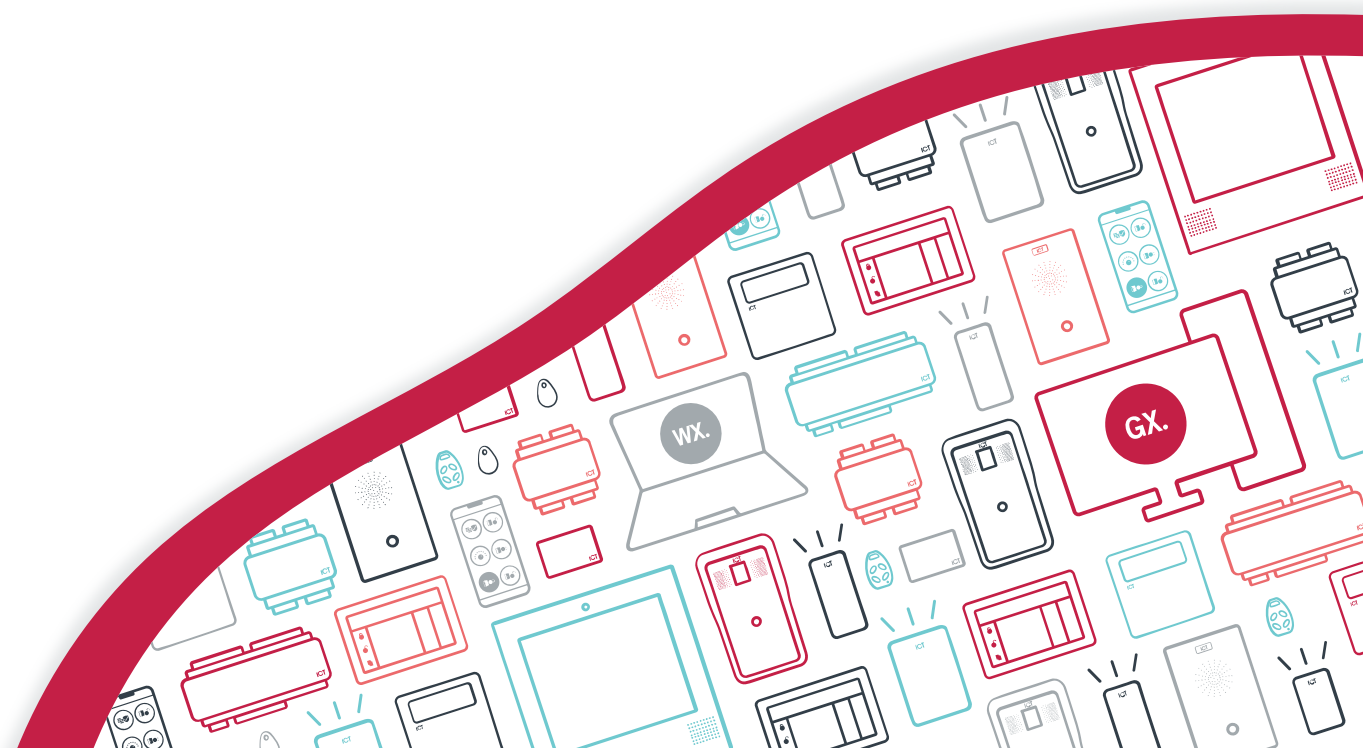
AN-318

BACnet Programming Using Commands

Application Note



Last Published: 26-Nov-21 10:04 AM

Contents

Introduction

The Building Automation Control network (BACnet) is a standardized protocol for communication between building automation devices, providing interoperability between cooperating systems.

BACnet building automation elements are known as objects and are generally classified as analog, binary or multi-state objects of defined object types. Protege GX controllers are able to monitor and control BACnet objects using the BACnet protocol. This is configured by applying specific configuration commands to the controller.

Corresponding records must be programmed in Protege GX to represent the BACnet objects, and mapped accordingly. Once the Protege GX records are configured and mapped to BACnet objects they can be programmed to monitor and control the BACnet objects in exactly the same way as other outputs and data values on the Protege network, including use with programmable functions.

Certain Protege GX records can also be exposed to the BACnet network to allow BACnet objects to interact with them as though they were connected on the BACnet system.

A maximum of 512 BACnet objects can be mapped to each Protege GX controller, and a maximum of 512 Protege GX records can be exposed to the BACnet network by each controller.

Terminology

The BACnet protocol represents control equipment as a collection of **devices** containing **objects**.

Devices

A BACnet device is the network-visible representation of control equipment connected to a BACnet network. This is effectively a BACnet control module to which BACnet objects are assigned.

Protege GX controllers are configured as devices on the BACnet network, and communicate with other BACnet devices over ethernet to monitor BACnet input objects and control BACnet output objects.

In this integration, a Protege GX controller is configured as a **local device**, while a BACnet network device is identified and defined as a **remote device**.

Each device has an **instance number** which identifies a device uniquely on the entire interconnected BACnet network. This instance number is also known as its **Device ID**, and must be unique for each device.

Objects

A BACnet object makes one specific control function of a device visible to other devices on a BACnet network.

BACnet uses objects to represent physical inputs, outputs and values. This is the general reference to sensors, actuators and other functional elements that make up a BACnet device.

For simplicity, a BACnet object is essentially an input or an output. Input objects have a present value associated with them. Output objects are referred to as commandable.

BACnet objects are **read from** (input) or **written to** (output), or can be both.

Protege GX controllers can expose defined Protege GX records to the BACnet network so that they may be read from and written to as objects in BACnet programming. Protege GX records exposed to the BACnet network are identified as **local objects**. BACnet objects on other devices are identified as **remote objects**, and need to be mapped to corresponding Protege GX records so that they can be monitored and controlled within Protege GX.

An object is uniquely identified by an **object identifier**, which consists of the **device** that it is on, an **object type** and **instance number**. The object identifier must be unique across the network.

How It Works

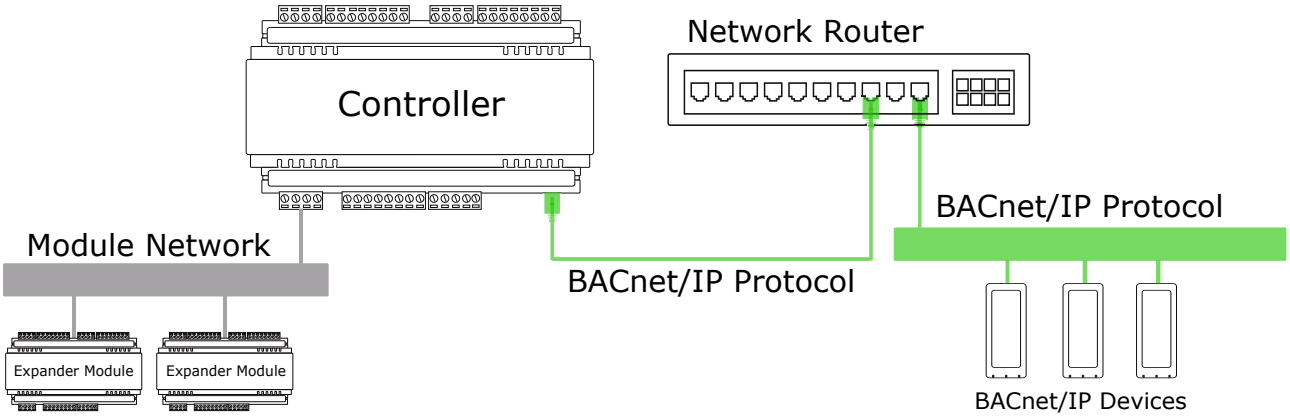
Protege GX controllers connect to BACnet devices over ethernet using the BACnet/IP network connection, which enables the controller to communicate directly with peer devices on the network.

BACnet MS/TP (Master Slave Token Passing) is not supported by Protege GX controllers.

BACnet/IP uses UDP for device communication. This is a connectionless protocol, so a device is considered connected or initialized once interrogated by the remote device for some basic properties, including its object list and the supported properties of each object.

The IP address is used to identify devices on the ethernet network. The IP address is used to physically route messages on the network, while the BACnet instance number identifies each device in the BACnet system.

The Protege GX BACnet interface supports local network connection only, so to communicate with BACnet devices the controller must be connected to the same local network as the BACnet devices. This means the controller must also be configured with the same IP range as the BACnet/IP network.



Protege GX modules communicate with the controller on the Protege GX module network as normal, independently of the BACnet network.

Prerequisites

All system components must be online and operational.

Controllers

Controller	Firmware Version	Notes
PRT-CTRL-DIN	2.08.590 or higher	Fractional analog values (see page 15) are supported in firmware version 2.08.755 or higher.
PRT-CTRL-DIN-1D		

Licensing

License	Order Code	Notes
Protege GX BACnet Core Service License	PRT-GX-BAC-CORE	1 license required per integration. Enables integration with up to 32 objects.
Protege GX BACnet 32 Object License	PRT-GX-BAC-PL32	1 license required for each additional 32 objects beyond the core license.

For example, for a system with 100 BACnet objects you would need 1 core license (first 32 objects) and 3 object licenses (additional 68 objects).

Preparing Protege GX Mapping

To ensure a smooth programming process, the following records should be prepared.

1. You will need the **IP address** and **Device ID** of all BACnet devices that the Protege GX controller(s) will communicate with. This will be available from the BACnet network administrator.
2. Identify a unique **Device ID** for each Protege GX controller that will communicate with BACnet devices.
3. You will need the **object type** and **instance number** of each BACnet object that you want to map to, along with the **device** they are assigned to.
4. Prepare the Protege GX records that will represent the BACnet objects.

   Note the **Database ID** of the Protege GX records. This will be required for mapping to the BACnet objects.

   - All **binary** and **multi-state** objects need to be created in Protege GX as **outputs**.
     - Remote BACnet **input** objects are **read from** by the controller, and control the Protege GX output records they are mapped to.
     - Remote BACnet **output** objects are **written to** by the controller, controlled by the Protege GX output records they are mapped to.
   - All **analog** objects need to be created in Protege GX as **data values**.
     - Each data value must be assigned to a virtual analog expander module. Add an analog expander module and enable the **Virtual Module** option. You can assign one data value to each of the four analog channels.

     Protege GX software automatically decides which controller data values should be downloaded to, based on usage. Programming the BACnet service using commands **does not** create this association, so data values must be assigned to virtual analog expanders to trigger the download to the appropriate controller.

   For more information on the required Protege GX records, refer to Remote Object Mapping (see page 8).
5. Record the **Database ID** of all Protege GX records that you want to expose to the BACnet network.
   Valid record types include data values, outputs, inputs, trouble inputs, areas, doors and floors.

   For more information on valid local record types, refer to Local Object Mapping (see page 11).

BACnet Programming

All BACnet programming is achieved through commands applied to the Protege GX controller. This includes identification and mapping of BACnet devices and objects, defining Protege GX controllers as local BACnet devices, and mapping to expose Protege GX records to the BACnet network as local objects.

It is important that all commands are entered exactly as provided as they are case and space sensitive.

The steps required to program BACnet operation in Protege GX include:

1. Enable the BACnet protocol and configure the controller as a device on the BACnet network.
2. Define BACnet remote devices.
3. Map BACnet remote objects.
4. Map local objects to expose Protege GX records to the BACnet network.

Enabling the BACnet Protocol

The Protege GX controller needs to be programmed to enable communication on the BACnet network. This includes enabling the BACnet protocol and configuring the controller as a local device to be recognized on the BACnet network.

1. Navigate to **Sites | Controllers** and select the controller that will communicate with BACnet devices.
2. Expand the **Commands** window and add the following commands:

Command	Function
`BACnet = true`	Enables the BACnet protocol on the controller.
`BNPort = 47808`	Defines the UDP port on which the BACnet protocol transmits and receives data. The port is configurable, but the default should be used where possible. The default port is 47808.
`BNDevId = <ID>`	Defines the Device ID which the controller will present itself as on the BACnet network. The Device ID should be coordinated with the BACnet network administrator to prevent ID conflicts with existing BACnet devices.
`BNUpdate = 5`	Sets the time in seconds between polling remote objects. Setting this too low or too high may cause undesirable outcomes. The recommended default is 5 seconds.

Note: The integration currently does not support BACnet COV (Change Of Value).

Multiple Protege GX controllers can be enabled to communicate on the BACnet network. Each controller needs to be configured as a local device and requires a unique BACnet Device ID.

Defining Remote Devices

Each BACnet device that the Protege GX controller(s) will communicate with needs to be identified, and defined as a **remote device**. You must know the IP address and device ID (instance number) of the remote device in order to configure a connection.

Add the following command to the controller for each device:

`BNDev0 = <IP address>,<Device ID>`

This command identifies a remote device on the BACnet network by defining its IP address and device ID, and assigning a BNDev number for local identification.

Example

`BNDev0 = 192.168.1.22,1234`

This demonstrates a command where 192.168.1.22 is the IP address of a BACnet device with device ID 1234.

To define more than one remote device, add multiple commands of the form:

`BNDev0 = ###, BNDev1 = ###, BNDev2 = ###`, etc.

The BNDev number is used in the Remote Object Mapping below to identify the BACnet device that remote objects are assigned to. Remote device numbering must start from BNDev0.

Remote Object Mapping

Each BACnet object to be monitored or controlled in Protege GX needs to be mapped to Protege GX records through controller commands. The commands automatically map the remote objects to the appropriate Protege GX record type, based on the **object type** defined in the remote object mapping.

- All **binary** and **multi-state** objects are mapped to Protege GX **outputs**.
- All **analog** objects are mapped to Protege GX **data values**.

The remote object mapping command maps a single remote object on the BACnet network to a corresponding Protege GX record, using its database ID. Add the following command for each object:

`BNRO = <BNDev>,<Object Type>,<Instance Number>,<Database ID>`

- `<BNDev>`: The defined BNDev number of the BACnet device that the object is assigned to.

  BNDev numbers are configured in Remote Device Mapping(see previous page).
- `<Object Type>`: The BACnet object type code for the remote object, as defined in the table below.
- `<Instance Number>`: The instance number of the remote object.
- `<Database ID>`: The database ID of the Protege GX record the remote object will be mapped to.

BACnet Object Type Codes - Remote Objects

BACnet Object Type	BACnet Object Type Code
Analog Input	AI
Analog Output	AO
Analog Value	AV
Binary Input	BI
Binary Output	BO
Binary Value	BV
Multi-State Input	MI
Multi-State Output	MO
Multi-State Value	MV

Example

`BNRO = 0,BO,1234,20`

The above command line will map the remote object as:

- In remote device 0, as configured in the remote device mapping
- A binary output (BO)
- With instance 1234
- Mapped to the Protege GX output record with a database ID of 20

Array of Remote Objects Mapping

The array of remote objects mapping command provides a count value, allowing you to map an array of remote objects on the BACnet network to corresponding sequential Protege GX records. Add the following command for each array:

```
BNARO = <BNDev>,<Object Type>,<Instance Number>,<Database ID>,<Count>
```

When a count value is present, the mapping will apply to that number of sequential remote object instance numbers, starting with the defined entry (e.g. 1234, 1235, 1236...)

The corresponding Protege GX records will start with the defined Database ID, then use the next available valid IDs in increasing order.

Example

```
BNARO = 0,BO,1234,20,5
```

The above command line will map 5 remote objects as:

- In remote device 0, as configured in the remote device mapping
- Binary outputs (BO)
- Starting from instance 1234
- To the Protege GX output records starting with the database ID of 20

Only valid record IDs will be mapped. For example, if there were Protege GX data value records with database IDs 20, 21, 22, 30, 31, the above command line would map:

- remote binary output object instance 1234 to local output 20
- remote binary output object instance 1235 to local output 21
- remote binary output object instance 1236 to local output 22
- remote binary output object instance 1237 to local output 30
- remote binary output object instance 1238 to local output 31

Array of Physically Mapped Remote Objects

The array of physically mapped remote objects command allows you to map an array of remote objects on the BACnet network directly to the physical outputs on a Protege GX output expander, using the module address and output number of the physical expander outputs rather than the database ID of specific output records.

A Protege GX output record is still necessary for each mapped physical output, even though the object is not mapped directly to the output record in the command.

The BACnet remote objects will be mapped to the defined module address, starting at the defined module output number. Add the following command for each array:

```
BNPARO = <BNDev>,<Object Type>,<Instance Number>,<Address>,<Output>,<Count>
```

- `<Address>`: The module address of the output expander module to map the remote objects to.
- `<Output>`: The module output number as programmed in Protege GX.

Physical mapping is valid for **binary** and **multi-state** BACnet objects only (**not** analog), and only for mapping to Protege GX **output expanders**.

Example

```
BNPARO = 0,BO,5,1,1,16
```

The above command line will map 16 remote objects as:

- In remote device 0, as configured in the remote device mapping
- Binary outputs (BO)
- Starting from instance 5
- To the Protege GX output expander with the module address of 1
- Starting at module output 1

The above command line would map:

- remote binary output object instance 5 to the output expander with module address 1, output 1
- remote binary output object instance 6 to the output expander with module address 1, output 2

  ...
- remote binary output object instance 20 to the output expander with module address 1, output 16

Programming Example

A full command sequence to monitor 68 analog inputs and 272 binary inputs may be as follows:

```
BACnet = true
BNPort = 47808
BNDevId = 2
BNUpdate = 5
BNDev0 = 192.168.1.55.1

BNARO = 0,AI,0,0,68

BNPARO = 0,BI,0,1,1,16
BNPARO = 0,BI,16,2,1,16
BNPARO = 0,BI,32,3,1,16
BNPARO = 0,BI,48,4,1,16
BNPARO = 0,BI,64,5,1,16
BNPARO = 0,BI,80,6,1,16
BNPARO = 0,BI,96,7,1,16
BNPARO = 0,BI,112,8,1,16
BNPARO = 0,BI,128,9,1,16
BNPARO = 0,BI,144,10,1,16
BNPARO = 0,BI,160,11,1,16
BNPARO = 0,BI,176,12,1,16
BNPARO = 0,BI,192,13,1,16
BNPARO = 0,BI,208,14,1,16
BNPARO = 0,BI,224,15,1,16
```

`BNPARO = 0,BI,240,16,1,16`

`BNPARO = 0,BI,256,17,1,16`

This sequence would correspond to:

- 68 remote analog input objects, with instance numbers starting at 0 and sequentially increasing to 67,
- mapped to 68 Protege GX data values with database IDs starting at 0 and increasing to 67.

It would also map:

- 272 remote binary input objects, with instance numbers starting at 0 and sequentially increasing to 271,
- to the local physical outputs located on 17 output expanders, each with 16 outputs, with sequential module addresses from 1 to 17.

To ensure the data values are downloaded to the controller on which the BACnet service is running, it would be necessary to program 17 virtual analog expander modules and assign four data value records to each.

Local Object Mapping

Certain Protege GX records can be exposed to the BACnet network. To do this the record needs to be defined as a recognized object type, and its properties and identification defined.

The local object mapping command maps a single Protege GX record as a local object on the BACnet network.

`BNLO = <Object Type>,<Read/Write>,<Database ID>`

- `<Object Type>`: The BACnet object type code to define the object type of the local object, as defined in the table below.
- `<Read/Write>`: Defines whether the local object can be read from (**R**), written to (**W**) or both read from and written to (**RW**).
- `<Database ID>`: The Database ID of the Protege GX record that will be exposed to the BACnet network.

BACnet Object Type Codes - Local Objects

Protege GX Record Type	BACnet Object Type Code
Data Value	DV
Output	OP
Input	IP
Trouble Input	TI
Area	AR
Door	DR
Floor	FL

Only the Protege GX record types included in the table above can be exposed to the BACnet network.

Example

`BNLO = DR,RW,36`

The above command line will expose the Protege GX door record with Database ID 36 to the BACnet network as:

- A door (DR)
- Which can be read from and written to (RW)

When the controller exposes local objects to the BACnet network, the instance number of the objects is defined by the order in which the commands are programmed. The first local object is assigned instance 0.

Local Object Values

The table below describes the values for **local objects** that the controller exposes to the BACnet network.

Data Type	Read Type	Value	Description
Data Value	Analog	0 - 65535	The current value of the data value
Output	Binary	0	Output is off
		1	Output is on or pulsing
	Multi-State	0	Output is off
		1	Output is on
		2	Output is pulsing
		3	Output is on for a timed duration
		5	Output is off due to conditional override
		6	Output is on due to conditional override
Input	Multi-State	0	Input is closed
		1	Input is open
		2	Input is short circuit
		3	Input is open circuit (tamper)
Trouble Input	Multi-State	0	Trouble input is closed
		1	Trouble input is open
Area	Multi-State	0	Area is disarmed
		1	Area is disarming
		2-6	Area is checking conditions prior to arming
		7	Area has open inputs preventing arming
		8	Area has too many inputs bypassed, preventing arming
		9	Area failed to arm due to open trouble inputs
		10	Area has inputs that are bypassed
		11	Area count is greater than zero, preventing arming
		12	Area is disarming a child area
		13	Area arming is paused waiting for user feedback at a keypad
		14	Area arming is waiting for a remote controller to respond
		128	Area is armed
		129	Area is in exit delay
		130	Area is in entry delay
		131	Area is in disarm delay
		132	Area is in code delay

Data Type	Read Type	Value	Description
Door	Multi-State	**Lower 3 bits**	
		0	Door is closed
		1	Door is open
		2	Door is in the pre-alarm state
		3	Door is left open
		4	Door is forced open
		Upper 5 bits	
		0	Locked
		1	Unlocked by user
		2	Unlocked by schedule
		3	Unlocked by user timed
		4	Unlocked by user latched
		5	Unlocked by exit device
		6	Unlocked by entry device
		7	Unlocked by operator
		8	Unlocked by operator timed
		9	Unlocked by operator latched
		10	Unlocked by area
		11	Unlocked by fire alarm
		12	Locked by conditional exception
		13	Unlocked by conditional exception
		14	Unlocked by user using extended door time
		15	Unlocked by exit device using extended door time
		16	Unlocked by entry device using extended door time
		17	Unlocked by operator using extended door time
		18	Locked using extended door time
		19	Locked down (entry allowed)
		20	Locked down (exit allowed)
		21	Locked down (entry/exit allowed)
		22	Locked down (full lockdown)
		23	Not locked (in the locked state but not secure)
		24	Not locked conditional (in the locked state but not secure, with a calendar action active)
Floor	Multi-State	0	Floor is restricted access
		1	Floor is free access

Multi-State Values Re-Mapping

Multi-state values can be configured so that the value that is actually applied to the local object is different to the value that is sent to the controller on the BACnet network.

The mapping scheme is applied to all objects of the specified type, so all outputs use the same mapping, all inputs use the same mapping, etc.

The multi-state values re-mapping command configures the value applied to all local objects of a defined object type from the defined value received from the BACnet network.

`BNMAP = <Object Type>,<Written Value>,<Applied Value>`

- `<Object Type>`: The BACnet object type code to define the object type of the local object, as defined in the table below.
- `<Written Value>`: The value written to the controller from the BACnet network.
- `<Applied Value>`: The value applied to the local objects by the controller.

BACnet Object Type Codes - Local Objects

Protege GX Record Type	BACnet Object Type Code
Output	OP
Input	IP
Trouble Input	TI
Area	AR
Door	DR

Example

`BNMAP = IP,1,2`

The above command line will re-map the multi-state values as:

- For all inputs (IP)
- When the written value is 1
- The applied value will be 2

Additionally, d represents default, so the command line **`BNMAP = IP,d,3`** means that if any value that is not explicitly mapped is written to a local input object, the actual value applied to that local input object will be 3.

Local Object Multi-State Values

The table below describes the valid values that can be applied to **local objects** that the controller exposes to the BACnet network.

Data Type	Value	Description
Outputs	0	Turn on
	1	Turn off
Inputs	0	Sealed
	1	Alarm
	2	Short
	3	Tamper
Trouble Inputs	0	Sealed
	1	Alarm
Areas	0	Disarm normal
	1	Disarm 24
	2	Disarm all
	3	Arm
	4	Force arm
	5	Instant arm
	6+	Reserved
Doors	0	Lock
	1	Unlock
	2	Unlock and latch
	3+	Reserved

Fractional Analog Values

For the BNLO, BNRO and BNARO commands (i.e. excluding the array of physically mapped remote objects) it is possible to add an optional **scale factor** and **sign** when reading and writing **analog** values.

Internally the Protege GX controller treats all data values as 16 bit unsigned values (ranging from 0 to 65535). If a device on the BACnet network uses real numbers rather than integers, the real number can be scaled before being stored in the Protege GX controller.

To achieve this add **,1,u** to the end of the command, where the **1** is the number of decimal places to shift the value (the scale) and the **u** defines whether the value is signed (**s**) or unsigned (**u**).

Unsigned Example

BNLO = DV,RW,5,2,u

The above command would expose the local data value with database ID 5 to the BACnet network for reading and writing, as an unsigned value.

The **2** decimal places in the command represents scaling by two orders of magnitude.

1. When an analog value is **written to** the data value, the value will be **multiplied by 100** and stored.
2. When the value is **read from** the data value, the stored value will be **divided by 100** before being sent over the BACnet network.

- If the value 12.345 is written it will be internally stored as 1234
  When it is read back it will be returned as 12.340
- If the value 556.2 is written it will be internally stored as 55620
  When it is read back it will be returned as 556.20
- If the value 100000 is written it will be capped and stored as 65535 (the highest number that can be stored)
  When it is read back it will be returned as 655.35

Signed Values

If the **u** at the end is replace by an **s** the number is treated as signed.

- The number stored inside the controller still has the range 0 to 65535.
- Values from **0** to **32767** are treated as **positive**.
- Values from **32678** to **65535** are treated as **negative**.

Stored Value	Written Value
65535	-1
65534	-2
...	...
32769	-32767
32768	-32768
32767	32767
32766	32766
...	...
1	1
0	0

Signed Example

```
BNLO = DV,RW,5,2,s
```

- If the value 12.345 is written it will be internally stored as 1234
  When it is read back it will be returned as 12.340
- If the value -12.345 is written it will be internally stored as 64302 (65536 - 1234)
  When it is read back it will be returned as -12.340
- If the value 100000 is written it will be capped and stored as 32767 (the highest number that can be stored)
  When it is read back it will be returned as 327.67

BACnet Tools

BACnet Discovery Tool

While questions regarding BACnet system configuration and connection should generally be directed to the site's BACnet network administrator, the BACnet Discovery Tool (BDT) can be a useful tool for monitoring the BACnet network.

https://www.ccontrols.com/sd/bdt.htm

Wireshark

Other network tools such as Wireshark can also be useful for troubleshooting network connections:

https://www.wireshark.org/

Designers & manufacturers of integrated electronic access control, security and automation products.
Designed & manufactured by Integrated Control Technology Ltd.