



**by Schneider Electric**

# BACnet Addons Library User Guide

**Publisher:** HVAC Solution Center, Alpago (BL) – Italy

**Authors:** Federico Marcassa, Pierpaolo Armeli

**Doc. Version:** v1.4





## Table of Contents

<b>1. Introduction to BACnet</b>	<b>7</b>
1.1. Overview	7
1.2. BACnet/IP and BACnet MS/TP on Evolution/Advance	7
1.2.1. BACnet profile and options	8
1.3. BACnet Objects on Evolution/Advance	8
1.4. BACnet Objects on FreeStudio	9
1.4.1. Object Attributes	9
1.4.1.1. Object Type, Identifier and Name	9
1.4.1.2. Present Value, Priority Array and Relinquish Default	10
1.4.1.3. Attributes Properties	10
1.4.2. HowTo Restore the EEPROM BACnet defaults	11
1.4.3. Device Object	12
1.4.4. Analog Objects	13
1.4.4.1. Analog_Input	13
1.4.4.2. Analog_Output, Analog_Value	13
1.4.5. Binary Value Object	13
1.4.6. Multi State Objects	14
1.4.7. Calendar Object	14
1.4.8. Schedule Object	14
1.4.9. Notification Class Object	15
<b>2. BACnet Addons Library</b>	<b>16</b>
2.1. Overview	16
2.2. Description	16
2.3. Priority Array and Relinquish Default Management in the Library	17
2.4. Function Blocks and Programs	18
2.4.1. List of the FBs and Progs	18
2.4.2. BACnet_PVUpdPr	19
2.4.3. BACnet_AI_SetPV	20
2.4.4. BACnet_RelDef_AV_link_*EE	21
2.4.4.1. BACnet_RelDef_AV_link_iEE	21
2.4.4.2. BACnet_RelDef_AV_link_rEE	25
2.4.4.3. BACnet_RelDef_AV_link_uiEE	25
2.4.5. BACnet_BI_SetPV	25
2.4.6. BACnet_RelDef_BV_link_xEE	27
2.4.7. BACnet_MSI_*SetPV	29
2.4.7.1. BACnet_MSI_diSetPV	29
2.4.7.2. BACnet_MSI_udiSetPV	31
2.4.7.3. BACnet_MSI_usiSetPV	31
2.4.8. BACnet_RelDef_MV_link_*EE	31
2.4.8.1. BACnet_RelDef_MV_link_iEE	31
2.4.8.2. BACnet_RelDef_MV_link_rEE	34
2.4.8.3. BACnet_RelDef_MV_link_udiEE	34
2.4.8.4. BACnet_RelDef_MV_link_uiEE	35
2.4.8.5. BACnet_RelDef_MV_link_usiEE	35



2.4.9.	BACnet_ObjID	35
2.4.10.	BACnet_ObjID_De_store_EE	35
2.4.11.	BACnet_UdAsReal_read	36
2.4.12.	BACnet_UdAsReal_write	37
2.4.13.	BACnet_udiValue_link_rEE	38
<b>3.</b>	<b>BACnet Application Project Sample</b>	<b>39</b>
3.1.	Overview	39
3.2.	Sample thermostat application	39
3.2.1.	hysteresis_ST	39
3.2.2.	Thermostat	39
3.2.3.	usiOperating_State	39
3.2.4.	Other status variables and EEPROM parameters	39
3.3.	BACnet Objects Templates	40
3.4.	Implementing the BACnet protocol into an existing application	40
3.4.1.	Load the BACnet_IEC.PLL and BACnet_Addons.plclib libraries	40
3.4.2.	Add the Device BACnet Object	40
3.4.3.	Add an Analog Value BACnet Object	40
3.4.4.	Add a Binary Value BACnet Object	41
3.4.5.	Add a Multi State Value BACnet Object	41
3.4.6.	Define the EEPROM parameters & StatusVars used in the BACnet setup	41
3.4.7.	HowTo Code the BACnet_Setup_Init program	42
3.4.8.	HowTo Code the BACnet_BBMD_ReinitDevice program	43
3.4.9.	HowTo Automate the Present_Value update procedure	44
3.4.10.	HowTo Link an EEPROM Parameter to a BACnet Analog Value Object	45
3.4.11.	HowTo Link an EEPROM Parameter to a BACnet Binary Value Object	46
3.4.12.	HowTo Configure a Multi State Input BACnet Object	47
3.4.13.	HowTo Configure a Binary Input BACnet Object	47
3.4.14.	HowTo Configure an Analog Input BACnet Object	48
3.4.15.	HowTo Link an EEPROM Parameter to a BACnet Multi State Value Object	48
3.4.16.	HowTo Setup a Schedule BACnet Object	49
3.4.17.	HowTo Setup a Calendar BACnet Object	51
3.4.18.	HowTo Setup a Notification Class BACnet Object	51
3.4.19.	EEPROM variable names and HMI	51
3.5.	Testing BACnet with YABE	52
3.5.1.	HowTo Add a BACnet Device in YABE and Subscribe to an Object	52
3.5.2.	HowTo Subscribe a BACnet Object	52
3.5.3.	HowTo Change the Relinquish_Default	53
3.5.4.	HowTo Write on the Priority_Array	53
3.5.5.	HowTo Edit a Calendar Object	53
3.5.6.	HowTo Edit a Schedule Object	54
3.5.7.	HowTo Reinitialize the Device via BACnet	54
<b>4.</b>	<b>Appendix</b>	<b>55</b>
4.1.	Hardware Information	55
4.1.1.	Usage of the EEPROM vs BACnet	55
4.2.	Acronyms	55
<b>5.</b>	<b>Publisher's Info</b>	<b>56</b>



## Safety Information and Good Practices

### *Before You Begin*

#### General

The products specified in this document have been tested under actual service conditions. Of course, your specific application requirements may be different from those assumed for this and any related examples described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any examples specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

© 2018 Eliwell Controls Srl. All rights reserved.

#### **WARNING**

##### **REGULATORY INCOMPATIBILITY**

Be sure that all equipment applied and systems designed comply with all applicable local, regional and national regulations and standards

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

#### Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Eliwell Controls for any consequences arising out of the use of this material. A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved. Failure to observe this information can result in injury or equipment damage.

The use and application of the information contained herein require expertise in the design and programming of automated control systems. Only the user or integrator can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or process, and can therefore determine the automation and associated equipment and the related safeties and interlocks which can be effectively and properly used. When selecting automation and control equipment, and any other related equipment or software, for a particular application, the user or integrator must also consider any applicable local, regional or national standards and/or regulations.

Some of the major software functions and/or hardware components used in the proposed architectures and examples described in this document cannot be substituted without significantly compromising the performance of your application. Further, any such substitutions or alterations may completely invalidate any proposed architectures, descriptions, examples, instructions, wiring diagrams and/or compatibilities between the various hardware components and software functions specified herein and in related documentation. You must be aware of the consequences of any modifications, additions or substitutions.



A residual risk, as defined by EN/ISO 12100-1, Article 5, will remain if:

- it is necessary to modify the recommended logic and if the added or modified components are not properly integrated in the control circuit;
- you do not follow the required standards applicable to the operation of the machine, or if the adjustments to and the maintenance of the machine are not properly made (it is essential to strictly follow the prescribed machine maintenance schedule);
- the devices connected to any safety outputs do not have mechanically-linked contacts.

### CAUTION

#### **EQUIPMENT INCOMPATIBILITY**

Read and thoroughly understand all device and software documentation before attempting any component substitutions or other changes related to the application examples provided in the document

**Failure to follow these instructions can result in injury, or equipment damage.**

## ***Start-Up and Test***

Before using electrical control and automation equipment after design and installation, the application and associated functional safety system must be subjected to a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such testing be made and that enough time is allowed to perform complete and satisfactory testing.

### CAUTION

#### **EQUIPMENT OPERATION HAZARD**

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices
- Remove tools, meters, and debris from equipment.

**Failure to follow these instructions can result in injury, or equipment damage.**

Verify that the completed system, including the functional safety system, is free from all short circuits and grounds, except those grounds installed according to local regulations. If high-potential voltage testing is necessary, follow the recommendations in equipment documentation to help prevent injury or equipment damage.



## Operations and Adjustments

Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly installed and operated.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the hands and other parts of the body are free to enter the pinch points or other hazardous areas where serious injury can occur. Software products alone cannot protect an operator from injury. For this reason, the software cannot be substituted for or take the place of point-of-operation protection.

### WARNING

#### **UNGUARDED MACHINERY CAN CAUSE SERIOUS INJURY**

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

**Failure to follow these instructions can result in death, serious injury, or equipment damage.**

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

**NOTE:** Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the examples and implementations suggested herein. It is sometimes possible to adjust the equipment incorrectly and this produce unsatisfactory or unsafe operation. Always use the manufacturer instructions as a guide to functional adjustments. Personnel who have access to these adjustments must be familiar with the equipment manufacturer instructions and the machinery used with the electrical equipment.

Only those operational adjustments actually required by the machine operator should be accessible to the operator. Access to other controls should be restricted to help prevent unauthorized changes in operating characteristics.



## Introduction to BACnet

### 1.1. Overview

BACnet (Building Automation Control networks) is a communications protocol for building automation and control networks. It follows the ASHRAE, ANSI, and ISO standards and it is a registered ASHRAE trademark. BACnet® is an 'open' – i.e. non-proprietary – data exchange protocol.

It was designed to allow communications/information exchange between computerized building automation devices and control systems communications in a range of applications including HVAC, lighting control, detection systems, etc.



In this chapter you can find the details about which are the BACnet functionalities implemented in the Evolution/Advance controllers. For all the details about the BACnet protocol, the key reference is the ASHRAE BACnet specification (ANSI/ASHRAE Standard 135-2008).



The Advance device family is BTL Certified.

### 1.2. BACnet/IP and BACnet MS/TP on Evolution/Advance

The BACnet protocol guarantees full interoperability between control devices for BMS by defining a set of simple, unambiguous rules for communication between devices made by different manufacturers on a common platform.

The various Evolution/Advance controllers implement the BACnet IP protocol (it works through IP address-based Ethernet connection and therefore uses UDP/IP frames) and/or the BACnet MS/TP protocol (using the RS485 connection).

BACnet MS/TP and the BACnet IP protocols cannot be active at the same time.

Note that the BACnet MS/TP and the BACnet IP protocols are both already embedded onboard on the **EVP** and **Advance** products.

On the **EVD** and **EVC** controllers, *a plug-in communication module is required* to implement the BACnet protocol.

On the **Advance 4DIN** products, the BACnet MS/TP is already on-board. To have BACnet/IP, the above mentioned plug-in is required.

The BACnet protocol defines a certain number of objects and a series of services used for BMS communication. The BACnet /IP and MS/TP protocols management is implemented in the **Application (to define BACnet Objects)** and Device (*Load\_BACnet\_E2\_Defaults*, *Port\_BACnet\_IP* parameters) work environments of FreeStudio.

Note that BACnet/IP can coexist with all the other TCP functionalities.





### 1.2.1. BACnet profile and options

In this chapter you can find some details regarding which profile and options of the BACnet protocol are implemented in the Evolution/Advance controllers.

The implemented **Device Profile** is the BACnet Advanced Application Controller (B-AAC) with the highest conforming profile, and therefore the B-ASC, B-SA and B-SS profiles are also supported.

The implemented BACnet/IP profile has the **segmentation capability** on the transmission side and with a window size of 1476 byte/seg. It is not implemented for the received messages.

The BACnet/IP protocol operates with a data rate of 10/100 MBPS. The BACnet MS/TP slave protocol can use the following data rates: 9600, 19200, 38400, 76800 BPS. The Evolution/Advance controllers are not able to act as a Router for Data Link.

The device supports the registration as a foreign device. The supported character sets are ANSI X3.4 and ISO 8859-1.

### 1.3. BACnet Objects on Evolution/Advance

The BACnet protocol has a **standard set of objects**, which contain a standard set of attributes that can be read and written by other devices in the BACnet network. The objects are **controlled by other BACnet devices via their attributes**.

The **supported BACnet Object Types** on the Evolution/Advance controllers are the following:

Object Type		Description	MAX instances
ANALOG VALUE	Device	Describes the properties (e.g. name, identifier, firmware version). It is the only type of object which is compulsory and it can have only one instance defined, which MUST be unique.	1
	Analog Value	It represents an Analog Value. It can be used to manage analog setpoints.	256 NOTE 1
	Analog Input	It represents an Analog Input. It can be used to manage analog inputs, temperatures, values.	
	Analog Output	It represents an Analog Output. It can be used to manage analog outputs.	
MULTI STATE VALUE	Binary Value	It represents a binary value, which can be only two distinct values. It can be used for ON/OFF relay, digital I/O, TRUE/FALSE, etc.	256
	Multi State Value	It represents a vector of up to 5 enumerated value/input. It can be used for the Status of a process, e.g. the operating mode of a device.	32 NOTE 1
	Multi State Input		
	Calendar	It assumes a Boolean TRUE value if the current date and time is within the specified set or list of dates (defined as properties of the object); it is FALSE otherwise (if not within the specified dates). It can be used to specify Working days in the year, national and weekday holidays, etc.	4
	Schedule	It is assimilable to the definition of time bands: within a set of dates it can assume different values on certain days of the week and in certain instances. It can be used to define the lessons schedule for a class, to program a room thermostat, etc.	16





Notification Class	Notifications to be sent to devices in the network. It is used for Alarm messages to be sent.
--------------------	--

16

NOTE 1: For this type of object, the max number of instances is defined as the sum of all the types belonging to the father type (i.e. Analog Value number of instances has to be calculated as An. Value + An. Input + An. Output).

## 1.4. BACnet Objects on FreeStudio

**Note** that not all the BACnet Objects are supported on FreeStudio. Even if they are selectable from the drop-down list, they are treated as the root object type, when compiling on FreeStudio.

This is the case of the *Binary Input*, the *Binary Output* and the *Multi State Output*.

See the next paragraphs about them for more details. For example, a function block to use a *Binary Value* as a *Binary Input* is included in the library.

In this chapter, a description of each supported BACnet Object type is provided. A **BACnet Object Template** per each type is attached to the library: this helps to implement the various objects, as the attributes in the templates have already the suggested settings, which in general are the mostly required by the application.

### 1.4.1. Object Attributes

Depending on how it is defined by the ASHRAE standard on BACnet, each object can have scores of more or less complex **attributes** that can be specified in the PLC application using the various fields each attribute has. The attributes which are not specified in attributes table (see Fig. 2 - *Example of Object Attributes* for the table sample) will anyway be present in the Object and they will have the default properties. It is not possible to define an Object with only a few attributes.

Note: Before adding an object, it is required to connect to the *BACnet\_IEC.pll* library. See §3.2.1 for more details.

#### 1.4.1.1. Object Type, Identifier and Name

Each object has always the following attributes already characterized, which are used to uniquely identify each BACnet Object:

Attribute	Description
Object_Type	Defines the type of Object. It is not in the properties table list, but in the section above it (see Fig. 1) and is only present for the Objects that can have more than one type.
Object_Identifier	Generated automatically. 32-bit identifier linked to data / address type.
Object_Name	Text string used in the searches broadcast by the BACnet devices (used as Alias).

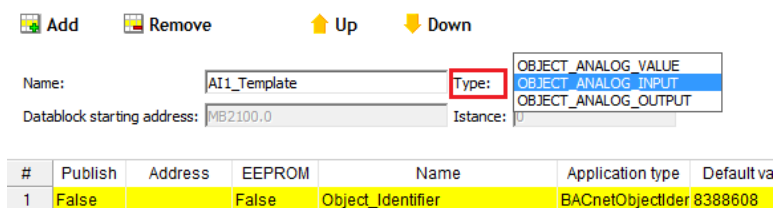
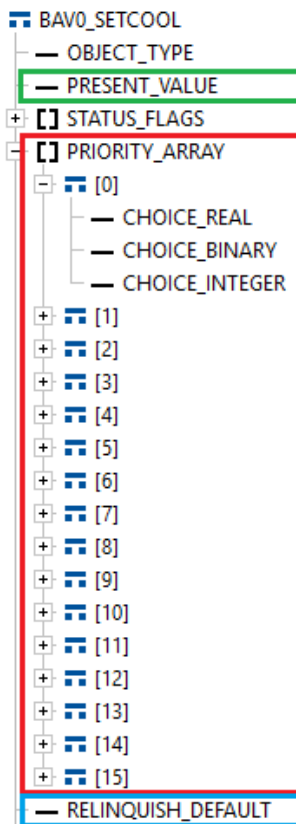


Fig. 1 - Object\_Type Location

The *Object\_Identifier* related to the Device BACnet Object is particularly relevant as it defines the BACnet ID of the entire device. See §1.4.3 *Device Object* for details.



#### 1.4.1.2. Present Value, Priority Array and Relinquish Default



Apart from the *Device* and the *Notification Class*, all the BACnet Objects have the **Present Value** attribute. This is a key attribute, which contains the current value of each BACnet Object and which has to be used for every calculation in the IEC code. It is also the value which, as a standard, a BACnet supervisor will read to inspect the value of a BACnet Object.

Two other key attributes are the **Priority Array** and the **Relinquish Default**, which are used by the BACnet stack in the determination of the **Present Value**.

The (supported) BACnet Objects which have these two attributes are:

- Analog Value;
- Analog Output;
- Binary Value;
- Multi State Value.

The **Present Value** is equal to the valorized (not *null*) variable with the highest level of priority. The variables taken into consideration for this evaluation are the **Priority Array** and the **Relinquish Default**.

The **lowest priority** variable is the **Relinquish Default**. The **Priority Array** has a **higher priority than** the **Relinquish Default**. Inside the **Priority Array**, the **highest priority** variable is the row [0] of the array, whilst the row [15] is the one with the **lowest priority**.

Therefore, for *example*, the **Present Value** will be equal to the **Relinquish Default** only if all the rows of the **Priority Array** are filled with null values.

The **Relinquish Default** is commonly used for the default value of a certain parameter, whilst the supervisor would normally write on the **Priority Array** to temporarily change the parameter value.

#### 1.4.1.3. Attributes Properties

The available fields to specify the properties of each attribute are the following:

Field	Description	
Publish	Defines if the attribute has to be published at Modbus/CAN level	
	<b>TRUE</b>	The attribute will be published at the address reported in the Address field, which will be assigned automatically.
	<b>FALSE</b>	Not published; the Address field is empty and the attribute will not be usable at Modbus/CANopen level.
Address	Modbus/CAN address. It is assigned automatically if Publish is TRUE. It is not assignable/editable if Publish is FALSE.	
EEPROM	Defines if the attribute has to be stored in non-volatile memory. The available space is 16 Kbytes.	
	<b>TRUE</b>	The attribute will be set as retentive (e.g. for counters, timers, etc.)
	<b>FALSE</b>	The attribute will be only available in RAM.



Application Type	Type of data used in the IEC code (UINT, BOOL, etc.). It can also be an enumeration available in the BACnet_IEC library, whose default value can be set using the drop-down menu which appears in "Default value".	
Default value	Initial value of the attribute.	
Format	Display format for the Default, Min and Max value	
Min	Minimum value of the attribute in case it is published	
Max	Maximum value of the attribute in case it is published	
Unit	Unit of Measurement in case it is published	
Read only	Defines if the attribute is read only or R/W on the ModBUS/CAN side.	
	<b>TRUE</b>	It is set as read only.
	<b>FALSE</b>	It is set as read/write
Description	Attribute description in case it is published	

**Note** that only the attributes configured as publish (i.e. `Publish = TRUE`) will be displayed in *Device* and will have their own ModBUS address. These are also the only attributes whose default values will be loaded in EEPROM (if the EEPROM field is TRUE) after that a download of the parameters default values is performed (e.g. during the *Download all* procedure in *Device*).

In case of not published attributes, to load their default values, follow what is illustrated in the next paragraph (§1.4.2 HowTo Restore the EEPROM BACnet).

It is possible to specify the properties of each BACnet attribute (e.g. if that attribute has to be published, if it has to be stored in EEPROM, etc.) using the table shown in Fig. 2.

#	Publish	Address	EEPROM	Name	Application type	Default value
1	False		False	Object_Identifier	BACnetObjectIdentifier	25165824
2	False		False	Object_Name	STRING	
3	False		True	Description	STRING	Calendar_Temp
4	False		False	Present_Value	BOOL	False
5	False		True	Date_list[0].date.year	USINT	255

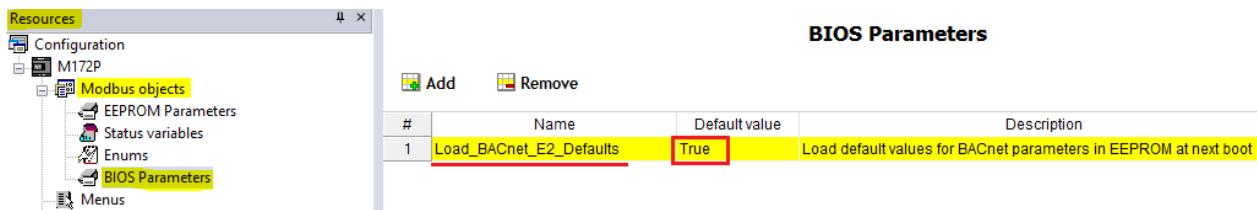
Fig. 2 - Example of Object Attributes

### 1.4.2. HowTo Restore the EEPROM BACnet defaults

The default values of the attributes of a BACnet Object that have the EEPROM field set as TRUE are normally not restored after a restart of the device. However, their default value is restored on the EEPROM using the *Download all* functionality of the *Device* work environment in case the attribute is published. The published attributes (`Publish = TRUE`) have a ModBUS address assigned and get therefore restored on the EEPROM.

In case of not published attributes (`Publish = FALSE`), the way to write the defaults on the EEPROM is to set the `Load_BACnet_E2_Defaults` parameter to TRUE. If it is TRUE, the defaults BACnet values are loaded in EEPROM at the next boot.

To enable this permanently, in the *Application* work environment, in *Resources*, under *Modbus objects\BIOS Parameters*, add a new parameter and select `Load_BACnet_E2_Defaults` from the drop-down list and set its *Default value* to TRUE.



When using *Device* to perform a *Download all*, the application asks if you want to download the parameters default values into the controller, answering Yes, it will download also the BIOS parameters defaults specified in the above BIOS Parameters section (i.e. also the Load\_BACnet\_E2\_Defaults value) and all EEPROM parameters with a ModBUS address. However, just right after that, when the device is rebooted (you need to answer Yes when you will be asked to, or do it manually), the BACnet EEPROM parameters defaults will be loaded, since the Load\_BACnet\_E2\_Defaults value will be TRUE.

**Note** that, after each reboot, the Load\_BACnet\_E2\_Defaults value is reset to FALSE. Therefore, to download the EEPROM BACnet defaults, a *Download all* from Device has to be performed. A simple restart (power off/ power on) of the controller is not sufficient, as the value of the Load\_BACnet\_E2\_Defaults parameter has to be set to TRUE before each reboot.

### 1.4.3. Device Object

The Device object is the only object that MUST always be present in a BACnet project. If it is not present and another BACnet Object of any other type is present, there will be a compilation error.

This object has two specific editable fields: Subnet and Node number. The combination of them will define the Object\_Identifier, which will identify the device (i.e. EVD/AVD) in the network of devices.

**NOTE:** the Object\_Identifier of each BACnet device **MUST be unique** in the BACnet network which the BACnet device is connected to. The IP address of each device must also be unique.

It is useful to have a non-static *Object\_Identifier* because the ID that the device can have depends also on the other devices present on the network to which the EVD/AVD will be connected. In this way, it will be not application-dependent and therefore it will be easier to connect more controllers with the same application on the same network.

This can be implemented by using the function BACnet\_ObjID, which calculates the ID based on two parameters (*Subnet* and *Node number*) that can be stored in the EEPROM. See §3.4.7 *HowTo Code the BACnet\_Setup\_Init* for more details.

The attributes Vendor\_Name and Vendor\_Identifier refer to the vendor details and are released by bacnet.org. For Eliwell Controls the ID is 10.

The Firmware\_Revision can be setup using the *BACnet\_Setup\_Init* (see §3.4.7).

The Application\_Software\_Version could be setup using a parameter linked to the application.

#	Publish	Address	EEPROM	Name	Application type	Default value	Format	Min	Max	Unit	Read only	Description
1	False		False	Object_Identifier	BACnetObjectIdentifier	33554432					True	
2	False		False	Object_Name	STRING	BACnetDevice_Template					False	
3	False		False	Application_Software_Version	STRING	1.0					False	
4	False		False	Firmware_Revision	STRING						False	
5	False		False	Vendor_Identifier	UINT	10					False	

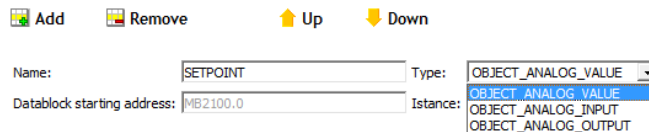
Attached to this documentation is available a template of the *Device BACnet Object*.



#### 1.4.4. Analog Objects

The *Analog Value* can be of three types:

- *Analog Input*;
- *Analog Output*;
- *Analog Value*.



##### 1.4.4.1. Analog\_Input

The *Present\_Value* of an *Analog Input* object type is directly writable from the IEC code. An AI BACnet Object will only read from the IEC / be written by the IEC code, but there will be no write operation from the BACnet side to the IEC code.

Templates to read a temperature and a humidity value from BACnet using an AI Object are provided together with this documentation.

The *Reliability* attribute for an *Analog Input* is valorized as follows:

Reliability value	Description	PV value (depending on the data format)
NO_SENSOR	generic probe error	-32768
		-3276.8
		-327.68
UNRELIABLE_OTHER	communication error with an I/O expansion	-32767... -32760
		-3276.7... -3276.0
		-327.67... 327.60
NO_FAULT_DETECTED	no error	in all the other cases

##### 1.4.4.2. Analog\_Output, Analog\_Value

In the case of an *Analog Value* or an *Analog Output*, the *Present\_Value* is calculated based on the *Relinquish\_Default* and the *Priority\_Array* contents.

The *Reliability* attribute has to be set by the PLC.

An *Analog Value* object template is provided together with this documentation.

#### 1.4.5. Binary Value Object

The *Binary Value* object is used for variables representing Boolean values such as enable Settings or relay ON/OFF states.

**Note** that the *Binary Input* and the *Binary Output* object types are not really supported and will be treated as *Binary Value* in the BACnet environment.

A template for a *Binary Value* is provided together with this documentation.

You can also find a template with the name *BI\_Template.xml* that will allow you to use a *Binary Value* as a *Binary Input*. It can be used in connection with the *BACnet\_BI\_SetPV* function block to implement a *Binary Input*.



#### 1.4.6. Multi State Objects

The *Multi State Value* object can vary within a finite number of distinct values. The maximum supported *Number of states* is five (5).

It is normally used for enumeration-type variables representing multiple states such as HEAT/COOL/OFF, defrost cycles (request, start, duration, dripping, etc.).

A text string is associated to each state. Note that the index used for the strings starts from 0, while the state number starts from 1.

Templates for a *Multi State Input* and a *Multi State Value* are provided together with this documentation.

**Note** that the *Multi State Output* object type is not really supported and will be treated as a *Multi State Value* in the BACnet environment.

#### 1.4.7. Calendar Object

The *Calendar* object defines a set of dates. Depending on the set and based on the current date of the device (RTC settings), the *Present\_Value* of the object will be TRUE or FALSE. The former if the current date is within the set, the latter if it is not.

The dates which can be defined are the following:

- a single date (a day);
- an interval of consecutive dates/days (a week, a month, ...);
- days that repeat in a cycle (first Monday of the month, every Saturday, every weekend, ...).

The *Calendar* is normally a static object.

The template included with this documentation can be used to have all the attributes properties already setup.

#### 1.4.8. Schedule Object

The *Schedule* object defines the planning of actions to be carried out in a certain dates interval, with a possible weekly recurrence and with possible exceptions, which can be defined either within the *Schedule* object itself or alternatively using a *Calendar* object in combination with it.

Based on which is the way you prefer to follow, you will find two Templates for the *Schedule* object, one for with and one without the exceptions already defined as EEPROM attributes.

Depending on the current date and time, the *Present\_Value* of the *Schedule* can assume **variable values** as defined in the properties of the object itself.

**Note** that the variable values will be stored in an UDINT variable. Therefore, even if a value in a different data type is passed, it is important to check if this is compatible with the UDINT data type. It is suggested to use the USINT, the UINT or the UDINT data format for the value to be passed.

Additionally, in case you would like to use REAL as data type, two functions to read and write an UDINT as REAL have been added to the BACnet Addons Library (see §362.4.11 and §2.4.12)





#### 1.4.9. Notification Class Object

The *Notification Class* object contains a list of devices to which notifications of messages and/or alarms are sent on the basis of certain events defined in the *Analog*, *Binary*, and *Multi State* objects.

Each event is associated to an action, and the *Notification Class* describes where and how the chosen message will be sent.

The `Notification_Class` attribute of each *Notification Class* object is of type Unsigned, it shall indicate the numeric value of this notification class and shall be equal to the instance number of the Notification Class object.

**Note** that the various Evolution/Advance do not exchange messages between them in the network. The notifications are sent to the SCADA devices, which could eventually send them to the Evolution/Advance.

**Note** also that each *notification* is generated by the objects themselves (e.g. Analog Value, Binary Value, etc.) and not by the *Notification Class* object which is linked to those objects.

Each of those objects actually contains the following attributes, which allow to activate a certain type of notification:

- `Limit_Enable[0], Limit_Enable[1];`
- `Event_Enable[0], Event_Enable[1], Event_Enable[2];`
- `Notification_Class.`

The value 4194303 for the `Notification_Class` attribute of an object shall be used to mark that object as not linked to any notification class.





## BACnet Addons Library

### 2.1. Overview

The current documentation (v1.4) is related to the **BACnet Addons library v.1.2.5**. The library is tested/supported on **FreeStudio 3.9.1 and greater versions**.

### 2.2. Description

The BACnet Addons library helps to simplify the implementation of the BACnet protocol into either new or existing projects. This library is complementary to the BACnet\_IEC library.

The **key functionalities** of the current library version are:

- Link an EEPROM parameter to:
  - an Analog Value BACnet Object (supports `INT`, `UINT`, `REAL`);
  - a Multi State Value BACnet Object (supports `INT`, `UINT`, `UDINT`, `USINT`, `REAL`);
  - a Binary Value BACnet Object (supports `BOOL`).
  - any `UDINT` attribute of any BACnet Object (supports `REAL`).
- Use an IEC variable to set the Present Value of:
  - an Analog Input BACnet Object (supports `INT`, `REAL`);
  - a Binary Value BACnet Object (supports `BOOL`);
  - a Multi State Input BACnet Object (supports `DINT`, `UDINT`, `USINT`).
- Set the `Object_Identifier` of the Device BACnet Object using EEPROM parameters and store in those parameters any change to the Identifier originating from the BACnet side.
- Automate the Present Value update procedure of each BACnet Object type.
- Be able to read and write an `UDINT` value as `REAL`.
- Ability to manage the Schedule BACnet Object value as `REAL`.

In addition, in §3 *BACnet Application Project Sample*, you will find a guide about how to implement the following functions:

- Setup the BBMD Service.
- Define the Warm Restart and Cold Restart procedures, which are executed by the `ReinitializeDevice` service.
- Write the BACnet Setup Init, which for instance contains also the call to the `Object_Identifier` function.

The **targets** supported by this library version are:

- **EVD** with BIOS 423.26 and greater;
- **EVC** with BIOS 477.26 and greater;
- **EVP** with BIOS 489.19 and greater;
- **AVD** with BIOS 596.6 and greater;
- **AVD 4DIN** with BIOS 668.6 and greater.



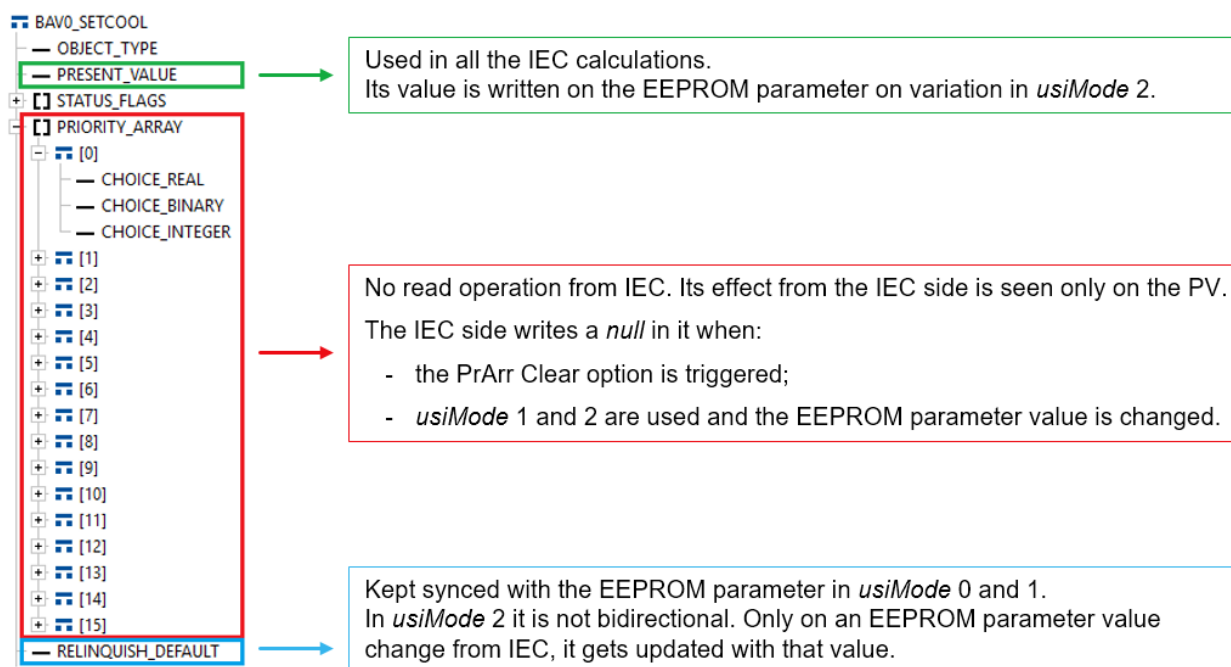
Together with the library and the implementation example, **templates of several BACnet Objects** are provided to achieve a smoother and easier implementation of the various Objects.

### 2.3. Priority Array and Relinquish Default Management in the Library

Note that the §1.4.1.2 *Present Value, Priority Array and Relinquish Default* paragraph is a prerequisite to the contents of the current one.

The “link” function blocks, which are detailed in the following chapters, can be used to link an EEPROM parameter to a BACnet Object. These function blocks, except the `BACnet_udiValue_link_rEE` which is mostly designed to be used in connection with the Schedule Object, are designed for the BACnet Objects which have the Priority Array and the Relinquish Default attributes.

The picture shows how these two attributes are managed by the library and where the Present Value is used.



Note that the Present Value is the first output of all the “link” function blocks.

The “link” function blocks have the `usiMode` input which defines how the update of the Relinquish Default and of the EEPROM parameter takes place. See the description of each function block for more details about it.



## 2.4. Function Blocks and Programs

In this chapter, the function blocks and programs that are part of the library are described, and for each one an example of use is reported.

### 2.4.1. List of the FBs and Progs

The **function blocks (FB)**, **functions (F)** present in the library are the following:

Type	Name	Description
FB	BACnet_PVUpdPr	Automate the Present Value update procedure of each BACnet Object type.
FB	BACnet_AI_SetPV	Use an IEC variable to set the Present Value of an Analog Input BACnet Object (supports INT, REAL).
FB	BACnet_RelDef_AV_link_iEE	Link an EEPROM parameter to an Analog Value BACnet Object (supports INT, UINT, REAL).
FB	BACnet_RelDef_AV_link_rEE	
FB	BACnet_RelDef_AV_link_uiEE	
FB	BACnet_BI_SetPV	Use an IEC variable to set the Present Value of a Binary Value BACnet Object (supports BOOL).
FB	BACnet_RelDef_BV_link_xEE	Link an EEPROM parameter to a Binary Value BACnet Object (supports BOOL).
FB	BACnet_MSI_diSetPV	Use an IEC variable to set the Present Value of a Multi State Input BACnet Object (supports DINT, UDINT, USINT).
FB	BACnet_MSI_udiSetPV	
FB	BACnet_MSI_usiSetPV	
FB	BACnet_RelDef_MV_link_iEE	Link an EEPROM parameter to a Multi State Value BACnet Object (supports INT, UINT, UDINT, USINT, REAL).
FB	BACnet_RelDef_MV_link_rEE	
FB	BACnet_RelDef_MV_link_udiEE	
FB	BACnet_RelDef_MV_link_uiEE	
FB	BACnet_RelDef_MV_link_usiEE	
F	BACnet_ObjID	Set the Object_Identifier of the Device BACnet Object using EEPROM parameters.
FB	BACnet_ObjID_De_store_EE	It stores in the EEPROM eventual changes to the Device Object_Identifier from the BACnet side.
F	BACnet_UdAsReal_read	Read and write an UDINT value as REAL.
F	BACnet_UdAsReal_write	
FB	BACnet_udiValue_link_rEE	Link a REAL EEPROM parameter to the value of an UDINT BACnet attribute.
F	BACnet_r_ROUND_di	It rounds a value properly in its conversion from REAL to DINT / INT / UDINT / UINT.
F	BACnet_r_ROUND_i	
F	BACnet_r_ROUND_udi	
F	BACnet_r_ROUND_ui	



### 2.4.2. BACnet\_PVUpdPr

This function block updates the `Present_Value` (PV) of all the supported BACnet Object Types: Analog Value (AV), Binary Value (BV), Multi State Value (MV), Calendar (Ca).

The update timeframe is defined by `uiPrd` (in tenths of seconds).

It can be chosen if the Object Types have to be updated synchronously (all at the same time) or asynchronously (alternatively, one per cycle), using the `BOOL xSync` input:

- If synchronously, all the Object Types will be updated at each `uiPrd`.
- If asynchronously, at each `uiPrd` defined timeframe, only one Object Type will be updated, in a rotational sequence; therefore, the Object Types will be all updated in a timeframe equal to four times the `uiPrd`.

You can decide to force the update of a specific Object Type at each `uiPrd`, overriding the asynchronous setting for that Object Type only, by setting the related `xForce...` input to `TRUE`.

#### Context of use

It is strongly suggested to use this function block in every project where the BACnet protocol is implemented, because otherwise the `Present_Value` will not be updated, unless this functionality is implemented in another part of the code.

#### Inputs

Name	Type	Description
<code>uiPrd</code>	UINT	Present Value update period in [sec/10]
<code>xSync</code>	BOOL	Defines if the update of the BACnet Object Types Present Value has to be synchronous or only one at each execution.
<code>xForceAVUpd</code>	BOOL	Forces the Analog Value Objects present value update at every run, if <code>TRUE</code> .
<code>xForceBVUpd</code>	BOOL	Forces the Binary Value Objects present value update at every run, if <code>TRUE</code> .
<code>xForceMVUpd</code>	BOOL	Forces the Multi State Value Objects present value update at every run, if <code>TRUE</code> .
<code>xForceCaUpd</code>	BOOL	Forces the Calendar Objects present value update at every run, if <code>TRUE</code> .

#### Outputs

Name	Type	Description
<code>wBACnetObjUpdStat</code>	WORD	Status of the BACnet Objects Present Value update. See the FB description for details.

#### Present Value Update Status

The output variable `wBACnetObjUpdStat` is a `WORD`, which should be read as described below:

- the bits 0-3 contain the info about the last execution;
- the bits 8-11 contain the info about the first run of the update.

The four bits of each category (0-3 and 8-11) are correlated to the Object Types as follows (the numbers refer to the bit number):

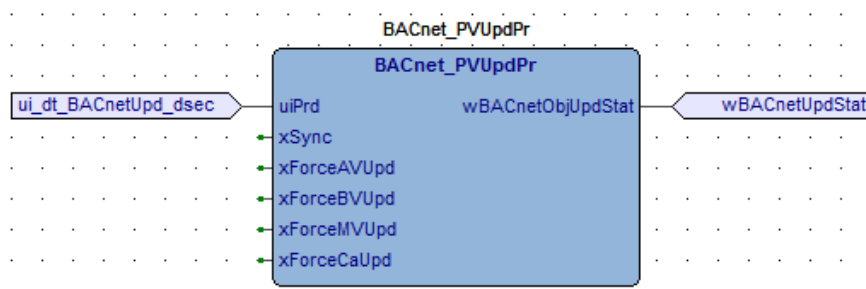
0. Analog Value;
1. Binary Value;
2. Multi State Value;
3. Calendar.



## FB Implementation Example

The following image shows an example of implementation of the `BACnet_PVUpdPr` FB.

In this case, no update of any Object Type is forced at each run and the update is asynchronous: at each run, which is executed every `ui_dt_BACnetUpd_dsec` tenths of seconds, only one Object Type is updated, in sequence.



### 2.4.3. BACnet\_AI\_SetPV

This function block sets the `Present_Value` of an Analog Input BACnet Object.

It takes as input the value to be written on the `Present_Value` and, using the passed BACnet Object pointer, it writes the value on the BACnet Object and gives back the `Present_Value` as output.

#### Context of use

Note that the logic value to be written on the `Present_Value` of an Analog Input BACnet Object can be of any kind, provided that the requirements regarding the data format are satisfied. E.g. it can be the value of a physical input, a physical Output, a generic IEC variable, etc.

The wording *Input* in the name of this BACnet Object underlines that this object accepts only inputs from the IEC and cannot write on the IEC environment; therefore, it is a BACnet object which can be only read.

#### Present Value Update

The present value update can be forced by setting `xLocalPV_Upd` to `TRUE`, which by default is set to `FALSE`.

In case this option is set to `FALSE` or not configured (as in the implementation example below), the Present Value update **must** be managed by using the `BACnet_PVUpdPr` function block.

#### Inputs

Name	Type	Description
xEn	BOOL	Used to set the visibility of the BACnet Object. If xEn = FALSE, It will set Out_Of_Service to TRUE, but iValue or rValue will still be written on the PV and passed to the rPV output.
ptrAI_BACnet	@BACNET_ANALOG_VALUE	Pointer to the Analog Input BACnet Object (use ADR function).
iValue	INT	INT value to be written in PV. If iValue = 32767, it is disabled and rValue is used to set the PV instead. It is multiplied by rScaler to calculate the PV.
rValue	REAL	REAL Value to be written in PV. Enabled only if iValue = 32767. It is multiplied by rScaler to calculate the PV.



rScaler	REAL	Scaling factor. Default is 1.0. It cannot be 0.0. NOTE: it is multiplied by the iValue or the rValue to determine the PV.
xLocalPV_Upd	BOOL	Forces the local PV Update of the interested BACnet variable. By default, it is set to FALSE.

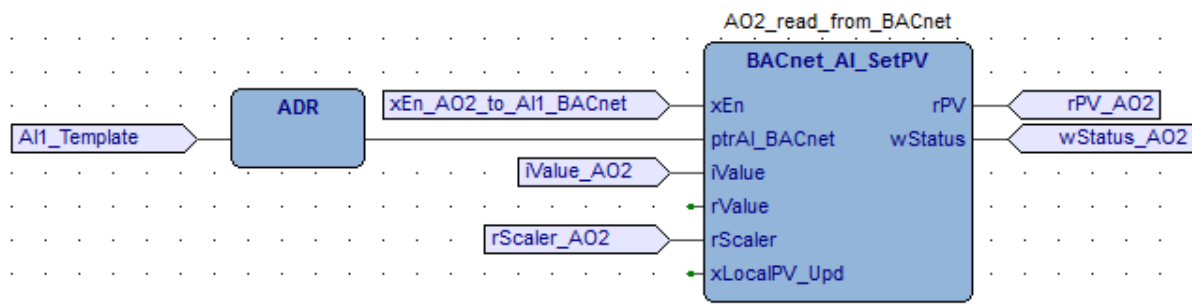
## Outputs

Name	Type	Description
rPV	REAL	It returns the Present Value.
wStatus	WORD	Bit 0: OK (IEC code operating properly). Bit 1: Out Of Service. Bit 2: Configuration Error.

## FB Implementation Example

The following image shows an example of implementation of the `BACnet_AI_SetPV` FB. The purpose is to write the INT value of the Analog Output 2 (AO2) into the BACnet Object `AI1_Template`.

In this way, the value of the AO2 will be available to be read by a supervisor, which can access it by reading the PV of the `AI1_Template`.



Note that also an AI can be made available through a BACnet AI Object. This is true as well for any kind of variable, whose value the user is interested to read from a supervisor.

### 2.4.4. *BACnet\_RelDef\_AV\_link\_\**EE

#### 2.4.4.1. *BACnet\_RelDef\_AV\_link\_iEE*

This function block allows to link an INT EEPROM parameter to a BACnet Analog Value Object. It can be used to implement BACnet in either new or existing projects, where an EEPROM parameter has to be kept aligned with a BACnet Analog Value.

The FB writes and reads the `Relinquish_Default` attribute of the BACnet Object to keep it synchronized with the EEPROM parameter, considered that it is scaled (multiplied by) using the `rScaler` and re-scaled back when the PV is given as output of this FB.

The main inputs of this FB are the BACnet Object address `ptrAV_BACnet`, the address of the EEPROM parameter `ptr_iEE` and the working mode `usiMode`.



## Working Mode

The FB operation depends on the `usiMode` value. It can be:

- 0: the EEPROM parameter is linked to the Relinquish Default and they get updated both ways (default);
- 1: this mode works as the mode 0, but in addition, when the EEPROM parameter is changed from the PLC side, the Priority Array is cleared (filled with *null*);
- 2: in this mode:
  - a change in PV updates the EEPROM parameter;
  - a change in the EEPROM parameter originating from the PLC side updates the Relinquish Default and the Priority Array is cleared (filled with *null*);

The **mode 2 does not require necessarily to show the Present Value and the EEPROM parameter on the HMI, but allows to show only the EEPROM parameter**, as the PV is always equal to the EE parameter.

**NOTE: only 100.000 value changes are guaranteed on each EEPROM location. Therefore, the expected reliability of an EEPROM parameter has to be determined also based on the frequency with which BACnet supervisors write diverse values on the Priority Array.**

See §4.1.1 for further details.

usiMode	EE updated on variation of:	an EE variation updates:	PrArr is cleared on EE PLC side variation:	# val to show on HMI
0	RelDef	RefDef	NO	2
1	RelDef	RefDef	YES, all	2
2	PV	RefDef	YES, all	1

**Note 1:** consider the effect of your choice when selecting the Mode. This has an impact on the prioritization between the BACnet supervisor and the PLC (when an EEPROM parameter is changed from the PLC/HMI).

### Note 2:

- If you use the FB *link* in **mode 2**, it is not required to connect the `iPV_scaled` output, as the EEPROM parameter will be always aligned with the PV and it is convenient to use directly the EEPROM parameter in the IEC code.
- If you use **mode 0** or **mode 1**, the `iPV_scaled` has to be used in the IEC code. In this case, note that if the *link* FB is called in a background task, it can be possible that a timed task using the `iPV_scaled` will receive a wrong value during the first executions, if the *link* FB instanced in the background task has not been executed yet for the first time. Therefore, if the *link* FB is called in a background task, it is important to initialize the global variable connected to the `iPV_scaled` output in an init task, by setting it equal to the EEPROM parameter. If the *link* FB is called in a timed task instead, this initialization is not required, but it is important to run that task before the other timed tasks that use the `iPV_scaled`.





### Priority Array Clear option On Demand

By changing the `xPrArrClear_OD` input from FALSE to TRUE, the Priority Array is cleared (filled with *null* values). This is a way to regain control over BACnet when requested. This can be useful in case the mode 0 is used and the user wants to restore the HMI local control.

The input has to be manually reset to FALSE after each use of this functionality.

### Present Value Update

The present value update can be forced by setting `xLocalPV_Upd` to TRUE, which by default is set to FALSE.

In case this option is set to FALSE or not configured (as in the implementation example below), the Present Value update **must** be managed by using the [BACnet\\_PVUpdPr](#) function block.

### Out-of-range writes and Reliability attribute

The inputs `iRelMin` and `iRelMax` allow to limit the value that the EEPROM parameter can assume and will be used to saturate the `Relinquish` in case the limits are exceeded. This avoids out-of-range writes from the supervisor side.

- In case of an out-of-range write from the supervisor on `Relinquish`, this write will not be accepted (the EEPROM does not get updated with the written value) and the value which is present in the EEPROM will be restored on the `Relinquish`. The `Reliability` attribute does not get changed in this case by the IEC code.
- In case of an out-of-range write from the supervisor on the `Priority_Array`, which will cause an out-of-range `Present_Value`, the output of the FB (`iPV_scaled`) will be saturated using the **scaled** `iRelMin` (if lower than it) or the `iRelMax` (if higher than it) and the `Reliability` attribute of the BACnet Object will be set to `RELIABILITY_UNDER_RANGE` in the former case and to `RELIABILITY_OVER_RANGE` in the latter one.

The `Reliability` is restored to `RELIABILITY_NO_FAULT_DETECTED` once the `Present_Value` gets back into the limits.

Note that `iRelMin` and `iRelMax` are both scaled by multiplying them by the `rScaler` before using them for the comparisons.

### Inputs

Name	Type	Description
<code>xEn</code>	BOOL	Used to set the visibility of the BACnet Object. If <code>xEn</code> = FALSE, It will set <code>Out_Of_Service</code> to TRUE, but <code>iEE</code> will still be written on the <code>RelDef</code> and passed to the <code>iPV_scaled</code> output.
<code>ptrAV_BACnet</code>	@BACNET_ANALOG_VALUE	Analog Value Object Address (use ADR function)
<code>ptr_IEE</code>	@INT	EEPROM Parameter Address (use ADR function)
<code>iRelMin</code>	INT	Lower limit for Relinquish Default in INT. It is scaled by multiplying it by the <code>rScaler</code> .
<code>iRelMax</code>	INT	Higher limit for Relinquish Default in INT. It is scaled by multiplying it by the <code>rScaler</code> .
<code>rScaler</code>	REAL	Scaling factor. Default is 1.0. It cannot be 0.0.
<code>xLocalPV_Upd</code>	BOOL	Forces the local PV Update of the interested BACnet variable. By default, it is set to FALSE.



usiMode	USINT	Mode 0: EE <-> RelDef. Mode 1: as 0 + Clear PrArr On Change from PLC. Mode 2: PV -> EE, EE -> RelDef + Clear PrArr On Change from PLC (max 100.000 writes on EE).
xPrArrClear_OD	BOOL	PrArr Clear On Demand. Only on transition FALSE -> TRUE, it clears the Priority Array. It requires an external reset to FALSE.

## Outputs

Name	Type	Description
iPV_scaled	INT	Scaled INT Present Value
wStatus	WORD	Bit 0: OK (IEC code operating properly). Bit 1: Out Of Service. Bit 2: Configuration Error. Bit 4: Priority Active (override from supervisor).
xPriorityActive	BOOL	It takes trace of eventual overrides from the Supervisor (which does it by writing in the PriorityArray).

## FB Implementation Example

The following image shows an example of implementation of the BACnet\_RelDef\_AV\_link\_iEE FB.

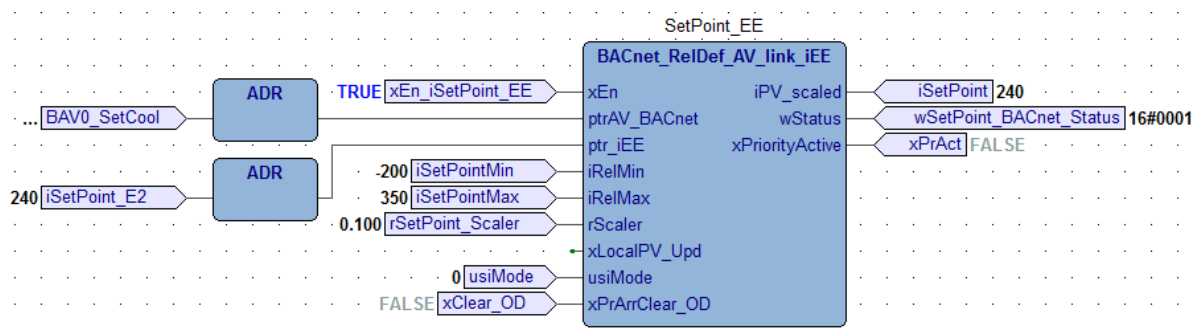
The purpose is to implement the BACnet protocol in an **existing project** where the setpoint parameter, which is in EEPROM, has been renamed from iSetPoint to iSetPoint\_E2 and is now passed to the EEPROM pointer using the ADR function.

A new global variable has been created using the old name of the EEPROM parameter, which was iSetPoint, and it is used as output of this function block, so that it will keep its value updated.

Therefore, all the instances of the IEC code where the old EEPROM parameter iSetPoint was used will not need to be changed, as the new global variable has the same name the EEPROM parameter had.

The working mode usiMode has to be selected according to how the developer wants to manage the priorities between the PLC side and the BACnet side. Have a look at the *Working Mode* paragraph reported above for further details.

In this example, xLocalPV\_Upd is not connected because for the PV update the BACnet\_PVUpdPr function block is used. In case BACnet\_PVUpdPr is not used, you should connect a variable to xLocalPV\_Upd and set it to TRUE.





#### 2.4.4.2. *BACnet\_RelDef\_AV\_link\_rEE*

This function block allows to link a REAL EEPROM parameter to a BACnet Analog Value Object.

It operates as the *BACnet\_RelDef\_AV\_link\_iEE* function block, with the exception that the pointed EEPROM parameter and the PV output will be of REAL data type instead of INT. Refer to the *BACnet\_RelDef\_AV\_link\_iEE* chapter for information about its functionalities and how to implement it.

#### 2.4.4.3. *BACnet\_RelDef\_AV\_link\_uiEE*

This function block allows to link an UINT EEPROM parameter to a BACnet Analog Value Object.

It operates as the *BACnet\_RelDef\_AV\_link\_iEE* function block, with the exception that the pointed EEPROM parameter and the PV output will be of UINT data type instead of INT. Refer to the *BACnet\_RelDef\_AV\_link\_iEE* chapter for information about its functionalities and how to implement it.

#### 2.4.5. *BACnet\_BI\_SetPV*

This function block sets the *Present\_Value* of a Binary Input BACnet Object.

It takes as input the value to be written on the *Present\_Value* and, using the passed BACnet Object pointer, it writes the value on the BACnet Object *Priority\_Array[15]*. The output is equal to the input passed to the FB and does not come from the BACnet Object attributes.

**Note:** by setting the *xClearAlways* input to TRUE, your application may result being not 100% compliant with the BTL standard.

##### Context of use

Note that the value to be written on the *Present\_Value* of a Binary Input BACnet Object can be of any kind, provided that the requirements regarding the data format are satisfied. E.g. it can be a Digital Input, a Digital Output, a generic Boolean IEC variable, etc.

The wording *Input* in the name of this object underlines that this object accepts only inputs from the IEC and cannot write on the IEC environment; therefore, it is a BACnet object which can only read.

##### Present Value Update

The present value update can be forced by setting *xLocalPV\_Upd* to TRUE, which by default is set to FALSE.

In case this option is set to FALSE or not configured (as in the implementation example below), the Present Value update **must** be managed by using the **BACnet\_PVUpdPr** function block.



## Inputs

Name	Type	Description
xEn	BOOL	Used to set the visibility of the BACnet Object. If xEn = FALSE, It will set Out_Of_Service to TRUE, but xValue will still be written on the Priority_Array[15] and passed to the xPV_current output.
ptrBI_BACnet	@BACNET_BINARY_VALUE	Binary Value Object Address (use ADR function)
xValue	BOOL	EEPROM Parameter Address (use ADR function)
xLocalPV_Upd	BOOL	Forces the local PV Update of the interested BACnet variable. By default, it is set to FALSE.
xClearAlways	BOOL	If TRUE, the PrArr is cleared at each execution, if required, in order for the Present_Value to reflect the Input Value. Default if FALSE.

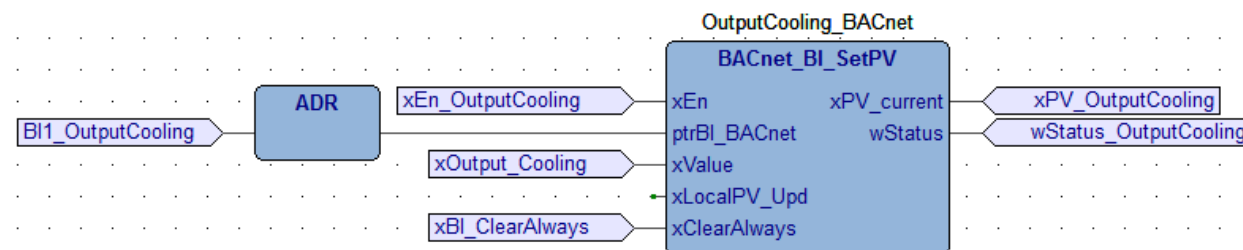
## Outputs

Name	Type	Description
xPV_current	BOOL	BOOL Present Value
wStatus	WORD	Bit 0: OK (IEC code operating properly). Bit 1: Out Of Service. Bit 2: Configuration Error. Bit 4: Priority Active (override from supervisor).

## FB Implementation Example

The following image shows an example of implementation of the BACnet\_BI\_SetPV FB. The purpose is to write the BOOL value of the xOutput\_Cooling variable into the BACnet Object BI1\_OutputCooling.

In this way, the value of the xOutput\_Cooling will be available to be read by a supervisor, which can access it by reading the PV of the BI1\_OutputCooling.





#### 2.4.6. BACnet\_RelDef\_BV\_link\_xEE

This function block allows to link a BOOL EEPROM parameter to a BACnet Binary Value Object. It can be used to implement BACnet in either new or existing projects, where an EEPROM parameter has to be kept aligned with a BACnet Analog Value.

The FB writes and reads the `Relinquish_Default` attribute of the BACnet Object to keep it synchronized with the EEPROM parameter.

The main inputs of this FB are the BACnet Object address `ptrBV_BACnet`, the address of the EEPROM parameter `ptr_xEE` and the working mode `usiMode`.

##### Working Mode

The FB operation depends on the `usiMode` value. It can be:

- 0: the EEPROM parameter is linked to the Relinquish Default and they get updated both ways (default);
- 1: this mode works as the mode 0, but in addition, when the EEPROM parameter is changed from the PLC side, the Priority Array is cleared (filled with *null*);
- 2: in this mode:
  - a change in PV updates the EEPROM parameter;
  - a change in the EEPROM parameter originating from the PLC side updates the Relinquish Default and the Priority Array is cleared (filled with *null*);

The **mode 2 does not require necessarily to show the Present Value and the EEPROM parameter on the HMI, but allows to show only the EEPROM parameter**, as the PV is always equal to the EE parameter.

**NOTE: only 100.000 value changes are guaranteed on each EEPROM location. Therefore, the expected reliability of an EEPROM parameter has to be determined also based on the frequency with which BACnet supervisors write diverse values on the Priority Array.**

See §4.1.1 for further details.

usiMode	EE updated on variation of:	an EE variation updates:	PrArr is cleared on EE PLC side variation:	# val to show on HMI
0	RelDef	RefDef	NO	2
1	RelDef	RefDef	YES, all	2
2	PV	RefDef	YES, all	1

**Note 1:** consider the effect of your choice when selecting the Mode. This has an impact on the prioritization between the BACnet supervisor and the PLC (when an EEPROM parameter is changed from the PLC/HMI).

##### Note 2:

- If you use the FB *link* in **mode 2**, it is not required to connect the `xPV_current` output, as the EEPROM parameter will be always aligned with the PV and it is convenient to use directly the EEPROM parameter in the IEC code.



- If you use **mode 0** or **mode 1**, the xPV\_current has to be used in the IEC code. In this case, note that if the *link* FB is called in a background task, it can be possible that a timed task using the xPV\_current will receive a wrong value during the first executions, if the *link* FB instanced in the background task has not been executed yet for the first time. Therefore, if the *link* FB is called in a background task, it is important to initialize the global variable connected to the xPV\_current output in an init task, by setting it equal to the EEPROM parameter. If the *link* FB is called in a timed task instead, this initialization is not required, but it is important to run that task before the other timed tasks that use the xPV\_current.

### Priority Array Clear option On Demand

By changing the xPrArrClear\_OD input from FALSE to TRUE, the Priority Array is cleared (filled with *null* values). This is a way to regain control over BACnet when requested. This can be useful in case the mode 0 is used and the user wants to restore the HMI local control.

The input has to be manually reset to FALSE after each use of this functionality.

### Present Value Update

The present value update can be forced by setting xLocalPV\_Upd to TRUE, which by default is set to FALSE.

In case this option is set to FALSE or not configured (as in the implementation example below).

### Inputs

Name	Type	Description
xEn	BOOL	Used to set the visibility of the BACnet Object. If xEn = FALSE, It will set Out_Of_Service to TRUE, but xEE will still be written on the RelDef and passed to the xPV_current output.
ptrBV_BACnet	@BACNET_BINARY_VALUE	Binary Value Object Address (use ADR function)
ptr_xEE	@BOOL	EEPROM Parameter Address (use ADR function)
xLocalPV_Upd	BOOL	Forces the local PV Update of the interested BACnet variable. By default, it is set to FALSE.
usiMode	USINT	Mode 0: EE <-> RelDef. Mode 1: as 0 + Clear PrArr On Change from PLC. Mode 2: PV -> EE, EE -> RelDef + Clear PrArr On Change from PLC (max 100.000 writes on EE).
xPrArrClear_OD	BOOL	PrArr Clear On Demand. Only on transition FALSE -> TRUE, it clears the Priority Array. It requires an external reset to FALSE.

### Outputs

Name	Type	Description
xPV_current	BOOL	BOOL Present Value
wStatus	WORD	Bit 0: OK (IEC code operating properly). Bit 1: Out Of Service. Bit 2: Configuration Error. Bit 4: Priority Active (override from supervisor).
xPriorityActive	BOOL	It takes trace of eventual overrides from the Supervisor (which does it by writing in the PriorityArray).



## FB Implementation Example

The following image shows an example of implementation of the BACnet\_RelDef\_BV\_link\_xEE FB.

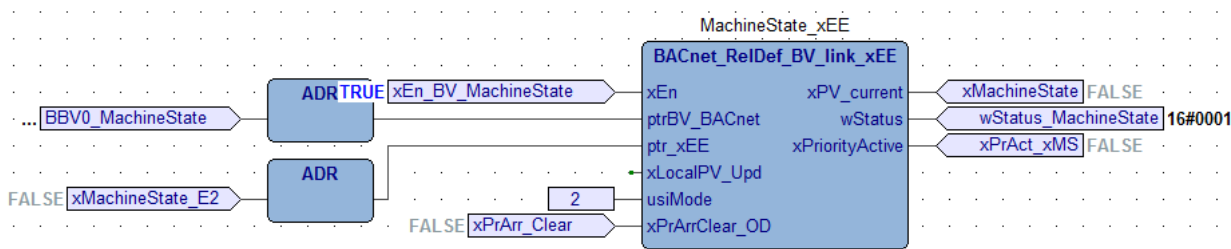
The purpose is to implement the BACnet protocol in an **existing project** where the machine state, which is in EEPROM, has been renamed from xMachineState to xMachineState\_E2 and is now passed to the pointer to the EEPROM using the ADR function.

A new global variable has been created using the same name of the old EEPROM parameter, which was xMachineState and it is used as output of this function block, so that it will keep its value updated.

Therefore, all the instances of the IEC code where the old EEPROM parameter xMachineState was used will not need to be changed, as the new global variable has the same name the EEPROM parameter had.

The working mode `usiMode` has to be selected according to how the developer wants to manage the priorities between the PLC side and the BACnet side. Have a look at the *Working Mode* paragraph reported above for further details.

In this example, `xLocalPV_Upd` is not connected because for the PV update the BACnet\_PVUpdPr function block is used. In case BACnet\_PVUpdPr is not used, you should connect a variable to `xLocalPV_Upd` and set it to TRUE.



## 2.4.7. BACnet\_MSI\_\*SetPV

### 2.4.7.1. BACnet\_MSI\_diSetPV

This function block sets the `Present_Value` of a Multi State Input BACnet Object (DINT input)..

It takes as input `diValue`, which is the value that, scaled by the `iOffset`, has to be written on the `Present_Value` of the BACnet Object passed through a pointer. The input is a DINT value. **Check the range limitations given by this setting before using this variant of the MSI SetPV FBs Family.**

It gives back the `Present_Value` as output.

#### Context of use

The wording *Input* in the name of this object underlines that this object accepts only inputs from the IEC and cannot write on the IEC environment; therefore, it is a BACnet object which can only read.

#### Present Value Update

The present value update can be forced by setting `xLocalPV_Upd` to TRUE, which by default is set to FALSE.





In case this option is set to FALSE or not configured (as in the implementation example below), the Present Value update **must** be managed by using the **BACnet\_PVUpdPr** function block.

### Inputs

Name	Type	Description
xEn	BOOL	Used to set the visibility of the BACnet Object. If xEn = FALSE, It will set Out_Of_Service to TRUE, but diValue will still be written on the PV and passed to the udiPV output.
ptrMSI_BACnet	@BACNET_MULTI_STATE_VALUE	Pointer to the Multi State Input BACnet Object (use ADR function).
diValue	DINT	DINT Value to be written in PV, scaled by summing the iOffset to it. Its value must be such that the sum diValue + iOffset is in the range 1... Number_Of_States.
iOffset	INT	INT offset that is summed to the input value.
iWrongInputValue	INT	Value that is written on the Present Value in case of wrong input value (outside the allowed range of the object). Its value MUST be in the range 1 ... Number_Of_States.
xLocalPV_Upd	BOOL	Forces the local PV Update of the interested BACnet variable. By default, it is set to FALSE.

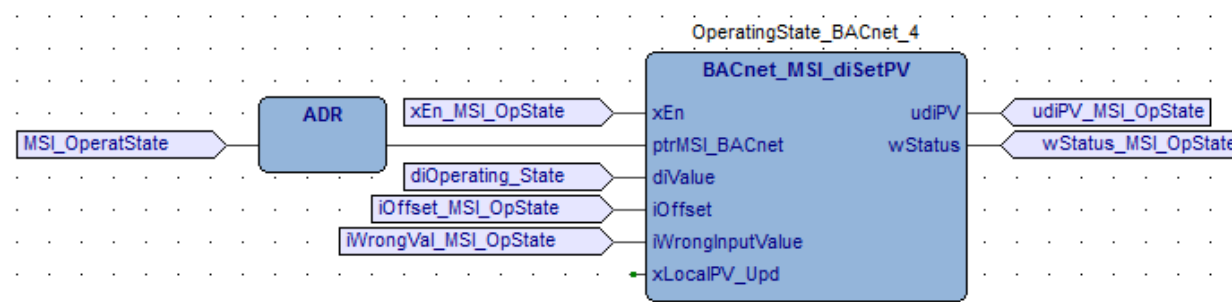
### Outputs

Name	Type	Description
udiPV	UDINT	Returns the Present Value of the MSI BACnet Object.
wStatus	WORD	Bit 0: OK (IEC code operating properly). Bit 1: Out Of Service. Bit 2: Configuration Error. Bit 3: BACnet not aligned with input value.

### FB Implementation Example

The following image shows an example of implementation of the **BACnet\_MSI\_diSetPV** FB. The purpose is to write the DINT value of the **diOperating\_State** status variable into the BACnet Object **MSI\_OperatingState**, which is an Enumerative.

In this way, it is possible to read the **diOperating\_State** of the machine from a supervisor, by reading the PV of the **MSI\_OperatingState**.





#### 2.4.7.2. BACnet\_MSI\_udiSetPV

This function block sets the `Present_Value` of a Multi State Input BACnet Object (UDINT input).

**Check the range limitations given by this setting before using this variant of the MSI SetPV FBs Family.**

It operates as the `BACnet_MSI_diSetPV` function block, with the exception that the input will be of UDINT data type instead of DINT. Refer to the `BACnet_MSI_diSetPV` chapter for information about its functionalities and how to implement it.

#### 2.4.7.3. BACnet\_MSI\_usiSetPV

This function block sets the `Present_Value` of a Multi State Input BACnet Object (USINT input).

**Check the range limitations given by this setting before using this variant of the MSI SetPV FBs Family.**

It operates as the `BACnet_MSI_diSetPV` function block, with the exception that the input will be of USINT data type instead of DINT. Refer to the `BACnet_MSI_diSetPV` chapter for information about its functionalities and how to implement it.

#### 2.4.8. BACnet\_RelDef\_MV\_link\_\*EE

##### 2.4.8.1. BACnet\_RelDef\_MV\_link\_iEE

This function block allows to link an INT EEPROM parameter to a BACnet Multi State Value Object.

It can be used to implement BACnet in either new or existing projects, where an EEPROM parameter has to be kept aligned with a BACnet Multi State Value.

The FB writes and reads the `Relinquish_Default` attribute of the BACnet Object to keep it synchronized with the EEPROM parameter, considered that it is scaled (summed to) using the `iOffset` and re-scaled back when the PV is given as output of this FB.

The main inputs of this FB are the BACnet Object address `ptrMV_BACnet`, the address of the EEPROM parameter `ptr_iEE`, the offset (`iOffset`) and the value to be written on the PV in case of wrong input value (`iWrongLinkValue`).

**Check the range limitations given by this setting before using this variant of the MSI SetPV FBs Family. A TO\_UDINT conversion takes place before writing the INT number on the Relinquish.**

#### Working Mode

The FB operation depends on the `usiMode` value. It can be:

- 0: the EEPROM parameter is linked to the Relinquish Default and they get updated both ways (default);
- 1: this mode works as the mode 0, but in addition, when the EEPROM parameter is changed from the PLC side, the Priority Array is cleared (filled with *null*);
- 2: in this mode:
  - a change in PV updates the EEPROM parameter;
  - a change in the EEPROM parameter originating from the PLC side updates the Relinquish Default and the Priority Array is cleared (filled with *null*);



The **mode 2** does not require necessarily to show the Present Value and the EEPROM parameter on the HMI, but allows to show only the EEPROM parameter, as the PV is always equal to the EE parameter.

**NOTE: only 100.000 value changes are guaranteed on each EEPROM location.** Therefore, the expected reliability of an EEPROM parameter has to be determined also based on the frequency with which BACnet supervisors write diverse values on the Priority Array.

See §4.1.1 for further details.

usiMode	EE updated on variation of:	an EE variation updates:	PrArr is cleared on EE PLC side variation:	# val to show on HMI
0	RelDef	RefDef	NO	2
1	RelDef	RefDef	YES, all	2
2	PV	RefDef	YES, all	1

**Note 1:** consider the effect of your choice when selecting the Mode. This has an impact on the prioritization between the BACnet supervisor and the PLC (when an EEPROM parameter is changed from the PLC/HMI).

**Note 2:**

- If you use the FB *link* in **mode 2**, it is not required to connect the *iPV\_current* output, as the EEPROM parameter will be always aligned with the PV and it is convenient to use directly the EEPROM parameter in the IEC code.
- If you use **mode 0** or **mode 1**, the *iPV\_current* has to be used in the IEC code. In this case, note that if the *link* FB is called in a background task, it can be possible that a timed task using the *iPV\_current* will receive a wrong value during the first executions, if the *link* FB instanced in the background task has not been executed yet for the first time. Therefore, if the *link* FB is called in a background task, it is important to initialize the global variable connected to the *iPV\_current* output in an init task, by setting it equal to the EEPROM parameter. If the *link* FB is called in a timed task instead, this initialization is not required, but it is important to run that task before the other timed tasks that use the *iPV\_current*.

### Priority Array Clear option On Demand

By changing the *xPrArrClear\_OD* input from FALSE to TRUE, the Priority Array is cleared (filled with *null* values). This is a way to regain control over BACnet when requested. This can be useful in case the mode 0 is used and the user wants to restore the HMI local control.

The input has to be manually reset to FALSE after each use of this functionality.

### Present Value Update

The present value update can be forced by setting *xLocalPV\_Upd* to TRUE, which by default is set to FALSE.

In case this option is set to FALSE or not configured (as in the implementation example below), the Present Value update **must** be managed by using the **BACnet\_PVUpdPr** function block.



### Out-of-range writes and Reliability attribute

The value to be written on the PV must be in the range [1 ... Number\_Of\_States]. It is calculated by the summing the value in EEPROM to the iOffset.

In case it is outside the allowed range, the iWrongLinkValue is written in place of it and the Reliability attribute is set to RELIABILITY\_UNRELIABLE\_OTHER.

The Reliability is restored to RELIABILITY\_NO\_FAULT\_DETECTED once the value to be written gets back into the limits.

Note that iWrongLinkValue must be in the range [1 ... Number\_Of\_States].

### Inputs

Name	Type	Description
xEn	BOOL	Used to set the visibility of the BACnet Object. If xEn = FALSE, It will set Out_Of_Service to TRUE, but iEE will still be written on the RelDef and passed to the iPV_current output.
ptrMV_BACnet	@BACNET_MULTI_STATE_VALUE	Multi State Value Object Address (use ADR function)
ptr_iEE	@INT	EEPROM Parameter Address (use ADR function)
iOffset	INT	INT Offset that is summed to the EEPROM value.
iWrongLinkValue	INT	Value that is written on the Present Value in case of wrong input value (outside the allowed range of the object). Its value MUST be in the range 1 ... Number_Of_States.
xLocalPV_Upd	BOOL	Forces the local PV Update of the interested BACnet variable. By default, it is set to FALSE.
usiMode	USINT	Mode 0: EE <-> RelDef. Mode 1: as 0 + Clear PrArr On Change from PLC. Mode 2: PV -> EE, EE -> RelDef + Clear PrArr On Change from PLC (max 100.000 writes on EE).
xPrArrClear_OD	BOOL	PrArr Clear On Demand. Only on transition FALSE -> TRUE, it clears the Priority Array. It requires an external reset to FALSE.

### Outputs

Name	Type	Description
iPV_current	INT	INT Present Value
wStatus	WORD	Bit 0: OK (IEC code operating properly). Bit 1: Out Of Service. Bit 2: Configuration Error. Bit 3: BACnet not aligned with input value. Bit 4: Priority Active (override from supervisor).
xPriorityActive	BOOL	It takes trace of eventual overrides from the Supervisor (which does it by writing in the PriorityArray).

### FB Implementation Example

The following image shows an example of implementation of the BACnet\_RelDef\_MV\_link\_iEE FB.

The purpose is to implement the BACnet protocol in an existing project where the operating mode of the machine is stored in an EEPROM parameter. This last one has been renamed from iSeasonMode to iSeasonMode\_E2 and is now passed to the pointer to the EEPROM using the ADR function block. The iSeasonMode variable is an Enumerative.



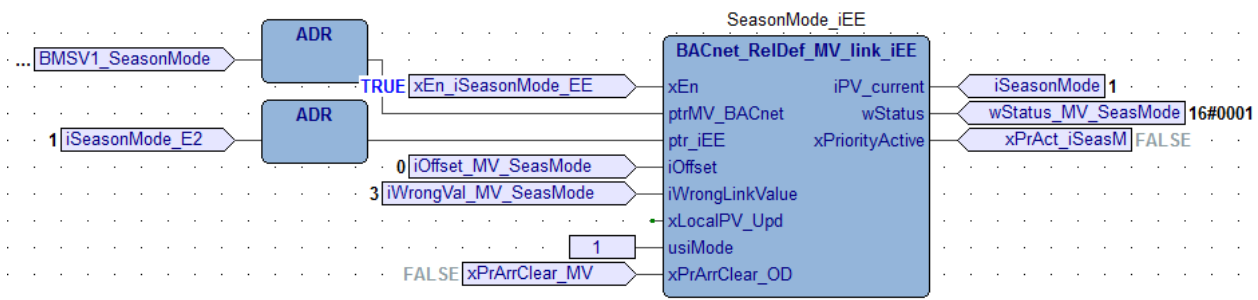
A new global variable has been created using the same name of the old EEPROM parameter, which was `iSeasonMode` and it is used as output of this function block, so that it will keep its value updated.

Therefore, all the instances of the IEC code where the old EEPROM parameter `iSeasonMode` was used will not need to be changed, as the new global variable has the same name the EEPROM parameter had.

In this example, the value which will be written in case of wrong inputs (`iWrongLinkValue`) is 2, which corresponds to the “Auto” setting of the Operating Mode.

The working mode `usiMode` has to be selected according to how the developer wants to manage the priorities between the PLC side and the BACnet side. Have a look at the *Working Mode* paragraph reported above for further details.

In this example, `xLocalPV_Upd` is not connected because the `BACnet_PVUpdPr` function block is used for the PV update. In case `BACnet_PVUpdPr` is not used, you should connect a variable to `xLocalPV_Upd` and set it to TRUE.



#### 2.4.8.2. BACnet\_RelDef\_MV\_link\_rEE

This function block allows to link a REAL EEPROM parameter to a BACnet Multi State Value Object.

**Check the range limitations given by this setting before using this variant of the MSI SetPV FBs Family. A TO UDINT conversion takes place before writing the REAL number on the Relinquish.**

It operates as the `BACnet_RelDef_MV_link_iEE` function block, with the exception that the pointed EEPROM parameter and the PV output will be of REAL data type instead of INT. Refer to the `BACnet_RelDef_MV_link_iEE` chapter for information about its functionalities and how to implement it.

#### 2.4.8.3. BACnet\_RelDef\_MV\_link\_udiEE

This function block allows to link an UDINT EEPROM parameter to a BACnet Multi State Value Object.

It operates as the `BACnet_RelDef_MV_link_iEE` function block, with the exception that the pointed EEPROM parameter and the PV output will be of UDINT data type instead of INT. Refer to the `BACnet_RelDef_MV_link_iEE` chapter for information about its functionalities and how to implement it.



#### 2.4.8.4. *BACnet\_RelDef\_MV\_link\_uiEE*

This function block allows to link a UINT EEPROM parameter to a BACnet Multi State Value Object.

It operates as the *BACnet\_RelDef\_MV\_link\_iEE* function block, with the exception that the pointed EEPROM parameter and the PV output will be of UINT data type instead of INT. Refer to the *BACnet\_RelDef\_MV\_link\_iEE* chapter for information about its functionalities and how to implement it.

#### 2.4.8.5. *BACnet\_RelDef\_MV\_link\_usiEE*

This function block allows to link a USINT EEPROM parameter to a BACnet Multi State Value Object.

It operates as the *BACnet\_RelDef\_MV\_link\_iEE* function block, with the exception that the pointed EEPROM parameter and the PV output will be of USINT data type instead of INT. Refer to the *BACnet\_RelDef\_MV\_link\_iEE* chapter for information about its functionalities and how to implement it.

#### 2.4.9. *BACnet\_ObjID*

This function calculates the BACnet Object Identifier given the BACnet node number and subnet value.

It is suggested to store the two parameters required by this function in the EEPROM of the device.

This function has to be called in the Init task in order to restore the Device Object Identifier after each device restart.

In this way, the BACnet Object Identifier will be kept always the same, even after a restart.

An implementation example of this function is provided in the §3 BACnet Application Project Sample.

##### Inputs

Name	Type	Description
uiBACnet_ID	UINT	BACnet ID value
usiBACnet_Subnet	USINT	BACnet Subnet value

##### Outputs

Name	Type	Description
BACnet_ObjID	UDINT	BACnet Object Identifier

#### 2.4.10. *BACnet\_ObjID\_De\_store\_EE*

This function block enables two functionalities:

1. Update the EEPROM parameters containing the Device ID and Device Subnet, which together define the BACnet Device Object ID, after a change from the supervisor side.
2. Detect a local change of the Device ID and Device Subnet EEPROM parameters and provide the info that a reboot is required for changes to take effect.

It is supposed to be used together with the *BACnet\_ObjID* function, that will have to be called into a program assigned to the `Init` task.



## Inputs

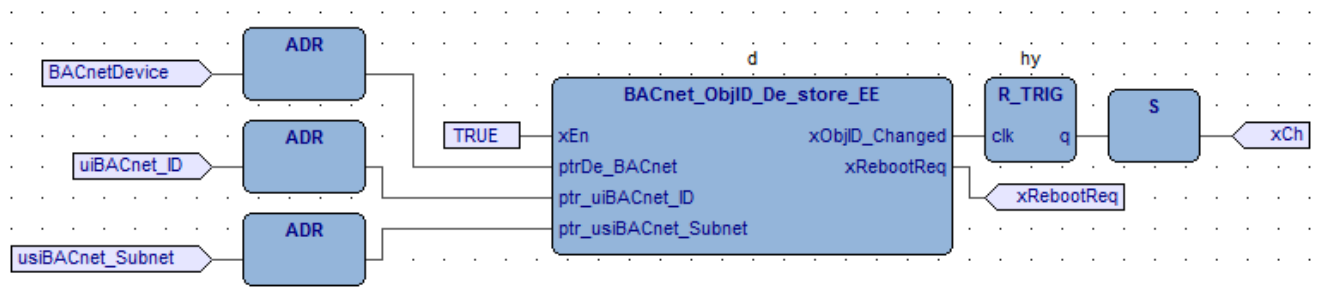
Name	Type	Description
xEn	BOOL	Used to enable the functionalities of the FB
ptrDe_BACnet	@BACNET_DEVICE	Device Object Address (use ADR function)
ptr_uiBACnet_ID	@UINT	BACnet_ID EEPROM Parameter Address (use ADR function)
ptr_usiBACnet_Subnet	@USINT	BACnet_Subnet EEPROM Parameter Address (use ADR function)

## Outputs

Name	Type	Description
xObjID_Changed	BOOL	BOOL that states if the ObjID has been changed from the BACnet side
xRebootReq	BOOL	BOOL that states if a reboot is required after an EEPROM parameters change from the PLC side

## FB Implementation Example

The following image shows an example of implementation of the BACnet\_ObjID\_De\_store\_EE FB.



### 2.4.11. BACnet\_UdAsReal\_read

This function allows you to read a REAL value which is stored as UDINT in a BACnet attribute.

Passing the pointer to the UDINT variable/attribute where a REAL value has been written instead, it performs the data format equivalence and gives back as output the REAL value that has originally been written on the variable/attribute.

## Inputs

Name	Type	Description
ptr_udi	@UDINT	Pointer to the BACnet attribute that has to be read.

## Outputs

Name	Type	Description
BACnet_UdAsReal_read	REAL	Original REAL value.





### 2.4.12. BACnet\_UdAsReal\_write

This function allows you to write a REAL value which is stored as UDINT in a BACnet attribute.

Passing the pointer to the UDINT variable/attribute where a REAL value has to be written instead, it performs the data format equivalence and writes the rValue. If the BACnet attribute is set to be in EEPROM, xE2 must be TRUE.

The returned value is the attribute value in REAL.

#### Input

Name	Type	Description
ptr_udi	@UDINT	Pointer to the BACnet attribute.
rValue	REAL	Value to be written.
xE2	BOOL	Used to define if the BACnet attribute is in EEPROM or not. TRUE if it is, FALSE if it is not.

#### Outputs

Name	Type	Description
BACnet_UdAsReal_write	REAL	Value which is stored in the UDINT attribute.



### 2.4.13. BACnet\_udiValue\_link\_rEE

This function block allows to link a REAL EEPROM parameter to the value of an UDINT BACnet attribute.

It can be used to implement BACnet in either new or existing projects, where you would like to keep a BACnet UDINT attribute aligned with a REAL EEPROM parameter.

The FB writes and reads the `attribute` value of a BACnet Object to keep it synchronized with the EEPROM parameter. No scaler is used.

The main inputs of this FB are:

- The UDINT value address `ptrBACnet_udiAttribute`. This is not the address of the BACnet Object, but the address of a specific attribute of it.
- The address of the EEPROM parameter `ptr_rEE`.

This FB can either be called in background or in timed, depending on the requirements of your application. Note that until its first execution, the value of the attribute (e.g. a Schedule Default) will not be correctly aligned. Therefore, in case you use the attribute in a timed task it is important either:

- to call an instance of the FB also in the Init task, when it is called in a background task;
- if the FB is called in a timed task, this has to be set before the other timed tasks which use the linked attribute.

#### Out-of-range writes

The inputs `rRelMin` and `rRelMax` allow to limit the value that the EEPROM parameter can assume and will be used to saturate the `attribute` value in case the limits are exceeded. This avoids out-of-range writes from the supervisor side.

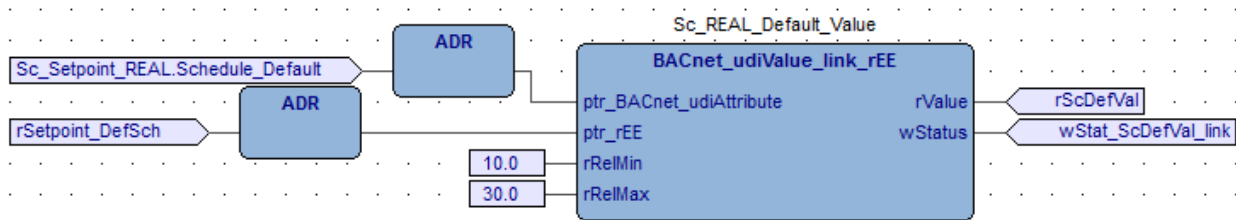
In case of an out-of-range write from the supervisor on the `attribute` value, this write will not be accepted (the EEPROM does not get updated with the written value) and the value which is present in the EEPROM will be restored on the `attribute` value.

#### FB Implementation Example

The following image shows an example of implementation of the `BACnet_udiValue_link_rEE` FB.

The purpose is to link the `rSetpoint_DefSch` EEPROM parameter to the `Schedule_Default` value of the `Sc_Setpoint_REAL_Schedule` Schedule BACnet Object.

**Note that this function block can be used for every type of BACnet Object.**





## BACnet Application Project Sample

### 3.1. Overview

The current chapter guides the user through the implementation of the BACnet protocol into a sample project, by using the BACnet Addons library.

An existing application will be used for this scope. It is about a simple control of a machine with a thermostat using a regulation based on hysteresis.

### 3.2. Sample thermostat application

The baseline application is the existing one, where the BACnet protocol is not yet implemented at all. In this paragraph an overview of the its contents is carried out.

#### 3.2.1. *hysteresis\_ST*

This function block contains the setpoint control based on hysteresis.

The inputs are the measured ambient temperature, the setpoint value and the differentiation.

A check for the probe disconnection is also implemented.

#### 3.2.2. *Thermostat*

The Thermostat program executes the *hysteresis\_ST* function block. It passed the temperature to it based on the AI1 and on the state of the machine (if the machine is set to *off*, an override temperature will be passed).

#### 3.2.3. *usiOperating\_State*

The *usiOperating\_State* IEC variable represents the operating state of the machine while it is on, which could be either active ("Running"), in standby ("Idle"), in defrost ("Defrost") or in a secure operational status after a certain alarm has been raised ("Safety").

This is normally an output of the control algorithm on the IEC side and should not be changed manually.

Just for the purpose of testing, in this project sample it has been chosen to make it editable from the HMI, so that you could see the effect of its changes on the BACnet side.

#### 3.2.4. *Other status variables and EEPROM parameters*

The *usiSeasonMode* represents the working mode based on the season (Summer, Winter, Auto). It is just an EEPROM parameter used to show how a MSV BACnet object could be setup.

The *udiOperating\_Mode* sets the working mode based on the room occupation (Normal, Comfort, Not occupied). It is just a status variable used to show what could be the output of a Schedule BACnet Object.



The *xNonWorkingDay* gives a feedback about whether a day is a working day or not. It will be used just to show how a Calendar BACnet Object could be used.

### 3.3. BACnet Objects Templates

Together with the library, 12 templates of BACnet Objects are attached. You will find already defined attributes, based on what is of common use for that Object. The use of the templates is described in the following chapters, as they are used to implement the BACnet protocol in the existing project sample.

In the templates, for the Objects which have the `Relinquish_Default` attribute, it is set to be stored in the EEPROM. This is not necessary in case you link the attribute to an EEPROM parameter and therefore you could disable the EEPROM option of the attribute in this case.

### 3.4. Implementing the BACnet protocol into an existing application

This chapter contains the step-by-step tutorial for implementing the BACnet protocol into the sample thermostat application.

#### 3.4.1. Load the BACnet\_IEC.PLL and BACnet\_Addons.plclib libraries

Before adding an object, it is required to connect to the `BACnet_IEC.pll` library.

Open the Library manager and add the library, which is located in the FreeStudio installation folder (`..\Catalog\FreeAdvance\PLC` or `..\Catalog\FreeEvolution\PLC`).

Alternatively, if you add a new BACnet Device object without adding the library before, you will be asked if you want to add the `BACnet_IEC` library. Answering *Yes*, you will have *Application* get the library automatically added for you.

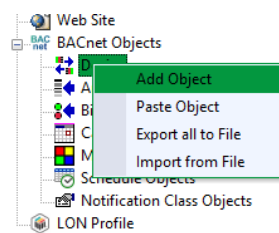
From the same window, load also the `BACnet_Addons.plclib`.

#### 3.4.2. Add the Device BACnet Object

In *Resources*, add a new Device BACnet Object, with the name e.g. *BACnet\_Device\_Thermostat*.

The *Subnet* and the *Node number* do not have to be specified, as they are used to determine the `Object_Identifier`, which will be set-up by a specific function of the BACnet Addons Library, called in the *BACnet\_Setup\_Init* program (see §3.4.7 *HowTo Code the BACnet\_Setup\_Init* program).

Alternatively, you can define the Device BACnet Object by importing the provided *Device\_Template.xml*.



#### 3.4.3. Add an Analog Value BACnet Object

Each BACnet Analog Value Object can be defined either manually or by using one of the provided templates.

In this case, we will add an Analog Value that represents the Temperature SetPoint, which we will call *BAV0\_SetCool*. For this, we will use the *AV\_Temp\_Template.xml* (import it from the Analog Value Object dropdown menu).



We will also add an Analog Input that will use to make the Ambient Temperature available on the BACnet side and for this we will use the *AI\_Temp\_template.xml* (import it from the Analog Value Object dropdown menu). We will call this object *BAI0\_AmbTemp*.

A template to setup a Humidity parameter is also provided.

#### **3.4.4. Add a Binary Value BACnet Object**

Each BACnet Binary Value Object can be defined either manually or by using one of the provided templates.

In this case, we will add a Binary Value that represents the Machine State, which we will call *BBV0\_MachineState*. For this, we will use the *BV\_Alarm\_Template.xml* (import it from the Binary Value Object dropdown menu), which we will then modify.

Binary Input Object type is not supported and it has to be treated using a Binary Value. We will add a Binary Value with info which would be normally represented using a Binary Input. To implement this Binary Value, the provided template (*BI\_Template.xml*) can be used in connection with the *BACnet\_BI\_SetPV* FB.

We will call this object *BI1\_OutputCooling* and it will be used to make the value of the *xOutputCooling* variable available on the BACnet side.

#### **3.4.5. Add a Multi State Value BACnet Object**

Each BACnet Multi State Value Object can be defined either manually or by using one of the provided templates.

In this case, we will add a Multi State Value that represents the Mode of the machine based on the season (Winter, Summer, Auto), which we will call *BMSV1\_SeasonMode*. For this, we will use the *MSV\_Template.xml* (import it from the Multi State Value Object dropdown menu).

We will also add a Multi State Input, using the *MSI\_Template* and representing Operating State of the machine (Running, Idle, Defrost, Safety).

We will call this object *MSI\_OperatState* and it will be used to make the value of the *usiOperating\_State* variable available on the BACnet side.

#### **3.4.6. Define the EEPROM parameters & StatusVars used in the BACnet setup**

For the *BACnet\_ObjID* function, it is required to add the parameters *n.1* and *n.2* in EEPROM. The parameter *n.3* has to be added in EEPROM to define the BACnet Object update timeframe (measured in 1/10 sec). The *BACnet\_Subnet* limits (Min: 0, Max: 63) must be also defined.

The EEPROM parameters from *n.4* to *n.9* are used to setup the BBMD functionality (see §3.4.8 *HowTo Code the BACnet\_BBMD\_ReinitDevice* program for details).



#	Name	Device type	Appl. type	Default	Min	Max	Scale	Offset
1	uiBACnet_ID	Unsigned 16-bit	UINT				1	0
2	usiBACnet_Subnet	Unsigned 8-bit	USINT		0	63	1	0
3	ui_dt_BACnetUpd_dsec	Signed 16-bit	UINT	10			1	0
4	BBMD_Ip1	Unsigned 8-bit	USINT	0			1	0
5	BBMD_Ip1	Unsigned 8-bit	USINT	0			1	0
6	BBMD_Ip1	Unsigned 8-bit	USINT	0			1	0
7	BBMD_Ip1	Unsigned 8-bit	USINT	0			1	0
8	BBMD_port	Unsigned 16-bit	UINT	0			1	0
9	BBMD_tmo	Unsigned 8-bit	USINT	0			1	0
10	xBACnet_Enable	Boolean	BOOL	TRUE			1	0

Define the following *Status variables*:

#	Name	Device type	Appl. type	Default	Min	Max	Scale	Offset	Read only
1	usiBACnet_ReinitDevice	Unsigned 8-bit	USINT	0	0	2	1	0	FALSE

### 3.4.7. HowTo Code the BACnet\_Setup\_Init program

Create a new ST program, name it e.g. BACnet\_Setup\_Init and assign it to the *Init* task.

- It is required to **restore the BACnet Device Object Identifier** at every boot, in order to keep the Identifier aligned with the one set in the supervisor.

Call the `BACnet_ObjID` function, passing the `uiBACnet_ID` and `usiBACnet_Subnet` parameters set in the EEPROM.

In this example, the call to be made is the following:

```
BACnet_Device_Thermostat.Object_Identifier:=BACnet_ObjID(uiBACnet_ID,usiBACnet_Subnet);
```

Note that `BACnet_Device_Thermostat` is the BACnet Device name which we have chosen in the previous steps. Have a look at the example for further details.

**NOTE:** the `Object_Identifier` of each BACnet device **MUST be unique** in the BACnet network which the BACnet devices are connected to. The IP address of each device must also be unique.

- To **store the Firmware Revision** in the BACnet Device attributes, it is possible to use the `sysMSK` and `sysVER` variables to generate it.

After having defined two local STRING variables (`string0` and `string1`), the *Firmware Revision* can be generated and stored in the BACnet Device Object as follows:

```
string0 := TO_STRING(sysMSK);  
string0 := CONCAT(string0, '.');  
string1 := TO_STRING(sysVER);  
BACnet_Device_Thermostat.Firmware_Revision := CONCAT(string0,string1);
```



- The Port\_BACnet\_IP global variable allows to set the BACnet/IP port number. It can be accessed from the *Device* work environment (in *All parameters\BACnet*) and if it is equal to 0 it sets the default port (47808). In case its value is 65535, it disables the BACnet stack on the ETH/TCP side (not visible), but it is still running on the PLC side.

By means of the EEPROM parameter xBACnet\_Enable, it is possible to set the value of Port\_BACnet\_IP to 65535 in case xBACnet\_Enable is FALSE, in order to **disable the BACnet protocol**.

The code to achieve it is the following:

```
IF NOT(xBACnet_Enable) AND Port_BACnet_IP<>65535 THEN
  (* Force Bacnet Disable*)
  xRet := sysWriteParUINT(ADR(Port_BACnet_IP),65535);
END_IF;
```

- Finally, at the end of the initialization task related to BACnet, **set the sysBACnet\_ReinitDevice value to 0**, which means no system reset has to be executed (0=idle), as a command that could be request form the BACnet side. This parameter will be clarified in the *ReinitDevice* program (see §3.4.8 *HowTo Code the BACnet\_BBMD\_ReinitDevice program* for details).

### 3.4.8. HowTo Code the BACnet\_BBMD\_ReinitDevice program

This program allows to set up the BBMD (Foreign Devices Handling) and to define the Warm restart and Cold restart procedures which can be triggered by the supervisor (using the sysBACnet\_ReinitDevice variable).

Create a new ST program, name it e.g. BACnet\_BBMD\_ReinitDevice and assign it to the *Background* task.

Paste the following code:

```
(* BBMD Service *)
(* IP 0.0.0.0 ==> BBMD disabled *)
xRet := sysBACnet_BBMD(BBMD_Ip1,BBMD_Ip2,BBMD_Ip3,BBMD_Ip4,BBMD_port,BBMD_tmo);

(* Reinit Device *)
(* 0=Idle, 1=WARMSTART, 2=COLDSTART; it is set 1 or 2 by DM-RD-B command and must be set to 0 by PLC application *)

(* The action related to warm and coldstart is defined by the developer.... *)
TonWTD(IN:=(sysBACnet_ReinitDevice<>0),PT:=2000);
IF sysBACnet_ReinitDevice=1 THEN
  (* Watchdog Call - This is just a case, it could be also a "switch off the unit" *)
  IF TonWTD.Q THEN
    bret := sysWD_Background(0);
  END_IF;
  usiBACnet_ReinitDevice := sysBACnet_ReinitDevice;
ELSIF sysBACnet_ReinitDevice=2 THEN
  (* Reload Bacnet default *)
  xRet := sysWriteParBOOL(ADR(sysLoadBACnetE2Defaults),TRUE);
  (* Watchdog Call *)
  IF TonWTD.Q THEN
    bret := sysWD_Background(0);
  END_IF;
  usiBACnet_ReinitDevice := sysBACnet_ReinitDevice;
ELSE
  sysBACnet_ReinitDevice := usiBACnet_ReinitDevice;
END_IF;
```





And define the following local variables:

#	Name	Type
1	xRet	BOOL
2	TonWTD	TON

The BBMD service is configured calling the EEPROM parameters previously defined; no modification to this code is required.

The second part of the code contains the instructions regarding the *ReinitDevice Service*. Here the Application developer can define the list of operations which will be run when the *WarmRestart* and the *ColdRestart* procedures are executed. They are run based on the `sysBACnet_ReinitDevice` value (1 for *WarmRestart*, 2 for *ColdRestart*). When it is equal to 0, the service is in idle state (not being executed).

The value of the `sysBACnet_ReinitDevice` variable is normally changed by the supervisor to trigger a restart, but it can eventually also be modified from the IEC side, by changing the `usiBACnet_ReinitDevice` status variable, which is also accessible from ModBUS, as it has been so defined.

The two restart procedures can contain all the instruction that the developer would like to be executed and they are naturally application-dependent. For example, in case of a thermostat application, the SetPoint default could be restored.

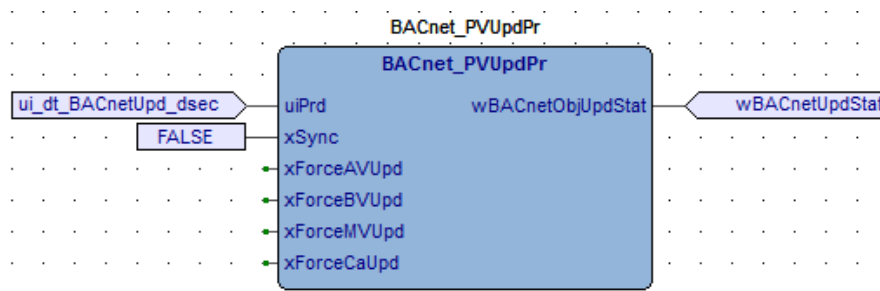
### 3.4.9. HowTo Automate the Present\_Value update procedure

The *Present\_Value* of each BACnet Object has to be updated by calling the update function for each specific type and object.

This can either be done object by object (e.g. enabling the `xLocalPV_Upd` input on the FB that is used to implement that BACnet Object) or it can be executed for all the objects together in one shot. This last option practically automates the *Present\_Value* update procedure and is implemented using the *BACnet\_PVUpdPr* function block.

Create a new program in FBD language with the name `BACnetPVObjUpd_P` and assign it to the *Background* or to the *Timed* task. Note that, in case it will be assigned to the *Timed* task, it is required to take into consideration the total execution time of the task, which naturally increases with the number of BACnet objects.

Structure the program as below:



The `dt_BACnetUpd_dsec` is the EEPROM parameter defined before, which contains the info about the *Present\_Value* update period in [sec/10]. The `wBACnetUpdStat` is a WORD, local or global variable depending on whether the status info about the *Present\_Value* update procedure is used in the IEC code or not.



If you set up the FB as shown above, the Present\_Value will be updated asynchronously (one object type per time) at each defined *uiPrd* period, in a rotational sequence.

If you would like to setup a different update strategy, have a look at the FB properties or §2.4.2 BACnet\_PVUpdPr.

### 3.4.10. HowTo Link an EEPROM Parameter to a BACnet Analog Value Object

In this paragraph the procedure of how to implement BACnet for an existing EEPROM parameter will be explained.

The BACnet AV will be implemented for the EEPROM *iSetPoint* parameter. The steps to do that are the following:

1. Rename the EEPROM parameter from *iSetPoint* to *iSetPoint\_E2*;
2. Create a new global variable (same type as the EEPROM parameter) with the same name the EEPROM parameter originally had. In our case, it is *iSetPoint*.
3. Create a new FBD program (e.g. *BACnet\_Objs\_P*) and assign it to the Background task. In the program, call the *BACnet\_RelDef\_AV\_link\_IEE* FB and:
  - a. use the EEPROM *iSetPoint\_E2* as input (pass its address to *ptr\_IEE*);
  - b. use the new *iSetPoint* global variable as output (connect it to *iPV\_scaled*).

The implementation of what above described is illustrated in the image below.

The BACnet Object which will be linked to the SetPoint EEPROM parameter is the BAV0\_SetCool Analog Value Object.

Define the *iRelMin*, the *iRelMax* and the *rScaler* parameters as constants or as global variables, in case you need them to be dynamic, assigning the correct default value to them.

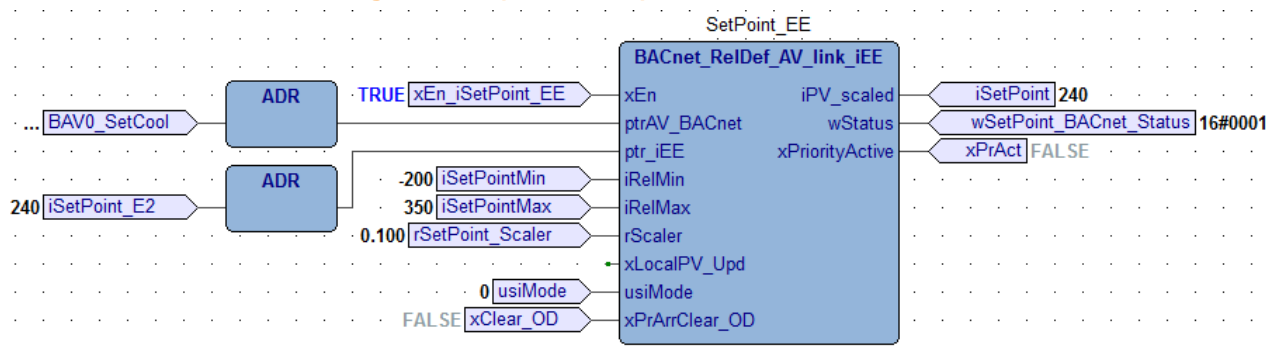
The enable variable (*xEn*) can be used to set the *Out\_of\_Service* state and therefore it is proper to pass a variable to it, instead of a constant value.

The output *wSetPoint\_BACnet\_Status* is a WORD, local or global variable depending on whether the status of FB is used in the IEC code or not.

The *usiMode* has to be set according to how you would like to manage the priorities between the IEC and the BACnet side. Read the function block description for more details.

The *xClear\_OD* input will be associated to a procedure that will set it from FALSE to TRUE, and then restore it to FALSE, in order to clear the Priority Array on demand. The information about whether the Priority Array is being used is provided by the *xPrAct* output.

**In the project sample, an HMI (Setpoint page) has been added, which allows you to test all the three available working modes (0, 1 and 2).**





If you have already an HMI in your existing project, read the §3.4.19 EEPROM variable names and HMI.

### 3.4.11. HowTo Link an EEPROM Parameter to a BACnet Binary Value Object

In this paragraph we will link the *xMachineState* EEPROM parameter to the *BBV0\_MachineState* Binary Value BACnet Object. It represents the state of the machine, which could be either ON or OFF.

The procedure to follow to implement it is very similar to what seen for the Analog Value (§3.4.10). You actually will need to:

1. Rename the EEPROM parameter from *xMachineState* to *xMachineState\_E2*;
2. Create a new global variable (same type as the EEPROM parameter) with the same name the EEPROM parameter originally had, i.e. *xMachineState*.
3. Use the FBD program *BACnet\_Objs\_P* created in §3.4.10 and insert a new FBD free row where you will call the *BACnet\_RelDef\_BV\_link\_xEE* FB and:
  - a. use the EEPROM *xMachineState\_E2* as input (pass its address to *ptr\_xEE*);
  - b. use the new *xMachineState* global variable as output (connect it to *xPV\_scaled*).

The implementation of what above described is illustrated in the image below.

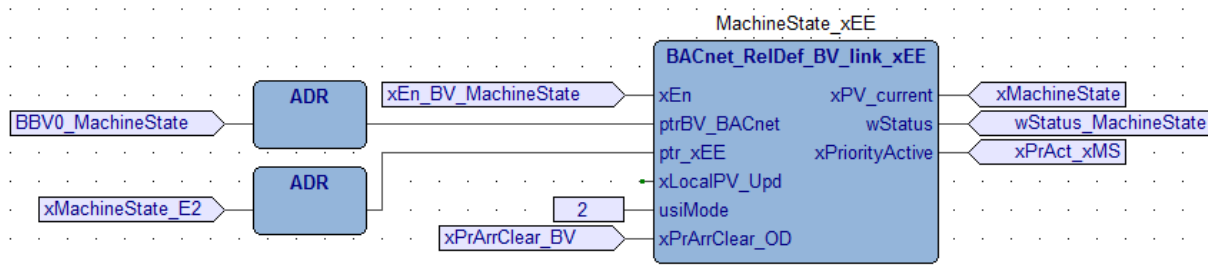
The BACnet Object which will be linked to the *xMachineState* EEPROM parameter is the *BBV0\_MachineState* Analog Value Object.

The enable variable (*xEn*) can be used to set the *Out\_of\_Service* state and therefore it is proper to pass a variable to it, instead of a constant value.

The output *wStatus\_MachineState* is a WORD, local or global variable depending on whether the status of FB is used in the IEC code or not.

The *usiMode* has to be set according to how you would like to manage the priorities between the IEC and the BACnet side. Read the function block description for more details.

The *xPrArrClear\_BV* input will be associated to a procedure that will set it from FALSE to TRUE, and then restore it to FALSE, in order to clear the Priority Array on demand. The information about whether the Priority Array is being used is provided by the *xPrAct\_xMS* output.



If you have already an HMI in your existing project, read the §3.4.19 EEPROM variable names and HMI.



### 3.4.12. HowTo Configure a Multi State Input BACnet Object

In this paragraph we will configure the *MSI\_OperatState* Multi State Input BACnet Object to be synchronized with the *usiOperating\_State* IEC variable.

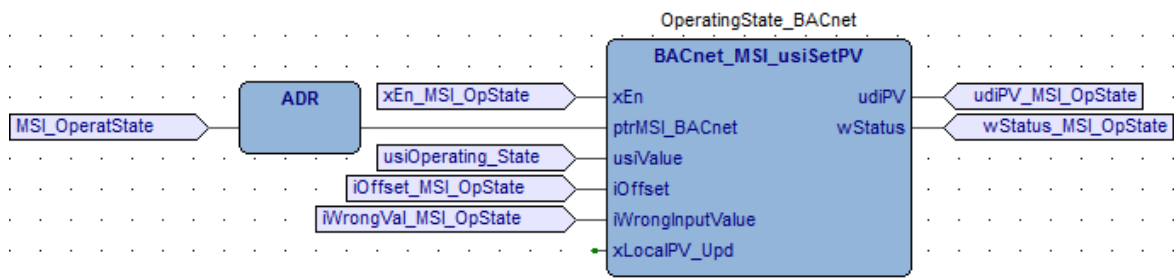
The implementation is relatively simple, as it is just required to call the *BACnet\_MSI\_usiSetPV* FB into a program in *Background* (e.g. the one used for the other objects: *BACnet\_Objs\_P*) and provide as inputs:

- the value to be written on the BACnet Object (i.e. *usiOperating\_State*);
- the offset, if the value is not in the [1-5] format;
- the pointer to the BACnet Object;
- the value to be written in case of wrong input value.

The implementation of what above described is illustrated in the image below.

The enable variable (*xEn*) can be used to set the *Out\_of\_Service* state and therefore it is proper to pass a variable to it, instead of a constant value.

The output *udiPV\_MSI\_OpState*, which returns the *Present\_Value* of the MSI Object, and the output *wStatus\_MSI\_OpState*, which is a *WORD* reporting the status of the FB operation, are local or global variables, depending on whether one or both the information are used in the IEC code or not.



### 3.4.13. HowTo Configure a Binary Input BACnet Object

In this paragraph we will configure the *BI1\_OutputCooling* Binary Input BACnet Object to be synchronized with the *xOutput\_Cooling* IEC variable.

The implementation is relatively simple, as it is just required to call the *BACnet\_BI\_SetPV* FB into a program in *Background* (e.g. the one used for the other objects: *BACnet\_Objs\_P*) and provide as inputs:

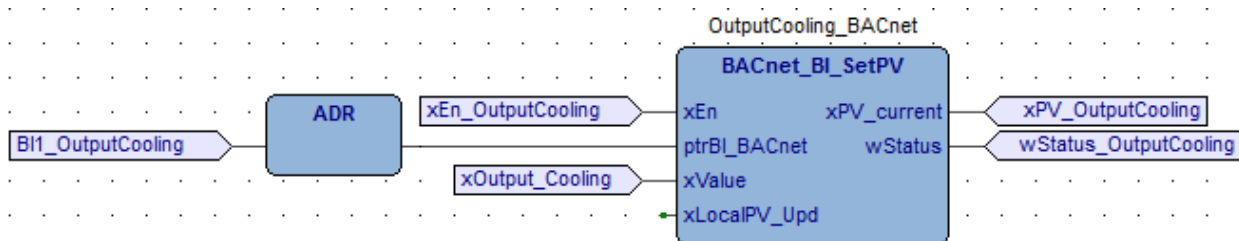
- the value to be written on the BACnet Object (i.e. *xOutput\_Cooling*);
- the pointer to the BACnet Object.

The implementation of what above described is illustrated in the image below.

The enable variable (*xEn*) can be used to set the *Out\_of\_Service* state and therefore it is proper to pass a variable to it, instead of a constant value.



The output *xPV*, which returns the *Present\_Value* of the BI Object, and the output *wStatus*, which is a WORD reporting the status of the FB operation, are local or global variables, depending on whether one or both the information are used in the IEC code or not.



### 3.4.14. HowTo Configure an Analog Input BACnet Object

In this paragraph we will configure the *BAIO\_AmbTemp* Analog Input BACnet Object to be synchronized with the Analog Output 1 (variable *AO\_AmbTemp*).

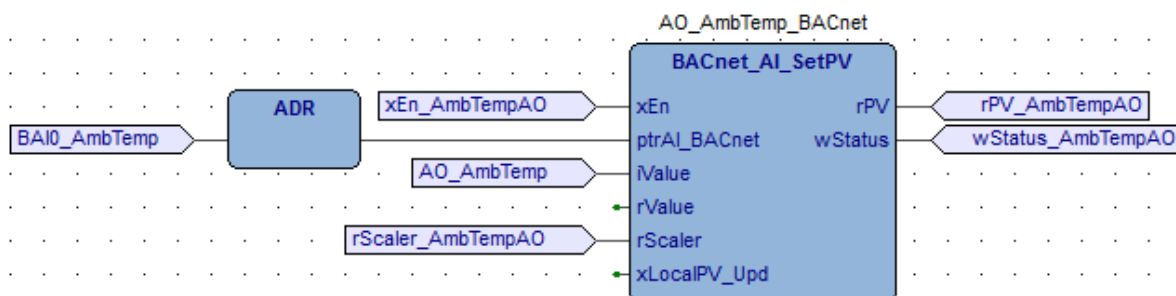
The implementation is relatively simple, as it is just required to call the **BACnet\_AI\_SetPV** FB into a program in *Background* (e.g. the one used for the other objects: *BACnet\_Objs\_P*) and provide as inputs:

- the value to be written on the BACnet Object (i.e. *AO\_AmbTemp*) in INT or REAL format;
- the *rScaler* (in any case);
- the pointer to the BACnet Object.

The implementation of what above described is illustrated in the image below.

The enable variable (*xEn*) can be used to set the *Out\_of\_Service* state and therefore it is proper to pass a variable to it, instead of a constant value.

The output *rPV*, which returns the *Present\_Value* of the AI Object, and the output *wStatus*, which is a WORD reporting the status of the FB operation, are local or global variables, depending on whether one or both the information are used in the IEC code or not.



### 3.4.15. HowTo Link an EEPROM Parameter to a BACnet Multi State Value Object

In this paragraph we will link the *usiSeasonMode* EEPROM parameter to the *BMSV1\_SeasonMode* Multi State Value BACnet Object. It represents the operating mode of the machine (summer, winter, auto).

The procedure to follow to implement it is very similar to what seen for the Analog Value (§3.4.10). You actually will need to:

1. Rename the EEPROM parameter from *usiSeasonMode* to *usiSeasonMode\_E2*;
2. Create a new global variable (same type as the EEPROM parameter) with the same name the EEPROM parameter originally had, i.e. *usiSeasonMode*.



3. Use the FBD program *BACnet\_Objs\_P* created in §3.4.10 and insert a new FBD free row where you will call the *BACnet\_RelDef\_MV\_link\_usiEE* FB and:
  - a. use the EEPROM *usiSeasonMode\_E2* as input (pass its address to *ptr\_usiEE*);
  - b. use the new *usiSeasonMode* global variable as output (connect it to *usiPV\_scaled*).
4. Provide the offset used for the EEPROM parameter, if the value is not in the [1-5] format;
5. Provide the value to be written on the PV in case of wrong input value.

The implementation of what above described is illustrated in the image below.

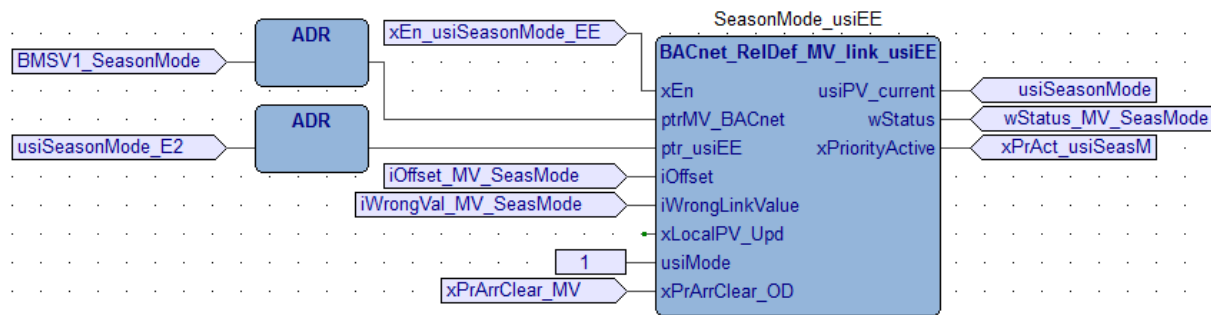
The BACnet Object which will be linked to the *usiSeasonMode* EEPROM parameter is the *BMSV1\_SeasonMode* Multi State Value Object.

The enable variable (*xEn*) can be used to set the *Out\_of\_Service* state and therefore it is proper to pass a variable to it, instead of a constant value.

The output *wStatus\_MV\_SeasMode* is a WORD, local or global variable depending on whether the status of FB is used in the IEC code or not.

The *usiMode* has to be set according to how you would like to manage the priorities between the PLC and the BACnet side. Read the function block description for more details.

The *xPrArrClear\_MV* input will be associated to a procedure that will set it from FALSE to TRUE, and then restore it to FALSE, in order to clear the Priority Array on demand. The information about whether the Priority Array is being used is provided by the *xPrAct\_usiSeasM* output.



If you have already an HMI in your existing project, read the §3.4.19 EEPROM variable names and HMI.

### 3.4.16. HowTo Setup a Schedule BACnet Object

Import the Schedule Template as Schedule Object. This template contains all the parameters that would normally be used. You can choose to load the one with or the one without exceptions, based on your needs.

**Three** different examples of implementation of the Schedule BACnet Object type are provided with the project sample. In all the three examples, the schedule template without exceptions is used.





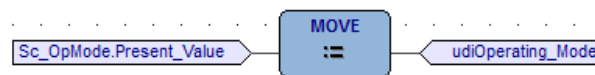
### Example 1

In this example, the schedule object (*Sc\_OpMode*) contains the operating mode (Normal: 0, Comfort: 1, Not\_occupied: 2) for working week:

- Monday to Friday:
  - o *Normal* from 0:00 to 6:29;
  - o *Comfort* from 6:30 to 17:59;
  - o *Not\_occupied* from 18:00 to 23:59;
- on Saturday and Sunday: *Not\_occupied*.

The default is *Normal*.

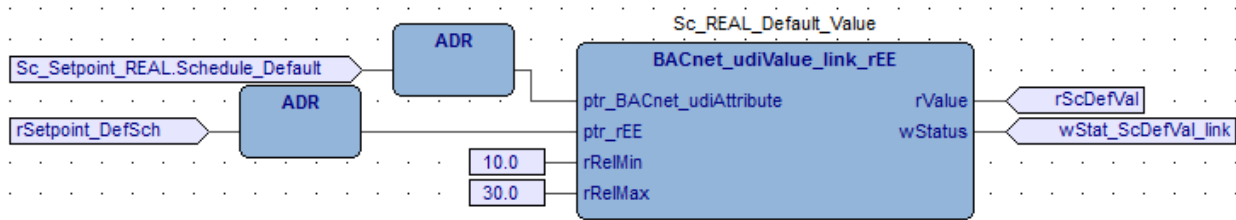
The setup is executed by modifying each attribute of the Object itself. For more clarifications, have a look at the tests done with the supervisor in §3.5 Testing BACnet with YABE.



### Example 2: HowTo Use the Schedule with REAL

In this example, the schedule template without exceptions is used. The implemented BACnet protocol in the Eliwell controllers supports only UDINT as values of the Schedule. Therefore, in case the supervisor uses REAL, it is required to read and write back the data as REAL. This can be done by the *BACnet\_UdAsReal\_read* and *BACnet\_UdAsReal\_write* functions.

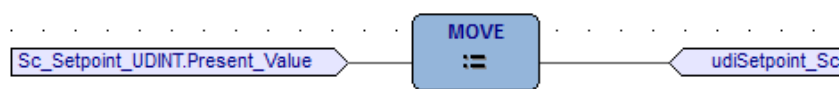
You typically would like to define the default value of the Schedule BACnet object as retentive (in EEPROM), however, it is not straightforward to set it up in a REAL-equivalent data format. To simplify its configuration, it is possible to link it to a REAL EEPROM parameter using the *BACnet\_udiValue\_link\_rEE* function block. Its implementation is shown in the picture below.



As it is configured, the value of the *rSetpoint\_DefSch* EEPROM parameter will be stored into the EEPROM *Schedule\_Default* attribute of the *Sc\_Setpoint\_REAL* Schedule BACnet Object.

### Example 3

In this example, the schedule template without exceptions is used. This is a trivial example that shows how a Schedule, which is read from a supervisor using the UDINT data format, can be set up. It is sufficient to define the Schedule default value in the Resources working space of Application. The picture shows how to read the present value of the Schedule, which is changed according to the settings normally modified by the supervisor.







### 3.4.17. *HowTo Setup a Calendar BACnet Object*

Import the Calendar Template as Calendar Object. The template contains all the available parameters for this object already setup.

The Calendar in the sample project contains the Italian holidays of 2018.

**NOTE** that once you set a value for one of the *weekNDay* parameters, in order to restore the null value, you need to remove it and add it again.

### 3.4.18. *HowTo Setup a Notification Class BACnet Object*

Import the Notification Class Template as Notification Class Object. The template contains all the available parameters for this object already setup.

At the moment, six notification class objects are setup in the sample project, but their configuration is left up to the developer according to possibilities provided by the BACnet protocol.

### 3.4.19. *EEPROM variable names and HMI*


If the existing project in which you would like to implement the BACnet project has already an HMI developed using the User Interface working environment, **note** that it is required to update the EEPROM parameters names used in the HMI project (i.e. the EEPROM parameters links to the IEC code will be lost, as their name has been modified in *Application*).



### 3.5. Testing BACnet with YABE

The BACnet protocol is normally used by supervisors. In order to test the implemented Object without having a supervisor available, it is possible to use a so-called *BACnet Explorer* software. In this chapter, the freeware YABE (Yet Another Bacnet Explorer) software will be used for testing.

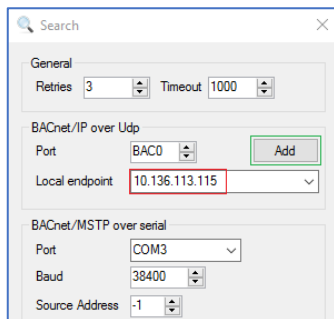
#### 3.5.1. HowTo Add a BACnet Device in YABE and Subscribe to an Object

In YABE, click on the  icon. The popup windows shown on the side will open. In *Local endpoint* enter the IP address of your computer and click on *Add*.

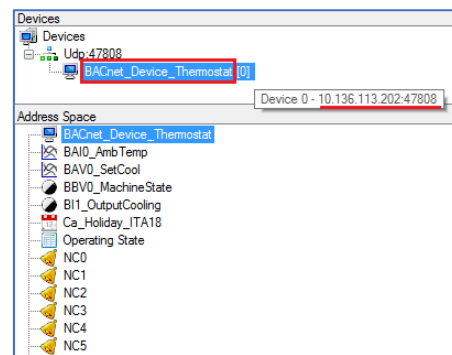
The BACnet devices found in the network will be listed.

Identify your Evolution/Advance from the BACnet Device name or from its IP address.

In *Address Space* tab you will see all the Object which are defined on the BACnet Device.



Search criteria for BACnet Devices



BACnet Device List and Address Space

#### 3.5.2. HowTo Subscribe a BACnet Object

Right-clicking an object, you can choose to *Subscribe* to an object. In the central tab of the application, you will you see all the active subscriptions.

Subscriptions, Periodic Polling, Events/Alarms					
Device	ObjectId	Name	Value	Time	Status
10.136.113.202:47808 - 0	OBJECT_ANALOG_INPUT:0	BAI0_AmbTemp	2	11:03:05	OK
10.136.113.202:47808 - 0	OBJECT_ANALOG_VALUE:0	BAV0_SetCool	51	11:03:05	FAULT
10.136.113.202:47808 - 0	OBJECT_BINARY_VALUE:0	BBV0_MachineState	0	11:03:05	OK
10.136.113.202:47808 - 0	OBJECT_BINARY_VALUE:1	BI1_OutputCooling	0	11:03:05	ALARM
10.136.113.202:47808 - 0	OBJECT_MULTI_STATE_INPUT:0	Operating State	1	11:03:05	OK
10.136.113.202:47808 - 0	OBJECT_MULTI_STATE_VALUE:0	BMSV1_SeasonMo...	1	11:03:05	FAULT



### 3.5.3. HowTo Change the Relinquish\_Default

The *Relinquish\_Default* parameter is used in the Analog Value, Binary Value and Multi State Value Objects.

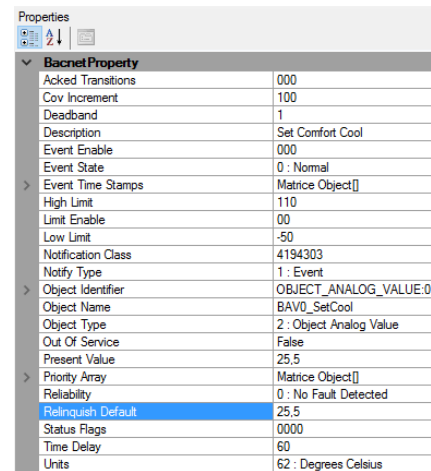
We will test it with the iSetPoint parameter, which is an Analog Value BACnet Object (BAV0\_SetCool) linked to the iSetPoint\_E2 EEPROM parameter.

Select the BAV0\_SetCool object in the *Address Space* tab. On the right, you have the *Properties* tab, where all the properties of the object attributes are shown.

Also in the HMI, you will see that the PV, the RelDef and the EEPROM value are aligned at 25,0 °C. Now you can change the *Relinquish\_Default*, by typing the desired value and pressing the return button. This action will change the RelDef, the PV and also align the RelDef to the EEPROM parameter.

If you put the object out of service, when you change the EEPROM parameter (e.g. using the HMI), the object will not be aligned: YABE will still show the old values and vice-versa.

**Note** that the *Properties* tab is not refreshed automatically. You need to click on another object and then click back again on the object of which you would like to see the updated properties.



Properties	
<b>BacnetProperty</b>	
Acked Transitions	000
Cov Increment	100
Deadband	1
Description	Set Comfort Cool
Event Enable	000
Event State	0 : Normal
Event Time Stamps	Matrice Object[]
High Limit	110
Limit Enable	00
Low Limit	-50
Notification Class	4194303
Notify Type	1 : Event
Object Identifier	OBJECT_ANALOG_VALUE:0
Object Name	BAV0_SetCool
Object Type	2 : Object Analog Value
Out Of Service	False
Present Value	25,5
Priority Array	Matrice Object[]
Reliability	0 : No Fault Detected
Relinquish Default	25,5
Status Flags	0000
Time Delay	60
Units	62 : Degrees Celsius

### 3.5.4. HowTo Write on the Priority\_Array

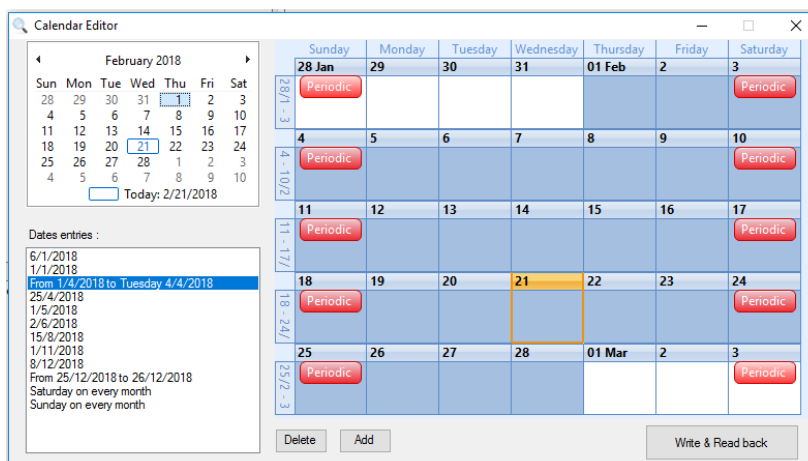
To write on the *Priority\_Array*, it is required to press the combination CTRL+ALT+*Number*, where *Number* is a number from 1 to 9, with an offset of 1 with respect to the *Priority\_Array* row in which you would like to write.

In this way, it is not possible to write on the rows 9-15.

To execute the write operation, type then the value in the *Present\_Value* field.

### 3.5.5. HowTo Edit a Calendar Object

Right click on the Calendar object and choose *Show Calendar*. This will open the *Calendar editor*.



Here you can add, modify or delete the various entries of the calendar. Click "Write & Read back" to write the values.

Then, you can see the effect of your choices on the *Present\_Value* of the Calendar, which is also shown on the HMI in form of "WorkDay".



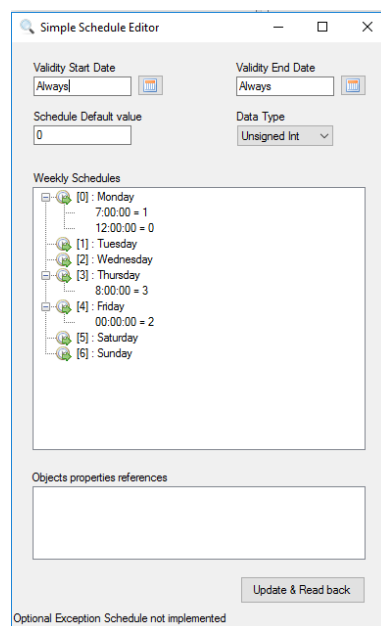
### 3.5.6. HowTo Edit a Schedule Object

Right click on the Schedule object and choose *Show Schedule*. This will open the *Schedule editor*.

Here you can add, modify or delete the various entries of the schedule. Click "Update & Read back" to write the values.

Note that only the values compatible with the UDINT data format are supported by the controller, therefore you need to set it up accordingly in YABE before any write operation is executed (e.g. select the *Unsigned Int* format).

YABE uses Real as default and when you open a Schedule object that contains already some entries it will interpret the UDINT data as REAL, leading therefore to a wrong read operation. To be able to write again after this, it is required to delete all the entries and re-create the Schedule.

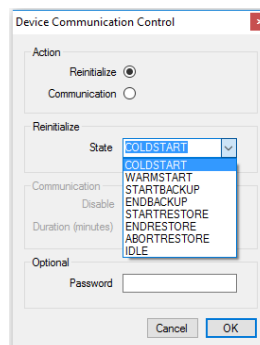
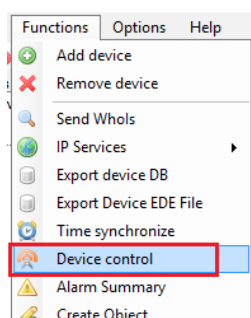


### 3.5.7. HowTo Reinitialize the Device via BACnet

The ReinitDevice procedure has been defined in the project and it can be triggered using YABE.

Go in *Functions\Device control*.

Select the Type of reinitialization that you would like to execute (cold or warm restart) and confirm your operation to run it.





## Appendix

### 4.1. *Hardware Information*

#### 4.1.1. *Usage of the EEPROM vs BACnet*

On the M17x platform controllers, on a single EEPROM location it is possible to write only 100.000 times (this is related to the controller hardware specs). Afterwards, it is not anymore guaranteed that the value will be retained correctly.

To each EEPROM parameter one or more EEPROM locations are assigned, depending on the data type.

The above-mentioned limitation has to be taken into account when using the FBs of this library that write on EEPROM. These include:

- The “link” FBs, and in this case what especially has to be taken under consideration is the **use of the Mode 2**. In Mode 2, the Present Value is written on the EEPROM, while in Mode 0 and 1 it is the Relinquish Default that gets written on the EEPROM instead. Normally, the Present Value is more likely to be subject to frequent changes than the Relinquish Default, therefore the risk to have more frequent writes is higher in Mode 2 than in Mode 0 or 1.
- The “BACnet\_ModBUS\_SL” FBs, when the Relinquish Default is set to be stored in EEPROM.

### 4.2. *Acronyms*

- When talking about the IEC side/code in this document, the reference is to the PLC side/code.



## Publisher's Info

---

The **publisher** of this library is the **HVAC Solution Center** based in Alpago (BL), Italy. Its main goal is to enhance L2G and G2L solutions like libraries, applications and examples to support the ADEs community.

The **authors** are:

- **Pierpaolo Armeli**  
[pierpaolo.armeli@schneider-electric.com](mailto:pierpaolo.armeli@schneider-electric.com)
- **Federico Marcassa**  
[federico.marcassa@schneider-electric.com](mailto:federico.marcassa@schneider-electric.com)