BACnet

BACnet communication protocol

Supported device types and versions

The BACnet communication protocol (**B**uilding **A**utomation and **C**ontrol **Net**works) implements ANSI/ASHRAE 135-2001 standard.
This implementation was tested with the following devices:

- Siemens
  - Desigo PXM20 (Control unit, LON interface, BACnet over LON)
  - Desigo PXC22 (Control station, LON interface, BACnet over LON)
  - Desigo PXC22-E.D (Control station, Ethernet interface, BACnet/IP)
  - Desigo PXG80-N (BACnet router, Ethernet interface, LON interface, BACnet/IP, BACnet over LON)
- Delta Controls
  - DSC-1212E (System controller, Ethernet interface, BACnet/IP)
  - DAC-633 (Application controller, MS/TP interface connected to DSC-1212E, which works as BACnet router)
  - DAC-633 (Application controller, MS/TP interface connected to the Moxa 5250 serial/ethernet converter and communicated directly as a BACnet MS/TP device in UDP mode)
  - DAC-1146 - (Application controller, MS/TP interface connected to DSC-1212E which works as BACnet router)
- Sauter
  - EYK220F001 (Automation station, Ethernet interface, BACnet/IP)
  - EYR203F001 (Universal controller connected to EYK220F001)
  - EYR207F001 (Universal controller connected to EYK220F001)
- York
  - BACnet MS/TP MicroGateway (Communication card for York coolers, RS485 interface, BACnet MS/TP)
- SE-Elektronic GmbH:
  - E-DDC3.1 (DDC automation station, Ethernet interface, BACnet/IP)
- Klimasoft
  - BACnet/Mbus converter MBG-MSTP: Converter from BACnet to Mbus protocol (RS485 interface, BACnet MS/TP)

Characteristics of the current version:

- Communication in Ethernet (BACnet/IP) and LONTalk networks.
- Limited support of MS/TP network (master-slave token-passing on RS-485): without automatic searching for Master stations.
- Support of BACnet router (the connection between BACnet/IP and LONTalk networks).
- Reading and writing of simple values (binary, integer, real, strings, date, time, etc..) and any ASN sequences.
- Support of the polling method of data reading (messages ReadProperty-Request and ReadPropertyMultiple-Request messages)
- Support of change method of data reading (an optional registration by SubscribeCOV-Request or SubscribeCOVProperty-Request and following processing of ConfirmedCOVNotification-Request and UnconfirmedCOVNotification-Request).
- Writing the values by WriteProperty-Request.
- Dynamic change of I/O tag address by TELL command SETPTADDR (to read the values of Schedule objects).
- Work with objects of Schedule type (schedules).

BACnet protocol considers all the participants of the communication as the network devices. Each network device contains at least one (mostly just one) object *Device* (its Object Identifier must be unique within the whole network). This object contains other objects of defined types (Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Calendar, Command, Event, Group, File, etc.). The object detection - see a description of Who-Is and Who-Has Request types.
Each object has properties, which can be mandatory or optional. Moreover, each producer of BACnet devices can implement other properties when necessary.
The messages in the BACnet protocol are related to the manipulation of objects and their properties. They are defined with the help of ASN.1 (Abstract Syntax Notation version 1) and encoded by a simple version of BER (Basic Encoding Rules - encoding of ASN.1 messages).
The messages contain, besides fixed defined items, also items of 'Abstract Syntax & Notation' type. It means that any sequence (or "tree"), the meaning of which is defined by an implementer, can be in its place in the message. When using BER, it enables parsing of the message with the unknown items. BER defines two basic item types (tags): application and context.
The application tags are predefined:

- **Null** - empty value
- **Boolean** - yes/no
- **Unsigned** - positive integer
- **Signed** - integer
- **Real** - 4-byte real number
- **Double**- 8-byte real number
- **Octet String** - a sequence of character
- **Character String** - charset + text string
- **Bit String** - a sequence of bits
- **Enumerated Value** - enumerated value
- **Date** - date
- **Time** - time
- **Object Identifier** - identifier of the object (32-bit number, it consists of 10-bit number - Object Type and 22-bit number - Instance)

The context tags depend on the context (on the position in the message). Without knowing the context (a description of the message that is being parsed), it is possible to find out that the context tag No. 5 with the length of 4 bytes is on the particular position, but you need additional information whether the value is Unsigned, Signed, Real, Bitstring or a different type of value.
Besides the simple application and context tags, the properties may be also complex:

- **Sequence** - the sequence that consists of other properties (both simple and complex one), they are either required or optional
- **Sequence of** - the sequence of N-tuple of properties
- **Choice** - one of N possibilities

Example - a dump from trace file of a KOM process with the debug enabled:

```
=== ASN Body beg ===
objectIdentifier (tag 0) OBJID 0 analog-input,10
listOfResults (tag 1) SEQUENCE {
  propertyIdentifier (tag 2) ENUM 85 present-value
  propertyValue (tag 4) SEQUENCE {
   ENUM 1
  }
}
=== ASN Body end ===
```

Interpretation:
It is the Sequence of two tags: *objectIdentifier* is a context tag with the number 0, of Object Identifier type. Its value is Object type=0 (analog input), Instance=10.
The tag listOfResults is a context tag with number 1 and it is a Sequence of two tags. The first one is *propertyIdentifier*. It is a context tag No. 2, Enum type, value = 85 that corresponds to "present-value". A second tag is a context one, No.4. It is a sequence that contains one Enumerated Value tag with the value=1 (application tag).
To parse this message, the D2000 KOM process must know ASN.1 definition of a message. Without it, the process can find out that the message contains the context tag 2 (value=1 byte) but cannot know that it is Enumerated Value. It is not able to interpret this byte (it could be Enumerated Value, Unsigned or Signed number) and does not know that the name of this context tag is *propertyIdentifier* and the value 85 corresponds to "present-value".

The properties of objects are mapped to I/O tags in the D2000 configuration. Due to the existence of context tags, you may specify an Application tag in the I/O tag. It determines the interpretation of the context tag. The parameter Complex address defines "a path" in the parse "tree" to get the values from the sequence that is defined by the implementer.

Communication line configuration

- Communication line category: TCP/IP-UDP, LonWorks, Serial, SerialOverUDP Device Redundant.
- TCP/IP-UDP parameters:
  - Host: IP address or of the network interface that is used for communication by the D2000 KOM process. A symbolic name that can be translated to an IP address can be entered too.
    Note: a symbolic name **ALL** or ***** can be entered - in which case all available interfaces are used.
  - Port: UDP port number that is used for communication by the D2000 KOM process (according to standard 0xBAC0, i.e. 47808).

Note: The parameters of the backup server (Host and Port) are not used in this protocol.

Line protocol parameters

Keyword	Full name	Meaning	Unit	Default value
DBGI	Debug Input	Debug information about the input data. Meaning of the bits: • 1. bit - debugging of ASN message parsing • 2. bit - debugging of the I/O tag names that received a new value • higher bits - not used	-	0
DTQ	Debug Timeout Queue	Debug information about messages in time queue.	-	False
DI	Device Instance	Non-zero value causes that KOM process answers to Who-Is request by I-Am message. It contains a defined Device Instance. Zero value causes that Who-Is request is ignored.	-	0
DOW	Display DayOfWeek	If the value is True, the conversion of the **Date** type tag to the text I/O tag will also contain the item "Day Of Week" (Monday=1 .. Sunday=7, undefined value=255), e.g. "20.12.2022**.2**". If the value of the parameter is False, the I/O tag contains only the items "Day", "Month", "Year", e.g. "12/20/2022".	-	False
RB	Receive Buffer	(for TCP/IP-UDP lines only) Size of the receive buffer which is configured for the UDP socket. A zero value means the buffer size remains unchanged. 8192 bytes is a normal size in Windows. If there are more stations or more intensive communication, the buffer should be enhanced.	bytes	0
RO	Receive Only	If the value is True, no messages are sent to any station on the line. This parameter may be used when listening to the LonTalk communication: Configure the address, which is the same as the address of an existing LonTalk device, on the line. Also, configure the station with the device address which communication you need to listen to. The communication between devices is recorded in the log file of the line. RO=True ensures that the KOM process does not influence the communication by its commands and responses.	-	False
SC	Send Count	(for LonWorks lines only) The retry count of one packet - default value is 1. However, in some situations when using *iLON(TM) 10 Ethernet Adapter*, the first message did not pass and the communication started to work correctly when SC=2. **Note:** Later we found out that this was caused because the Free topology bus had not been ended by a terminator. However, this parameter had been already implemented.	-	1
SD	Send Delay	(for LonWorks lines only) A complement to the *Send Count* parameter that defines a delay (in ms) after each sending of the packet.	ms	0
VI	Vendor ID	Parameter *Vendor ID* of *I-Am* message (see the parameter *Device Instance*).	-	1

Line protocol parameters specific for BACnet MS/TP

Keyword	Full name	Meaning	Unit	Default value
BR	MS/TP baud rate	Baud rate of the line. This parameter helps to recalculate some timeouts that are defined in a **bit time** in the communication line protocol. The bit time is a multiple of the period which is required for transfer of 1 bit at the particular baud rate.	bits /sec	9600
MIF	MS/TP $N_{max\_info\_frames}$	Maximum of information frames that may be sent by the KOM process before it must send a token. The standard does not specify a particular value. It recommends that the value must be 1 if this value is not configurable in a device. The higher value is set, the less time remains for other Masters. But on the contrary, it reduces the number of frames without information.	-	5
MO	MS/TP N_{min_octets}	A minimum number of data (bytes) received on the line to be received by the KOM process before it indicates the line as "active".	-	4
MY	MS/TP my address	Address of the KOM process on the line RS-485. The valid value is from the interval 0 - 127. It must be different from the addresses of other devices on the line (their addresses are defined in the station configuration).	-	1
TFA	T_{frame_abort}	A minimum time (the unit is the length of bit transmission, i.e. it depends on MS/TP baud rate), after the expiration of which, the whole frame is discarded if no character was received. According to standard, the value may be higher but it cannot exceed 100 ms in absolute time.	bits	60
TNT	T_{no_token}	Time (in milliseconds). After it expires, without receiving any data, the token will be considered lost.	ms	500
TR	$T_{reply\_timeout}$	The minimum time (specified in ms) that the KOM process must wait for the station to respond to the request.	ms	255
TS	T_{slot}	Time (specified in ms) during which the station can generate a token.	ms	10
TU	$T_{usage\_timeout}$	A minimum time (specified in ms) for which the KOM process must wait while a partner starts to use a token or responds to a *Poll for master* frame. The standard value is 20 ms. According to the standard, the value may be higher - a maximum of 100 ms.	ms	20

Communication station configuration

The communication station corresponds to a device on the BACnet network with which the KOM process communicates.

- **Station type:** BACnet/IP station must be configured on TCP/IP-UDP line. LonWorks station must be configured on the LonWorks line. MS/TP station must be configured on SerialOverUDP Device Redundant or Serial line.
- **Address:**
  - BACnet/IP station: IP address of station (in the form A.B.C.D, e.g. 172.16.0.99)
  - LonWorks station: address of LON subnet and LON node (in the form *subnet.node*, a subnet is an 8-bit number and a node is a 7-bit number)
  - MS/TP station: number of the node on the line (0-254, address 255 is a broadcast)
- **Port:** (only for BACnet/IP): UDP port number on station (according to standard 0xBAC0, i.e. 47808)
- **Domain:** (only for LonWorks): 0 or 1, it is related to the line configuration. On the LonWorks line, a membership to one or two domains can be configured. On the BACnet station, the selection of domain means that the device belongs to this domain (it influences 'domain' bit in LON address).
- **Source network:** source network number (i.e. a network with KOM process). This parameter may not be set for the LonWorks line. For TCP/IP-UDP line, it is a 16-bit number (or it is not set, see Note 2).
- **Destination networkEdit:** a 16-bit number of a destination network (i.e. a network including the device which communicates with the KOM process).
  Set this for the LonWorks line if the KOM process communicates with the device that is located behind a BACnet router. In that case, the Address of the station is the address of the BACnet router and the Destination address is the address of the destination device.
  For TCP/IP-UDP line, the **Destination network** is used in a similar way if there is communication between different BACnet networks.

  Note 1: This configuration was tested as follows:
  - Line: *TCP/IP-UDP*
  - Station type: *BACnet/IP*
  - Address: *172.16.99.1* (address of a BACnet router PXG80-N)
  - Destination network: *1*
  - Destination address: *1.1* (address of PXC22 on a LON network behind a BACnet router)

  The KOM process communicated with the PXC22 device which was connected to a LON network via PXG80-N BACnet router. The KOM process communicated with a BACnet router over Ethernet, so the line is TCP/IP-UDP. The communication between the BACnet router and the PXC22 station was performed over a LON network.

  Note 2: We tested a similar configuration. We used Delta Controls DSM-RTR (connected over Ethernet network) and a Klimasoft MBG device (a gateway to M-Bus) connected to Delta Controls via an MS/TP interface. The communication started only if the *Destination network* (value 50020) and *Destination address* (value 96) were configured and the *Source network* was not specified. However, in another configuration, the communication worked also with the *Source network* parameter specified. We recommend you try various settings of network parameters for the devices.

- **Destination address:** It is the address of the destination device if KOM communicates with it over the BACnet router. When setting this parameter, you can (but you do not have to, see note about E-DDC3.1) set also the parameter Destination network. The **Destination address** parameter should be in the *subnet.node* format (if the destination device is on a LON network) or in the A.B.C.D format (if the destination device is on a BACnet/IP network).

  Note 1: On a BACnet/IP station you can configure the **Destination address** in the subnet.node format (e.g. 1.31). This configuration corresponds to the BACnet router, which communicates with the KOM process over BACnet/IP and is connected to the destination device via a LONTalk network.

  Note 2: On BACnet/IP station you can configure the **Destination address** as a number from the interval 1-255. This configuration corresponds to the BACnet router, which communicates with the KOM process over BACnet/IP and is connected to the destination device by MS/TP bus (DAC-633).

  Note 3: On BACnet/IP station you can configure the **Destination address** as a bigger number (e.g. 2001), which works for E-DDC3.1.

- **Resubscribe interval:** Time in seconds. After it elapses, a station again gets a request to send changes of I/O tags. This parameter relates to the I/O tags with the Request type that is equal to SubscribeCOV or SubscribeCOVProperty.
- **Max APDU:** Maximum size of the message (APDU = Application Protocol Data Unit) that is sent by the KOM process. The default value is:
  - 1467 octets for TCP/IP-UDP line
  - 487 octets for SerialOverUDP Device Redundant or Serial lines (BACnet MS/TP)
  - 55 octets for LonWorks line

    The limitation depends on the size of packets that may be transmitted over Ethernet and LonWorks. For LonWorks, the maximum value is 206. The value 55 is due to the limitation of the iLON 10 Ethernet adapter.

  The changing of the default value is important for testing and complying with the stations which are able to process only smaller messages. Currently, the reduction of *Max APDU* influences only the size and amount of ReadPropertyMultiple-Request messages. These messages are intended for a periodic reading of the I/O tag value (see I/O tag configuration).

  Note: The setting of Max APDU does not affect the size of max-APDU-length-accepted in APDU BACnet-Confirmed-Request-PDU, by means of which the KOM process informs a partner how big messages it is capable to process. This parameter is configured by the station protocol parameter Segment-Response.

- **Priority:** A priority of a message in the BACnet protocol. There are 4 priorities: Normal (default), Urgent, CriticalEquipment, and LifeSafety.
- **Rpt_timer & reply:** (only for LonWorks) The parameters Repeat timer (default value = 1) and Retry (default value =1) of LonTalk protocol.
- **Tx_timer:** (only for LonWorks) Parameter Tx_timer in LonTalk protocol. Default value = 3.
- **Timeout and retry:** A timeout in milliseconds to confirm the message. The default value according to the BACnet protocol is 3000 ms. After the timeout elapses, the message is sent **retry**-times. If any confirmation is not received, an error count will increase for the station.

  Note: When testing the Siemens PXC64-U device (the communication over LonTalk), we had to set Retry=8, Timeout=300 (more retries with shorter timeout). Due to that, we had to increase the values COM_ERR=10, HARD_ERR=20 so that the station did not switch to an error state when retrying to send the message.
- **COM_ERR:** The value of the error counter for the station when the station switches to COM_ERR status. The situation when the station does not reply to a read/write request is considered as an error. A negative confirmation of a command (refusal of writing) is not an error. The default value is 5. See the parameters *Timeout and retry.*
- **HARD_ERR:** The value of the error counter for the station when the station switches to HARD_ERR status. The default value is 10. See the parameters *Timeout and retry.*
- **Register-Foreign-Device, R-F-D Time to live:** In this example, let's have a station located on a LONTalk network behind a BACnet router that communicates with the KOM process over Ethernet (e.g. Desigo PXG80-N). The BACnet router sends the broadcasts from LONTalk to Ethernet as UDP broadcasts. If the distribution of UDP broadcasts is disabled or the KOM process is placed in a different segment of the network than the BACnet router (so it does not receive any UDP broadcasts), you should check the option *Register-Foreign-Device* on the station. This will cause the KOM process to send the *Register-Foreign-Device* message to a BVLC router (BACnet Virtual Link Control) after the start. The message requests registration to the FDT table (Foreign Device Table) in the router. The router sends the broadcasts in the form of UDP unicast (whose distribution is not limited to one segment) to the devices that are registered in the FDT table. The TTL (time to live) - time in seconds (1-65535) is the parameter of the Register-Foreign-Device message . It defines the expiration of registration that stops the sending of UDP unicasts. That is why the KOM process must ask the BACnet router to re-register it before TTL expires. If there are more stations behind a BACnet router, just check *Register-Foreign-Device* on one of them.

  Note 1: If the router does not support BBMD functionality (BACnet/IP Broadcast Management Device), it replies to the Register-Foreign-Device message with an error code and it does not send LonTalk broadcasts to the KOM process in the form of UDP unicasts. In that case, you must use other solutions (the communication over iLon Ethernet Adapter, the placing of the KOM process on the same segment in the network on which the BACnet router is, etc.).
  Note 2: Router Desigo PXG80-N supports this functionality (tested). The control station Desigo PXC22-E.D does not support it probably (not tested yet).
  Note 3: In the case of Desigo devices, if the D2000 KOM process is on a different network segment than the Desigo device, this parameter must be checked at the station. Otherwise, Who-Is and Who-Has requests won't work (and thus addressing by object's name), as responses to these requests are sent as UDP broadcasts which will not go through a router.

- **Master:** (only for MS/TP): The station is of Master type. The KOM process transmits a token to the Master station which has the next larger address than is the address of the KOM process (the MS/TP address line parameter). If the addresses of all Master stations are lower than the address of the KOM process, the token is given to the Master station with the lowest address. If no Master station has been configured, the KOM process supposes to be the only master and does not pass the token. You should get the information about the type of station from a producer or device documentation.

  Note: The current implementation of the BACnet protocol does not contain the automatic search for the Master station. You can find more information in the section Comment on BACnet MS/TP implementation.

Example of station configuration on TCP/IP-UDP line:

- **Station type:** BACnet/IP
- **Adresa:** 10.0.0.1
- **Port:** 47808
- **Source network:** 1

Example of station configuration on LonWorks line:

- **Station type:** LonWorks
- **Address:** 1.15
- **Domain:** 0

Station protocol parameters

Keyword	Full name	Meaning	Unit	Default value
RSD	Receive-send Delay	Delay between the receiving of the reply from the station and sending the next packet.	ms	0
SR	Segment-Response	A byte that contains Max Segs and Max Resp parameters (see the specification of BACnet protocol). Only some values from the 0-127 are permitted, which are specified by the BACnet standard. The KOM process considers the value 128 as default: • LonWorks line: set the value to 0x70 (more than 64 segments are accepted, the maximum length of the message is 50 bytes) • TCP/IP-UDP line: set the value to 0x75 (more than 64 segments are accepted, the maximum length of the message is 1476 bytes) • Serial and SerialOverUDP Device Redundant line: set the value to 0x73 (more than 64 segments are accepted, the maximum length of the message is 480 bytes)	-	128

TSU	Time-Synchronization UTC	The parameter is important only if the synchronization is enabled on the "Time parameters" tab in the configuration of the station. If the parameter is True (default), the time synchronization is performed by the *UTCTimeSynchronization-Request* message (the synchronization in UTC time). If the parameter is False, the time synchronization is performed by the *TimeSynchronization-Request* message (the synchronization in the local time). **Notes**: • We recommend you to use the synchronization in UTC if it is supported by the device - you can avoid the problems with "jumping" time during the transition from/to DST time. • The requests for the time synchronization are unacknowledged messages, i.e. the device will not send the answer whether it supports the time synchronization or not. • The time synchronization has been tested on Siemens PXC36-E.D (HW=V3.02). This device supports the synchronization in both UTC and local time. You can find out the current time and date as local-date(56) property and local-time(57) property of the object of Device(8) type. From this object, you can find out also utc-offset(119) property which defines the offset of local time from the UTC (in minutes, i.e. -60 is Central European Time) as well as daylight-savings-status(24) property, which defines whether the device works in the summer-time (when testing in September 2012, the value on the device was True). After the time synchronization, the values of local-date(56) and local-time(57) have been changed.	-	True

I/O tag configuration

Type of I/O tags: **Ai,Ci,Di,TiA,TiR,TxtI,Ao,Co,Dout,ToA,ToR,TxtO**.

I/O tag corresponds to object property.

- **Request type:** Reading and writing of the object properties may be done in several ways:
  - **ReadProperty -** a periodical reading of object property as request-response. A polling period is configured on the *Time parameters* tab of station. The *ReadProperty-Request* message is used for the request and the *ReadProperty-Ack* message is the response from the device. The periodic reading burdens the network and is ineffective. That is why, if the device supports the sending of change data, we recommend you to use SubscribeCOV or SubscribeCOVProperty requests.
    The *ReadProperty-Request* message is sent if the Subscribe/read checkbox is checked.
  - **ReadPropertyMultiple** - the functionality is similar to the previous parameter. Unlike ReadProperty, more object properties are sent in one request/response, so communication is much more effective. The *ReadPropertyMultiple-Request* message is used for the request and the *ReadPropertyMultiple-Ack* message is the response from the device.
    The *ReadPropertyMultiple-Request* message is sent if the Subscribe/read checkbox is checked.
  - **WriteProperty -** the *WriteProperty-Request* message is used for writing the values. The Application tag parameter must be specified as well. If Subscribe/read is checked, the written value is verified by reading from the station using the *ReadProperty-Request* message after writing.
  - **SubscribeCOV** - activation of reading of object value when they change. If the Subscribe/read checkbox is checked, after starting, the KOM process sends the *SubscribeCOV-Request* message which asks the device to send information about the change of object value. You can specify whether the device will send the confirmed notifications (*ConfirmedCOVNotification-Request* message) or the unconfirmed ones (*UnconfirmedCOVNotification-Request*). The confirmed notification requires an explicit confirmation from the KOM (the *BACnet-SimpleACK-PDU* message), so puts more load on the network. However, the probability that the notification will be lost is lower than in case of using unconfirmed notifications (if the device does not receive a confirmation, it repeats the message).

    Note 1: Besides the dynamic registration by the *SubscribeCOV-Request* message, some devices can support also a static one (it is saved in the configuration). So the registration is not required and the checkbox Subscribe/read may be left unchecked.
    Note 2: The registration can be sent at regular intervals (e.g. because of a potential failure of the device power supply). You can set this interval on the station - the Resubscribe interval parameter.
    Note 3: *SubscribeCOV-Request* messages (and also *SubscribeCOVProperty-Request* messages, see the following point) are also sent after the connection with the station is re-established (after a failure or after switching from the StOFF state to the StOn).
  - **SubscribeCOVProperty** - the functionality is similar to *SubscribeCOV*. Moreover, you can specify the *Property identifier* (so you can monitor also the changes of other object properties than the present value) and *Increment* - a size of increment which causes the change to be sent (i.e. a dead band).
    The *SubscribeCOVProperty-Request* message is sent.

    Note: The Siemens PXC64-U device did not support the *Increment* parameter.
  - **WhoIs** - the identification message *Who-Is-Request* to detect the type of Device Object in a device. The *I-Am-Request* message is the response (it contains the iAmDeviceIdentifier, maxAPDULengthAccepted, segmentationSupported, and vendorID fields). If the I/O tag is TxtI, this information is extracted to the value of the I/O tag in a text form. After you identify the Device Object, you can configure the I/O tag for reading the property *object-list* of this Device Object. If the device implements this property, it returns the list of identifiers of all objects which it contains. Then you can query the properties of these objects (object-name, location, description, present-value ..)

    Note: For Siemens PXC64-U you must set the Array index and then read the property *object-list*. Array index=0 defines the number of array elements, Array index=1 up to N enables the access to the individual elements.
  - **WhoHas** - the identification message *Who-Has-Request* to detect the object name from the object identifier or vice versa. The response is the *I-Have-Request* message (it contains the fields deviceIdentifier, objectIdentifier, and objectName). The *Who-Has-Request* message is sent only once when initialization of I/O tag (or after the TELL command SETPTADDR). It is intended for the transfer between names and identifiers of objects.
    The message Who-Has-Request will contain either name or identifier of the object depending on whether the Address type has been configured as *Name* or *Object type+Instance* in I/O tag.
    If Subscribe/read is checked, you can use the information from the BACnet cache, which is much faster than detection from the communication.
  - **ReadWriteScheduler** - the *ReadProperty-Request* message is used for the request, the *WriteProperty-Request* message is used for write (it writes N *time-value* pairs). The configuration is used for the reading and writing of objects of *schedule* type - the *weekly-schedule(123)* attribute, see the Scheduler in Siemens Desigo devices paragraph.
  - **GetEventInformation**: detection of objects that are in alarm or error state or they need to be acknowledged, see the section Information about events.

        - **AcknowledgeAlarm**: the acknowledgment of alarms that have been read by the GetEventInformation request. See the section Information about alarms. I/O tag must be the text output (TxtO).
- **Address type:** Each object in the BACnet protocol is addressed by an Object identifier. When designing the application in the Desigo system, the objects are represented by a name, but the object address is not accessible and can vary following the changes of application. On the other hand, the Delta Controls devices contain the objects whose addresses are defined by the author of the application. For this reason, there are two ways how to define the address of I/O tag which corresponds to two *Address type*:
    - **Name:** enter the object name. A type of object and instance number is queried dynamically from the communication. To avoid the overloading of communication lines when starting the KOM process, data is stored in a BACnet cache.
    - **Object type + Instance:** enter the type of object and an instance number. This is recommended for BACnet objects with constant addresses.
- **Object type:** type of objects, whose properties will be read/written. You can use the predefined types or write the number of a new type of object which has been defined by a producer. The type of object is a 10-bit number.
- **Instance:** an object identification number within the object type. Each object has a unique Object Identifier in the device, which is a pair [Object type, Instance].
- **Object Name:** name of the object, when Address type = Name, i.e. it means the I/O tag address [Object type, Instance], is detected dynamically from the communication. *Object Name* must be specified exactly, i.e. spaces in the beginning and at the end are not tolerated, and the upper and lower case letters must correspond to the object name that is stored in the BACnet device.
- **Property type:** type of property - only *PropertyIdentifier* is specified for *Simple*, and both *PropertyIdentifier* and *Complex address* must be specified for *Complex*. The complex type of property is necessary for the parsing of OEM-extended standard messages (items of 'Abstract Syntax & Notation' type). When sending the *ReadProperty-Request, ReadPropertyMultiple-Request, SubscribeCOV-Request,* and *SubscribeCOVProperty-Request* messages, the *Complex address* is ignored.
- **Property identifier:** identifier of an object property. You can use the predefined properties or configure a numeric identifier of new property which was defined by the OEM manufacturer. The type of property identifier is Enumerated Value, the properties 0-511 are reserved for BACnet, the numbers from 512 to 4194303 can be used by the OEM manufacturer.
- **Array index:** some properties may be defined as arrays of values. In this case, a particular item of an array can be read or written.
- **Application tag:** it must be specified when writing the value (Request type=*WriteProperty*) and possibly for other types of requests if the parsed response contains context tags which application type is unknown because it is the extension of messages defined by the OEM manufacturer. The exception is an output tag of text type that is considered to be 'AnyTree', if the application tag is unspecified, and it can be used to write any user-specified ASN sequence.

  Note: If the value is *Invalid*, it is not written as the defined *Application tag*, but as *Application tag "Null"*.
- **Complex address:** address of a tag in a 'tree' in connection with the extension of messages that have been defined by the OEM manufacturer.
  Example of address: *[1].[3].2.1*
  Description:
  [1] - context tag No.1 (it is assumed that it is the sequence),
  [3] - it is the context tag No. 3 in this sequence (again, it must be a sequence),
  2 - it is the second tag in order in this sequence (again, it must be a sequence),
  1 - it is the first tag in order in this sequence.
  The address in 'tree' starts from the *propertyValue* level (see the examples below). The easiest way to view the parsed message is to turn on debugging for a line and watch the debug info on the console or in the line log file.

  Example 1: Let's have a device that contains the object of type 2 (*Analog Value*) with instance number 1. Let's assume that the device sends the object value as a triplet of numbers. The first number is the current value, the second one is a one-minute average and the third one is a ten-minute average. The log of the parsed message could be the following:

```
=== ASN Body beg ===
objectIdentifier (tag 0) OBJID 2 analog-value,1
listOfResults (tag 1) SEQUENCE {
 propertyIdentifier (tag 2) ENUM 85 present-value
 propertyValue (tag 4) SEQUENCE {
  REAL 1.40000E+00
  REAL 1.10000E+00
  REAL 1.30000E+00
 }
}
=== ASN Body end ===
```

  If you want all three values, you must configure three I/O tags (Object type=analog_value, Instance=1, Property-identifier=present-value, Property-type=complex), which differ in the complex address (for the first I/O tag specify 1, for the second one specify 2, and for the third one specify 3). Tick off the checkbox Subscribe/read in a configuration of one of these I/O tags only, because the response to one request is the message with three values. When sending the *ReadProperty-Request, ReadPropertyMultiple-Request, SubscribeCOV-Request, SubscribeCOVProperty-Request,* and *WriteProperty-Request* messages, the complex address is not used.

  Note: If you configure the I/O tag with Property-type=simple, its value would be set to the first found value after parsing the message (in the previous example it is 1.40000E+00).

  Example 2: Siemens Desigo PXC64-U contains I/O tag (Object type=schedule, Instance=6, Subscribe-read is checked, Property-identifier=weekly-schedule, Property-type=complex, Array index=1, Complex address=1). A debugging has been started on the line. After the I/O tag is saved, the KOM process sends the request and writes the response:

```
=== ASN Body beg ===
objectIdentifier (tag 0) OBJID 17 schedule,6
propertyIdentifier (tag 1) ENUM 123 weekly-schedule
propertyArrayIndex (tag 2) UNSIGNED 1
propertyValue (tag 3) SEQUENCE {
 SEQUENCE {
  TIME 0:0:0.0
  UNSIGNED 2
  TIME 4:0:0.0
  UNSIGNED 3
  TIME 22:0:0.0
  UNSIGNED 1
 }
}
=== ASN Body end ===
```

In *propertyValue*, there is the sequence of 6 values (time and positive number alternately). If you want to access the first time, you have to set Complex address=1.1, if to the first positive number, set Complex address=1.2. I.e. the first element - sequence - the second element in the order within it (UNSIGNED 2). If you need to access more times and/or values at the same time, you must configure several I/O tags and check the *Subscribe/read* checkbox in one of them only.

Note 1: If the I/O tag was created with a complex address 1, its value would remain Unknown, because this address matches the 1st element in *propertyValue*, which is a sequence, not a simple type.

Note 2: The I/O tag of *Text* type is able to contain not only simple value but also any ASN sequence. The values are written according to rules for the writing of ASN sequence. If you set Complex address=1 and change the I/O tag to text input or text output in the previous example, its value will be a string " T0:0:0.0; u2; T4:0:0.0; u3; T22:0:0.0; u1; ". If Property-type=complex, but Complex address is not defined, the value will be "0{ T0:0:0.0; u2; T4:0:0.0; u3; T22:0:0.0; u1; }".

- **Increment:** increment of value change in the object property which causes the reporting of a change (see SubscribeCOVProperty).
- **Confirmed:** the checkbox specifies whether the device should send the confirmed notifications (*ConfirmedCOVNotification-Request*) or unconfirmed one (*UnconfirmedCOVNotification-Request*) for the configured Request types SubscribeCOV and SubscribeCOVProperty.
- **Subscribe/read:** if the checkbox is ticked, the respective messages for reading/registration of value changes are sent for the configured Request types:
  ReadProperty: the *ReadProperty-Request* message
  ReadPropertyMultiple: the *ReadPropertyMultiple-Request* message
  SubscribeCOV: the *SubscribeCOV-Request* message
  SubscribeCOVProperty: the *SubscribeCOVProperty-Request* message
  ReadWriteScheduler: the *ReadProperty-Request* message

- **Period:** if a nonzero value is set and the Subscribe/read checkbox is ticked, the Subscribe/read messages will not be sent in the interval *Polling period*, which has been configured on the station, but in the period defined by this parameter (in seconds). In this way, you can configure various polling periods for different objects on one station. Moreover, you can detect whether the station communicates, using one I/O tag with Request type ReadProperty and a small value of the *Period* parameter.
- **Local time:** if the checkbox is ticked, the received/sent times and dates are considered to be in local time, otherwise they are considered to be monotonic UTC time.
- **Flags-to-flags:** if the checkbox is ticked, it causes the user flags FA, FB, FC, FD to be set beside the I/O tag value, for the SubscribeCOV and SubscribeCOVProperty Request types. The value of *status-flags* attribute (of BACnetStatusFlags type) is mapped to these flags, if it is sent. BACnetStatusFlags is a quaternion of bits (in-alarm, fault, overridden, out-of-service) that provides extra information about the object value.
- **Write priority:** for writing of 'commandable' properties, you can specify the priority 1-16. 1 = top priority, 16 = the lowest priority, 0 = none priority.

Browsing of address space

When clicking the "**Browse**" button in the **Address** tab of the I/O tag, the **BACnet Item Browser** dialog box opens. In this dialog box, the user may browse the address space of a device and insert its items to the address dialog of the I/O tag.

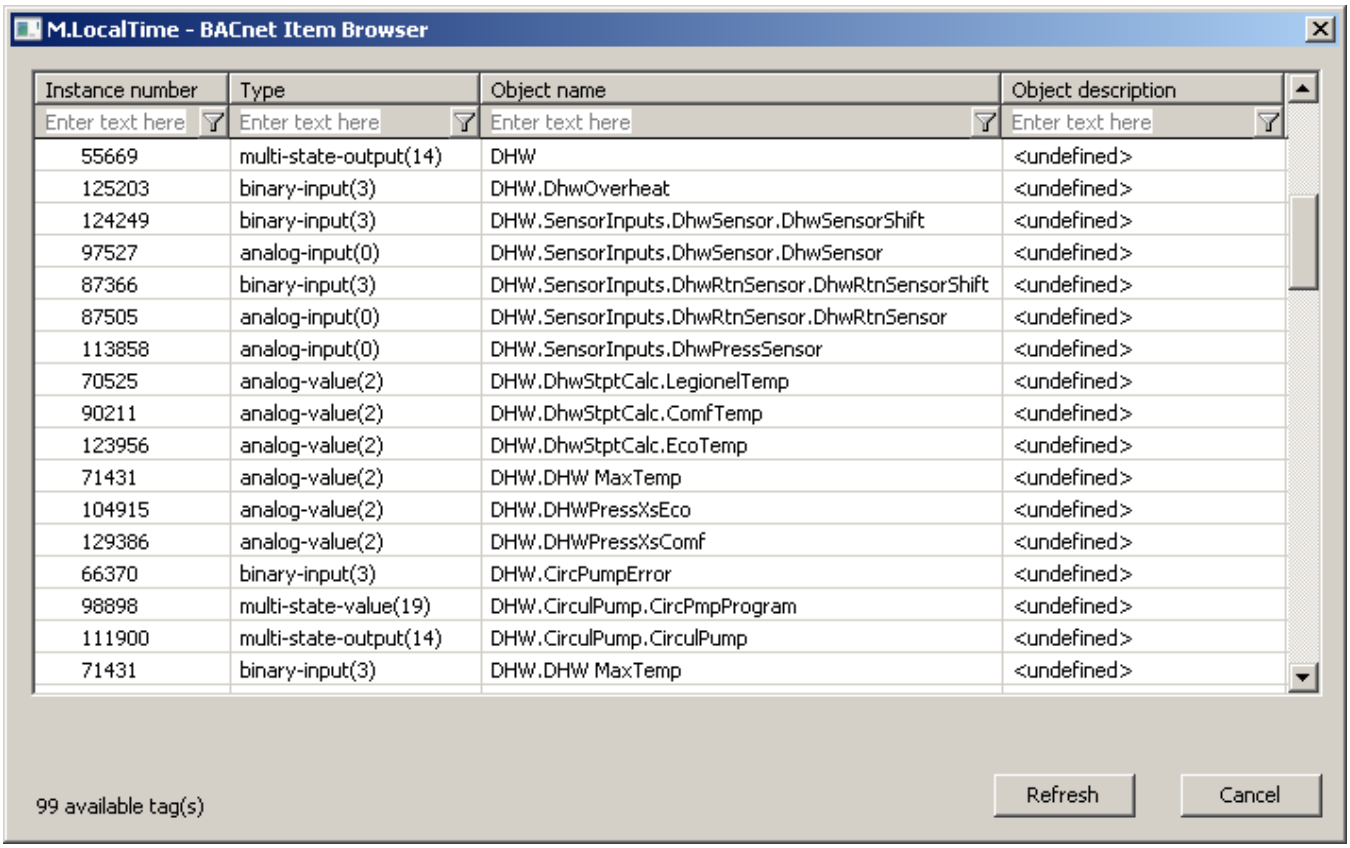
The items can be filtered based on four basic criteria:

- Instance number
- Type
- Object name
- Object description

Note: In the case of *Type*, *Object name,* and *Object description* it is not necessary to enter the full text in the filter field. The notation "*FILTERED EXPRESSION*" is supported. The symbol * represents any text before and after the expression.

Note: Using Ctrl+C it is possible to copy the content of the **BACnet Item Browser** into the Windows clipboard. All rows will be copied unless a specific row is selected.

Note: In versions from 20th December 2018 and newer, the recycling of browser dialog has been implemented. If the dialog is closed by the Cancel button or after selecting an object, it is actually only hidden and it is available for browsing by another I/O tag within the same station so that the tree structure, containing the browsed objects, is preserved. Clicking on the close icon at the top right corner will cause the dialog to be really closed.



Writing of any ASN sequence
Any ASN sequence can be written via the I/O tag of text output type (TxtO) which has the *Application tag* property undefined. The rules are the following:

- the element consists of the optional number of context tag, the letter defining the application tag, and the value
- the application tags are written as follows (usage with and without context tag):
  - Null: *[tag] n*, example: " n", " 3n",
  - Boolean: *[tag] b [0|1|n|y|N|Y]*, example: "b0", " 3b1"
  - Unsigned: *[tag] u value*, example: "u 123", " 10 u123"
  - Signed: *[tag] s value*, example: "s-123", " 10s 5"
  - Real: *[tag] r value*, example: "r 1.23", " 10r-3.14"
  - Double: *[tag] d value,* example: "d 1.23", " 10 d -3.14"
  - Octet string: *[tag] O string*, every byte of string is written as hex number (byte 1 is 01, byte 26 is 1A), example: "O 1A33f0", " 10 Obb004E"
  - Character string *[tag] C 'string'*, example: "C 'hello world' ", " 10C 'apostrophe " in the string' "
    The string must enclosed in the apostrophes. If the apostrophe occurs inside the string, it must be double (see the second example). Empty string can be set as follows: " C; "
  - Bit string: *[tag] B bits*, example: "B 100101", " 23B00101"
  - Enumerated value: *[tag] E value*, example: "E 123", " 10 E123"
  - Date: *[tag] D day.month.year[.day_of_week]*, example: "D 1.10.2005", " D3.4.2004.5" (Monday=1 .. Sunday=7)
  - Time: *[tag] T hour:minute:sec[.ms]*, example: "T 5:12:33.133", " T10:00:00"
  - Object identifier: *[tag] o type:instance* or *[tag] o objid*, (*type* is a 10-bit number, *instance* is a 22-bit number, *objid* is a 32-bit number. The examples show a 2-component and a single-component format of this application tag with type=3 (binary-input) and instance=2
    " o 3:2", " 3o3:2"," 3o 12582914"
- the sequence consists of individual elements separated by the blank spaces and/or semicolons, e.g. " 1b0 2u13 ; 3 B 1001;4E14"
- the sequence can contain nested sequences
- the nested sequence begins with an optional context tag number and an opening bracket character '{'. If no context tag is specified, 0 is used.
- at the end of a nested sequence there is a closing bracket character '}'
- an example of nested sequence: "1u2 2{ 1s-1; 2E0 }", two levels of nesting: " { 1{ u23 s34 } 2E56 3r7.89 }"

Note 1: If a result of the reading of I/O tag from communication is an ASN sequence and the I/O tag is of Txt type, this ASN sequence is written into it (according to the above-mentioned rules) using the following rules:

- a blank space is before each element and the element is followed by a semicolon (e.g. " 1E4; 2B111; 3u1;"),
- a blank space is not between the number of context tag and the letter specifying the application tag,
- a blank space is not between the letter specifying the application tag and the value,
- a context tag is before every nested sequence (e.g. " 0{ 0o1:2; 1E4; }",
- format of the time is hh:mi:ss.mss, e.g. " T11:01:02.000; 1T12:00:00.000; ",
- format of the date is dd.mm.yyyy, e.g. " D25.01.2005; 3D01.01.2005;".

Note 2: The empty sequence may be written by a string " ", the string of length 0 (i.e. "") is ignored.

Examples of I/O tag configuration (Siemens Desigo PXC64-U device):
Analog input:

- Request type: SubscribeCOV
- Object type: analog-input(0)
- Instance: 1
- Confirmed: Y
- Subscribe/read: Y
- Property type: Simple
- Property identifier: present-value(85)

Binary input:

- Request type: SubscribeCOV
- Object type: binary-input(3)
- Instance: 1
- Confirmed: Y
- Subscribe/read: Y
- Property type: Simple
- Property identifier: present-value(85)

Binary value:

- Request type: ReadProperty
- Object type: binary-value(5)
- Instance: 8
- Subscribe/read: Y
- Property type: Simple
- Property identifier: present-value(85)
- Application tag: Enum

Note: If the *Application tag* is not configured properly (for the binary value it is Enum), the Desigo station returns an error 'invalid-data-type' when attempting to write:

```
error-class ENUM 2 property
error-code ENUM 9 invalid-data-type
```

The Siemens Desigo PXC64-U device requires the following settings of the *Application tag*:

- binary-value, binary-output: Enum
- multi-state-value, multi-state-output: Unsigned
- analog-value, analog-output: Real

Comment on input-output I/O tag: you can configure the I/O tag which is both input and output:
I/O tag of Ao type - *Analog output*:

- Request type: ReadProperty
- Object type: analog-value(2)
- Instance: 45
- Subscribe/read: Y
- Property type: Simple
- Property identifier: present-value(85)
- Application tag: Real (this is necessary for writing)

When writing the value, the *WriteProperty-Request* message is used. As *Subscribe/read* is checked, the value is read after it's written. If the I/O tag would be configured with Request type WriteProperty, its behavior would differ only by an absence of periodic value reading (the period is configured on the station, in the *Time parameters* tab*)*.

Comment on Siemens Desigo devices: I/O tags has a text name in the Desigo control system. The instance of the I/O tag may be found out from the DOTS00.DAT file in the application configuration - it is located 24 bytes before the beginning of the name.

Scheduler in Siemens Desigo devices

D2000 supports the reading and writing of schedulers. A scheduler contains BACnet attributes *weekly-schedule(123)* and *exception-schedule(38)*.
Weekly-schedule is an array of 7 items (one item for each day of the week). Each item represents a sequence of time-value pairs that define the value changes of the scheduler in a given time. When reading and writing, you can configure also an *Array index* and access individual items of a *weekly-schedule(123)* array. The array index is 1-7 for individual days (1=Monday), the index 0 contains a size of the array (value 7 for the *weekly-schedule(123)* at tribute, value 0 up to N for the *exception-schedule(38)* attribute).
Exception-schedule is intended for the holidays that require a different mode as is configured for common days. Exception-schedule is the sequence of 0 up to N items. An item always contains a date (or range of dates), several time-value pairs (as in a *weekly-schedule*), and a priority (1= top, 16= the lowest). The priority defines which of the items will be used if they overlap.

Reading of scheduler (the weekly-schedule attribute)

- Value-by-value: a dynamic change of complex address (1.1, 1.2, 1.3, etc.) in a script enables us to read all values and times similarly as for other properties.
- All times and values for one day at the same time:

- Value type: a text input (reading of scheduler) or text output (read/write)
- Request type: ReadProperty (reading of scheduler), ReadWriteScheduler (read/write)
- Subscribe/read: Y
- Object type: schedule(17)
- Instance: an instance number (e.g. 6) that is found in a Desigo configuration or with the help of the *WhoIs* request.
- Property type: Complex
- Property identifier: weekly-schedule(123)
- Array index: 1 up to 7, it depends on the day being read
- Application tag: if it is not defined, Unsigned will be used (it is used only for writing)
- Complex address: 1 (address of a sequence)

The times and values separated by a semicolon are read into the text value (see Note).
When writing the value to a scheduler you should realize that the value can be sent with various application tags (Unsigned, Signed), however, the device expects a particular tag (the easiest way to find it, is by reading the value while line logging is active). The application tag of value is defined by the *Applicati on tag* item in the configuration. The valid value of the *Application tag* can be Boolean, Unsigned, Signed, Real, and Double. If any other type is set, the Unsigned value is automatically sent. A value type can be changed dynamically - if the first character of a text value is B, U, S, R or D, it stands for (B) oolean, (U)nsigned, (S)igned, (R)eal or (D)ouble.

Writing to the scheduler (the weekly-schedule attribute)

- You must configure Request type=ReadWriteScheduler and assign a sequence of time-value pairs separated by semicolon into I/O tag of text output type, e.g. "0:0:0; 1; 2:30:40.5; 2; 5:00:00;1".
- The text string with the length=0 is ignored so that the "empty sequence" will not be written to the scheduler after saving the D2000 configuration or when the KOM process is started. For this reason, if the time plan of the scheduler must be deleted for a particular day, the string of nonzero length (it contains neither time nor value: " ") must be written to the I/O tag.

Note: Another possibility to write weekly-schedule, besides the ReadWriteScheduler request, is writing of an ASN sequence, e.g. the sequence "{ T0:0:0 u1; T2:30:40.5 u2; T5:0:0 u1 }" corresponds to the value "0:0:0; 1; 2:30:40.5; 2; 5:00:00;1". The I/O tag configuration differs only in Request type=ReadProperty. You can also write the time schedule for the whole week, if Array index is not set and the value contains 7 sequences for individual days, e.g. "{ T0:10:0 u3 T1:3:0 u1; } {T2:0:0.0 u2 T5:30:10.0; u3; } { T6:0:0.0 u2 T7:0:0.0 u3} { T20:0:0.33; u1} { T21:0:0.0; u1} { T22:0:0.0; u2} { T0:0:0.0; u3; T1:2:0.0; u1; T2:0:0.0; u2; T5:30:10.0; u3}".

Writing to the scheduler (the exception-schedule attribute)

The writing to the exception-schedule is supported via the writing of an ASN sequence.
Example: I/O tag configuration:

I/O tag type: text output (TxtO)

Request type: WriteProperty

Subscribe/read: Y

Object type: schedule(17)

Instance: 6

Property type: exception-schedule

Application tag: not defined (write the whole ASN sequence)

Using the string "0{ 0D2.10.2005 } 2{ T1:0:0; u1; T12:0:0; u3 } 3u10" you can write the time schedule for the day October 2, 2005. The scheduler has the value 1 since 1:0:0, the value 3 since 12:00:00. The priority of the exception-schedule is 10. When reading a value after writing (Subscribe/Read was configured) with the activated debug on the line, you can see the following log:

```
=== ASN Body beg ===
objectIdentifier (tag 0) OBJID 17 schedule,6
propertyIdentifier (tag 1) ENUM 38 exception-schedule
propertyValue (tag 3) SEQUENCE 3 {
 readResult (tag 0) SEQUENCE 0 {
  date (tag 0) DATE 2.10.2005 NoDay
 }
 listOfTimeValues (tag 2) SEQUENCE 2 {
  time TIME 1:0:0.0
  value UNSIGNED 1
  time TIME 12:0:0.0
  value UNSIGNED 3
 }
 eventPriority (tag 3) UNSIGNED 10
}
=== ASN Body end ===
```

To set the exception-schedule for the range of dates, write the value "0{ 1{ D5.10.2005; D8.10.2005}} 2{ T1:0:0; u1; T7:0:0; u3; } 3u15". This value for the range of dates 5.10.2005-8.10.2005 sets the scheduler from 1:00:00 to the value 1 and from 7:00:00 to the value 3. The priority of the exception-schedule is 15. When reading a value after writing (Subscribe/Read was configured) with the activated debug on the line, you can see the following log:

```
=== ASN Body beg ===
objectIdentifier (tag 0) OBJID 17 schedule,6
propertyIdentifier (tag 1) ENUM 38 exception-schedule
propertyValue (tag 3) SEQUENCE 3 {
 readResult (tag 0) SEQUENCE 0 {
  dateRange (tag 1) SEQUENCE 1 {
   startDate DATE 5.10.2005 NoDay
   endDate DATE 8.10.2005 NoDay
  }
 }
 listOfTimeValues (tag 2) SEQUENCE 2 {
  time TIME 1:0:0.0
  value UNSIGNED 1
  time TIME 7:0:0.0
  value UNSIGNED 3
 }
 eventPriority (tag 3) UNSIGNED 15
}
=== ASN Body end ===
```

Several items of above mentioned types can be written to the exception-schedule using a string which contains joined sequences, e.g. "0{ 0D2.10.2005 } 2 { T1:0:0; u1; T12:0:0; u3 } 3u10 0{ 0D3.10.2005 } 2{ T0:0:0; u2; T5:0:0; u3; T14:0:0; u1 } 3u11 0{ 1{ D5.10.2005; D8.10.2005}} 2{ T1:0:0; u1; T7:0:0; u3; } 3u15 "

Scheduler in Delta Controls devices

We also tested the reading and writing to the scheduler in the Delta Controls DAC-1146 device. Only the BACnet attribute *weekly-schedule(123*) was tested. The weekly-schedule is an array with 7 items (one item for each day in the week). Each item is the sequence of time-value pairs that define the value changes of the scheduler at a specific time. In contrast to Siemens Desigo devices, the schedulers are implemented with Boolean values True/False that are externally presented as Enum values 0/1. This is the example of scheduler Delta Controls:
"{ T0:10:0 E0 T1:3:0 E1; } {T2:0:0.0 E1 T5:30:10.0; E0; } { T6:0:0.0 E0 T7:0:0.0 E1} { T20:0:0.33; E0} { T21:0:0.0; E1} { T22:0:0.0; E0} 0{ T0:0:0.0; E0; T1:2:0.0; E1; }"
The writing as well as subsequent reading of E2 value can be performed, however, we don't know the DAC-1146's interpretation of E2 :-) .

Information about events

The GetEventInformation-Request is intended to ask the list of objects that are in the *Offnormal* or *Fault* states or whose change to *Offnormal*, *Fault,* or *Normal* has not been acknowledged.
Example of I/O tag configuration:

- I/O tag type: text input (TxtI)
- Request type: GetEventInformation
- Subscribe/read: Y
- Object type: undefined
- Instance: undefined
- Property type: complex
- Application tag: undefined

A reply to *GetEventInformation-Request* is the *GetEventInformation-Ack* message, which contains the list of objects and the list of properties for each object:

- objectIdentifier: identifier of object
- event-state: object status (0=normal, 1=fault, 2=offnormal, 3=high-limit, 4=low-limit, 5=life-safety-alarm)
- acknowledgedTransitions: 3 bits that define whether the last transition to the *offnormal*, *fault,* or *normal* status was acknowledged
- eventTimeStamps: the timestamps of the last transition to the *offnormal*, *fault,* and *normal* status
- notifyType: defines whether it is a notification of alarm (0) or event (1)
- eventEnable: 3 bits that define whether the events report to *to-offnormal*, *to-fault*, *to-normal*
- eventPriorities: 3 unsigned values defining the event priorities

At the end of the list, there is the *moreEvents* attribute that is non zero if the list of events is incomplete (e.g. the exceeding of the maximum length of the message, etc.). In this case, you must reconfigure the I/O tag and set Object type and Instance to the last object in the list. It causes the generation of a new *GetEventInformation-Ack* message.

Example of *GetEventInformation-Ack* response:

```
=== ASN Body beg ===
listOfEventSummaries (tag 0) SEQUENCE 0 {
 objectIdentifier (tag 0) OBJID 0 analog-input,1
 event-state (tag 1) ENUM 3 high-limit
 acknowledgedTransitions (tag 2) BITSTRING 1,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventTimeStamps (tag 3) SEQUENCE 3 {
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 9.11.2005 Wed
   time TIME 13:28:49.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 9.11.2005 Wed
   time TIME 13:25:59.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 9.11.2005 Wed
   time TIME 13:25:56.0
  }
 }
 notifyType (tag 4) ENUM 0 alarm
 eventEnable (tag 5) BITSTRING 1,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventPriorities (tag 6) SEQUENCE 6 {
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 7
 }
 objectIdentifier (tag 0) OBJID 214,1
 event-state (tag 1) ENUM 2 offnormal
 acknowledgedTransitions (tag 2) BITSTRING 0,1,0 to-offnormal(0),to-fault(1),to-normal(2)
 eventTimeStamps (tag 3) SEQUENCE 3 {
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 18.11.2005 Fri
   time TIME 12:52:11.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 18.11.2005 Fri
   time TIME 11:16:23.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 9.11.2005 Wed
   time TIME 12:23:58.0
  }
 }
 notifyType (tag 4) ENUM 0 alarm
 eventEnable (tag 5) BITSTRING 0,0,0 to-offnormal(0),to-fault(1),to-normal(2)
 eventPriorities (tag 6) SEQUENCE 6 {
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 7
 }
}
moreEvents (tag 1) BOOLEAN FALSE
=== ASN Body end ===
```

Information about alarms

You can acknowledge the events and alarms, whose list has been received by the GetEventInformation request. According to the BACnet protocol, the format of this request is (the record in ASN.1 syntax):

```
  ********************** Confirmed Alarm and Event Services ******************
  AcknowledgeAlarm-Request ::= SEQUENCE {
    acknowledgingProcessIdentifier [0] Unsigned32,
    eventObjectIdentifier [1] BACnetObjectIdentifier,
    eventStateAcknowledged [2] BACnetEventState,
    timeStamp [3] BACnetTimeStamp,
    acknowledgmentSource [4] CharacterString,
    timeOfAcknowledgment [5] BACnetTimeStamp
  }
```

For a more detailed description of the parameters, see the literature.
The acknowledgment is executed by the writing of the above-mentioned sequence to the text of the output I/O tag according to the rules of writing of any ASN sequence.

Example: The following list of alarms and events was read using the GetEventInformation request:

```
===  ASN Body beg ===
listOfEventSummaries (tag 0) SEQUENCE 0 {
 objectIdentifier (tag 0) OBJID 0 analog-input,1
 event-state (tag 1) ENUM 4 low-limit
 acknowledgedTransitions (tag 2) BITSTRING 0,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventTimeStamps (tag 3) SEQUENCE 3 {
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 11:54:23.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 10:4:37.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 11:54:23.0
  }
 }
 notifyType (tag 4) ENUM 0 alarm
 eventEnable (tag 5) BITSTRING 1,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventPriorities (tag 6) SEQUENCE 6 {
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 7
 }
 objectIdentifier (tag 0) OBJID 0 analog-input,2
 event-state (tag 1) ENUM 3 high-limit
 acknowledgedTransitions (tag 2) BITSTRING 1,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventTimeStamps (tag 3) SEQUENCE 3 {
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 10:41:59.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 10:12:20.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 10:12:21.0
  }
 }
 notifyType (tag 4) ENUM 0 alarm
 eventEnable (tag 5) BITSTRING 1,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventPriorities (tag 6) SEQUENCE 6 {
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 7
 }
 objectIdentifier (tag 0) OBJID 214,1
 event-state (tag 1) ENUM 2 offnormal
 acknowledgedTransitions (tag 2) BITSTRING 0,1,1 to-offnormal(0),to-fault(1),to-normal(2)
 eventTimeStamps (tag 3) SEQUENCE 3 {
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 11:54:23.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 10:12:20.0
  }
  dateTime (tag 2) SEQUENCE 2 {
   date DATE 25.11.2005 Fri
   time TIME 10:12:21.0
  }
 }
 notifyType (tag 4) ENUM 0 alarm
 eventEnable (tag 5) BITSTRING 0,0,0 to-offnormal(0),to-fault(1),to-normal(2)
 eventPriorities (tag 6) SEQUENCE 6 {
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 3
  eventPriority UNSIGNED 7
 }
}
moreEvents (tag 1) BOOLEAN FALSE   === ASN Body end ===
```

The first object is the list
objectIdentifier (tag 0) OBJID 0 analog-input,1
which has the *low-limit* status
event-state (tag 1) ENUM 4 low-limit
and has unacknowledged transition to the *offnormal* status (i.e. according to D2000 terminology it is an active *low-limit* alarm which requires the acknowledgement):
acknowledgedTransitions (tag 2) BITSTRING 0,1,1 to-offnormal(0),to-fault(1),to-normal(2)
This transition occurred in time
dateTime (tag 2) SEQUENCE 2 {
date DATE 25.11.2005 Friri
time TIME 11:54:23.0
}
The alarm is acknowledged by writing a text value
0u1; 1o0:1 ; 2E2; 3{ 2{ D25.11.2005 T11:54:23} } 4C'D2000' 5{ 2{ D25.11.2005 T11:55:23 }}
and the individual items are as follows (see the definition of *AcknowledgeAlarm-Request* above):

- *0u1u1* - tag 0, unsigned value 1 - acknowledgingProcessIdentifier = identifier of process which acknowledges the alarm (it can be arbitrary)
- *1o0:1* - tag 1, the identifier of object of type 0, instance 1 - eventObjectIdentifier = the identifier of object which contains the alarm
- *2E2* - tag 2, enum value 2 - *eventStateAcknowledged* = the acknowledged value (BACnet standard defines the following events that can be acknowledged):
  normal (0),
  fault (1),
  offnormal (2),
  high-limit (3),
  low-limit (4),
  life-safety-alarm (5)
  in this case we acknowledge the transition to the *offnormal* status
- *3{ 2{ D25.11.2005 T11:54:23} }* - timeStamp = the sequence of date and time which must be acknowledged (it must match the received event date and time)
- *4C'D2000'* - acknowledgmentSource = a source of alarm acknowledgement (it seems it's an arbitrary string)
- *5{ 2{ D25.11.2005 T11:55:23 }}* - timeOfAcknowledgment = date and time when the alarm was acknowledged

After acknowledging of alarm and a repeated reading of the list of alarms and events by the GetEventInformation request , you will see the alarm has been acknowledged, as
*acknowledgedTransitions (tag 2) BITSTRING **0**,1,1 to-offnormal(0),to-fault(1),to-normal(2)*
will change to:
*acknowledgedTransitions (tag 2) BITSTRING **1**,1,1 to-offnormal(0),to-fault(1),to-normal(2)*

If the object had been in a *normal* state, it would not have been listed in the list of alarms at all. In this example it is in a *low-limit* state, i.e. it is an active acknowledged alarm.

To read the list of alarms, to show them in a browser, and to acknowledge the alarm, you must write the ESL scripts which will parse the text value of GetEventInformation request and compose a text value for AcknowledgeAlarm.

Note: in the specific case, when acknowledging the alarm, the DESIGO PXC 100E.D device required the day of the week to be specified within the date (eg *D8.6.2020.**1*** for Monday). A default value of 255 (unspecified) of a day of the week, which is sent when the day is not specified, did not suit the device when acknowledging the alarm.

Comment on address caching

If the station has at least one I/O tag with Address Type = Name, the numerical address is detected by *Who-Has-Request* from ObjectName when initializing these I/O tags. After getting the address, the data (the quadruplet ObjectName; Object Type; Instance; DeviceInstance) is saved to the cache file and is available for the next start of the KOM process.
There is one cache file for every station. It is located in an application directory, in a subdirectory Cache. Its name is Cache_*station_name*.txt, e.g. Cache_B.StationX1.txt.
When saving a BACnet station, the data is reloaded from this file. If the file was not found, the *Who-Has* messages are generated for all I/O tags that contain Address Type = Name in this station.
This behavior may be used after changing the configuration of the device which communicates with the KOM process (if the addresses of objects were changed when changing the configuration). Just delete the cache file in this station and save the station - within several seconds, the cache file will be created again and it will be filled with the acquired data.

Note: the interaction of cache and I/O tags with Request type = WhoHas.

- If the parameter Subscribe/read is checked in the configuration of the I/O tag, the cache is not searched and the *Who-Has-Request* message is sent to the communication. The obtained information is not written to the cache. It can be used for the objects that are generated and deleted dynamically, which would otherwise congest the cache.
- If Subscribe/read is not checked in the configuration of the I/O tag, the cache is searched. If *ObjectName* or *ObjectIdentifier* was not found, the *Who-Has-Request* message is sent to the communication. If the response comes, the information is written into the cache.

Comment on Delta Controls devices

The test configuration included a DSC-1212E device connected to the local Ethernet network and a DAC-633 device connected to DSC-1212E over MS/TP interface (RS-485). D2000 KOM communicated directly with DSC-1212E, and with a DAC-633 over DSC-1212E (which performed the function of a BACnet router).
We used the *ORCAview 3.30* Build 1481 software from which we obtained the following configuration information:

DSC-1212E
After logging on to the ORCAview and finding a *DSC-1212E* device, the object *BACnet Settings 3200* (3200 is a software address of the particular device) can be found in the right pane of the **Navigator** window. Clicking on *BACnet Settings 3200* opened a dialog window that contained several tabs. The first tab (*Setup*) contained also the UDP/IP port type. After clicking it, its parameters are displayed at the bottom of the window, among others *Network* (during the testing, we saw a specific number 40032), *IP Address,* and *UDP Port*.
When communicating with DSC-1212E, in the D2000 configuration of the station the *Source network* could be either set to the value of *Network* or left empty. The *IP Address* and *UDP Port* parameters must be used to configure the *Address* and *Port* parameters in the station configuration in D2000. A station type was *BACnet/IP*.

Note: On the *Advanced* tab, in the *BACnet Properties* area, the parameter *Local Network Number* was set to 10032 (it was the same as the *Network port* of the *Ethernet* type on the *Setup* tab). This port is used for BACnet over Ethernet communication (without the use of IP protocol), which is not supported by D2000 yet.

DAC-633
After logging on to the ORCAview and finding a *DAC-633* device, the object *BACnet Settings 3202* (3202 is a software address of the particular device) can be found in the right pane of the **Navigator** window. Clicking on *BACnet Settings 3202* opened a dialog window that contained several tabs. The tab (*Setup*) contained also the MS/TP port with the *Status* attribute and its value *Active*, and the attribute *Status Reference* with a value *BACnet Settings 3202 (NET1)*. After clicking it, its parameters displayed at the bottom of the window, among others *Network* and *MAC Address*.
When communicating with DAC-633, the following station parameters had to be configured in D2000:

- Station type: BACnet/IP
- Address: the parameter *IP Address* in the configuration of DSC-1212E
- Port: the parameter *UDP Port* in the configuration of DSC-1212E
- Source network: the parameter *Network* in the configuration of DSC-1212E
- Destination network: the parameter *Network* of the MS/TP port in the configuration of DAC-633
- Destination address: the parameter *MAC Address* of the MS/TP port in the configuration of DAC-633

As D2000 communicates with DAC-633 via DSC-1212E, the parameters *Address*, *Port,* and *Source network,* which correspond to DSC-1212E, and *Destination network* and *Destination address*, which correspond to MS/TP network between DSC-1212E and DAC-633, must be set in the station configuration.

Both devices support both polling and on-change methods of reading data (both *ReadProperty* and *SubscribeCOV*).
The *Object type* and *Instance* parameters in the configuration of the I/O tag were detected in ORCAview from the *Object* and *Description* attributes (e.g. object 3200.AI12 is Analog Input, Instance 12; the object 3202.PG3 is Program, Instance 3). The *Address type* parameter is *Object type+Instance* and *Request type* is *SubscribeCOV* unless declared otherwise:

Communicated I/O tag types:

- Analog-input
- Analog-output (Application tag: Real - can be written)
- Analog-value (Application tag: Real - can be written)
- Binary-input
- Binary-output (Application tag: Enum - can be written)
- Binary-value (Application tag: Enum - can be written)
- Calendar (Request type: ReadProperty, Property type: Complex, Property identifier: datelist(123), Application tag: Date, Complex address: 1,2,.. etc. - sequential reading of the array; possibility to write the whole calendar using the ASN sequence when the Complex address is not specified, e.g. "0D1.9.2006; 0D3.10.2006; 0D8.9.2006")
- Event-enrollment
- Schedule (Request type: ReadProperty or WriteProperty -can be written; Property identifier: weekly-schedule(123))
- Program (Request type: ReadProperty, Application tag: Boolean - can be written - stops and starts the program)
- Multi-state-value (Application tag: Unsigned - can be written)
- Trend-log (the cyclic buffer of values that are saved with the configured period)
    - Property identifier: 1074 - an array of trend timestamps in tens of milliseconds since the time of data reset
    - Property identifier: 1076 - bit string array (attributes of values?)
    - Property identifier: 1077 - an array of trend values
    - Property identifier: 1105 - the time of data reset

Comment on E-DDC3.1 devices

The communication worked with the following settings:

- Line: *TCP/IP-UDP*
- Station type: *BACnet/IP*
- Address: *10.0.6.23* (IP address of E-DDC3.1)
- Port: *47808*
- Source network: *not specified*
- Destination network: *not specified*
- Destination address: *2001* (this was acquired from manufacturer's BACnet OPC server - the "Device ID" parameter)

We tested ReadProperty, SubscribeCOV, WriteProperty (with the objects of Binary Output, Binary Value, Analog Value type).
Communicated I/O tag types (the test configuration contained only these types):

- Analog-input
- Analog-value (Application tag: Real - can be written)
- Binary-input
- Binary-output (Application tag: Enum - can be written)
- Binary-value (Application tag: Enum - can be written)

The device with the original firmware (2.01.05) did not properly handle the Who-Is request. We had to upgrade this firmware to the version 2.01.16. Then everything, including WriteProperty, worked properly.

Comment on Siemens Desigo devices

Based on the document "DESIGO INSIGHT: Installation, setup, and communication - Engineering guide", the recommended address settings for LonWorks communication are:

- DomainID: 0x49
- SubnetID: 1
- NodeID:
  - 1..100 - range reserved for automation stations (PX) and system devices (BACnet routers)
  - 101..120 - operating devices and DESIGO INSIGHT management stations
  - 121..127 - temporary operator devices (e.g. the PXM20 operator unit) and tools (DTS)

Comment on Klimasoft MBG-MSTP devices

The test configuration contained Moxa NPort 5150 (a converter from UDP to RS485) connected to the MBG-MSTP converter, which communicated with a heat meter Siemens UH50-A21R-SK06-G. The analog inputs were read, the MBG-MSTP converter supported ReadProperty only.
The communication worked with the following settings:

- Line: SerialOverUDP Device Redundant with the IP address and port of the Moxa NPort 5150
- The parameters of BACnet protocol which are configured on the line:
  MS/TP address: 6 (any address 1-254 that does not clash with other addresses on the RS485 bus)
  MS/TP N max_info_frames: 20. The default value is 5. If this parameter exceeds the number of I/O tags, it causes all I/O tags to be read in one cycle, which is not interrupted because of sending a token. We recommend not to increase the value of this parameter if other master devices are connected with RS485, which could have problems if they do not receive a token for a longer time.
  MS/TP usage_timeout: 99. According to a standard, the value must be under 100 ms. The default value of 20 ms caused problems with communication (MBG-MSTP did not manage to react within a specified timeout).
- Station type: *MS-TP*
- Address: *1* (according to the configuration of MBG-MSTP)
- I/O tag configuration:
  Request type: ReadProperty
  Object type: analog-input(0)
  Instance: according to the configuration of MBG-MSTP or a device which is behind it

Comment on iLON 10 Ethernet adapter

When using the iLON 10 Ethernet adapter for communication you should realize that the communication processor of this device (Neuron 3120) has relatively short default buffers (network buffer is 66 bytes). Therefore it does not receive longer packets. It can be seen in the web interface of iLON 10 - on the *Status* tab, the *Missed Messages* number increases when trying to read the value (e.g. always after saving the I/O tag if its parameter Subscribe/read is marked). This problem occurred during testing when a time plan of a scheduler contained more than 4 *time-value* pairs. When there were 4 pairs, the length of the LON message was 63 bytes. After adding another pair via PXM20, the reading of the scheduler failed.

Warning! The *NodeUtil* utility enables to increase the size of the received packet by increasing both the network and application buffers. However, if you set **improper values, you can damage the iLON 10 device** (Neuron 3120 stopped communication with the user processor).
If you decide to reconfigure the size of buffers, just change it from 66 bytes to 88 (and reduce the number of buffers) because the KOM process informs, in its packets, that it is able to receive 50-byte ASDU (+ 3 bytes BACnet header and 16 bytes LON header).
The buffers of the PCLTA-10 ISA adapter (built on the Neuron 3150 communication chip) had long enough buffers (255 bytes).
After we bought a new iLON 10, the configuration was successful with these parameters:

- configuration software: *Echelon Node Utility Release 1.82*
- sizes and quantities of buffers:

```
DEVICE:0> (B)uffer configuration
Node buffer configuration
Type                      Count        Size        Bytes
Receive transaction       11        13        143
Transmit transaction       2        28        56
App buffer in               2        82        164
App buffer out               2        82        164
Net buffer in               5        82        410
Net buffer out               2        82        164
App buff out priority       1        82        82
Net buff out priority       1        82        82
==> Total bytes = 1265
```

Comment on Easylon USB Interface+

The use of the *Easylon USB Interface+* adapter on 64-bit Windows 10 (with 32-bit D2000) has been tested.
It was necessary to specify the device name *EasyLVU11-3-Mip0* in the line configuration (the *EasyLVU11-3-Vni0* device did not work). The OpenLDV 5.1 driver had to be installed on the computer.

Comment on BACnet MS/TP implementation

The implementation of BACnet MS/TP is not complete yet. It was tested only in an elementary configuration (the D2000 KOM against a BACnet MS/TP MicroGateway produced by the York company). The communication works with both the Serial line (RS485/RS232 converter was used) and SerialOverUDP Device Redundant line (Moxa NPort series 5xxx was used).
MicroGateway (York company) implements the communication of an MS/TP Master type (i.e. it sends the *Poll for master* frame to detect the existence of other Master stations). The D2000 KOM relies on a partner station and does not implement a method of sending this frame type. Also, it does not handle a failure of the station to which it sends a token.
The present implementation is applicable only for communication with one Master station. It can be used for communication with one or more Slave stations (not tested). Also, the time ratio could cause some problems in heavily loaded systems, i.e. KOM could answer the *Poll for master* or *Token* frames later than required, which might cause collisions on the line. The time ratios may become even worse when you activate the logs on the line.

Comment on BBMD (BACnet Broadcast Management Devices) support

The following feature concerning the conversion of text names to numeric addresses of measured points (Siemens Desigo devices) on the TCP/IP-UDP line has been added to the BACnet protocol (and is available in versions published after 17.01.2012): if the KOM process sends a Who-Has request and the I-Have response comes from another IP address, the KOM process searches for the station that sent the Who-Has request with the text name specified in the response (within the line). If such a station is found, the answer is matched to its challenge.
This feature is useful when communicating with multiple Siemens routers (behind which are, for example, BACnet/LON devices), one of which is configured as a BBMD (Bacnet Broadcast Management Device) and the other is not.
Model situation:
The KOM process communicates with two Desigo PXG80-N BACnet routers (Rtr1 and Rtr2). Behind each router is a LON network, for simplicity with a single Desigo PXM20 station (Des1 or Des2). Router Rtr2 is configured to forward BACnet broadcasts from the LON network to Rtr1.

- Rtr1 has an IP address of 10.0.0.1 and has BBMD functionality enabled. Behind Rtr1 is the LON network with network address 11 and a Desigo PXM20 device (Des1) with address 1.1
- Rtr2 has an IP address of 10.0.0.2 and has BBMD functionality turned off. Behind Rtr2 is the LON network with network address 12 and a Desigo PXM20 device (Des2) with address 2.2

Let's have one TCP/IP-UDP communication line and on it two BACnet/IP stations representing Des1 and Des2

- L.Bacnet line: TCP/IP-UDP type
- Station B.Des1:
  Station type: BACnet/IP
  Address: 10.0.0.1 (address of Rtr1, behind which Des1 is located)
  Port: 47808 (standard BACnet port)
  Destination network: 11 (address of network behind Rtr1)
  Destination address: 1.1 (address of Des1 in a LON network)
  Register-Foreign-Device: enabled (for the KOM process to register as a broadcast recipient on Rtr1, which has active BBMD functionality)
- Station B.Des2:
  Station type: BACnet/IP
  Address: 10.0.0.2 (address of Rtr2, behind which Des2 is located)
  Port: 47808 (standard BACnet port)
  Destination network: 12 (address of network behind Rtr2)
  Destination address: 2.2 (address of Des2 in a LON network)
  Register-Foreign-Device: disabled (Rtr2 has inactive BBMD functionality)
- Any I/O tag M.Des2_test on B.Des2 station (e.g. of SubscribeCOV type) with Address type = Name

The KOM process sends a *Who-Has* request to the Des2 device to convert the text name configured in the M.Des2_test address. The IP address of Rtr2 (10.0.0.2) is used for sending.
Rtr2 forwards the request to the LON network to Des2 (according to the parameters *Destination network* = 12 and *Destination address* = 2.2 specified in the configuration of B.Des2 station). An *I-Have* broadcast response arrives from the BACnet/LON device Des2. As the BACnet router Rtr2 is without BBMD support, it forwards the response (assuming it is configured to do so) to the BACnet router Rtr1, which has BBMD support enabled.
Rtr1 forwards the I-Have message as a UDP packet to the KOM process because the *Register-Foreign-Device* parameter of B.Des1 station is configured and thus the KOM process has registered as a broadcast message recipient at Rtr1. This way, the *I-Have* message gets to the KOM process with a different IP address than the original request. The functionality described above matches such a response with a request from the B.Des2 station based on the match of the text name of the object in the sent request.

At the same time, one configuration limitation results from the described procedure - all the described activities take place in the context of one line, so it is necessary that all stations that are within the scope of one BBMD device are located on one line.

Tell commands

Command	Syntax	Description
STWATCH	STWATCH StationName	Tell command sends commands ReadProperty, ReadPropertyMultiple and Subscribe to the station based on the configuration of individual I/O tags.

Literature

ANSI/ASHRAE Standard 135-2001: BACnet - A Data Communication Protocol for Building Automation and Control Networks

ASN.1 Complete by Prof. John Larmouth

Blogs

You can read blogs about BACnet protocol:

- Communication - BACnet protocol
- Communication - BACnet protocol, part 2
- Communication - BACnet protocol, part 3

Changes and modifications

-

Document revisions

Ver. 1.0 - August 30, 2005 - first version

Ver. 1.1 - October 20, 2005 - support of schedulers, reading, and writing of ASN sequences

Ver. 1.2 - November 22, 2005 - support of dynamic detection of I/O tag address from the object name, address cache in a file

Ver. 1.3 - June 14, 2006 - support of BACnet router (tested with PXG80-N)

Ver. 1.4 - April 02, 2008 - partial support of BACnet MS/TP protocol (tested on BACnet MS/TP MICROGATEWAY on the cooler produced by York company)

Ver. 1.5 - June 25, 2010 - testing the cooperation with E-DDC3.1 from SE-Elektronic GmbH

Related pages:

Communication protocols