.17 FB_BACnet_RemoteMultiStateOutput

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteMultiStateOutput" can be used for read and write access to a remote BACnet object of type *MultiStateOutput* (MO). To this end the remote BACnet object was added to a local BACnet client.

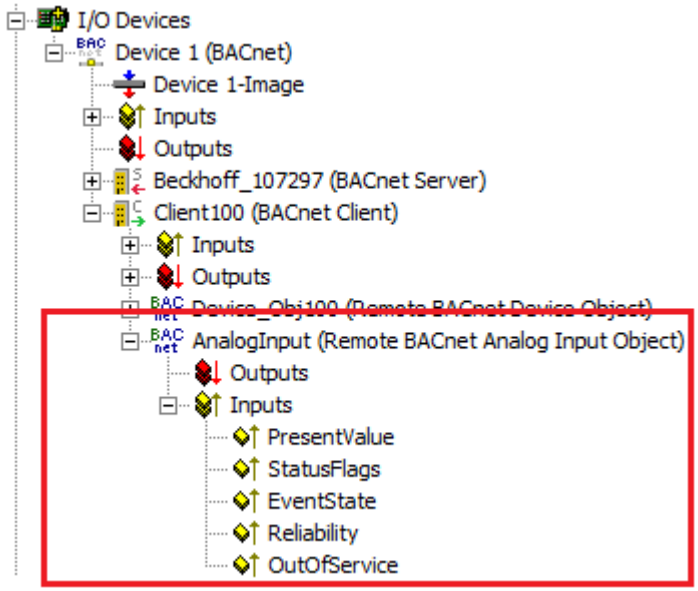


Fig. 167: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV     : BOOL;
nPV           : UDINT;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of the input **nPV**. This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

nPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and -3). Writing is enabled by setting the input **bEnablePV** to *TRUE*. Writing of the process data to the local client always takes place cyclically. After enabling via bEnablePV the data are sent via BACnet to the remote server either cyclically (Periodic Write) or when changes occur (Write On Change - recommended).

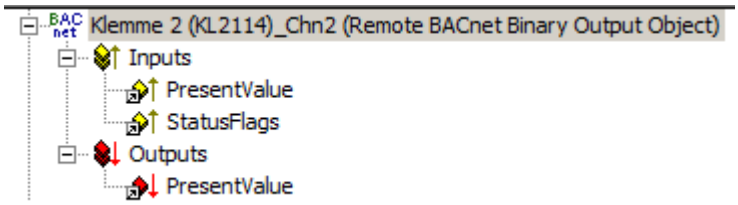


Fig. 168: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

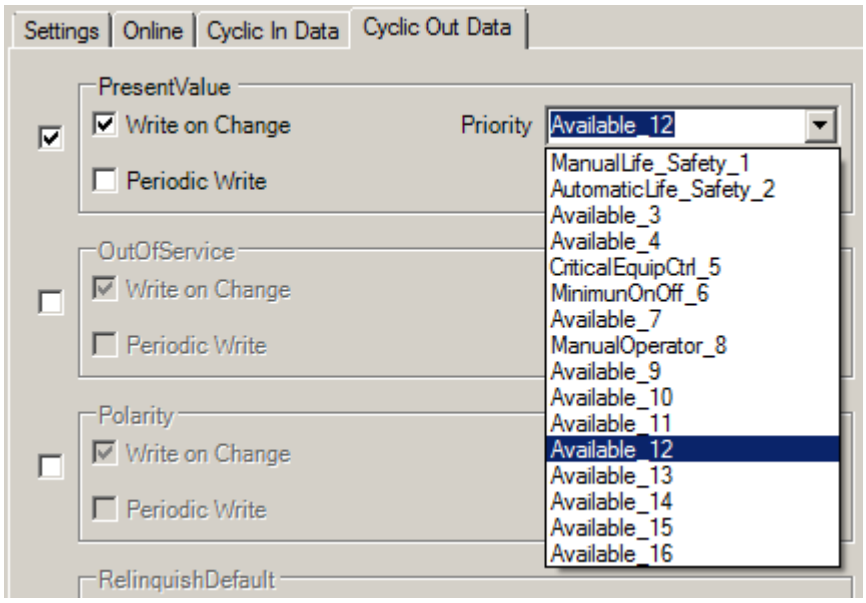


Fig. 169: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 is created as a mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
bError          : BOOL;
nErrorId        : UINT;
nPresentValue   : UDINT;
bOverridden     : BOOL;
bOutOfService   : BOOL;
bFault          : BOOL;
bInAlarm        : BOOL;
nNbrOfStates    : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_RemoteDevice reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateOutput* and property *Status_Flags*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateInput object can assume (1...*nNbrOfStates*). If **nNbrOfStates** is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

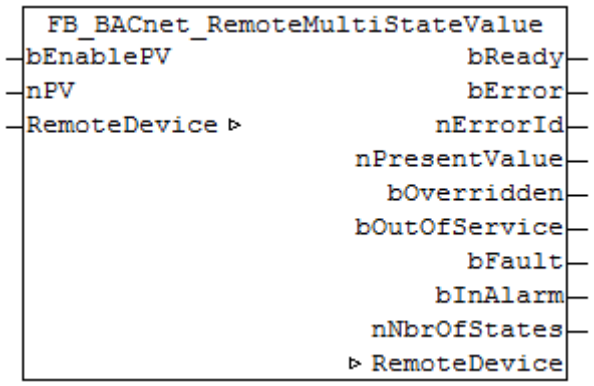
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.18 FB_BACnet_RemoteMultiStateValue

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteMultiStateValue" can be used for read and write access to a remote BACnet object of type *MultiStateValue* (MV). To this end the remote BACnet object was added to a local BACnet client.

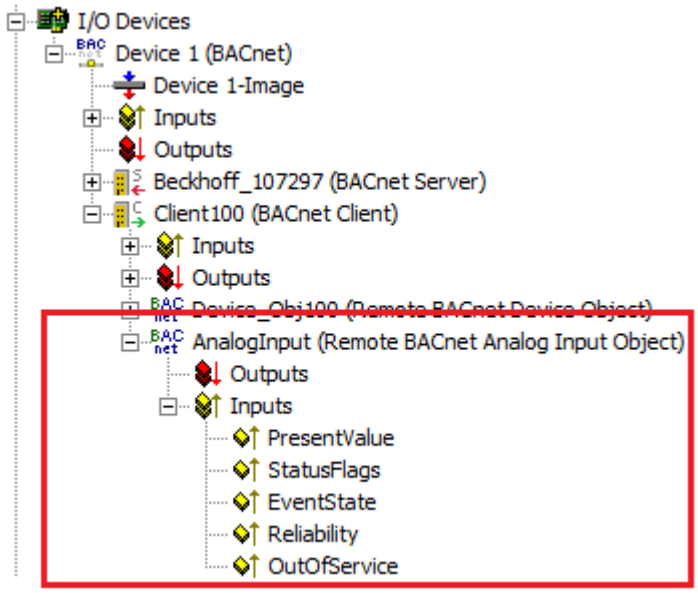


Fig. 170: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV    : BOOL;
nPV          : UDINT;
```

bEnablePV: Enables the value of input **nPV**. When the input is set to *TRUE*, an entry is made in the *Priority_Array* of the corresponding BACnet object with the value of the input **nPV**. This corresponds to a write access to the commandable property *Present_Value* with the set priority (default: 12 when PLC automapping is used or 16 for manual linking in the System Manager).
If bEnablePV is set to *FALSE*, the corresponding entry from the *Priority_Array* is removed again (write ZERO).

nPV: Value to be written to the *Priority_Array* of the commandable property *Present_Value*. The priority is based on the process data configuration and mapping of the BACnet object in the System Manager (see also Figs. 2 and -3). Writing is enabled by setting the input **bEnablePV** to *TRUE*. Writing of the process data to the local client always takes place cyclically. After enabling via bEnablePV the data are sent via BACnet to the remote server either cyclically (Periodic Write) or when changes occur (Write On Change - recommended).

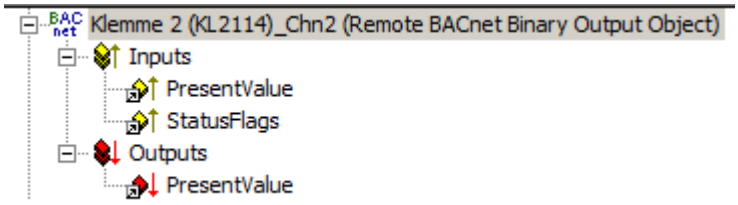


Fig. 171: Fig. 2: Example of a BACnet object with process data for writing the commandable property Present_Value.

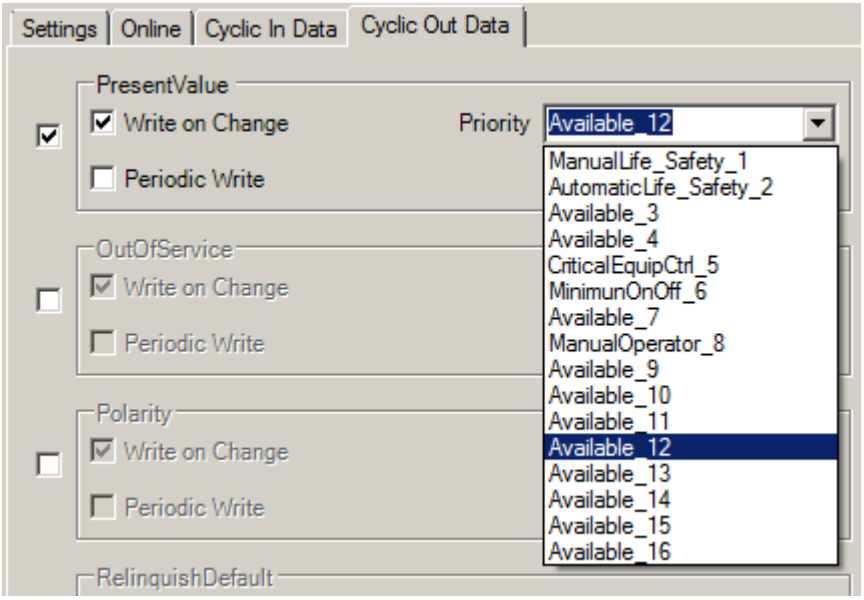


Fig. 172: Fig. 3: Example for a process data configuration of an BACnet object in the System Manager. Priority 12 is created as a mappable process data (see result in Fig. 2).

VAR_OUTPUT

```
bReady          : BOOL;
bError    : BOOL;
nErrorId   : UINT;
nPresentValue  : UDINT;
bOverridden    : BOOL;
bOutOfService  : BOOL;
bFault         : BOOL;
bInAlarm       : BOOL;
nNbrOfStates   : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

nPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Present_value*).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Status_Flags*.

bMultiStateFault, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Reliability*.

eEventState: See BACnet specification DIN EN ISO 16484-5 for BACnet object *MultiStateValue* and property *Event_State*.

nNbrOfStates: Reports the available number of values, which the *Present_Value* of the MultiStateValue object can assume (1...*nNbrOfStates*). If "nNbrOfStates" is 0, then there is no valid "state" which the object can assume (*Present_Value* is 0).

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the

FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

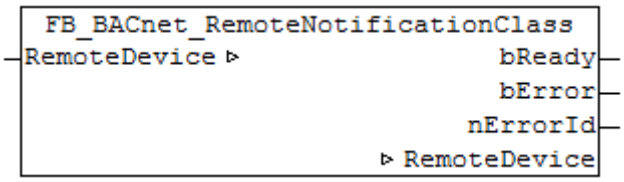
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.4.19 FB_BACnet_RemoteNotificationClass

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteNotificationClass" serves as placeholder for future functions.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
```

bReady: Notification of general readiness. If the output is *FALSE*, the associated function block FB_BACnet_RemoteDevice [▶ 459] reports "not operational".

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

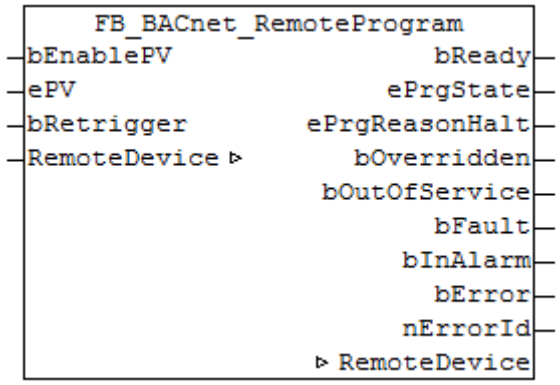
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.20 FB_BACnet_RemoteProgram

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteProgram" can be used for read and write access to a remote BACnet object of type *Program* (PROG). To this end the remote BACnet object was added to a local BACnet client.

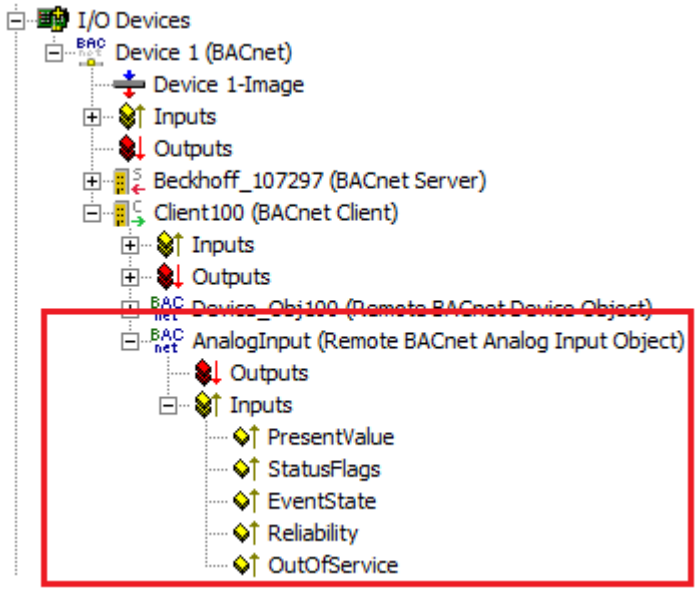


Fig. 173: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_INPUT

```
bEnablePV    : BOOL;
ePV          : E_BACnetProgramRequest;
bRetrigger   : BOOL; (* execute selected request on raising edge again *)
```

bEnablePV: Enables the value of input **ePV**. If the input is set to *TRUE*, the value of **ePV** is written to the property *Program_Change* of the object. If the input is set to *FALSE*, *0xFFFF* is written to the process data of the property *Program_Change* of the object. The value *0xFFFF* prevents writing to the remote server and therefore writing to the remote object.

ePV: Request to the remote program object. If **bEnablePV** was set to *TRUE*, the value of the input is written to the property *Program_Change* (see also Transition diagram).

bRetrigger: A change from *FALSE* to *TRUE* triggers writing of the input **ePV** to the property *Program_Change*, if input **bEnablePV** is set to *TRUE*.

VAR_OUTPUT

```
bReady           : BOOL;
ePrgState        : E_BACnetProgramState;
ePrgReasonHalt   : E_BACnetProgramError;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (ProgramState, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_RemoteDevice reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

ePrgState: Feedback of the current program state (see also Transition diagram).

ePrgChangeReq: Feedback on the state of the current program request (see also Transition diagram).

ePrgReasonHalt: Error feedback on program cancellation.

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Status_Flags*.

bProcessError, bOtherFault: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Program* and property *Reliability*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

Transition diagram

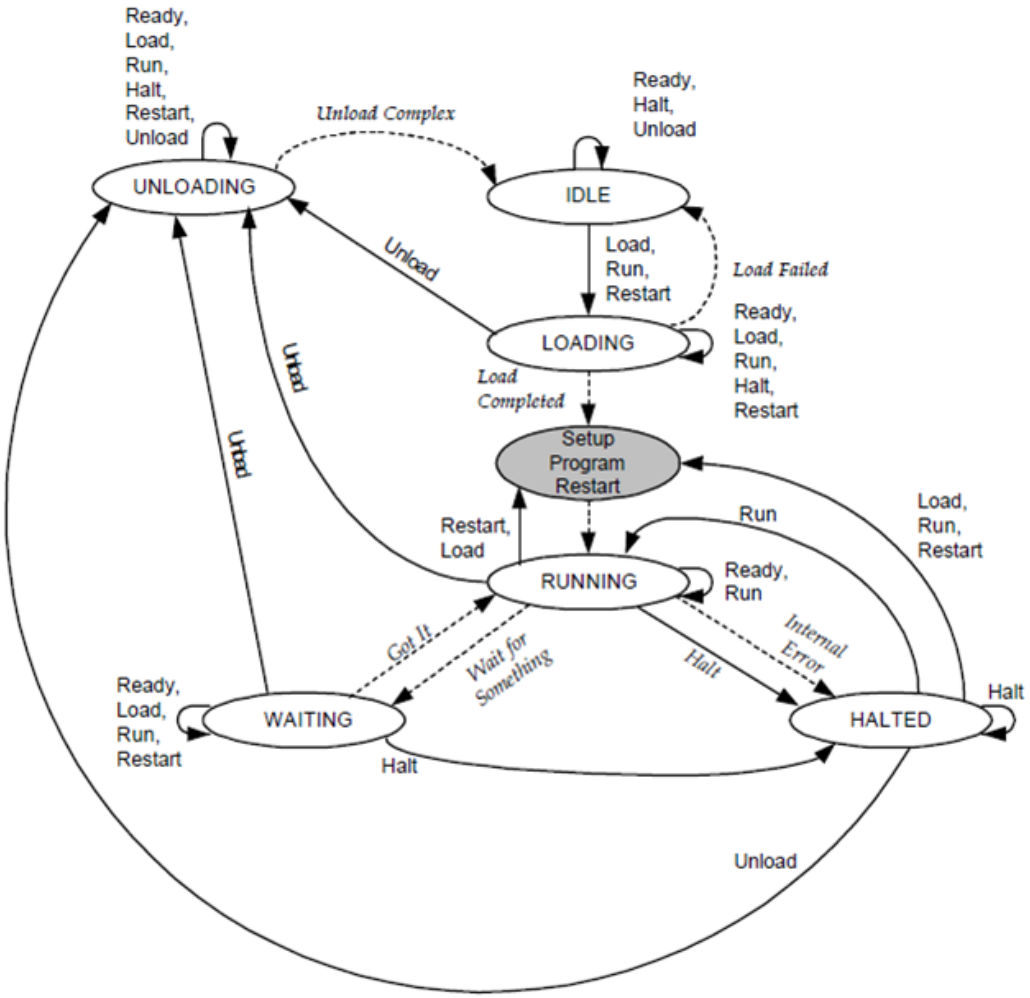
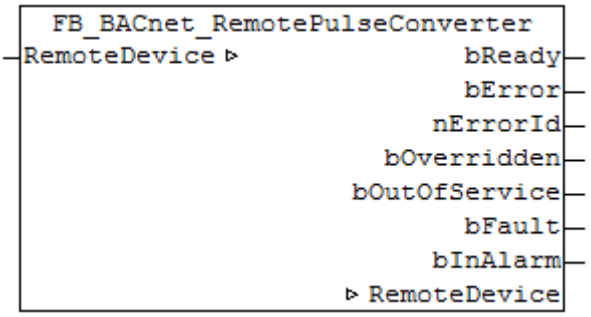


Fig. 174: Fig. 2: from BACnet specification DIN EN ISO 16484-5 for BACnet object Program, Fig. 12-3 "State Transitions for the program object"

5.4.21 FB_BACnet_RemotePulseConverter

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemotePulseConverter" can be used for read access to a remote BACnet object of type *PulseConverter* (PC). To this end the remote BACnet object was added to a local BACnet client.

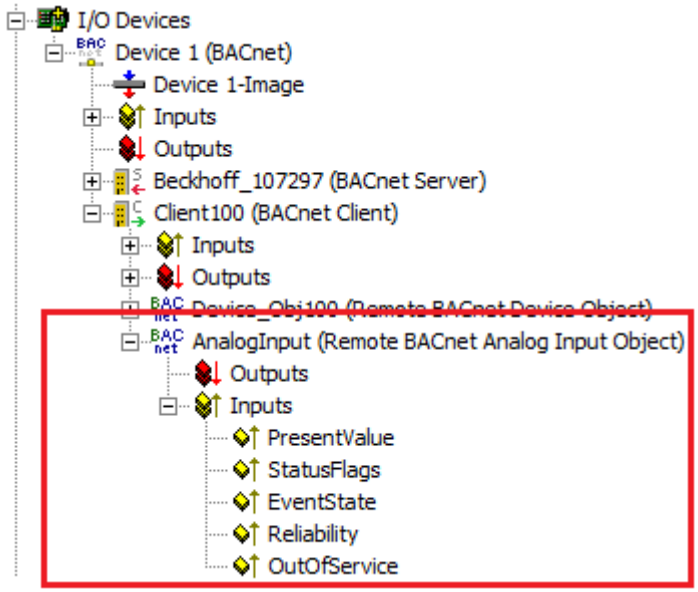


Fig. 175: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady           : BOOL;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
bError           : BOOL;
nErrorId         : UINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block FB_BACnet_RemoteDevice [▶ 459] reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

bOverride, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *PulseConverter* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

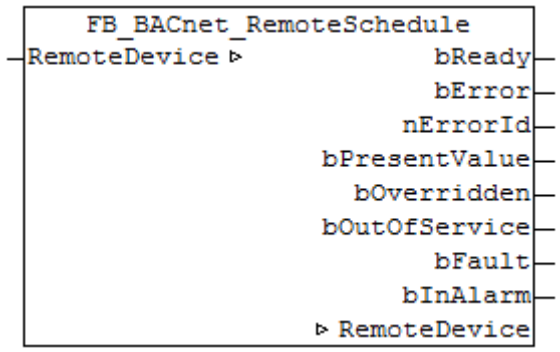
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.22 FB_BACnet_RemoteSchedule

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteSchedule" can be used for reading access to a remote BACnet object of type *Schedule* (SCHED). To this end the remote BACnet object was added to a local BACnet client.

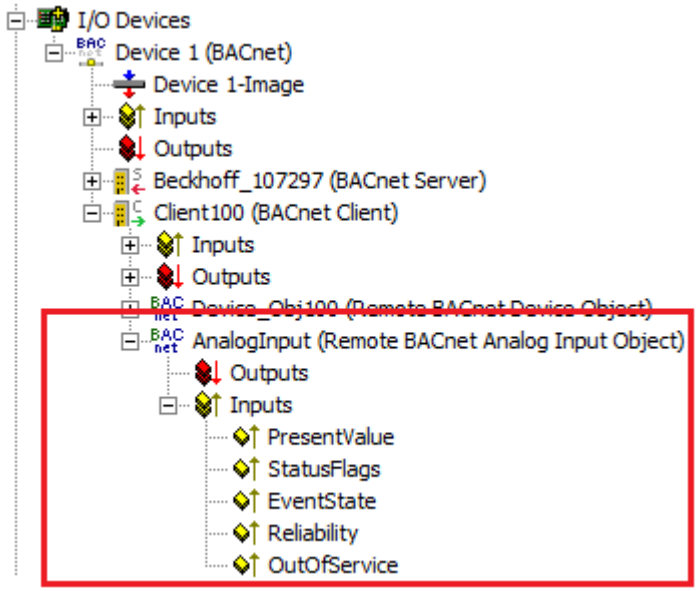


Fig. 176: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady           : BOOL;
bError           : BOOL;
nErrorId         : UINT;
bPresentValue    : BOOL;
bOverridden      : BOOL;
bOutOfService    : BOOL;
bFault           : BOOL;
bInAlarm         : BOOL;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (PresentValue, Overridden ...). If the output is *FALSE*, the corresponding function block "FB_BACnet_RemoteDevice" reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

bPresentValue: Current value of the BACnet object (see also BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Present_Value*). The state is specified in 3 stages (enumeration) (0 = INACTIVE, 1 = ACTIVE and 2 = ZERO).

bOverridden, bOutOfService, bFault, bInAlarm: See BACnet specification DIN EN ISO 16484-5 for BACnet object *Schedule* and property *Status_Flags*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice [▶ 459])
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

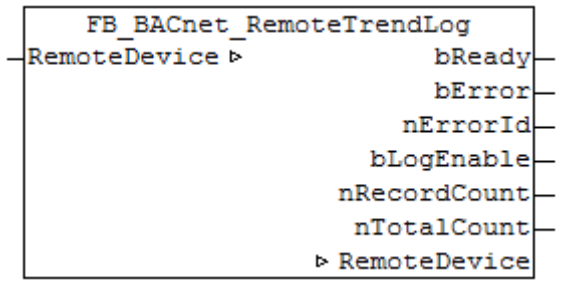
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter [▶ 375] and FB_BACnet_RemoteDevice [▶ 459] for further information.

5.4.23 FB_BACnet_RemoteTrendLog

The following function block is used for linking a remote BACnet object of the local BACnet client. The function block for the corresponding BACnet object is linked with the aid of process data. The data exchange with the remote BACnet server takes place via BACnet with the aid of WOC (Write-On-Change) and COV (Changes-On-Value) or via polling (not recommended).

The process data can be created manually in the BACnet object, linked manually, or they can be generated automatically via PLC automapping [▶ 62]. The comments required for PLC automapping [▶ 62] ((* ~ (BACnet... | ??? | ???) *)) are already included in the declaration of the function block.



Use

The function block "FB_BACnet_RemoteTrendLog" can be used for read access to a remote BACnet object of type *TrendLog* (TREND). To this end the remote BACnet object was added to a local BACnet client.

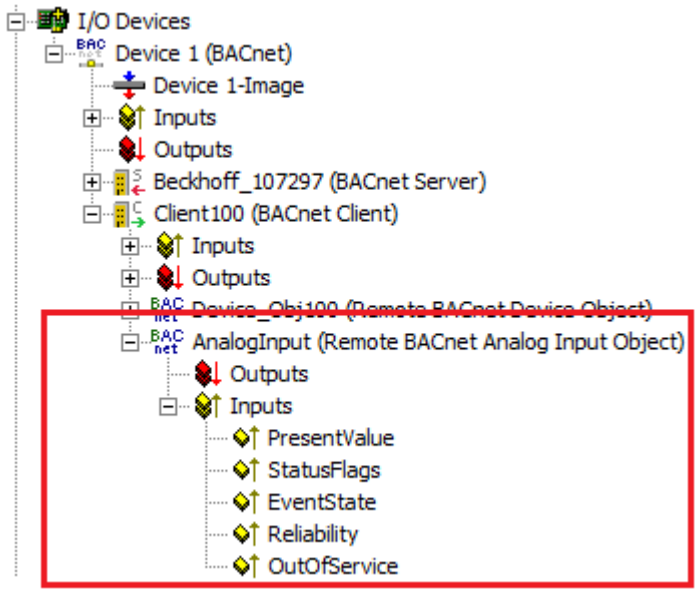


Fig. 177: Fig. 1: Example for a remote BACnet object under a local BACnet client.

VAR_OUTPUT

```
bReady       : BOOL;
bError       : BOOL;
nErrorId     : UINT;
bLogEnable   : BOOL;
nRecordCount : UDINT;
nTotalCount  : UDINT;
```

bReady: Notification of general readiness. If this output is set, the other status outputs are valid (LogEnable, RecordCount...). If the output is *FALSE*, the corresponding function block FB_BACnet_RemoteDevice reports "not operational", the block instances was not linked correctly in the System Manager or the remote server cannot be reached (gateway cannot be reached, no Ethernet link).

bLogEnable: See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Log_Enable*.

nRecordCount: See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Record_Count*.

nTotalCount: See BACnet specification DIN EN ISO 16484-5 for BACnet object *TrendLog* and property *Total_Record_Count*.

bError: An error is pending.

nErrorId: Error number
0 = no error
1 = function block of the corresponding client (RemoteDevice) is not called or not called regularly enough from the PLC program.
2 = incorrect process data mapping detected (check mapping in the System Manager; if necessary compile complete PLC project and reload)
3 = the corresponding BACnet client is not ready (**bOperational** = *FALSE* at instance of the FB_BACnet_RemoteDevice)
The error numbers can be queried as block constants via the FB instance (*FB_BACnet_Remote???*.**nERR_xxx**).

VAR_IN_OUT

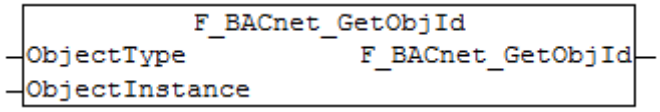
```
RemoteDevice     : FB_BACnet_RemoteDevice;
```

RemoteDevice: Block instance of the corresponding remote BACnet device object. The remote BACnet device object of a remote BACnet server has been added under a local BACnet client. Local client and remote server are linked via BACnet. Any number of clients can be linked via BACnet adapter. See FB_BACnet_Adapter and FB_BACnet_RemoteDevice for further information.

5.5 BACnet Helper

5.5.1 F_BACnet_GetObjId

Function for coding the object type and the object instance in the BACnet *Object_Identifier*.



VAR_INPUT

```
ObjectType       : E_BACnetObjectType;
ObjectInstance   : UDINT;
```

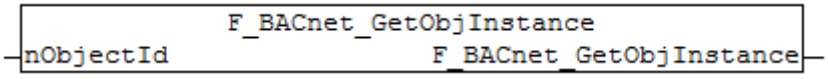
ObjectType: BACnet object type.

ObjectInstance: BACnet object number (instance, 0...4'194'303).

Return value is the resulting BACnet *Object_Identifier*.

5.5.2 F_BACnet_GetObjInstance

Function for decoding the BACnet *Object_Identifier* in the object instance (object number).



VAR_INPUT

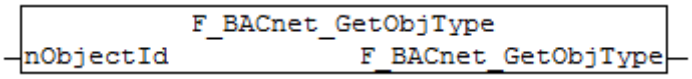
```
nObjectId    : DWORD;
```

nObjectId: BACnet *Object_Identifier*.

Return value is the BACnet object number (instance, 0...4'194'303).

5.5.3 F_BACnet_GetObjType

Function for decoding the BACnet *Object_Identifier* in the object type.



VAR_INPUT

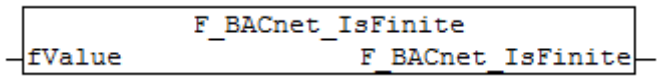
```
nObjectId    : DWORD;
```

nObjectId: BACnet *Object_Identifier*.

Return value is the resulting BACnet object type (see E_BACnetObjectType [▶ 507]).

5.5.4 F_BACnet_IsFinite

Function for checking a floating-point number for finite value range.



VAR_INPUT

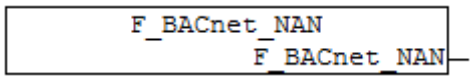
```
fValue       : REAL;
```

fValue: Floating-point number to be checked.

Return value: *TRUE* = value is within the finite value range, *FALSE* = number infinitely positive or infinitely negative.

5.5.5 F_BACnet_NAN

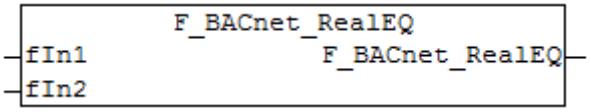
Function returns the floating-point value "**N**ot **AN**umber". This can be used for deleting priority entries in the *Priority_Array* of an *Analog_Value* or *Analog_Output* object.



Return value: "Not a number" or **#QNAN** (0x7FFF0000).

5.5.6 F_BACnet_RealEQ

Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("=" or "EQ") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range.



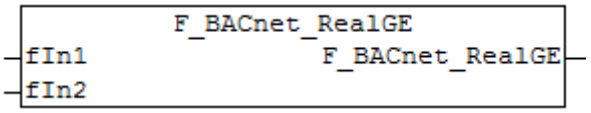
VAR_INPUT

```
fIn1         : REAL;
fIn2         : REAL;
```

Return value: *TRUE* = value 1 and value 2 are identical or both are not in the finite value range. *FALSE* = value 1 and value 2 differ.

5.5.7 F_BACnet_RealGE

Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators (">" or "GE") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range.



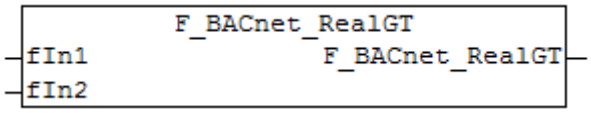
VAR_INPUT

```
fIn1         : REAL;
fIn2         : REAL;
```

Return value: *TRUE* = value 1 is equal or greater than value 2. *FALSE* = value 2 is smaller than value 1, or one of the two values is not in the finite value range.

5.5.8 F_BACnet_RealGT

Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators (">" or "GT") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range.



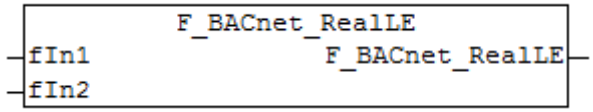
VAR_INPUT

```
fIn1          : REAL;
fIn2          : REAL;
```

Return value: *TRUE* = value 1 is greater than value 2. *FALSE* = value 2 is smaller or equal value 1, or one of the two values is not in the finite value range.

5.5.9 F_BACnet_RealLE

Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("<" or "LE") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range.



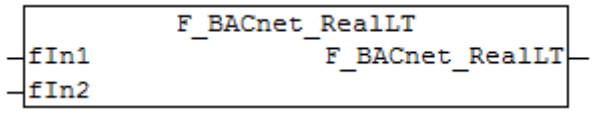
VAR_INPUT

```
fIn1          : REAL;
fIn2          : REAL;
```

Return value: *TRUE* = value 1 is smaller or equal value 2. *FALSE* = value 2 is greater than value 1, or one of the two values is not in the finite value range.

5.5.10 F_BACnet_RealLT

Function for comparing two floating-point values taking into account the value range. This function should be used instead of the standard operators ("<" or "LT") for values from BACnet, since otherwise a PLC stop is triggered if the values to be compared are not in the finite range.



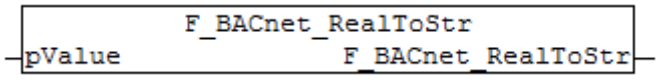
VAR_INPUT

```
fIn1          : REAL;
fIn2          : REAL;
```

Return value: *TRUE* = value 1 is smaller than value 2. *FALSE* = value 2 is greater or equal value 1, or one of the two values is not in the finite value range.

5.5.11 F_BACnet_RealToStr

Function for converting a floating-point number to a string, taking into account the value range.



VAR_INPUT

```
pValue        : POINTER TO REAL;
```

pValue: Pointer to the floating-point number to be converted ("ADR()").

Return value: For a finite value range the string from the floating-point number is output in the same format as after conversion with "REAL_TO_STRING()". If the value is in the infinite range, "#QNAN" (positive infinite) or "-#QNAN" (negative infinite) is output, depending on the sign.

Example

```
ADR(fValue)
```

```
F_BACnet_RealToStr
pValue                F_BACnet_RealToStr
```

Fig. 178: BACnetSPS_RealToStrBeispiel

Fig. 1: Example for pointer determination with "ADR()". "fValue" represents a variable of type REAL.

5.5.12 FB_BACnet_GetDiagInfo

Function block for access to BACnet diagnosis via ADS.

```
FB_BACnet_GetDiagInfo
sNetId              bBusy
bExecute            bError
tTimeout            nErrID
                    stInfo
```

Use

The function block instance is created by the PLC program and called cyclically.

The required NetID (AMS NetID) of the BACnet device can be read in the System Manager (see Figure 1: The NetID does not match the local NetID and always has to be specified - empty strings cannot be used!). Another possibility is to query the NetID from the device function block or adapter function block in the PLC (see Figure 2).

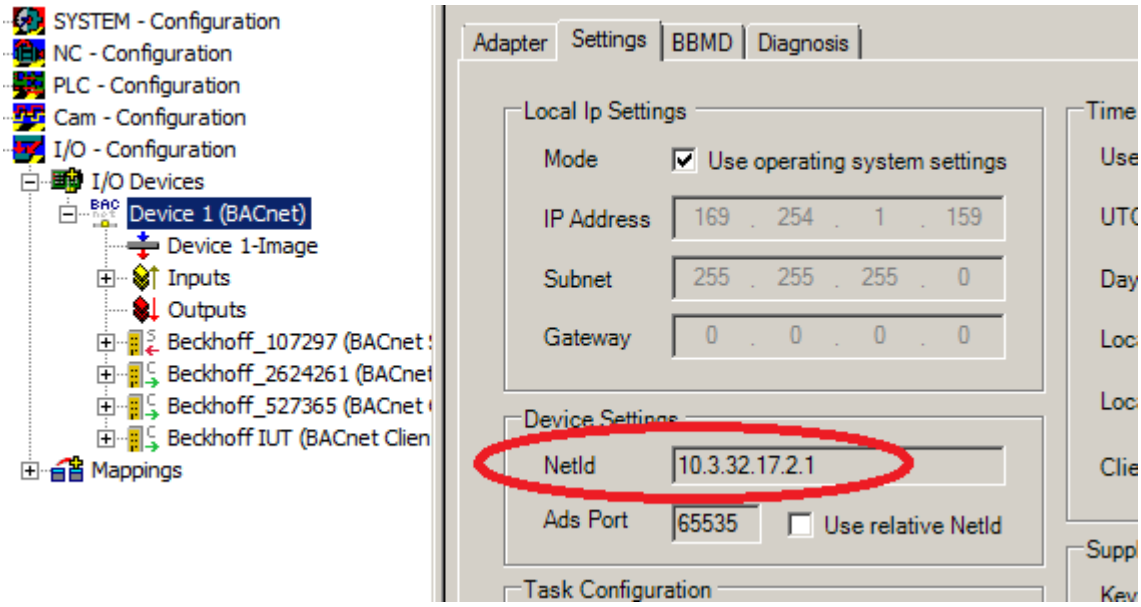


Fig. 179: Figure 1: AMS NetID of the BACnet device

For information on the operating principle of the adapter function block see FB_BACnet_Adapter [▶ 375].

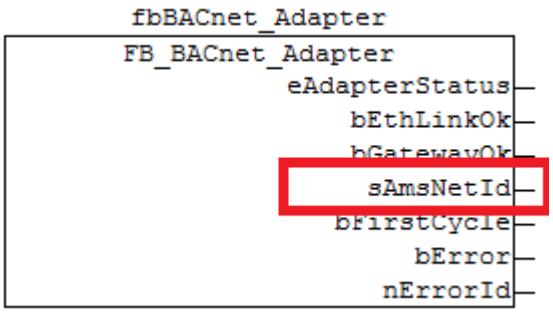


Fig. 180: Figure 2: AMS NetID of the BACnet device

VAR_INPUT

```
sNetId       : T_AmsNetId;
bExecute     : BOOL;
tTimeout     : TIME := tBACnet_ADSTimeOut;
```

sNetId: AMS NetID of the BACnet device (adapter) under which the server or client was configured.

bExecute: Rising edge at the input starts the read process.

tTimeout: Optional input, monitoring time for ADS access (default: tBACnet_ADSTimeOut [▶ 511]).

VAR_OUTPUT

```
bBusy        : BOOL;
bError       : BOOL;
nErrID       : UDINT;
stInfo       : ST_BACnet_Diagnosis;
```

bBusy: The block is busy.

bError: Error during processing.

nErrID: ADS error code.

stInfo: Structure with diagnostic information on BACnet.

Example

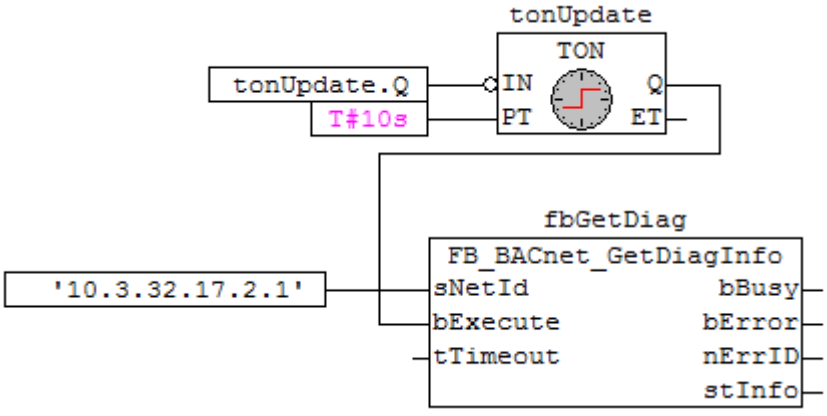
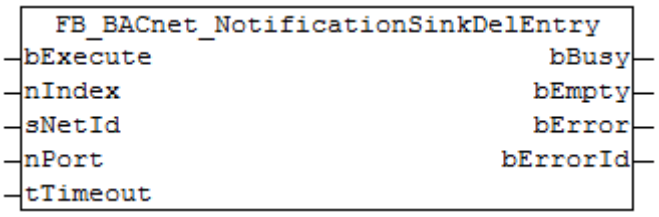


Fig. 181: Fig. 3: Example for cyclic reading of the diagnostic information

5.5.13 FB_BACnet_NotificationSinkDelEntry

Function block for deleting entries from the "notification sink" via ADS.



Use

The function block instance is created by the PLC program and called cyclically. The input **nPort** is assigned with the ADS port from the System Manager configuration (see Figure 1). Ports may vary, depending on the configuration.

The function block can be used to delete entries from anywhere in the "Notification Sink". If all entries were removed or the specified index is greater than the number of list entries, then the function block returns **bEmpty**.

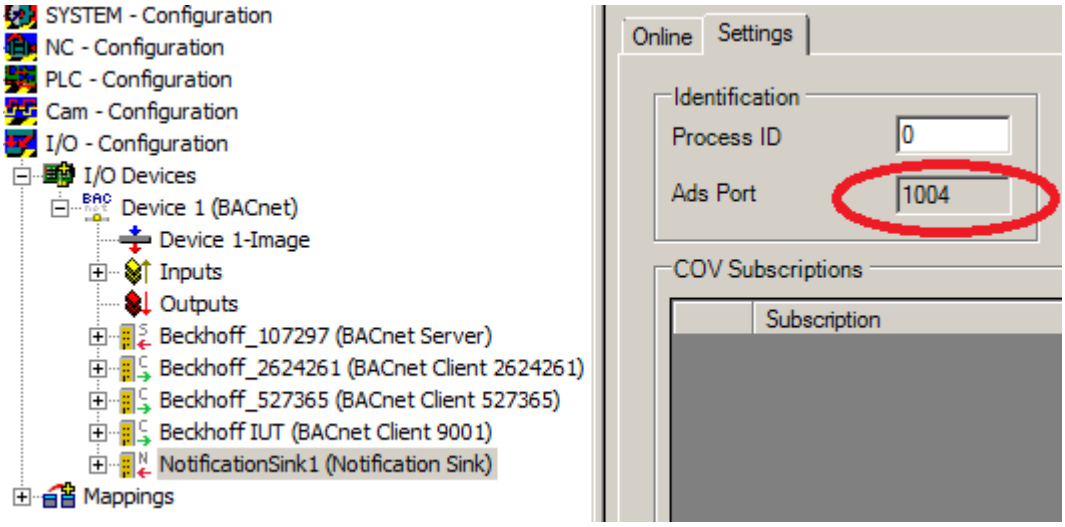


Fig. 182: Figure 1: ADS port in the System Manager

The required NetId (AMS NetId) of the BACnet device can be read in the System Manager (see Figure 2).

The NetID does not match the local NetID and always has to be specified - empty strings cannot be used!). Another possibility is to query the NetID from the device function block or adapter function block in the PLC (see Figure 3).

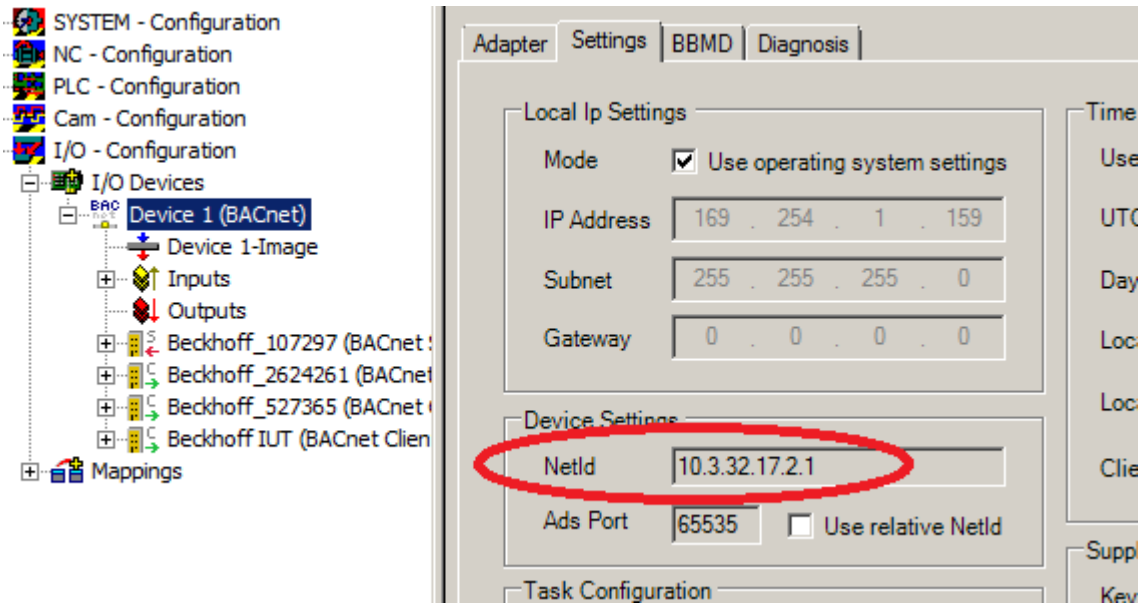


Fig. 183: Figure 2: AMS NetID of the BACnet device

For information on the operating principle of the adapter function block see FB_BACnet_Adapter [▶ 375].

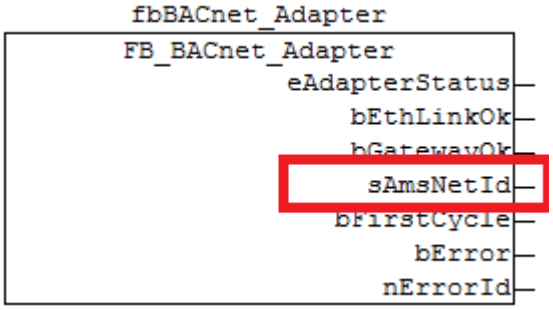


Fig. 184: Figure 3: AMS NetID of the BACnet device

VAR_INPUT

```
bExecute     : BOOL;
nIndex       : UDINT;
sNetId       : T_AmsNetId;
nPort        : T_AmsPort;
tTimeout     : TIME := tBACnet_ADSTimeOut;
```

bExecute: Rising edge at the input starts the read process.

nIndex: Index of the entry to be deleted (0 ... n-1).

sNetId: AMS NetID of the BACnet device (adapter) under which the server or client was configured.

nPort: ADS port under which the "notification sink" was created (see also under Use [▶ 490]).

tTimeout: Optional input, monitoring time for ADS access (default: tBACnet_ADSTimeOut [▶ 511]).

VAR_OUTPUT

```
bBusy        : BOOL;
bEmpty       : BOOL;
bError       : BOOL;
nErrorId     : UDINT;
```

bBusy: the function block is busy.

bEmpty: is set, if an attempt is made to delete a non-existing entry.

Once the last entry (index 0) has been deleted, the **bEmpty** message is only sent if another attempt is made to delete an entry (index 0) from the empty list.

bError: error during processing.

nErrorId: ADS error code.

Example

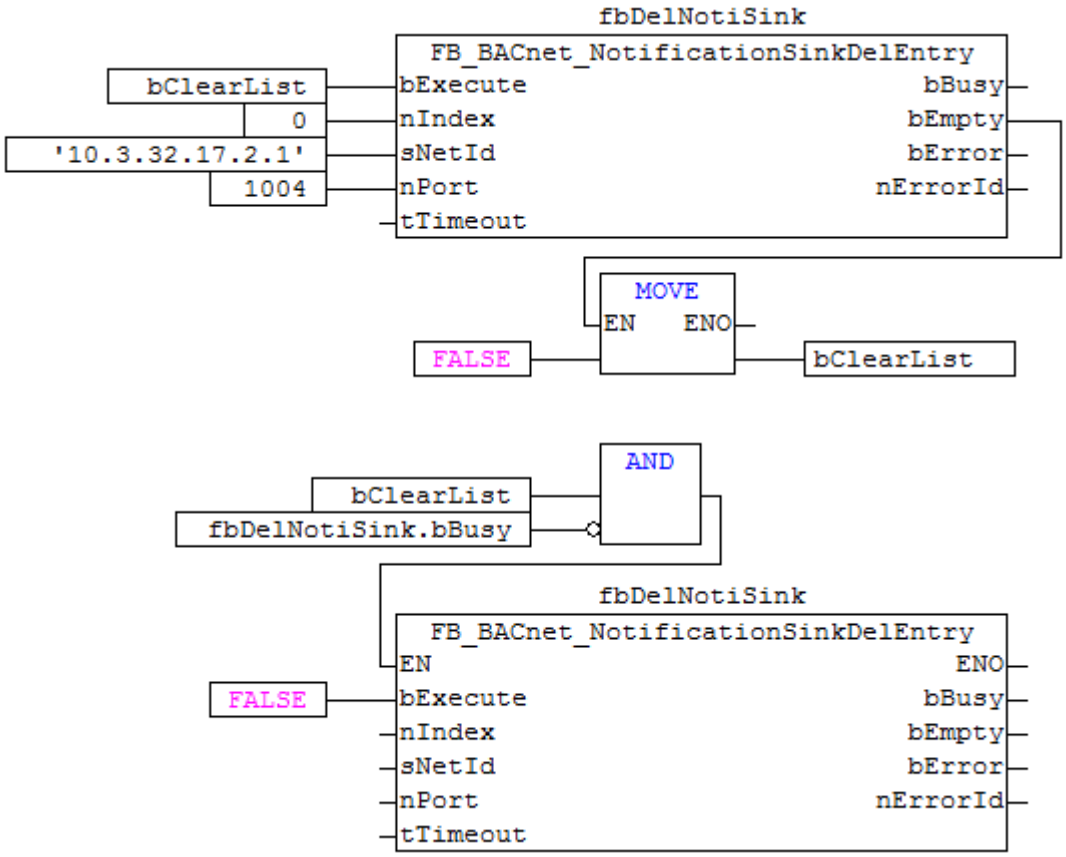
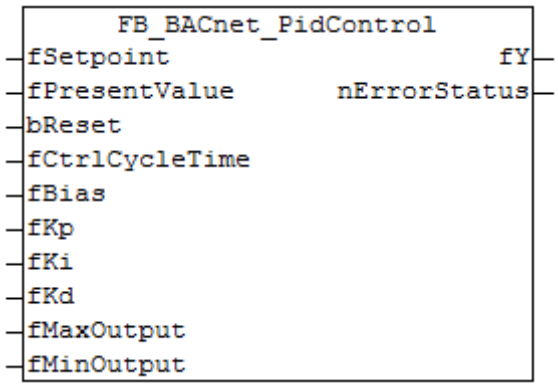


Fig. 185: Fig. 4: Example for deleting the whole "notification sink" under port 1004.

If the bit "bClearList" is set, deleting of entry "0" is repeated until the block reports **bEmpty**. The bit "**bClearList**" is then reset automatically.

5.5.14 FB_BACnet_PidControl

PID controller block in parallel configuration (Kp has no effect on I and D component). The controller block serve as a basis for the object "LOOP" (FB_BACnet_Loop [▶ 420]).



Block diagram

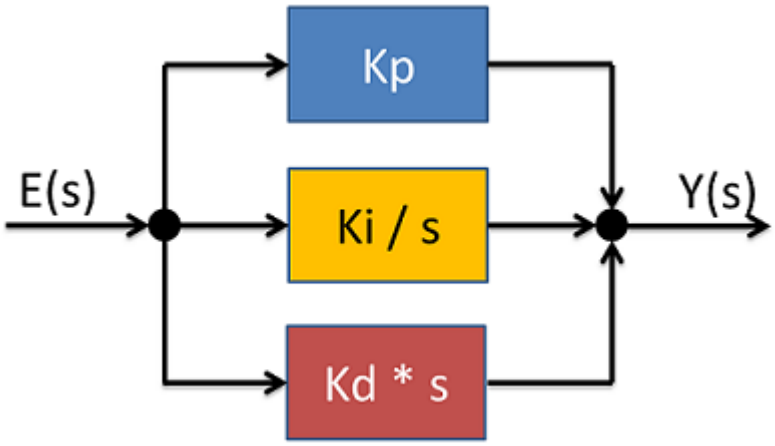


Fig. 186: Fig. 1: Block diagram of the transfer function (frequency range).

Use

The controller block must be called cyclically in the PLC program. Please ensure that the cycle time of the call at input **fCtrlCycleTime** is transferred as precisely as possible. The cycle time can be determined from the global structure "SystemTaskInfoArr", for example. A possible controller configuration is presented under Example [▶ 494].

VAR_INPUT

```
fSetpoint         : REAL; (* setpoint value *)
fPresentValue     : REAL; (* actual value *)
bReset            : BOOL; (* reset flag *)
fCtrlCycleTime    : REAL; (* controller cycle time in seconds [s] *)
fBias             : REAL; (* bias also active when reset is set to 1 *)
fKp               : REAL; (* proportional gain Kp (P) *)
fKi               : REAL; (* integral gain Ki (I) *)
fKd               : REAL; (* derivative gain Kd (D) *)
fMaxOutput        : REAL;
fMinOutput        : REAL;
```

fSetpoint: setpoint value input (e.g. temperature specification for a room in [°C])

fPresentValue: actual value (e.g. actual room temperature in [°C])

bReset: resets the controller, the output is set to **fBias** for the duration and limited through **fMaxOutput** and **fMinOutput**.

If **fMinOutput** is set to 5% and **fBias** to 0%, for example, the output **fY** is set to 5% for the duration of the reset.

fCtrlCycleTime: cycle time with which the controller is called in seconds [s].

fBias: output offset. This value is offset with the output. The unit must match the unit of the control value (e.g. heating capacity in [%]).

fKp: gain factor P. The control deviation is multiplied with P and added to the output (output behaves proportional to the input deviation).

fKi: integral gain I. The control deviation is multiplied with I (taking into account the cycle time), added to the previous result and added to the output (high control deviation = fast increasing output).

fKd: differential gain D. The control deviation is offset with the previous controller difference, the result multiplied with D (taking into account the cycle time) and added to the output (strong controller variation = strong response).

fMaxOutput: upper limit of the output value **fY**. The unit must match the unit of the control value (e.g. heating capacity maximum 90%).

fMinOutput: lower limit of the output value **fY**. The unit must match the unit of the control value (e.g. heating capacity minimum 5%).

VAR_OUTPUT

```
fY           : REAL; (* controller output command *)
nErrorStatus     : UINT; (* controller error output (0: no error; >0:error) *)
```

fY: Control value (e.g. heating capacity in [%]).

nErrorStatus: Error status (*0 = no error*, 1 = invalid parameters: 2 = invalid cycle time). Error messages do not have to be acknowledged.

Example

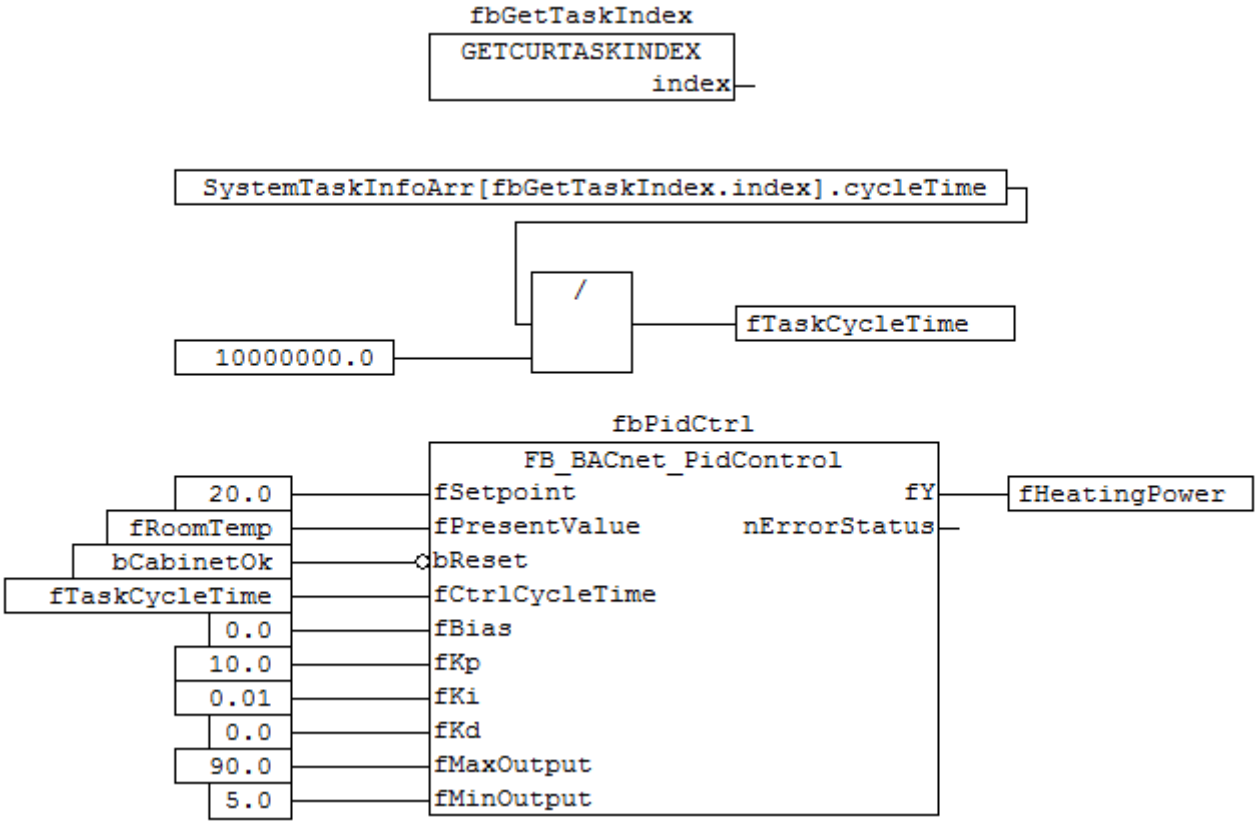


Fig. 187: Figure 4: Application example

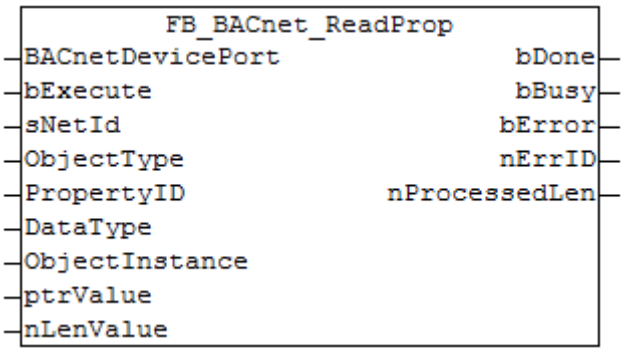
Note re. P = 10 --> a temperature difference of 1 K results in 10% heating capacity, for example

Note re. I = 0.01 --> a temperature difference of 1 K results in 0.01% heating output increase per second, for example

Note re. D = 0 --> no D part.

5.5.15 FB_BACnet_ReadProp

Function block for access to BACnet properties via ADS. All BACnet properties of server and client objects can be read or written via ADS. If a client is accessed via ADS, the ADS accesses are converted to "Read_Property" or "Read_Range" BACnet requests and sent to the remote server.



Use

The function block instance is created by the PLC program and called cyclically. The input **BACnetDevicePort** is assigned with the ADS port from the System Manager configuration (see Figure 1). Ports for servers and clients may vary, depending on the configuration.

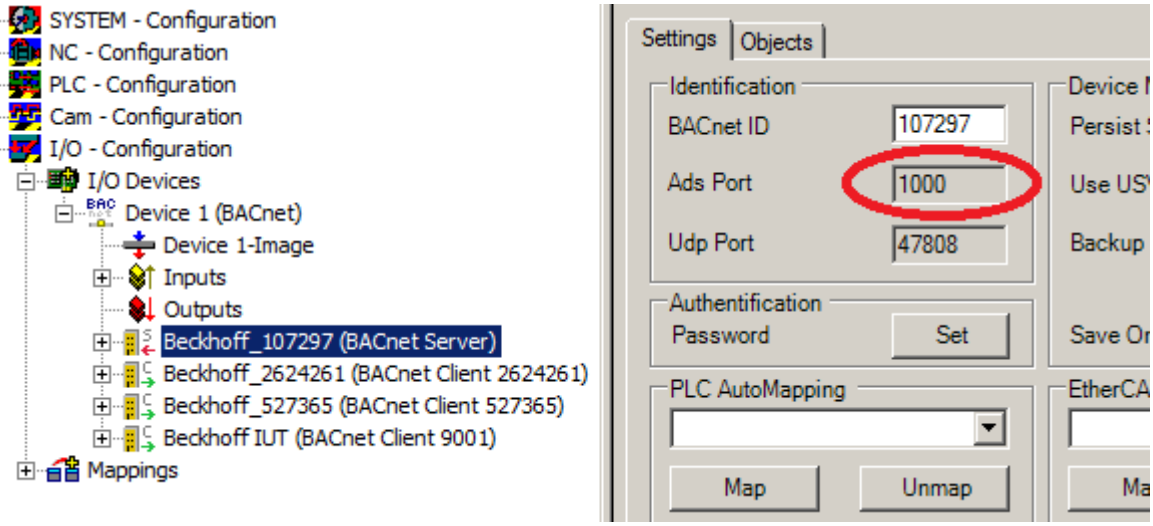


Fig. 188: Figure 1: ADS port in the System Manager

The NetId (AMS NetId) of the BACnet device can also be read in the System Manager (see Figure 2).

The NetID does not match the local NetID and always has to be specified - empty strings cannot be used!). Another possibility is to query the NetID from the device function block or adapter function block in the PLC (see Figure 3).

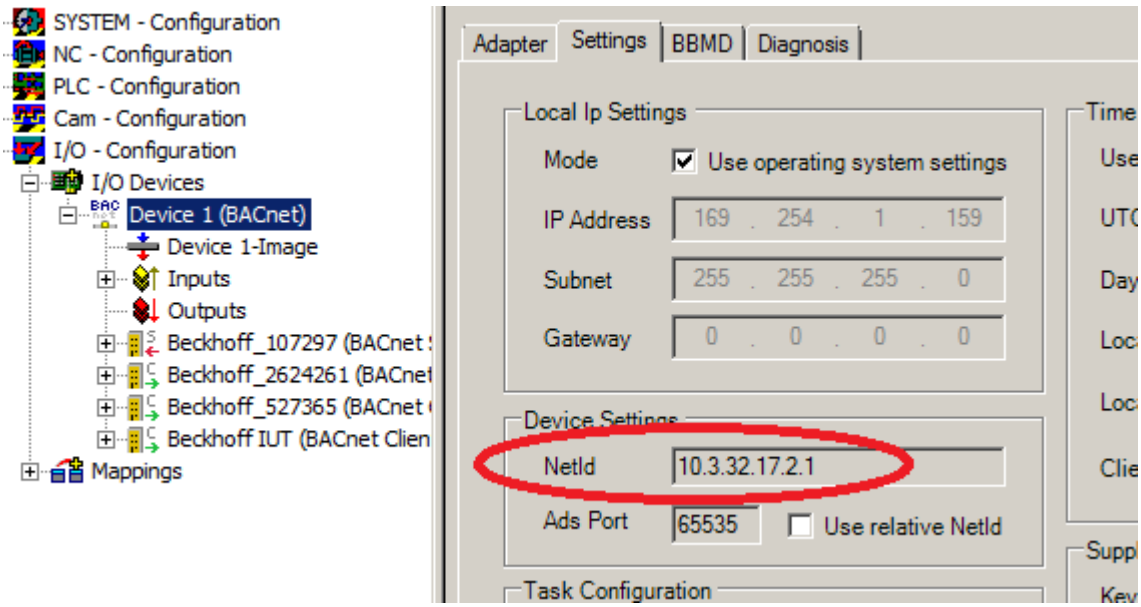


Fig. 189: Figure 2: AMS NetID of the BACnet device

For information on the operating principle of the adapter function block see FB_BACnet_Adapter [▶ 375].

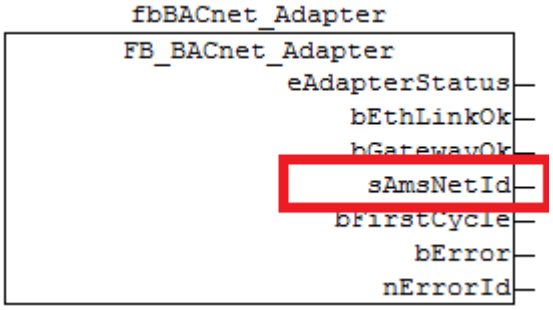


Fig. 190: Figure 3: AMS NetID of the BACnet device

VAR_INPUT

```
BACnetDevicePort : UINT := 16#FFFF;
bExecute         : BOOL;
sNetId           : T_AmsNetId;
ObjectType       : E_BACnetObjectType;
PropertyID       : E_BACnetPropertyIdentifier;
DataType         : E_BACnetDataTypes;
ObjectInstance   : DINT;
ptrValue         : POINTER TO BYTE;
nLenValue        : UDINT; (* byte size of value (SIZEOF) *)
```

BACnetDevicePort: ADS port under which the required object was created (see also under use).

bExecute: Rising edge at the input starts the read process. Falling edge deletes the output **bDone**.

sNetId: AMS NetID of the BACnet device (adapter) under which the server or client was configured.

ObjectType: Enumeration of the object type (AnalogValue, BinaryInput...).

PropertyID: Enumeration of the property ID (Present_Value, Status_Flags...).

ObjectInstance: Object number of the BACnet object (the lower 22 bits of the Object_Identifier).

ptrValue: Pointer to the variable in which the read data are to be stored (can be determined with ADR()).

nLenValue: Byte length of the variable in which the read data are to be stored (can be determined with SIZEOF()).

VAR_OUTPUT

```
bDone         : BOOL;
bBusy         : BOOL;
bError        : BOOL;
nErrID        : UDINT;
nProcessedLen: UDINT; (* the received data length (can be different to given length) *)
```

bDone: Read of the data completed successfully. **bDone** remains set until **bExecute** is reset. If **bExecute** is reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrID: ADS error code.

nProcessedLen: Byte length of the read data. The length may differ from the target length **nLenValue** (smaller/equal).

Example

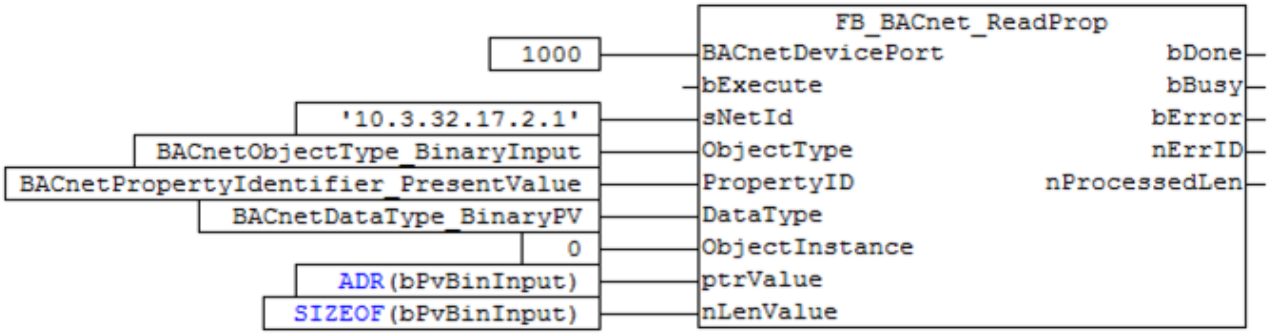
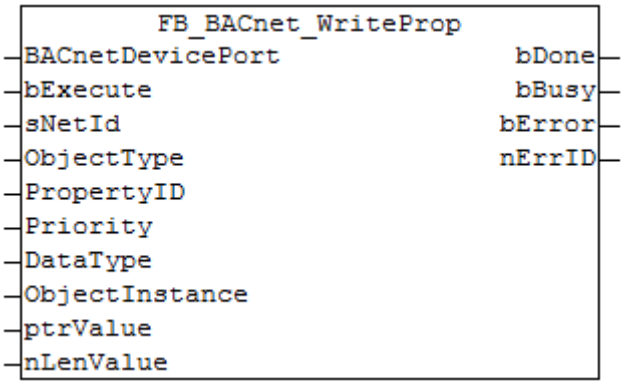


Fig. 191: Fig. 4: Application example

5.5.16 FB_BACnet_WriteProp

Function block for access to BACnet properties via ADS. All BACnet properties of server and client objects can be read or written via ADS. If a client is accessed via ADS, the ADS accesses are converted to "Write_Property" BACnet requests and sent to the remote server.



Use

The function block instance is created by the PLC program and called cyclically. The input **BACnetDevicePort** is assigned with the ADS port from the System Manager configuration (see Figure 1). Ports for servers and clients may vary, depending on the configuration.

By using the WriteProp function block for accessing the Present_Value, for example, WOC (Write On Change) to local BACnet server objects can be realized. In contrast to process data mapping the priority of the write access can be adjusted online. In addition, process data mapping to local server objects results in cyclic writing of the data, so that remote clients do not have access with the same priority as the local PLC.

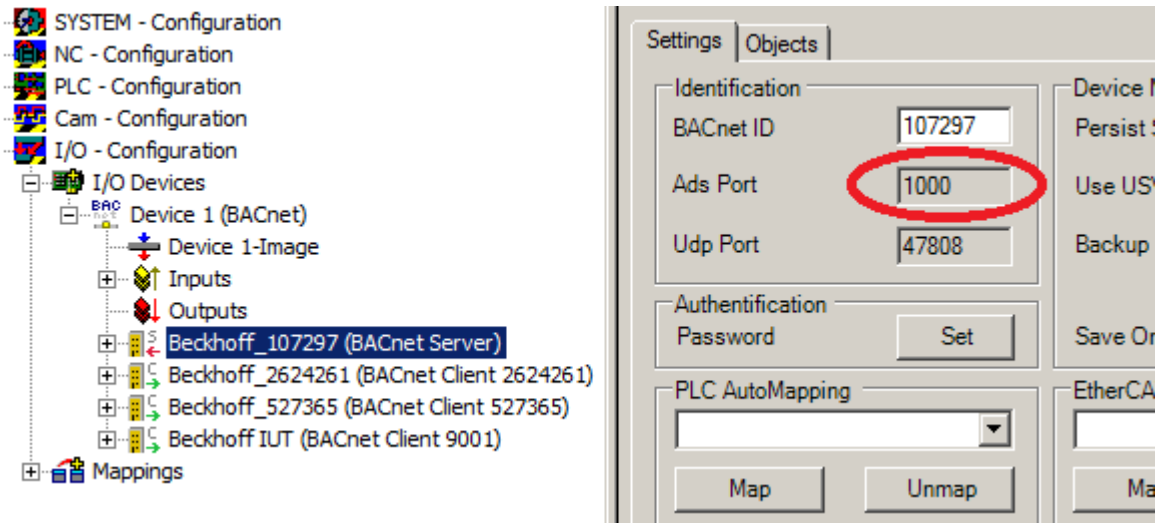


Fig. 192: Figure 1: ADS port in the System Manager

The NetId (AMS NetId) of the BACnet device can also be read in the System Manager (see Figure 2).

The NetID does not match the local NetID and always has to be specified - empty strings cannot be used!). Another possibility is to query the NetID from the device function block or adapter function block in the PLC (see Figure 3).

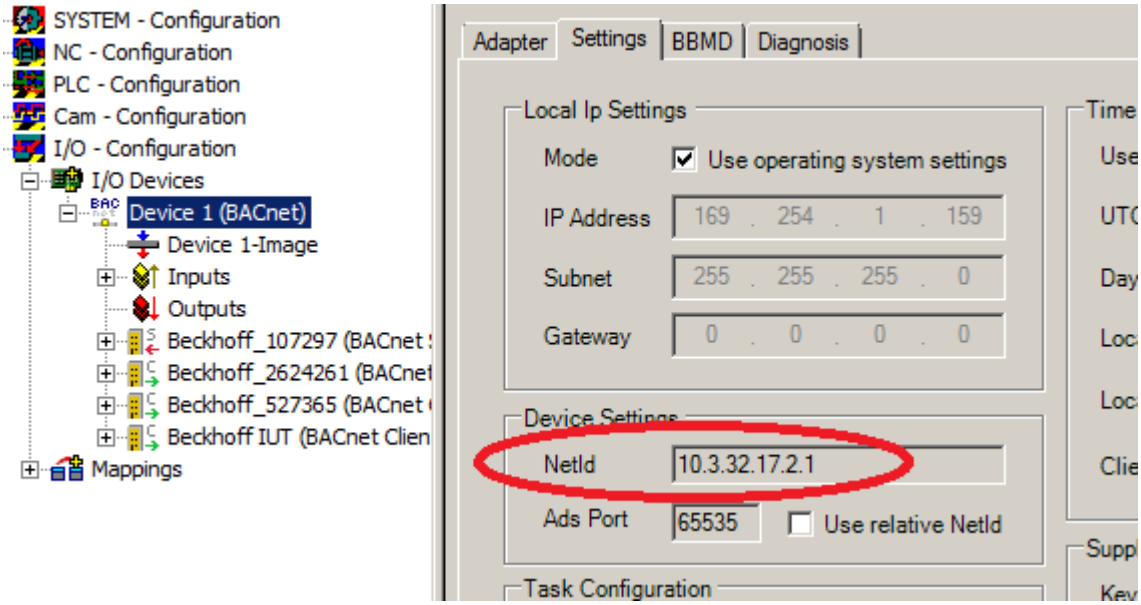


Fig. 193: Figure 2: AMS NetID of the BACnet device

For information on the operating principle of the adapter function block see FB_BACnet_Adapter [▶ 375].

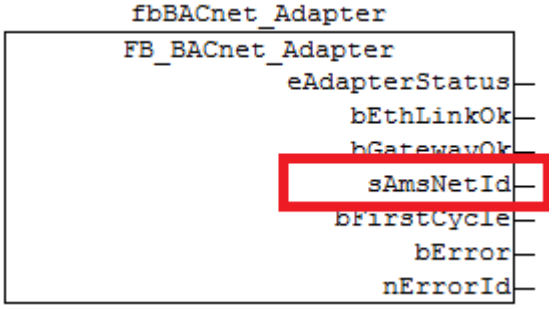


Fig. 194: Figure 3: AMS NetID of the BACnet device

VAR_INPUT

```
BACnetDevicePort : UINT := 16#FFFF;
bExecute         : BOOL;
sNetId           : T_AmsNetId;
ObjectType       : E_BACnetObjectType;
PropertyID       : E_BACnetPropertyIdentifier;
Priority         : E_BACnetPriority;
DataType         : E_BACnetDataTypes;
ObjectInstance   : UDINT;
ptrValue         : POINTER TO BYTE;
nLenValue        : UDINT; (* byte size of value (SIZEOF) *)
```

BACnetDevicePort: ADS port under which the required object was created (see also under use).

bExecute: Rising edge at the input starts the read process. Falling edge deletes the output **bDone**.

sNetId: AMS NetID of the BACnet device (adapter) under which the server or client was configured.

ObjectType: Enumeration of the object type (AnalogValue, BinaryInput...).

PropertyID: Enumeration of the property ID (Present_Value, Status_Flags...).

Priority: Enumeration of the write access priority. The priority is required for write accesses to prioritisable properties (e.g. Present_Value). Access with priority x then generates an entry at location x in the corresponding Priority_Array.

ObjectInstance: Object number of the BACnet object (the lower 22 bits of the Object_Identifier).

ptrValue: Pointer to the variable from which the data to be written are taken (can be determined with ADR()).

nLenValue: Byte length of the variable from which the data to be written are taken (can be determined with SIZEOF()).

VAR_OUTPUT

```
bDone        : BOOL;
bBusy        : BOOL;
bError       : BOOL;
nErrID       : UDINT;
```

bDone: Writing of the data completed successfully. **bDone** remains set until **bExecute** is reset. If **bExecute** is reset before **bDone** is active, **bDone** is set for one cycle.

bBusy: The block is busy.

bError: Error during processing.

nErrID: ADS error code.

Example

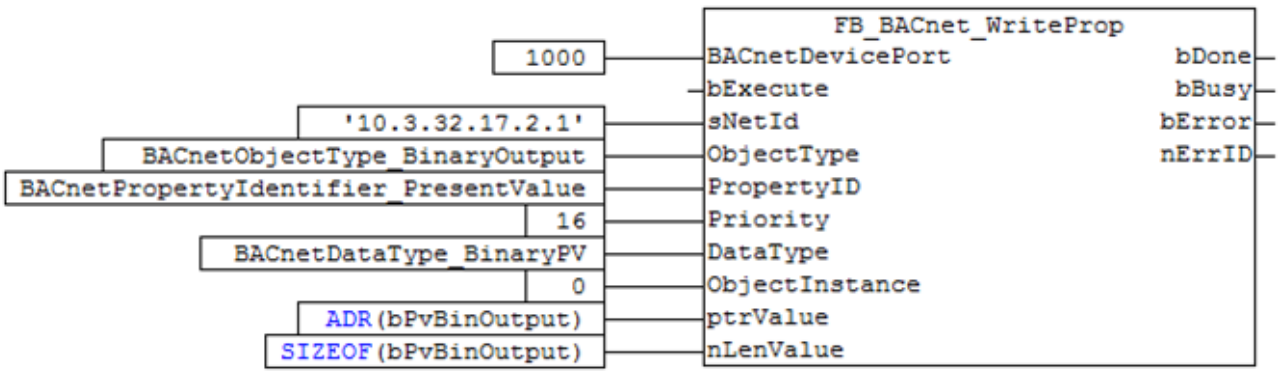


Fig. 195: Fig. 4: Application example

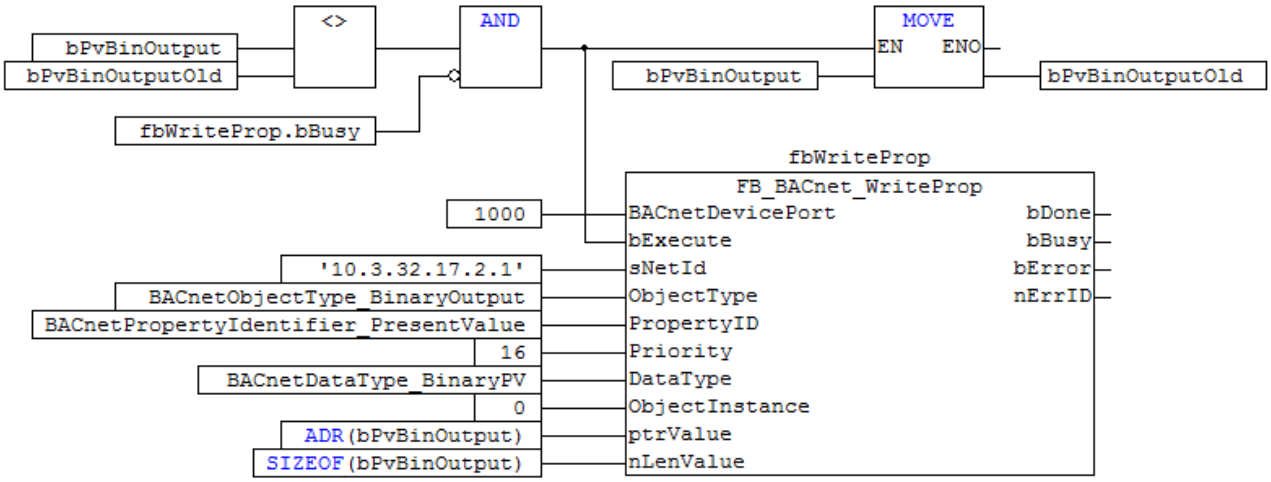


Fig. 196: Fig. 5: Example for Write on Change (WOC)

5.6 BACnet Datatypes

5.6.1 ST_BACnet_Date

Structure with date information.

```
nYear        : BYTE; (* year minus 1900 X'FF' = unspecified *)
nMonth       : BYTE; (* (1.. 12) 1 = January X'FF' = unspecified *)
nDay         : BYTE; (* day OF month (1..31), X'FF' = unspecified *)
nDayOfWeek   : BYTE; (* day OF week (1..7) 1 = Monday -- 7 = Sunday -- X'FF' = unspecified *)
```

nYear: year number minus 1900 (0 ... 254 = 1900 ... 2154, 0xFF = not specified).

nMonth: month number (1...12, 1 = January, 0xFF = not specified).

nDay: day of the month (1...31, 0xFF = not specified).

nDayOfWeek: day of the week (1...7, 1 = Monday, 7 = Sunday, 0xFF = not specified).

5.6.2 ST_BACnet_DateTime

Structure with date/time information.

```
stDate : ST_BACnet_Date;
stTime : ST_BACnet_Time;
```

5.6.3 ST_BACnet_ConfirmedServiceDiag

Structure with information on confirmed BACnet services (confirmed).

```
nReqSend         : DWORD; (* ULONG *)
nReqSendFail     : DWORD; (* ULONG *)
nReqRecv         : DWORD; (* ULONG *)
nReqRecvFail     : DWORD; (* ULONG *)
nAckSend         : DWORD; (* ULONG *)
nAckSendFail     : DWORD; (* ULONG *)
nAckRecv         : DWORD; (* ULONG *)
nAckRecvFail     : DWORD; (* ULONG *)
```

nReqSend: Number of confirmed requests, including retries.

nReqSendFail: Number of failed requests (no acknowledgement received).

nReqRecv: Number of confirmed requests.

nReqRecvFail: Number of confirmed requests that could not be processed (no free memory, wrong coding etc.).

nAckSend: Number of sent acknowledgements).

nAckSendFail: Number of failed acknowledgements.

nAckRecv: Number of received acknowledgements.

nAckRecvFail: Number of faulty received acknowledgements (no free memory, wrong coding etc.).

5.6.4 ST_BACnet_DiagEthStatistics

Structure with information on the Ethernet adapter.

```
ethStat    : ST_BACnet_TcIoEthStatistic;
txRxErrCnt : ST_BACnet_TcIoEthTxRxErrorCount;
```

5.6.5 ST_BACnet_DiagnosisTiming

Structure with information on faulty network frames from the network driver.

```
msIoInCurExecutionTime    : DWORD; (* ULONG *)
msIoInMinExecutionTime    : DWORD; (* ULONG *)
msIoInMaxExecutionTime    : DWORD; (* ULONG *)
msIoOutCurExecutionTime   : DWORD; (* ULONG *)
msIoOutMinExecutionTime   : DWORD; (* ULONG *)
msIoOutMaxExecutionTime   : DWORD; (* ULONG *)
msCycleCurExecutionTime   : DWORD; (* ULONG *)
msCycleMinExecutionTime   : DWORD; (* ULONG *)
msCycleMaxExecutionTime   : DWORD; (* ULONG *)
msInputCurDistanceTime    : DWORD; (* ULONG *)
msInputMinDistanceTime    : DWORD; (* ULONG *)
msInputMaxDistanceTime    : DWORD; (* ULONG *)
```

msIoInCurExecutionTime: Current processing time of the input data in the BACnet task (ms).

msIoInMinExecutionTime: Minimum processing time of the input data in the BACnet task (ms).

msIoInMaxExecutionTime: Maximum processing time of the input data in the BACnet task (ms).

msIoOutCurExecutionTime: Current processing time of the output data in the BACnet task (ms).

msIoOutMinExecutionTime: Minimum processing time of the output data in the BACnet task (ms).

msIoOutMaxExecutionTime: Maximum processing time of the output data in the BACnet task (ms).

msCycleCurExecutionTime: Current processing time of the BACnet task (ms).

msCycleMinExecutionTime: Minimum processing time of the BACnet task (ms).

msCycleMaxExecutionTime: Maximum processing time of the BACnet task (ms).

msInputCurDistanceTime: Current time duration between the start of the input data processing (ms).

msInputMinDistanceTime: Minimum time duration between the start of the input data processing (ms).

msInputMaxDistanceTime: Maximum time duration between the start of the input data processing (ms).

5.6.6 ST_BACnet_DiagnosisTiming

Structure with information on BACnet frames.

```
confirmed             : ARRAY[0..29] OF ST_BACnet_ConfirmedServiceDiag; (* [SERVICE_CONFIRMED_MAX + 1
]; 29 + 1 *)
unConfirmed           : ARRAY[0..9] OF ST_BACnet_UnConfirmedServiceDiag; (* [SERVICE_UNCONFIRMED_MAX
+ 1]; 9 *)
nSegmSendTimeouts     : DWORD; (* ULONG *)
nFrameAllocFailCnt    : DWORD; (* ULONG *)
nFrameSendCnt         : DWORD; (* ULONG *)
nFrameSendFailCnt     : DWORD; (* ULONG *)
nFrameRecvCnt         : DWORD; (* ULONG *)
nFrameRecvFailCnt     : DWORD; (* ULONG *)
nLinkStatusChangedCnt: DWORD; (* ULONG *)
```

nSegmSendTimeouts: Timeout during receipt of acknowledgement of segmented packages (remote station does not acknowledge a segmented response, segmented ACK)

nFrameAllocFailCnt: Number of lost frames due to problems with memory allocation (no free memory).

nFrameSendCnt: Number of sent frames.

nFrameSendFailCnt: Number of failed frames.

nFrameRecvCnt: Number of received frames.

nFrameRecvFailCnt: Number of faulty received frames.

nLinkStatusChangedCnt: Frequency of network status changes (cable pulled/link lost, cable plugged/link OK).

5.6.7 ST_BACnet_Info

Structure with general information on the BACnet task.

```
nAmsAllocCnt: DINT; (* LONG *)
nAmsAllocCntMax: DWORD; (* ULONG *)
nAmsAllocFailCnt: DWORD; (* ULONG *)
nAvailPhysMemBytes: DWORD; (* ULONG *)
nAvailFlashMemBytes: ARRAY[0..1] OF DWORD; (* ULONGLONG *)
nRecvFrameFifoSize: DWORD; (* ULONG *)
nEmptyFrameFifoSize: DWORD; (* ULONG *)
nRecvSegmUDPFramesListSize: DWORD; (* ULONG *)
nRecvSegmFramesListSize: DWORD; (* ULONG *)
nSendSegmFramesListSize: DWORD; (* ULONG *)
nClientScanListSize: DWORD; (* ULONG *)
nAdsRequestListSize: DWORD; (* ULONG *)
```

nAmsAllocCnt: Number of reserved router memory blocks (1 block= 1024 bytes). The number should not increase continuously (a maximum of 2000 blocks=2 MB are available by default). Otherwise increase the router memory in the System Manager (see Fig. 1).

nAmsAllocCntMax: Peak value of the simultaneously allocated router memory blocks.

nAmsAllocFailCnt: Number of failed router memory reservations (allocation fails if no more blocks are available).

nAvailPhysMemBytes: Free physical memory of the operating system in bytes.

nAvailFlashMemBytes: Free hard disk space of the operating system in bytes.

nRecvFrameFifoSize: Number of frames currently in the receive frame queue (for receive queue see "Network adapter settings")

nEmptyFrameFifoSize: Available memory for buffering received frames (frame is copied and then copied to RecvFrameFifo)

nRecvSegmUDPFramesListSize: Number of currently received buffered IP segments (name is wrong, not UDP but IP)

nRecvSegmFramesListSize: Number of currently received buffered BACnet frame segments

nSendSegmFramesListSize: Number of current BACnet frame segments that were buffered before sending

nClientScanListSize: Number of currently known BACnet devices in the network (DeviceAddressBinding in the device object is formed from this)

nAdsRequestListSize: Number of currently open internal ADS requests for reading/writing of files (stack accesses local files via ADS)

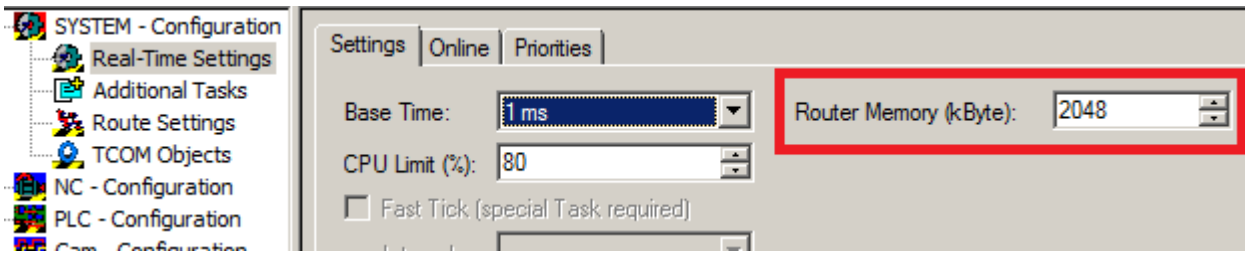


Fig. 197: Fig. 1: AMS router memory setting

5.6.8 ST_BACnet_ServerStatistics

Structure with information on the BACnet server.

```
nOpenRequests       : DWORD; (* ULONG *)
nRequestRetries     : DWORD; (* ULONG *)
nRequestTimeOuts    : DWORD; (* ULONG *)
nPropChanges        : DWORD; (* ULONG *)
nPersistWrites      : DWORD; (* ULONG *)
persistChangeSetId  : DWORD; (* ULONG *)
secpersistNextWrite : DWORD; (* ULONG *)
persistLastWrite    : ST_BACnet_DateTime;
```

nOpenRequests: Number of open requests.

nRequestRetries: Number of repeated requests.

nRequestTimeOuts: Number of failed requests.

nPropChanges: Number of property value changes (Present_Value, Status_Flags... etc.).

nPersistWrites: Number of write cycles for persistent data (frequency since system start).

persistChangeSetId: Absolute number of write cycles since the system was loaded (safe restart).

secpersistNextWrite: Seconds until the persistent data are saved next.

persistLastWrite: Time when the persistent data were saved last.

5.6.9 ST_BACnet_TcIoEthStatistic

Structure with information for sending and receiving of network frames from the network driver.

```
dcTimeStamp    : ARRAY[0..1] OF DWORD; (* ULONGLONG *)
sendFrames     : DWORD; (* ULONG *)
recvFrames     : DWORD; (* ULONG *)
sendTimeRTime  : DWORD; (* ULONG *)
recvTimeRTime  : DWORD; (* ULONG *)
sendTimeNdis   : DWORD; (* ULONG *)
recvTimeNdis   : DWORD; (* ULONG *)
```

dcTimeStamp: Real-time timestamp (DC time).

sendFrames: Sent Ethernet frames of the BACnet adapter.

recvFrames: Processed Ethernet frames of the BACnet adapter.

sendTimeRTime: Non-BACnet specific information from the network driver (internal diagnosis).

recvTimeRTime: Non-BACnet specific information from the network driver (internal diagnosis).

sendTimeNdis: Non-BACnet specific information from the network driver (internal diagnosis).

recvTimeNdis: Non-BACnet specific information from the network driver (internal diagnosis).

5.6.10 ST_BACnet_TcIoEthTxRxErrorCount

Structure with information on faulty network frames from the network driver.

```
txCnt    : DWORD; (* ULONG *)
rxCnt    : DWORD; (* ULONG *)
```

txCnt: Number of faulty sent frames (lost sent packets).

rxCnt: Number of faulty received frames (discarded received packets).

5.6.11 ST_BACnet_UnConfirmedServiceDiag

Structure with information on unconfirmed BACnet services (unconfirmed).

```
nReqSend        : DWORD; (* ULONG *)
nReqSendFail    : DWORD; (* ULONG *)
nReqRecv        : DWORD; (* ULONG *)
nReqRecvFail    : DWORD; (* ULONG *)
```

nReqSend: Number of sent requests (unconfirmed requests).

nReqSendFail: Number of failed requests.

nReqRecv: Number of received requests (unconfirmed requests).

nReqRecvFail: Number of requests that could not be processed (no free memory, wrong coding etc.).

5.6.12 ST_BACnet_ProgramHandshakeRequests

Structure with program request flags:

```
bLoad   : BOOL;
bSetup  : BOOL;
bRun    : BOOL;
bHalt   : BOOL;
bUnload : BOOL;
```

bLoad: Request loading of a program.

bSetup: Request setting up of a program.

bRun: Request starting of a program.

bHalt: Request stopping of a program.

bUnload: Request stopping and unloading of the program.

5.6.13 ST_BACnet_ProgramHandshakeStates

Structure with program handshake-flags:

```
bIdle      : BOOL;
bLoading   : BOOL;
bRunning   : BOOL;
bWaiting   : BOOL;
bHalted    : BOOL;
bUnloading : BOOL;
```

bIdle: Program is unloaded.

bLoading: Program is being loaded.

bRunning: Program was started.

bWaiting: Program is waiting for an external condition.

bHalted: Program stopped.

bUnloading: Program is being unloaded.

5.6.14 ST_BACnet_Time

Structure with time information.

```
nHour       : BYTE; (* hour (0..23), (X'FF') = unspecified *)
nMinute     : BYTE; (* minute (0..59), (X'FF') = unspecified *)
nSecond     : BYTE; (* (0..59), (X'FF') = unspecified *)
nHundredths : BYTE; (* hundredths (0..99), (X'FF') = unspecified *)
```

nHour: hour (0..23, 0xFF = not specified).

nMinute: minute (0..59, 0xFF = not specified).

nSecond: second (0..59, 0xFF = not specified).

nHundredths: hundredth of seconds (0..99, 0xFF = not specified).

5.6.15 ST_BACnet_Diagnosis

Structure with information from the BACnet device adapter. Includes dynamic response, statistics for server and client and diagnostic information on network traffic.

```
ethDevice        : ST_BACnet_DiagEthStatistics;
execTiming       : ST_BACnet_DiagnosisTiming;
frameStatistics  : ST_BACnet_FrameStatistics;
nRes1            : DWORD;
info             : ST_BACnet_Info;
nRes2            : DWORD;
serverInfo       : ST_BACnet_ServerStatistics;
lastUpdate       : ST_BACnet_DateTime;
nRes3            : DWORD;
```

lastUpdate: Timestamp of the diagnostic data (time at which the data were read).

5.7 BACnet Enumerations

5.7.1 E_BACnetAdapterStatus

```
BACnetAdapterStatus_Init := 16#0,
BACnetAdapterStatus_CheckIpAddr := 16#1,
BACnetAdapterStatus_CheckParam := 16#2,
BACnetAdapterStatus_GetGatewayMAC := 16#3,
BACnetAdapterStatus_WaitGatewayMAC := 16#4,
BACnetAdapterStatus_Complete := 16#8
```

5.7.2 E_BACnetBinaryPV

```
BACnetBinaryPV_inactive :=0,
BACnetBinaryPV_active :=1,
BACnetBinaryPV_NULL := 2
```

5.7.3 E_BACnetDataTypes

```
BACnetDataType_Null :=0,
BACnetDataType_Bool :=1,
BACnetDataType_UnsignedInteger :=2,
BACnetDataType_SignedInteger :=3,
BACnetDataType_Real :=4,
BACnetDataType_Double :=5,
BACnetDataType_OctetString :=6,
BACnetDataType_CharacterStringExt :=7,
BACnetDataType_BitString :=8,
BACnetDataType_Enumerated :=9,
BACnetDataType_Date :=10,
BACnetDataType_Time :=11,
BACnetDataType_ObjectIdentifier :=12,
BACnetDataType_DateTime :=14,
BACnetDataType_ArrayIndex :=15,
BACnetDataType_CalendarEntry :=16,
BACnetDataType_ObjectType :=17,
```

```
BACnetDataType_EventTransitionBits :=18,
BACnetDataType_ActionList :=19,
BACnetDataType_StatusFlags :=20,
BACnetDataType_EventState :=21,
BACnetDataType_Reliability :=22,
BACnetDataType_Polarity :=23,
BACnetDataType_EngineeringUnits :=25,
BACnetDataType_PriorityValue :=26,
BACnetDataType_LimitEnable :=27,
BACnetDataType_NotifyType :=29,
BACnetDataType_TimeStamp :=30,
BACnetDataType_DeviceObjectPropertyReference :=31,
BACnetDataType_DeviceStatus :=32,
BACnetDataType_ServicesSupported :=33,
BACnetDataType_ObjectTypesSupported :=34,
BACnetDataType_Segmentation :=35,
BACnetDataType_VTClass :=36,
BACnetDataType_VTSession :=37,
BACnetDataType_SessionKey :=38,
BACnetDataType_Recipient :=39,
BACnetDataType_AddressBinding :=40,
BACnetDataType_COVSubscription :=41,
BACnetDataType_EventType :=42,
BACnetDataType_EventParameter :=43,
BACnetDataType_FileAccessMethod :=44,
BACnetDataType_ReadAccessSpecification :=45,
BACnetDataType_ReadAccessResult :=46,
BACnetDataType_ObjectPropertyReference :=47,
BACnetDataType_SetpointReference :=48,
BACnetDataType_Destination :=49,
BACnetDataType_ProgramState :=50,
BACnetDataType_ProgramRequest :=51,
BACnetDataType_ProgramError :=52,
BACnetDataType_DateRange :=53,
BACnetDataType_DailySchedule :=54,
BACnetDataType_SpecialEvent :=55,
BACnetDataType_ClientCOV :=56,
BACnetDataType_LogRecord :=57,
BACnetDataType_LifeSafetyState :=58,
BACnetDataType_LifeSafetyMode :=59,
BACnetDataType_SilencedState :=60,
BACnetDataType_LifeSafetyOperation :=61,
BACnetDataType_BinaryPV :=62,
BACnetDataType_DaysOfWeek :=63,
BACnetDataType_WeekNDay :=64,
BACnetDataType_TimeValue :=65,
BACnetDataType_Value :=66,
BACnetDataType_Address :=67,
BACnetDataType_PropertyIdentifier :=68,
BACnetDataType_EnumerationValueType :=69,
BACnetDataType_BitFieldBit :=70,
BACnetDataType_LogDatum :=71,
BACnetDataType_AnyType :=72,
BACnetDataType_PresentValue :=73,
BACnetDataType_ContextTag :=74,
BACnetDataType_ValueChoice :=75,
BACnetDataType_LogStatus :=76,
BACnetDataType_RecipientProcess :=77,
BACnetDataType_SubscribeCOVPropertyRequest :=78,
BACnetDataType_PropertyReference :=79,
BACnetDataType_PropertyResult :=80,
BACnetDataType_DeviceObjectPropertyValue :=81,
BACnetDataType_COVNotificationRequest :=82,
BACnetDataType_EventNotificationRequest :=83,
BACnetDataType_NotificationParameters :=84,
BACnetDataType_Change_of_Bitstring :=85,
BACnetDataType_Change_of_State :=86,
BACnetDataType_Change_of_Value :=87,
BACnetDataType_Command_failure :=88,
BACnetDataType_Floating_limit :=89,
BACnetDataType_Out_of_Range :=90,
BACnetDataType_PropertyValue :=91,
BACnetDataType_Complex_Tag :=92,
BACnetDataType_Change_of_life_safety :=93,
BACnetDataType_Extended :=94,
BACnetDataType_Buffer_ready :=95,
BACnetDataType_Unsigned_range :=96,
BACnetDataType_PropertyResultValue :=97,
BACnetDataType_ServiceCOVPropSubscription :=98,
BACnetDataType_ServiceCOVSubscription :=99,
```

```
BACnetDataType_COVNotification :=100,
BACnetDataType_AcknowledgeAlarmRequest :=101,
BACnetDataType_VTOpenRequest :=102,
BACnetDataType_Prescale :=103,
BACnetDataType_Scale :=104,
BACnetDataType_Action :=105,
BACnetDataType_Error :=106,
BACnetDataType_ErrorType :=107,
BACnetDataType_ErrorClass :=108,
BACnetDataType_ErrorCode :=109,
BACnetDataType_RejectReason :=110,
BACnetDataType_AbortReason :=111,
BACnetDataType_BDTEntry :=112,
BACnetDataType_IpEntry :=113,
BACnetDataType_MaskEntry :=114,
BACnetDataType_UnsignedIntegerArr :=258,
BACnetDataType_CharacterStringExtArr :=263,
BACnetDataType_ObjectIdentifierArr :=268,
BACnetDataType_CalendarEntryArr :=272,
BACnetDataType_ActionListArr :=275,
BACnetDataType_PriorityValueArr :=282,
BACnetDataType_TimeStampArr :=286,
BACnetDataType_DeviceObjectPropertyReferenceArr :=287,
BACnetDataType_VTClassArr :=292,
BACnetDataType_VTSessionArr :=293,
BACnetDataType_SessionKeyArr :=294,
BACnetDataType_RecipientArr :=295,
BACnetDataType_AddressBindingArr :=296,
BACnetDataType_COVSubscriptionArr :=297,
BACnetDataType_ReadAccessSpecificationArr :=301,
BACnetDataType_ReadAccessResultArr :=302,
BACnetDataType_DestinationArr :=305,
BACnetDataType_SpecialEventArr :=311,
BACnetDataType_LogRecordArr :=313,
BACnetDataType_LifeSafetyStateArr :=314,
BACnetDataType_TimeValueArr :=321,
BACnetDataType_UnsignedIntegerList :=514,
BACnetDataType_CharacterStringExtList :=519,
BACnetDataType_ReadAccessSpecificationList :=557,
BACnetDataType_ReadAccessResultList :=558,
BACnetDataType_SpecialEventList :=567,
BACnetDataType_LogRecordList :=569,
BACnetDataType_TimeValueList :=577,
BACnetDataType_ValueList :=578,
BACnetDataType_LogDatumList :=583,
BACnetDataType_PropertyReferenceList :=591,
BACnetDataType_PropertyResultList :=592,
BACnetDataType_COVNotificationRequestList :=594,
BACnetDataType_EventNotificationRequestList :=595,
BACnetDataType_PropertyValueList :=603,
BACnetDataType_DailyScheduleList :=1089
```

5.7.4 E_BACnetDeviceStatus

```
BACnetDeviceStatus_Operational :=0,
BACnetDeviceStatus_OperationalReadOnly :=1,
BACnetDeviceStatus_DownloadRequired :=2,
BACnetDeviceStatus_DownloadInProgress :=3,
BACnetDeviceStatus_NonOperational :=4,
BACnetDeviceStatus_BackupInProgress :=5
```

5.7.5 E_BACnetEventState

```
BACnetEventState_state_normal :=0,
BACnetEventState_fault :=1,
BACnetEventState_offnormal :=2,
BACnetEventState_high_limit :=3,
BACnetEventState_low_limit :=4,
BACnetEventState_life_safety_alarm :=5
```

5.7.6 E_BACnetObjectType

```
BACnetObjectType_AnalogInput :=0,
BACnetObjectType_AnalogOutput :=1,
BACnetObjectType_AnalogValue :=2,
BACnetObjectType_BinaryInput :=3,
```

```
BACnetObjectType_BinaryOutput :=4,
BACnetObjectType_BinaryValue :=5,
BACnetObjectType_Calendar :=6,
BACnetObjectType_Command :=7,
BACnetObjectType_Device :=8,
BACnetObjectType_EventEnrollment :=9,
BACnetObjectType_File :=10,
BACnetObjectType_Group :=11,
BACnetObjectType_Loop :=12,
BACnetObjectType_MultiStateInput :=13,
BACnetObjectType_MultiStateOutput :=14,
BACnetObjectType_NotificationClass :=15,
BACnetObjectType_Program :=16,
BACnetObjectType_Schedule :=17,
BACnetObjectType_Averaging :=18,
BACnetObjectType_MultiStateValue :=19,
BACnetObjectType_TrendLog :=20,
BACnetObjectType_LifeSafetyPoint :=21,
BACnetObjectType_LifeSafetyZone :=22,
BACnetObjectType_Accumulator :=23,
BACnetObjectType_PulseConverter :=24
```

5.7.7 E_BACnetPriority

```
BACnetPriority_None :=0,
BACnetPriority_1 :=1,
BACnetPriority_2 :=2,
BACnetPriority_3 :=3,
BACnetPriority_4 :=4,
BACnetPriority_5 :=5,
BACnetPriority_6 :=6,
BACnetPriority_7 :=7,
BACnetPriority_8 :=8,
BACnetPriority_9 :=9,
BACnetPriority_10 :=10,
BACnetPriority_11 :=11,
BACnetPriority_12 :=12,
BACnetPriority_13 :=13,
BACnetPriority_14 :=14,
BACnetPriority_15 :=15,
BACnetPriority_16 :=16
```

5.7.8 E_BACnetProgramError

Enumeration with error feedback for the program object (program error):

```
BACnetProgramError_program_normal :=0,
BACnetProgramError_load_failed :=1,
BACnetProgramError_progerror_internal :=2,
BACnetProgramError_program :=3,
BACnetProgramError_other :=4
```

BACnetProgramError_program_normal: No error.

BACnetProgramError_load_failed: Loading of the program has failed.

BACnetProgramError_progerror_internal: Internal program error.

BACnetProgramError_program: Error during program execution.

BACnetProgramError_other: Unknown error.

5.7.9 E_BACnetProgramRequest

Enumeration with request to the program object (program request):

```
BACnetProgramRequest_ready :=0,
BACnetProgramRequest_load :=1,
BACnetProgramRequest_run :=2,
BACnetProgramRequest_halt :=3,
BACnetProgramRequest_restart :=4,
BACnetProgramRequest_unload :=5
```

BACnetProgramRequest_ready: The program is ready for a request or has executed/aborted the last request.

BACnetProgramRequest_load: Request loading of the program.

BACnetProgramRequest_run: Request initiating of the program.

BACnetProgramRequest_halt: Request stopping of a program.

BACnetProgramRequest_restart: Request restarting of the program.

BACnetProgramRequest_unload: Request stopping and unloading of the program.

5.7.10 E_BACnetProgramState

Enumeration with the state of the program object (program state):

```
BACnetProgramState_idle :=0,
BACnetProgramState_loading :=1,
BACnetProgramState_running :=2,
BACnetProgramState_waiting :=3,
BACnetProgramState_halted :=4,
BACnetProgramState_unloading :=5
```

BACnetProgramState_idle: Program is unloaded.

BACnetProgramState_loading: Program is being loaded.

BACnetProgramState_running: Program was started.

BACnetProgramState_waiting: Program is waiting for an external condition.

BACnetProgramState_halted: Program stopped.

BACnetProgramState_unloading: Program is being unloaded.

5.7.11 E_BACnetPropertyIdentifier

```
BACnetPropertyIdentifier_AckedTransitions :=0,
BACnetPropertyIdentifier_AckRequired :=1,
BACnetPropertyIdentifier_Action :=2,
BACnetPropertyIdentifier_ActionText :=3,
BACnetPropertyIdentifier_ActiveText :=4,
BACnetPropertyIdentifier_ActiveVtSessions :=5,
BACnetPropertyIdentifier_AlarmValue :=6,
BACnetPropertyIdentifier_AlarmValues :=7,
BACnetPropertyIdentifier_All :=8,
BACnetPropertyIdentifier_AllWritesSuccessful :=9,
BACnetPropertyIdentifier_ApduSegmentTimeout :=10,
BACnetPropertyIdentifier_ApduTimeout :=11,
BACnetPropertyIdentifier_ApplicationSoftwareVersion :=12,
BACnetPropertyIdentifier_Archive :=13,
BACnetPropertyIdentifier_Bias :=14,
BACnetPropertyIdentifier_ChangeOfStateCount :=15,
BACnetPropertyIdentifier_ChangeOfStateTime :=16,
BACnetPropertyIdentifier_NotificationClass :=17,
BACnetPropertyIdentifier_ControlledVariableReference :=19,
BACnetPropertyIdentifier_ControlledVariableUnits :=20,
BACnetPropertyIdentifier_ControlledVariableValue :=21,
BACnetPropertyIdentifier_CovIncrement :=22,
BACnetPropertyIdentifier_Datelist :=23,
BACnetPropertyIdentifier_DaylightSavingsStatus :=24,
BACnetPropertyIdentifier_Deadband :=25,
BACnetPropertyIdentifier_DerivativeConstant :=26,
BACnetPropertyIdentifier_DerivativeConstantUnits :=27,
BACnetPropertyIdentifier_Description :=28,
BACnetPropertyIdentifier_DescriptionOfHalt :=29,
BACnetPropertyIdentifier_DeviceAddressBinding :=30,
BACnetPropertyIdentifier_DeviceType :=31,
BACnetPropertyIdentifier_EffectivePeriod :=32,
BACnetPropertyIdentifier_ElapsedActiveTime :=33,
BACnetPropertyIdentifier_ErrorLimit :=34,
BACnetPropertyIdentifier_EventEnable :=35,
BACnetPropertyIdentifier_EventState :=36,
```

```
BACnetPropertyIdentifier_EventType :=37,
BACnetPropertyIdentifier_ExceptionSchedule :=38,
BACnetPropertyIdentifier_FaultValues :=39,
BACnetPropertyIdentifier_FeedbackValue :=40,
BACnetPropertyIdentifier_FileAccessMethod :=41,
BACnetPropertyIdentifier_FileSize :=42,
BACnetPropertyIdentifier_FileType :=43,
BACnetPropertyIdentifier_FirmwareRevision :=44,
BACnetPropertyIdentifier_HighLimit :=45,
BACnetPropertyIdentifier_InactiveText :=46,
BACnetPropertyIdentifier_InProcess :=47,
BACnetPropertyIdentifier_InstanceOf :=48,
BACnetPropertyIdentifier_IntegralConstant :=49,
BACnetPropertyIdentifier_IntegralConstantUnits :=50,
BACnetPropertyIdentifier_IssueConfirmedNotifications :=51,
BACnetPropertyIdentifier_LimitEnable :=52,
BACnetPropertyIdentifier_ListOfGroupMembers :=53,
BACnetPropertyIdentifier_ListOfObjectPropertyReferences :=54,
BACnetPropertyIdentifier_ListOfSessionKeys :=55,
BACnetPropertyIdentifier_LocalDate :=56,
BACnetPropertyIdentifier_LocalTime :=57,
BACnetPropertyIdentifier_Location :=58,
BACnetPropertyIdentifier_LowLimit :=59,
BACnetPropertyIdentifier_ManipulatedVariableReference :=60,
BACnetPropertyIdentifier_MaximumOutput :=61,
BACnetPropertyIdentifier_MaxApduLengthAccepted :=62,
BACnetPropertyIdentifier_MaxInfoFrames :=63,
BACnetPropertyIdentifier_MaxMaster :=64,
BACnetPropertyIdentifier_MaxPresValue :=65,
BACnetPropertyIdentifier_MinimumOfftime :=66,
BACnetPropertyIdentifier_MinimumOntime :=67,
BACnetPropertyIdentifier_MinimumOutput :=68,
BACnetPropertyIdentifier_MinPresValue :=69,
BACnetPropertyIdentifier_ModelName :=70,
BACnetPropertyIdentifier_ModificationDate :=71,
BACnetPropertyIdentifier_NotifyType :=72,
BACnetPropertyIdentifier_NumberOfAPDURetries :=73,
BACnetPropertyIdentifier_NumberOfStates :=74,
BACnetPropertyIdentifier_ObjectIdentifier :=75,
BACnetPropertyIdentifier_ObjectList :=76,
BACnetPropertyIdentifier_ObjectName :=77,
BACnetPropertyIdentifier_ObjectReference :=78,
BACnetPropertyIdentifier_ObjectType :=79,
BACnetPropertyIdentifier_Optional :=80,
BACnetPropertyIdentifier_OutOfService :=81,
BACnetPropertyIdentifier_OutputUnits :=82,
BACnetPropertyIdentifier_EventParameters :=83,
BACnetPropertyIdentifier_Polarity :=84,
BACnetPropertyIdentifier_PresentValue :=85,
BACnetPropertyIdentifier_Priority :=86,
BACnetPropertyIdentifier_PriorityArray :=87,
BACnetPropertyIdentifier_PriorityForWriting :=88,
BACnetPropertyIdentifier_ProcessIdentifier :=89,
BACnetPropertyIdentifier_ProgramChange :=90,
BACnetPropertyIdentifier_ProgramLocation :=91,
BACnetPropertyIdentifier_ProgramState :=92,
BACnetPropertyIdentifier_ProportionalConstant :=93,
BACnetPropertyIdentifier_ProportionalConstantUnits :=94,
BACnetPropertyIdentifier_ProtocolObjectTypesSupported :=96,
BACnetPropertyIdentifier_ProtocolServicesSupported :=97,
BACnetPropertyIdentifier_ProtocolVersion :=98,
BACnetPropertyIdentifier_ReadOnly :=99,
BACnetPropertyIdentifier_ReasonForHalt :=100,
BACnetPropertyIdentifier_Recipient :=101,
BACnetPropertyIdentifier_RecipientList :=102,
BACnetPropertyIdentifier_Reliability :=103,
BACnetPropertyIdentifier_RelinquishDefault :=104,
BACnetPropertyIdentifier_Required :=105,
BACnetPropertyIdentifier_Resolution :=106,
BACnetPropertyIdentifier_SegmentationSupported :=107,
BACnetPropertyIdentifier_Setpoint :=108,
BACnetPropertyIdentifier_SetpointReference :=109,
BACnetPropertyIdentifier_StateText :=110,
BACnetPropertyIdentifier_StatusFlags :=111,
BACnetPropertyIdentifier_SystemStatus :=112,
BACnetPropertyIdentifier_TimeDelay :=113,
BACnetPropertyIdentifier_TimeOfActiveTimeReset :=114,
BACnetPropertyIdentifier_TimeOfStateCountReset :=115,
BACnetPropertyIdentifier_TimeSynchronizationRecipients :=116,
BACnetPropertyIdentifier_Units :=117,
```

```
BACnetPropertyIdentifier_UpdateInterval :=118,
BACnetPropertyIdentifier_UtcOffset :=119,
BACnetPropertyIdentifier_VendorIdentifier :=120,
BACnetPropertyIdentifier_VendorName :=121,
BACnetPropertyIdentifier_VtClassesSupported :=122,
BACnetPropertyIdentifier_WeeklySchedule :=123,
BACnetPropertyIdentifier_AttemptedSamples :=124,
BACnetPropertyIdentifier_AverageValue :=125,
BACnetPropertyIdentifier_BufferSize :=126,
BACnetPropertyIdentifier_ClientCovIncrement :=127,
BACnetPropertyIdentifier_CovResubscriptionInterval :=128,
BACnetPropertyIdentifier_EventTimeStamps :=130,
BACnetPropertyIdentifier_LogBuffer :=131,
BACnetPropertyIdentifier_LogDeviceObjectProperty :=132,
BACnetPropertyIdentifier_LogEnable :=133,
BACnetPropertyIdentifier_LogInterval :=134,
BACnetPropertyIdentifier_MaximumValue :=135,
BACnetPropertyIdentifier_MinimumValue :=136,
BACnetPropertyIdentifier_NotificationThreshold :=137,
BACnetPropertyIdentifier_ProtocolRevision :=139,
BACnetPropertyIdentifier_RecordsSinceNotification :=140,
BACnetPropertyIdentifier_RecordCount :=141,
BACnetPropertyIdentifier_StartTime :=142,
BACnetPropertyIdentifier_StopTime :=143,
BACnetPropertyIdentifier_StopWhenFull :=144,
BACnetPropertyIdentifier_TotalRecordCount :=145,
BACnetPropertyIdentifier_ValidSamples :=146,
BACnetPropertyIdentifier_WindowInterval :=147,
BACnetPropertyIdentifier_WindowSamples :=148,
BACnetPropertyIdentifier_MaximumValueTimestamp :=149,
BACnetPropertyIdentifier_MinimumValueTimestamp :=150,
BACnetPropertyIdentifier_VarianceValue :=151,
BACnetPropertyIdentifier_ActiveCovSubscriptions :=152,
BACnetPropertyIdentifier_BackupFailureTimeout :=153,
BACnetPropertyIdentifier_ConfigurationFiles :=154,
BACnetPropertyIdentifier_DatabaseRevision :=155,
BACnetPropertyIdentifier_DirectReading :=156,
BACnetPropertyIdentifier_LastRestoreTime :=157,
BACnetPropertyIdentifier_MaintenanceRequired :=158,
BACnetPropertyIdentifier_MemberOf :=159,
BACnetPropertyIdentifier_Mode :=160,
BACnetPropertyIdentifier_OperationExpected :=161,
BACnetPropertyIdentifier_Setting :=162,
BACnetPropertyIdentifier_Silenced :=163,
BACnetPropertyIdentifier_TrackingValue :=164,
BACnetPropertyIdentifier_ZoneMembers :=165,
BACnetPropertyIdentifier_LifeSafetyAlarmValues :=166,
BACnetPropertyIdentifier_MaxSegmentsAccepted :=167,
BACnetPropertyIdentifier_ProfileName :=168,
BACnetPropertyIdentifier_InternalUse :=169,
BACnetPropertyIdentifier_LastNotifyRecord :=173,
BACnetPropertyIdentifier_ScheduleDefault :=174,
BACnetPropertyIdentifier_AcceptedModes :=175,
BACnetPropertyIdentifier_AdjustValue :=176,
BACnetPropertyIdentifier_Count :=177,
BACnetPropertyIdentifier_CountBeforeChange :=178,
BACnetPropertyIdentifier_CovPeriod :=180,
BACnetPropertyIdentifier_InputReference :=181,
BACnetPropertyIdentifier_LimitMonitoringInverval :=182,
BACnetPropertyIdentifier_LoggingObject :=183,
BACnetPropertyIdentifier_LoggingRecord :=184,
BACnetPropertyIdentifier_Prescale :=185,
BACnetPropertyIdentifier_PulseRate :=186,
BACnetPropertyIdentifier_Scale :=187,
BACnetPropertyIdentifier_ScaleFactor :=188,
BACnetPropertyIdentifier_UpdateTime :=189,
BACnetPropertyIdentifier_ValueBeforeChange :=190,
BACnetPropertyIdentifier_ValueSet :=191,
BACnetPropertyIdentifier_ValueChangeTime :=192
```

5.8 BACnet_Globals

```
VAR_GLOBAL
(* global linkable BACnet adapter *)
    fbBACnet_Adapter : FB_BACnet_Adapter;
(* global visible BACnet adapter reference *)
```

```
    pBACnet_Adapter : POINTER TO FB_BACnet_Adapter;
END_VAR
VAR_GLOBAL CONSTANT
    tBACnet_ADSTimeOut : TIME := t#5s;
END_VAR
```

6 Appendix

6.1 ADS Interface

All acyclic data are transferred with ADS Read, ADS Write, ADS ReadWrite or ADS Write Control to a BACnet device, a notification sink, a BACnet server or a BACnet client. Each BACnet device has a dedicate Net ID, which is displayed via the Settings tab and supports the following ports:

Port	Description
0xFFFF	Addresses the BACnet device itself, i.e. data that are not specific for a BACnet server, a notification sink or a BACnet client
1000 - 8192	Addresses a BACnet server, a notification sink or a BACnet client. The first configured BACnet server/client can be reached via port 1000, the second via port 1001 etc. Deleting a BACnet server/client does not lead to shifting of the ports, i.e. the assignment of the ADS port can therefore not necessarily be deduced from the order in the System Manager. The respective port can be viewed via the Settings tab of a BACnet server/client.

ADS-Read

An overview of the IndexGroup/IndexOffset supported by a BACnet device for ADS Read is provided below.

IndexGroup with local addressing of the BACnet device (port 0xFFFF)

IndexGroup (Lo-Word)	IndexGroup (Hi-Word)	IndexOffset	Description
0x0006	0x8000	always 0	Starting a scan for other BACnet devices in the network.
0x0008	0x8000	always 0	Stopping a scan for other BACnet devices in the network. Preparation for reading the results list.

0x0007	0x8000	0 : Number of entries in the results list
		(number of clients*2)
		2*n: IP address of device n
		2*n+1: Device ID of device n
		0xffffffff
			A scan with the aid of a BACnet device follows the following pattern:
			1. Starting the scan
			2. Waiting for any timeout
			3. Stopping the scan
			4. Reading the scan results list
			The scan results are read via this IndexGroup. The number of entries in the results list can be determined via IndexOffset 0. This corresponds to twice the number of BACnet devices found in the network. Even IndexOffsets > 0 address the IP address of a device; odd IndexOffsets address the device ID. Before reading the results list the scan process has to be stopped.
			If, for example, 2 devices are found in a network, the query for index 0 comes up with 4. IndexOffset 1 relates to the device ID of device 1, IndexOffset 2 the IP address of device 1, IndexOffset 3 the device ID of device 2 and IndexOffset 4 the IP address of device 2.
			
			Reading the objects present in the server. An array of type BACnetScanResult with the number of found clients is returned:
			enum BACnetAddressChoice
			{
			Ethernet,Arcnet,MsTp,LonTalk,LonTalkNeuronId,
			Address_3,Address_4,Address_5,Broadcast
			}

IndexGroup (Lo-Word)	IndexGroup (Hi-Word)	IndexOffset	Description

			struct BACnetAddress
			{
			BACnetAddressChoice choice;
			BYTE layerAddress[7]; // MS/TP...LonTalk
			BYTE reserved;
			USHORT network_number;
			}
			struct BACnetScanResult
			{
			BACnetObjectIdentifier deviceObjIdentifier;
			BACnetAddress netAddress;
			ULONG IPadress;
			WORD UdpPort;
			WORD vendorId;
			BYTE ethernetMacAddr[6];
			}

0x000A	0x8000	always 0	Starting a backup. (triggering the BACnet service Backup Start)
0x001A	0x8000	always 0	Stopping a backup. (triggering the BACnet service Backup End) Between Backup Start and End the generated backup files can be saved with the aid of the Property Configuration Files of device object and the ADS file service.
0x002A	0x8000	always 0	Starting a restore. (triggering the BACnet service Restore Start)
0x003A	0x8000	always 0	Stopping a restore (triggering the BACnet service Restore End)
0x004A	0x8000	always 0	Aborting a restore (triggering the BACnet service Restore Abort).
0x005A	0x8000	always 0	Restarting the TwinCAT computer (triggering the BACnet service Cold Start)
0x006A	0x8000	always 0	Restarting TwinCAT (triggering the BACnet service Warm Start).
0x000C	0x8000	always 0	Reading the supplement key status: • 0x0000000 - supplement key is not valid • 0x0000001 - supplement key is valid

0x000D	0x8000	always 0	Reading the current Foreign Device Table (FDT) as array of type FDTEntry:
			
			struct FDTEntry
			{
			ULONG ipAddr;
			USHORT udpPort;
			USHORT nTtlSecs;
			USHORT nPurgeSecs; // seconds until entry is deleted
			USHORT reserved; // alignment
			}

0x000E	0x8000	always 0	Reading the current Broadcast Distribution Table (BDT) as array of type BDTEntry: struct BDTEntry { ULONG ipAddr; ULONG mask;

IndexGroup (Lo-Word)	IndexGroup (Hi-Word)	IndexOffset	Description

			USHORT udpPort;
			USHORT reserved; // alignment
			}

0x000F	0x8000	always 0	Reading the BACnet diagnosis in the form of BACnetDiagnosis. (see BACnet device tab Diagnostics).

0x001F	0x8000	always 0	Reading the Advanced Parameters in the form of Advanced Parameters
			struct AdvancedParameters
			{
			ULONG ap_MAX_SUBSCRIBE_COV_ENTRIES;
			ULONG ap_MAX_PROPERTY_ELEMENTS;
			ULONG ap_LIST_ALLOC_MEM_BYTES;
			ULONG ap_MAX_DEVICE_BINDINGS;
			ULONG ap_HASHTABLE_MAX_OBJECTS;
			ULONG ap_HASHTABLE_MAX_CLIENTS;
			ULONG ap_MAX_TASK_LIST_ENTRIES;
			ULONG ap_MAX_TRENDLOG_ENTRYSIZE;
			ULONG ap_DYN_TRENDLOG_BUFFERSIZE;
			ULONG ap_MAX_MULTISTATE_DYNSTATES;
			ULONG ap_BACNET_MAX_RECV_BAC_SEGM_FRAMES;
			ULONG ap_BACNET_MAX_SEND_BAC_SEGM_FRAMES;
			ULONG ap_IO_STARTUP_TIMEOUT;
			}

IndexGroup for addressing a BACnet server/client (port 1000-8191) - property access

Property specific ADS services can be used with an IndexGroup/IndexOffset with the following scheme:

- Bit 0-21 contains the property ID of the required BACnet property
- Bit 22-30 contains the group type describing the respective functionality
- Bit 31 is always 1 and activates the group type mode for using the property-specific services
- The IndexOffset contains the object ID of the required object

IndexGroup	IndexOffset	Description
0x80000000 + property ID Group type (bits 22-30): 0	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Reading the current data of a property. Example: The property PresentValue of object BinaryInput: 42 can be read via IndexGroup: 0x80000055 and IndexOffset: 0x00C0002A.
0x80400000 + property ID Group type (bits 22-30): 1	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Reading the BACnet data type of a property. A list of data type coding can be found in the appendix.
0x80800000 + property ID Group type (bits 22-30): 2	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Reading the data length of the current property value.
0x80C00000 + property ID Group type (bits 22-30): 3	Object ID • Bits 0 - 21: object instance	Reading the write protection status of a property. The read data are interpreted as follows: • 0x00 property value can be written • 0x01 property value is read-only

IndexGroup	IndexOffset	Description
	• Bits 22 - 31: object type	
0x81400000 + property ID Group type (bits 22-30): 5	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Reading the number of elements of list and array types.

IndexGroup for addressing a BACnet server (port 1000-8191)

IndexGroup	IndexOffset	Description
0x82400000	0 number of list entries > 0 and < 0xffffffff	Reading the objects present in the server. The objects created on a server can also be determined by reading the property ObjectList of the device object. Index 0: Number of entries Index n*2: object type for entry n Index n*2+1: object instance for entry n
0x83400000	always 0	Reading the list of dynamically created objects. For each object the list has a 4-byte entry containing the respective object ID.

IndexGroup for addressing a notification sink (port 1000-8191)

IndexGroup	IndexOffset	Description
0x00000001	always 0	Reading the current number of event notifications in the event notification buffer
0x00000002	always 0	Reading the current number of COV notifications in the COV notification buffer
0x00000003	Number of the event notification (0 .. number-1)	Reading an event notification. The structure of event notifications can be determined in the online dialog of the notification sink.
0x00000004	Number of the COV notification (0 .. number-1)	Reading a COV notification. The structure of COV notifications can be determined in the online dialog of the notification sink.
0x00000005	Number of the event notification (0 .. number-1)	Deleting an event notification
0x00000006	Number of the COV notification (0 .. number-1)	Deleting a COV notification
0x00000007	Number of the event notification (0 .. number-1)	Reading the time of receipt of an event notification
0x00000008	Number of the COV notification (0 .. number-1)	Reading the time of receipt of a COV notification

ADS Write

An overview of the IndexGroup/IndexOffset supported by TwinCAT BACnet/IP for ADS Write is provided below.

IndexGroup with local addressing of the BACnet device (port 0xFFFF)

IndexGroup	IndexOffset	Description
0x8000002D	always 0	Triggering a global broadcast UTC synchronisation. A valid BACnetDateTime time stamp is required. struct BACnetTime { BYTE hour; BYTE minute; BYTE second; BYTE hundredths; } structBACnetDate { BYTE year; BYTE month; BYTE day; BYTE dayOfWeek; } structBACnetDateTime { BACnetDate date; BACnetTime time; }
0x8000000F	always 0	BACnet diagnosis control: Transferred data size 4: • 0x00000000 : deactivate time measurement • 0x00000001 : activate time measurement Data size not equal 4: resetting the BACnet diagnosis

0x8000000E	always 0	Writing the Broadcast Distribution Table (BDT) as array in the form of BDTEntry:
		
		struct BDTEntry
		{
		ULONG ipAddr;
		ULONG mask;
		USHORT udpPort;
		USHORT reserved; // alignment
		}

IndexGroup for addressing a BACnet server/client (port 1000-8191) - property access

Property specific ADS services can be used with an IndexGroup with the following scheme:

- Bits 0-21 contains the property ID of the required BACnet property
- Bit 22-30 contains the group type describing the respective functionality
- Bit 31 is always 1 and activates the group type mode for using the property-specific services
- The IndexOffset contains the object ID of the required object

IndexGroup	IndexOffset	Description
0x80000000 + property ID	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Writing a property value. Prioritized write access is available via bits 22-30 of the IndexGroup. Priority 1 (top priority in BACnet) can be defined via the value 0x010 in bits 22-30. Priority 16 (the lowest priority in BACnet) can be defined with value 0x100.
0x84000000 + property-ID	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Write access with priority 1.
...		
0xC0000000 + property ID	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Write access with priority 16.

IndexGroup for addressing a BACnet server/client (port 1000-8191)

IndexGroup	IndexOffset	Description
0x80800000	always 0	Creating an object dynamically. The following data must be transferred in the write request, to parameterize the object creation: • 4-byte object type • 4-byte object ID (the object type of the object ID must match the object type) • 4-byte flags (0x00000001 remote object (for client) , 0x00000000 local object (for server)) • 8-byte 0x00 for an empty list of initial parameters, or a list of initials parameters with data type ReadPropertyMultipleResult
0x81000000	always 0	Deleting a dynamically created object. The following data must be transferred in a write request: • 4-byte object type • 4-byte object ID (the object type of the object ID must match the object type) • 4-byte flags (0x00000001 remote object (for client) , 0x00000000 local object (for server)) • 8-byte 0x00
0x80C00000	always 0	Triggering saving of the persistent data. Only valid for BACnet servers if Persist Online Data is active.

ADS-ReadWrite

An overview of the IndexGroup/IndexOffset supported by TwinCAT BACnet/IP for ADS ReadWrite is provided below.

IndexGroup for addressing a BACnet server/client (port 1000-8191)

IndexGroup	IndexOffset	Description
0x82C00000	Object ID • Bits 0 - 21: object instance • Bits 22 - 31: object type	Reading the property list of an object. This service consolidates all properties of an object, like the BACnet service ReadPropertyMultiple. When the ADS ReadWrite command is executed a list with the required properties must be transferred. Currently only triggering of the complete list is supported. To this end 4 bytes with the value 0x00000008 must be transferred. A list of type ReadPropertyMultipleResult is returned. The list contains list headers and property IDs, followed by the property data.
0x8B800000	always 0	Query of the object ID with the name of an object. The name of the requested object must be written as ASCII string. The corresponding object ID (4 bytes) is returned. If an object is not found the ADS error NOT_FOUND is returned.

ADS Error Codes

The 32-bit ADS error code always consists of a general ADS error code (low word, see ADS documentation). The appropriate text message will also be displayed in the TwinCAT System Manager Logger.

6.2 Data Types

In the automation interface and via ADS complex properties are coded as byte arrays. This section explains how the BACnet data concepts are mapped to C data structures within TwinCAT BACnet/IP, and how they can be used during configuration via the automation interface or ADS. Section 21 of the BACnet standard ("FORMAL DESCRIPTION OF APPLICATION PROTOCOL DATA UNITS") specifies the BACnet data structures.

Apart from a few exceptions, the mapping from BACnet to C is based on the BACnet specification. Exceptions are dynamic data, which are mapped to lists or AnyTypes. The type of a property can generally be determined with the aid of the TwinCAT System Manager. The data type of each property, including the structure (by expanding the property data), can be seen in the Type column in the Settings and Online tabs. Data type names in the Type column ending in "[]" are arrays, names ending in List are lists. Choice-based data types can be recognised by the selection combo box in the Type column, data types with optional fields by a checkbox in the ID field.

Primitive data types

0 = zero	Mapped through a data length of 0.
1 = Boolean	Mapped to ULONG (4 bytes) with the values 0x00000001 for TRUE and 0x00000000 for FALSE
2 = Unsigned Integer	Mapped to ULONG (4 bytes). An optimised variant can be used, whereby integer values up to 255 are shown as 1 byte and integer values up to 65535 as 2 bytes (USHORT).
3 = Signed Integer	Mapped to LONG (4 bytes)
4 = Real	Mapped to FLOAT (4 bytes)
5 = Double	Currently not used
6 = Octet String	Mapped to BYTE[]
7 = Character String	Mapped to CHAR[]. I.e. 0-terminated ISO 8859-1 strings. Currently TwinCAT BACnet/IP supports strings of length 256.
8 = Bit String	Mapped to BYTE[]. The highest-value (most significant) byte contains the number of bits that are not used in the following byte (0..7). The highest-value bit of the subsequent byte contains the first Boolean value. Apart from a few exceptions, bit strings have a length of 2 bytes.
9 = Enumerated	Mapped to C enumerations (4 bytes), starting from 0.

10 = Date	Mapped to:
	`struct BACnetDate`
	`{`
	`  BYTE year; // year minus 1900 X'FF' = unspecified`
	`  BYTE month; // (1.. 12) 1 = January X'FF' = unspecified`
	`  BYTE day; // day of month (1..31), X'FF' = unspecified`
	`  BYTE dayOfWeek; // day of week (1..7) 1 = Monday -- 7 = Sunday -- X'FF' = unspecified`
	`}`

11 = Time	Mapped to:
	`struct BACnetTime`
	`{`
	`  BYTE hour; // hour (0..23), (X'FF') = unspecified`
	`  BYTE minute; // minute (0..59), (X'FF') = unspecified`
	`  BYTE second; // (0..59), (X'FF') = unspecified`
	`  BYTE hundredths; // hundredths (0..99), (X'FF') = unspecified`
	`}`

12 = BACnetObjectIdentifier	Mapped to a 4-byte ObjektIdentifier with 10 bits for the object type and a 22-bit object instance
	`struct BACnetObjectIdentifier`
	`{`
	`  DWORD objInst : 22;`
	`  DWORD objType : 10;`
	`}`

Structures

BACnet sequences without optional fields are mapped to structures. An example is shown below.

```
BACnetPrescale ::=
SEQUENCE
{
  multiplier [0] Unsigned,
  moduloDivide [1] Unsigned
}
```

```
struct BACnetPrescale
{
  ULONG multiplier;
  ULONG moduloDivide;
}
```

Optional fields

BACnet sequences with optional fields are mapped to structures with an additional field *validFields*. The additional field indicates which fields are active in the structure. Bit 0 of *validFields* indicates whether the next field (*ObjectIdentifier* in the example) is active. For non-optional fields the respective bit always must be set.

```
BACnetObjectPropertyReference ::= SEQUENCE
{
  objectIdentifier [0] BACnetObjectIdentifier,
  propertyIdentifier [1] BACnetPropertyIdentifier
,
  propertyArrayIndex [2] Unsigned OPTIONAL
}
```

```
struct BACnetObjectPropertyReference
{
  ULONG validFields;
  BACnetObjectIdentifier objectIdentifier; // [0]
  BACnetPropertyIdentifier propertyIdentifier; //
 [1]
  ULONG propertyArrayIndex; // [2] Unsigned OPTIO
NAL
}
```

Choice

A BACnet choice is mapped to a structure with the field *choiceField* and a union. The *choiceField* determines which field of the union is active. The length of the data is determined by the largest field of the union. The *choiceField* is an enumeration (4 bytes).

```
BACnetRecipient ::=
CHOICE
{
  device [0] BACnetObjectIdentifier,
  address [1] BACnetAddress
}
```

```
enum BACnetRecipient_Choice
{
  device,
  address
};struct _BACnetRecipient
{
  BACnetRecipient_Choice choiceField;
  union
  {
      BACnetObjectIdentifier device;
      BACnetAddress address;
  };
}
```

Arrays

BACnet arrays with fixed data sizes are mapped to C arrays. Some BACnet arrays may contain data of variable length (lists or AnyTypes). In this case BACnet arrays are mapped to lists, as described in the following section.

Lists

BACnet lists and some BACnet arrays are mapped to a special list structure. Each list element has a header (ListEntryHdr), followed immediately by the element data. The highest bit of *nEntrySize* (*nEntrySize* & 0x80000000) indicates whether a further element follows (bit 31 = 1), or whether it is the last list element (bit 31 = 0). The size of the subsequent data is coded in the lower 31 bit of *nEntrySize* as the number of bytes. The header contains a further internally used *reserved* field, followed by the data. The following list element starts at a 4-byte-aligned boundary, which follows the data.

```
struct
_ListEntryHdr
{
    DWORD nEntrySize;
    DWORD reserved;
}
```

For example, if a list contains a first element of length 1 with value 1, the list is coded as follows: 08 00 00 01 00 00 00 00 01, followed by 3 alignment bytes 00 00 00. If the next list element is the last element of the list with 1 byte of data with the value 3: 00 00 00 01 00 00 00 00 03. The data length of the example list would be 24 bytes since the last element is followed by 3 alignment bytes.

An empty list is coded as (4 bytes): 00 00 00 00.

AnyTypes

BACnet data of type *ABSTRACT-SYNTAX.&Type* are mapped through the structure AnyType. The field *nSize* contains the size of the following data, including the field dataType (4 bytes). The actual data therefore have the size nSize-4. The field dataType indicates the data type of the data element. A list of data types is provided below.

```
struct
AnyType
{
    ULONG nSize;
    BACnetDataTypes dataType;    //... Data
}
```

The data type codings used for AnyTypes are listed below:

```
#define ARRAYTYPE 0x100
#define LISTTYPE 0x200
enum BACnet  DataTypes
{
// Primitive Data
  DataTypeNull = 0,
  DataTypeBool = 1,
  DataTypeCharacterStringExt = 2,
  DataTypeUnsignedInteger = 3,
  DataTypeSignedInteger = 4,
  DataTypeReal = 5,
  DataTypeDouble = 6,
  DataTypeOctetString = 7,
  DataTypeEnumerated = 8,
  DataTypeBACnetObjectIdentifier = 9,
  DataTypeAddressBinding = 10,
  DataTypeBitString = 11,
  DataTypeDate = 12,
  DataTypeTime = 13,
// Complex Data
  DataTypeDateTime = 14,
  DataTypeArrayIndex = 15,
  DataTypeBACnetCalendarEntry = 16,
  DataTypeBACnetObjectType = 17,
  DataTypeBACnetEventTransitionBits = 18,
  DataTypeBACnetStatusFlags = 20,
  DataTypeBACnetEventState = 21,
  DataTypeBACnetReliability = 22,
  DataTypeBACnetPolarity = 23,
  DataTypeBACnetDateTime = 24,
  DataTypeBACnetEngineeringUnits = 25,
  DataTypeBACnetPriorityValue = 26,
  DataTypeBACnetLimitEnable = 27,
  DataTypeBACnetNotifyType = 29,
  DataTypeBACnetTimeStamp = 30,
  DataTypeBACnetDeviceObjectPropertyReference = 31,
  DataTypeBACnetDeviceStatus = 32,
  DataTypeBACnetServicesSupported = 33,
  DataTypeBACnetObjectTypesSupported = 34,
  DataTypeBACnetSegmentation = 35,
  DataTypeBACnetVTClass = 36,
  DataTypeBACnetVTSession = 37,
  DataTypeBACnetSessionKey = 38,
  DataTypeBACnetRecipient = 39,
  DataTypeBACnetAddressBinding = 40,
  DataTypeBACnetCOVSubscription = 41,
  DataTypeBACnetEventType = 42,
  DataTypeBACnetEventParameter = 43,
  DataTypeBACnetFileAccessMethod = 44,
  DataTypeReadAccessSpecification = 45,
  DataTypeReadAccessResult = 46,
  DataTypeBACnetObjectPropertyReference = 47,
  DataTypeBACnetSetpointReference = 48,
  DataTypeBACnetDestination = 49,
  DataTypeBACnetProgramState = 50,
  DataTypeBACnetProgramRequest = 51,
  DataTypeBACnetProgramError = 52,
  DataTypeBACnetDateRange = 53,
  DataTypeBACnetDailySchedule = 54,
  DataTypeBACnetSpecialEvent = 55,
  DataTypeBACnetClientCOV = 56,
  DataTypeBACnetLogRecord = 57,
  DataTypeBACnetLifeSafetyState = 58,
```

```
DataTypeBACnetLifeSafetyMode = 59,
DataTypeBACnetSilencedState = 60,
DataTypeBACnetLifeSafetyOperation = 61,
DataTypeBACnetBinaryPV = 62,
DataTypeBACnetDaysOfWeek = 63,
DataTypeBACnetWeekNDay = 64,
DataTypeBACnetTimeValue = 65,
DataTypeBACnetValue = 66,
DataTypeBACnetAddress = 67,
DataTypeBACnetPropertyIdentifier = 68,
DataTypeEnumerationValueType = 69,
DataTypeBitFieldBit = 70,
DataTypeBACnetLogDatum = 71,
DataTypeAnyType = 72,
DataTypePresentValue = 73,
DataTypeContextTag = 74,
DataTypeValueChoice = 75,
DataTypeBACnetLogStatus = 76,
DataTypeBACnetRecipientProcess = 77,
DataTypeSubscribeCOVPropertyRequest = 78,
DataTypeBACnetPropertyReference = 79,
DataTypeBACnetPropertyResult = 80,
DataTypeBACnetDeviceObjectPropertyValue = 81,
DataTypeCOVNotificationRequest = 82,
DataTypeEventNotificationRequest = 83,
DataTypeBACnetNotificationParameters = 84,
DataTypeChange_of_Bitstring = 85,
DataTypeChange_of_State=86,
DataTypeChange_of_Value=87,
DataTypeCommand_failure=88,
DataTypeFloating_limit=89,
DataTypeOut_of_Range =90,
DataTypeBACnetPropertyValue=91,
DataTypeComplex_Tag=92,
DataTypeChange_of_life_safety = 93,
DataTypeExtended=94,
DataTypeBuffer_ready =95,
DataTypeUnsigned_range = 96,
DataTypeBACnetPropertyResultValue = 97,
DataTypeServiceCOVPropSubscription = 98,
DataTypeServiceCOVSubscription = 99,
DataTypeCOVNotification = 100,
DataTypeAcknowledgeAlarmRequest = 101,
DataTypeBACnetVTOpenRequest = 102,
DataTypeBACnetPrescale = 103,
DataTypeBACnetScale = 104,
DataTypeBACnetAction = 105,
DataTypeBACnetError = 106,
DataTypeBacnetErrorType = 107,
DataTypeErrorClass = 108,
DataTypeErrorCode = 109,
DataTypeBACnetRejectReason = 110,
DataTypeBACnetAbortReason = 111,
DataTypeBDTEntry = 112,
DataTypeIpEntry = 113,
DataTypeMaskEntry = 114,
DataTypeBACnetEthernetAddress = 115,
DataTypeBACnetLonTalkAddress = 116,
DataTypeBACnetLonTalkNeuronId = 117,
DataTypeBACnetNodeType = 118,
DataTypeBACnetDeviceObjectReference = 119,
DataTypeBACnetActionCommand = 120,
DataTypeGetEventInformation = 121,
DataTypeGetEnrollmentSummaryCriteria = 122,
DataTypeGetEnrollmentSummaryCriteriaResponse = 123,
DataTypeDeviceCommunicationControl = 124,
DataTypeInt16 = 125,
DataTypeUInt16 = 126,
DataTypeUInt8 = 127,
DataTypeInt8 = 128,
DataTypeByte = 129,
DataTypeFDTEntry = 130,
DataTypeBACnetAddress_3 = 131,
DataTypeBACnetAddress_4 = 132,
DataTypeBACnetAddress_5 = 133,
DataTypePersistentDataState = 134,
DataTypeFallbackRealValue = 135,
DataTypeFallbackBinaryValue = 136,
DataTypeRealNull = 137, // PresentValue in AnalogObj
DataTypeEnumeratedNull = 138, // PresentValue in BinaryObj
```

```
  DataTypeUnsignedIntegerNull = 139, // PresentValue in Multistate
  DataTypeUnknown = 140,
  DataTypeBACnetDiagnosisPerformance = 200,
  DataTypeBACnetDiagnosisEthStatistics = 201,
  DataTypeBACnetFrameStatistics = 202,
  DataTypeBACnetInfo = 203,
  DataTypeBACnetDiagnosis = 204,
  DataTypeTcIoEthStatistic = 205,
  DataTypeTcIoEthTxRxErrorCount = 206,
  DataTypeTcIoEthPortStatistic = 207,
  DataTypeBACnetServerStatistics = 208,
  DataTypeBACnetFrameServicesDiag = 209,
// Array types
  DataTypeBACnetCalendarEntryArr = ARRAYTYPE +   DataTypeBACnetCalendarEntry,
  DataTypeBACnetObjectIdentifierArr = ARRAYTYPE +   DataTypeBACnetObjectIdentifier,
  DataTypeCharacterStringExtArr = ARRAYTYPE +   DataTypeCharacterStringExt,
  DataTypeBACnetPriorityValueArr = ARRAYTYPE +   DataTypeBACnetPriorityValue,
  DataTypeBACnetTimeStampArr = ARRAYTYPE +   DataTypeBACnetTimeStamp,
  DataTypeBACnetVTClassArr = ARRAYTYPE +   DataTypeBACnetVTClass,
  DataTypeBACnetVTSessionArr = ARRAYTYPE +   DataTypeBACnetVTSession,
  DataTypeBACnetSessionKeyArr = ARRAYTYPE +   DataTypeBACnetSessionKey,
  DataTypeBACnetRecipientArr = ARRAYTYPE +   DataTypeBACnetRecipient,
  DataTypeBACnetAddressBindingArr = ARRAYTYPE +   DataTypeBACnetAddressBinding,
  DataTypeBACnetCOVSubscriptionArr = ARRAYTYPE +   DataTypeBACnetCOVSubscription,
  DataTypeUnsignedIntegerArr = ARRAYTYPE +   DataTypeUnsignedInteger,
  DataTypeBACnetDestinationArr = ARRAYTYPE +   DataTypeBACnetDestination,
  DataTypeBACnetDeviceObjectPropertyReferenceArr = ARRAYTYPE +   DataTypeBACnetDeviceObjectPropertyR
eference,
  DataTypeBACnetLifeSafetyStateArr = ARRAYTYPE +   DataTypeBACnetLifeSafetyState,
  DataTypeEnumeratedArr = ARRAYTYPE +   DataTypeEnumerated,
  DataTypeReadAccessSpecificationArr = ARRAYTYPE +   DataTypeReadAccessSpecification,
  DataTypeReadAccessResultArr = ARRAYTYPE +   DataTypeReadAccessResult,
  DataTypeBACnetSpecialEventArr = ARRAYTYPE +   DataTypeBACnetSpecialEvent,
  DataTypeBACnetLogRecordArr = ARRAYTYPE +   DataTypeBACnetLogRecord,
  DataTypeBACnetTimeValueArr = ARRAYTYPE +   DataTypeBACnetTimeValue,
  DataTypeBACnetDeviceObjectReferenceArr = ARRAYTYPE +   DataTypeBACnetDeviceObjectReference,
// Listes
  DataTypeBACnetTimeValueList = LISTTYPE +   DataTypeBACnetTimeValue,
  DataTypeBACnetSpecialEventList = LISTTYPE +   DataTypeBACnetSpecialEvent,
  DataTypeBACnetDailyScheduleList = LISTTYPE +   DataTypeBACnetTimeValueList,
  DataTypeCharacterStringExtList = LISTTYPE +   DataTypeCharacterStringExt,
  DataTypeBACnetLogRecordList = LISTTYPE +   DataTypeBACnetLogRecord,
  DataTypeUnsignedIntegerList = LISTTYPE +   DataTypeUnsignedInteger,
  DataTypeReadAccessResultList = LISTTYPE +   DataTypeReadAccessResult,
  DataTypeReadAccessSpecificationList = LISTTYPE +   DataTypeReadAccessSpecification,
  DataTypeBACnetValueList = LISTTYPE +   DataTypeBACnetValue,
  DataTypeBACnetLogDatumList = LISTTYPE +   DataTypeBACnetLogDatum,
  DataTypeBACnetPropertyResultList = LISTTYPE +   DataTypeBACnetPropertyResult,
  DataTypeBACnetPropertyReferenceList = LISTTYPE +   DataTypeBACnetPropertyReference,
  DataTypeCOVNotificationRequestList = LISTTYPE +   DataTypeCOVNotificationRequest,
  DataTypeEventNotificationRequestList = LISTTYPE +   DataTypeEventNotificationRequest,
  DataTypeBACnetPropertyValueList = LISTTYPE +   DataTypeBACnetPropertyValue,
  DataTypeBACnetActionList = LISTTYPE +   DataTypeBACnetActionCommand,
  DataTypeBACnetActionListList = LISTTYPE +   DataTypeBACnetActionList,
};
```

6.3 Automation Interface

The TwinCAT System Manager offers an XML-based interface for programmatic configuration of TwinCAT BACnet/IP. All BACnet elements in a TwinCAT configuration can also be generated programmatically. For further information please refer to the general documentation for the TwinCAT Automation Interface. This section explains which parameters can be used for BACnet-specific components.

Generating BACnet client/server objects

The System Manager offers the CreateChild procedure for generating TwinCAT configuration elements. A TreeItem name, sub-type (for BACnet always 48), an insert position string or sub-type-specific information can be used as parameter.

```
serverItem.CreateChild("Bi_Demo", 48, "", 0x03060203);
```

The last parameter is also referred to as Class-ID. It identifies the element to be created. It is possible to create a BACnet device (CID_TcIoBACnetBACnetDevice), a BACnet server (CID_TcIoBACnetServer) or a BACnet client (CID_TcIoBACnetClient). BACnet server objects can be created with the Class-ID CID_TcIoBACnetObjBase together added with the object type. A BinaryInput object has the Class ID 0x03060203. BACnet client objects with base Class-ID CID_TcIoBACnetObjRemoteBase.

```
ULONG CID_TcIoBACnetBACnetDevice   = 0x03060001;
ULONG CID_TcIoBACnetServer     = 0x03060110;
ULONG CID_TcIoBACnetClient     = 0x03060120;
ULONG CID_TcIoBACnetObjBase    = 0x03060200;
ULONG CID_TcIoBACnetObjRemoteBase  = 0x03060300;
ULONG CID_TcIoBACnetDynamicObject  = 0x030602FE;
```

Property configuration

The procedures ProduceXml and ConsumeXml are used to configure the individual TwinCAT components. Use the System Manager menu "Action" > "Export XML Description ..." to export and view the XML description for test purposes.

```
<Parameter>
  <Name>TcIoBACnetPropCfg_PropTimeDelay</Name>
  <BaseType GUID="{18071995-0000-0000-0000-000000000007}">DWORD</BaseType>
  <PTCID>#x03060171</PTCID>
</Parameter>
...
<Value>
  <Name>TcIoBACnetPropCfg_PropTimeDelay</Name>
  <Value>0</Value>
</Value>
```

The figure shows a section of the XML description of a BACnet object generated with ProduceXml. The structure and the type of a parameter are described in the Parameter section. The actual value is specified in the Value section. The parameter ID of BACnet properties consists of 0x03060100 plus the respective property ID. The following C# source code extract shows how the value of the property TimeDelay, as shown in the "Settings" tab, can be read.

```
XmlDocument doc = new System.Xml.XmlDocument();
s = objItem.ProduceXml(true);
doc.LoadXml(s);
XmlNodeList list = doc.SelectNodes("//Value[./Name='" + "TcIoBACnetPropCfg_PropTimeDelay"+ "']/
Value");
UInt32 timeDelay = Convert.ToUInt32(list[0].InnerText);
```

When writing the property data it is advisable to first read the XML description via ProcuceXml, then to edit the parameter value and finally to read the parameter data via ConsumeXml.

```
XmlDocument doc = new System.Xml.XmlDocument();
s = objItem.ProduceXml(true);
doc.LoadXml(s);
XmlNodeList list = doc.SelectNodes("//Value[./Name='" + "TcIoBACnetPropCfg_PropTimeDelay"+ "']/
Value");
list[0].InnerText = 10;
objItem.ConsumeXml(doc.OuterXml);
```

Depending on the data type of the property, the parameter section may have to be adjusted, in addition to the parameter value. For string data types the string length must be entered under data type STRING(XX).

```
<Parameter>
  <Name>TcIoBACnetPropCfg_PropObjectName</Name>
  <BaseType GUID="{18071995-0000-0000-0000-000000010006}">STRING(5)</BaseType>
  <PTCID>#x0306014d</PTCID>
</Parameter>
...
<Value>
  <Name>TcIoBACnetPropCfg_PropObjectName</Name>
  <String>CAL_0</String>
</Value>
```

For properties with complex data types the configuration data are stored as byte arrays. Data coding is described in more detail in section "Data types". To change complex parameter data, the *BitSize* and the *Elements* of the *ArrayInfo* which specifies the number of bytes must be adjusted in the Parameter section. The actual data are coded in HEXBIN format in the Value section (2 hexadecimal values per byte).

```
<Parameter>
  <Name HideSubItems="1">TcIoBACnetPropCfg_PropDatelist</Name>
  <BitSize>96</BitSize>
```

```
  <BaseType GUID="{18071995-0000-0000-0000-000000000001}">BYTE</BaseType>
  <ArrayInfo>
    <LBound>0</LBound>
    <Elements>12</Elements>
  </ArrayInfo>
  <PTCID>#x03060117</PTCID>
</Parameter>
...
<Value>
  <Name>TcIoBACnetPropCfg_PropDatelist</Name>
  <Data>00000000ffffffff00000000</Data>
</Value>
```

Process data configuration

The process data configuration within the XML description can also be modified via the Automation Interface. Each property that was activated as a process data variable has an entry in section DataArea. In TwinCAT BACnet/IP two DataAreas are available, one for input process data and one for output process data. The example shows the configuration of the DataArea with two input process data. A symbol exists for each activated property. The symbol contains the BitOffset, which determines the position of the property data in the process data image. It is important that the properties start at an aligned address (WORD symbol at addresses that can be divided by 2, DWORD symbols at addresses that can be divided by 4). The ByteSize of the DataArea indicates the size of the symbols.

```
<DataAreas>
  <DataArea>
    <AreaNo AreaType="InputSrc" ChildArea="1">0</AreaNo>
    <Name>Inputs</Name>
    <ContextId>1</ContextId>
    <ByteSize>4</ByteSize>
    <Symbol>
      <Name>PresentValue</Name>
      <BaseType GUID="{18071995-0000-0000-0000-000000000004}">WORD</BaseType>
      <BitSize>16</BitSize>
      <BitOffs>0</BitOffs>
    </Symbol>
    <Symbol>
      <Name>StatusFlags</Name>
      <BaseType GUID="{18071995-0000-0000-0000-000000000004}">WORD</BaseType>
      <BitSize>16</BitSize>
      <BitOffs>16</BitOffs>
    </Symbol>
  </DataArea>
  <DataArea>
    <AreaNo AreaType="OutputDst" ChildArea="1">1</AreaNo>
    <Name>Outputs</Name>
    <ContextId>1</ContextId>
    <ByteSize>0</ByteSize>
  </DataArea>
</DataAreas>
```

In addition to the configuration of the DataArea symbols, the parameters TcIoBACnetPdInCfg and TcIoBACnetPdOutCfg must contain information for the BACnet stack as to which property data should be treated as process data. If the parameters TcIoBACnetPdInCfg and TcIoBACnetPdOutCfg are changed, the data length must be adjusted in the Parameter section. The structure of the two parameters is explained below.

```
<Value>
  <Name>TcIoBACnetPdInCfg</Name>
  <Data>0000000055000000100000006f000000</Data>
</Value>
```

BACnet Server

For each activated process data property, the parameters TcIoBACnetPdInCfg and TcIoBACnetPdOutCfg must contain an element with structure ProcessDataServerCfg. The structure must specify to which bit offset the data should be copied, the property type (*propId*), and for output process data the priority with which the property data are to be copied to the PriorityArray.

```
Struct ProcessDataServerCfg
{
    ULONG bitOffs;
    USHORT propId;
    USHORT priority;
}
```

BACnet Client

Client objects also have further parameters in addition to the symbol configuration, which specify the configuration of the property process data. In client objects a distinction is made between input process data and output process data. The structure fields match the configuration dialogs in the System Manager.

```
Struct ProcessDataClientInCfg
{
    USHORT propId;
    USHORT useCov      : 1;
    USHORT usePolling  : 1;
    USHORT reserved    : 6;
    USHORT tickModulo  : 8;
    ULONG timeDataCOV;
    ULONG timeDataPoll;
    ULONG bitOffs;
    ULONG bitLen;
    floatcovIncrement;
}
```

```
struct ProcessDataClientOutCfg
{
    USHORT propId;
    USHORT useWriteOnChange : 1;
    USHORT usePeriodicWrite : 1;
    USHORT usePriority      : 1;
    USHORT reserved         : 5;
    USHORT tickModulo       : 8;
    ULONG priority;
    ULONG cycleTime;
    ULONG bitOffs;
    ULONG bitLen;
}
```

6.4 Automation Interface and the Microsoft .NET Framework

The functionality presented in this chapter is not covered by the general Beckhoff support. Use at your own risk. We are unable to guarantee the correctness of the presented functions. Feedback on any errors will be appreciated.

The library BACnetExtensionCtrl.dll (supplied with TwinCAT) is available for using the Automation Interface from the Microsoft .NET Framework. The library includes wrapper functions, which facilitate programmatic creation of a TwinCAT BACnet/IP configuration with C#, for example. This chapter uses selected examples to explain how to use the library. Not all options of the implemented functions are discussed in detail. A full overview of all functions can be obtained by using Visual Studio IntelliSense. All BACnet-specific functions are located in the TwinCAT.BACnet namespace.

To use the library, references to the following files have to be added to a .NET project:

- C:\TwinCAT\Io\AddIns\BACnetExtensionCtrl.dll
- C:\TwinCAT\Io\AddIns\TCatSysManagerLib.dll

The examples are based on the following namespace declaration:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TCatSysManagerLib;
using TwinCAT.BACnet.Extension;
using TwinCAT.BACnet.ExtensionHelper;
using TwinCAT.BACnet.BACnetDefinitions;
using System.Xml;
```

First obtain a reference to the TwinCAT System Manager.

```
ITcSysManager ipSysMan = (ITcSysManager)Activator.CreateInstance(Type.GetTypeFromProgID("TCatSysMana
ger.TcSysManager"));
```

BACnet device configuration

In the TwinCAT.BACnet.Extension namespace a BACnetDevice can be created with the aid of the BACnetExtension class (CreateBACnetDevice). Elements are configured in the Automation Interface with the functions ProduceXml and ConsumeXml. ProduceXml is used for read parameters, ConsumtXml for writing. To obtain default parameters, a configuration should normally be read before parameters are modified. This can be done with the wrapper function BACnetExtension.GetXml. Further wrapper functions can then be used to manipulate data in the XML document. Modified parameters can be saved with the function XmlHelper.ConsumeAll. The function XmlHelper.ConsumeParameters can be used in cases where only parameters were modified, but not process data. This makes processing of ConsumeParameters more efficient. In general, it is advisable to use the ConsumeXml functions of the BACnet library, because it includes optimisations that speed up project generation for larger projects.

Most functions of BACnetExtensionCtrl.dll support 2 options. One option uses an ITcSmTreeItem as parameter. In this case ProduceXml and ConsumeXml are called internally. In the second variant an XmlDocument is transferred, and the system expects reading and writing of the data to be realised externally. For complex operations on a TreeItem the functions with XmlDocument should be used, since ConsumeXml is a potentially calculation-intensive operation that should not be called repeatedly.

The following example describes the creation of a BACnet device. The mode for relative AmsNetIDs is activated. This is recommended for automatically generated projects, since in this case a valid AmsNetID is automatically calculated on the target system. The first four digits of the system Net ID are used for this purpose. The function SetNetworkAdapterVirtualName can be used to create the network adapter configuration via a virtual device name. This is sensible on Windows CE-based systems, since the generated configuration can be used on any CXxxx devices, without having to manually select a network adapter during commissioning. By default, the IP address is automatically obtained from the operating system: SetUseOsIpSettingsMode. In addition, automatic summertime/wintertime adjustment through the operating system is activated (ActivateOsTimeMode).

```
ITcSmTreeItemipDevice = BACnetExtension.CreateBACnetDevice(ipSysMan);
XmlDocument doc = BACnetExtension.GetXml(ipDevice);
BACnetDeviceConfiguration.EnableRelativeAmsNetIdMode(doc);
BACnetDeviceConfiguration.SetNetworkAdapterVirtualName(doc, "TCIXPNPE1");
// use ip given by operating system
BACnetDeviceConfiguration.SetUseOsIpSettingsMode(doc, true);
// set daylightsavings status to automatic ( is also default )
BACnetDeviceConfiguration.ActivateOsTimeMode(doc);
// Commit all changes made to BACnet-Device configuration
XmlHelper.ConsumeAll(ipDevice, doc);
```

BACnet server configuration

The function BACnetDeviceConfiguration.CreateBACnetServer can be used to create a BACnet server. To this end an ITcSmTreeItem must be transferred to a BACnet device, with a BACnet ID as parameter. The example explains how to configure a password and how to activate persistent data.

```
ITcSmTreeItem ipServer = BACnetDeviceConfiguration.CreateBACnetServer(ipDevice, 123);
doc = BACnetExtension.GetXml(ipServer);
BACnetServerConfiguration.SetPassword(doc, newBACnetCharacterString("secure"));
// enable persistent data
BACnetServerConfiguration.SetPersistentDataMode(doc, true);
BACnetServerConfiguration.SetPersistentDataUsvMode(doc, false);
BACnetServerConfiguration.SetPersistentDataInterval(doc, 30 * 60); // set persistent data interval t
o 30 min.
// Commit all changes made to BACnet-Server configuration
XmlHelper.ConsumeAll(ipServer, doc);
```

BACnet object configuration

Object creation and property access

The function CreateObject can be used to create new BACnet objects. The system supports a mode with automatic ID generation and explicit configuration with an object ID. The function WriteProperty can be used to manipulate property values. The data type is specified in <> after the function. The generally Automation Interface section describes how the property data types are derived. The function ReadProperty can be used without <> to determine the data types of properties. In this way parameters can be read, and the type of the returned .NET object generated as output.

In the example several properties are configured. For properties of data type String a suitable UTF8 coding is created automatically. In other words, the UCS2 strings used in .NET are converted to the BACnet-specific UTF8 format.

```
// automatic mode -> obj-id and objname are assigned automatically
ITcSmTreeItem biObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.BinaryInput)
;
// manual obj-id configuration
ITcSmTreeItem boObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.BinaryOutput
, 17);
doc = BACnetExtension.GetXml(boObj);
BACnetObjectConfiguration.WriteProperty<string>(doc, BACnetPropertyIdentifier.Description, "компании
 Beckhoff"); // UTF-8 Support
BACnetObjectConfiguration.WriteProperty<string>(doc, BACnetPropertyIdentifier.ObjectName, "bo_12_x")
;
BACnetObjectConfiguration.WriteProperty<bool>(doc, BACnetPropertyIdentifier.OutOfService, true);
BACnetObjectConfiguration.WriteProperty<Byte[]>(doc, BACnetPropertyIdentifier.EventEnable, newByte[]
 { 0x05, 0xE0 });
XmlHelper.ConsumeAll(boObj, doc);
```

In the following example 10 MultistateValue objects are generated and configured with an increasing number of states (NumberOfStates).

```
// create 10 multistate value objects with increasing number of states
for (uint i = 0; i < 10; i++)
{
    ITcSmTreeItem mvObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.MultiSta
teValue);
    doc = BACnetExtension.GetXml(mvObj);
    CharacterStringExtList stateText = newCharacterStringExtList();
    for( int j=0; j<i+2; j++ )
    stateText.AddCharacterString("MV_State_" + j.ToString());
    BACnetObjectConfiguration.WriteProperty<CharacterStringExtList>(doc, BACnetPropertyIdentifier.St
ateText, stateText);
    BACnetObjectConfiguration.WriteProperty<string>(doc, BACnetPropertyIdentifier.Description, "Obje
ct with " + i.ToString() +" states");
    BACnetObjectConfiguration.WriteProperty<UInt32>(doc, BACnetPropertyIdentifier.NumberOfStates, i
+ 2);
    XmlHelper.ConsumeAll(mvObj, doc);
}
```

The library can also be used to configure complex BACnet properties. In the following example the Datelist property of a Calendar object is configured with DateRange, Date and WeekNDay entries. The WriteProperty library function converts the .NET classes/objects to serial data streams (byte arrays) and stores them in the XmlDocument in HexBin format.

```
ITcSmTreeItem calObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.Calendar);
doc = BACnetExtension.GetXml(calObj);
BACnetCalendarEntry[] calEntries = newBACnetCalendarEntry[3];
calEntries[0] = newBACnetCalendarEntry(newBACnetDate(112, 1, 1, 255)); // Add date: 1.Jan.2012
calEntries[1] = newBACnetCalendarEntry(newBACnetDateRange(newBACnetDate(255, 1,1,255), newBACnetDate
(255, 7,19,155))); // DateRange 1.7.-19.7
calEntries[2] = newBACnetCalendarEntry(newBACnetWeekNDay(255, 255, 7)); // every sunday
BACnetObjectConfiguration.WriteProperty<BACnetCalendarEntry[]>(doc, BACnetPropertyIdentifier.Datelis
t, calEntries);
XmlHelper.ConsumeAll(calObj, doc);
```

The following examples shows the configuration of a schedule object's exception schedule. The exception list will be active if the calendar object created before is active. The example demonstrates blinking with a frequency of 10 Hz of a BinaryOutput object having the object identifier instance 10.

```
// remember calendar object for schedule configuration
BACnetObjectIdentifier calObjId = BACnetObjectConfiguration.ReadProperty<BACnetObjectIdentifier>(doc
, BACnetPropertyIdentifier.ObjectIdentifier);
ITcSmTreeItem schedObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.Schedule)
;
doc = BACnetExtension.GetXml(schedObj);

// blink with 10 Hz
BACnetSpecialEvent specialEvent = newBACnetSpecialEvent(calObjId, 1, newBACnetTimeValueList());
    specialEvent.AddBACnetTimeValue(
    newBACnetTimeValue(newBACnetTime(255, 255, 255, 0),
    newBACnetValue(BACnetBinaryPV.active)));
    specialEvent.AddBACnetTimeValue(
    newBACnetTimeValue(newBACnetTime(255, 255, 255, 50),
    newBACnetValue(BACnetBinaryPV.inactive)));

BACnetObjectConfiguration.WriteProperty<BACnetSpecialEventList>(
```

```
    doc,
    BACnetPropertyIdentifier.ExceptionSchedule,
    newBACnetSpecialEventList(newBACnetSpecialEvent[] { specialEvent }));
// set object/property reference to BO:0
BACnetObjectConfiguration.WriteProperty<BACnetDeviceObjectPropertyReference[]>(
    doc,
    BACnetPropertyIdentifier.ListOfObjectPropertyRefs,
    newBACnetDeviceObjectPropertyReference[]{newBACnetDeviceObjectPropertyReference(
    newBACnetObjectIdentifier(BACnetObjectType.BinaryOutput, 17), BACnetPropertyIdentifier.PresentVa
lue)});
    XmlHelper.ConsumeAll(schedObj, doc);
```

Process data configuration

The library can also be used to activate process data of BACnet objects and link them via the LinkVariables functions of the Automation Interface. In the example a BinaryInput object and a BinaryOutput object is created, and the PresentValue properties are linked via Device2Device mapping. The function XmlHelper.GetBaseTypeBitSize can be used to determine the bit length of the property process data, which is required as parameter for the function LinkVariables.

```
// create to objects to be linked together ( BO is on whenever BI is on )
ITcSmTreeItem pdBiObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.BinaryInpu
t);
ITcSmTreeItem pdBoObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.BinaryOutp
ut);
// add property PresentValue as input process data
doc = BACnetExtension.GetXml(pdBiObj);
string biVarName = BACnetObjectConfiguration.AddInputSymbolChecked(doc, BACnetPropertyIdentifier.Pre
sentValue, BACnetObjectType.BinaryInput, false, null);
XmlHelper.ConsumeAll(pdBiObj, doc);
// add property PresentValue as output process data with priority 10
doc = BACnetExtension.GetXml(pdBoObj);
string boVarName = BACnetObjectConfiguration.AddOutputSymbolChecked(doc, BACnetPropertyIdentifier.Pr
esentValue_Priority10, BACnetObjectType.BinaryOutput, false, null);
XmlHelper.ConsumeAll(pdBoObj, doc);
// get Property description to get process data size of present value properties
PropEntryDesc eDesc = PropertyDescriptions.GetPropDesc(BACnetPropertyIdentifier.PresentValue);
uint bitLen = XmlHelper.GetBaseTypeBitSize(eDesc.GetProcessDataTmcType(BACnetObjectType.BinaryInput)
);
// link the objects via process data
ipSysMan.LinkVariables(pdBiObj.PathName + "^Inputs^" + biVarName, pdBoObj.PathName + "^Outputs^" + b
oVarName, 0, 0, (int)bitLen);
```

Optional properties

Optional properties can be activated and deactivated with the library functions EnableProperty and DisableProperty. The example also demonstrates the process of iteration over all properties of an object type. DisableProperty supports disabling of related properties. If a property is disabled that requires the existence of a second property, this is also automatically disabled. For example, to deactivate all properties of Instrinsic Reporting, it is sufficient to deactivate the property NotifyType.

```
ITcSmTreeItem bvObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.BinaryValue)
;
doc = BACnetExtension.GetXml(bvObj);
// disable all optional properties
foreach (BACnetPropertyIdentifier prop inPropertyDescriptions.GetSupportedActiveOnlineProps(doc, BAC
netObjectType.BinaryValue, true))
{
    eDesc = PropertyDescriptions.GetPropDesc(prop);
    if (PropertyDescriptions.IsOptProp(BACnetObjectType.BinaryValue, prop))
    PropertyDescriptions.DisableProperty(doc, BACnetObjectType.BinaryValue, prop);
}
XmlHelper.ConsumeAll(bvObj, doc);
```

Write protection configuration

The function EnablePropertyWriteProtection can be used to activate write protection for a property.

```
ITcSmTreeItem bv2Obj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.BinaryValue
);
doc = BACnetExtension.GetXml(bv2Obj);
// enable write protection for all writable properties
foreach (BACnetPropertyIdentifier prop inPropertyDescriptions.GetSupportedActiveOnlineProps(doc, BAC
netObjectType.BinaryValue, true))
{
    eDesc = PropertyDescriptions.GetPropDesc(prop);
```

```
    if (PropertyDescriptions.IsWritable(BACnetObjectType.BinaryValue, prop))
    PropertyDescriptions.EnablePropertyWriteProtection(doc, BACnetObjectType.BinaryValue, prop);
}
XmlHelper.ConsumeAll(bv2Obj, doc);
```

The function ObjectDescriptions.SupportedOfflineProps can be used to iterate over all properties that can be configured in the Settings dialog for the objects. In contrast, GetSupportedActiveOnlineProps iterates over all object properties that exist at runtime.

BACnet client configuration

Functions are also available for configuring remote BACnet devices. CreateBACnetClient creates a BACnet client with specified BACnet ID. The function CreateRemoteObject can be used to create BACnet objects. Process data of client objects can be activated and linked via the LinkVariables function of the Automation Interface. A RemotePdCfg has to be transferred to the functions AddInputSymbol and AddOutputSymbol, which configures the process data processing mode. RemotePdCfg consolidates parameters for read and write property access. This means that the remote configuration can initially be created and then be used for creating input and output symbols. The modes correspond to the options that can be configured in the System Manager (Polling, COV, WriteOnChange etc.). The parameters RemotePdReadTickModulo and RemotePdWriteTickModulo can be used to specify in which cycle client interactions are to take place. In systems with many client objects, it is advisable to distribute the interactions over several cycles.

Functions with the suffix Checked verify whether a process data variable has already been activated before attempting to activate it. If necessary, the variable is deactivated and then reactivated.

```
ITcSmTreeItem ipClient = BACnetDeviceConfiguration.CreateBACnetClient(ipDevice, 42);
// create BACnet remote object
ITcSmTreeItem remAV = BACnetClientConfiguration.CreateRemoteObject(ipClient, BACnetObjectType.Analog
Value, 17);
doc = BACnetExtension.GetXml(remAV);
// add input symbol - read data from remote device
BACnetObjectConfiguration.AddInputSymbolChecked(
    doc,
    BACnetPropertyIdentifier.PresentValue,
    BACnetObjectType.AnalogValue,
    true,
    newBACnetObjectConfiguration.RemotePdCfg(BACnetObjectConfiguration.RemotePdCfg.RemotePdReadMode.
COV_POLL, 1000, 1.0f, 240));
// add output symbol - with explicit remote configuration
// write property present value on change with priority 10
BACnetObjectConfiguration.RemotePdCfg remoteCfg = newBACnetObjectConfiguration.RemotePdCfg();
remoteCfg.WriteMode = BACnetObjectConfiguration.RemotePdCfg.RemotePdWriteMode.ONCHANGE;
remoteCfg.UsePriority = true;
remoteCfg.priority = 10;
remoteCfg.RemotePdWriteTickModulo = 2;
remoteCfg.resubsInterval = 240;
remoteCfg.cycleTimeWrite = 2000;
BACnetObjectConfiguration.AddOutputSymbolChecked(
    doc,
    BACnetPropertyIdentifier.PresentValue,
    BACnetObjectType.AnalogValue,    true,
    remoteCfg);
XmlHelper.ConsumeAll(remAV, doc);
```

Notification sink configuration

The BACnetExtensionCtrl library can also be used to create a notification sink. In the following example a notification sink with the process ID 100 is created. Subsequently a NotificationClass object is created and configured for sending local EventNotifications to the notification sink.

```
ITcSmTreeItem ipSink = BACnetDeviceConfiguration.CreateNotificationSink(ipDevice);
doc = BACnetExtension.GetXml(ipSink);
BACnetNotificationSinkConfiguration.SetProcessId(doc, 100);
XmlHelper.ConsumeAll(ipSink, doc);
ITcSmTreeItem ipNotifyObj = BACnetServerConfiguration.CreateObject(ipServer, BACnetObjectType.Notifi
cationClass);
doc = BACnetExtension.GetXml(ipNotifyObj);
BACnetDestination[] destArr = newBACnetDestination[1];
destArr[0] = newBACnetDestination();
destArr[0].processIdentifier = 100;
destArr[0].recipient = newBACnetRecipient(newBACnetObjectIdentifier(BACnetObjectType.Device, 123));
BACnetObjectConfiguration.WriteProperty<BACnetDestination[]>(doc, BACnetPropertyIdentifier.Recipient
List, destArr);
XmlHelper.ConsumeAll(ipNotifyObj, doc);
```

The configuration can then be saved. The SaveConfiguration function of the Automation Interface is used to save the configuration.

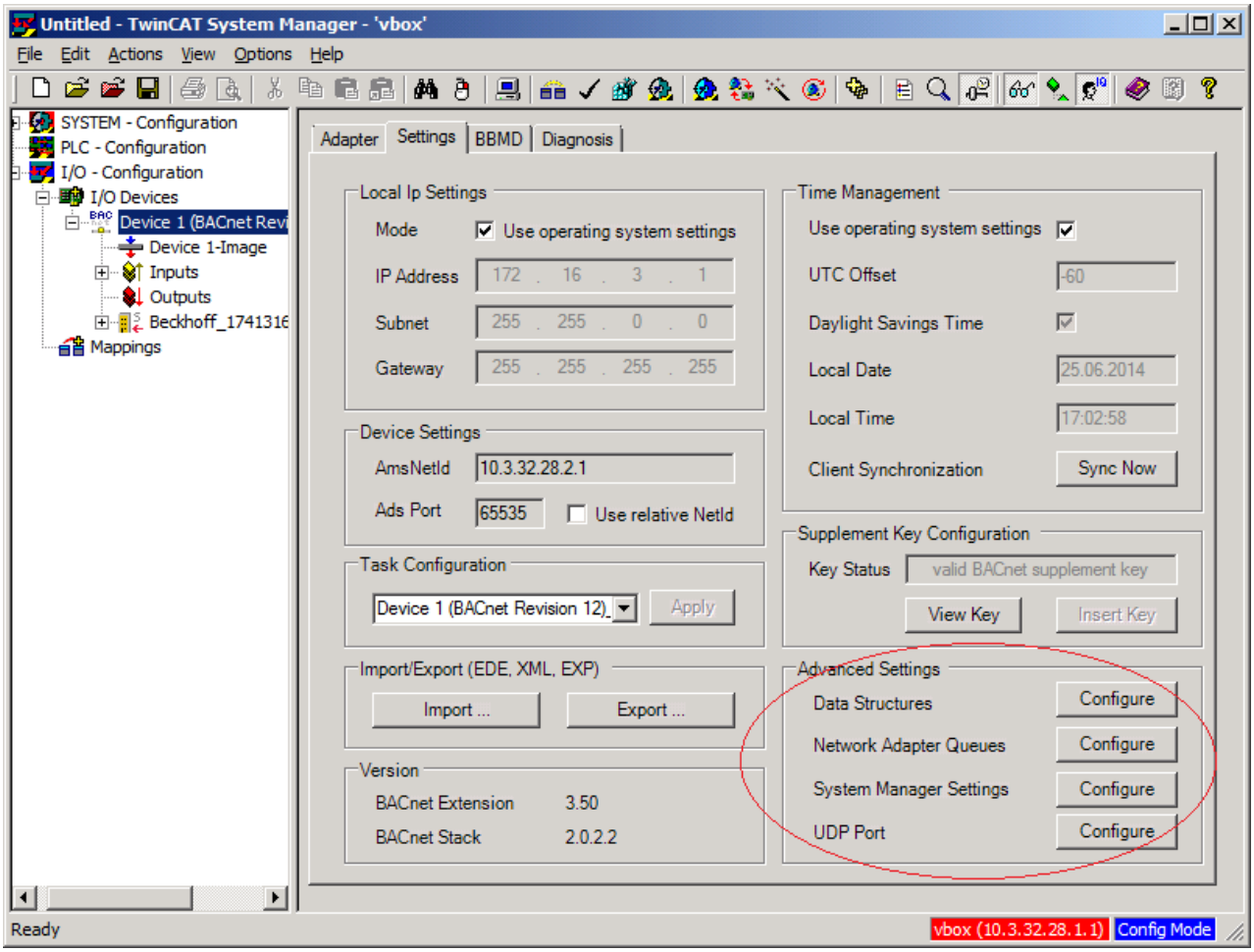
```
ipSysMan.SaveConfiguration("C:\\BACnetExample.tsm");
```

All source code extracts presented in this section are available in the form of an executable Visual Studio project in the following .zip file. The project can also be used as a starting point for a separate project.

https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749058955/.zip https://infosys.beckhoff.com/content/1033/tcbacnet/Resources/12749058955/.zip

6.5 Advanced settings

Advanced Settings can be made in the Settings tab of the BACnet device, as explained below.



Memory-specific parameters

Data structure-specific parameters can be adjusted via the Advanced settings. Essentially these parameters affect the available memory in the BACnet stack.

Parameter	Default value (min./max.)	Description
MAX_OBJ_SUBS_COV_ENTRIES	10 (1 - 255)	The COV subscriptions are managed in TwinCAT BACnet/IP for each object. The default setting is 10 subscriptions per BACnet object. These parameters can be used to specify the maximum number of COV subscriptions per object. For performance reasons it may make sense to reduce

Parameter	Default value (min./max.)	Description
		the value of this parameter. in order to avoid excessive COV notifications if there are many changes in the system.
MAX_PROPERTY_ARRAYELEMENTS	200 (10 - 10000)	For BACnet properties with array data types in TwinCAT BACnet/IP (data type ends with []), a memory for a pre-defined number of elements is created on startup, so that elements can be added dynamically during operation. By default up to 200 elements can be stored in an array property. If more than 200 elements are configured in an array property in the Settings dialog, additional memory capacity is automatically created during startup in order to accommodate all elements.
LIST_ALLOC_MEM_BYTES	8192 (500 - 4294967295)	For properties with list data type in TwinCAT BACnet/IP (data type ends with list), the memory capacity required per element may vary. On startup a predefined memory is created, so that elements can also be added and modified dynamically for these properties. The default configuration is 8 kbytes. Via this parameter the memory capacity per list property can be defined. As for the array properties, if the property is configured with more memory in the Settings dialog, more memory is created.
MAX_DEVICE_BINDINGS	1000 (10 - 10000)	Each device in the BACnet network (which sends an I-Am packet) is administered in the DeviceBindings table, where all the parameters required for the communication with a device are stored. The size of this table can be specified via this parameter. The table is cleared on each restart and when a scan process is triggered (e.g. in the System Manager).
MAX_OBJECTS	1000 (10 - 4294967295)	BACnet objects for each BACnet client and BACnet server are administered in hash tables. This parameter defines the size of these tables. If a configuration is intended to contain BACnet servers and BACnet clients with more than 1000 objects, this parameter has to be increased. The number of objects refers to the total number of all objects configured under BACnet servers and BACnet clients and must be selected accordingly. When BACnet objects are added in the System Manager or through scanning of BACnet clients this parameter is automatically incremented in steps of 100. To support creation of a large number of dynamic objects at runtime this parameter should be increased manually.
MAX_CLIENTS	255 (10 - 10000)	The clients created in the System Manager are administered in a list. If more than 255 (statically configured in the System Manager) clients are to be used, this parameter must be increased.
MAX_TRENDLOG_ENTRYSIZE	56 (56 - 10000)	For TrendLog objects the memory for the LogBuffer is also created on startup. The parameter BufferSize can be used to specify the number of entries as a BACnet property. The memory size required for an entry depends on the logged property. The minimum size of 56 bytes is adequate for error log entries and for logging simple data types (REAL, Integer ...). If the entry:

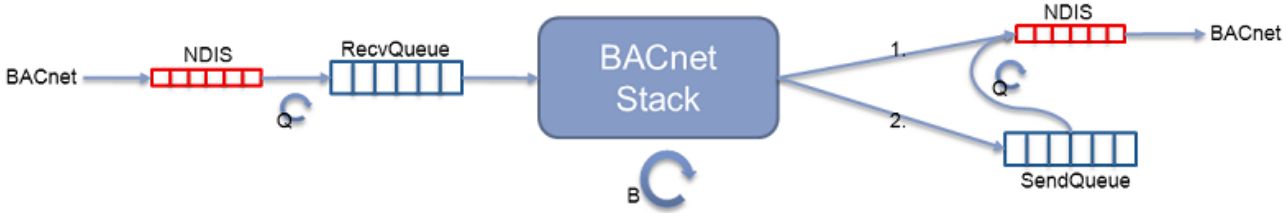
Parameter	Default value (min./max.)	Description
		"Resources:No_space_to_write_property" appears in the LogBuffer, the available memory is not sufficient and the value of this parameter must be increased.
DYN_TRENDLOG_BUFFERSIZE	100 (10 - 10000)	Since the memory for the TrendLog buffer is created when the BACnet object is generated, for dynamically created TrendLog objects the value of the BACnet property BufferSize must be known. This is specified via this parameter.
MAX_MULTISTATE_DYNSTATES	255 (2 - 10000)	For MultiState objects memory must be created for the BACnet property StateText during startup. The maximum possible number of states (NumberOfStates) must be known, since the BACnet property NumberOfStates can be modified during runtime. This parameter defines the maximum value of the property NumberOfStates and therefore the amount of memory to be created during startup.
MAX_RECV_BAC_SEGM_FRAMES	1000 (100 - 10000)	The components of segmented frames have to be held in the buffer, so that they can be recombined to the original message once all components have been received. This parameter has to be set accordingly, depending on the envisaged parallel processing of segmented messages.
MAX_SEND_BAC_SEGM_FRAMES	1000 (100 - 10000)	During sending messages may also have to be segmented. Frames prepared for sending are stored in the table. The size of this table can be set via this parameter.
IO_STARTUP_TIMEOUT	2000 (0 - 10000)	The initialization of a connected fieldbus (K-bus, EtherCAT) may take longer than starting of the BACnet server. This may result in the automatic fieldbus monitoring (Reliability, StatusFlags) setting all IO BACnet objects to an error state, which on the one hand generates trend log entries with set fault bits in the StatusFlags, but may also lead to notifications being sent. The StateMachine of the BACnet server can therefore be delayed by an I/O start time, in order to await successful initialization of the fieldbus during startup.
MAX_PARALLEL_COV_SUBS	250 (1 - 255)	When using client process data in COV mode, slow remote stations may result in communication failures, or COV subscriptions may not be completed successfully. This parameter can be used to set the maximum number of concurrent COV subscription requests per BACnet client.
AVERAGING_MAX_BUFFERSIZE	1000 (1 - 10000)	Sets the size of the internal buffer for averaging. This buffer stores data for the calculation of statistical data (average value etc.). If the calculation is based on more than 1000 samples, this parameter must be increased.
MAX_SENDFRAMES_PERCYCLE	4294967295 (1 - 4294967295)	In order to limit temporary peak loads relating to sent BACnet telegrams, the number of frames to be sent per cycle can be limited. With a cycle time of 50 ms and a value of 1, the maximum number of frames that are sent per second is 20. This mechanism can be used to prevent flooding of the BACnet network in certain cases. However, in

Parameter	Default value (min./max.)	Description
		general it is better to find the cause for the large number of sent frames and rectify it, e.g. through the use of filters for AnalogInput objects.
PROPOSED_WINDOW_SIZE	16 (1 - 127)	Segmentation is used in BACnet when transferring large amounts of data. The maximum number of open unconfirmed sent messages is "negotiated" when the connection is established. This parameter determines what "window size" (i.e. the number of unconfirmed messages) is suggested by this device. For the CX8091, this value must be reduced to 4, because otherwise file access will not work due to limited network storage.
FREERUN_CYCLETIME_MODULO	10 (1 - 100)	In Freerun mode, the internal state machines of the BACnet objects are called with a cycle time of 2 ms. For large BACnet configurations this can lead to system overload. This parameter specifies after how many cycles BACnet objects are called. In the default case (10), the BACnet objects are therefore processed only every 20 ms.
ARCHIVE_BUFFER_SIZE	65535 (65535 - 16777216)	A buffer is used for persistent data that have to be stored. This buffer should be at least as large as the largest property in the system. Typically, it is the property LogBuffer of a TrendLog object. With a size of 40 bytes per entry (normal value, no error), the default value of 65535 for this parameter is exceeded from 1638 entries. In this case, this parameter should be increased. With a LogBuffer of 3000 entries, the value should therefore be set to 120,000 bytes (40*3000)

Network adapter queues

Small changes can easily lead to the sending of many messages in BACnet. E.g. because many COV subscriptions to a certain property exist, or many *notification recipients* have been entered in a NotificationClass. Due to this asynchronous nature, BACnet runs in a low-priority task, in order to avoid delays in the PLC and the IO subsystem. For large configurations, the cycle time of the task should be configured in the range 40 ms to 100 ms. This processing "in the background" results in network adapter functions being executed less frequently. When receiving messages, this may result in buffer overflow in the driver and therefore loss of messages. Network hardware overload situations can also occur during sending of BACnet messages, if too many messages have to be sent within a cycle.

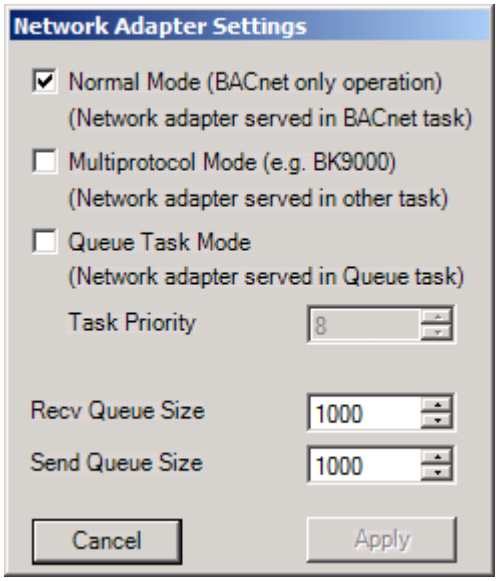
BACnet therefore offers different modes for processing network messages, in order to optimize the functionality of the network adapter. To cope with overloading during sending and receiving, BACnet uses a send buffer and a receive buffer with sizes that can be defined separately. The following diagram illustrates the operating principle of these "queues". On the left, BACnet messages are received via the NDIS driver, for example. A task (Q) copies these messages to the Recv queue. The messages are then processed in the context of the BACnet task (B), and potential messages are sent. If adequate resources are available in the network adapter, the message is sent directly (1); otherwise, messages are buffered in the Send queue and sent in the context of the task Q.



BACnet supports 3 network adapter modes, which essentially differ in terms of the context of which task (Q shown in the image) the functions for receiving and sending the network adapter are called:

- In **Normal mode**, network adapter-specific functions are executed in the context of the BACnet task itself. At the start of the cycle, all messages are copied into the receive buffer and processed further from there.
- In **multi-protocol mode**, network adapter-specific functions are executed in the context of another task. This mode must be used if further Ethernet-based drivers (such as real-time Ethernet) are operated in parallel with BACnet via the same network interface.
- In **queue task mode**, network adapter-specific functions are executed in the context of a special queue task. This mode is used when the BACnet task is operated with high cycle times, for example. The high-priority queue task with lower cycle time copies the messages from the network driver into the internal BACnet Recv queue and/or send messages from the Send queue. As a result, faster use is made of the hardware resources, while complex calculations continue to be executed with low priority.

The "Network Adapter Settings" dialog can be used for network adapter-specific settings. The size of send and receive queues can specified separately. In addition, the priority of the queue task can be specified, and the mode for the network adapter operation can be selected, of course.



The individual modes should be selected based on the following recommendations:

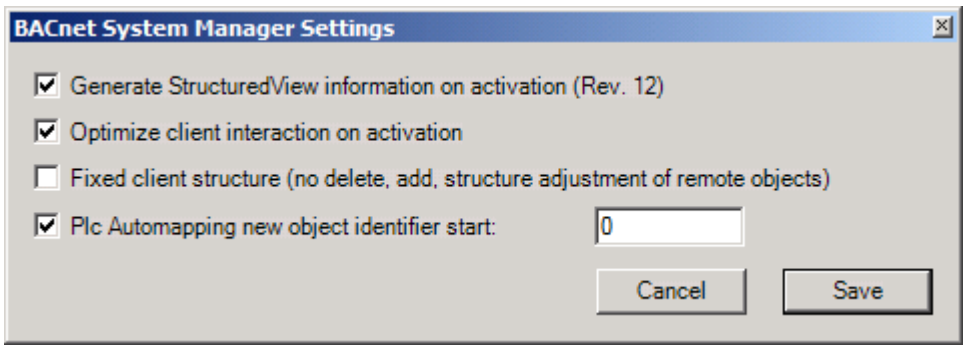
- Normal Mode
  - Short BACnet cycle times (<20 ms)
  - High-performance device (CX50XX)
- Multiprotocol Mode
  - If real-time Ethernet is used!
  - The trigger cycle should not be too long (<10 ms)
- Queue Task Mode
  - Low-capacity devices (CX9010, CX8091, CX9020)
  - Always for CCAT (CX8091)
  - Long BACnet cycle time (>20 ms)
  - Queue task with 1 ms at best (costs approx. 5-10% performance)

Typical setting for CX8091

BACnet task: 50 ms – 100 ms

SystemManager Settings

The dialog "BACnet System Manager Settings" can be used for BACnet-specific settings for the configuration software, i.e. the TwinCAT SystemManager.



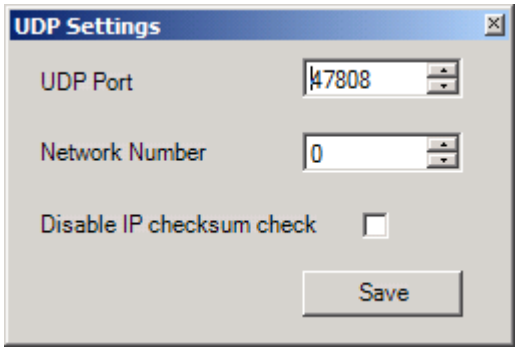
The **"Generate..."** item can be used to specify whether StructuredView-specific property contents are created automatically when a BACnet configuration is activated. This enables the hierarchical representation of the SystemManager to be displayed in the BACnet network. This item is enabled by default, i.e. the StructuredView information is generated. In principle, StructuredView enables display of several views (e.g. at the device level, location-dependent or commercial). To implement specific requirements for StructuredView objects in a project, which go beyond the order of BACnet objects, this item can be disabled, so that StructuredView objects can be configured manually.

The **"Optimize ..."** item can be used to specify whether the interaction modulo factors for the interaction with remote BACnet devices should be optimized automatically. This feature is enabled by default.

The **"Fixed ..."** item can be used to specify that the object structure under a BACnet client should not be changed during scanning. If only a few objects are integrated in the cross communication, it may be useful to add only the actually used BACnet objects via PLC automapping. If this item is enabled, the object identifiers are verified during scanning, based on the BACnet object name. If changes are detected, new ObjectIdentifiers can be applied. This feature is also useful if only the object names are available during project planning, but not the ObjectIdentifier.

The **"Plc ..."** item can be used to specify the starting point of instance numbers for object identifiers that were generated automatically during PLC automapping. This functionality is required for the Remap function, in order to avoid collisions with outdated persistent data (.wbp).

UDP-Port Settings



The dialog "UDP settings" can be used to specify the UDP port used by BACnet.

The BACnet network number can also be set here. In pure BACnet/IP systems, configuration of the network number is usually not required. If the network number is set to a value other than 0 for purpose of additional splitting of a BACnet/IP network, then the following behavior is implemented in the BACnet stack:

- The network number is sent in the I-Am frame
- Only BACnet frames without network number or with the configured network number are accepted
- Optionally, the configured network number is transferred as the source address

The "*Disable IP checksum check*" item can be used to disable verification of the IP checksum. This function should only be activated if it is essential to communicate with devices that do not implement a correct checksum calculation.

6.6 Support and Service

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

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

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

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

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

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

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

Beckhoff Service

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

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

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

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

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

More Information:
www.beckhoff.com/ts8020

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