

OpenBAS

BUILDING
AUTOMATION
SYSTEM

CONTENTS

1.0	Understanding the Building Automation Controller Software	4
1.1	OpenBAS-HV-NX10P Database Description	4
1.2	OpenBAS Configuration Tool Description	9
1.3	Communication Setup.....	10
1.4	Device Status.....	12
1.5	Active Project Selection	13
1.6	View Local and Slaves Inputs and Outputs and Manual Output Commanding	14
1.7	Analog I/O Viewing and Setup Type and Calibration.....	15
1.8	Digital I/O Viewing and Commanding	19
1.9	Remote Points and Fieldbus Communications	22
1.10	Labeling and Tagging Hardware and Software Objects	26
1.11	Scheduling for Lighting and General Purpose Uses	28
1.12	Data Logging and Graphics.....	32
1.13	Virtual Web Server Enabling	34
1.14	PLC (Programmable Logic Controller)	36
1.15	Database Parameters Viewing and Modifying.....	37
1.15.1	ADF 32-bit registers stored in EEPROM	38
1.15.2	ADI 16-bit registers stored in EEPROM	39
1.15.3	ADB 8-bit registers stored in EEPROM	40
1.15.4	RES_BIT 1-bit result registers stored in RAM	41
1.15.5	RES_FLT 32-bit result registers stored in RAM.....	42
1.15.6	TMR 16-bit system timers stored in RAM	43
1.15.7	MIN/MAX Setting of Minimum and Maximum limits for data registers	43
1.16	General System Configuration	45
1.16.1	Configure the Fieldbus Communication Ports	46
1.16.2	Set up Remote Points in Fieldbuses	48
1.16.3	Set up Expanded Remote Points in Fieldbuses	48
1.16.4	Set up Dual Core Remote Points in Fieldbuses	48
1.16.5	Set up Wireless Remote Points for Wireless Thermostats	49
1.16.6	Service EEPROM viewing and Programming.....	49

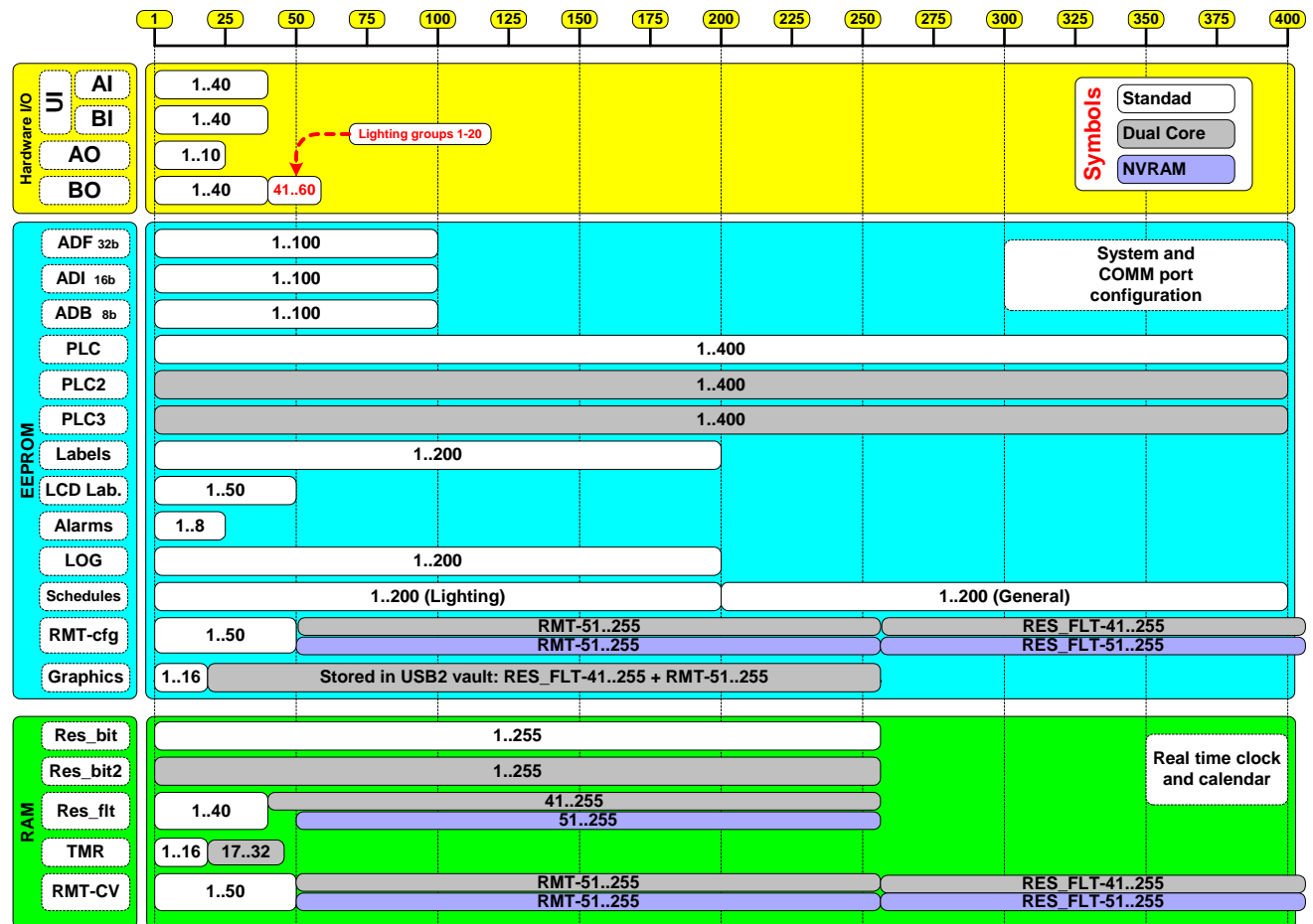
1.16.7	Reset controller to Factory Default Values	51
1.16.8	Set up what happens to Memory and I/O on Power On	52
1.16.9	Set up real Time Clock.....	53
1.16.10	Set up Daylight Saving Time	53
1.16.11	Set up Holidays for Scheduling operation.....	54
1.16.12	Create and Edit Label Tags for identifying all your I/O and Database variables.....	54
1.16.13	Create and Edit Label Tags for Remote points to show on LCD and BACnet names	55
1.16.14	Set up alarm messages for LCD operator display	56
1.16.15	Set up personalized screens for user interaction for LCD operator display	57
1.16.16	Set up SMS messages to send over the OpenBAS-NWK-SMS GSM/SMS module	59
1.16.17	Set up IP parameters for Network Card and e-mails for event and alarm reporting	60
1.16.18	Set up Passwords for different levels Operator, Technical and Administrator accounts	64
1.16.19	Create backups or restore all your Building Automation Controller programming	65
1.16.20	View historic log of events of your controller.....	66
1.16.21	Reset remotely controller and set up firmware updates.....	67
1.17	Reviewing version history	69
1.18	Open Password access for Password locked controller.....	69
2.0	PLC Ladder Programming Basics	70
2.1	PLC Ladder Programming Introduction.....	73
2.2	OpenBAS-HV-NX10P PLC Canvas	76
2.3	Moving Through your PLC Ladder Logic Program.....	77
2.4	Copy and Paste of a PLC Instruction	78
2.5	Create a new PLC Instruction	78
2.6	Disabling, Enabling, and Erasing PLC Instructions.....	79
2.7	Copy, Move or Erase a Block if PLC Instructions	79
2.8	Search a specific Instruction or Database Variable in your PLC Program.....	80
2.9	Save and Restore your PLC Ladder Logic Program to and from Disk in your PC	81

2.10	Generate your PLC Documentation	82
2.11	Library Import and Export.....	85
2.12	Template Based programming.....	87
2.13	Script Based Programming.....	88
2.14	Advanced C Programming for repetitive and advanced users	89
3.0	PLC Ladder Programming Detailed Programming Description	91
3.1	Logical Boolean Instructions, AND, NAND, OR, NOR, XOR, NXOR, INVERT	92
3.2	Hysteresis / In Range compare	95
3.2.1	HYSTERESIS MODE	95
3.2.2	IN RANGE MODE	97
3.3	Start stop instruction Easy start / stop with emergency stop	99
3.4	Combined logic AND – OR Combined logic instruction	100
3.5	Math Instructions,.....	103
3.6	Compare instructions, >, >=, <, <=, == (Equal), != (Not Equal)	105
3.7	Jump Instructions, Conditional and Unconditional Program Branching.....	107
3.8	Subroutine Calls, Conditional and Unconditional Subroutine Calling.....	109
3.9	Totalizer, Total Accumulator, Period Totalization, Energy Totalization	111
3.10	Lighting Groups, Creating a Lighting Group	116
3.11	Timer Instructions, Timer, Free run Oscillator, Value to Frequency Converter.....	120
3.12	Output Assignment, making things happen in the real world	125
3.13	Proportional Control, PID for Automation and HVAC applications	129
3.14	Alternate and Paralleling, Alternate and Paralleling of Pumps and Machinery	132
3.15	AHU and RTU staging, Create easily Simple or Complex HVAC sequences	136
3.16	Hour Counter, Create an hour counter	141
3.17	Section Labels, Organize and Document your PLC programs	143
3.18	Special User Programs Link your interface to user “C language” created Instructions	144
3.19	Program End Insert END instructions to terminate PLC or exit subroutine calls.....	144

1.0 Understanding the Building Automation Controller Software

1.1 OpenBAS-HV-NX10P Database Description

The internal database of the OpenBAS-HV-NX10P is a collection of objects that can be read or written by the controller itself, as well as by any of the communication channels. In the picture below the list of this collection of objects is shown.



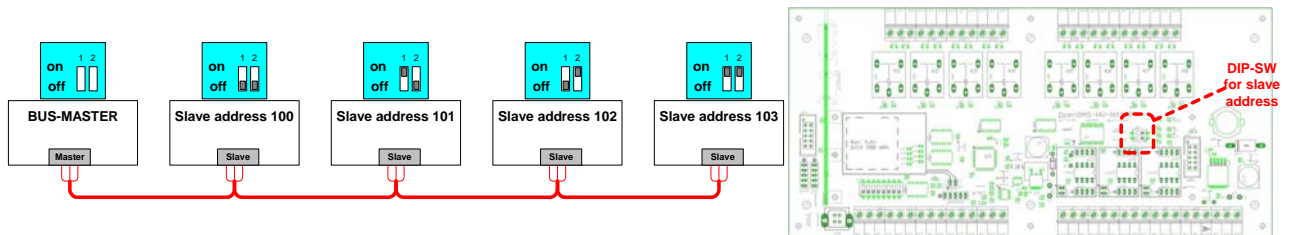
In the first section in YELLOW, all the **HARDWARE** objects can be seen, even while a single OpenBAS-HV-NX10P controller only has the following physical hardware:

- (8) Universal inputs (That can be Analog or Binary Inputs)
- (8) Binary Outputs
- (2) Analog Outputs

However, the logic of the OpenBAS-HV-NX10P can address up to:

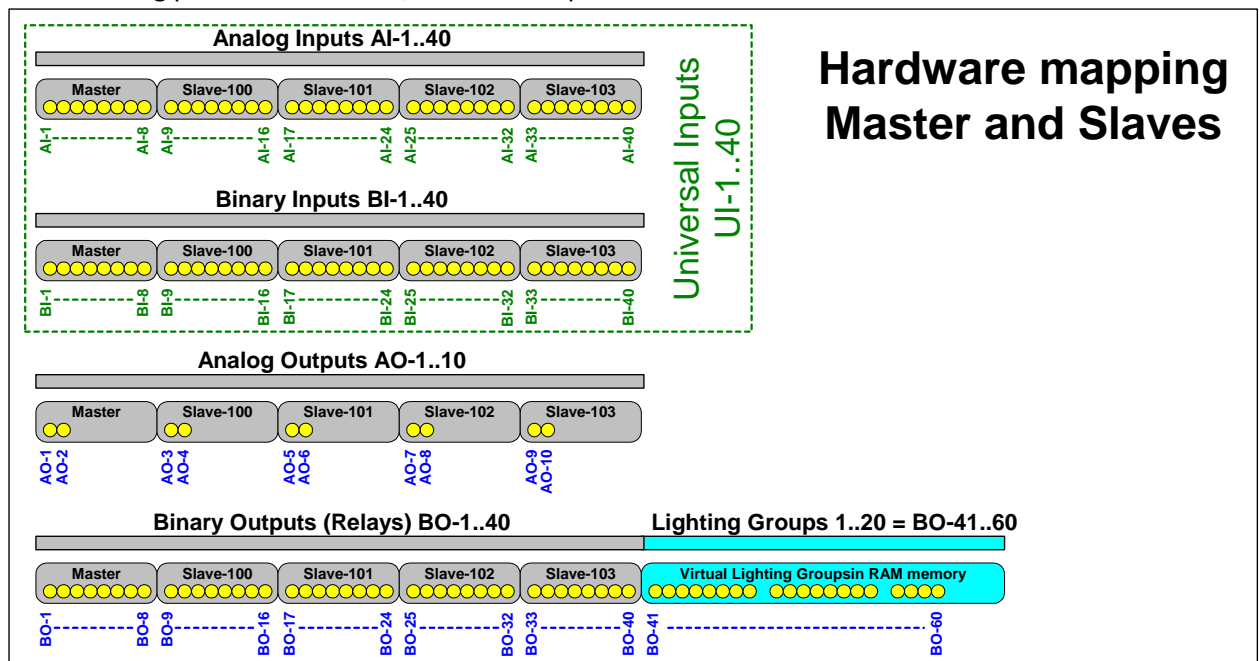
- (40) Analog Inputs (Shared with the Binary Inputs through the Universal Inputs)
- (40) Binary Inputs (Shared with the Analog Inputs through the Universal Inputs)
- (60) Binary Outputs (40 physical relays and 20 virtual Lighting Groups)
- (10) Analog Outputs.

This is possible because each OpenBAS-HV-NX10P controller can have four additional slave devices via the communication Fieldbuses, and the operating system handles all necessary logic so that the programmer can use this additional hardware directly. The following picture shows this relationship.

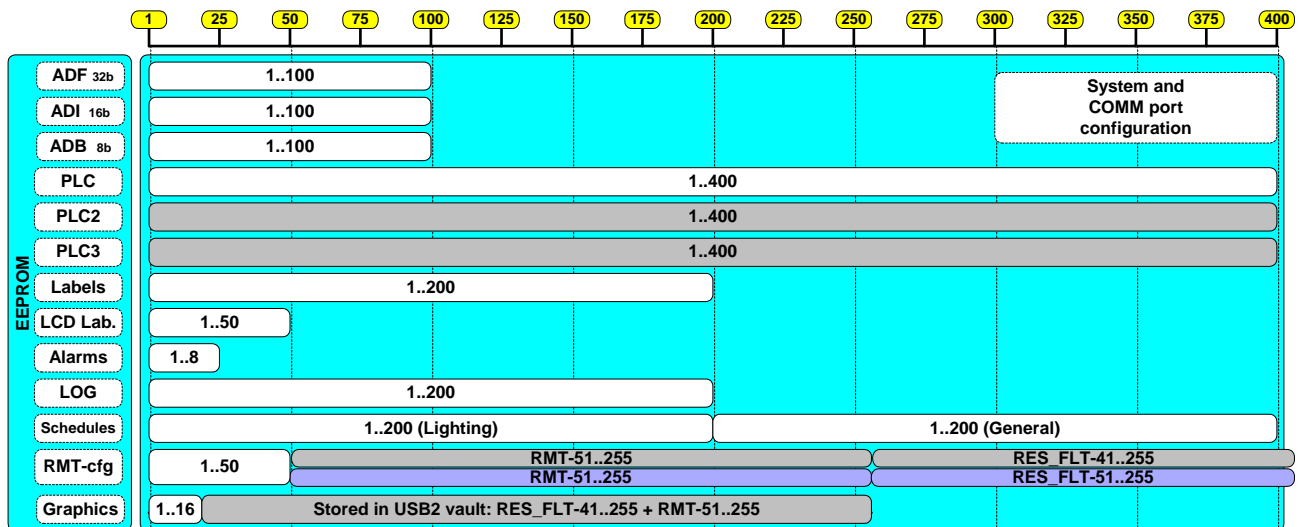


When configuring the controller's communication ports (see section 6.3 Communication Setup) each of the additional expansion boards can be configured as a **Master or as a Slave**. If configured as a slave, the address can be input directly by software or leave that field as ZERO and select the slave address by means of setting DIP-SW (2) to the positions depicted in the above image.

The following picture shows this I/O relationship in the Master & Slave devices.



The **EEPROM** section that is highlighted in CYAN contains the core of the database. The first three object types contain 300 data registers of 32 bits (ADF), 16 bits (ADI) and 8 bits (ADB) that can be read or written. Its main use is as set points as they are EEPROM based and their values will remain even if the battery is discharged.



Then following this 300 data registers, goes the PLC instructions, the standard OpenBAS-HV-NX10P has 400 INSTRUCTIONS, sometimes often referred as BLOCKS or EQUATIONS. If a Dual Core is installed, a second PLC2 adds additional 400 INSTRUCTIONS that are run inside the main or first core processor. Also a third PLC3 adds 400 additional INSTRUCTIONS that run in the second core.

Then comes 200 text labels used to tag all the objects in the database, these tags provide the programmer a clear text to label each of the objects used in the program. Following are 50 additional text labels used to display on the LCD the tagged names of the remote points. This text is also available when reading this remote points via the BACnet protocol.

The LCD operator displays up to eight alarms that can be shown, and the text that is to be displayed is stored in this section of the database.

The following is a historical LOG event of 200 registers that keeps track of events such as: power off, power on, SMS messages sent, battery failure, clock setting etc.

Then comes a section that keeps information of up to 400 schedules, divided into 200 mainly targeted as lighting schedules, and 200 for general scheduling use.

The remote (RMT) point configuration is stored also in EEPROM whereas the remote current value is stored in RAM.

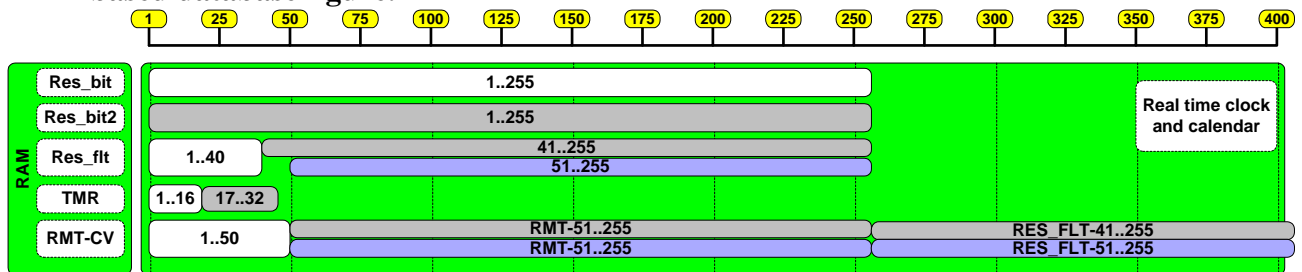
At the end of the EEPROM parameters comes the graphic configuration and the graphic data is also stored here. For each graphic, up to 246 samples can be stored, with the addition of the OpenBAS-ACC-EE256 256K Bytes external EEPROM up to 1000 samples can be stored for each of the 16 graphics.

If a Dual Core processor is installed also the following information is stored without restrictions of size, as it is only limited by the USB memory installed.

- Unlimited storage of the 16 basic graphics.
- Unlimited storage of the 256 remote points and the 256 result float registers.
- Unlimited storage of the 40 Binary Inputs, 40 Binary outputs.
- USB insertion and extraction log.

All configuration pertaining the controller configuration and the Fieldbuses are stored in this EEPROM area.

RAM based database figure.



The **RAM** section of the database is backed up by a rechargeable battery that when fully charged can keep this information stored for up to seven days. After the battery fully discharges, all the registers are set to ZERO on power up. The RTCC (Real Time Clock and Calendar) is backed up every 15 minutes in EEPROM while the controller is energized, so in the event the device is powered off for more than seven days and the battery fully discharges, the contents of the clock at the moment of the last power off will remain stored for reference.

In the RAM are stored values that change frequently. The EEPROM has the limitation that the values can only be changed 1,000,000 times, after this the EEPROM cells wear out and the data cannot longer be stored. Whereas the RAM has unlimited read / write cycles but must be battery backed up.

The first objects stored in RAM are 255 RES_BIT (Result Bit) registers that keep results of binary operations and can only be 0 / 1. If the Dual Core is installed 255 additional RES_BIT2 registers exist, but because they are inside the Dual Core and its RAM has no battery backup, all values are set to ZERO on a power failure.

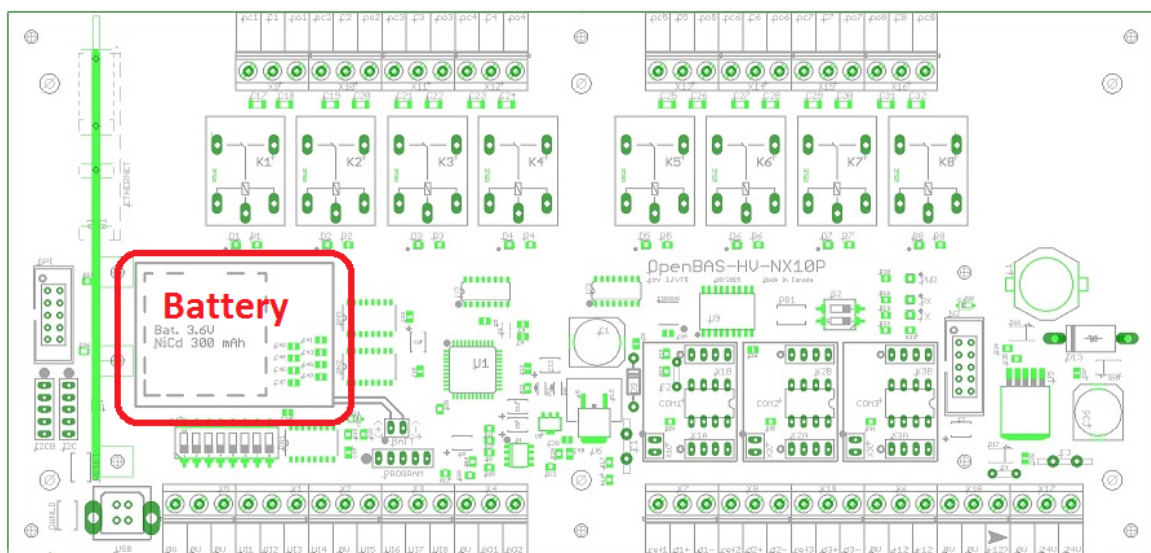
Then comes a block of 40 RES_FLT (Result Float) registers, that keeps storage of operations with real values stored in IEEE 32 bit registers. Also if the Dual Core is installed, additional RES_FLT registers 41..255 are present in the Dual Core RAM and are also not battery backed up and set to ZERO on a power off event.

The controller has 16 system timers that decrement to ZERO in 1/10 of a sec or 1 second intervals depending on their setup, and this reverse counters are also stored in RAM. An additional 16 timers are present in the Dual Core RAM and are also not battery backed up and set to ZERO on a power off event.

The remote points CURRENT VALUES that are acquired via the Fieldbus communication ports are also stored in RAM, REMT-1..50 are battery backed up. RMT-51..255 and RES_FLT-41..255 that are present if the Dual Core is installed are in their RAM and are also not battery backed up and set to ZERO on a power off event. The exception to this is if instead of the Dual Core the OpenBAS-ACC-NV32K memory expansion is installed. These additional remote points CURRENT VALUES are stored in NON VOLATILE RAM and their values persist between power off events.

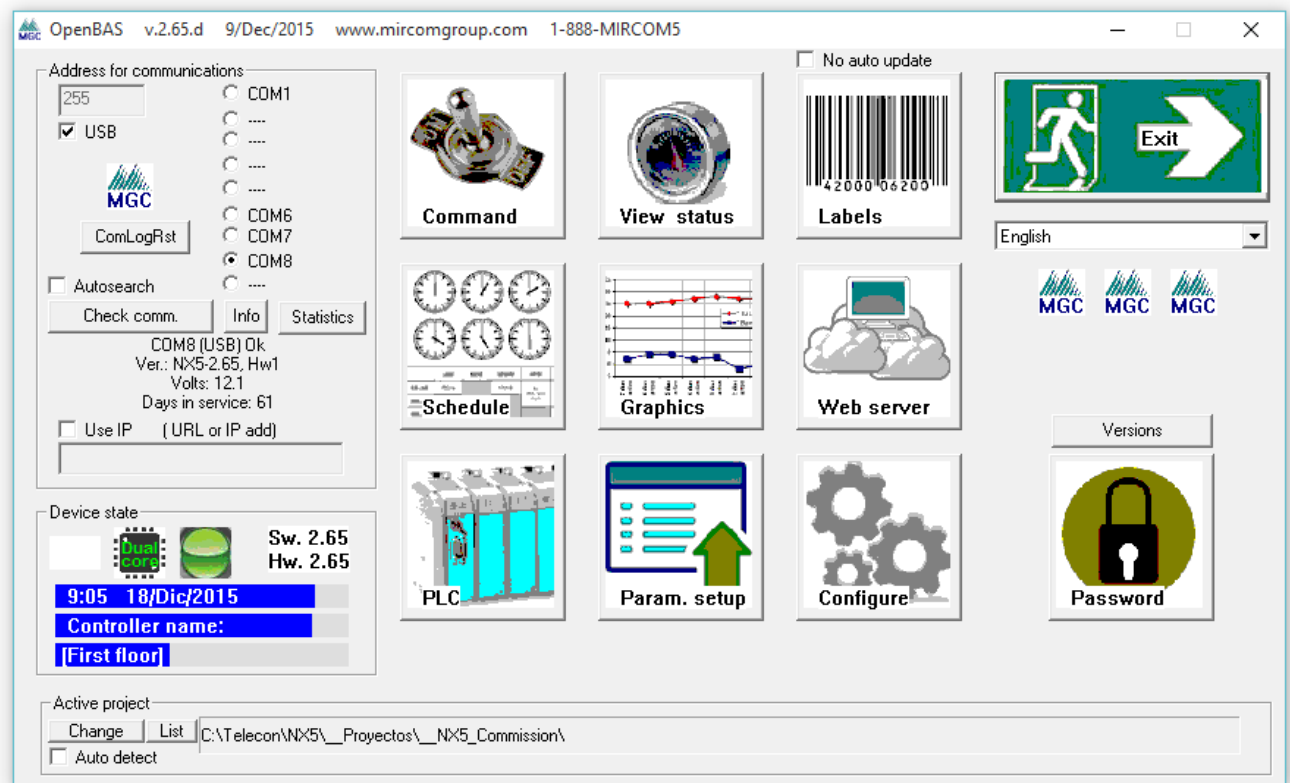
The RTCC (Real Time Clock and Calendar) is also backed up by the same battery that keeps RAM, and as the RTCC resides physically inside of the main processor and shares it's storage with the microcontroller's RAM, both memories are backed up by the rechargeable 3.6 Volt 300 mA/h rechargeable battery.

The battery is only needed by master controllers, slave controllers usually don't need the battery installed, unless they also act as masters in any of their Fieldbuses or have a PLC program that depends on schedules. The following picture shows the location of the battery on the controller board.

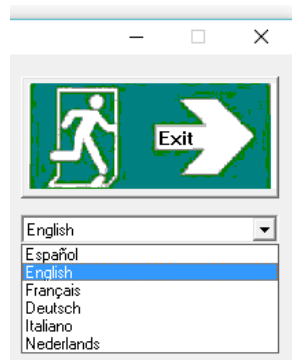


1.2 OpenBAS Configuration Tool Description

To program the OpenBAS-HV-NX10P series of controllers a companion software is available. The **OpenBAS-SW-CFGTL** configuration tool allows to fully program every detail of the controller, to perform the duty it is assigned to do. The following image shows the program's main screen, in the following sections each of the main components will be described in full detail.

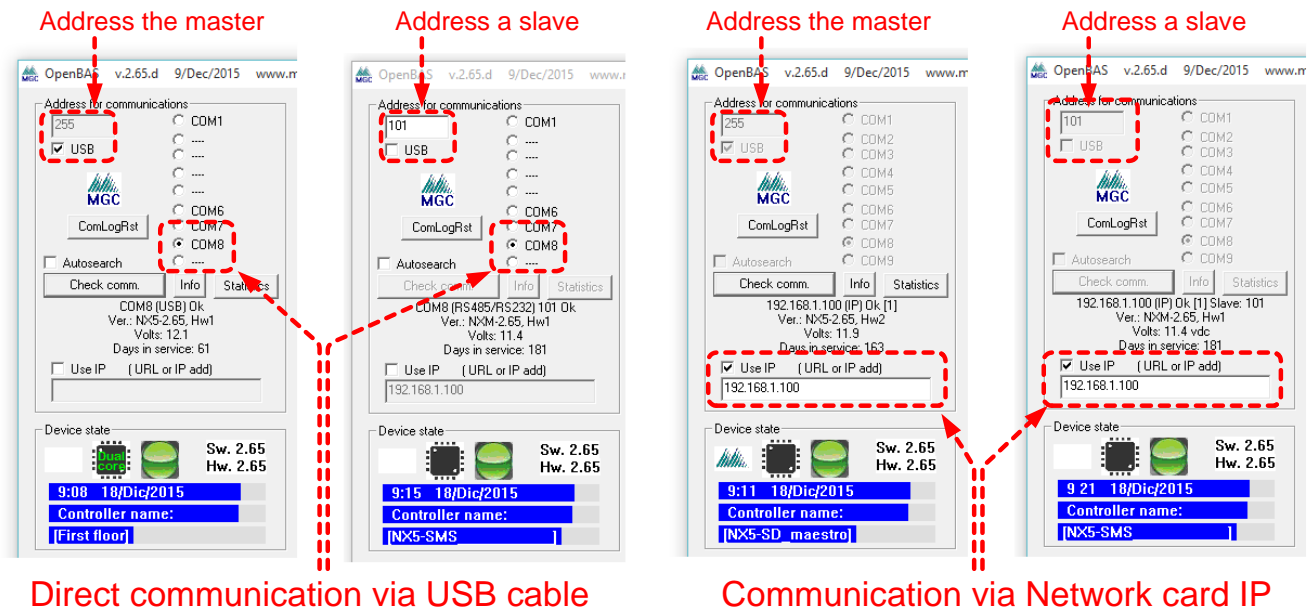


The configuration tool currently supports six languages and by selecting the dropdown list, the interface of the program is automatically changed to the selected language.



1.3 Communication Setup

The first section of the configuration tool is the communication section where the method for connecting to the controller must be selected.



The first step is to select if we want to communicate with the master controller or with any of the slaves connected thru the master's Fieldbuses. By selecting the USB checkbox, the address is changed to "255" and now communication is with the MASTER (or the device physically attached to the other end of the USB cable) this is called DIRECT CONNECTION.

If the USB checkbox is de-selected, then the address field is enabled, and the address of the slave connected to the master's Fieldbus will be now addressed. For the software to be able to communicate to a slave the following must be previously configured:

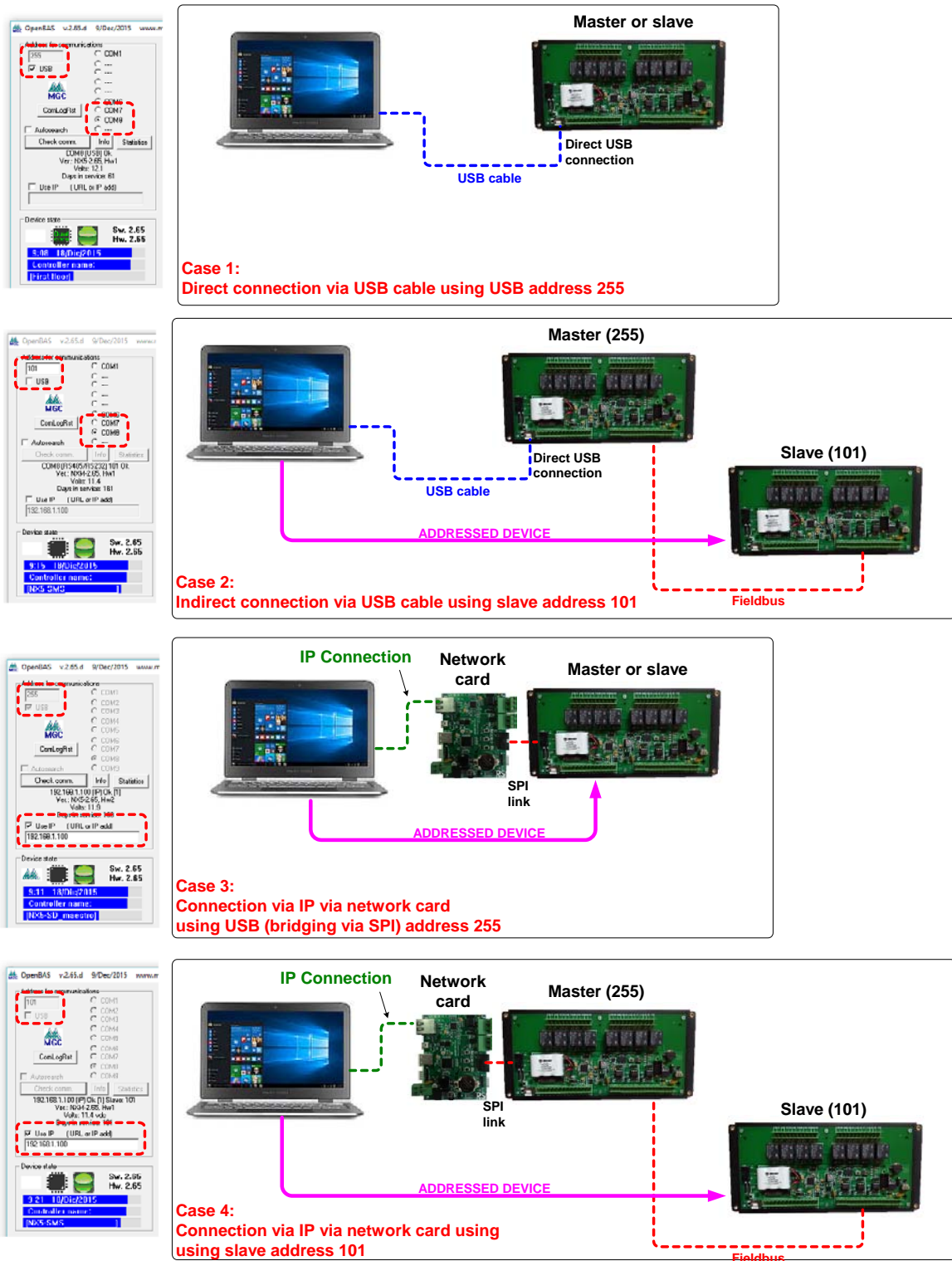
- The Fieldbus of the master must be set as master with the correct settings of protocol and baud rate.
- The remote points must be added in the master, so it knows which master it must look for.
- The slaves must be set accordingly and connected in the network and be powered.

Once the slaves come "online" in the master, they can be accessed indirectly via the master as if they were connected directly using the USB port.

See the following sections for more information on how to set up slaves:

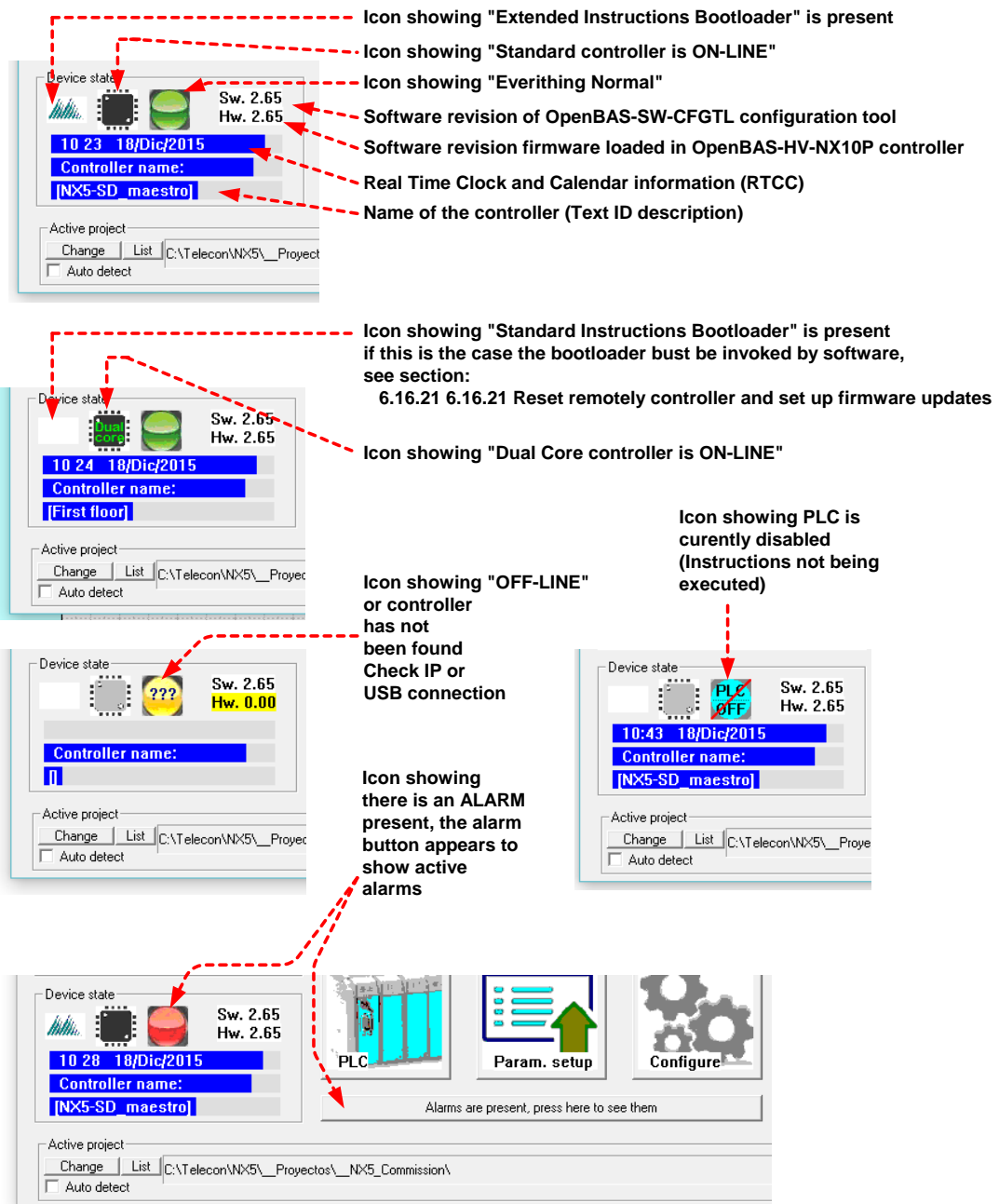
- 6.16.1 Configure the Fieldbus Communication Ports
- 6.16.2 Set up Remote Points in Fieldbuses

The following diagram illustrates the connections made with different selections:



1.4 Device Status

On the bottom left of the screen, the DEVICE STATE group gives graphical information of the current status of the device being addressed.



Example 1: Normal Status

- Icon showing "Extended Instructions Bootloader" is present
- Icon showing "Standard controller is ON-LINE"
- Icon showing "Everything Normal"
- Software revision of OpenBAS-SW-CFGTL configuration tool
- Software revision firmware loaded in OpenBAS-HV-NX10P controller
- Real Time Clock and Calendar information (RTCC)
- Name of the controller (Text ID description)

Example 2: Dual Core controller is ON-LINE

- Icon showing "Standard Instructions Bootloader" is present if this is the case the bootloader must be invoked by software, see section: 6.16.21 6.16.21 Reset remotely controller and set up firmware updates
- Icon showing "Dual Core controller is ON-LINE"

Example 3: OFF-LINE

- Icon showing "OFF-LINE" or controller has not been found. Check IP or USB connection

Example 4: Disabled PLC

- Icon showing PLC is currently disabled (Instructions not being executed)

Example 5: Alarm Present

- Icon showing there is an ALARM present, the alarm button appears to show active alarms

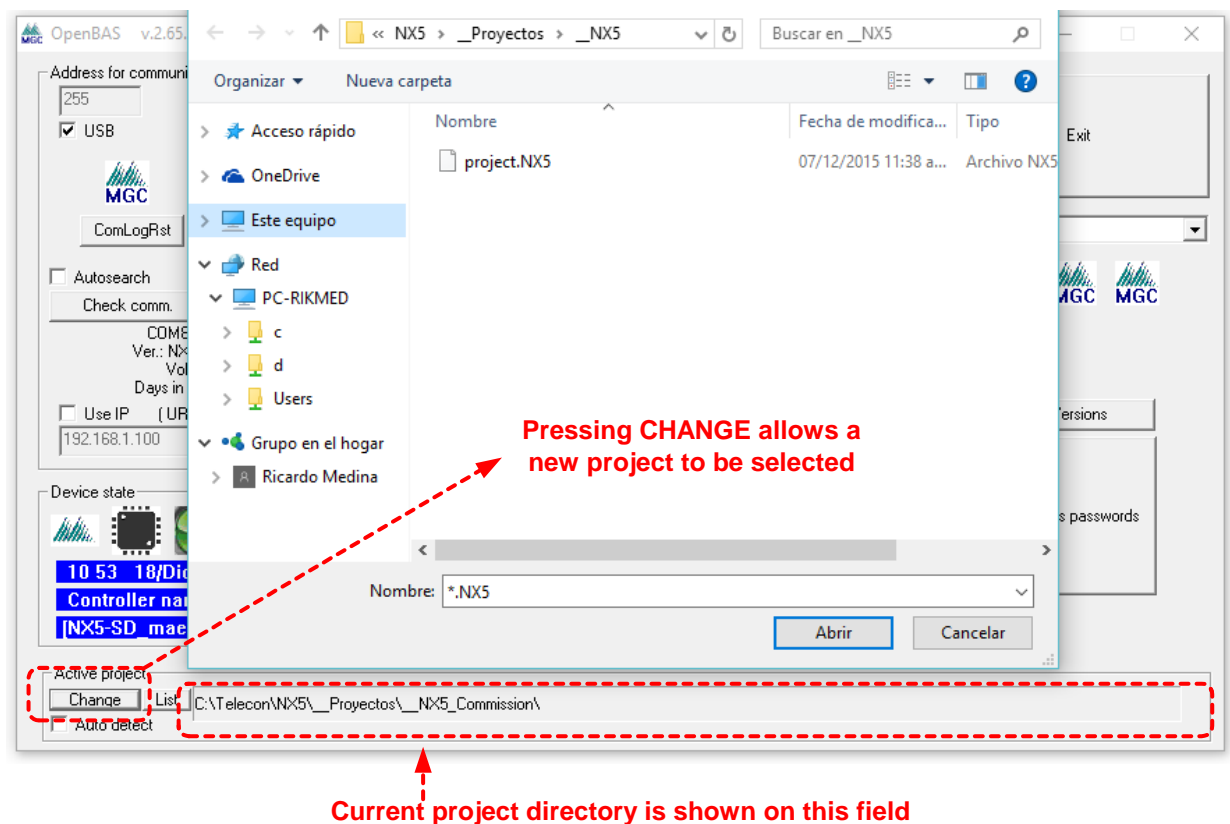
Bottom Panel:

- PLC
- Param. setup
- Configure
- Alarms are present, press here to see them

1.5 Active Project Selection

On the bottom of the program screen is a section called ACTIVE PROJECT, when interacting with the OpenBAS-HV-NX10P controller the LIVE or ON-LINE database is always being accessed.

When there is the need to save the current changes to disk, a project must be selected where the files that contain the changes made online can be stored and later retrieved for restoring if needed.



There are two buttons: CHANGE that allows the user to select a new existing project and LIST that brings a list of most recently used projects.

To create a new project, refer to section:

- 6.16.19 Create backups or restore all your Building Automation Controller programming

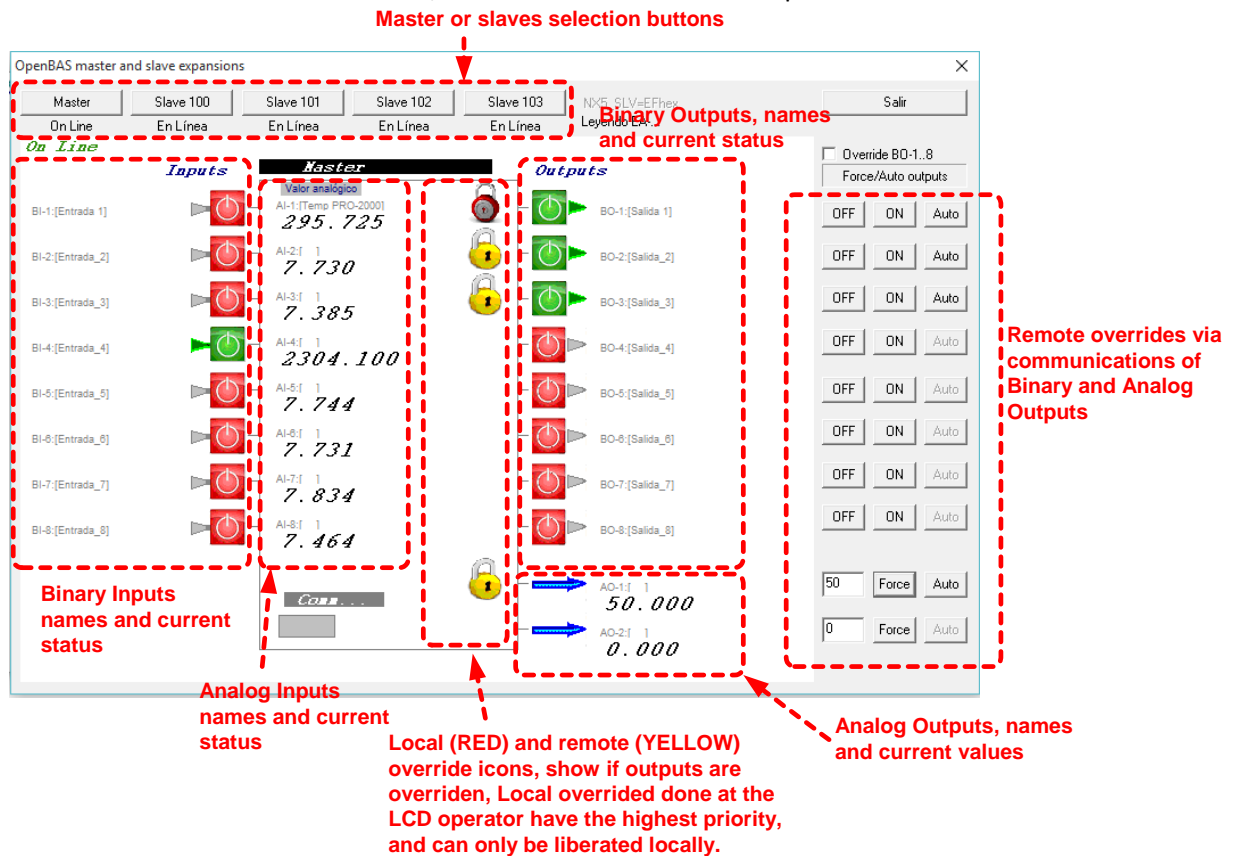
1.6 View Local and Slaves Inputs and Outputs and Manual Output Commanding

When on the main screen, pressing the COMMAND button, opens a screen where the status and values of all the HARDWARE database I/O can be viewed and the outputs commanded or overridden.



When on the main screen, pressing the COMMAND button opens a screen where the status and values of the entire HARDWARE database I/O can be viewed and the outputs commanded or overridden.

Master or slaves selection buttons



Binary Inputs names and current status

Analog Inputs names and current status

Binary Outputs, names and current status

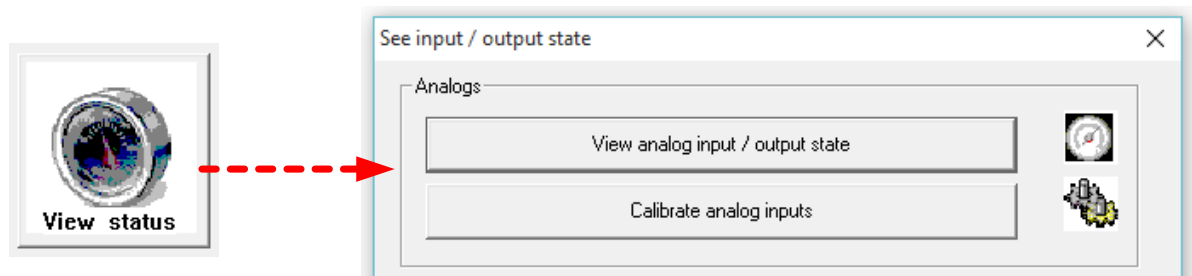
Analog Outputs, names and current values

Remote overrides via communications of Binary and Analog Outputs

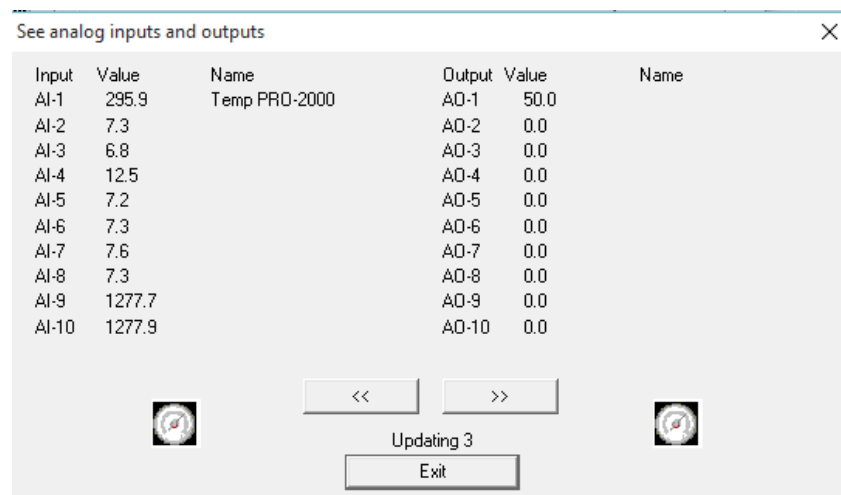
Local (RED) and remote (YELLOW) override icons, show if outputs are overridden, Local overridden done at the LCD operator have the highest priority, and can only be liberated locally.

1.7 Analog I/O Viewing and Setup Type and Calibration

When on the main screen, pressing the VIEW STATUS button opens a screen where the current value of the analog inputs can be viewed and the analog inputs can be configured and calibrated.



The actual status of the Analog Inputs AI-1..40 and the analog outputs AO-1..10 can be viewed along with any label tag that has been setup. The two buttons at the bottom allow to visualize the >> following 10 or the << previous 10.



Input	Value	Name	Output	Value	Name
AI-1	295.9	Temp PRO-2000	AO-1	50.0	
AI-2	7.3		AO-2	0.0	
AI-3	6.8		AO-3	0.0	
AI-4	12.5		AO-4	0.0	
AI-5	7.2		AO-5	0.0	
AI-6	7.3		AO-6	0.0	
AI-7	7.6		AO-7	0.0	
AI-8	7.3		AO-8	0.0	
AI-9	1277.7		AO-9	0.0	
AI-10	1277.9		AO-10	0.0	

At the bottom of the window, there are navigation buttons: '<<' and '>>', a status indicator 'Updating 3', and an 'Exit' button. Small gauge icons are also present on the bottom left and right.

To select the type of analog input and adjust the calibration select the button: CALIBRATE ANALOG INPUTS, in the screen that we see on the next page, select the Analog Input number, the type and the value that indicates the multiplier / offset is the calibration value can be adjusted.

If the selected type is of a resistive temperature sensor type or thermocouple, the calibration value will be an offset that will add or subtract (in case of being negative) to the current value. For the other types, it will be a multiplier to convert the value of input to a real value.

IMPORTANT, if the input type is resistive 1000 ohms in Fahrenheit degrees or Celsius, you should set the corresponding DIP-SW to the ON position so that it reads the value correctly. For all the other types the DIP-SW must be in the OFF position.



For example, for a voltage signal of 0-10 volts that represents a pressure of 0-250 PSI, the value of the multiplier will be of 25. So when reading the control 10 volts, it will multiply it by 25 and the showed value will be 250.

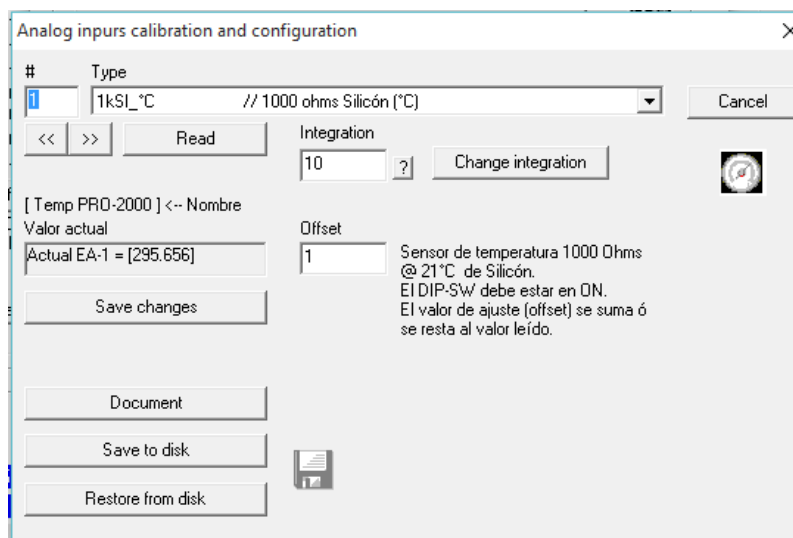
For the type of 4-20 ma. With a multiplier of ONE (1.0) the value will measure 0-100 percent of the value. If following the previous example, we use a pressure transducer in which the range of 4-20 milliamperes represents 0-250 PSI, the multiplier value must be of 2.5, being that the 100% the value when there is 20 mA. multiplied by 2.5 will give per result the correct value of 250.

IMPORTANT, if the input type 4-20 ma, between the input and the common of 0V you must install a resistor of 250 ohms @ ½ watt 1%, for the measurement to be correct, view the connection section for the wiring detail.

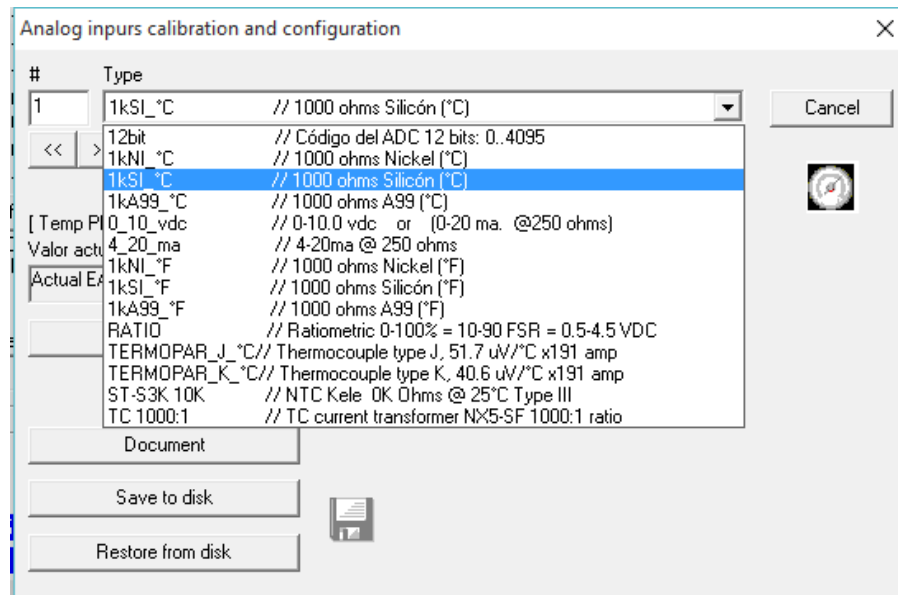
You can adjust in small fractions using the rule of three between the current value and what you read with an accurate reference if you want to further calibrate the measurement.

It is also a good practice to avoid mistakes, to put the multiplier initially on ONE (1.0), then calculate the ratio of value of the multiplier by dividing the given read value with the current measured value.

The buttons  and  allows us visualize the previous or the following input, or can key in directly the desired input number to see, and select the button READ.



After selecting the Analog Input type of the list, or changing the calibration value, select SAVE to send the changes to the controller.



You can save the configuration to the PC to **BACK UP** or to transfer to another control. The adjusted values are stored in the archive **CALIB_AI.INI**. Don't modify this file manually,

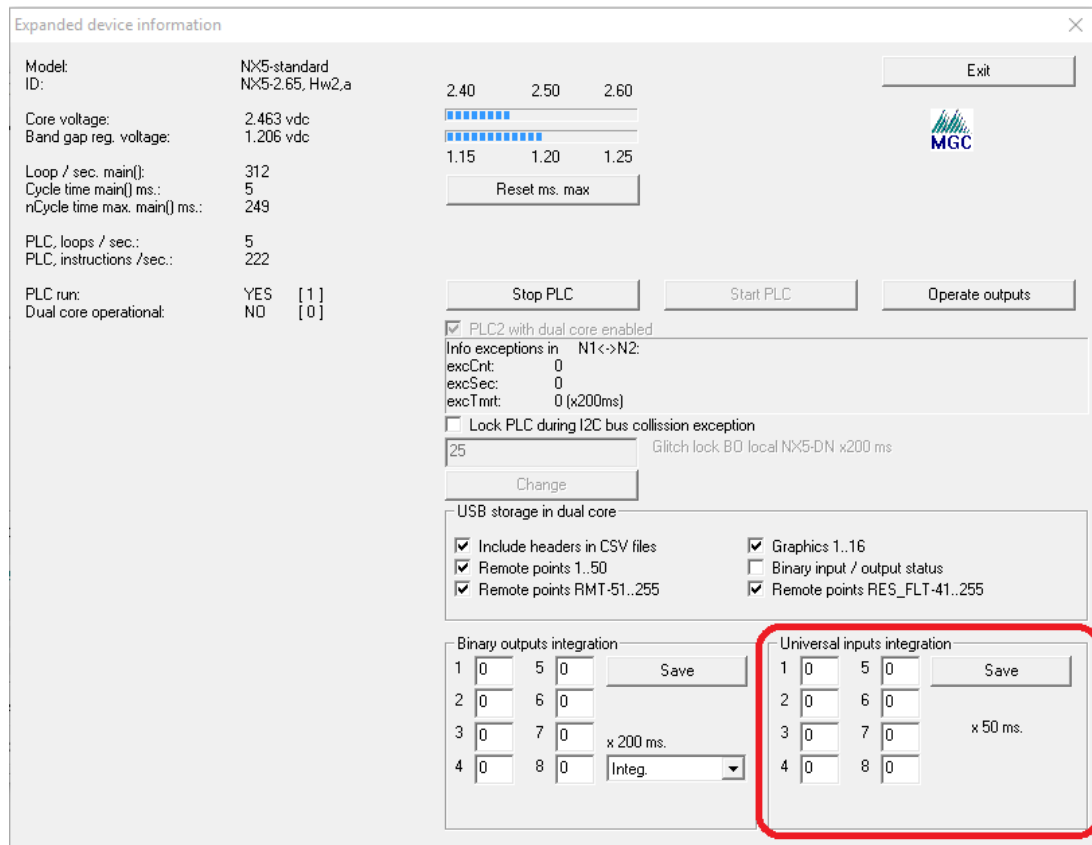
In the **SYSTEM ADJUST** section there is an option to modify the **INTEGRATION TIME** for the analog inputs. Increase it if the measurement varies due to the nature of the process or due to interferences in the field, or reduce it if you need a fast response of the variable's measurement. If the integration value is **ZERO**, the measurement will be instantly with no integration at all.

The scanning time of the Analog Inputs is **200** milliseconds. With that they will be able to have five readings per second. If the process variable varies too quickly, the integration range of **0..250** (1/5 of second) will give an integration time of **ZERO** up to **50 seconds**.

It is recommended to use integration values of 10 seconds or higher for HVAC process like pressure controls loops.

There is a new feature by pressing the INFO button on the main screen that the individual integration time for each of the Analog Inputs can be set. See the picture in the following page.

Here the integration time can be set individually per Analog Input, and a resolution of 50 milliseconds is available.



Expanded device information

Model: NX5-standard
 ID: NX5-2.65, Hw2,a

Core voltage: 2.463 vdc
 Band gap reg. voltage: 1.206 vdc

Loop / sec. main(): 312
 Cycle time main() ms.: 5
 nCycle time max. main() ms.: 249

PLC, loops / sec.: 5
 PLC, instructions / sec.: 222

PLC run: YES [1]
 Dual core operational: NO [0]

2.40 2.50 2.60
 1.15 1.20 1.25
 Reset ms. max

Stop PLC Start PLC Operate outputs

☒ PLC2 with dual core enabled

Info exceptions in N1<->N2:
 excCnt: 0
 excSec: 0
 excTmit: 0 (x200ms)

☐ Lock PLC during I2C bus collision exception
 25 Glitch lock B0 local NX5-DN x200 ms
 Change

USB storage in dual core

☒ Include headers in CSV files
☒ Remote points 1..50
☒ Remote points RMT-51..255

☒ Graphics 1..16
☐ Binary input / output status
☒ Remote points RES_FLT-41..255

Binary outputs integration

1	0	5	0
2	0	6	0
3	0	7	0
4	0	8	0

x 200 ms.
 Integ. ▼

Universal inputs integration

1	0	5	0
2	0	6	0
3	0	7	0
4	0	8	0

x 50 ms.
 Save

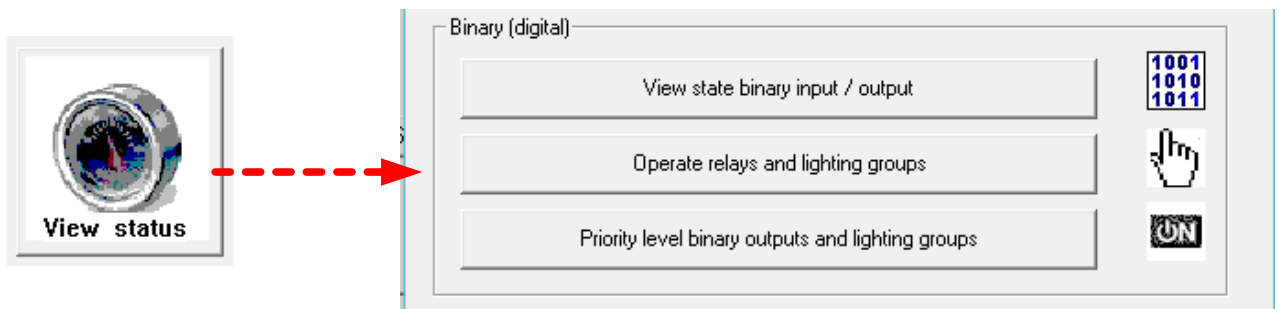
As this is a software filtering feature available only to local hardware Analog Inputs, only AI-1..8 can be adjusted, whereas the setting on the previous page applies to all Analog Inputs AI-1..40.

The setting on this screen has precedence over the general setting. For example the value set here is ZERO, the AI will use the general integration value, but if set with a value between 1..255 then the AI-1..8 with that non ZERO value will use the new integration constant.

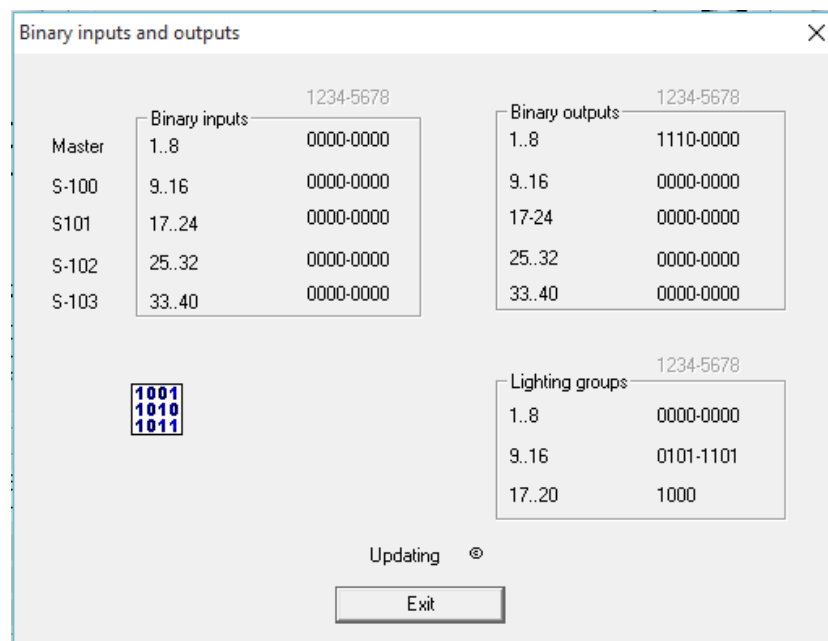
This can help to filter a specific Analog Input that is affected by noise without affecting the setting of the rest of the Analog Inputs.

1.8 Digital I/O Viewing and Commanding

When on the main screen, pressing the VIEW STATUS button opens a screen where the current status of the Binary Inputs and Outputs can be viewed. Also there is a second button to command each Binary Output individually, in groups or all at the same time. The third button shows the priority level that the Binary Outputs are commanded to.



A view that shows all the 40 Binary Inputs and the 40 Binary outputs plus the additional virtual 20 Binary Outputs listed as Lighting Groups 1..20 are shown. The current values can be: 1/0.



To operate the binary outputs, the OPERATE RELAYS AND LIGHTING GROUPS button opens up a dialog to allow the user to do the following:

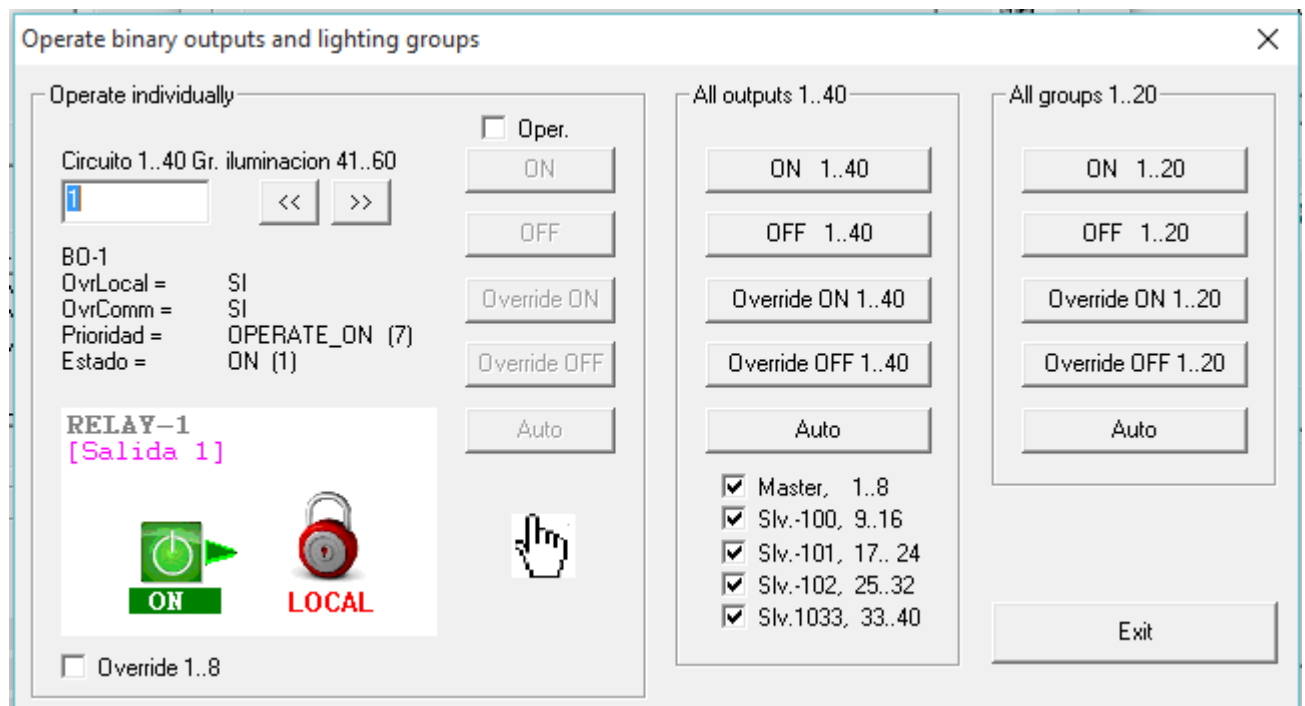
Command a CIRCUIT (relay) or all the physical circuits BO-1..40 or all the lighting groups (virtual relays) BO-41..60.

In the section of INDIVIDUAL command, there is a box where we can put the circuit number between the 1..60 to command, the buttons << y >> allows us to navigate between this range.

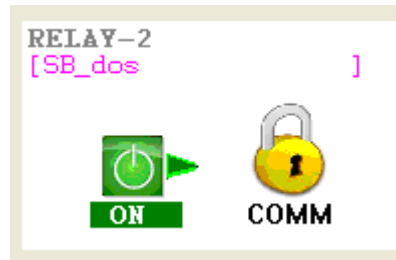
Bellow we can see its current state and if it has an override (override).

The priority level of the relays from **HIGHER** to **LOWER** is the following:

- LOCAL OVERRIDE Override done from the LCD operator keyboard.
- OVERRIDE COMM Override remote by USB, COM1, COM2 or Ethernet.
- LOGIC_XX Command to ON/OFF from some logic block of the program.
- SCHEDULE / COMM / OPER. Has the same internal priority, and is the lowest of all.



Communication overrides are shown with a yellow lock instead.



If there is a **LOCAL OVERRIDE**, the command buttons ON/OFF and of OVERRIDE COMM and RELEASE will be disabled. The only way of liberate them will be locally in the LCD operator display.

If there is an **OVERRIDE COMM**, you could **RELEASE** so the logic or the schedules can command it.

If there is a level of **LOGIC** command, the buttons ON/OFF that command it, will be disabled because the logic has precedence. However there is a **CHECK BOX OPERATE** to force this operation.

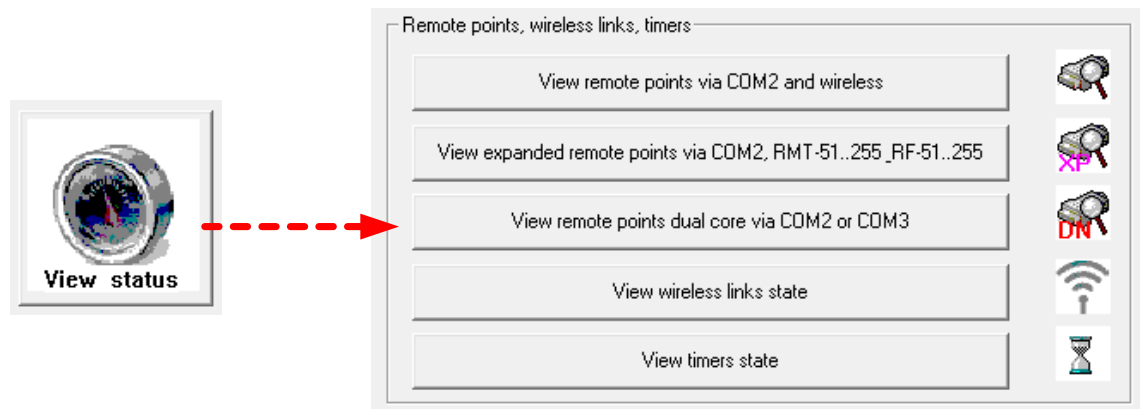
The options of **ALL THE CIRCUITS** and **all the groups** make a similar function in the relays blocks **BO-1..40** and **BO-41..60** respectively.

The third button **PRIORITY LEVEL** shows if there is any active override either LOCAL or COMM and a descriptive text of the current priority and current status ON / OFF.

Binary output priority viewer							
#	OvrLoc	OvrCom	Priority	Current state	Nombres		
1	SI	SI	OPERATE_ON (7)	ON (1)	Salida_1		
2	-	SI	OPERATE_ON (7)	ON (1)	Salida_2		
3	-	SI	OPERATE_ON (7)	ON (1)	Salida_3		
4	-	-	OPERATE_OFF (6)	OFF (0)	Salida_4		
5	-	-	OPERATE_OFF (6)	OFF (0)	Salida_5		
6	-	-	OPERATE_OFF (6)	OFF (0)	Salida_6		
7	-	-	OPERATE_OFF (6)	OFF (0)	Salida_7		
8	-	-	OPERATE_OFF (6)	OFF (0)	Salida_8		
9	-	-	OPERATE_OFF (6)	OFF (0)			
10	-	-	OPERATE_OFF (6)	OFF (0)			

1.9 Remote Points and Fieldbus Communications



When on the main screen, pressing the VIEW STATUS button opens a screen where remote points can be viewed and setup. In addition the I2C wireless adapter can be adjusted, and the last button at the bottom is to view the timer's status.



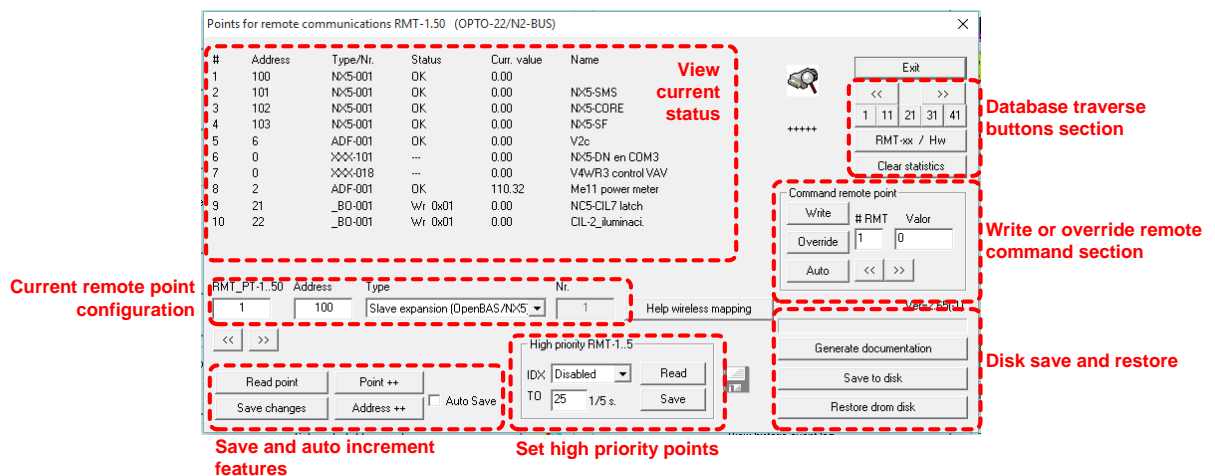
The OpenBAS-HV-NX10P has up to three Fieldbuses that can be set as masters with different protocols. The following is a list of the protocols that are available to use as master according to the Fieldbus communication port:

Port	Protocol	Type	Baud	Implemented
COM1	Modbus RTU master	RS485/RS232 single master	2,400 - 19,200	Yes
COM1	Optomux	RS485 slave	1,200 - 76-800	Yes
COM1	BACnet/MSTP master	RS485 multimaster	9600 - 38,400	Partially
COM2	ASCII	RS485/RS232 peer-peer	1,200 - 76-800	Yes
COM2	Optomux	RS485 single master	1,200 - 76-800	Yes
COM2	N2-Open	RS485 single master	9600	Yes
COM2	ECM	RS232 peer to peer	9600	Yes
COM3	Modbus RTU master	RS485/RS232 single master	2,400 - 19,200	Future
COM3	BACnet/MSTP master	RS485 multimaster	9600 - 38,400	Future
COM3	Optomux	RS485 single master	1,200 - 76-800	Yes
COM3	N2-Open	RS485 single master	9600	Yes

It is possible to create up to **50** remote points in the **NX5** (**470** with Dual Core or **460** with NVRAM memory installed) into up to 24 different hardware addresses (HW REMOTE).

In the following picture we can see how they are visualized 10 by 10, you can easily traverse the next or previous 10 with the buttons  and  respectively. With the button **Points / HARDWARE** we change to view the status and actual value of the **50** remote points as well as the address status of the hardware of the **24** hardware points that can be set up. These are created automatically when you add a new point.

Depending on the use of the first 50 Remote Points whether they are: Modbus RTU or BACnet/MSTP or Optomux/N2-Open the configuration screen will change accordingly to reflect the protocol supported objects.



The following table shows the ranges set up the remote points for Optomux/N2-Open:

<u>TYPE</u>	<u>RANGE</u>	<u>HW address</u>	<u>NOTES</u>
• Analog input	EA-1..255	1..99 and 104..250	100 to 103 reserved for slaves
• Binary input	EB-1..255	1..99 and 104..250	100 to 103 reserved. for slaves
• Analog output	SA-1..255	1..99 and 104..250	100 to 103 reserved. for slaves
• Binary output	SB-1..255	1..99 and 104..250	100 to 103 reserved. for slaves
• Float data	ADF-1..255	1..99 and 104..250	100 to 103 reserve. for slaves
• Word data	ADI-1..255	1..99 and 104..250	100 to 103 reserved. for slaves
• Byte data	ADB-1..255	1..99 and 104..250	100 to 103 reserved. for slaves
• NX slave	NX-X-1..4	100 to 103	DIP_SW 00,01,10,11
• Wireless	WLS-1..130	254	View mapping in next table

For wireless points, we can set up up-to **10** wireless links, to see each one, and the points of each wireless transmitter. The number of the wireless mapping point will be:

<u>VARIABLE</u>	<u>RANGE</u>	<u>NOTES</u>
• Degrees C	WLS-1..10	The first one corresponds to the first link and the 10 to the tenth wireless link.
• Degrees F	WLS-11..20	
• Humidity %HR	WLS-21..30	
• Mode	WLS-31..40	
• Fan	WLS-41..50	
• Keyboard	WLS-51..60	
• Sp_Temp	WLS-61..70	
• Sp_Hum	WLS-71..80	
• Sp_T1	WLS-81..90	
• Sp_Pb	WLS-91..100	
• Sp_Unocc	WLS-101..110	
• Voltage_Batt	WLS-111..120	
• An_Inp	WLS-121..130	

At the bottom of the dialog box, we can see the buttons to **BACK UP** the current database configuration to the hard drive to the archive **REMOTE.INI** and can also **RESTORE** them to the same or to another OpenBAS-HV-NX10P control.

- **High priority points:**

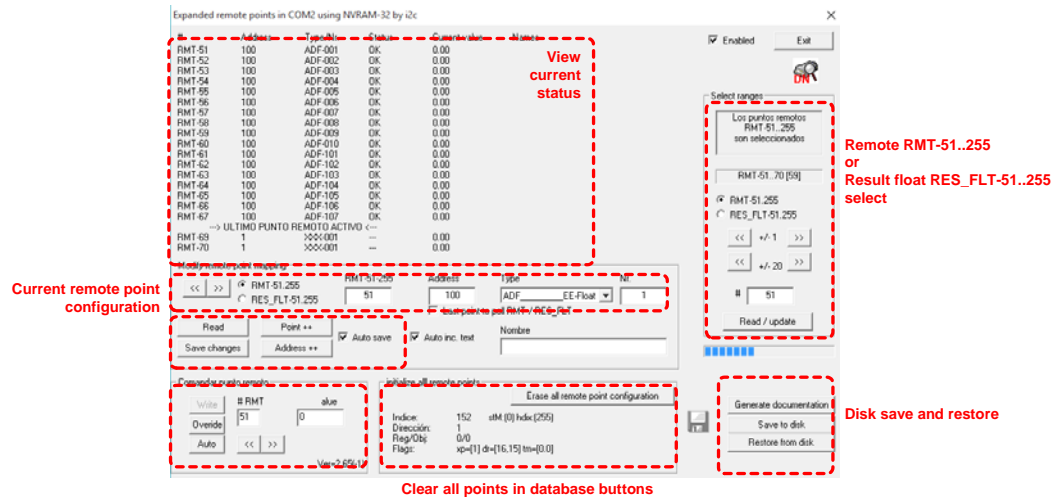
The high priority option is to program on the remote points 1-10 preferably the slaves with the addresses 100..103. This ensures that even if all the points are set up including the 205 additional of the Dual Core. These high priority points will be refreshed at last every 0.5-10 seconds as set the K of refresh TO.

- **Virtual port function**

The OpenBAS-HV-NX10P functions as a virtual port or converter of USB to RS-485. Just set up a point, and the software of this equipment can be accessed as if it were installed through a standard serial port, or from a USB converter to RS-232 + a converter of RS-232 to RS-485.

If the NVRAM memory expansion is installed, any standard OpenBAS-HV-NX10P controller can have access to 410 additional points (RMT-51..255 and RES_FLT-51.255) to give the database a total of 460 remote points with the base RMT-1.50 points included.

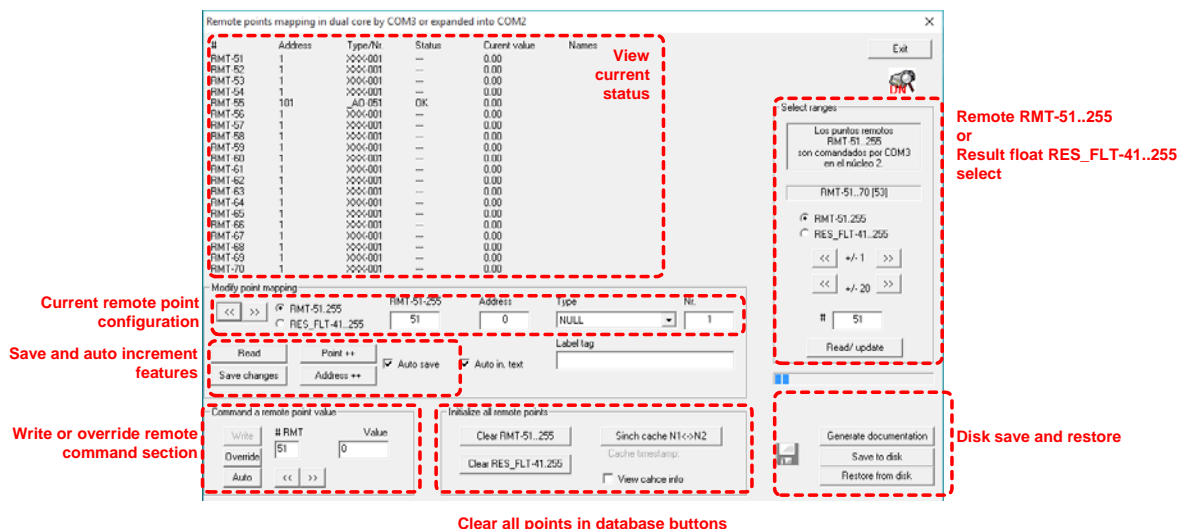
The following picture shows the configuration screen for the expanded remote points when using the NVRAM expansion memory.



If instead of the NVRAM expansion memory, a Dual Core is installed, it adds 420 additional remote points (RMT-51..255 and RES_FLT-41..255) to give the database a total of 470 remote points with the base RMT-1..50 points included. The Dual core has the added capability of enabling the third Fieldbus COM3, and the database can be further split in any of the following combinations:

Port	Protocol	Remote Point	
COM1	Modbus RTU master	RMT-1..50	If COM1 is selected as master of RMT-51..255 then COM2 will only be able to manage RMT-51..255, but only if it is set up as master of them.
COM1	BACnet/MSTP master	RMT-1..50	
COM2	Optomux	RMT-1..50 + RMT-51..255	If COM2 is selected as master of RMT-51..255 then COM2 will manage all remote points RMT-1..50 + RMT-51..255 and COM3 will only manage RES_FLT-41..255.
COM2	N2-Open	RMT-1..50 + RMT-51..255	
COM3	Modbus RTU master	RMT-51..255 and/or RES_FLT-41..255	If COM3 is selected as master of RMT-51..255 it will also manage RES_FLT-41..255 and COM2 will only manage remote points RMT-1..50
COM3	BACnet/MSTP master	RMT-51..255 and/or RES_FLT-41..255	
COM3	Optomux	RMT-51..255 and/or RES_FLT-41..255	
COM3	N2-Open	RMT-51..255 and/or RES_FLT-41..255	

The following picture shows the configuration for the additional points in the Dual Core.



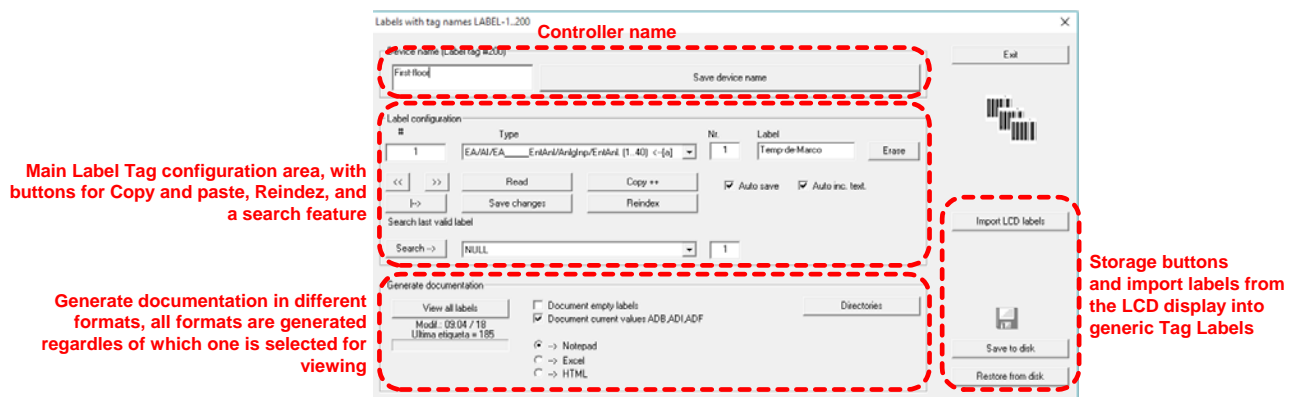
1.10 Labeling and Tagging Hardware and Software Objects

When on the main screen, pressing the LABELS button, opens a screen where the Label tags can be edited.



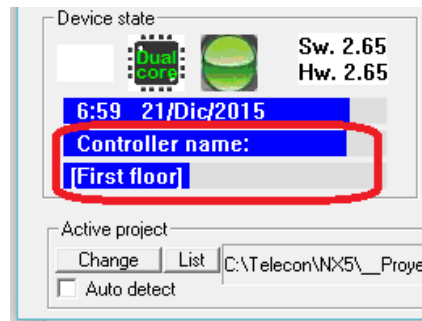
The use of label Tags is to enforce program order and readability. As programs grow it is easier to reference them with a meaningless name such as: “Zone Temperature”, “Water Pump #1”, instead of object names such as “Analog Input 1!, Binary Output 3” etc.

The OpenBAS-HV-NX10P controller provides 200 tags to name most of objects in the database. These tags are used elsewhere in the program, so when you open the different windows to view the database, modify values and parameters, document your program, activate the virtual web server, etc. These Label Tags provide additional information to better understand the program.

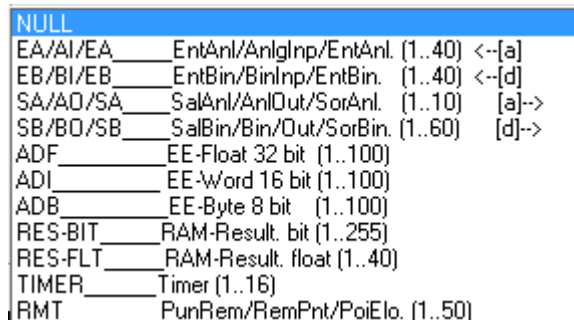


As shown in the above picture, the first section controls the name of the controller itself. The last label #200 is a placeholder to store the controller’s name, to provide it a meaningless name such as: “First Floor HVAC”, Mechanical Room 1” etc. This name will always be shown when on the main screen as can be seen in the next screen shot.

Picture showing information of controller name on the lower left corner of the main screen inside of the “Device State” section.



The variable type has to be introduced and in the LABEL field in which you can set any name up to 18 characters. If you want to delete a Label Tag simply undefined its type with NULL as shown on the image above.



There are several options to copy and paste with and increment which saves a lot of time when inputting consecutive tags. Sorting them with the “Re-index” feature is very important if you want the Label Tags to be visible on the LCD screen, they must be sorted after any changes are applied. There is an option to check if a database object is already defined with the “Search →” feature.

At the right side we can see the buttons to BACK UP all the labels to the hard drive to the archive LABELS.INI and can RESTORE THEM to the same or to another control.

At the bottom of the dialog there are several options to document all labels, using different file formats such as Notepad, Excel or HTML web pages.

1.11 Scheduling for Lighting and General Purpose Uses

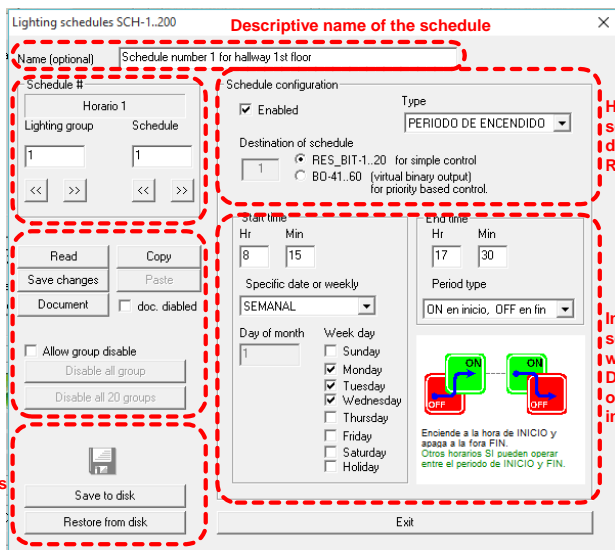
When on the main screen, pressing the SCHEDULE button, opens a screen that allows up to 400 schedules to be edited.



Scheduling is a very important feature in any automation controller, and the OpenBAS-HV-NX10P has enough schedules for any automation sequence you will ever need, as each master or slave that is added to the system provides up to 400 schedules. They are further divided into 200 Lighting and 200 General schedules.

CONFIGURE LIGHTING SCHEDULES

The first 200 schedules are mainly targeted for Lighting, but that doesn't mean they can't be used for anything else. The term "Lighting Schedules" comes from the fact that they can be set up to either operate on the first 20 RES_BITS registers whose primary purpose is for lighting, but they can also be set to command the virtual Binary Outputs used for Lighting Groups (Relays 41..60). The lighting groups can use any of the **200** schedules (**10 by Lighting Group**), in the next picture we can see the fields to setup a lighting schedule:



Lighting schedules SCH-1..200

Descriptive name of the schedule

Name (optional): Schedule number 1 for hallway 1st floor

Area used to select schedule 1..200. Schedules are grouped 10 per 20 Lighting Groups, RES_BIT-1..20 control Lighting Groups 1..20 respectively.

Schedule #

Horario 1

Lighting group: 1

Schedule: 1

Read Copy

Save changes Paste

Document doc. disabled

General control buttons to operate on the schedules

Allow group disable

Disable all group

Disable all 20 groups

Storage buttons

Save to disk

Restore from disk

Schedule configuration

☒ Enabled

Type: PERIODO DE ENCENDIDO

Destination of schedule

1 ☒ RES_BIT-1..20 for simple control

☐ BO-41..60 (virtual binary output) for priority based control

Start time

Hr: 8 Min: 15

End time

Hr: 17 Min: 30

Specific date or weekly

SEMANAL

Day of month: 1

Week day

☒ Monday

☒ Tuesday

☒ Wednesday

☐ Thursday

☐ Friday

☐ Saturday



☐ Holiday

Period type

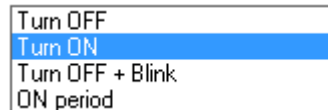
ON en inicio, OFF en fin

In this area the main information of the schedule such as if it's going to be a weekly or specific date schedule. Days that it will operate and the start and optionally end times, which must be input in 24 Hr format between 0..00 and 23:59

Enciende a la hora de INICIO y apaga a la hora FIN. Otros horarios SI pueden operar entre el periodo de INICIO y FIN.

At the top of the image on the previous page, we can see the fields to input the **GROUP-1..20** and **SCHEDULE-1..10**, and also we have the buttons  and  to navigate through each one, or simply input the Schedule number and press the button **READ**.

In the “Schedule Configuration” area there is a check box named: **ENABLED**, with which every schedule can be enabled or disabled. It is followed by a **TYPE** selector drop box, in which we can select:



In the field **DESTINATION OF SCHEDULE** it can be selected who will receive the command when the real time clock and calendar (RTCC) matches a schedule, and it can be:

- The bit type register for Lighting Groups **RES_BIT-1..20**
- Or the virtual relays used for Lighting Groups **SB-41..60**.

NOTE:

If the control bits “**RES_BIT**” is selected, make sure that the Lighting Groups that are to be commanded, have that specific RES_BIT as their control source.

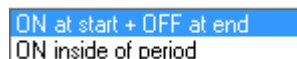
Also if the “**Binary Output**” if selected, make sure that in the Lighting Group, the correct control source is the VIRTUAL BINARY OUTPUT **BO-41..60**.

Set the “Start” and “End” **HOUR** and **MINUTES** in a 24-hour format between 0:00 and 23:59 hours.

Then select a specific date by selecting a month and a specific day of the month, or put a weekly schedule by selecting on the field of the month: **WEEKLY**. If a weekly schedule is selected, select all the days of the week in where the schedule is to be executed.

If the selected schedule type is: **ON-PERIOD**, the second field containing the **HOUR** and **MINUTES** to set the hour when the schedule will **TURN OFF** the Binary Output or RES_BIT, will be enabled.

There a field to select the period type option, that can be:



The difference between the first and the second, is that **ON AT START AND OFF AT END** will send a command to the RES_BIT or Binary Output to **TURN ON** at the start time, and **TURN OFF** at the end hour. Inside of this period, any other option like another schedule or remote command to the operator or logic functions can send a **TURN OFF** or **TURN ON** again to the controlled RES_BIT or Binary Output.

While the option **ON INSIDE OF PERIOD**, maintains the RES_BIT or Binary Output “ON” all the time that the schedule is valid. Only a **LOCAL OVERRIDE** command or **REMOTE OVERRIDE** command will have

higher priority on the Binary Outputs in case **BO-41..60** are selected as the target of the schedule. If the **RES_BIT-1..20** are selected instead; there is no way to prioritize commands.

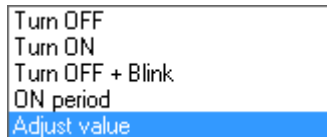
There is a check box to enable the buttons to disable the 10 schedules of every group selected, or all the schedules of all the groups. This option can be useful to disable all schedules at once, but caution must be taken on using it, as there is no option to re-enable all schedules at once, and this must be done manually.

On the bottom left of the dialog, there are buttons to save and restore all the schedules to and from the hard disk to the file: **HOR_ILUM.INI**.

CONFIGURE GENERAL SCHEDULES

In addition to the 200 lighting groups, there are other 200 schedules, called GENERAL SCHEDULES, that can also be used to command Lighting Groups or Binary Outputs, but have the added feature that they can also be used to set a value of a variable at a specific time, using the type ADJUST VALUE.

This is very useful for example changing the set point of a temperature to a different value according to the time of day.

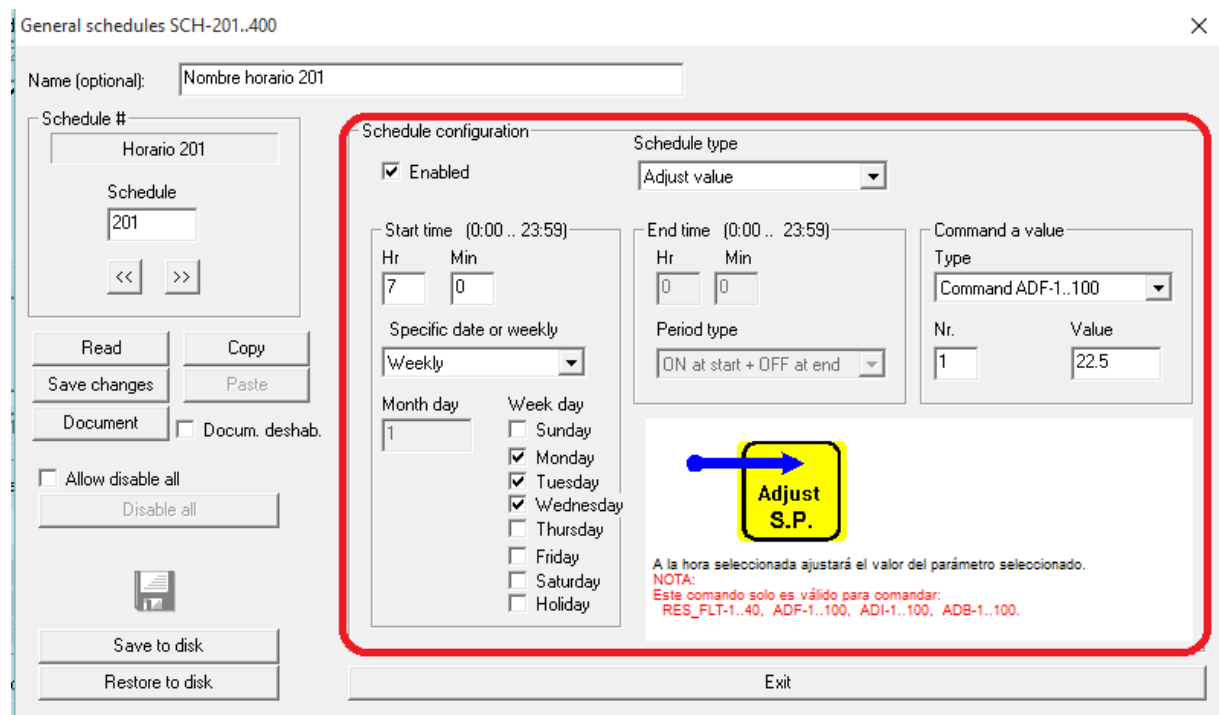


The programming of the General Schedules is similar to the Lighting Schedules, with the difference that if the adjust value is selected, the value to be commanded, must be input in the entry field called **COMMAND A VALUE** besides the time at which this new value will be set to the target database register.

For the binary outputs **BO-1..60** and **RES_BIT-1..255** we must only use the types: **TURN ON, TURN OFF or TURN ON PERIOD** only.

In the example on the following can be seen how to use this schedules.

In this example we can see how to set up the value of register ADF-1 to change by a schedule to a new value of 22.5 at 7:00 every Monday, Tuesday and Wednesday.



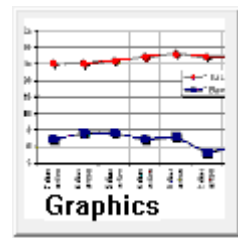
Any combination of schedules and values can be used, when many schedules are set, the DOCUMENT feature can be very handy to obtain a list of all schedules as seen on the next figure.

```
=====
[ GENERAL SCHEDULES 201..400 ]
=====
```

Nr.	h/d	hh:mm a	hh:mm	Day/Month	Command
#201	[1]	07:00		[LMM-----]	ADF-1 (22.500)
#202	[1]	07:15		[---JV---]	ADF-1 (21.300)

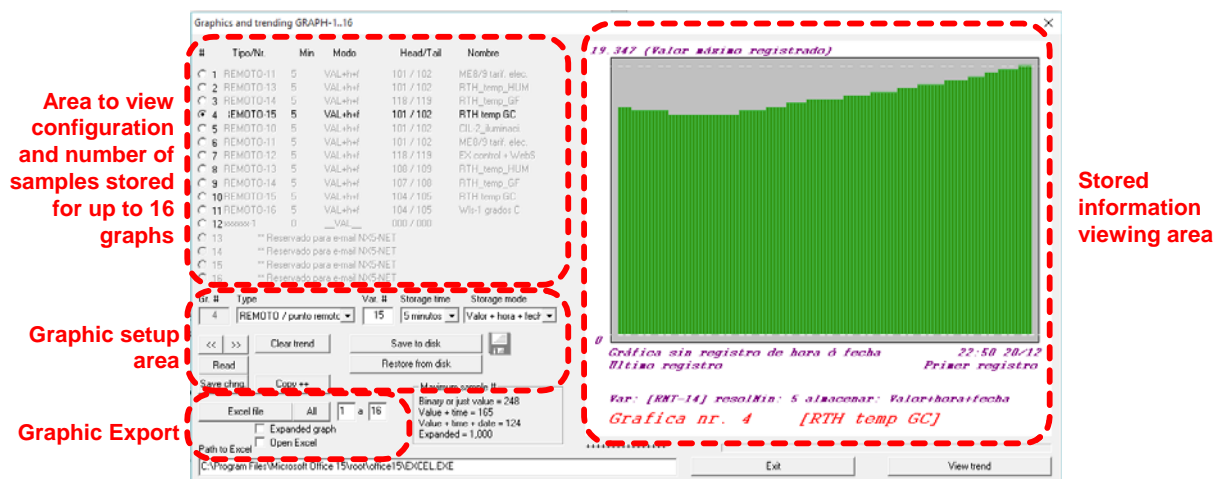
1.12 Data Logging and Graphics

When on the main screen, pressing the GRAPHICS button, opens a screen where trending information is set and can be viewed and exported.



A powerful feature of the OpenBAS-HV-NX10P, is it's graphing and logging capability for process variables, and the capability to export this data directly to **EXCEL** with a minimum user intervention. This allows you to keep track of any process that is running in the automation controller.

As it can be seen at the following figure, up to 16 different graphics can be set per each OpenBAS-HV-NX10P controller, and each slave adds 16 more graphics to the system, so a master with its 4 remote expansions can have up to a total of (16 x 5) 80 graphics.



Each graphic can store from 124 samples up to 248 depending the time stamp information set, which can be: **VALUE** or **VALUE + HOUR** or **VALUE + HOUR + DATE**.

The graphic resolution can be set to: **1, 5, 10, 15, 20, 30 or 60 minutes**.

The changes to the graphics configuration can be **BACKED UP** to the file **GRAFICAS.INI** and be **RESTORED** later again when necessary.

It is important to set the path to where EXCEL.EXE is located, so the software can load it after creation of the exported files.

Path to Excel

C:\Program Files\Microsoft Office 15\root\office15\EXCEL.EXE

Even if EXCEL is not found, the generated files will be stored in the path of the currently active project. Use the SEARCH option in WINDOWS to find this path to EXCEL.

Generated GRAFIC*.CSV files will be stored in the current project directory

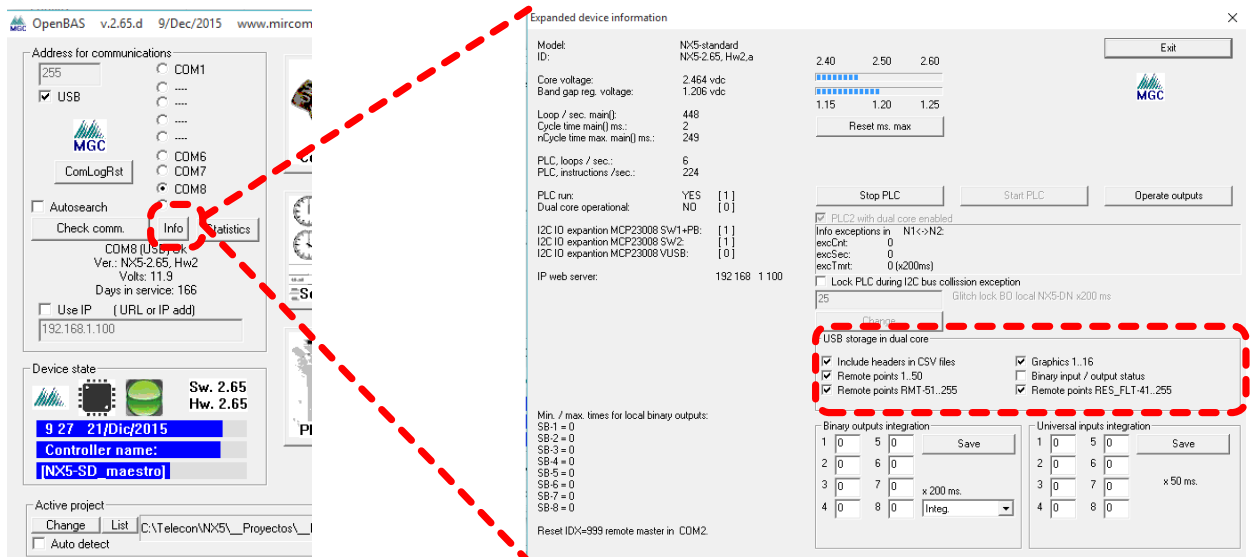
Active project

Change List C:\Telecom\NX5\Proyectos\NX5_Commission\

Auto detect

If a Dual Core is added with a USB memory installed, unlimited storage for these 16 graphs will be added, plus storage of Binary Inputs and Outputs, and RES_FLT registers and remote points.

Pressing the INFO button on the main screen will open a dialog where the storage options for this USB memory can be selected.

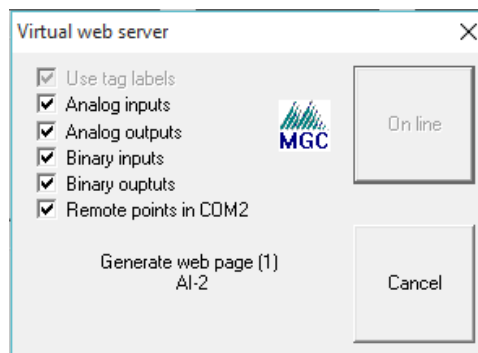


1.13 Virtual Web Server Enabling

When on the main screen, pressing the WEB SERVER button, opens a screen to activate a web server feature to convert any PC into a storage web server.



With the addition of the OpenBAS-SW-CFGTL and any standard PC to the OpenBAS-HV-NX10P controller, the PC can be turned into a web server at the press of a button.






By selecting what do you want to get updated on the web pages and pressing the ON LINE button, the OpenBAS-SW-CFGTL communicates with the OpenBAS-HV-NX10P controller and generates in the active project directory a set of dynamic web pages that update regularly at a 15 second interval.

By opening the main page _NX5.HTML the web pages will be shown in any Web Browser of your choice, reflecting the current values on the controller. If these pages are made available to any IP connection, a free web server is obtained.

<input checked="" type="checkbox"/>	_nx5.html	21/12/2015 09:35 a...	Firefox HTML Doc...	1 KB
<input type="checkbox"/>	ai.html	21/12/2015 09:35 a...	Firefox HTML Doc...	4 KB
<input type="checkbox"/>	ao.html	21/12/2015 09:35 a...	Firefox HTML Doc...	2 KB
<input type="checkbox"/>	bi.html	21/12/2015 09:35 a...	Firefox HTML Doc...	4 KB
<input type="checkbox"/>	bo.html	21/12/2015 09:35 a...	Firefox HTML Doc...	5 KB
<input type="checkbox"/>	LABELS.HTML	06/04/2015 11:13 a...	Firefox HTML Doc...	4 KB
<input type="checkbox"/>	rm.html	26/08/2015 07:15 a...	Firefox HTML Doc...	5 KB

Also trending of the analog data will be logged in the following files:


 ai_log.csv	21/12/2015 09:41 a...	Archivo de valores...	3 KB
 ao_log.csv	21/12/2015 09:35 a...	Archivo de valores...	1 KB
 rmt_log.csv	21/12/2015 09:41 a...	Archivo de valores...	2 KB

All the HTML files and CSV files will be generated in the active (current) project's directory.

Active project

☐ Auto detect

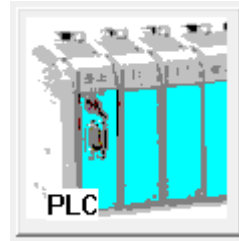
There is a file named LOGO.JPG that can be modified by the user to personalize the image shown in the generated web pages. By editing this JPG file and running the Web Server tool the new web pages will contain this graphical information in them. This file must be present in the active project directory for the Web Pages to render the image correctly.

 LOGO.JPG	26/05/2011 10:38 a...	Archivo JPG	18 KB
---	-----------------------	-------------	-------

Reloj interno NX5: Fecha: 21/12/2015 Hora: 9:51:06		
Entradas analogicas	Salidas analogicas	Entradas binarias
	Salidas binarias	Puntos remotos
ENTRADAS ANALOGICAS		
NUMERO	VALOR ACTUAL	NOMBRE
AI-1	295.90	Temp PRO-2000
AI-2	7.76	

1.14 PLC (Programmable Logic Controller)

When on the main screen, pressing the PLC button opens a screen use the Programmable Logic Controller feature of the OpenBAS-HV-NX10P.



Sections:

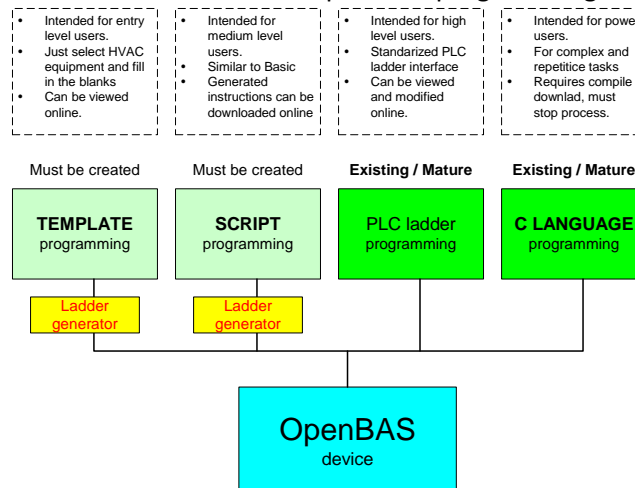
- 7.0 PLC Ladder Programming Introduction and
- 8.0 PLC Ladder Programming detailed programming description

Will provide with the necessary knowledge to allow the creation of powerful programs to tackle any automation solution needed in the field.

The **OpenBAS-HV-NX10P** controller provides with 400 PLC instructions, also called “Blocks” or “Equations” that when combined with schedules, remote communications capabilities, can create complex and powerful automation sequences through large facilities.

When a Dual Core is installed, two additional PLC’s are available to provide the user with 1200 PLC instructions in the master alone. Also each slave added has additional 400 PLC instructions, that if enabled in a distributed computing scenario, can provide a master equipped with a Dual Core and 4 slave devices with up to 2,800 PLC instructions.

More programming languages will be added in the future to allow for easy creation of programs using fill in the blanks templates, and HVAC standardized Script Based programming.

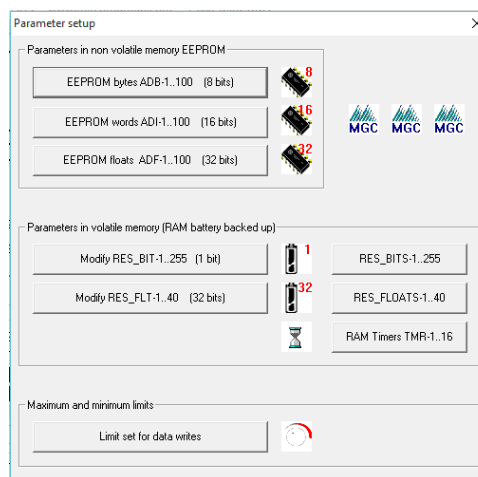


1.15 Database Parameters Viewing and Modifying

When on the main screen, pressing the PARAMETERS SETUP, opens up a secondary dialog box that allows access to the database data objects stored in both EEPROM and RAM for viewing and editing.



This database objects represent the data part of the program, and the same they can be read and written manually in this section. They can be read and written by any communication Fieldbus, the Ethernet Network Card, the PLC program logic, any operator LCD display added or by the Dual Core processor if installed.



The first block of three object types, represent the EEPROM stored database objects. These variables once modified will preserve their values even if the battery gets discharged.

The second block defines database objects stored in RAM and they will lose values if the controller is not powered for more than seven days and the battery gets fully discharged.

At the end is an option to limit the values of database objects. This is very useful to set disallow commands received by any communication port to write objects outside of they allowed minimum and maximum values.

1.15.1 ADF 32-bit registers stored in EEPROM

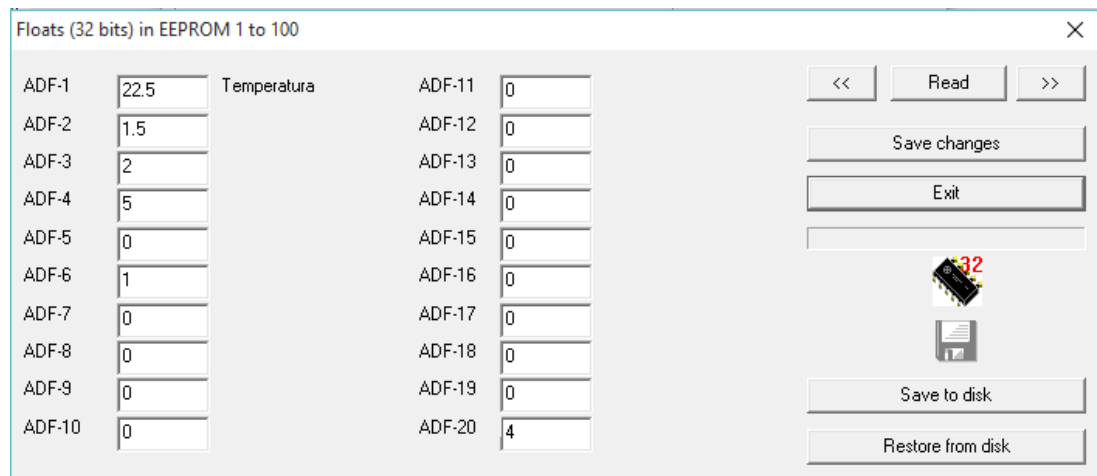
The ADF 32-bit floating point type set points which are stored in EEPROM memory, are used to represent values that are real world values, such as temperatures, pressures, weight, etc.



By using an internal format of 4 bytes or 32 bits using the standard IEEE format, the values that this data variable can have are in the ranges:

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{126} * (2 - 2^{-15}) \approx 6.80564693e + 38$

However, the values are internally limited to span in a range from -99999.9999 to +99999.9999 to allow them to be observed in the LCD operator display.

There are 100 registers available: ADF-1 to ADF-100 that can be used. The following picture illustrates the dialog used to edit the ADF registers.



The buttons  and  allow you to navigate the 100 registers, 20 at a time, and with the **Save to Disk** and **Restore to Disk**, the EEPROM registers can be read or written to into the file **ADF.INI**.

Any time any values are edited on the dialog screen, the SAVE CHANGES button needs to be pressed to reflect the changes made into the ON LINE database.

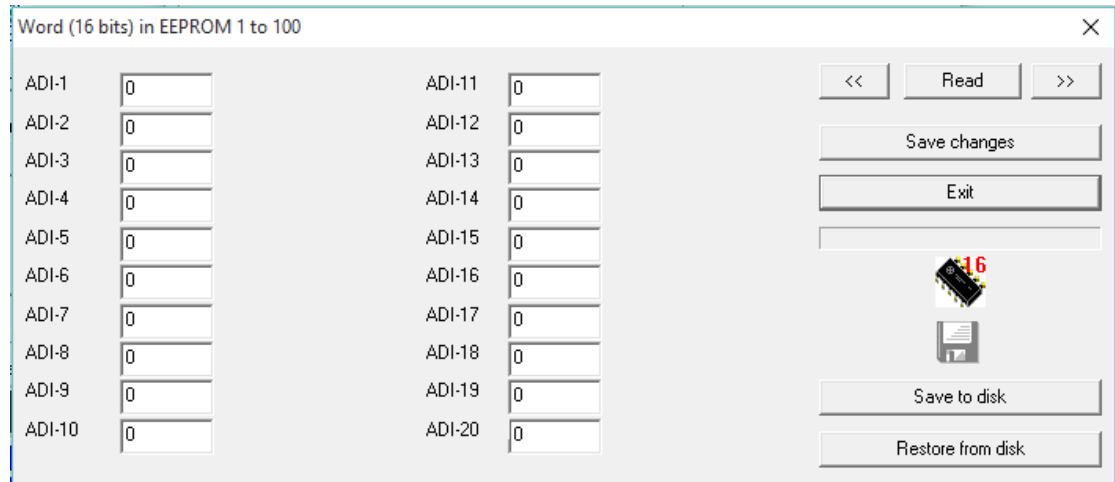
Any Label Tag that has been assigned to an ADF register will be shown to the right of the value.



1.15.2 ADI 16-bit registers stored in EEPROM

The ADI 16-bit registers are used to store values in the range 0-65535. These values are only integer positive and cannot contain fractional or negative values.

Their main intended use is to store large integers for timers and any other value that needs integer values in the above given range.

There are 100 registers available: ADI-1 to ADI-100 that can be used. The following picture illustrates the dialog used to edit the ADI registers.



The buttons  and  allow you to navigate the 100 registers, 20 at a time, and with the **Save to Disk** and **Restore to Disk**, the EEPROM registers can be read or written to into the file **ADI.INI**.

Any time any values are edited on the dialog screen, the SAVE CHANGES button needs to be pressed to reflect the changes made into the ON LINE database.

Any Label Tag that has been assigned to an ADF register will be shown to the right of the value.

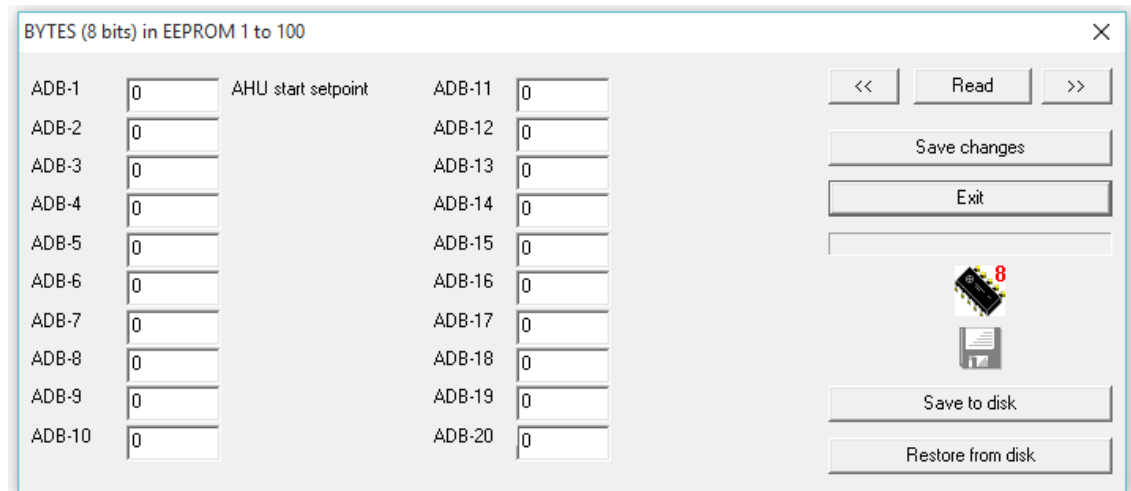
1.15.3 ADB 8-bit registers stored in EEPROM



The ADB 8-bit registers are used to store values in the range 0-255. These values are only integer positive and cannot contain fractional or negative values.

Their main intended use is to store any variable that can present a multiple states such as OFF (0), LOW (1), MED (2), HIGH (3), or any other value that needs integer values in the above given range.

Also storing binary information such as if a device must be OFF (0) or ON (1) is a standard use of this register, as schedules can modify this registers based on time of day.

There are 100 registers available: ADB-1 to ADB-100 that can be used. The following picture illustrates the dialog used to edit the ADB registers.



The buttons  and  allow you to navigate the 100 registers, 20 at a time, and with the **Save to Disk** and **Restore to Disk**, the EEPROM registers can be read or written to into the file **ADB.INI**.

Any time any values are edited on the dialog screen, the SAVE CHANGES button needs to be pressed to reflect the changes made into the ON LINE database.

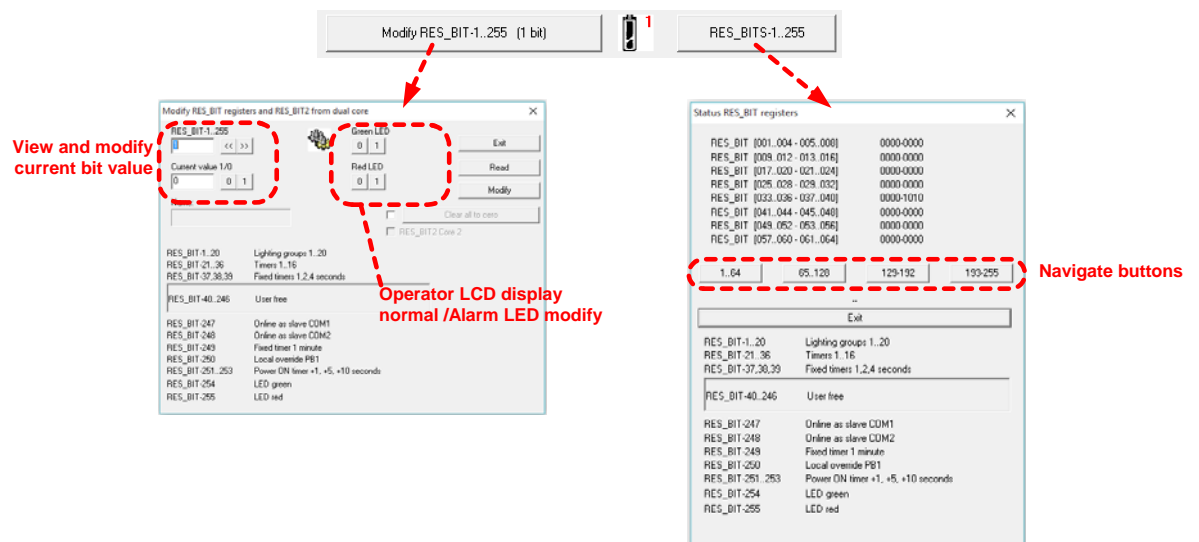
Any Label Tag that has been assigned to an ADF register will be shown to the right of the value.

1.15.4 RES_BIT 1-bit result registers stored in RAM

The RES_BIT term stands for “Result Bits”, the OpenBAS-HV-NX10P has 255 result bits labeled from RES_BIT-1 to RES-BIT-255. They are grouped and some of them have a specific use which is listed in the table below:

RES_BIT-1..20	Lighting groups 1..20
RES_BIT-21..36	Timers 1..16
RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
RES_BIT-40..246	User free
RES_BIT-247	Online as slave COM1
RES_BIT-248	Online as slave COM2
RES_BIT-249	Fixed timer 1 minute
RES_BIT-250	Local override PB1
RES_BIT-251..253	Power ON timer +1, +5, +10 seconds
RES_BIT-254	LED green
RES_BIT-255	LED red

The two buttons that are used for the RES_BITS are: The left one labeled: MODIFY RES_BITS that as its name implies, allows for interaction with the online database to read and write the RES_BIT registers. The button on the right allows to view the RES_BIT registers in 64 bit blocks.



1.15.5 RES_FLT 32-bit result registers stored in RAM

The RES_FLT 32-bit floating result registers are used to store the results of the PLC instructions that provide real values, such as the math instructions, proportional control, etc.

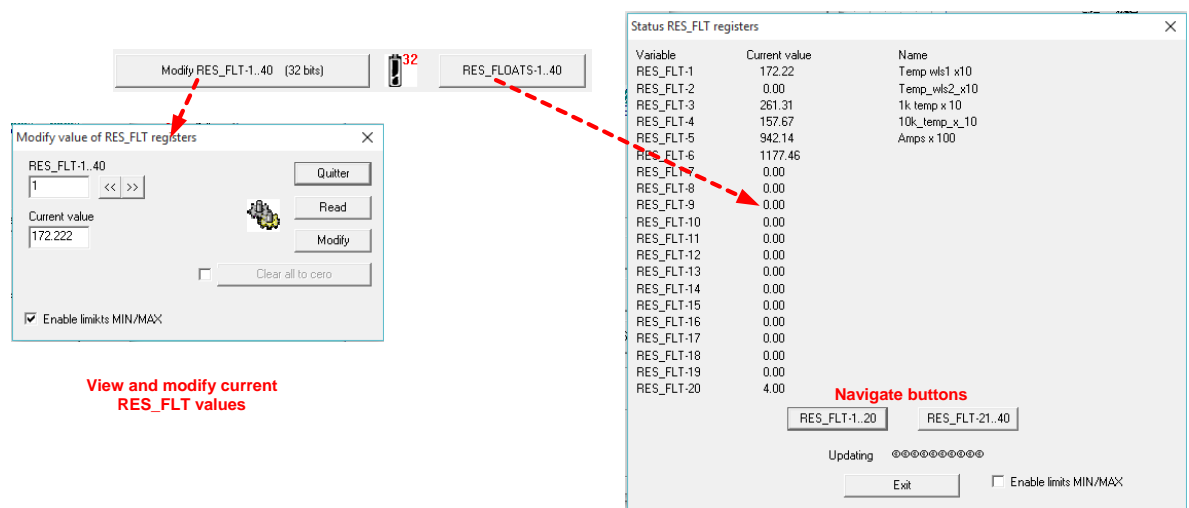
By using an internal format of 4 bytes or 32 bits using the standard IEEE format, the values that this data variable can have are in the ranges:

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2 - 2^{-126}) \approx 6.80564693e + 38$

However, the values are internally limited to span in a range from -99999.9999 to +99999.9999 to allow them to be observed in the LCD operator display.

There are 40 registers available: RES_FLT-1 to RES_FLT-40 that can be used. If a Dual Core is installed, additional 210 registers are added RES_FLT-41 to 255. They can be used as remote points or as standard registers for whatever the program needs if they are not used as remote points.

The two buttons that are used for the RES_FLOATS are: The left one labeled: MODIFY RES_FLT that as its name implies, allows for interaction with the online database to read and write the RES_FLT registers. The button on the right allows to view the RES_FLT registers in 20 registers blocks.



Modify value of RES_FLT registers

RES_FLT-1..40 (32 bits)

RES_FLOATS-1..40

Modify value of RES_FLT registers

RES_FLT-1..40

Current value: 172.222

Buttons: Quitter, Read, Modify, Clear all to zero

☒ Enable limits MIN/MAX

Status RES_FLT registers

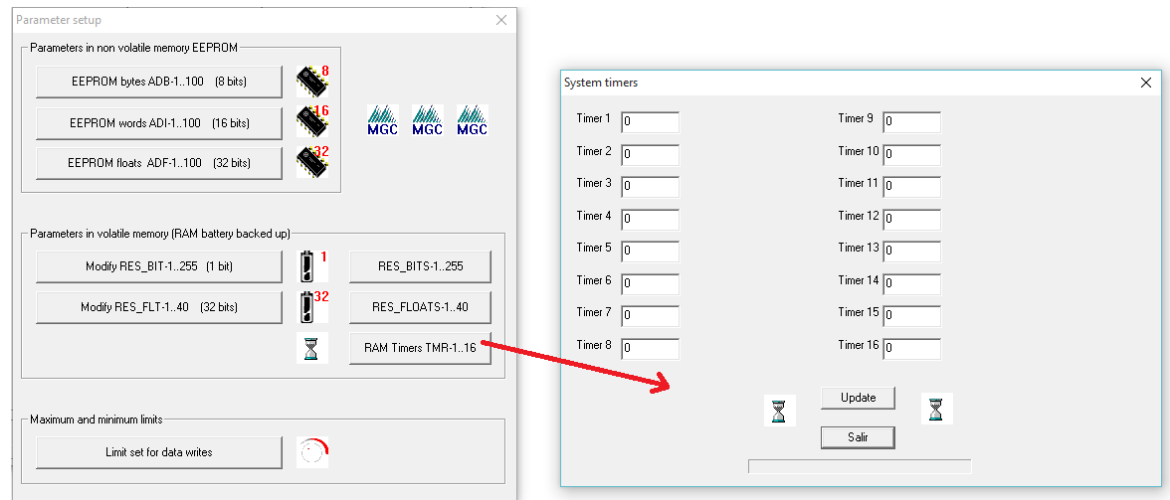
Variable	Current value	Name
RES_FLT-1	172.22	Temp wls1 x10
RES_FLT-2	0.00	Temp wls2 x10
RES_FLT-3	261.31	1k temp x 10
RES_FLT-4	157.67	10k temp x 10
RES_FLT-5	942.14	Amps x 100
RES_FLT-6	1177.46	
RES_FLT-7	0.00	
RES_FLT-8	0.00	
RES_FLT-9	0.00	
RES_FLT-10	0.00	
RES_FLT-11	0.00	
RES_FLT-12	0.00	
RES_FLT-13	0.00	
RES_FLT-14	0.00	
RES_FLT-15	0.00	
RES_FLT-16	0.00	
RES_FLT-17	0.00	
RES_FLT-18	0.00	
RES_FLT-19	0.00	
RES_FLT-20	4.00	

Buttons: RES_FLT-1..20, RES_FLT-21..40, Exit, Enable limits MIN/MAX

The RES_FLT registers from the Dual Core are described on section 6.9 remote points.

1.15.6 TMR 16-bit system timers stored in RAM

The 16 system timers TMR-1 to TMR-16 can be viewed on the following screen.



The system timers are decremented every 1 second or 1/10 second depending on their setting when their control bit (RES_BIT-21..36) is = 0, until they reach a ZERO count

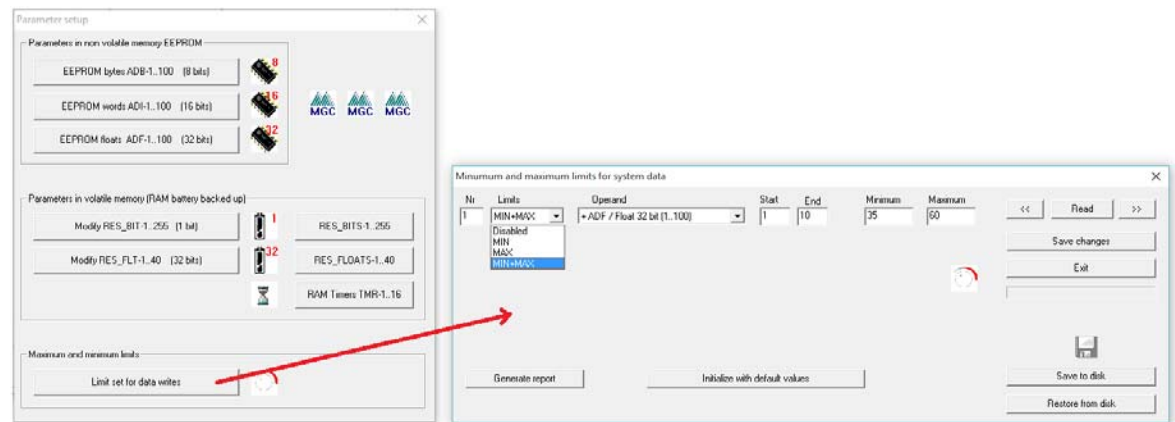
When the control bit for the timer is = 1, the timer is reloaded with its stored value which is hold into any of the ADI registers as configured in the timer instruction.

1.15.7 MIN/MAX Setting of Minimum and Maximum limits for data registers

When you need to limit the values of some of the database objects so they can't be modified outside a certain range by any command received by any communication channel, the MIN / MAX configuration can help to define up to 16 different registers, or block of registers areas that must be constrained by:

- Minimum limit
- Maximum limit
- Minimum and Maximum limits

The following picture shows the Min / Max configuration screen with an example of how to limit a block of registers ADF-1 to 10 to have a minimum limit of 35 and a maximum limit of 60.



Any combination of limits can be set. The configuration can be saved and restored to disk to de file LIMITES.INI



Also a report can be generated, and there is a button to reset the MIN / MAX to default values.

1.16 General System Configuration

When on the main screen, pressing the CONFIGURE button, opens secondary dialog box that provides access to the different configuration screens.

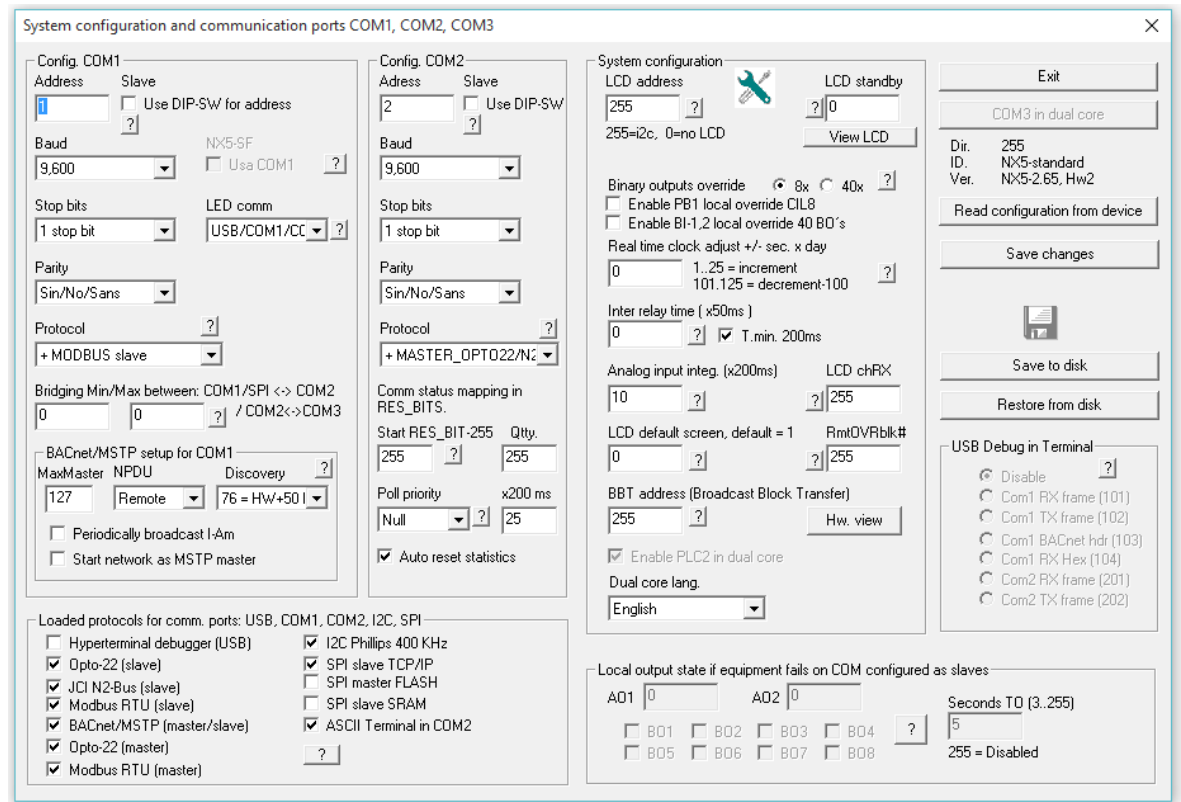


The different options are explained in detail in the following sections.

General system configuration	
6.16.1	System and communication port set up
6.16.2	General system, COM1, COM2, COM3
6.16.3	Remote points in COM2, RMT-1..50
6.16.4	Expanded remote points in COM2, RMT-51..255 RF-51..255
6.16.5	Remote points dual core in COM2 and/or COM3
	View status wireless links
6.16.6	Memory configuration EEPROM and RAM
6.16.7	Manually read and write EEPROM memory
6.16.8	Initialize EEPROM memory to factory values
	Initialize RAM memory on power failure
6.16.9	Clock and calendar
6.16.10	Set up real time clock
6.16.11	Set up daylight saving time
	Set up holiday days 1..25
6.16.12	Labels, alarms and messages
	Label names 1..200
6.16.13	Label names LCD remote points 1..50
6.16.14	Alarm set up for LCD
6.16.15	LCD screen personalized by user
6.16.16	SMS text messages set up
6.16.17	IP parameters and e-mail messages set up
6.16.18	System security, storage, event log and reset
6.16.19	Modify access passwords
6.16.20	Save and restore program to and from disk
6.16.21	View historic event log
	Software reset controller

1.16.1 Configure the Fieldbus Communication Ports

It is important to note that if these parameters are incorrectly set, it will can affect the control operation or loose the communications link with slave devices or remote points.



The screenshot shows the 'System configuration and communication ports COM1, COM2, COM3' window. It is divided into several sections:

- Config. COM1:** Address (1), Baud (9,600), Stop bits (1 stop bit), Parity (Sin/No/Sans), Protocol (+ MODBUS slave), Bridging Min/Max between: COM1/SPI <-> COM2 (0), COM2 <-> COM3 (0).
- Config. COM2:** Address (2), Baud (9,600), Stop bits (1 stop bit), Parity (Sin/No/Sans), Protocol (+ MASTER_OPT022/N2), Comm status mapping in RES_BITS (255), Start RES_BIT-255 (255), Poll priority (Null), x200 ms (25), Auto reset statistics (checked).
- System configuration:** LCD address (255), LCD standby (0), Binary outputs override (8x, 40x), Real time clock adjust (+/- sec. x day) (0), Inter relay time (x50ms) (0), Analog input integ. (x200ms) (10), LCD chRX (255), LCD default screen, default = 1 (0), BBT address (Broadcast Block Transfer) (255), Dual core lang. (English).
- Right sidebar:** Exit, COM3 in dual core, Dir. 255, ID. NX5-standard, Ver. NX5-2.65, Hw2, Read configuration from device, Save changes, Save to disk, Restore from disk, USB Debug in Terminal (Disable, Com1 RX frame (101), Com1 TX frame (102), Com1 BACnet hdr (103), Com1 RX Hex (104), Com2 RX frame (201), Com2 TX frame (202)).
- Loaded protocols for comm. ports:** USB, COM1, COM2, I2C, SPI. Includes checkboxes for Hyperterminal debugger (USB), Opto-22 (slave), JCI N2-Bus (slave), Modbus RTU (slave), BACnet/MSTP (master/slave), Opto-22 (master), Modbus RTU (master), I2C Phillips 400 KHz, SPI slave TCP/IP, SPI master FLASH, SPI slave SRAM, and ASCII Terminal in COM2.
- Local output state if equipment fails on COM configured as slaves:** A01 (0), A02 (0), Seconds TO (3..255) (5), 255 = Disabled.

The first and second sections, are to configure the communications of **COM1** and **COM2**

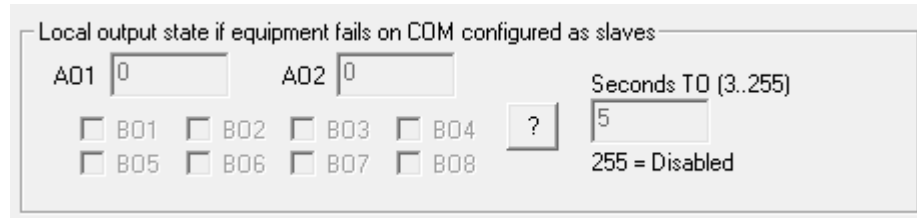
- Address: 1..254
- Baud rate: 2,400 up to 76,800
- Stop bits: 0,1
- Parity: No parity, Even or Odd
- Protocol: **Any available protocol as either slave or master**

(NOTE: The supported protocols on each port have the “+” symbol).

The third block of SYSTEM parameters have the following fields to configure:

- **LCD** (operator interface) should be **255** if it is on the **I2C bus** or a number 1..254 if it is setup using the port RS-485, a value of ZERO disables the LCD display.
- **Setting real time clock**, is a parameter to compensate the real time clock with a range of up to +/- seconds per day. A value of 1..25 will advance this number of seconds the clock every day, and a number from 101 to 125 will delays from 1 to 25 (n-100) seconds per day.
- **Inter-relay time**, is a delay between the relays (inter-relay) applied when used as a lighting controller to avoid turning all relays simultaneously, the value is a multiple of 50 milliseconds. By example a value of 5 will give a delay of 250 milliseconds or ¼ of second between each relay, when they are operated manually from the operator interface or in the case of power failure.
- **Integration to the analog inputs** this time is to avoid abrupt changes with sensors that have a signal that changes very quickly or with large wiring that is susceptible to electrical interferences. The value is a multiple of 200 milliseconds or 1/5 of second, so a value of 10 indicates integration constant of two seconds. See section 6.7 Analog I/O Viewing and Setup Type and Calibration for additional information
- **Initial screen on the LCD** when pushing the **HOME** button on the LCD operator interface will go to a pre-programmed screen, the default is 1.
- **LCD CHrx** Is a special parameter for applications that will send special characters to the LCD, the value is between 1..32 or 255 as default.
- **Remote Override block #** will send an OVERRIDE to a Remote Point if the current OUTPUT ASSIGN instruction number is < to this parameter. Or a simple WRITE if the current OUTPUT ASSIGN instruction number is >= this parameter. Accepted ranges are 0 to 254, or the default value of 255 sends only OVERRIDES to Remote Points regardless of the current instruction number being currently executed.

The fourth block is enabled if any of the two ports COM1 or COM2 are set as slave. It allows to set a default value to which the **eight binary outputs (relays)** and the **two analog outputs** will go, if communications are lost with the master for a predetermined amount of time in seconds.



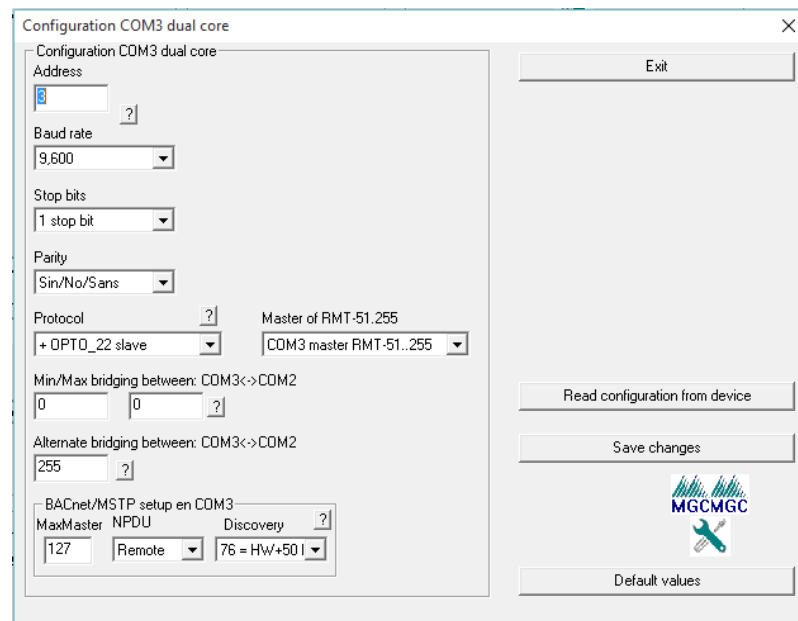
Local output state if equipment fails on COM configured as slaves

A01 A02

☐ B01 ☐ B02 ☐ B03 ☐ B04 ☐ B05 ☐ B06 ☐ B07 ☐ B08

Seconds TO (3..255) 255 = Disabled

In controls with Dual Core, the configuration screen of the **COM3** is the following:



Configuration COM3 dual core

Address ?

Baud rate

Stop bits

Parity

Protocol ? Master of RMT-51.255

+ OPT0_22 slave COM3 master RMT-51..255

Min/Max bridging between: COM3<->COM2

?

Alternate bridging between: COM3<->COM2

?

BACnet/MSTP setup en COM3

MaxMaster NPDU Discovery ?

Remote

Exit

Read configuration from device

Save changes

MGCMGC

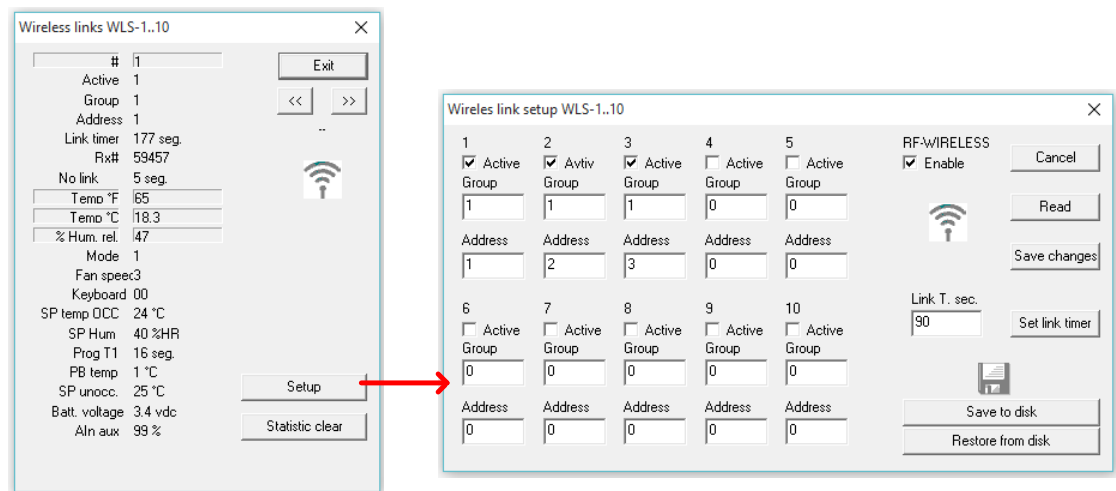
Default values

- 1.16.2 Set up Remote Points in Fieldbuses
- 1.16.3 Set up Expanded Remote Points in Fieldbuses
- 1.16.4 Set up Dual Core Remote Points in Fieldbuses

See section 6.9 details on setting up remote points in fieldbuses.

1.16.5 Set up Wireless Remote Points for Wireless Thermostats

If the I2C wireless adapter module OpenBAS-HV-RX433R is installed, up to 10 wireless thermostats can be added to the remote points, on the following screen the dialog to view is shown. If the SETUP button is pressed, a setup dialog box will appear where this 10 wireless links can be set.



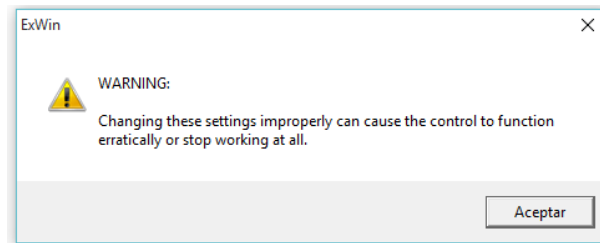
For details on how to set up the remote points refer to section 6.9 details on setting up remote points in fieldbuses.

1.16.6 Service EEPROM viewing and Programming

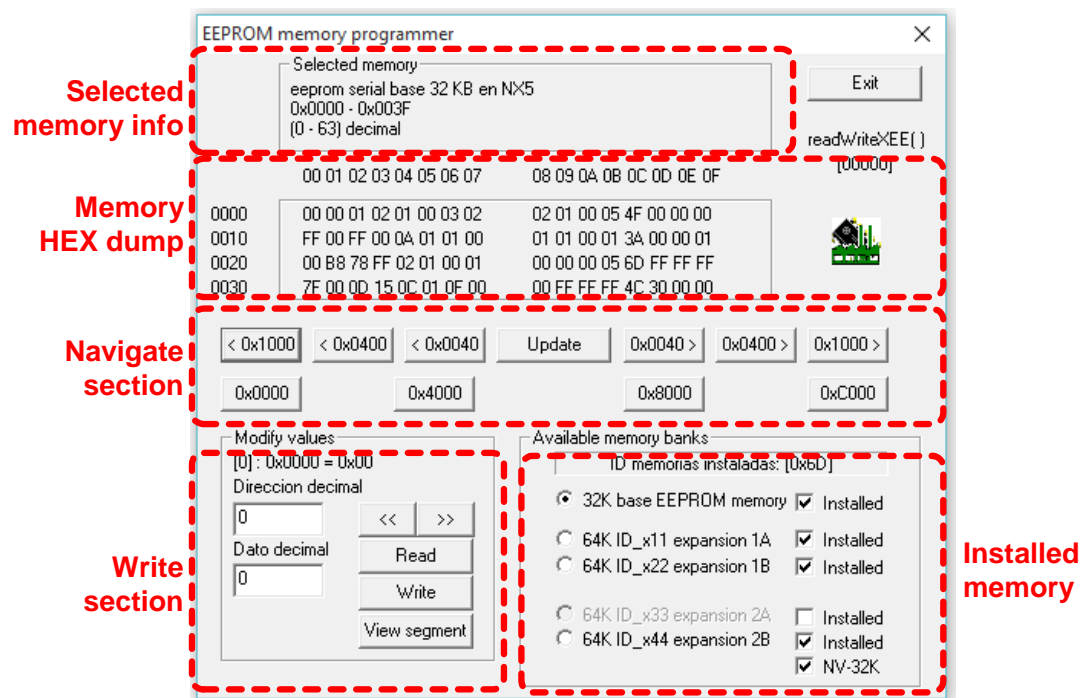
This option is to be used only by qualified personal who knows what each memory position means because by modifying any value erroneously can lock the controller or make it behave erratically

The whole range of EEPROM memory or memories installed can be viewed and modified using this option, however caution must be taken as moving any memory position to an incorrect value can render the controller out of service.



The following message shows every time this EEPROM viewer is invoked to remember the user of this.





The base 32 Kbyte EEPROM memory is always available, and the expansion memory will only be enabled if installed and detected correctly by the system.



The buttons  and  allows forward or backward 64 bytes (0x0040 hex).

The buttons  and  allows forward or backward 1024 bytes (0x4000 hex).

The buttons  and  allows forward or backward 4096 bytes (0x1000 hex).

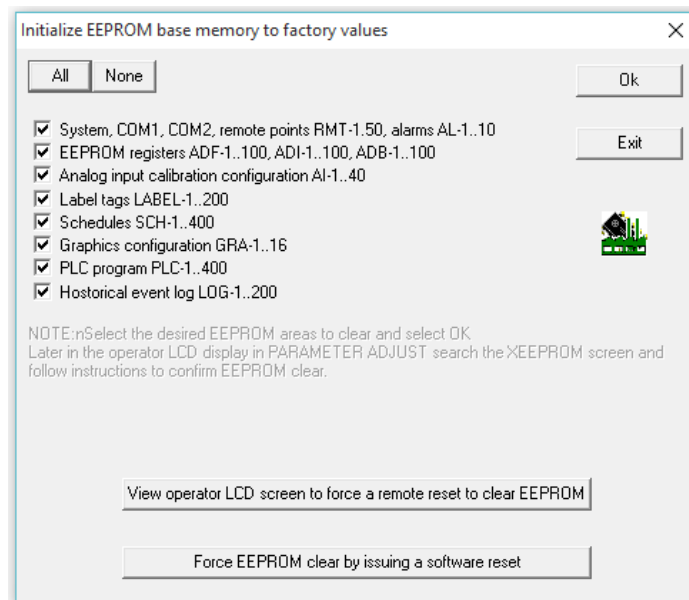
1.16.7 Reset controller to Factory Default Values

When there comes the need to partially or completely erase the memory of the controller. This option allows the full memory of the controller to be reset to default values.

NOTE:

Doing this will erase all the user programming. There is a security lock that should be activated from the operator interface so that the initialization of the EEPROM is carried out. The following steps to end the EEPROM initialization consists of selecting the SET PARAMETERS option on the controller and on the ERASE XEEPROM section push ENTER to get the actual erase procedure carried out.

First all sections meant to be cleared must be selected from the dialog box.

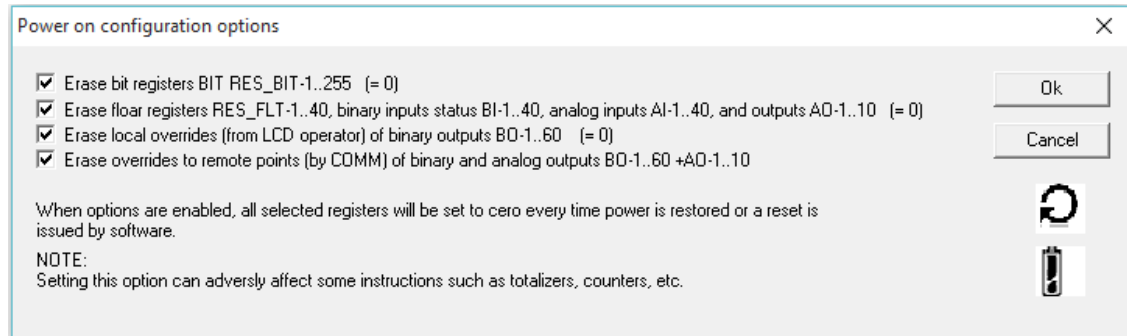


When all the boxes are selected, an option is enabled at the bottom to force the EEPROM erased from the program. There we have to introduce the service ZERO password to perform the erase procedure. The default password is **NX5servicio or OpenbasService**, and the password is case sensitive.

Once the **EEPROM** has been completely erased, you can **RESTORE** back the different parts of the program and settings files previously stored in the PC.

1.16.8 Set up what happens to Memory and I/O on Power On

This option allows you to erase registers in RAM of the sections: RES_BIT, RES_FLT and local or remote overrides that the user has set.



NOTE:

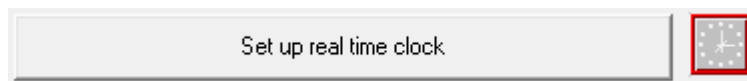
Is important to note that locking or overriding a Binary Output (Relay) is not a valid procedure of lockout and tag out that can protect a service technician from an electrical discharge or to suffer an accident by any mechanism that can be activated remotely

Always a mechanical lockout must be used, follow approved routines of lockout and tag out (LOCKOUT TAGOUT) with locks as approved by OSHA.

1.16.9 Set up real Time Clock

The adjust of the real time clock and calendar RTCC, can be done locally using the LCD display or the OpenBAS-HV-NX10P software.

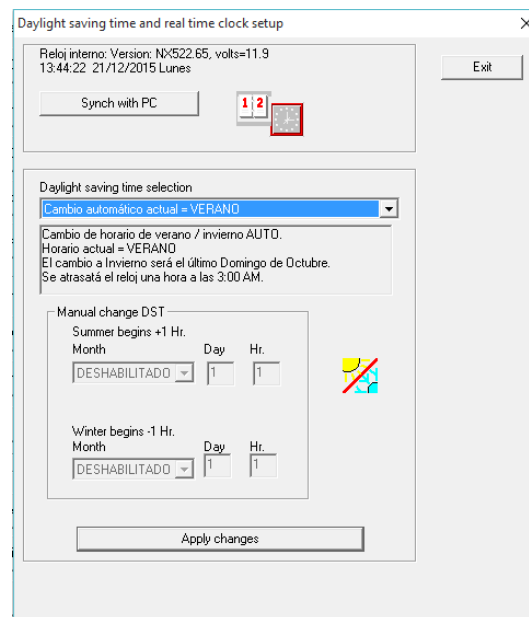
By clicking on the following button, the RTCC time will be synchronized with that of the computer.



Verify that the clock of the PC is correct before issuing this command as setting a wrong clock can affect scheduled operations.

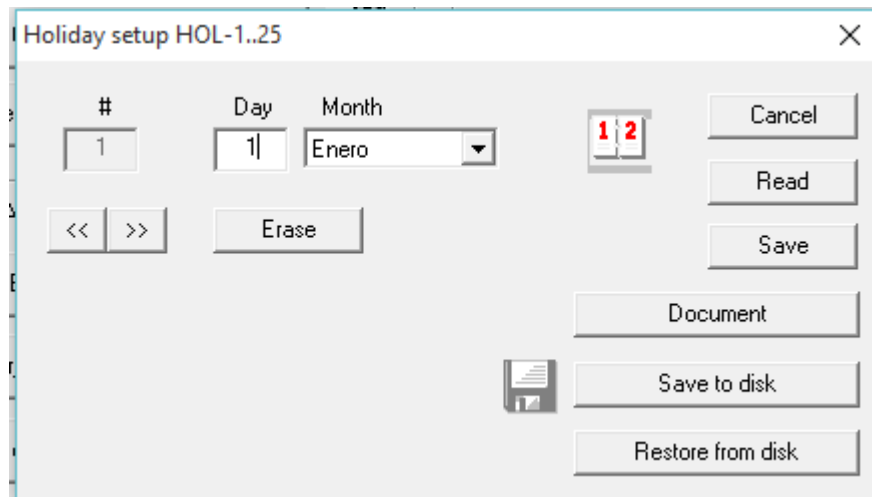
1.16.10 Set up Daylight Saving Time

From this screen the daylight saving time can be set. It is also possible for the controller to calculate DST automatically. If this option is programmed it, will change to **SUMMER TIME** the first Sunday of April and will change to **WINTER TIME** on the last Sunday from October.



1.16.11 Set up Holidays for Scheduling operation

The schedules have the option to operate on holidays. The controller has the capability to program in advance up to 25 holidays for the year. When setup and the user selects the option of holidays on the WEEKLY schedules, if the actual day is a holiday, the schedule will also operate.



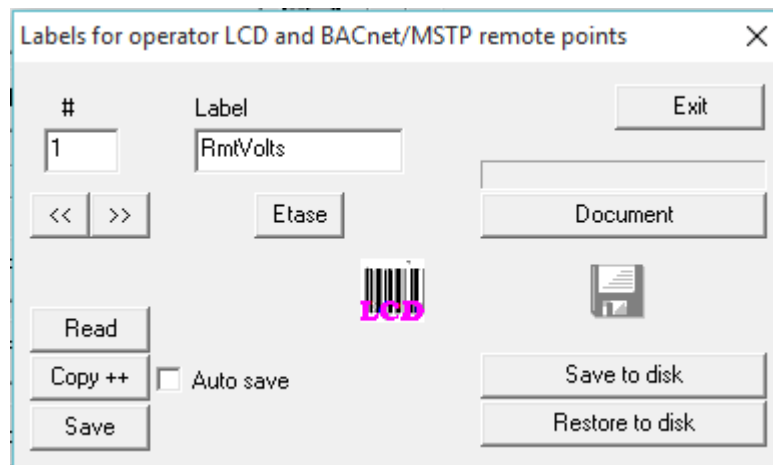
The document option prints a list of all setup holidays, and they can be saved and restored to the file FESTIVOS.INI.

1.16.12 Create and Edit Label Tags for identifying all your I/O and Database variables

See section 6.10 for details on editing and creating Label tags.

1.16.13 Create and Edit Label Tags for Remote points to show on LCD and BACnet names

Each one of the **50** remote points, can have a label of up to 8 characters, and each one can be shown on the **operator LCD**. In this screen we can modify each one of the 50 parameters.



These labels are independent of the 200 labels to identify the other points and variables included the same remote points, but these are only visible on the operator LCD.

On the lower right part of the dialog screen we can see the buttons to **BACKUP** the points set up to the hard drive to the archive **LCD_REMOTE.INI** and we can **RESTORE THEM** to the same or to another control.

This label information will be available as text for the name property of the 50 remote points on **BACnet / MSTP**.

1.16.14 Set up alarm messages for LCD operator display

There is a screen available to set the alarms that can be visualized on the LCD of the operator and on the OpenBAS-SW-CFGTL software. As can be seen in the following screen, on the main page it is showed as an indicator showing: GREEN, RED or YELLOW that tells the actual state of the alarms. Refer to section 6.4 on details for the alarms being shown.




When the **ALARMS** button is depressed, a screen appears where we can program which of the 255 registers of the **RES-BIT-1 to the 255** are active for each one of the alarms, as well as the text of them. The alarm 1 has the higher priority and the alarm 8 has the lowest priority.

Configure system alarms


Priority	#	RES_BIT	controlling	Message alarm in LCD	Erase	Active
High	1	1	RES_BIT-1	Alarm Text 1	Erase	--
	2	2	RES_BIT-2	01234567890	Erase	--
	3	3	RES_BIT-3	01234567890	Erase	--
	4	4	RES_BIT-4	01234567890	Erase	--
	5	5	RES_BIT-5	01234567890	Erase	--
	6	6	RES_BIT-6	01234567890	Erase	--
	7	7	RES_BIT-7	01234567890	Erase	--
	8	80	RES_BIT-80	ABC-DEFG-HIJK	Erase	--
Low						

Exit



Read

Save changes



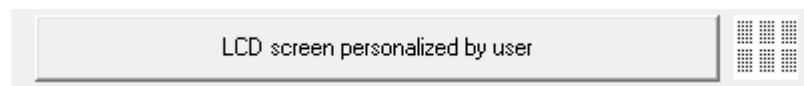
Save to disk

Restore from disk

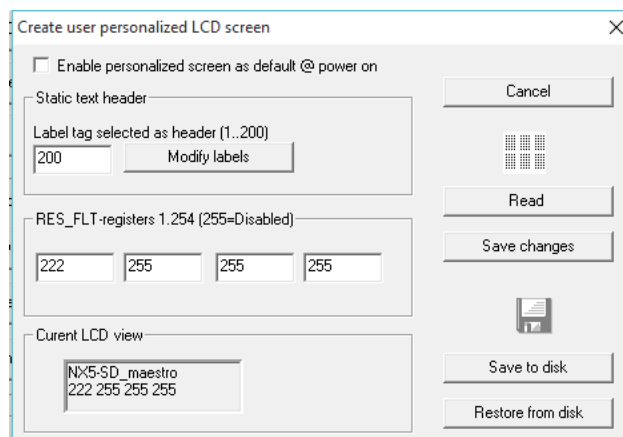
Disable all alarms

1.16.15 Set up personalized screens for user interaction for LCD operator display

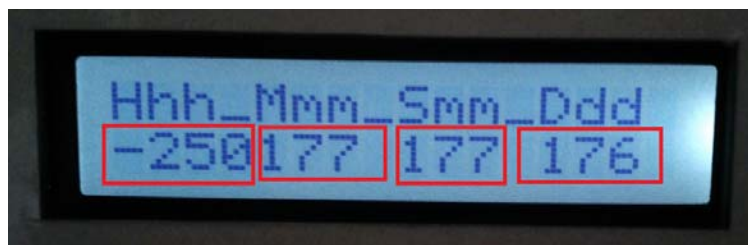
Starting from version 2.49 there is this option to allow the user to configure the LCD screen with custom data.



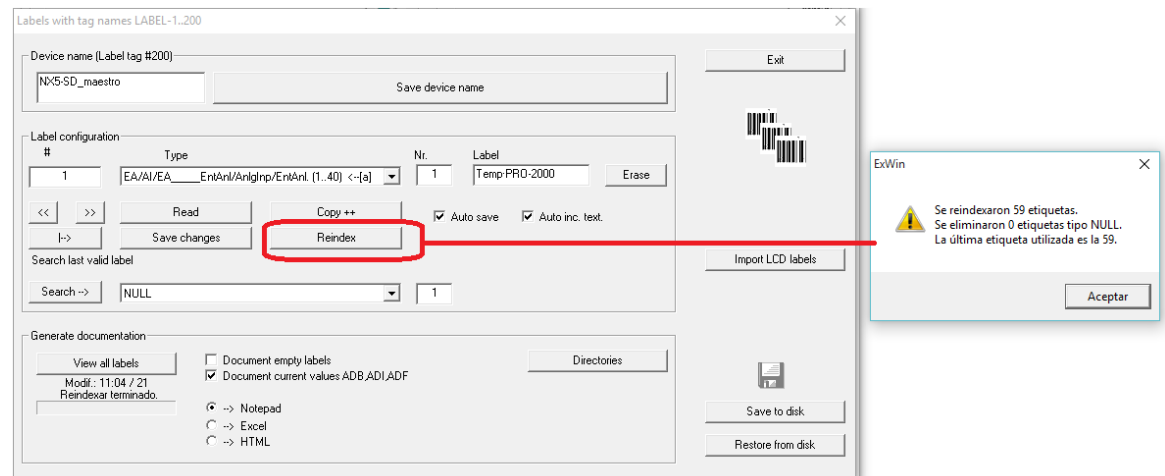
The user can configure this main screen of the LCD.



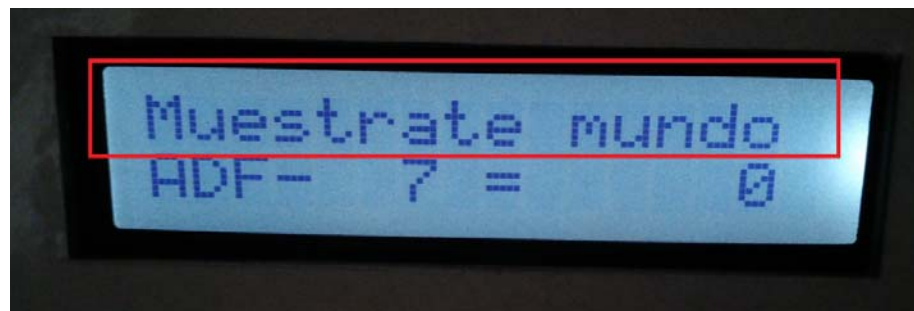
You can set a customized label with up to 4 variables type RES_FLT on the screen:



Also starting from version 2.49 we can see on the LCD in the parameter adjust screens of the Label Tags defined for the types: ADB, ADI and ADF in the upper part of the LCD, for the corresponding variable. Use the reindexing option on the Label Tags, see section 6.10 for setting up Label Tags.



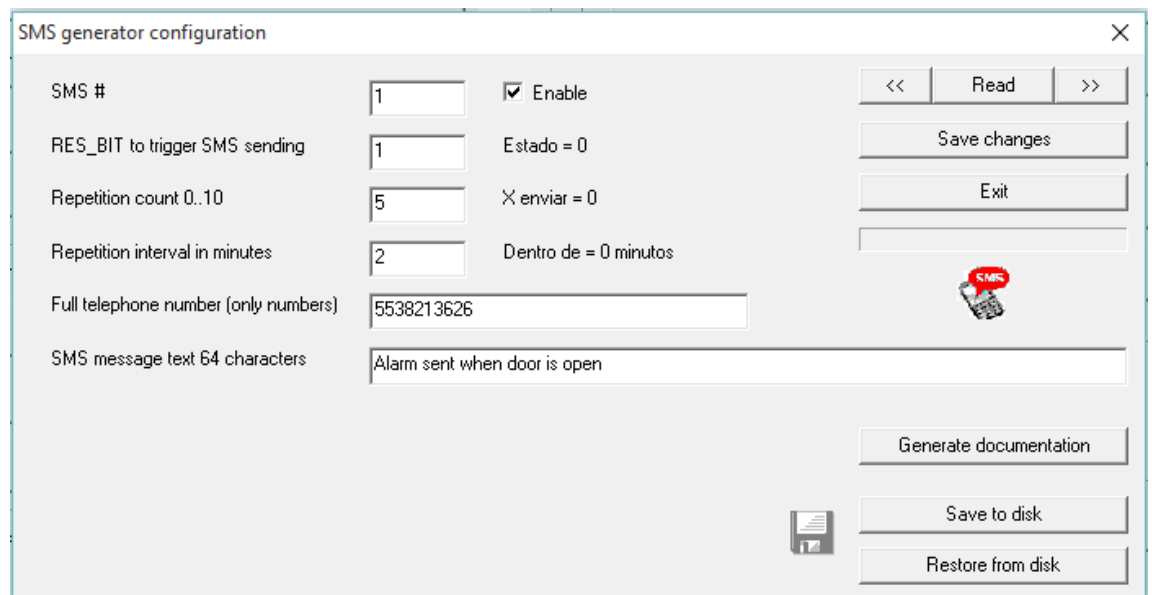
Here an example of how we can see this Label Tag, this way the operator or user can view the names of each variable directly on the LCD display.



Due to LCD display limitations only the first 16 characters of the Label Tag will be displayed.

1.16.16 Set up SMS messages to send over the OpenBAS-NWK-SMS GSM/SMS module

Using the the OpenBAS-NWK-SMS control it is easy to configure up to 20 messages that will be sent up to 20 different cellular phones.

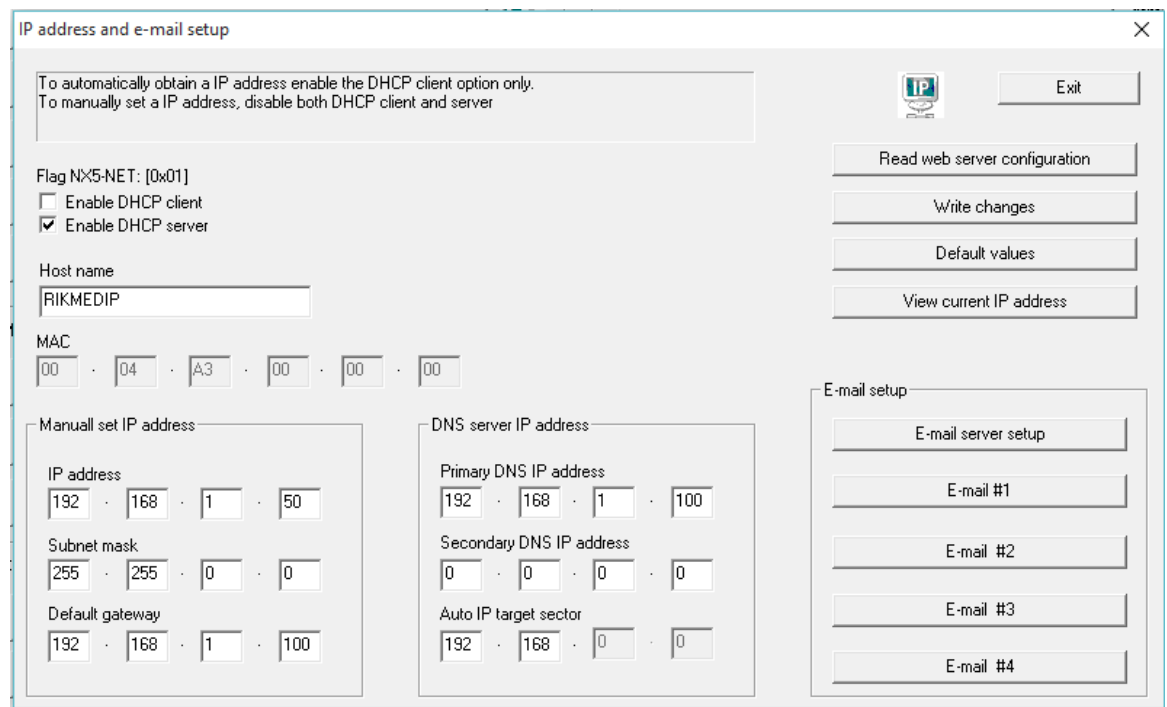


- On the **SMS #** field select the message number between 1 and 20.
- On the field **RES_BIT of control** we have to input which of the 255 register bits of the NX5 will send the SMS message when changing from ZERO to ONE. The message will be sent again each time that there is a change of ZERO to ONE.
- On the **repetitions number** field, we have to introduce ZERO to disable this function or a number between 1 and 10 to program the number of times that we want the SMS be resent automatically always that the control bit be maintained on a ONE level.
- On the **repetition interval in minutes'** field we introduce the minutes between each message sent always that the control bit being is maintained on ONE, the range is 1 to 60 minutes.
- On the **phone number** field, we have to introduce the number, same as we dial manually, including area or city or country code if it applies.
- On the **text message field**, we have to introduce up to 64 message characters that will be sent on the SMS.

Each time that a message is sent, it will generate a LOG register to keep an events log of its operation. We can send any message combination to any phone numbers as wished inside the limit of the 20 different SMS messages.

1.16.17 Set up IP parameters for Network Card and e-mails for event and alarm reporting

Using the the OpenBAS-NWK-XXX network controllers it is possible to access the controller via IP, and also generate e-mails on response to programmed events in a similar ways as SMS messages.



IP address and e-mail setup

To automatically obtain a IP address enable the DHCP client option only.
To manually set a IP address, disable both DHCP client and server

Flag NX5-NET: [0x01]

☐ Enable DHCP client
☒ Enable DHCP server

Host name
RIKMEDIP

MAC
00 · 04 · A3 · 00 · 00 · 00

Manual set IP address

IP address
192 · 168 · 1 · 50

Subnet mask
255 · 255 · 0 · 0

Default gateway
192 · 168 · 1 · 100

DNS server IP address

Primary DNS IP address
192 · 168 · 1 · 100

Secondary DNS IP address
0 · 0 · 0 · 0

Auto IP target sector
192 · 168 · 0 · 0

E-mail setup

E-mail server setup

E-mail #1

E-mail #2

E-mail #3

E-mail #4

Exit

Read web server configuration

Write changes

Default values

View current IP address

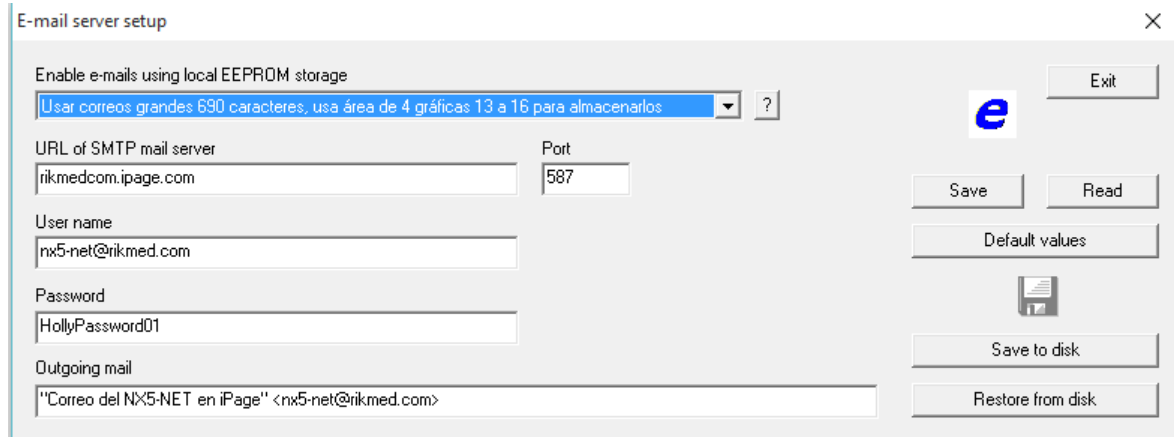
On the IP settings screen, the network Card can be set up to dynamically obtain a dynamic IP address by setting it as a DHCP client, or in small networks without a DHCP server, it can act as a DHCP server and offer clients a dynamic address by being set as a DHCP server.

If not using a dynamic IP address, all the IP parameters have to be manually set, consult your IT administrator if unsure of what type of IP addressing is in use in your network.

Also to set up e-mail generation, first the E-Mail server has to be set up. Refer to the next page for detailed instructions on setting up the e-mail server.

First if e-mail service is to be enabled, some part of the memory used for graphic storage must be spared to store the outgoing e-mail body. Three options are available:

- E-mails disabled
- Small e-mail body 190 characters long, which uses 2 graphics (15 and 16)
- Long e-mail body 690 characters long, which uses 4 graphics (13 to 16)



Once the storage is defined, the following information must be set:

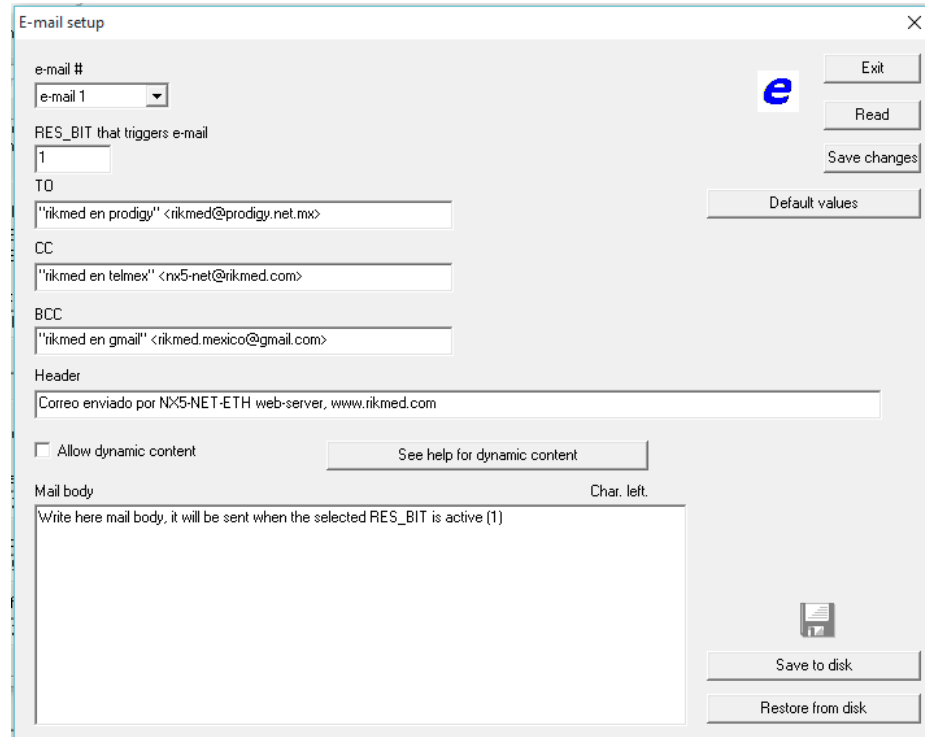
- URL and PORT information of SMTP mail server
- User name and password
- A text and an e-mail address that will be sent in all outgoing mails

All this information can be saved and restored from disk.

On the next page the detailed instructions on how to set up the four e-mails that can be generated is shown.

Mails are generated in a similar fashion as SMS messages, a RES_BIT is assigned to each e-mail, and when it is active, the e-mail will be generated and sent. A valid connection to the internet is necessary for the e-mails to be sent.

On the following picture, the general setup of e-mail 1 of 4 is shown with all fields that must be filled.



The screenshot shows the 'E-mail setup' window with the following fields and options:

- e-mail #:** A dropdown menu set to 'e-mail 1'.
- RES_BIT that triggers e-mail:** A text box containing the number '1'.
- TO:** A text box containing '"rikmed en prodigy" <rikmed@prodigy.net.mx>'.
- CC:** A text box containing '"rikmed en telmex" <nx5-net@rikmed.com>'.
- BCC:** A text box containing '"rikmed en gmail" <rikmed.mexico@gmail.com>'.
- Header:** A text box containing 'Correo enviado por NX5-NET-ETH web-server, www.rikmed.com'.
- Mail body:** A large text area with the placeholder text 'Write here mail body, it will be sent when the selected RES_BIT is active (1)'. To the right of the text area is a 'Char. left.' label.
- Buttons:** 'Exit', 'Read', 'Save changes', 'Default values', 'Allow dynamic content' (checkbox), 'See help for dynamic content', 'Save to disk', and 'Restore from disk'.

The Web Sever incorporated in the Network Card can be used to verify if access to the internet and to reach the SMTP server is ok.

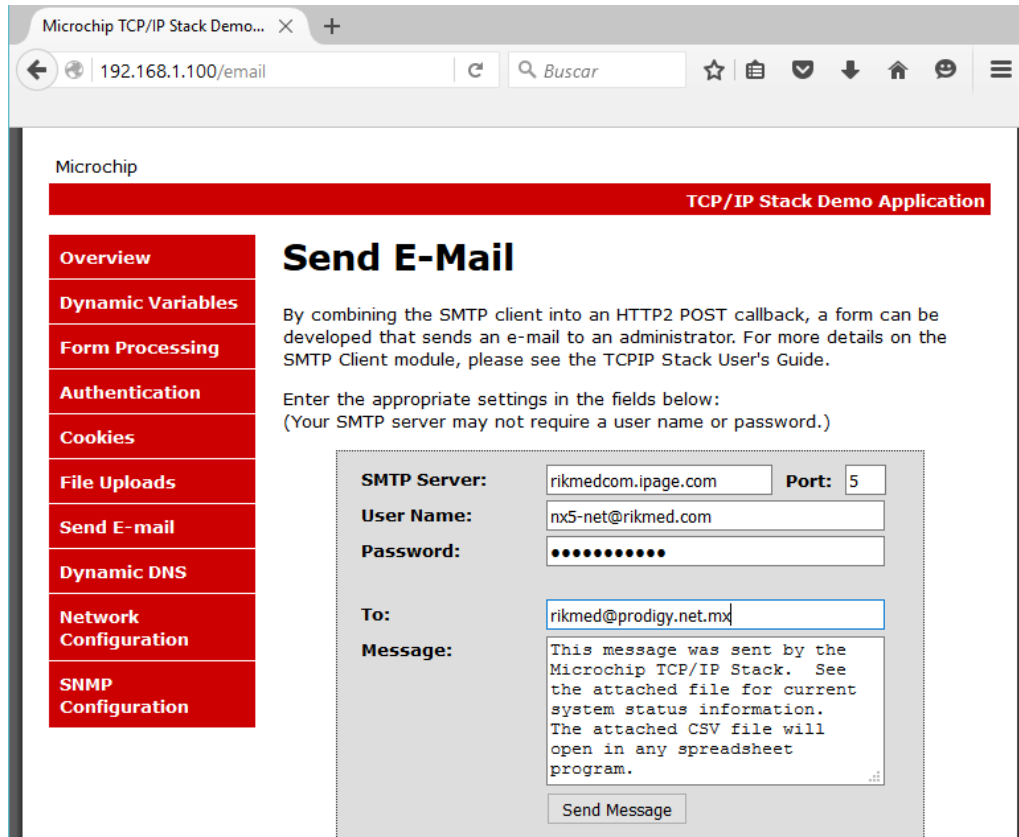
The first step after locating the address of your network card is to open the e-mail web page generator:

For example for the IP address of the network card 192.168.1.100 the following must be input in the address field of the web browser:

<http://192.168.1.100/email>

If communication is ok, the web page on next screen should pop up. Populate with the correct meters and press the SEND MESSAGE button.

Web page for the e-mail tester web page.



Microchip TCP/IP Stack Demo Application

Send E-Mail

By combining the SMTP client into an HTTP2 POST callback, a form can be developed that sends an e-mail to an administrator. For more details on the SMTP Client module, please see the TCP/IP Stack User's Guide.

Enter the appropriate settings in the fields below:
 (Your SMTP server may not require a user name or password.)

SMTP Server: rikmedcom.ipage.com Port: 5

User Name: nx5-net@rikmed.com

Password:

To: rikmed@prodigy.net.mx

Message: This message was sent by the Microchip TCP/IP Stack. See the attached file for current system status information. The attached CSV file will open in any spreadsheet program.

Send Message

If all settings are correct, a message showing that the message was sent will show up on the web server.

Send E-Mail

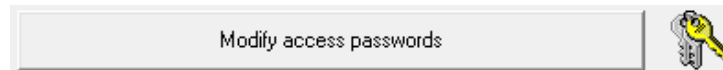
Your message has been sent.

If some parameters are wrong, an error message will show. Correct the parameters and retry until successful.

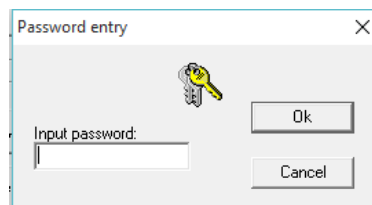
Send E-Mail

ERROR: Your message could not be sent.
Check your SMTP server settings and try again.

1.16.18 Set up Passwords for different levels Operator, Technical and Administrator accounts

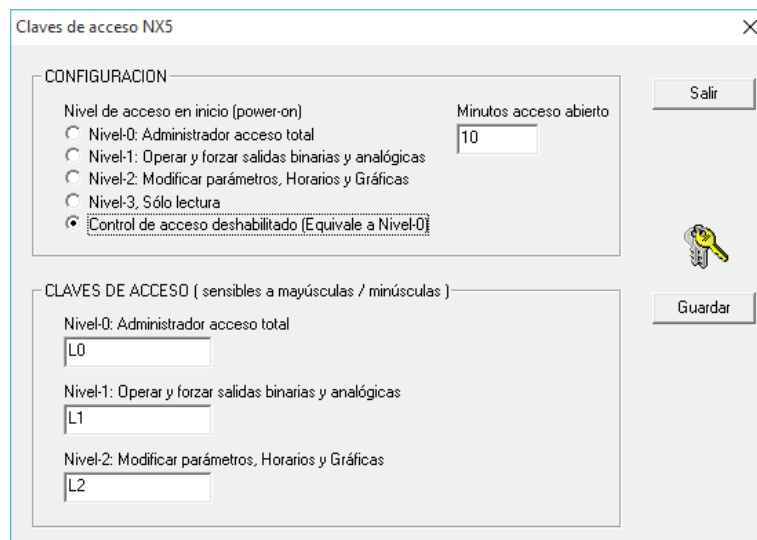


With this option it can be limited what the user can do. To enter to the code configuration password you have to input the service code **LEVEL ZERO** to perform the changes, or in its default state.



The default password is **NX5programador or OpenbasProgram**

The access code is case sensitive. There are **THREE** levels with which the OpenBAS-HV-NX10P can grant access, which are applied since the equipment is powered up, the image below shows the access levels **LEVEL_0**, **LEVEL_1**, **LEVEL_2** and the compatibility mode with previous versions in which that there is no access control enabled.

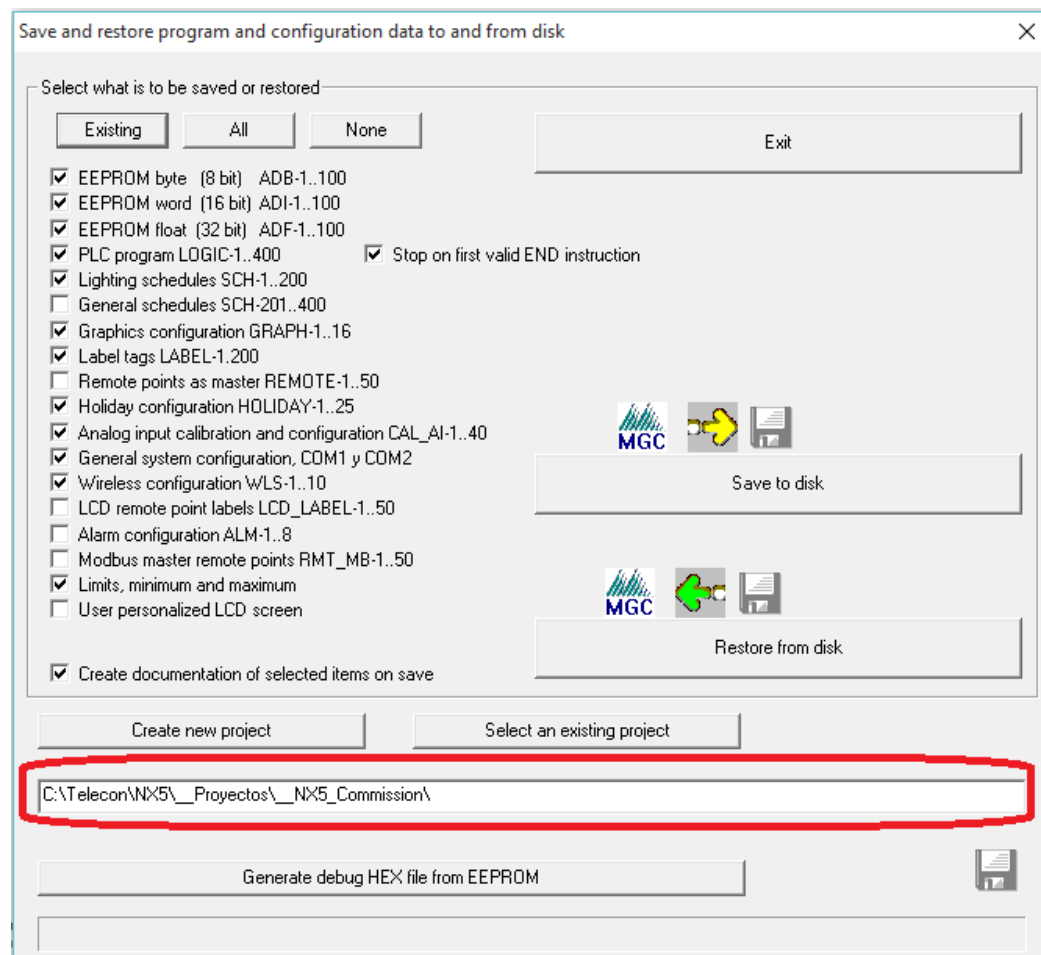


The passwords are **EIGHT** characters long, and are case; the minutes that the access will be open must be between **1 and 60 minutes**.

1.16.19 Create backups or restore all your Building Automation Controller programming

Backups are an important part of programming and the OpenBAS-SW-CFGTL offers an advanced batch save and restore feature. Here the programmer can save all the programs for backup or for transferring to another controller.

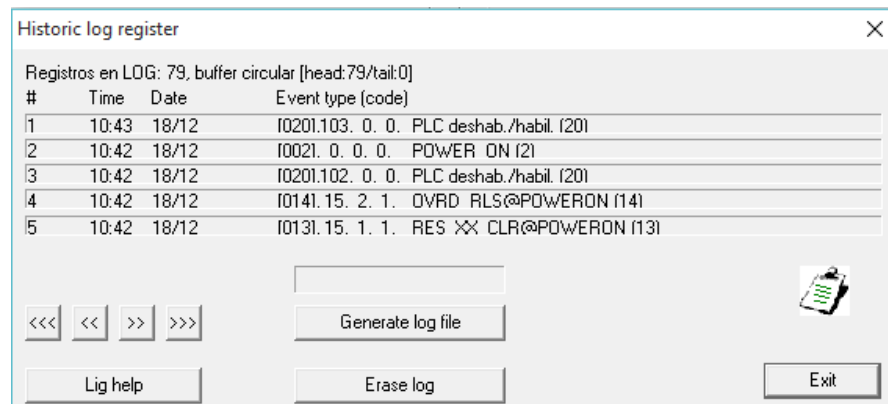
Keep in mind all files will be created in the “Active Project” directory.







If the CREATE DOCUMENTATION check box is selected, all documentation files will be created when saving the programming to disk.

1.16.20 View historic log of events of your controller

Use the events LOG to view important information analyze and/or store. This option allows you to visualize and/or store as well as erase this event log register.



With the navigation buttons  and  we can navigate forward or backward the registers from one by one, and with the buttons  and  we will do it on a ten by ten basis.

You can generate a log archive that creates the archive **LOG.INI** and will open it with the text editor **NOTEPAD.EXE**.

The events that will be stored, including the hour and date timestamp of the event are:

<u>CODE</u>	<u>NUMBER</u>	<u>DESCRIPTION</u>
• POWER_OFF	1	Power failure
• POWER_ON	2	Power returns
• RESET_REQUESTED	3	Requested reset LOCALor REMOTE
• CODE_DOWNLOAD requested on LCD	4	Upload of requested program x USB
• COM1_SLV_OFFFL	5	Slave on COM1 offline > 1 min.
• COM2_SLV_OFFFL	6	Slave on COM2 offline > 1 min.
• COM1_SLV_OK	7	Slave on COM1 returns online
• COM2_SLV_OK	8	Slave on COM2 returns online
• PT_ALARM	9	Alarmed point
• CLEAR_LOG	10	Erase LOG requested remote
• XEEPROM_REQUEST_PC	11	Erase parameters EEPROM requested from PC

- | | | |
|--------------------------|----|--|
| • XEEPROM_CLEAR | 12 | Erase parameters EEPROM confirmed on LCD |
| • RES_CLEAR_AT_POWON | 13 | Erase RES_BIT / RES_FLT programmed @ power on |
| • OVRD_RLS_AT_POWON | 14 | Free overrides COM/MAN programmed @ power on |
| • RTCLK_BATT_FAULT | 15 | Battery failure and clock, restoring the EEPROM hour +/- 20 min. |
| • DST_SUMMER | 16 | Change to summer daylight saving time |
| • DST_WINTER | 17 | Change to winter daylight saving time |
| • LOCAL_OVERRIDE | 18 | Local override by P.B. o EB-1,2 |
| • RTCLK_ADJUST_USB (USB) | 19 | Clock setting using address 0xFF=255 |

1.16.21 Reset remotely controller and set up firmware updates

When it is necessary to load a software update to the OpenBAS-HV-NX10P via the USB port (see section 11 appendix B), or to simulate a power failure, we can command the controller to a reset remotely.

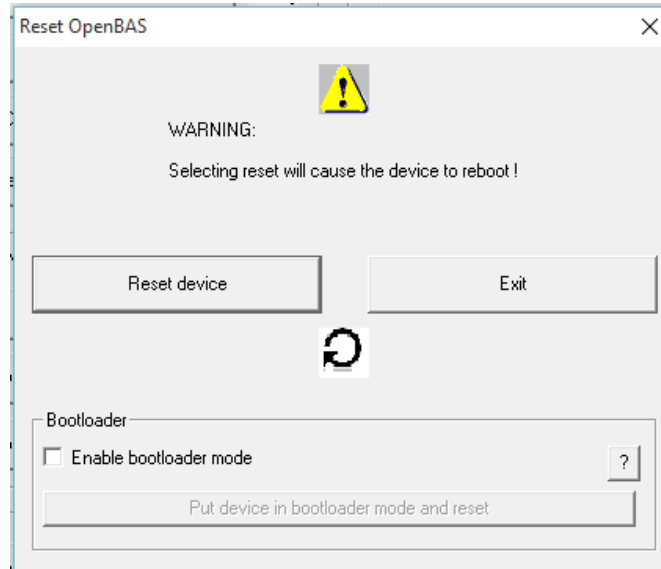


NOTE:

Before issuing a RESET to the controller, it is important to know that the OpenBAS-HV-NX10P can be controlling some important process or lighting of occupied areas or also machinery, so by executing this command you can affect some area or process.

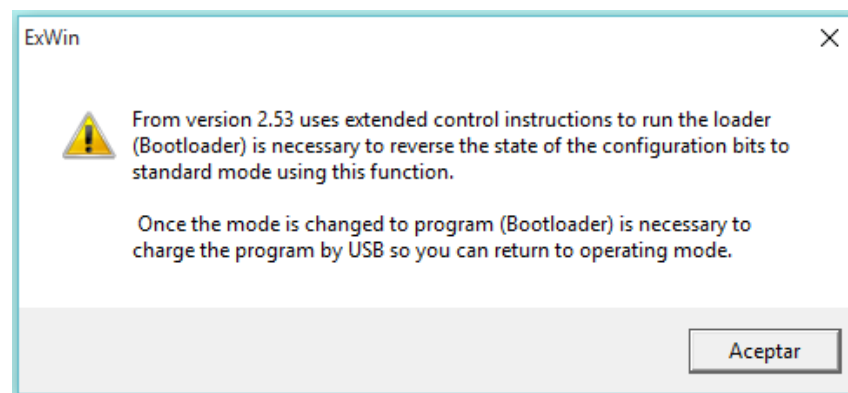
The picture in the following screen shows the dialog box for the reset command.

Reset dialog box allows for sending a software reset command to the controller.



Some confirmation will be asked before proceeding to actually reset the device.

Also if not using the extended instruction bootloader it will be necessary to activate the standard instructions bootloader. Is is only necessary for Legacy NX5 non OpenBAS controllers that don't have the extended instruction bootloader. To upgrade the bootloader, a Microchip programmer is necessary to burn the image on the FLASH memory of the controller.



For detailed instructions on how to upgrade the firmware or burn a new bootloader image see section 11 appendix B.

1.17 Reviewing version history

To view the revision history of the OpenBAS series controllers, press the VERSIONS button on top of the Password button.

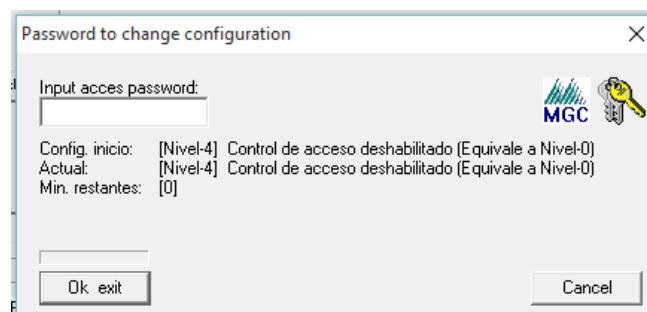


1.18 Open Password access for Password locked controller

To unlock the controller if password is enabled, press the PASSWORD button and input the appropriate password that has been provided for the three different access levels.



Passwords are case sensitive. When selecting to introduce the password option or if we try a not authorizes access, a dialog box will pop up where we can introduce the password to complete the desired function.

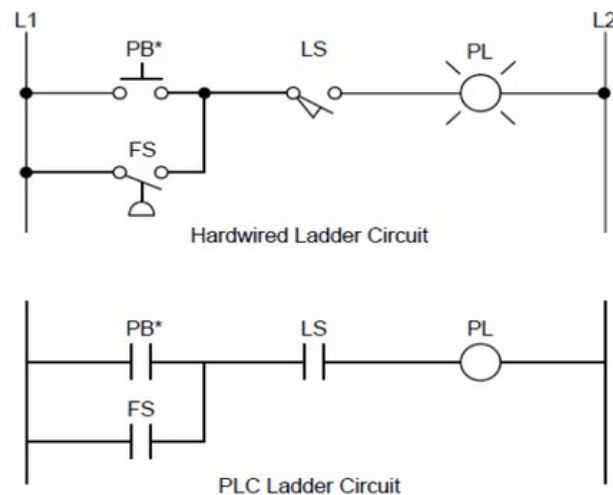


2.0 PLC Ladder Programming Basics

At the heart of the OpenBAS-HV-NX10P Building Automation Controller lies a PLC (Programmable Logic Controller), here a brief history of the PLC. In the late 1960's an American company named Bedford Associates released a computing device they called the MODICON. As an acronym, it meant Modular Digital Controller, and later became the name of a company division devoted to the design, manufacture, and sale of these special-purpose control computers. Other engineering firms developed their own versions of this device, and it eventually came to be known in non-proprietary terms as a PLC, or Programmable Logic Controller. The purpose of a PLC was to directly replace electromechanical relays as logic elements, substituting instead a solid-state digital computer with a stored program, able to emulate the interconnection of many relays to perform certain logical tasks.

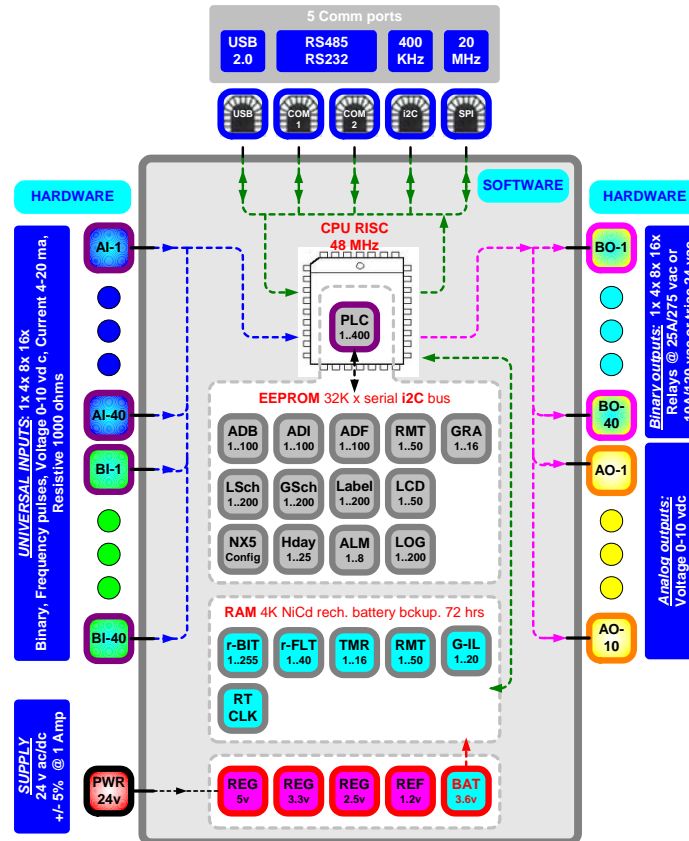
A PLC has many “input” terminals, through which it interprets “high” and “low” logical states from sensors and switches. It also has many output terminals, through which it outputs “high” and “low” signals to power lights, solenoids, contactors, small motors, and other devices lending themselves to on/off control. In an effort to make PLCs easy to program, their programming language was designed to resemble ladder logic diagrams. Thus, an industrial electrician or electrical engineer accustomed to reading ladder logic schematics would feel comfortable programming a PLC to perform the same control functions.

LADDER LANGUAGE



The name is based on the observation that programs in this language resemble ladders, with two vertical rails and a series of horizontal rungs between them. While ladder diagrams were once the only available notation for recording programmable controller programs, today other forms are standardized in IEC 61131-3.

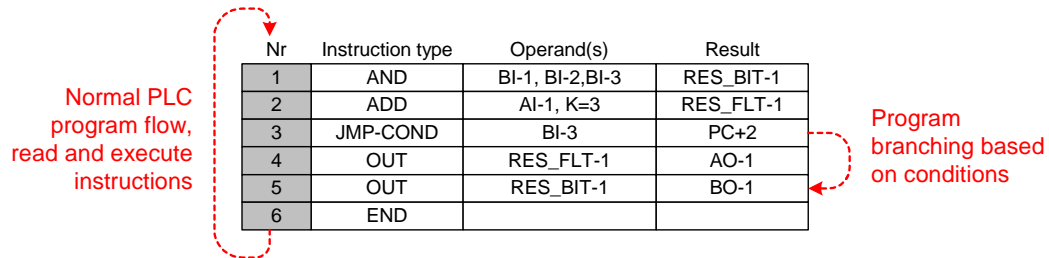
Hundreds of books have been written on writing PLC programs, and thousands of links to PLC programming and tutorials can be found on the internet. So the focus on this section to keep it simple to explain in detail Mircom's implementation targeted to Building Automation Solutions using the **OpenBAS-HX-NX10P** controller.



A simplified diagram of the **OpenBAS-HX-NX10P** controller is shown in the picture above, at the core we can see a RISC microcontroller that has a database stored in EEPROM and RAM, as well as hardware inputs to the left side and outputs to the right side, on top of the main block some communication channels are also shown.

When the controller is powered up, it starts scanning the PLC instruction database, to see what it needs to do. The basic controller has 400 instructions, so at the very beginning instruction #1 is read from the database and executed. After which the next instruction will also be read and executed, and unless a jump or call instruction is found that could divert the order of the instructions, the process will repeat until instruction #400 or an END is found. At this moment the circle will close and the program will start processing from instruction #1 again. A picture showing this relationship is shown on next page.

Picture showing normal PLC instruction execution.



Nr	Instruction type	Operand(s)	Result
1	AND	BI-1, BI-2, BI-3	RES_BIT-1
2	ADD	AI-1, K=3	RES_FLT-1
3	JMP-COND	BI-3	PC+2
4	OUT	RES_FLT-1	AO-1
5	OUT	RES_BIT-1	BO-1
6	END		

As can be seen in the picture, every instruction (also called block or equation) has a type that defines what the instruction should perform, a set of operands to perform the operation and a result register pointer that tells the instruction where to store the result of the operation.

So basically to summarize what the **OpenBAS-HX-NX10P** controller does after being powered on, is the following:

- Scan the hardware Analog and Binary Inputs to update the live database.
- Process the PLC instructions and modify the live database based on results of the instructions.
- Update the hardware Binary and Analog Outputs to reflect the PLC operations.
- Check the schedules every minute and process if any is active.
- Check if any message arrived via any communication port using the multiprotocol logic to translate messages and respond properly.
- Communicate with LCD operator display and check if keyboard is active to update interface.
- If set up as master on any Fieldbus port, poll remote points and update live database.
- If slave devices are set up in the remote point database, communicate with them to read and write outputs as frequently as possible.
- Log graphics to main memory and USB storage if available.

As can be seen in the previous list there are a lot of things to do simultaneously, and all this has to happen in real time. Latency for communications is at the 1 millisecond level, also PLC processing is done at a speed of 400-800 instructions per second, with complete processing cycle of the main loop in the 4-10 millisecond range. With all this being said, let's proceed to the nitty gritty of PLC programming.

For those in the need for simpler programming, TEMPLATE based programming which is a "fill in the blanks programming" for fast and easy system setup, and the straightforward SCRIPT programming will be explained in sections 7.12 and 7.13 respectively.

For those who really need to tune up the hardware and get the most the controller can give, section 7.15 describing advanced C programming is explained.

2.1 PLC Ladder Programming Introduction

As explained at the beginning in section 7.0 PLC programming basics, Ladder Programming is mainly targeted to allow an industrial electrician or electrical engineer or HVAC technician, who usually knows how to service or build an electromechanical control, so they will easily understand what an electronic control written in PLC Ladder is doing.

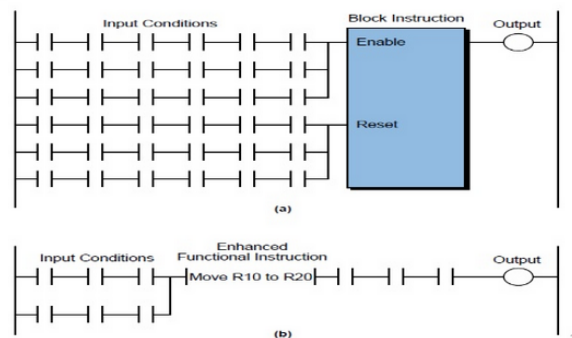
Before the advent of solid-state logic circuits, logical control systems were designed and built exclusively around electromechanical relays. Relays are far from obsolete in modern design, but have been replaced in many of their former roles as logic-level control devices, relegated most often to those applications demanding high current and/or high voltage switching.

Ladder diagram can be very complex as shown in the following illustration. And some PLCs implement hundreds of different instructions in a single block.

Ladder matrix

(a) functional block instructions

(b) Enhanced ladder format functional instructions.



To reduce the learning curve and simplify things for the programmer a RISC (Reduced Instruction Set Computing) paradigm was chosen. In this programming concept, only a limited but powerful set of instructions are implemented, and each instruction has a reduced set of operands that can range between ZERO up to EIGHT (0..4) operands. Furthermore most instructions only implement a simple ONE output register to write their result.

As a consequence of this, each PLC instruction is stored as 10 bytes in the database.

1	Instr. Type		} Operand(s), 8 BYTES
2,3	OP	1	
4,5	OP	2	
6-7	OP	3	
8-9	OP	4	
10	Result Pointer		Result register pointer 1 BYTE

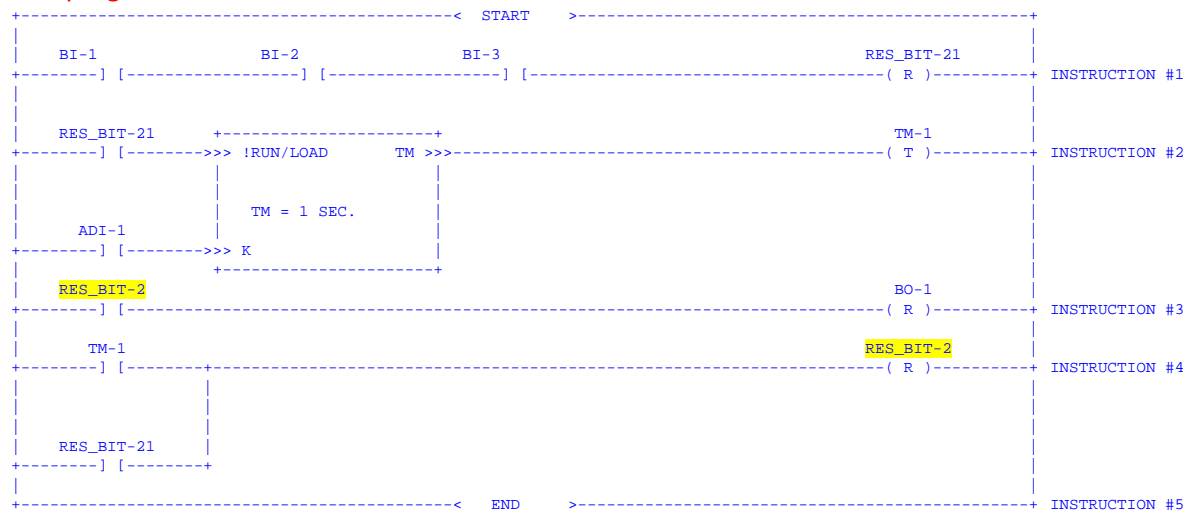
The following is a PLC instructions table containing the defined instructions for the OpenBAS-HV-NX10P controller which are sometimes also known as blocks or equations:

Instruction TYPE ID	Instruction Name	Instruction Description	Operation Result Register(s)	Operand(s)	Number of Operands
0	NULL	Do nothing instruction	-	-	-
1	LIGHT_GROUP	Outputs the operand into 1 and up to 8 Binary Outputs	Rly-n..n+8	RES_BIT-n	1
2	AND	Boolean logic AND instruction	RES_BIT-n	Any	1..4
3	NAND	Boolean logic NAND instruction	RES_BIT-n	Any	1..4
4	OR	Boolean logic OR instruction	RES_BIT-n	Any	1..4
5	NOR	Boolean logic NOR instruction	RES_BIT-n	Any	1..4
6	XOR	Boolean logic XOR instruction	RES_BIT-n	Any	2
7	NXOR	Boolean logic NXOR instruction	RES_BIT-n	Any	2
8	INVERT	Boolean logic INVERT instruction	RES_BIT-n	Any	1
9	ADD	Math instruction ADDITION	RES_FLT-n	Any	1..4
10	SUBSTRACT	Math instruction SUBTRACTION	RES_FLT-n	Any	1..4
11	MULTIPLY	Math instruction MULTIPLICATION	RES_FLT-n	Any	2
12	DIVISION	Math instruction DIVISION	RES_FLT-n	Any	2
13	>	Compare instruction GREATER THAN	RES_BIT-n	Any	2
14	>=	Compare instruction GREATER OR EQUAL THAN	RES_BIT-n	Any	2
15	<	Compare instruction LESS THAN	RES_BIT-n	Any	2
16	<=	Compare instruction LESS OR EQUAL THAN	RES_BIT-n	Any	2
17	==	Compare instruction EQUAL THAN	RES_BIT-n	Any	2
18	!=	Compare instruction NOT EQUAL THAN	RES_BIT-n	Any	2
19	TIMER	Timer instruction, decrements when RES_BIT=0, else reloads timer with ADI value	TMR-n	RES_BIT-n ADI-n	2
20	JUMP	Conditional or unconditional jump	-	RES_BIT-n	1
21	OUT_ASSIGN	Outputs source operand into destination operand	DESTINATION	SOURCE	1..2
22	PROP_CTRL	Proportional control, calculates output value based on: Process Variable, Set Point, Proportional band, Min, Max, Integration	RES_FLT-n	PV, SP, PB MIN, MAX, INT	5
23	TOTALIZER	Totalizes or accumulates number of state changes	RES_FLT-n ADF-n	Any	1
24	MIN	Math instruction calculates MIN of operands	RES_FLT-n	Any	2..4
25	MAX	Math instruction calculates MAX of operands	RES_FLT-n	Any	2..4
26	AVG	Math instruction calculates AVERAGE of operands	RES_FLT-n	Any	2..4
27	ALT_SIM	Alternate and paralleling (simultaneous output) of up to eight circuits	Rly-n..n+8 or RES_BIT-n..n+8	PV, STG, LD TMn, TMn+1 SP, PB	2..7
28	HR_CNT	Hour counter, counts when control register >= 1	RES_FLT-n ADF-n	Any	1
29	LABEL	Stores a text label, used for programe documentation	-	-	-
30	SUB_CALL	Subroutine call conditional or unconditional	-	RES_BIT-n	1
31	AHU	HVAC Air Handling Unit / Roof top Unit stage	ANY	RUN, SP, DIF, TM-n, TM-n+1, TM-n+2, H/C	8
32-99	FREE	Not assigned yet	-	-	-
100-127	SPECIAL	Special instructions to be implemented in C Language	Any	Any	0..8
128-254	DISABLED	Disabled instructions	-	-	-
255	END	End instruction	-	-	-

So if each instruction only does one very simple thing, how is it that a complex program gets created? The “**glue**” to link PLC instructions into a big usable program are the **database objects**, for example to create a complex sequence that will do the following:

- Turn ON a light in a room only when all three doors are closed.
- The doors are sensed by means of magnetic switches connected to Binary Inputs.
- The light will remain ON for at least one minute, even if any of the doors are open during that period.
- When any door has remained open for more than one minute, the light will be turned OFF.

This program can be written as follows:



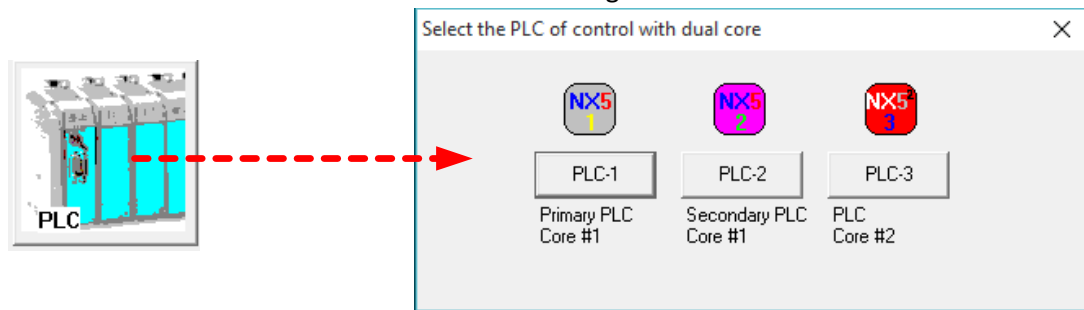
- Instruction #1 does the AND Boolean logic of the Binary Inputs 1, 2 and 3, and sets RES_BIT-1 to ONE when all three doors are closed (magnetic switch closed), if any door is open the RES_BIT-1 will be ZERO.
- Instruction #2 is a timer that when the !RUN signal is at ZERO starts decrementing and when it is ONE loads the timer with 60 seconds, so we take the input from the previous instruction and use it as a! RUN/LOAD operand of the timer. That will SET the timer to 60 every time all three doors are closed, and decrement to all the way to ZERO when any door is open.
- Instruction #3 outputs the current value of RES_BIT-2 register to the Binary Output 1, that turns the light ON or OFF.
- Instruction #4 is a Boolean OR instructions that will set RES_BIT-2 to ONE if all three doors are closed, or until the Timer expires after any door is opened.
- Instructions #5 is an END instruction that tells the PLC there is nothing else to do beyond this point.

One thing to make note about, is that in instruction #3 we are making use of the register RES_BIT-2 (highlighted in yellow) that has not been set until instruction #4. But because we know the PLC will cyclically process all the instructions, on the next pass it will use the result of instruction #4 as an

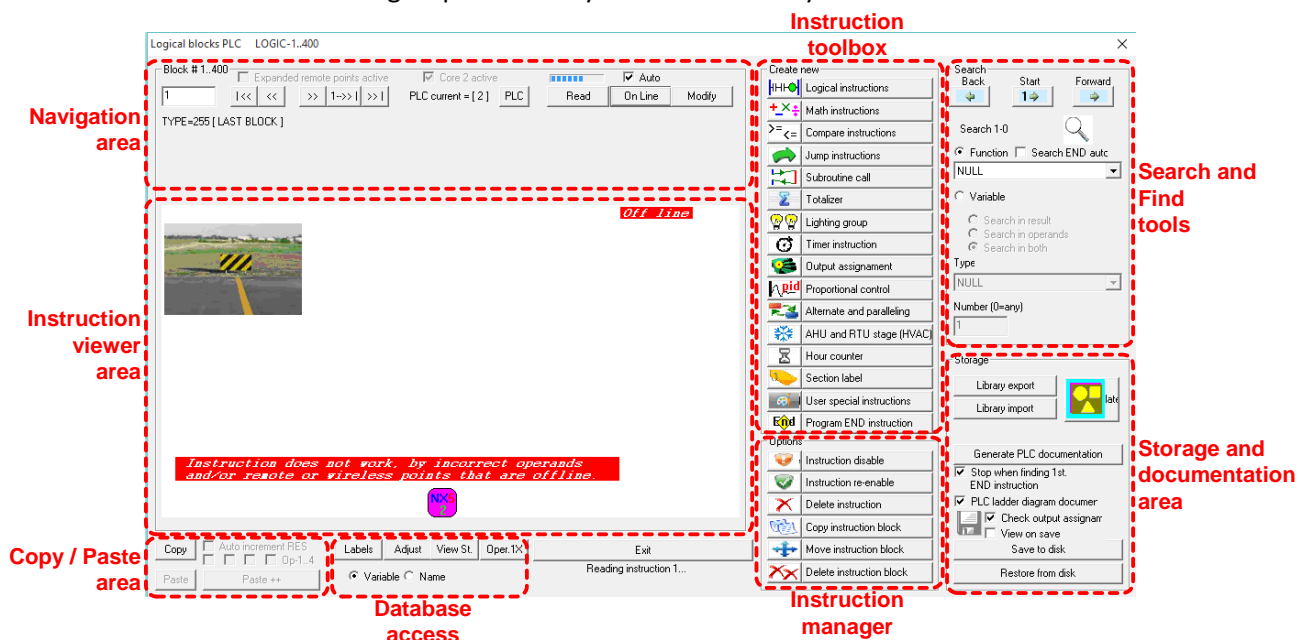
operand in instruction #3. Even while this might cause a delay of some milliseconds on the output, it is often not noticeable. PLC programming is usually very forgiving with OUT OF ORDER instructions.

2.2 OpenBAS-HV-NX10P PLC Canvas

With all the latter being said, let's jump into explaining the PLC canvas of the OpenBAS-SW-CFGTL software. First of all, if a Dual Core is installed there will be not one but three active PLC's, and upon entry we need to select with which PLC we will be working with.



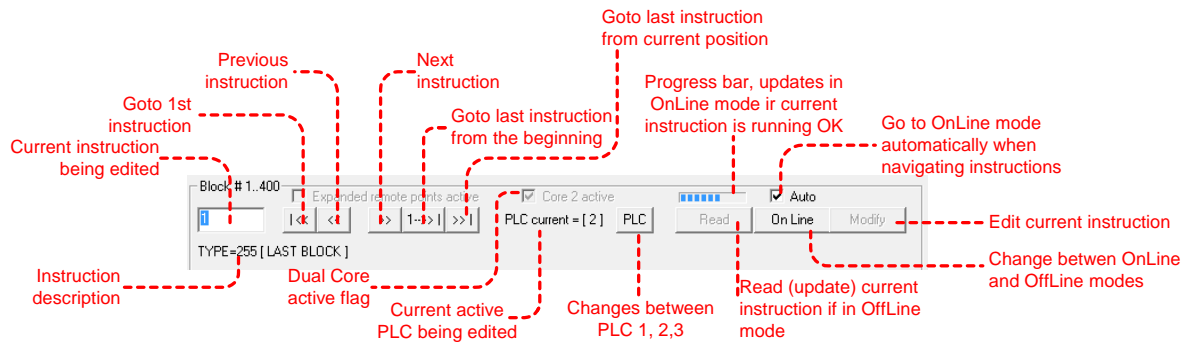
Now that the correct PLC has been chosen, the PLC canvas will occupy the focus of the screen. It is divided into several sections that group controls by their functionality.



On the next sections, each one of the controls of the PLC instruction editor (PLC canvas) will be described with great detail.

2.3 Moving Through your PLC Ladder Logic Program

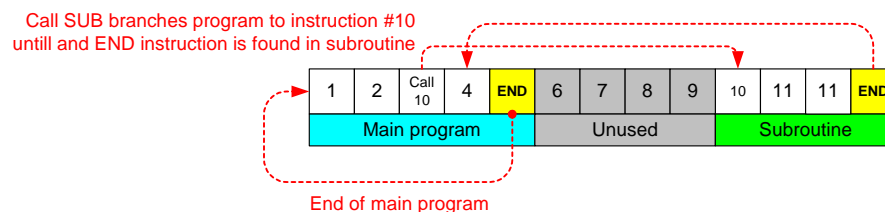
The navigational tool allows you to transverse back and forth through all the 400 instructions of the PLC. To go to a specific instruction simply type the instruction number between 1 and 400 in the edit field of the current instruction being edited.



The arrow buttons allow you to move forward or backward, go to the first and last instructions. The last instruction is defined as the point where the PLC will revert to instruction #1. And is where it finds the last active END instruction.

When the PLC comes from the factory, all 400 instructions contain the END instruction, so as you move forward in the program, those END instructions will be replaced by active instructions.

When calling a subroutine that might be above the END instruction that reverts to instruction #1 again, the END instruction found inside the subroutine only marks the END of the subroutine and not the end of the program. This is depicted in the image below.



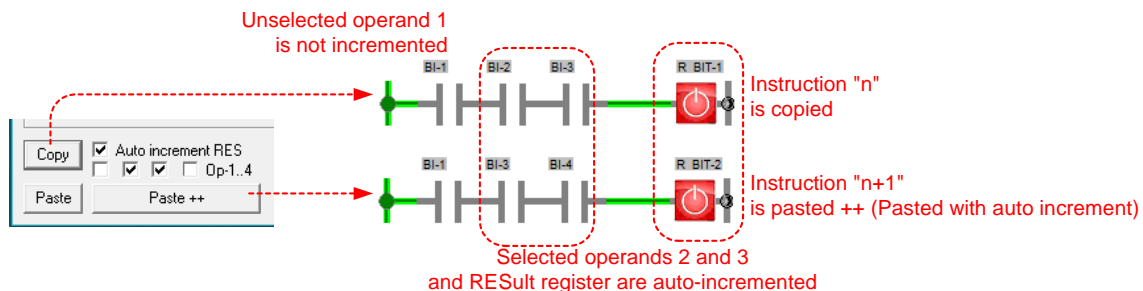
The “**OnLine**” mode as its name implies is On Line with the PLC and everything that changes in the process will be reflected on the screen. When the “**OffLine**” mode is selected the connection between the software and the controller is still active and the PLC is still doing its job. However the screen updating is stopped to enable editing the current instruction, save or restore the program to and from disk, and enable the documentation feature. It will become evident as the “instruction viewing area” will reflect the OFF-LINE mode in red color and all editing buttons will become active.

2.4 Copy and Paste of a PLC Instruction

A nice feature when creating programs is the capability of copying an instruction and pasting it. The PLC editor goes a step further by enabling auto incrementing of:

- Instruction number
- Operands
- And result register

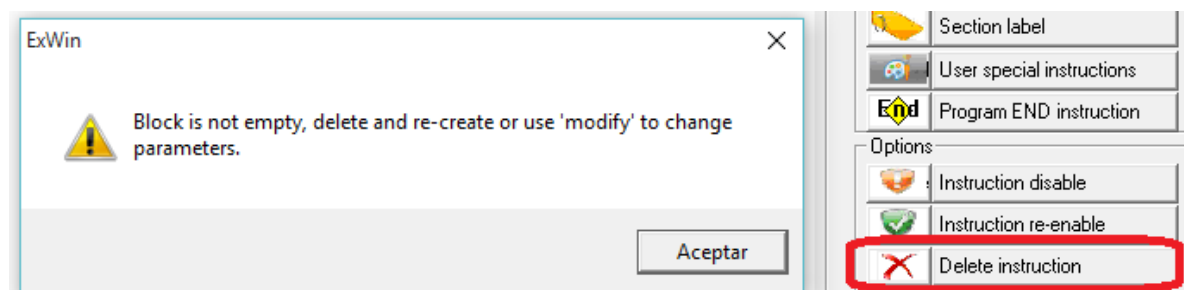
So the next instruction being copied will have the selected operands and optionally the result register auto incremented, also before pasting the instruction, the current instruction is auto incremented thus saving keystrokes and time.



The simple “Paste” instructions works as expected, as it pastes a previously copied instruction in the currently selected instruction just the way it is without modifying anything. The paste instruction will overwrite whatever the current instruction might already contain.

2.5 Create a new PLC Instruction

Before creating an instruction, if the instruction already contains something a warning message will tell you that it first must be erased. That is done with the DELETE INSTRUCTION button.



Once the current instruction is empty (containing a NULL instruction) you can proceed to add a new instruction.

2.6 Disabling, Enabling, and Erasing PLC Instructions

Sometimes when creating a program things don't work as expected, and it is a nice feature to be able to disable an instruction that you might think is causing a problem without actually having to erase it. The INSTRUCTION DISABLE does exactly that, as it marks the current instruction as being "disabled", and the PLC will no execute it.

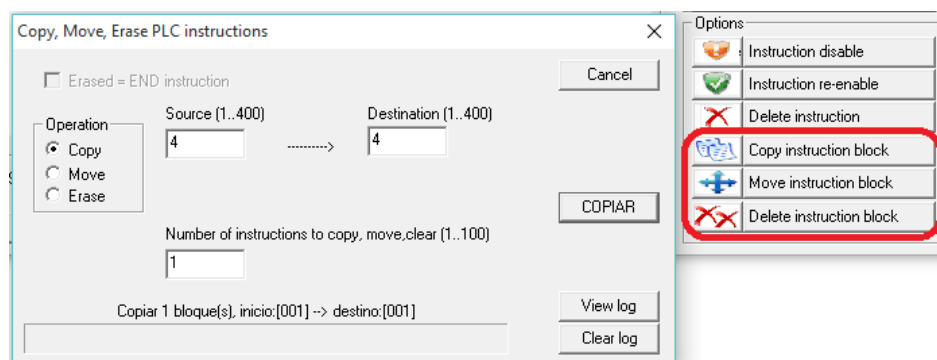


When you want to bring it back to action again, just position the current instruction on the disabled instruction and use the INSTRUCTION RE-ENABLE button.

If you decide that a specific instruction is not needed anymore, then the DELETE INSTRUCTION button will erase the currently selected instruction and place a NULL instruction instead. Null instructions are special instructions that when loaded by the PLC executive are simply ignored.

2.7 Copy, Move or Erase a Block of PLC Instructions

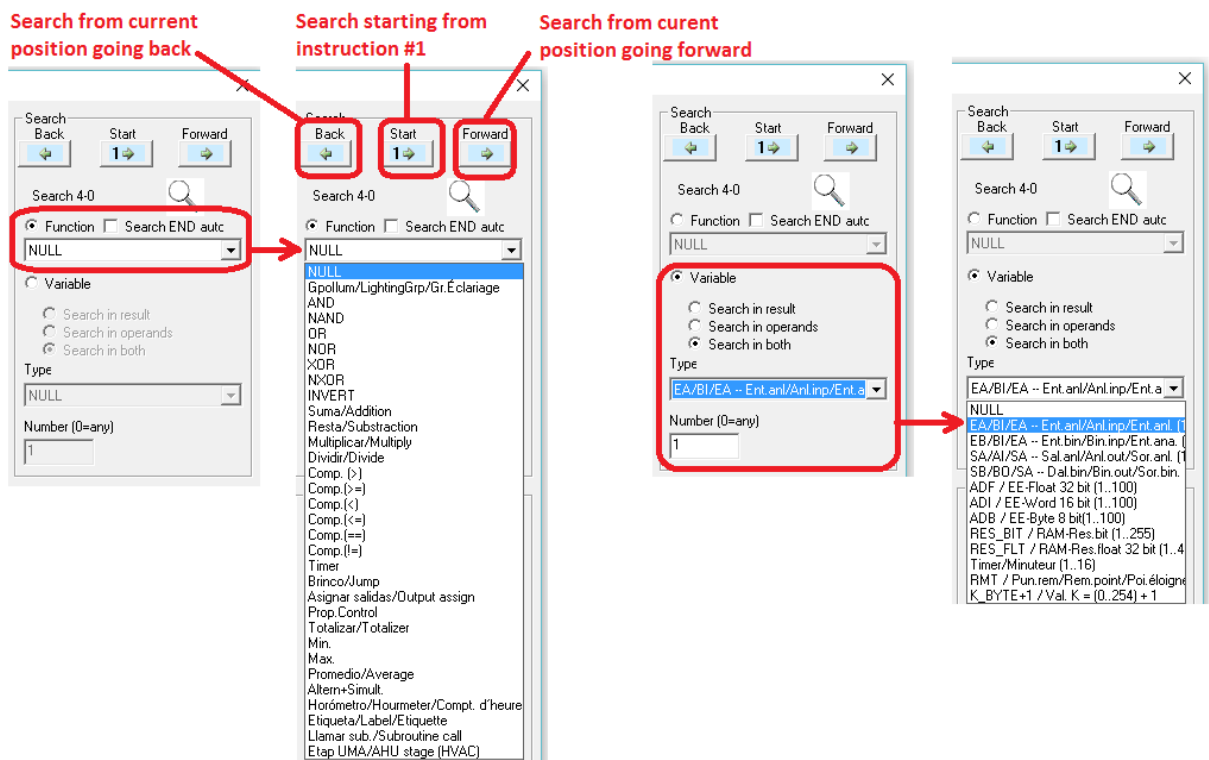
If there is a block of instructions that needs to be copied, or moved (repositioned) or erased (cleared), there is a group of three buttons that do precisely that task in block 1 of up to 100 instructions.



When any of the three buttons is depressed, a dialog box containing the source instruction, the destination instruction, as well as the number of instructions to copy, move, or clear pops up. Before actually doing the copy, move or erase a second pop up dialog will show asking you to confirm the selected operation.

2.8 Search a specific Instruction or Database Variable in your PLC Program

As the program grows, sometimes it becomes difficult to remember where a specific instruction is, or where a database object gets used as either an operand or as a result. This is where the search feature comes in; helping the programmer to find anything he or she is looking for.



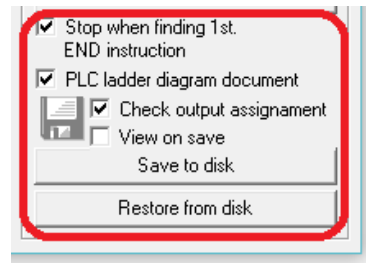
If you are looking for a specific FUNCTION (or instruction TYPE), select the option in the left image, and select the desired instruction function (instruction TYPE) to look for. You can select to start the search moving backwards or forwards from the current position, or to begin the search starting from instruction number one.

The same applies if you are looking for a specific VARIABLE from the database, in the case in the right of the image; a "variable" can be searched when it is used as either an operand, or as a result or both. You can search for a specific variable. For example AI-1 (Analog input #1), or for any Analog Input, in this case selecting AN-0, ZERO is a special case that will search the first occurrence of any Analog Input, and stop the search when it is found.

The same as when looking for a function, when searching for a variable, you can select to start the search moving backwards or forwards from the current position, or to begin the search starting from instruction number one.

2.9 Save and Restore your PLC Ladder Logic Program to and from Disk in your PC

When you work with the PLC editor, you are directly modifying the ON-LINE database running inside the PLC. If you want to store any changes made to the disk of your PC for storage or for transferring the same program to another controller, you can easily do it by just selecting the “Save to Disk” or “Restore from Disk” buttons when enabled.



- If the STOP WHEN FINDING 1st INSTRUCTION checkbox is disabled, the 400 instructions will be saved to disk. If it is enabled (default) it will save only the valid program up to the 1st valid END instruction.
- The PLC LADDER checkbox will toggle between documenting as LADDER or INSTRUCTION LIST styles.
- The CHECK OUTPUT ASSIGNMENT if enabled, will look for duplicate output assignments that could potentially create error in the PLC operation, as an output could be commanded by two different conflicting logics and toggle between an ON / OFF state erratically.
- The VIEW ON SAVE option if enabled, shows each instruction in the view as it is being saved, slowing down the save to disk process.

The SAVE TO DISK button will transfer the ON-LINE program to disk, and if a file already exists in your computer it will ask if you want to override it.

The RESTORE FROM button will do exactly the opposite, it will transfer the program stored on disk into the ON-LINE database. It will first ask if you really want to override the program currently running inside of the PLC with the one stored on disk.



NOTE:

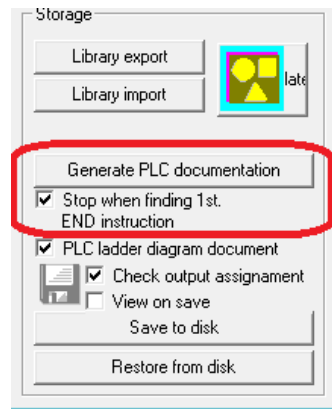
In controllers with DUAL CORE that have three PLC's, each PLC must be saved or restored independently of the others.

Keep in mind that the PLC program is just a piece of the whole database, and it is dependant on other components such as: Schedules, Setpoints, Remote points, Comm port configuration, etc.

2.10 Generate your PLC Documentation

Documenting a program is always a need when deploying a Building Automation Solution, as the customer is expecting an “As Built” not only in the electrical wiring but also in the programming logic itself.

This is where the documenting feature of the OpenBAS-HV-NX10P becomes so important as the PLC is the core of the program, and here all the database objects will show it’s relationship. For the final documentation is always good to have a printout of your program to be delivered to the customer for service and future reference.



The GENERATE PLC DOCUMENTATION button does exactly that, it generates a printout of the PLC program to enable the programmer to document it’s program.

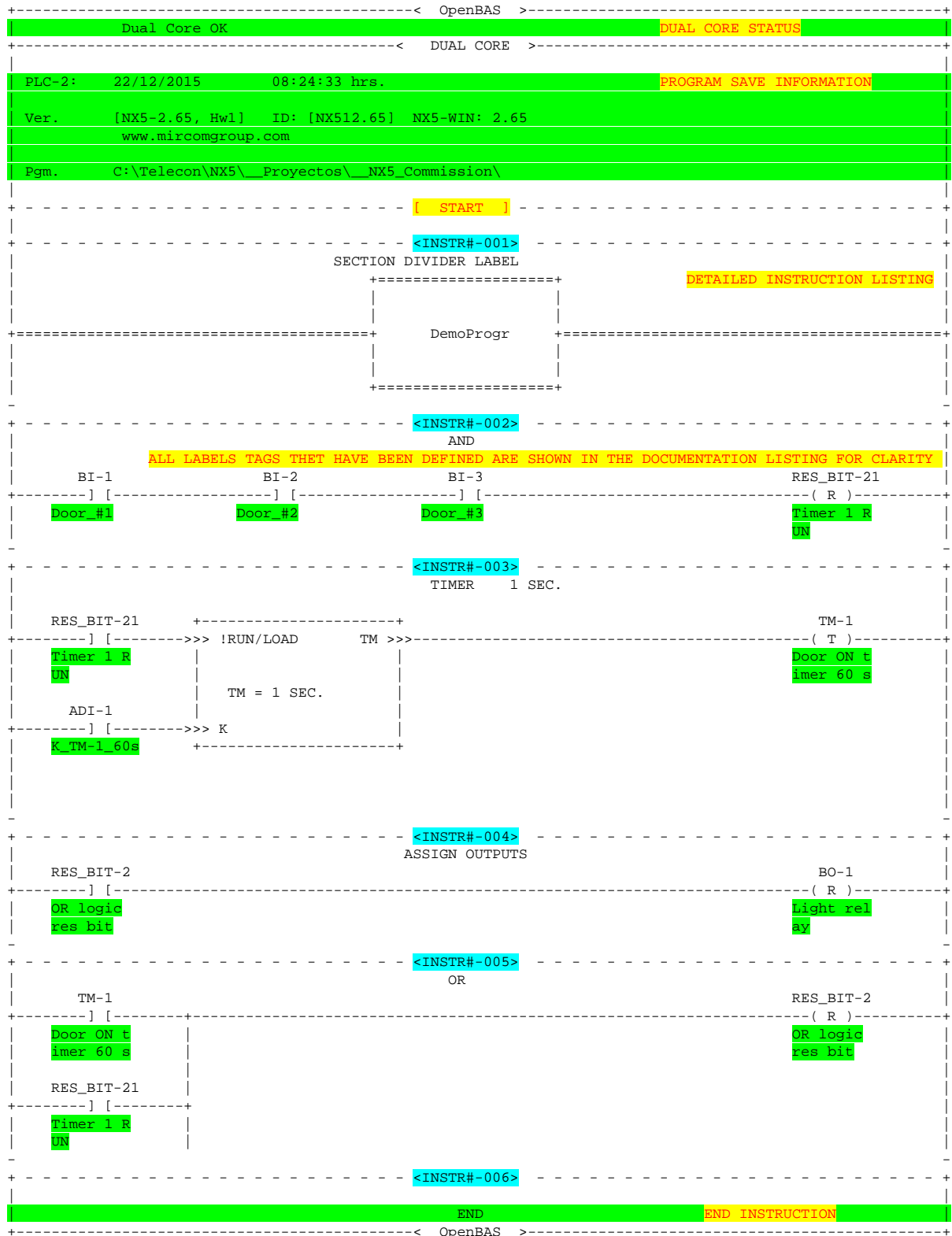


NOTE:

In controllers with DUAL CORE that have three PLC’s, each PLC must be saved or restored independently of the others.

Keep in mind that the PLC program is just a piece of the whole database, and it is dependant on other componentssuch as: Schedules, Setpoints, Remore points, Comm port configuration, etc.

We will now analyze the demo program we created back in section 7.1 to demonstrate how a simple program works. On the next page the full program listing is shown, **highlighting in yellow** the most important features of the documentation (some lines of the documentation listings have been removed for clarity and due to space limitations).



```

=====
Resume: 5 instructions used from 400, 98.75% available
=====
=====
TYPE          # Count
-----
TYP_AND       1
TYP_OR        1
TYP_TIMER     1
TYP_OUT_ASGN  1
TYP_SECTION_LABEL 1
TYP_END       1 (FREE)
=====

CROSS REFERENCES
=====
BINARY INPUTS
BI-01      Name: [Door_#1]
           found as operand: 1
           found as result: 0
           total found 1
           In instr.#: [002],
BI-02      Name: [Door_#2]
           found as operand: 1
           found as result: 0
           total found 1
           In instr.#: [002],
BI-03      Name: [Door_#3]
           found as operand: 1
           found as result: 0
           total found 1
           In instr.#: [002],
=====
BINARY OUTPUTS
BO-01      Name: [Light relay]
           found as operand: 0
           found as result: 1
           total found 1
           In instr.#: [004],
           Warning, this variable is not being used anywhere in the program.
=====
EEPROM WORD ADI
ADI-01     Name: [K_TM-1_falla_energ]
           found as operand: 1
           found as result: 0
           total found 1
           In instr.#: [003],
=====
RAM RES_BIT
LIGHT_GRP-2
RES_BIT-02 Name: [OR logic res bit]
           found as operand: 1
           found as result: 1
           total found 2
           In instr.#: [004], [005],
BIT_TIMER-1
RES_BIT-21 Name: [Timer 1 RUN]
           found as operand: 2
           found as result: 1
           total found 3
           In instr.#: [002], [003], [005],
=====
TIMER STATE
TM_ST-01   Name: [Door ON timer 60 s]
           found as operand: 1
           found as result: 1
           total found 2
           In instr.#: [003], [005],
=====
Total warnings found: 1

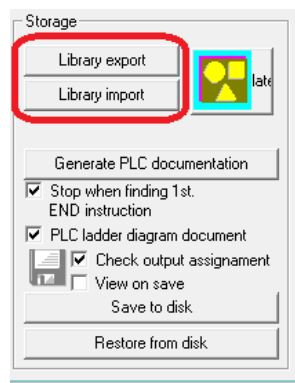
=====
Jump and subroutine call graph, Last used instruction number = (6)
=====
#      Instruction      Notes
==
001    [SECTION_LABEL]  Name: [DemoProgr]
002    [AND]            ]
003    [TIMER]          ]
004    [OUT_ASSIGN]     ]
005    [OR]             ]
006    [PROGRAM_END]    ***( Last PLC instruction )***

```

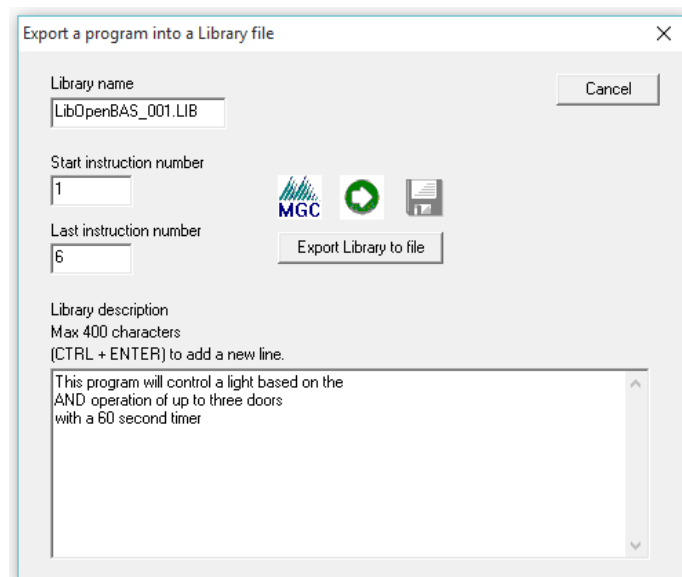
2.11 Library Import and Export

When you create a subroutine or a piece of code that works, and want to copy into another program, the Library Import and Export do exactly that. When you are inside the PLC editor and want to copy a piece of code into the same PLC program simply use the COPY BLOCK instruction that was described back in section 7.7.

But if you want to copy a section or a block of instructions into another PLC or into another controller or simply want to create a Library for future use, this command is the one to use.



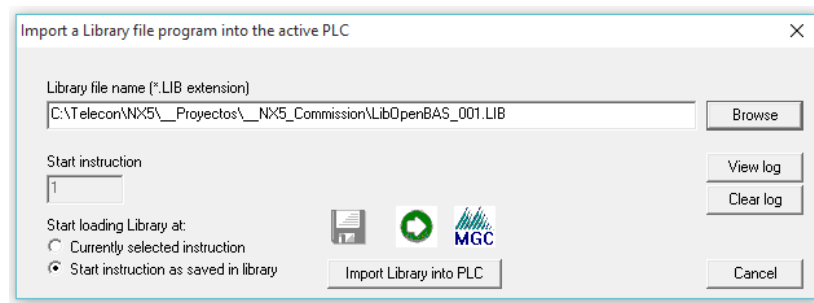
By selecting the LIBRARY EXPORT button, a dialog box appears in where we can select: The name of the file under which the Library will be saved, the start and end instructions range, and finally a 400 characters long descriptive text of what this library does. It is important to fully describe the purpose of the library, so when you or someone else comes back to it later, fully understands what this Library is intended to do.



Now that the library has been successfully exported to a file on disk, we can proceed to import it into another PLC in a Dual Core controller, or into a completely different device.

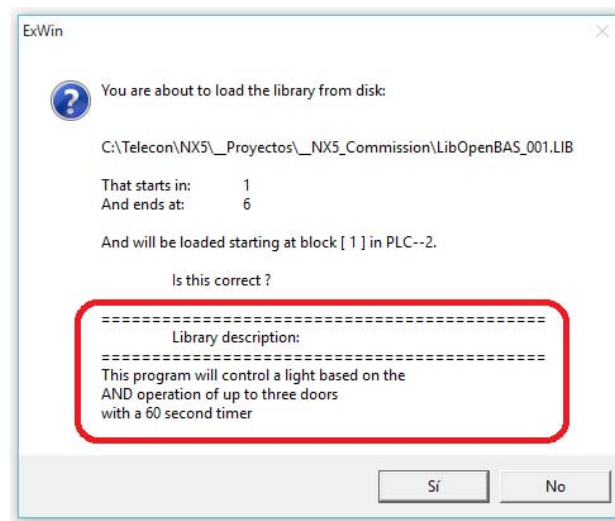
By selecting now, the IMPORT LIBRARY button, a dialog pops up where you can browse the desired library file. Then select where in the PLC the Library will be imported, there are two options:

- The current instruction position in the PLC
- Or the same position that the Library had when it was saved



Now that the library has been successfully exported to a file on disk, we can proceed to import it into another PLC in a Dual Core controller, or into a completely different device.

When the IMPORT LIBRARY TO PLC button is selected, the following dialog appears:



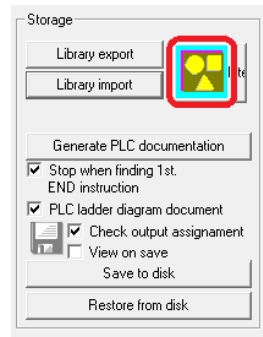
Now it becomes apparent why filling in a descriptive text of what the library is supposed to do was so important, because that same information is now showed on the IMPORT dialog to let you know if this is the correct Library you need.

2.12 Template Based programming

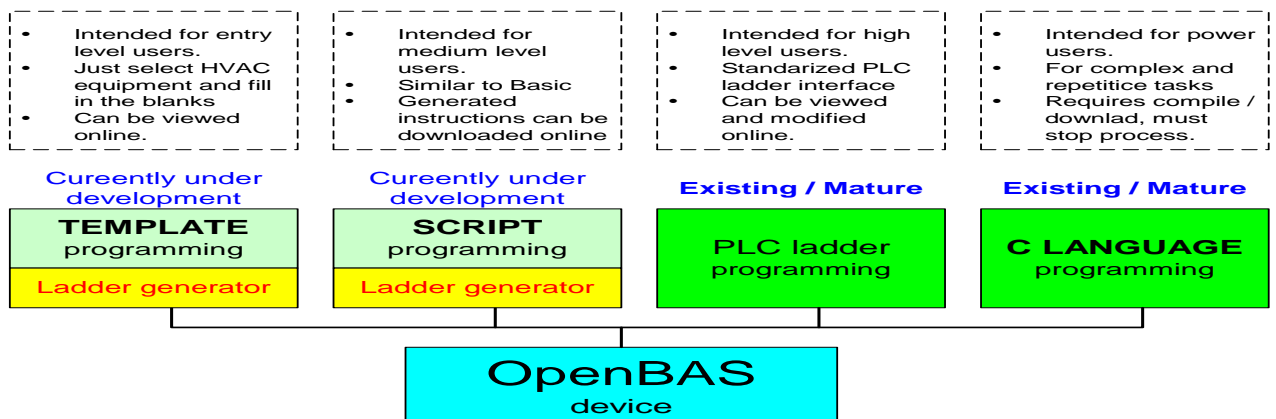
Template based programming is an easy entry level programming currently at development by Mircom Engineering Group, in where a library of templates tailored for specific Building Automation Sequences will allow you to create with pre-existing templates, a complete sequence of operation for:

- Air Handling Units
- Roof Top Units
- VAV boxes
- FAN & COILS
- Lighting controllers
- Chillers
- Boilers
- Fan and pump arrays
- Demand Limiting and Load Rolling strategies
- Energy metering sequences

Template programming is a fill in the blanks process, in where a blank template is chosen and with the help of an “Automation Wizard” a full automation scheme can be prepared and downloaded into a controller:



Template based programming briefly described here along with Script programming described in section 7.13 and the Advanced C programming described in section 7.14 complement the PLC programming for the OpenBAS series of controllers to cover all the gaps in programming for Building Automation Systems.



2.13 Script Based Programming

In Script based programming, the user enters into a dialog box the control sequence in plain English using a structure similar to Basic. Different control sentences are there combined and translated to a PLC instruction program for downloading to the controller.

A sample program that resembles the demo described in PLC Ladder diagram in section 7.1 can be written as follows:

```
// Section defining variables used tying them to hardware inputs and outputs
DOOR_1 = BI_1
DOOR_2 = BI_2
DOOR_3 = BI_3
DOOR_SUM = RES_BIT_1
LIGHT = BO_1
DOOR_TIMER = TM_1
// Generic definitions for ON/OFF and OPEN/CLOSED
ON=OPEN=1
OFF=CLOSED=1
// Logic sentences using Boolean logic
if DOOR_1 is OPEN and DOOR_2 is OPEN and DOOR_3 is OPEN then let DOOR_SUM be 1
if DOOR_SUM is not 0 then let DOOR_TIMER be 60
if DOOR_TIMER is not 0 or DOOR_SUM is not 0 then let LIGHT be ON else let LIGHT be OFF
```

As can be seen in the above program, first plain names are defined so they can be used down in the program to make it very easy to follow.

Then in the logic sentences, colored syntax is used to highlight the Script language reserved keywords. The above program can be easily re-written into the following with exactly the same results:

```
// Redefine as FLOAT instead of BIT to keep track of number of doors open
DOOR_SUM = RES_FLT_1
DOOR_SUM = DOOR_1 + DOOR_2 + DOOR_3
// Logic sentences modified to use Math logic instead of Boolean logic
if DOOR_SUM >= 1 then DOOR_TIMER = 60
if DOOR_TIMER != 0 or DOOR_SUM >= 1 then LIGHT = ON else LIGHT = OFF
```

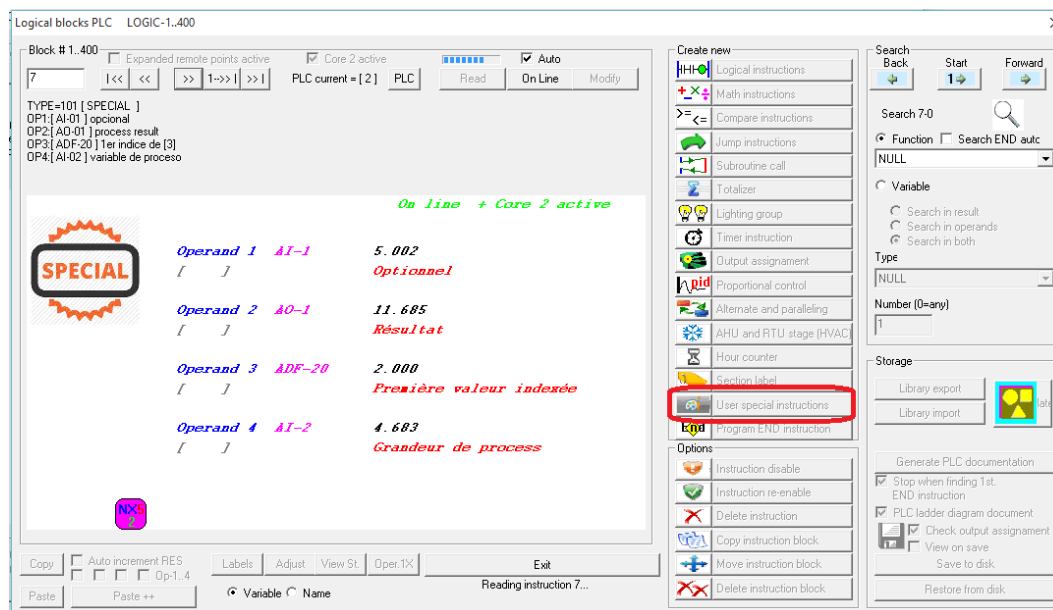
Both sentence sequences do the same job even while syntactically different, and both will produce a very similar PLC generated program. With this Scrip language the basic functionality of getting something done will be solved very easy in plain English.

2.14 Advanced C Programming for repetitive and advanced users

When the need comes that a very complex or repetitive automation task has to be implemented, there is no substitute for a high level language. This is where the C Language comes to the rescue.

The OS of the OpenBAS-HV-NX10P is fully written in the C Language, and libraries are available to create specific programs and be able to control them from the OpenBAS-SW-CFGTL software.

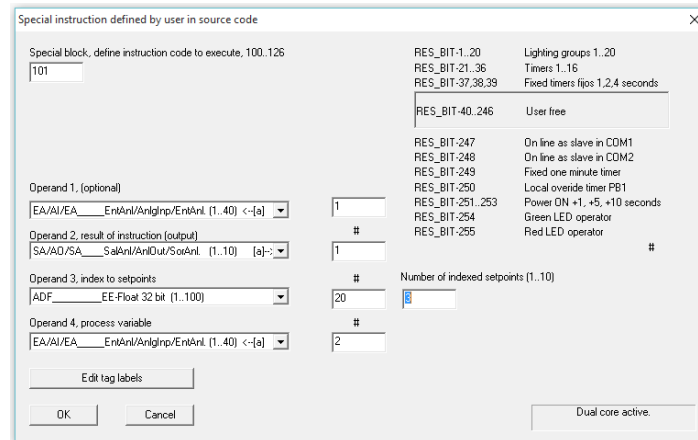
For this purpose, instructions from TYPE 100 and up to 127 have been reserved for this task. When a special instruction is set up in the PLC editor.



Here the power user can fine tune the application and the full set of tools that the OpenBAS OS has, are available to use in his or her C Language program.

This of course requires installing an IDE that is available from Microchip and a suitable C compiler for the PIC18 series of microcontrollers to be able to compile the source code and create a special version of the OS tuned to suit your automation needs.

By selecting multiple blocks, a special instruction can be reused by simply changing the database objects that it operates into.



Here is a sample code of code written in the C Language, that is used to calculate superheat in a refrigeration valve:

```

/*****
void specialSuperheatCalculation( void )
{
    //////////////////////////////////////
    // To calculate superheat from a valve we convert the read pressure and subtract the read temperature
    // this will give a value to protect a compressor if superheat falls below 15°F
    // OPERAND 1 Actual read pressure
    // OPERAND 2 Variable from database to store superheat calculation
    // OPERAND 3 Not used, set to null
    // OPERAND 4 Actual temperatura read
    float          valActualPressure;
    float          valActualTemperature;
    float          valTemperatureSuperheat;
    unsigned short pressure10X; // Table with conversion values stored
    unsigned char  i, x;

    // Read indexed values
    valActualPressure =     especial.operand1;
    valActualTemperature =  especial.pv;

















    pressure10X = (unsigned short) (valActualPressure * (float)10.0);
    // Do an educated guess to start in the middle of the index table
    x = ( pressure10X < temp_to_psig_R22[ 60 ] ) ? 0 : 60;

    // Now index upper and lower sections of table to find
    for( i = 0; i < 60; i++ )    superheat value
        if( temp_to_psig_R22[ i + x ] >= valTemperatureSuperheat )
            break; // When temperature is found, exit loop
    // Now superheat temperature is = ( index + LOoHI in table ) - 40°F,
    // first value in table is -40°F
    valTemperatureSuperheat = (float)(i + x) - (float)40.0;
    // Now fill result with value
    especialWriteResult( valorTemperaturaSuperheat );
}

```

3.0 PLC Ladder Programming Detailed Programming Description

In this section the detailed programming of each one of the logic instructions is described. The instruction toolbox is shown in the figure below, it has both a graphical icon to the left and a text description to the right that briefly describes it's function.

Instruction groups:	Described in section:
Create new	
 Logical instructions	8.1 Boolean instructions AND, NAND, OR, NOR, XOR, NXOR, INVERT
 Math instructions	8.2 Math instructions Addition, Substraction, Multiplication, Division
 Compare instructions	8.3 Compare instructions >, >=, <, <=, == Equal), != (Not equal)
 Jump instructions	8.4 Jump instructions Conditional and unconditional program branching
 Subroutine call	8.5 Subroutine call instructions Conditional and unconditional subroutine calling
 Totalizer	8.6 Totalizer Total accumulator, period totalization, energy totalization
 Lighting group	8.7 Lighting Groups Creating a lighting group
 Timer instruction	8.8 Timer Instructions Timer, free run oscillator, value to frequency converter
 Output assignment	8.9 Output Assignment Making things happen in the real world
 Proportional control	8.10 Proportional Control PID for automation and HVAC applications
 Alternate and paralleling	8.11 Alternate and Paralleling Alternate and paralleling of pumps and machinery
 AHU and RTU stage (HVAC)	8.12 AHU and RTU staging Create easily simple or complex HVAC sequences
 Hour counter	8.13 Hour Counter Create an hour counter
 Section label	8.14 Section Labels Organize and document your PLC programs
 User special instructions	8.15 Special User Programs Link your interface to user "C language" created Instructions
 Program END instruction	8.16 Program End Insert END instructions to terminate PLC or exit subroutine calls

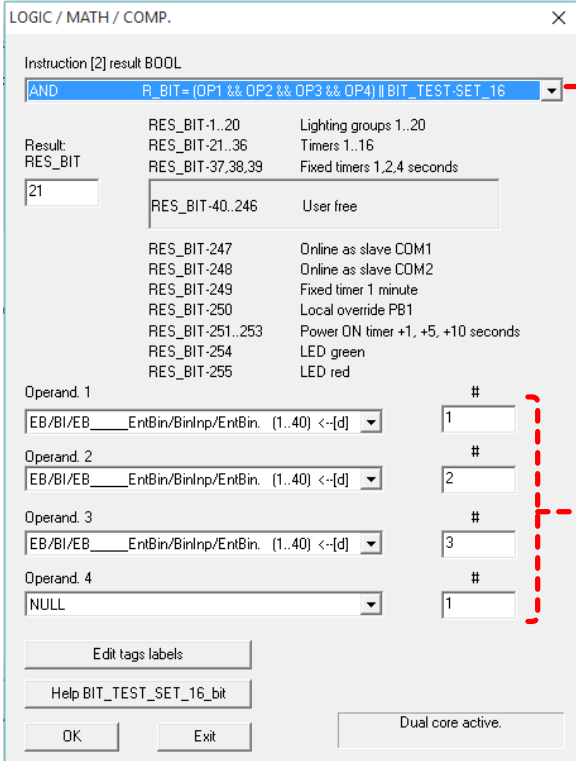
As described in section 7.3 Moving Through your PLC Ladder Logic Program, the instruction buttons are only enabled while in the OFF-LINE mode. An instruction that currently exists must be first erased if a different instruction needs to be put in place if it belongs to a different group of instructions, for example if you want to replace a Boolean instruction with a Timer instruction.

The rest of section 8 fully describes each one of the instruction groups in detail.

3.1 Logical Boolean Instructions, AND, NAND, OR, NOR, XOR, NXOR, INVERT

When this instruction is selected a dialog box that has common functionality for the following group instructions will open

- Logic functions AND, NAND, OR, NOR, XOR, NXOR, INVERT
- Mathematic functions ADD, SUBTRACT, MULTIPLY, DIVIDE
MINIMUM, MAXIMUM, AVERAGE
- Comparison functions >=, >, <, <=, equal, not equal.



The dialog box 'LOGIC / MATH / COMP.' is shown with the 'Instruction [2] result BOOL' dropdown set to 'AND'. The result register 'RES_BIT' is set to 21. The operands are configured as follows:

Operand	Value	Description
Operand 1	RES_BIT-1..20	Lighting groups 1..20
Operand 2	RES_BIT-21..36	Timers 1..16
Operand 3	RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
Operand 4	RES_BIT-40..246	User free

Below the operands, there are four dropdowns for 'Operand 1' through 'Operand 4', each with a '#' field next to it. The first three are set to 'EB/BI/EB' and the fourth to 'NULL'. The '#' fields are set to 1, 2, 3, and 1 respectively.

Red dashed arrows point from the 'AND' instruction in the dropdown to the 'Instruction TYPE selection' list on the right, and from the first operand dropdown to the 'Operand TYPE selection' list at the bottom right.

Instruction TYPE selection

NULL	
LIGHT_GROUP	Grupollum/LightingGrp/GroupeD'Eclairage
AND	R_BIT = (OP1 && OP2 && OP3 && OP4) BIT_TEST-SET_16
NAND	R_BIT = !(OP1 && OP2 && OP3 && OP4)
OR	R_BIT = (OP1 OP2 OP3 OP4)
NOR	R_BIT = !(OP1 OP2 OP3 OP4)
XOR	R_BIT = (OP1 xor OP2)
NXOR	R_BIT = !(OP1 xor OP2)
INVERT	R_BIT = !OP1
ADD	R_FLT = OP1 + OP2 + OP3 + OP4
SUBTRACT	R_FLT = OP1 - OP2 - OP3 - OP4
MULTIPLY	R_FLT = OP1 * OP2
DIVIDE	R_FLT = OP1 / OP2
COMP (>)	R_BIT = OP1 > OP2
COMP (>=)	R_BIT = OP1 >= OP2
COMP (<)	R_BIT = OP1 < OP2
COMP (<=)	R_BIT = OP1 <= OP2
COMP (==)	R_BIT = OP1 == OP2
COMP (!=)	R_BIT = OP1 != OP2
TIMER	Temporizador/Timer/Minuteur
JUMP	Binco/Jump/Saut/Springen/Saltare/Springen
OUTP_ASSIGN	Salida/Output/Sortie/Ausgang/Uschita/Uitgang
CONTROL_PROP	Contr. Prop./Proportional control/Commande proportionnelle
TOTALIZER	Totalizador/Totalizer/Totalizzatore/Totalizzatore
MINIMUM	R_FLT = min(OP1, OP2, OP3, OP4)
MAXIMUM	R_FLT = max(OP1, OP2, OP3, OP4)
AVERAGE	R_FLT = avg(OP1, OP2, OP3, OP4)
ALTERNATE	Altern+Simult/Alternate+Paralel/Alterné+Simultanée
HOURL_METER	Contador horas/Hour meter/Compteur d'heures/Betriebsstundenzähler
LABEL	Etiqueta/Label/Etiquette/etikett/Etiquette
CALL SUB	Llamar subrutina/Subroutine call/Appel de sous-programme
AHU_RTU	Etape AHU-UT/ AHU-RTU stage/Etape UTA-UST(HVAC)

Operand TYPE selection

NULL	
EA/AI/EA	EntAnl/Anlglnp/EntAnl. (1..40) <-[a]
EB/BI/EB	EntBin/Binlnp/EntBin. (1..40) <-[d]
SA/AO/SA	SalAnl/AnlOut/SorAnl. (1..10) [a]->
SB/BO/SB	SalBin/BinOut/SorBin. (1..60) [d]->
ADF	EE-Float 32 bit (1..100)
ADI	EE-Word 16 bit (1..100)
ADB	EE-Byte 8 bit (1..100)
RES-BIT	RAM-Result. bit (1..255)
RES-FLT	RAM-Result. float (1..40)
TIMER	Timer (1..16)
RMT	PunRem/RemPnt/PoiElo. (1..50)
K_BYTE+1	Val. K = (0..254) + 1

The first field is the **INSTRUCTION TYPE SELECTION** where the type of the instruction must be selected from the list of the dropdown control.

In the second field marked as **RESULT REGISTER** is the number of the result register RES_BIT where the result of the operation will be stored, must be between 1 and 255.

To the right of this field, a list of the usage of the RES_BITS 1 to 255 is shown for reference.

RES_BIT-1..20	Lighting groups 1..20
RES_BIT-21..36	Timers 1..16
RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
RES_BIT-40..246	User free
RES_BIT-247	Online as slave COM1
RES_BIT-248	Online as slave COM2
RES_BIT-249	Fixed timer 1 minute
RES_BIT-250	Local override PB1
RES_BIT-251..253	Power ON timer +1, +5, +10 seconds
RES_BIT-254	LED green
RES_BIT-255	LED red

It is recommended to use the free bits group reserved for the user: RES_BIT-40..246 unless the result of the instruction is going to command the lighting groups, in which case we should use RES_BIT-1..20 or if it is going to command the timers, use RES_BIT-21..36 instead. To command the LED of the operator interface use RES_BIT-254..255.

RES_BIT-37..39 are fixed system timers and are read only bits RES_BIT-247..253 are read only bits, and any intent to write to them will produce unexpected results.

In the **OPERAND 1 to 4** fields, the TYPE and INSTANCE # of the operands must be selected, not all instructions support 4 operands. Below is a list of the number of operands available for each of instruction types. Available operands will be enabled for any given operation, while the rest will be grayed out:

<u>Operands</u>	<u>Instruction type</u>
4	AND, NAND, OR, NOR, ADD, SUBTRACT, MIN, MAX, AVERAGE
2	XOR, NXOR, MULTIPLY, DIVIDE, <=, <, >, >=, ==, !=
1	INVERT,

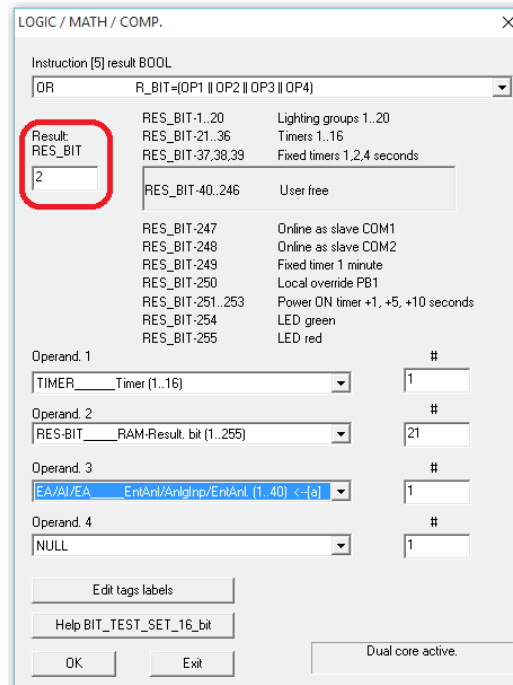
When an operand is not needed or is to be taken out of the equation, the **NULL** type must be selected. In this case the instance number it has is meaningless.

As mentioned before in section 7.1 PLC Ladder Programming Introduction, the instructions for the OpenBAS-HV-NX10P PLC are reduced instructions. Therefore if more than four operands are needed, multiple instructions need to be linked using the result registers.

For example, to add the value of eight analog inputs, two add instructions must be used, and then a third one adding the result register of the previous two instructions. NOTE: It is better to create these instructions sequentially to minimize processing time delays. This same logic applies to Boolean instructions too.

The following example shows the Boolean OR operation that if we write as a line is as follows:

- $RES_BIT_2 = TIMER_1 \text{ or } RES_BIT_21 \text{ or } ANALOG_INPUT_1$



Some notes on the above instruction worth mentioning are:

- For Binary operands, their binary values 0/1 are used
- Analog values are converted to ZERO value (0) if their value is less than (<) 0.5 and converted to a ONE value (1) if the analog value is greater or equal (>=) 0.5
- NULL variables are left out of the equation (not used)
- If a remote point is used as an operand, and the remote point is offline, its value will be set to ZERO and this value will be used in the equation.

This is the same OR ladder instruction shown above as documented by the PLC:



As can be seen in the Ladder diagram, the operand #4 (term D of the equation), which was declared as type NULL, has been taken out of the equation.

3.2 Hysteresis / In Range compare

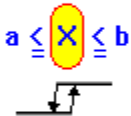
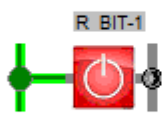
Compare instruction with high and low limits

The hysteresis and in range compare instructions allow you to compare the process variable PV to a set of low and high limits and set the result register according to the desired logic:

```

TYPE=32 [ HYST/InRANGE ]
RES_BIT:[ 1 ] = 0 [resBit:1100 valid:1110]
OP1:[ K_BYT-11 ] low limit
OP2:[ K_BYT-21 ] high limit
OP3:[ AI-01 ] PV
Flags:[ 06 ]
    
```

On line

Hysteresys mode

LOW	10.000	K_BYT-11
HIGH	20.000	K_BYT-21
PV	0.052	AI-1
Flags	0.000	NULL-7

3.2.1 HYSTERESIS MODE

In this mode of operation, the result register will be set to 1 when the PV is greater than (or greater or equal than optionally programmable) the high limit value. The result register will remain as 1 until the PV is less than (or less or equal than optionally programmable) the low limit value. At this moment the output will change to zero.

In the example above the result register will be set to one when the PV ≥ 20 and will remain at one level until the PV ≤ 10 .

This type of instruction is very common in HVAC applications, the instruction set screen and the ladder representation are shown on the next page.

Dialog to program a hysteresis comparator:

Hysteresis or in range comparator

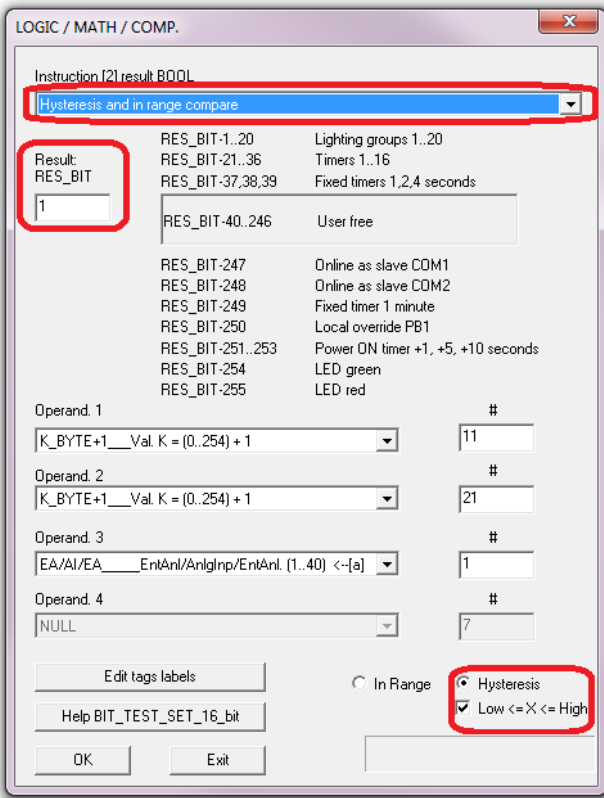
Result register

Low limit

High limit

Process variable

Flags to set <> OR <=>



Instruction (2) result B00L
 Hysteresis and in range compare

Result: RES_BIT
 1

RES_BIT-1..20 Lighting groups 1..20
 RES_BIT-21..36 Timers 1..16
 RES_BIT-37,38,39 Fixed timers 1,2,4 seconds
 RES_BIT-40..246 User free
 RES_BIT-247 Online as slave COM1
 RES_BIT-248 Online as slave COM2
 RES_BIT-249 Fixed timer 1 minute
 RES_BIT-250 Local override PB1
 RES_BIT-251..253 Power ON timer +1, +5, +10 seconds
 RES_BIT-254 LED green
 RES_BIT-255 LED red

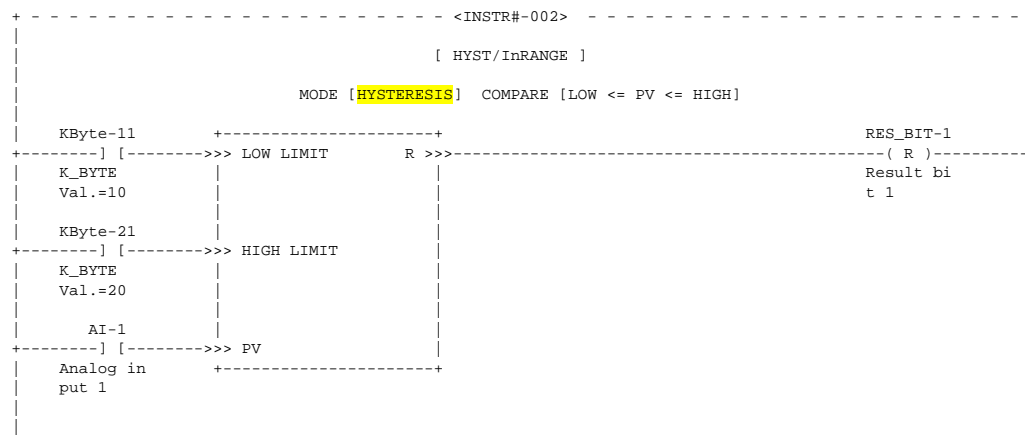
Operand 1: K_BYTE+1__Val. K = (0..254) + 1 # 11
 Operand 2: K_BYTE+1__Val. K = (0..254) + 1 # 21
 Operand 3: EA/AI/EA__EntAnl/AnlgInp/EntAnl. (1..40) <-[a] # 1
 Operand 4: NULL # 7

Edit tags labels
 Help BIT_TEST_SET_16_bit

OK Exit

In Range ☐ Hysteresis ☒
 Low <= X <= High ☒

Logic ladder representation of a Hysteresis instruction:



3.2.2 IN RANGE MODE

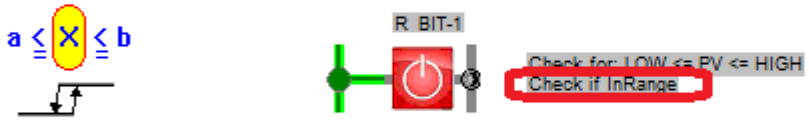
In this mode of operation, the result register will be set to 1 when the PV is inside the low and high limits, otherwise the result register will be set to 0.

The output can be inverted.

Also similar as in the hysteresis instruction, the greater / smaller and greater equal / smaller or equal can be programmed for this instruction.

TYPE=32 [HYST/InRange]
 RES_BIT:[1] = 0 [resBit:1100 valid:1110]
 OP1:[K_BYT-11] low limit
 OP2:[K_BYT-21] high limit
 OP3:[AI-01] PV
 Flags:[02]

On line



<i>LOW</i>	<i>10.000</i>	<i>K_BYT-11</i>
<i>HIGH</i>	<i>20.000</i>	<i>K_BYT-21</i>
<i>PV</i>	<i>0.052</i>	<i>AI-1</i>
<i>Flags</i>	<i>0.000</i>	<i>NULL-3</i>

This type of instruction is very common in HVAC applications, the instruction set screen and the ladder representation are shown on the next page.

Dialog to program an In Range comparator:

Hysteresis or In Range Comparator

Result register

Lower limit

Higher limit

Process variable

LOGIC / MATH / COMP.

Instruction [2] result BOOL

Hysteresis and in range compare

Result:	RES_BIT-1..20	Lighting groups 1..20
RES_BIT	RES_BIT-21..36	Timers 1..16
1	RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
	RES_BIT-40..246	User free
	RES_BIT-247	Online as slave COM1
	RES_BIT-248	Online as slave COM2
	RES_BIT-249	Fixed timer 1 minute
	RES_BIT-250	Local override PB1
	RES_BIT-251..253	Power ON timer +1, +5, +10 seconds
	RES_BIT-254	LED green
	RES_BIT-255	LED red

Operand 1: K_BYTE+1__Val. K = (0..254) + 1 11 #

Operand 2: K_BYTE+1__Val. K = (0..254) + 1 21 #

Operand 3: EA/AI/EA__EntAnl/AnlGlnp/EntAnl. (1..40) <-[a] 1 #

Operand 4: NULL 4 #

Edit tags labels
 Help BIT_TEST_SET_16_bit

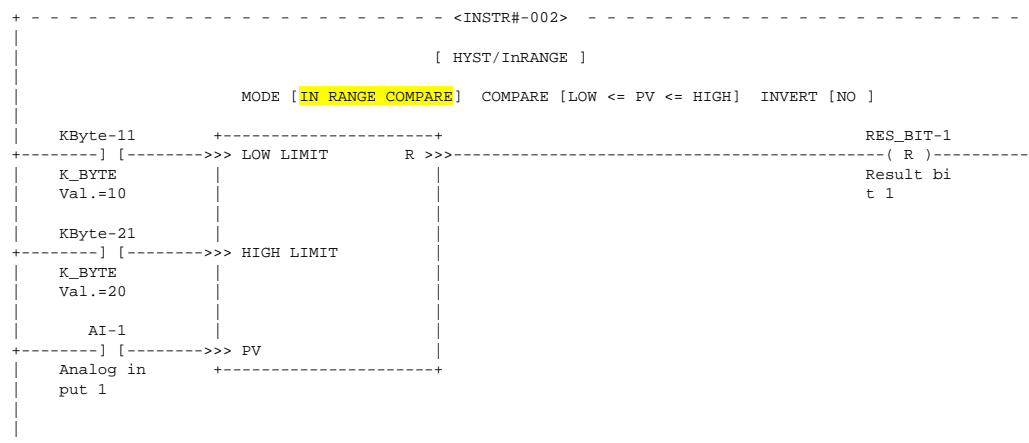
☒ In Range ☐ Hysteresis
☒ Invert ☒ Low <= X <= High

OK

Exit

Flags for output inversion
And <> or <=> comparison

Logic ladder representation of an In Range instruction:



3.3 Start stop instruction Easy start / stop with emergency stop

The start / stop instruction is commonly used as a latch to start and stop a motor. This instruction has an additional emergency stop signal in series with the logic, that if used can force a stop in all start /stop instructions on the system. A common use for this is a general emergency stop input for a group of motor starters.

If the emergency stop is not needed, the operand 1 can be replaced with a CONSTANT = 1 value.

The image of this instruction can be viewed on the following picture.



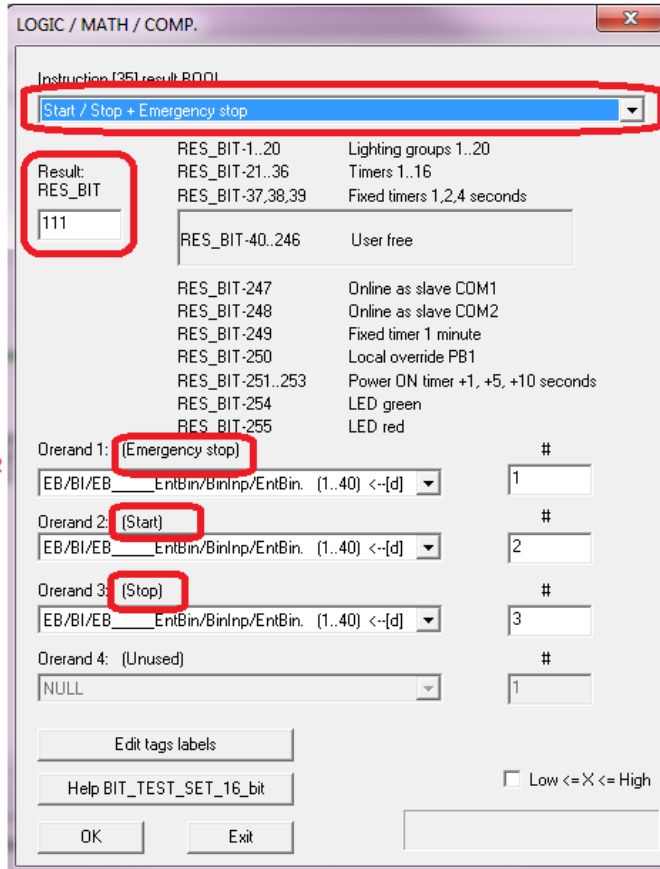
As can be seen is a typical representation of an electrical wired equivalent of a start stop circuit usually implemented using relay logic.

Dialog to program a start /stop instruction with emergency stop.

Start/Stop
instruction with
emergency stop

Result register

Set to K_BYTE-2
if unused



LOGIC / MATH / COMP.

Instruction [35] result B001

Start / Stop + Emergency stop

Result: RES_BIT

111

RES_BIT-1..20 Lighting groups 1..20

RES_BIT-21..36 Timers 1..16

RES_BIT-37,38,39 Fixed timers 1,2,4 seconds

RES_BIT-40..246 User free

RES_BIT-247 Online as slave COM1

RES_BIT-248 Online as slave COM2

RES_BIT-249 Fixed timer 1 minute

RES_BIT-250 Local override PB1

RES_BIT-251..253 Power ON timer +1, +5, +10 seconds

RES_BIT-254 LED green

RES_BIT-255 LED red

Orerand 1: (Emergency stop)

EB/BI/EB EntBin/BinInp/EntBin. (1..40) <-[d]

1

Orerand 2: (Start)

EB/BI/EB EntBin/BinInp/EntBin. (1..40) <-[d]

2

Orerand 3: (Stop)

EB/BI/EB EntBin/BinInp/EntBin. (1..40) <-[d]

3

Orerand 4: (Unused)

NULL

1

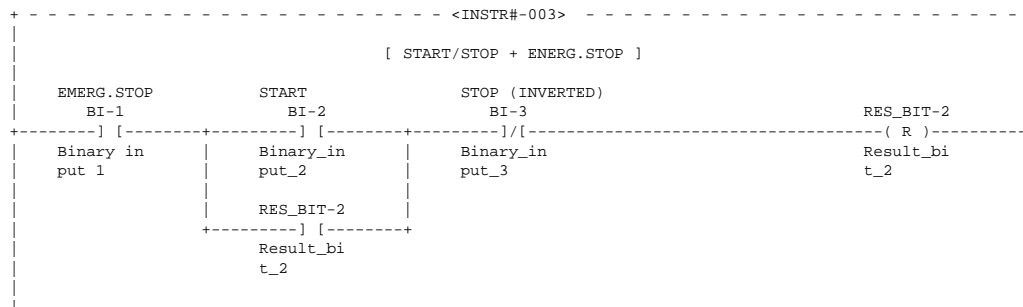
Edit tags labels

Help BIT_TEST_SET_16_bit

OK Exit

☐ Low <= X <= High

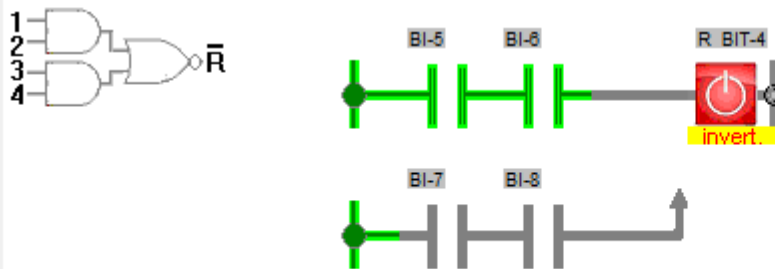
Logic ladder representation of a start /stop instruction with emergency stop.



3.4 Combined logic AND – OR Combined logic instruction

This instruction is a combination of an AND – OR instruction (with optional output inversion) that eases programming when combinatorial logic wants to be added in a single instruction.

```
TYPE=35[ AND/OR inverted ]
RES_BIT:[ 4 ] ~= 0 [resBit:1100 valid:1111]
OP1:[ BI-05 ] & OP2:[ BI-06 ]
||
OP3:[ BI-07 ] & OP4:[ BI-08 ]
```



On line

In this instruction a combinatorial OR (optionally NOR) of two AND instructions with two terms each is implemented. The following truth table illustrates the combinatorial states of this gate:

	Truth table for AND - OR instruction																																
Operand 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	AND	
Operand 2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1&2
Operand 3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	AND
Operand 4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	3&4

Result	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	OR of AND terms
Result inverted	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	NOR of AND terms

Dialog to program a AND – OR (optionally inverted) instruction.

AND - OR
AND - NOR (inverted)
Instructions

Result register

Operand 1

Operand 2

Operand 3

Operand 4

LOGIC / MATH / COMP.

Instruction (5) result: R001

[AND + OR (inverted) IR_BIT=(OP1 && OP2) || (OP3 && OP4)]

Result: RES_BIT

4

RES_BIT-1..20	Lighting groups 1..20
RES_BIT-21..36	Timers 1..16
RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
RES_BIT-40..246	User free
RES_BIT-247	Online as slave COM1
RES_BIT-248	Online as slave COM2
RES_BIT-249	Fixed timer 1 minute
RES_BIT-250	Local override PB1
RES_BIT-251..253	Power ON timer +1, +5, +10 seconds
RES_BIT-254	LED green
RES_BIT-255	LED red

Operand 1: [EB/BI/EB ___ EntBin/BinInp/EntBin. [1..40] <-[d]] **5** #

Operand 2: [EB/BI/EB ___ EntBin/BinInp/EntBin. [1..40] <-[d]] **6** #

Operand 3: [EB/BI/EB ___ EntBin/BinInp/EntBin. [1..40] <-[d]] **7** #

Operand 4: [EB/BI/EB ___ EntBin/BinInp/EntBin. [1..40] <-[d]] **8** #

Edit tags labels

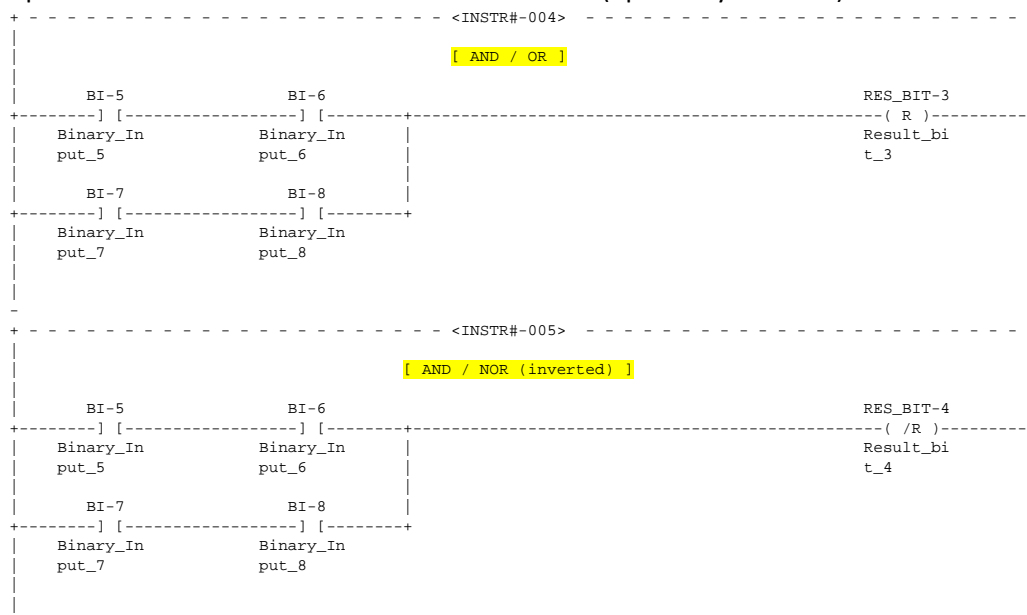
Help BIT_TEST_SET_16_bit

OK Exit

1st AND instruction

2nd AND instruction

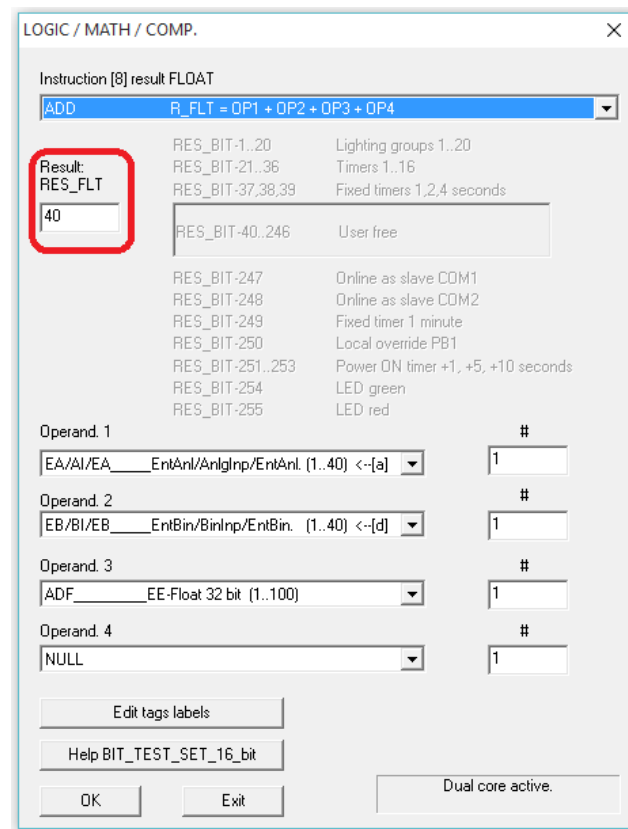
Logic ladder representation of a AND – OR and the NAD – NOR (optionally inverted) instruction are shown.



3.5 Math Instructions, ADD, SUBSTRACT, MULTIPLY, DIVIDE, MIN, MAX, AVERAGE

For math operations, the same dialog box used for the Boolean logic is used. The only difference is that instead of the RES-BIT result registers used in Boolean Logic that can only represent a 0 or 1 values, the RES_FLOAT floating types registers 1 to 40 are used instead.

This can be seen in the following picture, RES_FLOAT type is available, and the RES_BIT information to the is grayed out to highlight to the programmer that the RES_BIT registers are not the target for the result of this operation types.



Any type can be used as operands. In the example above, the result register 40 will contain the value of the addition of: Analog Input 1 + Binary Input 1 + ADF set point register 1 + NULL, the formula is depicted below:

$$\text{RES_FLT-40} = \text{AI_1} + \text{BI_1} + \text{ADF_1} + \text{NULL}$$

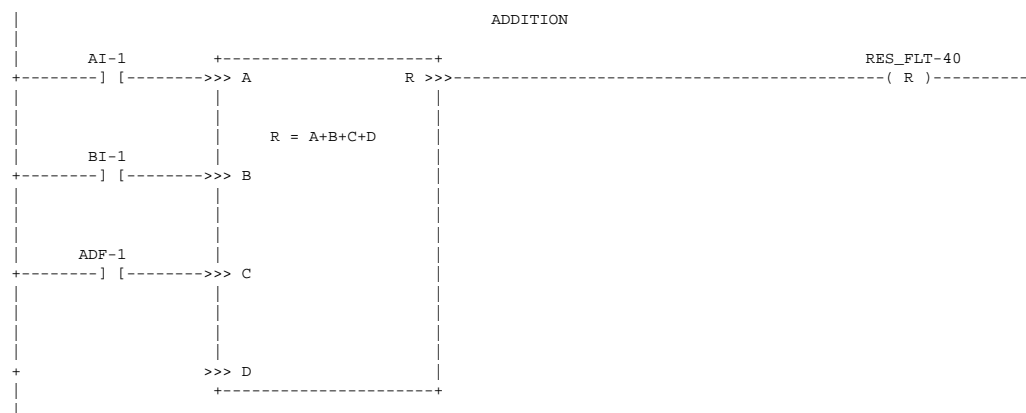
In the formula shown on the previous page **RES_FLT-40** = AI_1 + BI_1 + ADF_1 + NULL. Some notes are worth mentioning:

- Analog values are straightforward used
- For Binary operands, their binary values 0/1 are converted to analog values 0/1
- NULL variables are left out of the equation (not used).
- If a remote point is used as an operand and the remote point is offline, its value will be set to ZERO and this value will be used in the equation.
- Direct constants can be used as operands, for values between 0 and 254 by selecting the type: K_BYTE, and the instance number which can be any number between 1 and 255 will be the K value +1. For any other constant values outside this range, ADI's and ADF's database objects (or any register in RAM as well) must be used and their fixed value set in advance.

If a Dual Core is installed, the result registers RES_FLT 41 to 255 are also available to be used to store the result of the instruction.

However caution must be taken that the selected register is not also set up as a remote point and used as result of a logic instruction simultaneously, as unexpected results will happen, because the logic instruction will write the register with the result of the equation. When the remote point is scanned the same register will be overwritten with the remote point's value, there is no way to know which value is the correct at any given time, as both the instruction processing and the remote point scanning are asynchronous in nature.

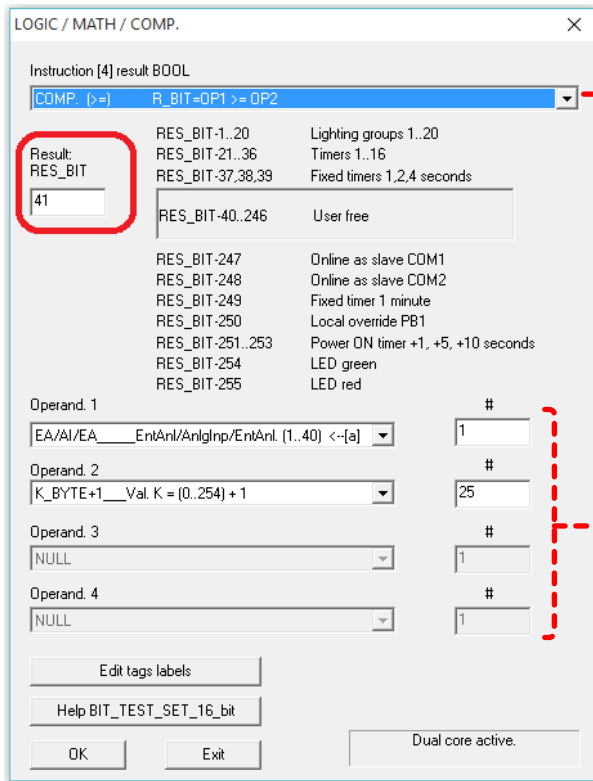
This is the same ADD Ladder instruction shown on the previous page as documented by the PLC:



As can be seen in the Ladder diagram, the operand #4 (term D of the equation), which was declared as type NULL, has been taken out of the equation.

3.6 Compare instructions, >, >=, <, <=, == (Equal), != (Not Equal)

For compare operations, the same dialog box used for the Boolean logic is used. The only difference is that only operands 1 and 2 are active.



Instruction TYPE selection

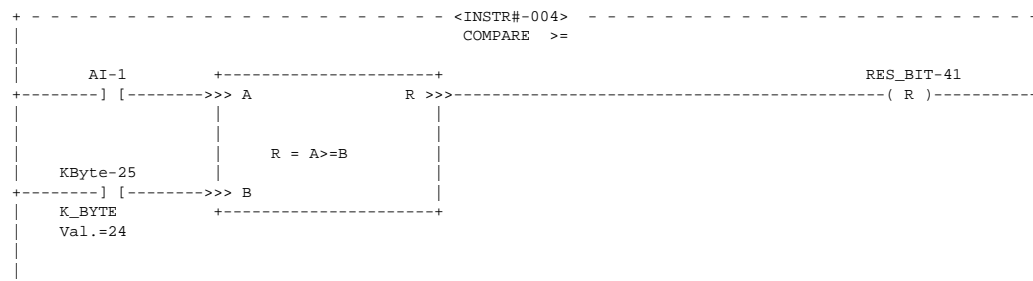
NULL	
LIGHT_GROUP	Gruppolum/LightingGrp/GroupeD'Eclairage
AND	R_BIT = (OP1 && OP2 && OP3 && OP4) BIT_TEST-SET_16
NAND	R_BIT = ((OP1 && OP2 && OP3 && OP4))
OR	R_BIT = (OP1 OP2 OP3 OP4)
NOR	R_BIT = ((OP1 OP2 OP3 OP4))
XOR	R_BIT = (OP1 xor OP2)
NXOR	R_BIT = ((OP1 xor OP2))
INVERT	R_BIT = (!OP1)
ADD	R_FLT = OP1 + OP2 + OP3 + OP4
SUBSTRACT	R_FLT = OP1 - OP2 - OP3 - OP4
MULTIPLY	R_FLT = OP1 * OP2
DIVIDE	R_FLT = OP1 / OP2
COMP. (>)	R_BIT = OP1 > OP2
COMP. (>=)	R_BIT = OP1 >= OP2
COMP. (<)	R_BIT = OP1 < OP2
COMP. (<=)	R_BIT = OP1 <= OP2
COMP. (==)	R_BIT = OP1 == OP2
COMP. (!=)	R_BIT = OP1 != OP2
TIMER	Temporizador/Timer/Minuteur
JUMP	Brinco/Jump/Saut/Springen/Saltare/Springen
OUTP_ASSIGN	Salida/Output/Sortie/Ausgang/Uschita/Uitgang
CONTROL_PROP	Contr.Prop./Proportional control/Commande proportionnelle
TOTALIZER	Totalizador/Totalizer/Totalisateur/Totalizzatore
MINIMUM	R_FLT = min(OP1, OP2, OP3, OP4)
MAXIMUM	R_FLT = max(OP1, OP2, OP3, OP4)
AVERAGE	R_FLT = avg(OP1, OP2, OP3, OP4)
ALTERNATE	Altern+Simult/Alternate+Parallel/Alterné+Simultanée
HOURL_METER	Contador horas/Hour meter/Compteur d'heures/Betriebsstundenzähler
LABEL	Etiqueta/Label/Étiquette/Etiket/Étiquette
CALL SUB	Llamar subrutina/Subroutine call/Appel de sous-programme
AHU_RTU	EtapA AHU-UT / AHU-RTU stage/Étape UTA-UST(HVAC)

Operand TYPE selection

NULL	
EA/AI/EA__	EntAnl/AnlgInp/EntAnl (1..40) <-[a]
EB/BI/EB__	EntBin/BinInp/EntBin (1..40) <-[d]
SA/AD/SA__	SalAnl/AnlOut/SorAnl (1..10) [a]->
SB/BD/SB__	SalBin/Bin/Out/SorBin (1..60) [d]->
ADF	EE-Float 32 bit (1..100)
ADI	EE-Word 16 bit (1..100)
ADB	EE-Byte 8 bit (1..100)
RES-BIT	RAM-Result. bit (1..255)
RES-FLT	RAM-Result. float (1..40)
TIMER	Timer (1..16)
RMT	PunRem/RemPnt/PoiElo. (1..50)
K_BYTE+1__	Val. K = (0.254) + 1

The first field is the **INSTRUCTION TYPE SELECTION** where the type of the instruction must be selected from the list of the dropdown control.

In the second field marked as **RESULT REGISTER** is the number of the result register RES_BIT where the result of the operation will be stored, must be between 1 and 255.



To the right of this field, a list of the usage of the RES_BITS 1 to 255 is shown for reference.

RES_BIT-1..20	Lighting groups 1..20
RES_BIT-21..36	Timers 1..16
RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
RES_BIT-40..246	User free
RES_BIT-247	Online as slave COM1
RES_BIT-248	Online as slave COM2
RES_BIT-249	Fixed timer 1 minute
RES_BIT-250	Local override PB1
RES_BIT-251..253	Power ON timer +1, +5, +10 seconds
RES_BIT-254	LED green
RES_BIT-255	LED red

It is recommended to use the free bits group reserved for the user: RES_BIT-40..246 unless the result of the instruction is going to command the lighting groups, in which case we should use RES_BIT-1..20 or if it is going to command the timers, use RES_BIT-21..36 instead, or to command the LED of the operator interface use RES_BIT-254..255.

RES_BIT-37..39 are fixed system timers and are read only bits, also RES_BIT-247..253 are read only bits, and any intent to write to them will produce unexpected results.

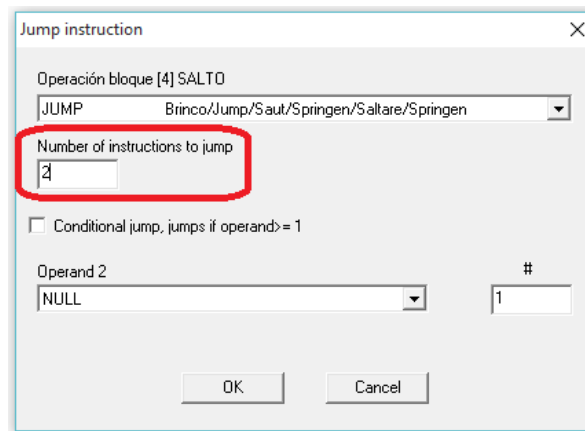
In the **OPERAND 1 to 2** fields, the TYPE and INSTANCE # of the operands must be selected and both operands must be declared with a valid TYPE. Some notes are worth mentioning:

- Analog values are straightforward used
- For Binary operands, their binary values 0/1 are converted to analog values 0/1
- NULL variables are left out of the equation (not used).
- If a remote point is used as an operand, and the remote point is offline, its value will be set to ZERO and this value will be used in the equation.
- Direct constants can be used as operands, for values between 0 and 254 by selecting the type: K_BYTE, and the instance number which can be any number between 1 and 255 will be the K value +1. For any other constant values outside this range, ADI's and ADF's database objects (or any register in RAM as well) must be used and their fixed value set in advance.

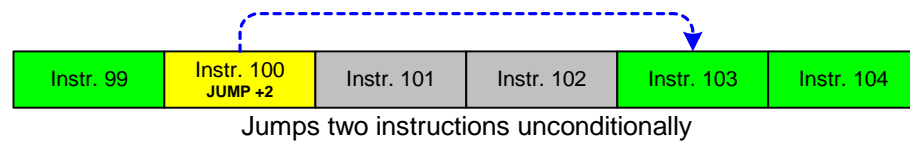
If the result of the comparison is **TRUE** the result bit will be set to **ONE**, and conversely if the result of the comparison is **FALSE** the result bit will be set to **ZERO**.

3.7 Jump Instructions, Conditional and Unconditional Program Branching

The jump instruction, allows to jump (skip or not execute) a number of blocks, which will not be processed by the **PLC**. This function is useful to disable a block of instructions, or execute part of the program conditionally according to existing conditions.

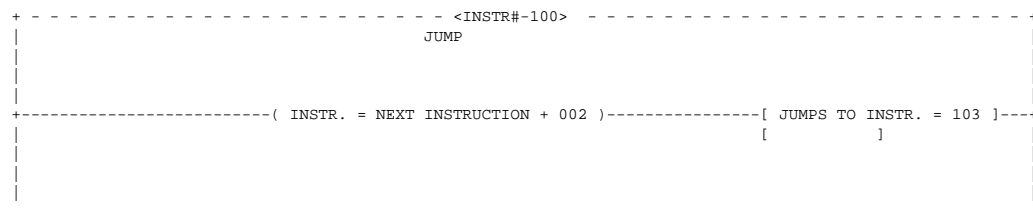


For example, if you program in the block number 100 a jump of two blocks, the blocks 101 and 102 won't execute (will be jumped or skipped).

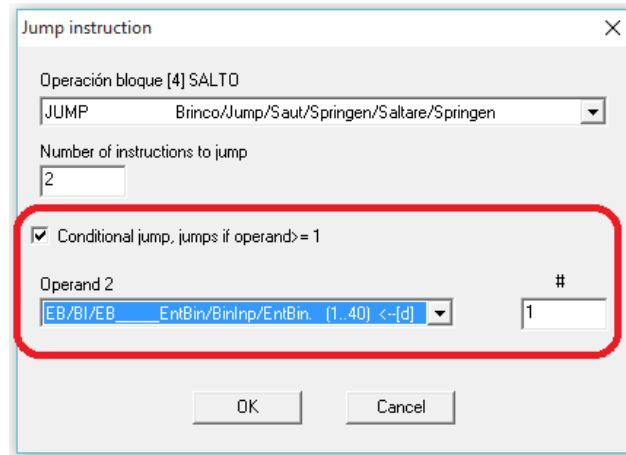


It is recommended to organize the program so that the blocks that are for common functions will be grouped so they can be jumped (skipped or not executed) as needed while debugging the program.

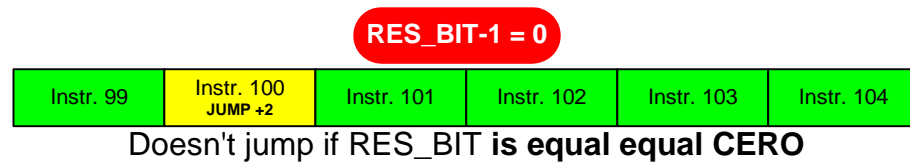
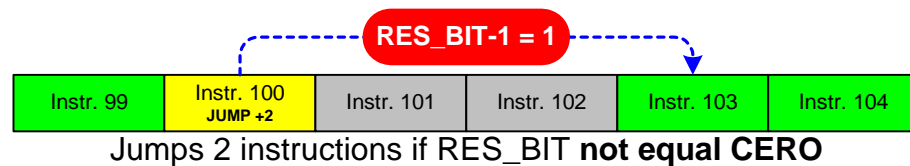
This is an example Ladder documentation of the above instruction by the PLC:



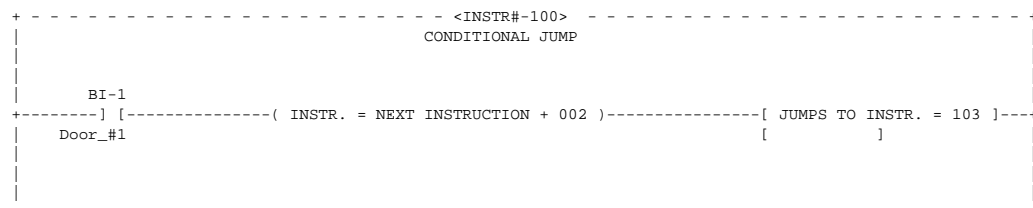
If the CONDITIONAL jump option is enabled, the PLC will jump (skip or not execute) the number of selected blocks when the value of the OPERAND 2 is \geq ONE.



In the above example two instructions will be jumped or skipped if Binary Output 1 is ONE (1).



This is an example Ladder documentation of the above instruction by the PLC. Note the Operand in the equation can be any type Analog or digital:



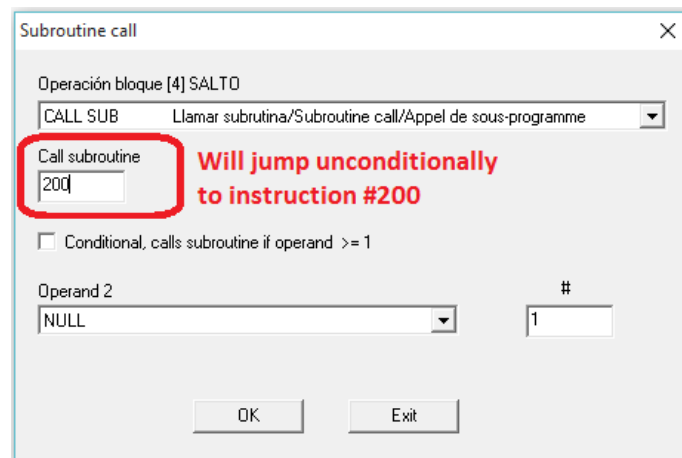
NOTE:

Analog values are evaluated and converted to ZERO value (0) if their value is less than ($<$) 0.5 and converted to a ONE value (1) if the analog value is greater or equal (\geq) 0.5

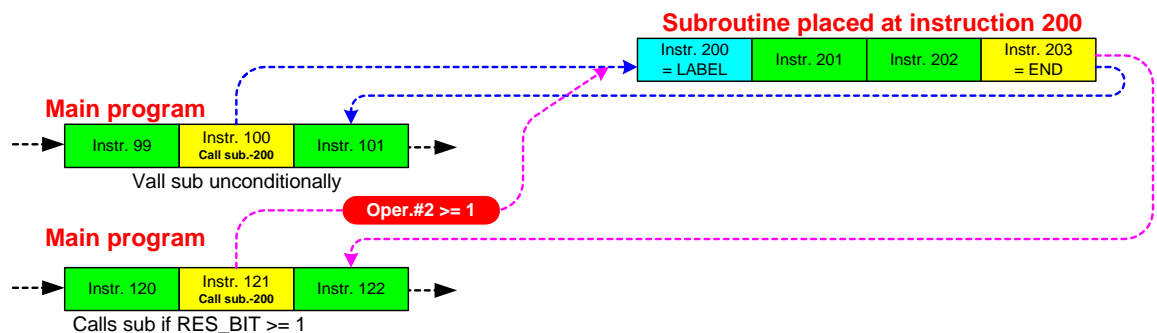
3.8 Subroutine Calls, Conditional and Unconditional Subroutine Calling

The use of the “Call subroutine” instruction, allows the creation of part of the program with common functionality as a subroutine, and then calling it from other sections of the program to save space and reuse code.

This allows the creation of a modular program. These subroutines can be further exported to and imported from disk for code reuse. See section 7.11 Library Import and Export.

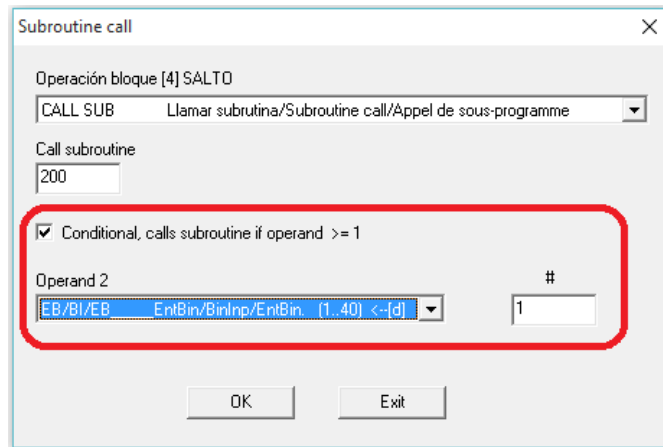


If we write a CALL SUBROUTINE instruction, when the program executes it, will jump to the instruction number pointed by the CALL_SUB instruction and continue running the program inside of the subroutine until an END instruction is found. In that moment the program will return and continue executing the instruction after the CALL_SUB instruction that triggered the jump into the subroutine, as depicted on the following picture.



In the example at the end of last page, when in the main program the CALL_SUB UNCONDITIONALLY in instruction 100 is executed, the program will jump from instruction 100 to instruction 200. Once inside the subroutine when the END instruction is found in instruction 203 and the program will return to main program executing instruction 101.

Sometime later in the main program another CALL_SUB CONDITIONALLY in instruction 121 is found. This time it is a conditional call and will only be executed if the Operand 2 evaluates to be a ONE, otherwise the program just continues with instruction 122.

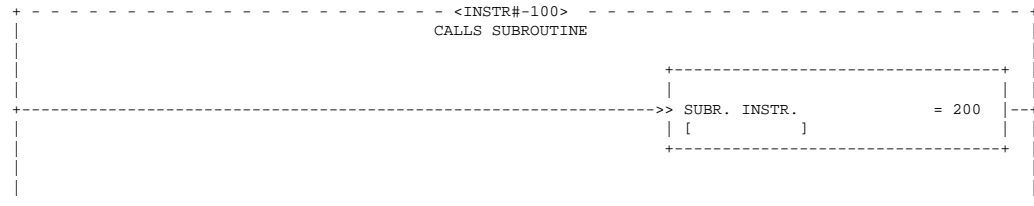


If the CALL_SUB CONDITIONALLY executed correctly because it's call operand was ≥ 1 , then the program will jump from instruction 120 to instruction 200. Once inside the subroutine when the END instruction is found in instruction 203 of the subroutine, the program will return to main program executing instruction 122.

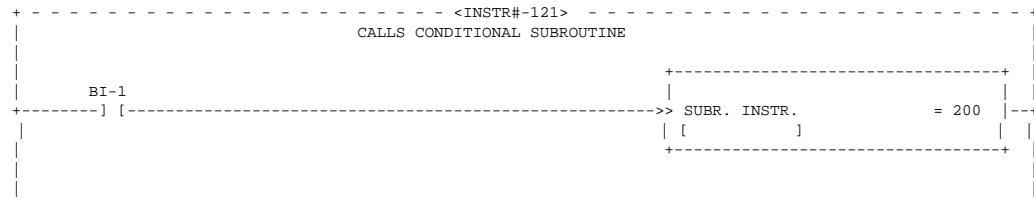
NOTE, A subroutine can't call another subroutine.

It is recommended that the first block of a subroutine always be a LABEL, so when generating the documentation, this name appears in the subroutine call and it will be easier to read the program.

The representation of an “unconditional CALL_SUB” instruction in the Ladder diagram is as follows:



The representation of a “conditional CALL_SUB” instruction in the Ladder diagram is as follows:
 Note the Operand in the equation; it can be any type Analog or digital:



NOTE:

Analog values are evaluated and converted to ZERO value (0) if their value is less than (<) 0.5 and converted to a ONE value (1) if the analog value is greater or equal (>=) 0.5

3.9 Totalizer, Total Accumulator, Period Totalization, Energy Totalization

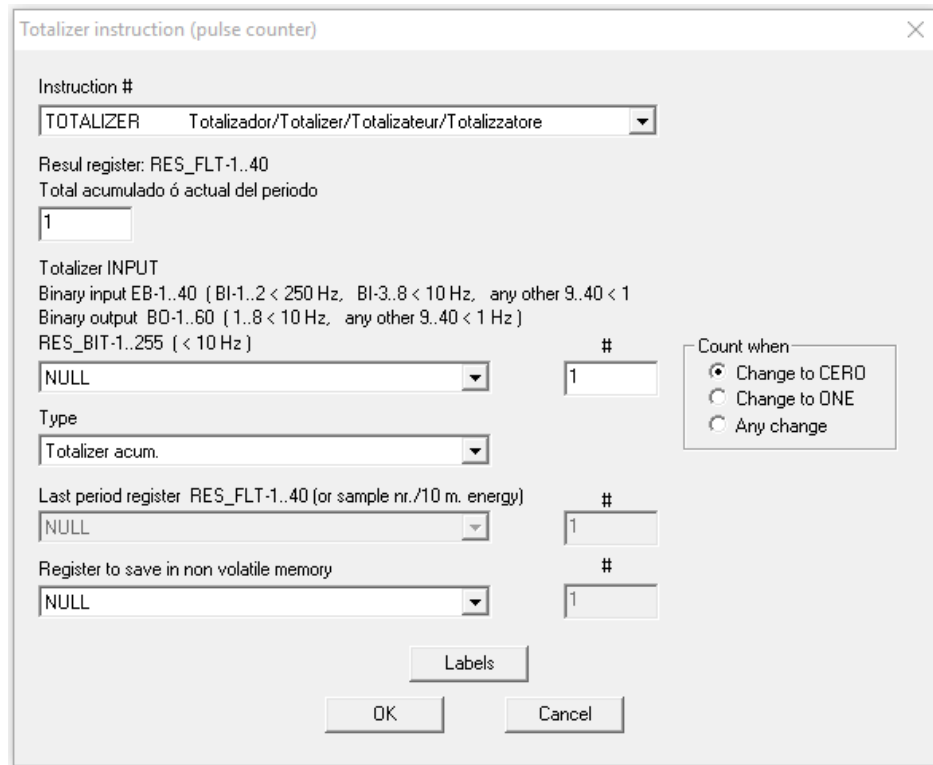
When we want to totalize or accumulate the count of some event, for example:

- A pulse sensor to measure water flow, gas consumption, energy, etc.
- Number of starts or stops of a motor, or how many times a door has been opened.
- The units per period of time like: Liters/hr., cubic ft. / min, Kilo Watts /hour
- Or the totalized units of any value we can use the totalizer instruction.

The input to the totalizer input can be any digital type:

- Digital inputs
- Digital outputs
- Bit registers

To configure a totalizer instruction:



Totalizer instruction (pulse counter)

Instruction #
 TOTALIZER Totalizador/Totalizer/Totalisateur/Totalizzatore

Result register: RES_FLT-1..40
 Total acumulado ó actual del periodo
 1

Totalizer INPUT
 Binary input EB-1..40 (BI-1..2 < 250 Hz, BI-3..8 < 10 Hz, any other 9..40 < 1
 Binary output BO-1..60 (1..8 < 10 Hz, any other 9..40 < 1 Hz)
 RES_BIT-1..255 (< 10 Hz)

NULL # 1

Type
 Totalizer acum.

Last period register RES_FLT-1..40 (or sample nr./10 m. energy)
 NULL # 1

Register to save in non volatile memory
 NULL # 1

Count when
☒ Change to ZERO
☐ Change to ONE
☐ Any change

Labels

OK Cancel

- Select the TOTALIZER in INSTRUCTION field.
- Then select in the RESULT_REGISTER the RES_FLT that will be used to store the accumulated value.
- Choose the TOTALIZER INPUT. This can be a Binary Input, Binary Output or RES_BIT register. Binary Inputs 1 and 2 accept signals up to 250 HZ to obtain a resolution of up to 15,000 pulses per minute.
- In the type field, select if it is going to be a: TOTALIZER ACCUMULATOR or PERIOD TOTALIZER or a KILOWATT HOUR TOTALIZER. If TOTALIZER ACCUMULATOR is selected, the register RES_FLT selected will increase each time that the variable changes state, and the only way of resetting it to ZERO is by doing it manually. See section 6.15.5.
- If PERIOD TOTALIZER is selected, the field that will store the previous period register will also be enabled, here select register RES_FLT-1..40. In this case the first register will keep the count of the actual period. For example if we are measuring water on liters per minute, it will contain the count of the actual minute, and the second register will have the count of the previous minute.

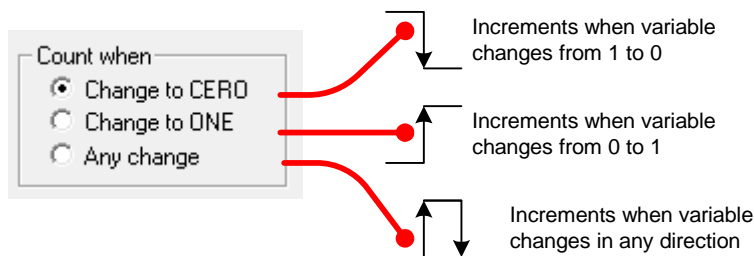
- The periods that it can handle are: 1 minute, 5 minutes, 10 minutes, 15 minutes, 20 minutes, 30 minutes, 60 minutes (1 hour).
- At the end select the nonvolatile memory register that will keep the value stored in case the RAM battery backed up value stored in the RES_FLT registers is lost.

The Binary Inputs 1 and 2 are available for high speed counters, the maximum frequency is 250 Hz (15,000 pulses per minute).

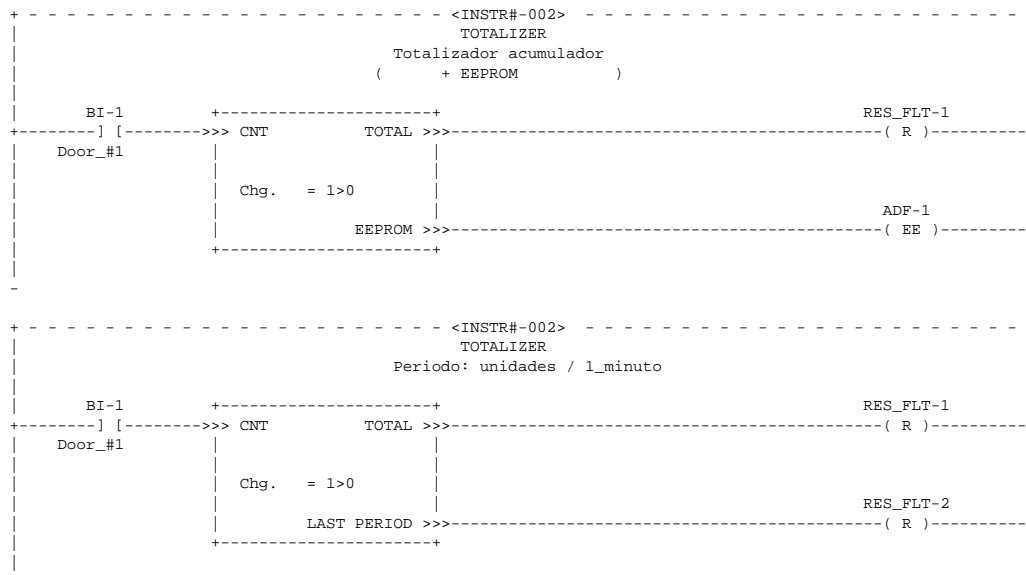
Binary Inputs 3 to 8, and Binary Outputs 1 to 8 and bit type registers RES_BIT-1..255 support frequencies up to 10 Hz (600 pulses per minute).

And finally the Binary Inputs 9 to 400 and the Binary Outputs 9 to 60 have a maximum frequency of 1 Hz or (60 pulses per minute), and are highly dependent on the communications speed so must be avoided unless low speed counting such as start / stop of motors is to be totalized.

The option of **COUNT AT** allows the selection of when the totalizer will increase:



This is a ladder representation of a TOTALIZER ACCUMULATOR and a PERIOD TOTALIZER:



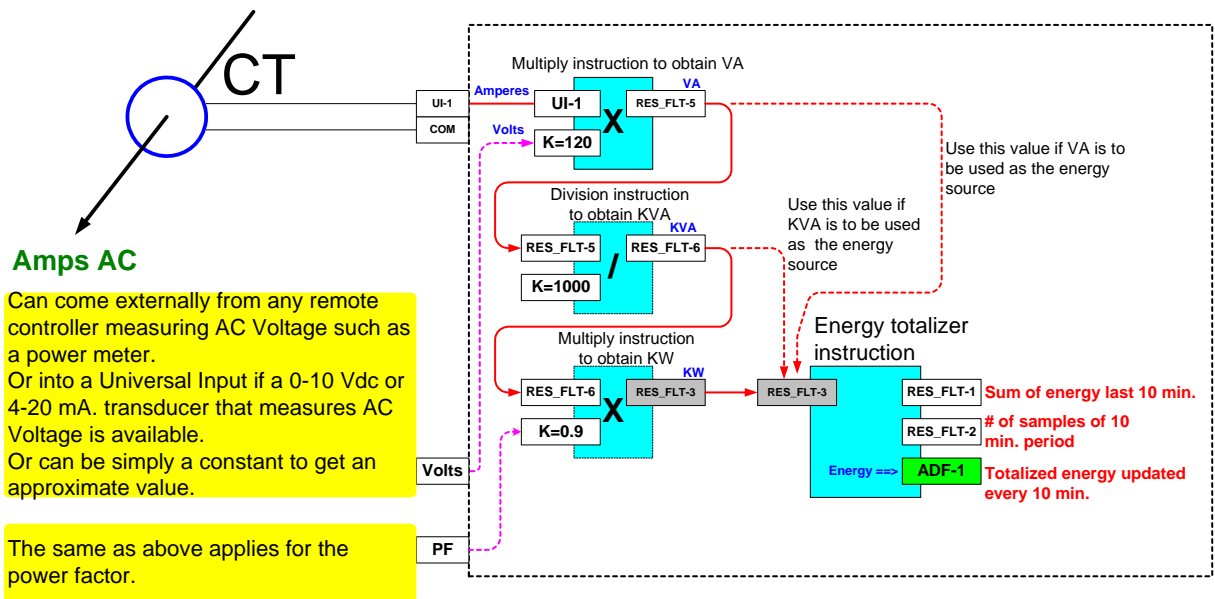
If energy information is available as a pulse coming from a power meter, simply follow the instructions on the previous page, but if energy will be calculated from an analog source such as an Ampere or instantaneous Kilowatt meter, the totalizer block has the capability of calculating energy and store it as Kilo Watt hours directly.

Following the diagram below, using a current signal coming from an analog current transformer (CT) that can be input directly to controllers accepting AC current transformers such as the OpenBAS-HV-SF or OpenBAS-HV-CUR, or if a current transducer with a 0-10 Volts DC or 4-20 mA. output is available it can be input directly to a Universal Input of an OpenBAS-HV-NX10P controller.

OpenBAS-HV-SF or OpenBAS-HV-USB

using CT's with a 1000:1

Or an **OpenBAS-HV-NX10P** using a 0-10 Vdc or 4-20 mA. current transducer.



If only VA will be used to get VA/h energy just one multiply instruction is needed to multiply Volts x Amperes to obtain VAs.

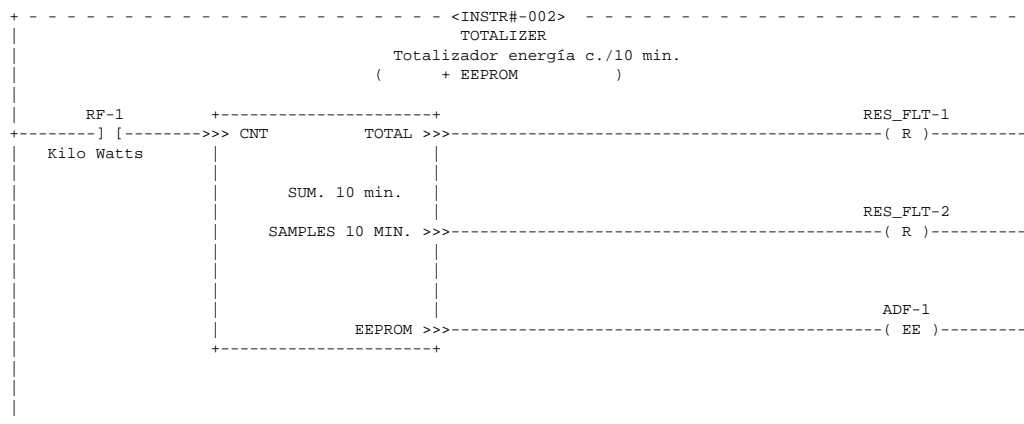
If the measurement is to be in KVA/h, then a second division instruction is needed to convert the previous VA into a KVA reading by dividing VA / 1000.

If the measurement is to be in KW/h, then a third multiplier instruction is needed to multiply KVA x the power factor (PF) and obtain KW.

As explained in the last picture, the Volts reading and the Power factor reading can come from real value transducers or remotely from a high AC voltage measuring device.

But if just an approximate value is enough to keep track of an energy trend, or for comparison purposes between different loads, then a constant value for both the Voltage and Power factor can be used and will give good results with a minimum of material and labor expense.

This is a ladder representation of a KILOWATT HOUR TOTALIZER:



The inputs to the module is the VA, KVA or KW reading from the previous multiply and divide instructions as described previously in this section. The first and second output registers of the instruction are the 10-minute sum of the energy and the number of samples per that 10-minute period.

After the 10-minute period finishes, the energy is calculated by dividing the energy sum of the 10-minute period between the number of samples of that same time period, to obtain a real KW/h real value.

Finally, this new KW/h value is scaled to a 10-minute KW/h value and added to the previous KW/h value which is stored in EEPROM as the third output register of the ENERGY TOTALIZER instruction.

3.10 Lighting Groups, Creating a Lighting Group

One of the uses of the OpenBAS-HV-NX10P is as a lighting controller. For this kind of application it is useful to group the physical Relays (Binary Outputs) in groups. With the LIGHTING GROUPS function it is possible to group up to 8 Relays in a single group. For this type of lighting groups the bit registers RES_BIT-1..20 or the virtual relays BO-41..60 are available as the source of control for the lighting group.

There is no limit as of how many lighting groups can be created except for the number of instructions of the PLC which is 400 for the standard controller and 1200 for a Dual Core device.

Even while the primary function of this lighting group instruction is intended for lighting, it can be used anytime that a group of outputs need to be grouped together so that a single control input will manage up to eight outputs.

Even when unlimited number of groups can be created, only limited by the available free instructions in the PLC, only 20 control bits and 20 virtual relays are available as the source of control for the groups.

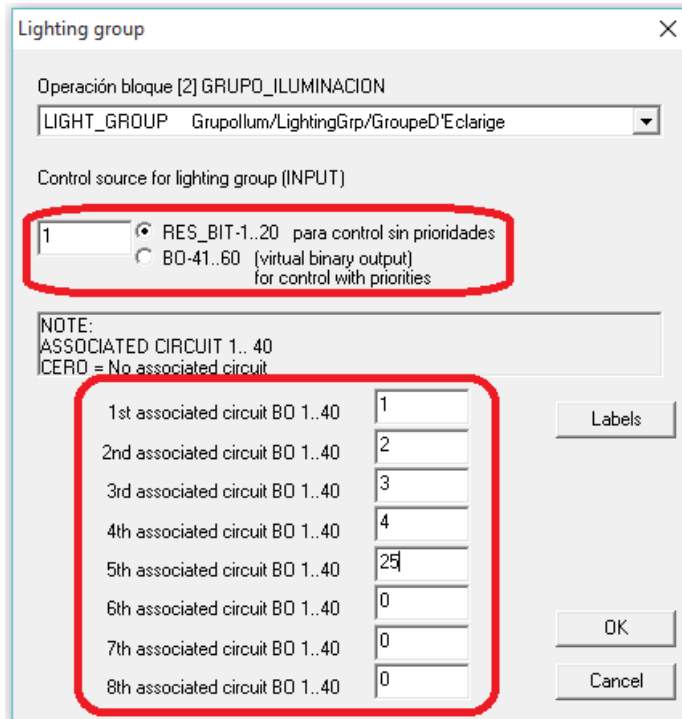
With this configuration there can be up to 40 different lighting groups, and multiple groups can use the same source of control to be linked as a bigger group. A single group can only manage eight outputs but multiple groups sharing a common source of control allow the creation of unlimited size groups.

The main difference between using the control bits RES_BIT-1..20 or the virtual relays BO-41..60 is that the bits only can have the status 0/1 (Off/On), while the virtual relays have the same priority level that the physical relays which is from maximum to minimum priority:

- LOCAL OVERRIDE Override issued from the LCD operator display.
- OVERRIDE COMM Remote override from USB, COM1, COM2 or Ethernet.
- LOGIC_XX Command to ON/OFF from some logic instruction of PLC.
- SCHEDULE / COMM / OPERATE Have the same internal priority.

The associated circuit can be any physical relay BO-1..40 or ZERO (same as NULL) to remove the assignment of some previously assigned relay.

Here a group was created having RES_BIT - 1 as the control source and five relays have been added to the group, BO-1, BO-2, BO-3, BO-4 and BO-25.



Lighting group

Operación bloque [2] GRUPO_ILUMINACION

LIGHT_GROUP Grupollum/LightingGrp/GroupeD'Eclairage

Control source for lighting group (INPUT)

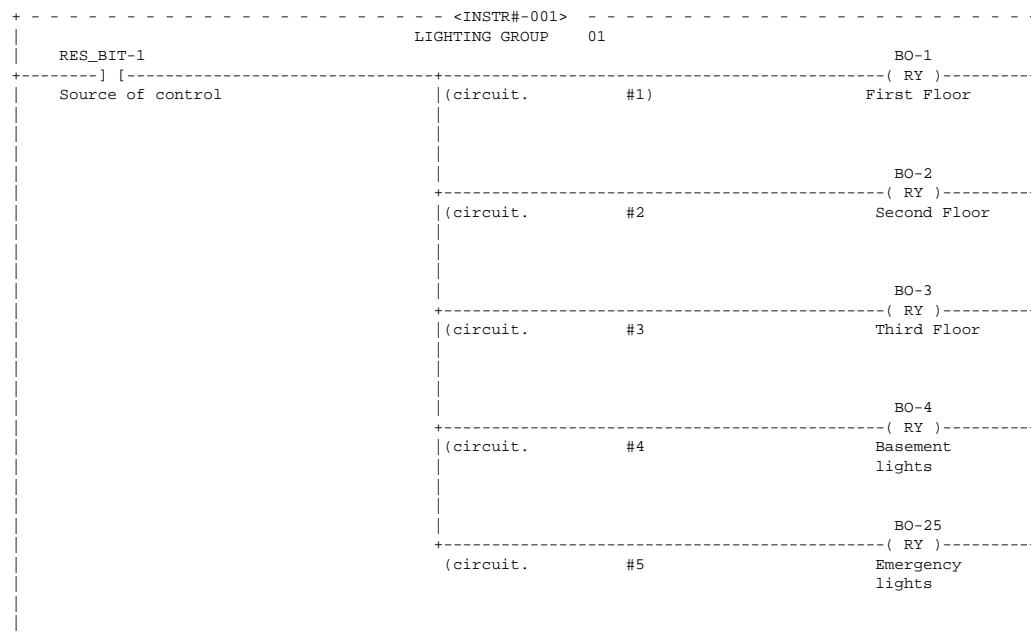
1 RES_BIT-1..20 para control sin prioridades
 BO-41..60 (virtual binary output) for control with priorities

NOTE:
 ASSOCIATED CIRCUIT 1.. 40
 ICERO = No associated circuit

1st associated circuit BO 1..40	1
2nd associated circuit BO 1..40	2
3rd associated circuit BO 1..40	3
4th associated circuit BO 1..40	4
5th associated circuit BO 1..40	25
6th associated circuit BO 1..40	0
7th associated circuit BO 1..40	0
8th associated circuit BO 1..40	0


Labels OK Cancel

The following is a ladder representation of the lighting group instruction programmed above, with one source of control and five associated relays (or Binary Outputs):



Once the group is created, we can see the source of who is controlling the lighting group. If we selected a number between 1 to 20, then the corresponding RES-BIT-1..20 register will be the source of control.

Logical blocks PLC LOGIC-1..400

Block # 1..400 ☐ Expanded remote points active ☒ Core 2 active  ☐ Auto

2 |<< << >> 1->>| >>| PLC current = [1] PLC

TYPE-1 [LIGHTING_GROUP_01]


Control = RES_BIT:[1] = 0

CIRCUITS [01] [00] [00] [00] [00] [00] [00] [00] [00] [00]

If instead the number selected was between 41 to 60, then the source of control of the lighting control group will be the virtual relays BO-41..60, which have the same priority level as the physical relays BO-1..40, and can be commanded manually from the LCD operator display (LOCAL OVERRIDE) or by communications port (REMOTE OVERRIDE) or the internal PLC logic or the schedules, etc..

Logical blocks PLC LOGIC-1..400

Block # 1..400

☐ Expanded remote points active ☒ Core 2 active  ☐ Auto

2 |<< << >> 1->>| >>| PLC current = [1] PLC Read On Line Modify

TYPE-1 [LIGHTING GROUP 01]

Control = SB:[41] = 0

CIRCUITS: [01] [02] [03] [04] [05] [06] [07] [08] [09] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100]

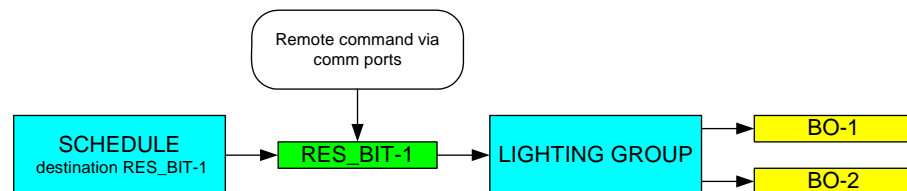
IMPORTANT NOTE

If a Binary Output or Lighting Group is being commanded through an ASSIGN OUTPUT instruction. This instruction will have a higher priority than the schedules.

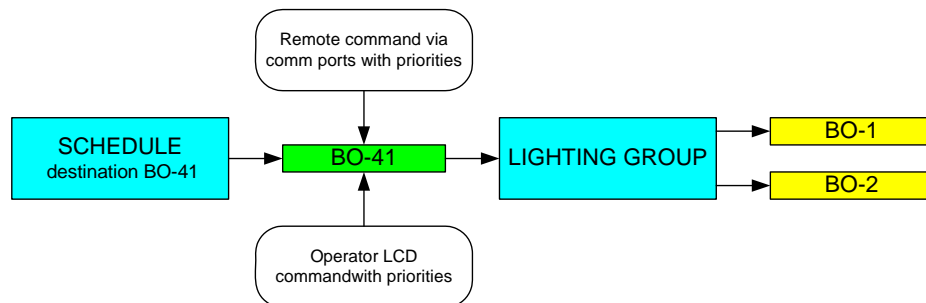
So then, for the schedules to operate correctly it is recommended to use the Lighting Schedules to control bit registers RES_BIT.1..20 or a virtual relay output BO-41..60, and then use some control logic employing Boolean instructions so that this logic is what drives the Lighting Group doing exactly what the end user wants with the correct priorities.

View some examples of lighting group combinations on the next page.

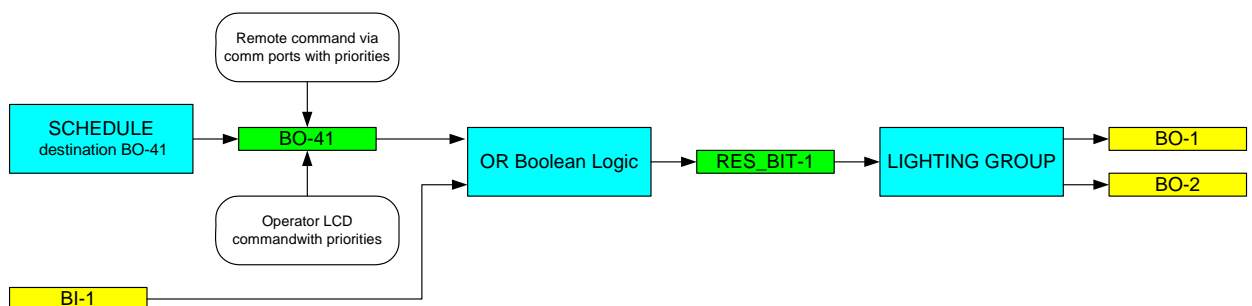
EXAMPLE-1: The relays BO-1 and 2 that are in a Lighting Group are turned ON/OFF based on a Schedule that is controlling the bit of register RES_BIT-1. No prioritization is available.



EXAMPLE-2: The relays BO-1 and 2 that are in a Lighting Group, are turned ON/OFF based on a Schedule that is controlling the virtual Binary Output 41 register BO-41. In this case the BO-41 can additionally be commanded by the LCD operator display, or remotely command via the communication ports, using the same priorities that the standard Binary Outputs 1-40 have.

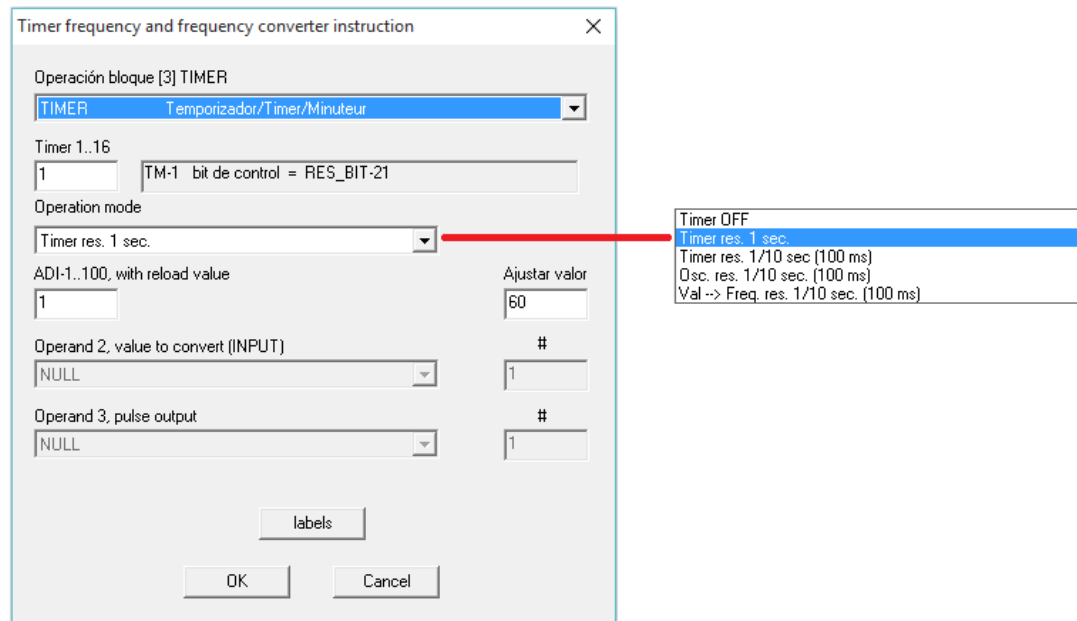


EXAMPLE-3: The relays BO-1 and 2 that are in a Lighting Group, are turned ON/OFF based on a Schedule that is controlling the virtual Binary Output 41 register BO-41. In this case the BO-41 can additionally be commanded by the LCD operator display, or remotely commanded via the communication ports, using the same priorities that the standard Binary Outputs 1-40 have. However outside of the ON schedule that controls BO-41, the relays can be turned on when a presence sensor that is connected to the binary input BI-1 is active, using a “OR” instruction that stores the result of the operation in result bit RES_BIT-1, which in turn is used as the source of control for the LIGHTING GROUP.



3.11 Timer Instructions, Timer, Free run Oscillator, Value to Frequency Converter

The timer instruction allows with the use of up to 16 system timers to program down counters, fixed or variable frequency oscillators.



The above figure shows the configuration of a typical timer. In the first selection box, TIMER must be selected.

In the second one of the sixteen timers available must be selected.

In the third field the OPERATION MODE must be set, the OpenBAS-HV-NX10P supports five different operation modes that are listed below:

- Timer OFF
- Down counting timer with a 1 second resolution
- Down counting timer with a 1/10 second resolution (100 milliseconds)
- Fixed time oscillator with a 1/10 second resolution (100 milliseconds)
- Value to frequency with a 1/10 second resolution (100 milliseconds)

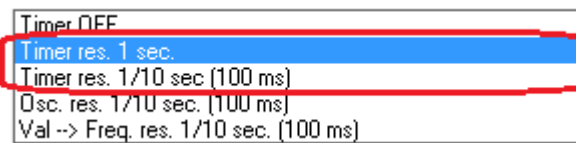
The fourth field is an index to select one of the 100 available ADI registers that will be used as the reload value. Fields five and six are dependent on the timer operation mode and are explained in more detail below.

The operations modes of the timers are as follows:

TIMER OFF

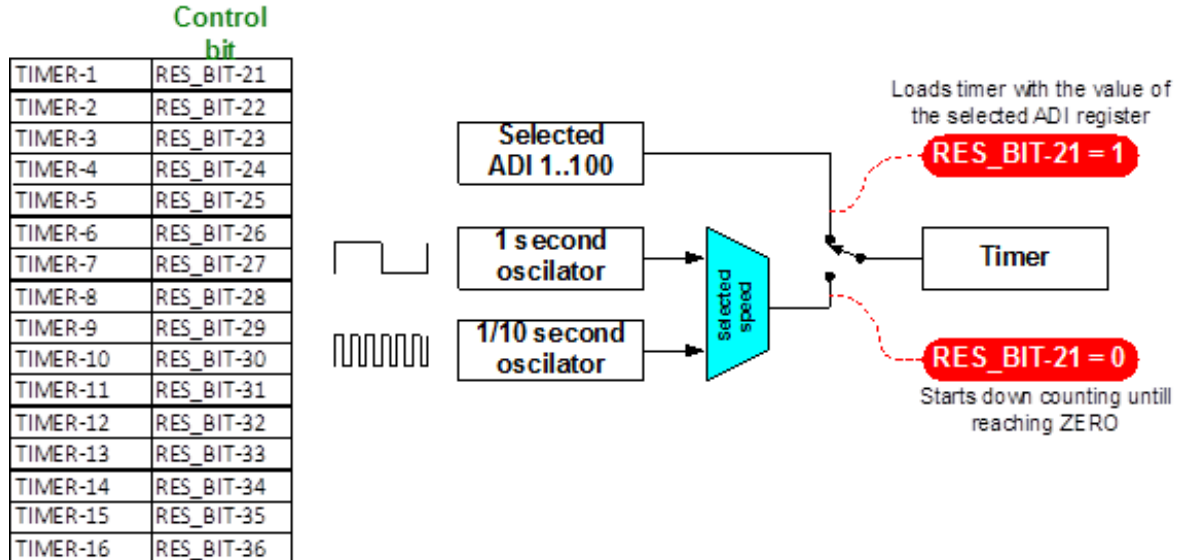
When this option is selected, the selected TIMER will be inactive, and it's current count if it is not already ZERO will be frozen at the current count.

TIMER WITH 1 SECOND or 100 MILISECONDS RESOLUTION



When in this mode of operation, the value of the counter will be loaded (or set) with the value of the selected ADI register, when its control bit is ONE.

When the control bit is ZERO it will begin decrementing the timer's value until it reaches ZERO and then will stop. This operation is represented graphically on the picture below.

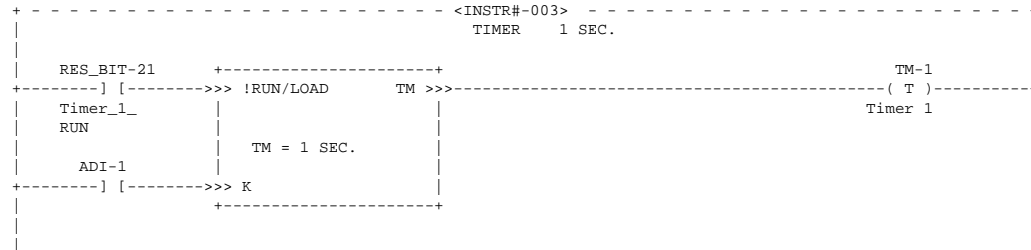


As seen on the table above, timers 1 to 16 are controlled by their corresponding control bits, which are result bit registers RES_BIT 21 to 36.

As these bits can be the result of any logical operation, a schedule or commanded manually, a great flexibility of when the timer is loaded and when running can be obtained.

In turn the TIMER status can be used in any logic block as OPERAND, so it can be tested whether it is ZERO or NOT ZERO.

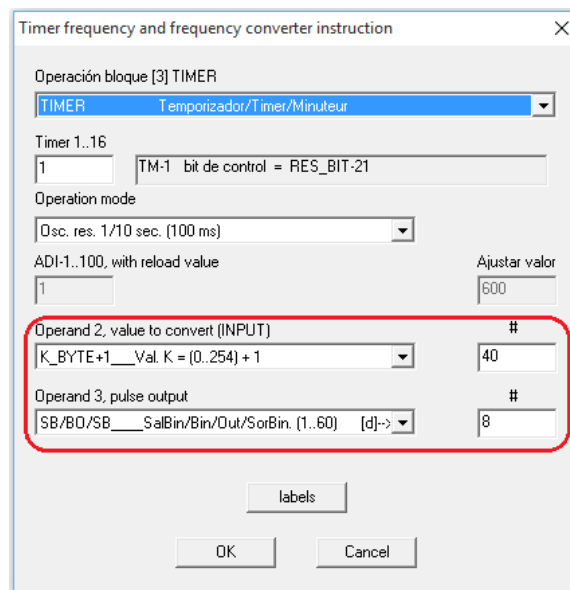
Below is the Ladder diagram from the PLC for a 1 second timer.



OSCILLATOR WITH 1/10 SECOND RESOLUTION

While in this operation mode, the selected timer is an OSCILLATOR whose period will be determined by the value of the selected register in OPERAND 2 divided by 10, to obtain a resolution of 1/10 of a second.

Optionally OPERAND 3 “Pulses Output” can be programmed so that the selected output will oscillate (change between 0 and 1 periodically) with the oscillation period. This parameter can be a RES_BIT register or a Binary Output.



Timer frequency and frequency converter instruction

Operación bloque [3] TIMER

TIMER Temporizador/Timer/Minuteur

Timer 1..16

1 TM-1 bit de control = RES_BIT-21

Operation mode

Osc. res. 1/10 sec. (100 ms)

ADI-1..100, with reload value

1 Ajustar valor 600

Operand 2, value to convert (INPUT)

K_BYTE+1 Val. K = (0..254) + 1 40

Operand 3, pulse output

SB/BO/SB SalBin/Bin/Out/SorBin. (1..60) [d]-> 8

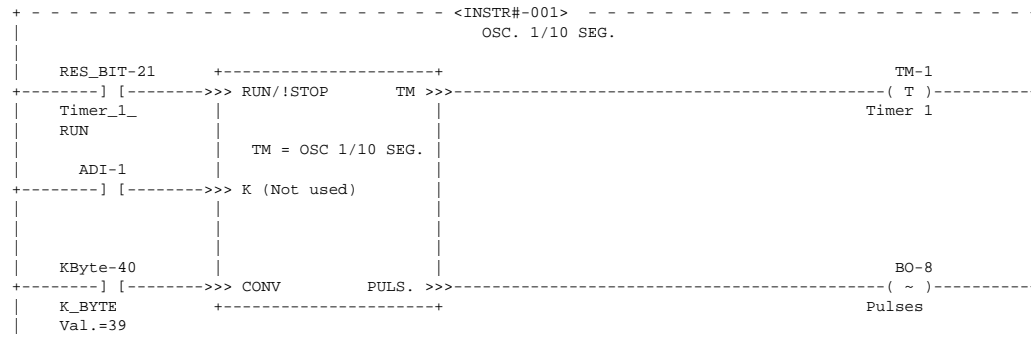
labels

OK Cancel

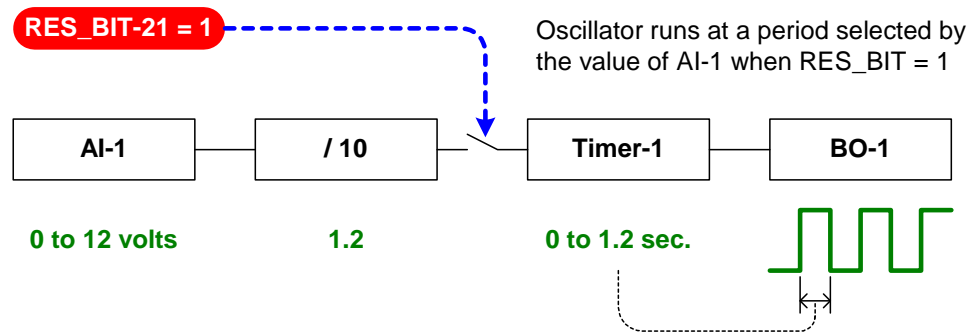
In the above example, TIMER-1 is an OSCILATOR whose output period will be set at a 3.9 seconds interval (39/10), and the output period of the pulses will be output to BINARY_OUTPUT-8.

On next page the Ladder diagram of the PLC is shown.

The timer will run if control register RES_BIT-21 is ONE, or stop if it is ZERO. Below is the ladder diagram representation.



OPERAND 2 that is the period of the oscillator can be any register, so the oscillator period can be varied and not only a fixed value. In the example below, Analog Input 1 is used as the source, so a 0-12 volts will be converted into a 0 to 1.2 seconds period when the control bit of the timer RES_BIT-21 is ONE.



NOTE:

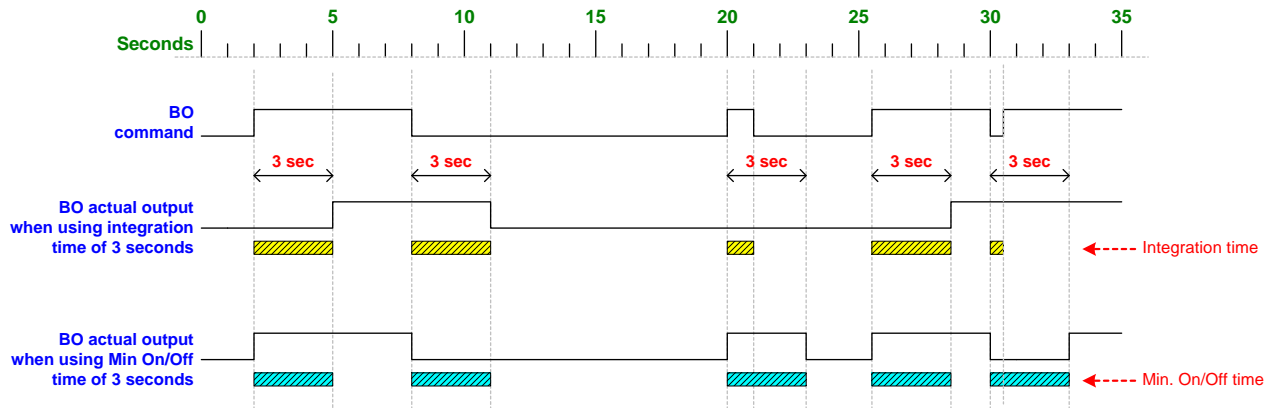
Relays are limited to change at a maximum rate of of 5 times per second (200 ms) to avoid mechanical damage. This value can be set to a higher time to further protect the relays and the associated load. Read following section for detailed instructions.

When on the main screen, by selecting the INFO button, a dialog screen will appear with some system information. At the bottom of this dialog, the integration or the minimum ON/OFF time of the Binary Outputs can be set for each individual local Binary Output (BO-1..8).

If INTEGRATION is selected, upon commanding a Binary Output to a different state than the current one, the Binary Output will not change until this integration time has expired.

If instead MIN ON/OFF mode is selected, once a Binary Output has changed state, it will remain in that state until the Min On/Off time has expired.

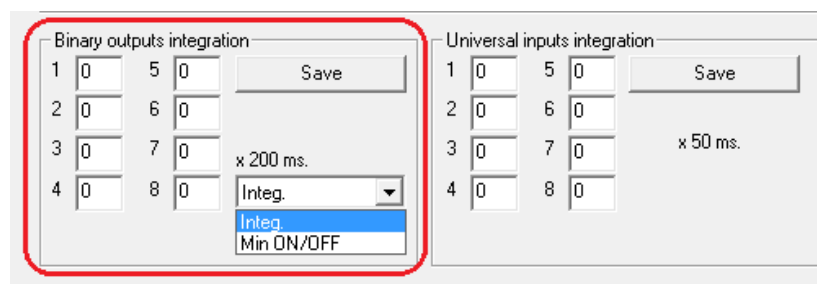
On the picture below Binary Output INTEGRATION or the MIN ON/OFF times mode of operation are depicted.



Here the actual operation of the Binary Output is shown for INTEGRATION and MIN ON/OFF modes when a 3 second (register set to 15 x 0.2) is selected. As shown on the time line if INTEGRATION is selected, the output will only change after the integration time has expired after the command to the Binary Output.

If the Minimum On/Off mode is selected, after a change of state is commanded and the output actually changes state, This minimum timer will prevent any further changes on the output until this timer expires.

The time can be set between 0.2 seconds up to 51 seconds by setting the appropriate BO integration register to a value between 0 to 255, each unit represents a 1/5 of a second (200 milliseconds).



The screenshot shows the configuration interface for the system. The 'Binary outputs integration' section is highlighted with a red box. It contains four rows of configuration for outputs 1 through 4. Each row has three input fields (1, 5, 0) and a 'Save' button. Below the input fields, there is a dropdown menu set to 'Integ.' with 'Integ.' and 'Min ON/OFF' as options. A multiplier 'x 200 ms.' is indicated. The 'Universal inputs integration' section is also visible, showing similar configuration for inputs 1 through 4 with a multiplier of 'x 50 ms.'.



NOTE:

The 200 millisecond lower limit for Binary Outputs can be completely disabled by setting the EEPROM position 32153 to a value of 253 (see XEE_SYS_NO_WAIT_BOs register in the source code). However doing so can damage the relays if improperly used and void the warranty.

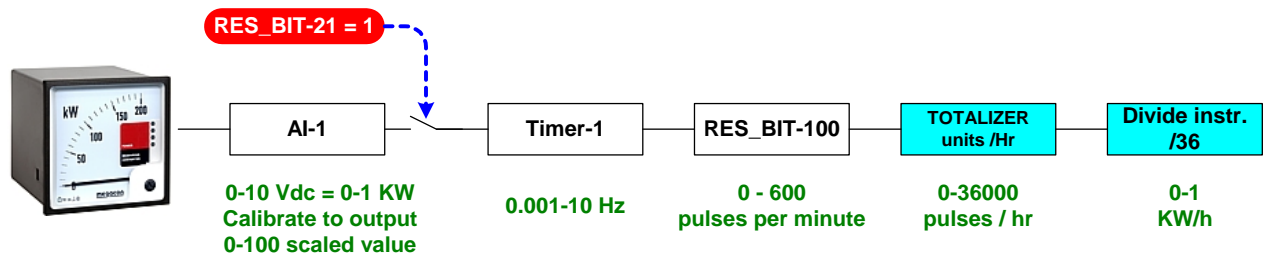
VALUE TO FREQUENCY CONVERTER WITH RESOLUTION OF 1/10 SECOND

This operation mode is similar to the OSCILATOR previously described, the difference of the VALUE TO FREQUENCY converter is that it will convert a value between 0-100, which in this case represents the 0-100% of the scale to a frequency of 0-10 Hz, this represents 0-10 pulses per second or 0-600 pulses per minute.

This can be fed into a totalizer instruction (see section 8.6) to get for example Liters per minute, Kilo Watts per hour, etc. out of analog transducers.

In the next example, TIMER-1 is used as a WATTS to FREQUENCY converter. The output period on the pulses will be the value of the Analog Input 1. In this case a Kilo Watt transducer that has a 0-10 Volts DC output will be fed into Analog Input 1, that needs to be calibrated to give a 0-100 (%) value range with the 0-10 Volts DC input.

This 0-100 % of KW range is input to the Timer 1 which has been set as a Value to frequency converter. If the value is at 100% the output frequency will be 10 Hz outputting 600 pulses per minute or 36,000 pulses per hour, if this value is further divided by 36 the value of 1 KH/w can be obtained.



Knowing that 10 Hz (1/10 seconds.) is the maximum frequency of the value to frequency converter module, we have to scale it correctly to be between a 0-100 range before being fed into the TIMER module to obtain the best possible accuracy.

3.12 Output Assignment, making things happen in the real world

Most of the instructions store their results on internal result registers, and the question is: How to use of those registers to command a hardware output?

The answer is on the ASSIGN OUTPUTS instruction; this instruction is in charge of sending the value of one database object to another. Think of it of as a copy instruction, where there is a SOURCE and DESTINATION, so the value of the source operand is copied into the value of the destination.

If the DESTINATION register happens to be a HARDWARE OUTPUT such as an Analog or Binary Output, then the value gets written directly to the hardware register.

If the DESTINATION register is a Remote Point, then the value is written to the Remote Point instead of being read on the next time that the Remote Point is polled.

The first operand is the SOURCE can be any type defined within the live database, with the exception of NULL type.

EA/AI/EA	EntAnl/AnlgInp/EntAnl.	(1..40)	<-[a]
EB/BI/EB	EntBin/BinInp/EntBin.	(1..40)	<-[d]
SA/AO/SA	SalAnl/AnlOut/SorAnl.	(1..10)	[a]->
SB/BO/SB	SalBin/Bin/Out/SorBin.	(1..60)	[d]->
ADF	EE-Float 32 bit	(1..100)	
ADI	EE-Word 16 bit	(1..100)	
ADB	EE-Byte 8 bit	(1..100)	
RES-BIT	RAM-Result. bit	(1..255)	
RES-FLT	RAM-Result. float	(1..40)	
TIMER	Timer	(1..16)	
RMT	PunRem/RemPnt/PoiElo.	(1..50)	
K_BYTE+1	Val. K = (0..254) + 1		

The second operand is the DESTINATION (or output) and can be any type defined within the live database, with the exception of the following:

- NULL type, Analog Inputs, Binary Inputs and Constants.

EA/AI/EA	EntAnl/AnlgInp/EntAnl.	(1..40)	<-[a]
EB/BI/EB	EntBin/BinInp/EntBin.	(1..40)	<-[d]
SA/AO/SA	SalAnl/AnlOut/SorAnl.	(1..10)	[a]->
SB/BO/SB	SalBin/Bin/Out/SorBin.	(1..60)	[d]->
ADF	EE-Float 32 bit	(1..100)	
ADI	EE-Word 16 bit	(1..100)	
ADB	EE-Byte 8 bit	(1..100)	
RES-BIT	RAM-Result. bit	(1..255)	
RES-FLT	RAM-Result. float	(1..40)	
TIMER	Timer	(1..16)	
RMT	PunRem/RemPnt/PoiElo.	(1..50)	
K_BYTE+1	Val. K = (0..254) + 1		



WARNING:

When using registers in EEPROM such as ADF, ADI, ADB as the DESTINATION , read the warning on netx page to avoid damaging the EEPROM memory cells, as their life is limited to 1,000,000 write cycles.

***** IMPORTANT NOTE *****

The EEPROM registers have a life of 1,000,000 write cycles. When writing to them by being the DESTINATION register inside of an OUTPUT ASSIGN instruction, precautions should be taken so that data is written to the EEPROM only when a specific condition changes.

Writing to the EEPROM registers should be limited to being done only inside conditional expressions, such as those found by employing the conditional jump instructions, that skip some instructions and only execute them when a certain condition is met. When the skipped instructions are executed, they will write the value to the EEPROM register.

As an additional protection, the operating system first reads the value on EEPROM and only writes it back if the new value is different from the old one.

A register in EEPROM must never be assigned as a DESTINATION, if the SOURCE variable will be always changing, such as an analog input or the result register of a PID.

Writing directly to a register in EEPROM without the previously listed precautions, can damage the cell in a period of time as short as 10 days.

If the DESTINATION of an OUTPUT ASSIGN instruction is a Remote Point, the value will then be sent to the Remote Point in the currently selected protocol.

There is a programming parameter in ADJUST SYSTEM PARAMETERS that indicates how the command will be sent, either as an Override or simple Write. See section 6.16.1 Configure the Fieldbus Communication Ports, for detailed instructions.

If the SOURCE operand is binary, the value it will take will be ZERO or ONE only.

If instead the SOURCE is some analog variable and the DESTINATION is digital, the output will be ZERO if the SOURCE operand is ZERO, and will be ONE for any other value of the SOURCE operand.

Since version 2.50 the option of assigning two outputs in a single instruction (dual assignment) is available, see the OUTPUT ASSIGN dialog and the generated Ladder diagram on the next page.

Output assign

Instruction ###
 OUTP_ASSIGN Salida/Output/Sortie/Ausgang/Uscita/Uitgang

First output assignment

Operand 1 (SOURCE) #
 EA/AI/EA___EntAnl/AnlgInp/EntAnl. (1..40) <-[a] 1
 AI-1 []

Operand 2 (DESTINATION) = Output #
 RES_FLT___RAM-Result. float (1..40) 129
 RES_FLT-129 []

Exchange first and second assignment

Second output assignment

☒ Enable double assignment

Operand 3 (SOURCE) #
 EB/BI/EB___EntBin/BinInp/EntBin. (1..40) <-[d] 1
 BI-1 [Door_#1]

Operand 4 (DESTINATION) = Output #
 SB/BO/SB___SalBin/Bin/Out/SortBin. (1..60) [d]-> 7
 BO-7 []

OK Exit
 Dual core active.
 Edit tags labels

Result bits RES_BITS	
RES_BIT-1..20	Lighting groups 1..20
RES_BIT-21..36	Timers 1..16
RES_BIT-37,38,39	Fixed timers 1,2,4 seconds
RES_BIT-40..246	User free
RES_BIT-247	Online as slave COM1
RES_BIT-248	Online as slave COM2
RES_BIT-249	Fixed timer 1 minute
RES_BIT-250	Local override PB1
RES_BIT-251..253	Power ON timer +1, +5, +10 seconds
RES_BIT-254	LED green
RES_BIT-255	LED red

```

+-----<INSTR#-006>-----
|                               ASSIGN OUTPUTS
+-----+-----+-----+-----+-----+-----+-----+-----+
| AI-1                                     RES_FLT-130                                |
| ] [                                     - ( R ) -                             |
| SOURCE                                 DESTINATION                           |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----<INSTR#-005>-----
|                               ASSIGN OUTPUTS (DUAL)
+-----+-----+-----+-----+-----+-----+-----+-----+
| AI-1                                     RES_FLT-129                                |
| ] [                                     - ( R ) -                             |
| SOURCE-1                               DESTINATION-1                         |
+-----+-----+-----+-----+-----+-----+-----+-----+
| BI-1                                   (      # 2      )                       BO-7          |
| ] [                                     - ( R ) -                             |
| SOURCE-2                               DESTINATION-2                         |
+-----+-----+-----+-----+-----+-----+-----+-----+

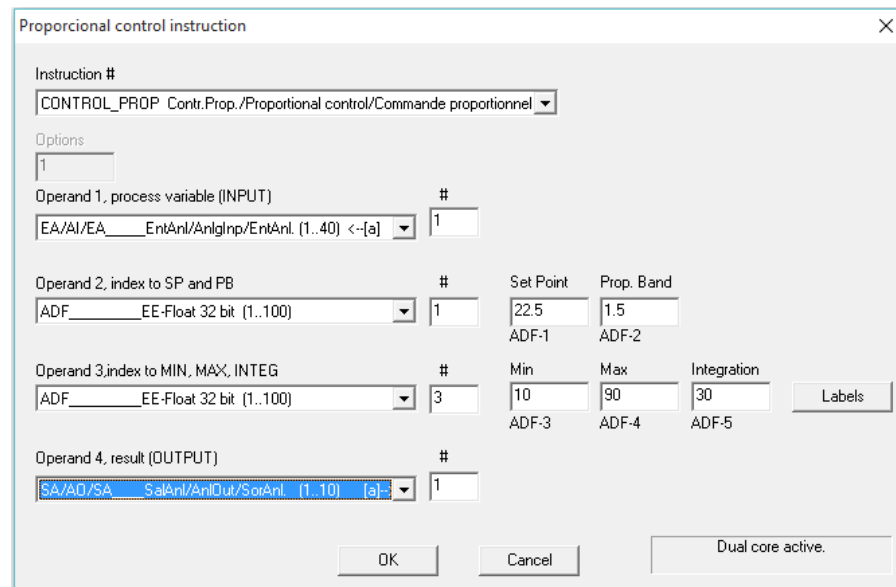
```

3.13 Proportional Control, PID for Automation and HVAC applications

This instruction creates control loops for speed drives, proportional valves for cooled or hot water, positioning dampers, dimming circuits, etc. that previously could only be obtained with complex sequences on the PLC or in dedicated ASC (Application Specific Controller) equipment with great effort and difficult start up.

All control loops require parameters to be configured, and this one is not the exception. The difference is that it is designed so that for that any programmer or technician that with a basic comprehension in control, can in a pair of minutes set up and fine tune a control loop.

As shown in the next picture of a **PROPORTIONAL CONTROL** there are the following fields available:



Proportional control instruction

Instruction #
 CONTROL_PROP Contr.Prop./Proportional control/Commande proportionnel

Options
 1

Operand 1, process variable (INPUT) #
 EA/AI/EA EntAnl/AnlInp/EntAnl. (1..40) <-[a] 1

Operand 2, index to SP and PB #
 ADF EE-Float 32 bit (1..100) 1

Set Point Prop. Band
 22.5 1.5
 ADF-1 ADF-2

Operand 3, index to MIN, MAX, INTEG #
 ADF EE-Float 32 bit (1..100) 3

Min Max Integration
 10 90 30
 ADF-3 ADF-4 ADF-5

Labels

Operand 4, result (OUTPUT) #
 SA/AQ/SA SaAnl/AnlOut/SoAnl. (1..10) [a] 1

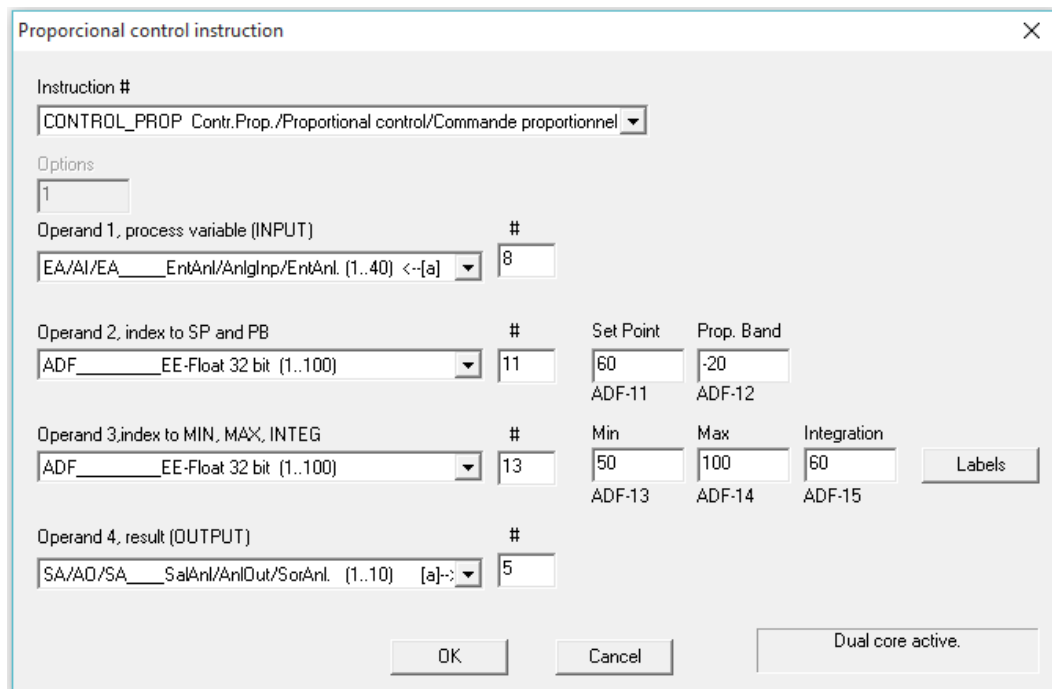
OK Cancel Dual core active.

- **Process variable or PV** This is the input (eyes) of a control loop, the variable that needs to be controlled.
- **Set point** It is the target value for the process variable in the loop.
- **Proportional band** Is a window inside where the process variable is expected to swing when the proportional control is operating correctly.
- **Minimum and maximum** Are the ranges inside of which the output can swing.
- **Integration** It is the speed at which the output varies.
- **Output** Is the control command (or hands) of the proportional control loop.

With the previous definitions known, an example of how a control loop works to control the pressure of a pump equipped with a variable speed drive or VFD.

In this example, a proportional loop whose control or PROCESS VARIABLE is the Analog Input eight AI-8, and whose control OUTPUT or result is Analog Output five AO-5. The adjust point or SET POINT is the value of register in AAPROM ADF-11 and the PROPORTIONAL BAND is the ADF-12.

The MINIMUM speed for the VFD is stored in register EEPROM ADF-13, the MAXIMUM on the ADF-14 and the change of speed or INTEGRATION constant the ADF-15.



Proportional control instruction

Instruction #
 CONTROL_PROP Contr.Prop./Proportional control/Commande proportionnel

Options
 1

Operand 1, process variable (INPUT) #
 EA/AI/EA EntAnl/AnlInp/EntAnl (1..40) <-[a] 8

Operand 2, index to SP and PB # Set Point Prop. Band
 ADF-EE-Float 32 bit (1..100) 11 60 -20
 ADF-11 ADF-12

Operand 3, index to MIN, MAX, INTEG # Min Max Integration
 ADF-EE-Float 32 bit (1..100) 13 50 100 60
 ADF-13 ADF-14 ADF-15

Operand 4, result (OUTPUT) #
 SA/AO/SA SalAnl/AnlOut/SorAnl (1..10) [a]-> 5

OK Cancel Dual core active.

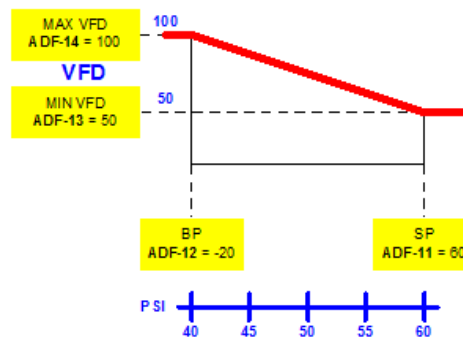
For this example, an inverse control loop is being used because the proportional band is set to a negative value (-20) in this case, a direct control loop, can be set in two different ways:

- Making the proportional band positive.
- Inverting the MIN and MAX values.

So for this example with a PROPORTIONAL BAND of -20, the control OUTPUT or result will be the MINIMUM or 50% when the pressure is 60 PSI, and will be the MAXIMUM when the pressure drops down to 60 PSI (SP-BP).

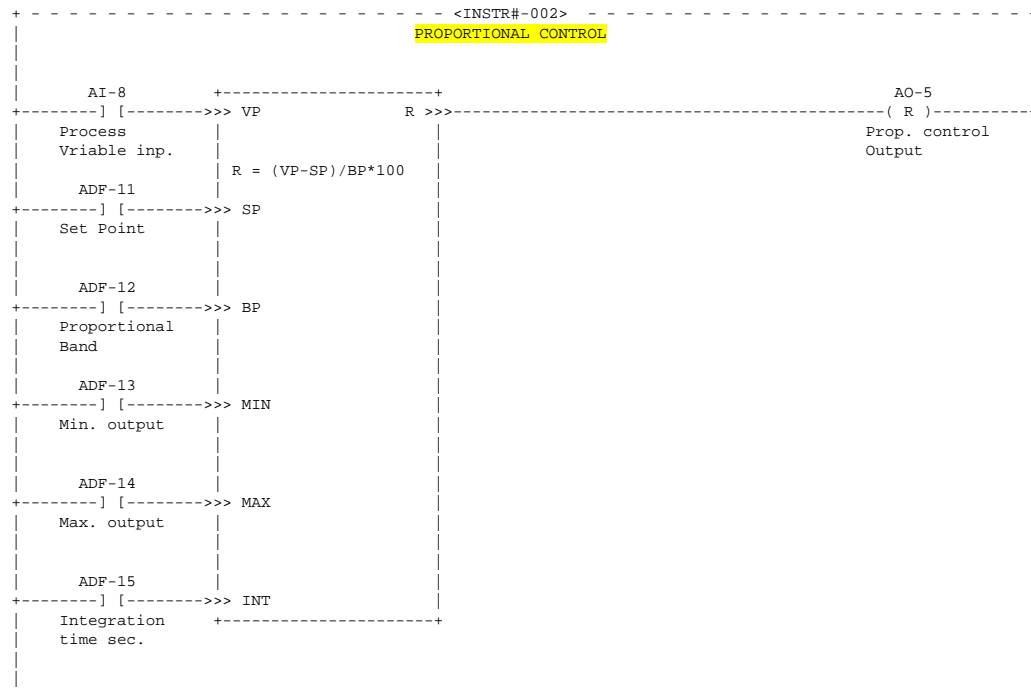
The rate of change (or speed of change) of the OUTPUT is given by the INTEGRARION value, in this case ADF-15. For this example, it has a value of 60 and for a sudden process variable change from 40 to 60 PSI, the OUTPUT will change in 60 seconds from the 100% MAXIMUM to the 50% MINIMUM.

A value of ZERO in the integration value will enable that the OUTPUT to change instantly from the MAXIMUM to MINIMUM value or vice versa.



As we can be seen on the picture above with the red line of the proportional control, the output will stay on the MAXIMUM for values below $PV < SP - BP$, and will stay in the MINIMUM for values above $PV > SP$.

The following is a Ladder diagram generated from a proportional control instruction.

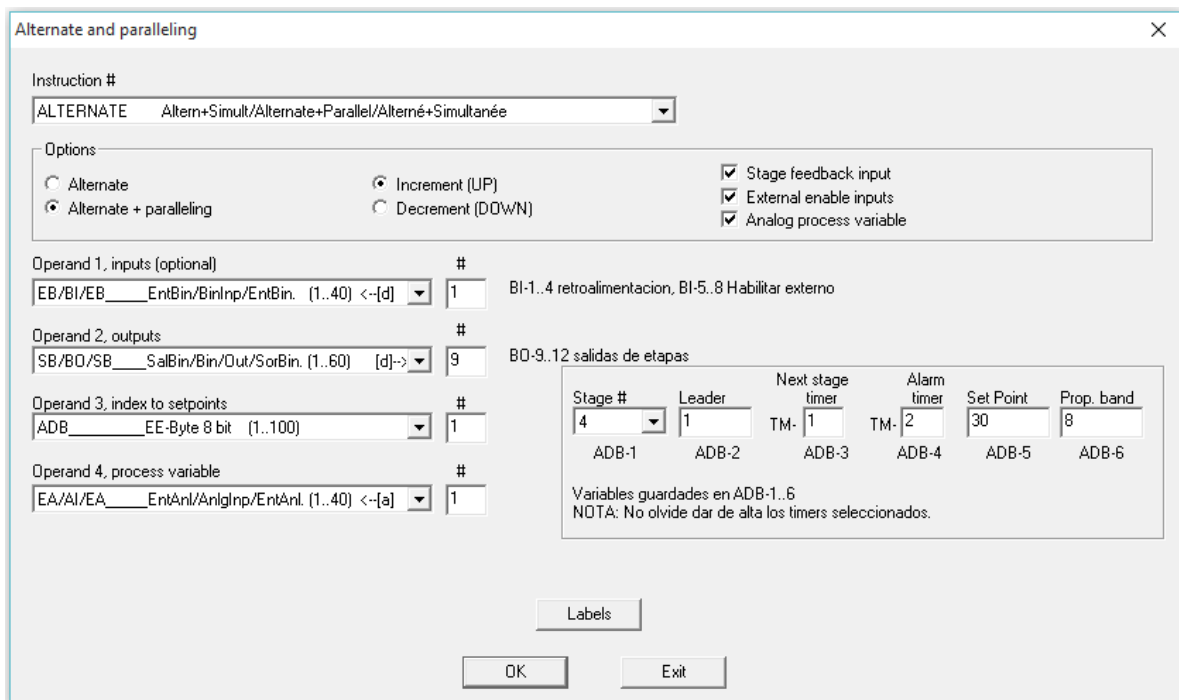


3.14 Alternate and Paralleling, Alternate and Paralleling of Pumps and Machinery

With this block is possible to alternate and optionally parallel equipment operation such as:

- PUMPS
- AIR HANDLERS
- FANS
- CHILLERS

This instruction programs form 2 up to 8 alternating or paralleling of outputs with a single instruction.



Alternate and paralleling

Instruction #
 ALTERNATE Altern+Simult/Alternate+Parallel/Alterné+Simultanée

Options

☐ Alternate
☒ Alternate + paralleling

☒ Increment (UP)
☐ Decrement (DOWN)

☒ Stage feedback input
☒ External enable inputs
☒ Analog process variable

Operand 1, inputs (optional) #
 EB/BI/EB EntBin/BinInp/EntBin. (1..40) <-[d] 1 BI-1..4 retroalimentacion, BI-5..8 Habilitar externo

Operand 2, outputs #
 SB/BO/SB SalBin/Bin/Out/SorBin. (1..60) [d]-> 9 BO-9..12 salidas de etapas

Operand 3, index to setpoints #
 ADB EE-Byte 8 bit (1..100) 1

Operand 4, process variable #
 EA/AI/EA EntAnl/AnlInp/EntAnl. (1..40) <-[a] 1

Stage #	Leader	Next stage timer	Alarm timer	Set Point	Prop. band
4	1	TM-1	TM-2	30	8
ADB-1	ADB-2	ADB-3	ADB-4	ADB-5	ADB-6

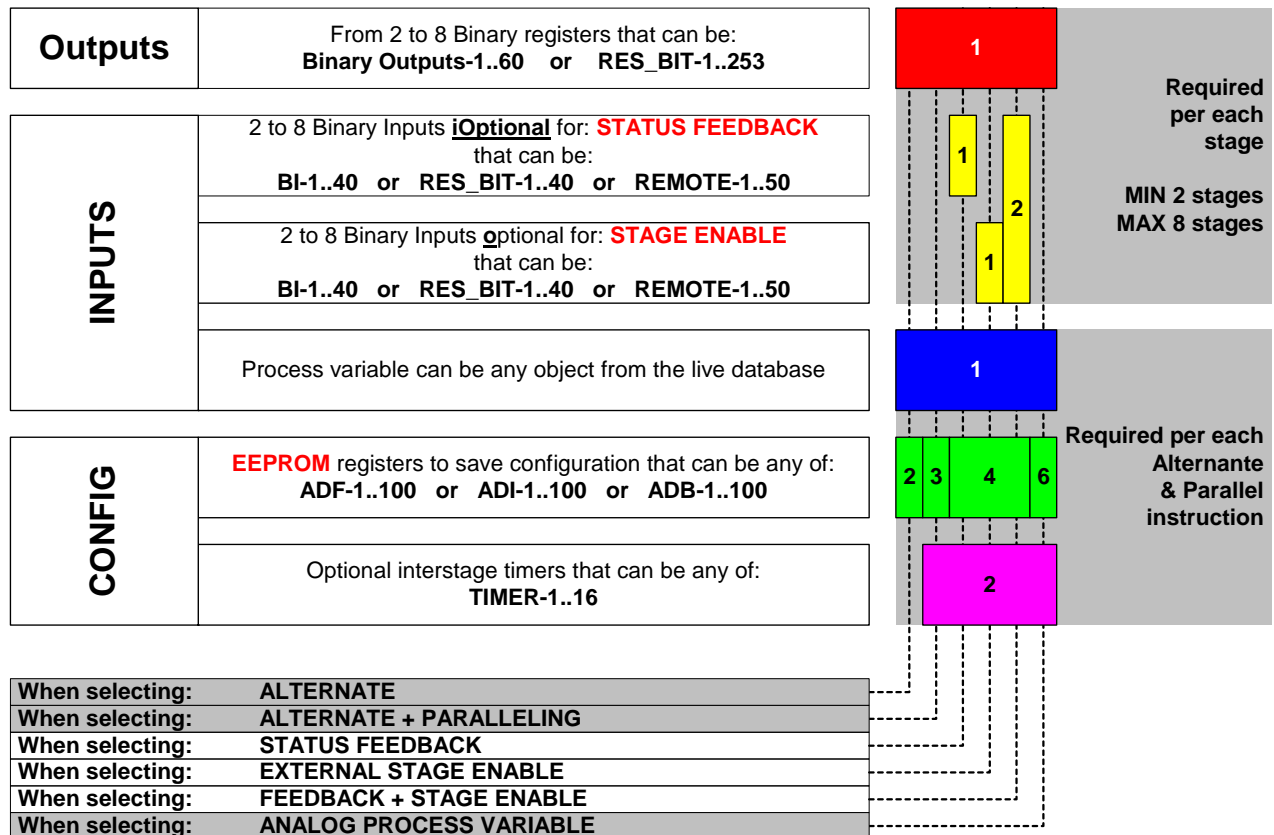
Variables guardadas en ADB-1..6.
 NOTA: No olvide dar de alta los timers seleccionados.

Labels

OK Exit

In the picture on next page, all necessary inputs and outputs and parameters used to program an alternate / parallel instruction, with all their different options are shown.

Table containing all the necessary Inputs, Outputs and Configuration data for an Alternate and Paralleling instruction.



As can be seen on the table, between 2 to 8 stages can be set up for this instruction. Each stage requires at minimum one output. The number of inputs required for this instruction is at least one input for the process variable.

That can be binary for signals as:

- Level switches.
- Pressure switches.
- Etc.

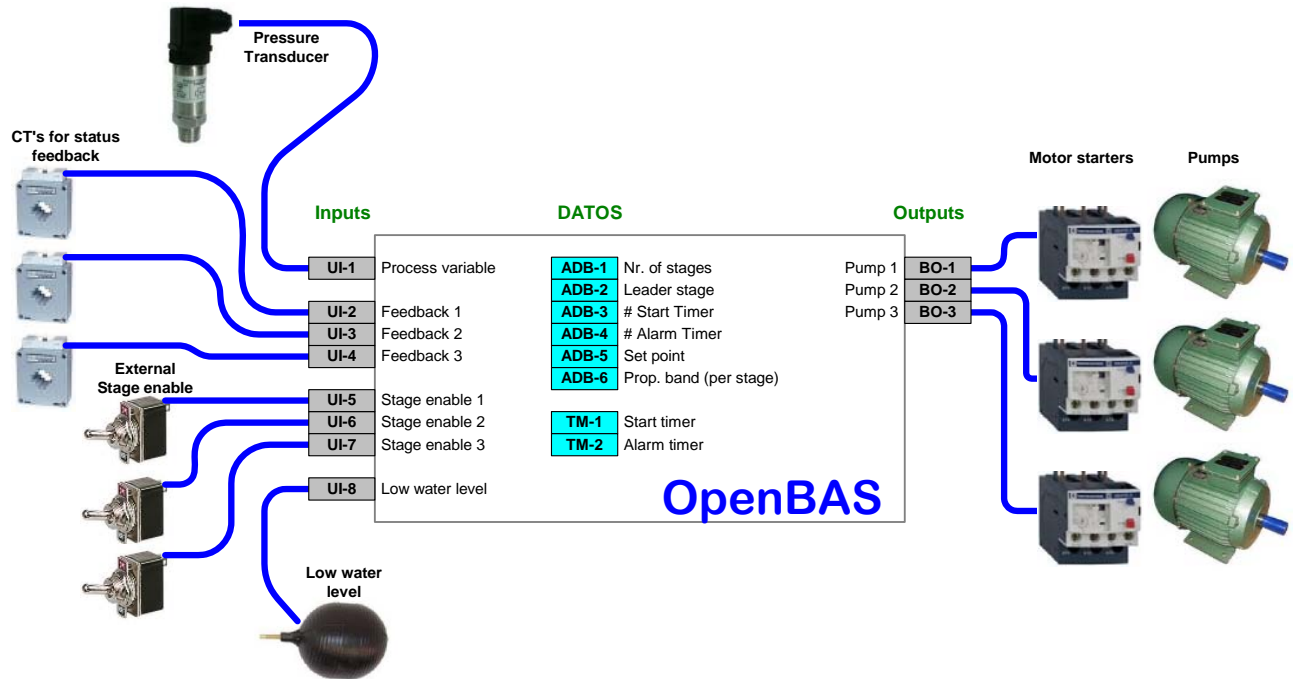
Or also can be analog for sensors and transducers such as:

- Pressure
- Temperature
- Etc.

A typical diagram to implement an ALTERNATE and PARALLELING module for 3 stages, using a pressure transducer to control the start and stop of the three pumps, with status feedback and an external stage enable signal is shown in diagram on next page.

There is also a level switch used to stop when low water level is detected. This is not part of the instruction but should be added as a protection to stop any pump to run if low water is detected.

Typical diagram for an alternate and paralleling circuit to control up to three pumps, between 2 and up to 8 can be set up per instruction.



When only ALTERNATE is selected, the module starts the selected leader pump when the pressure (Process variable) falls below SP-PB. When pressure increases and reaches a value of SET POINT, the pump is stopped, and the leader pump variable is incremented (or decremented depending on the setup) so the next time the PV falls below SET POINT-PROPORTIONAL BAND the next pump in the sequence will start.

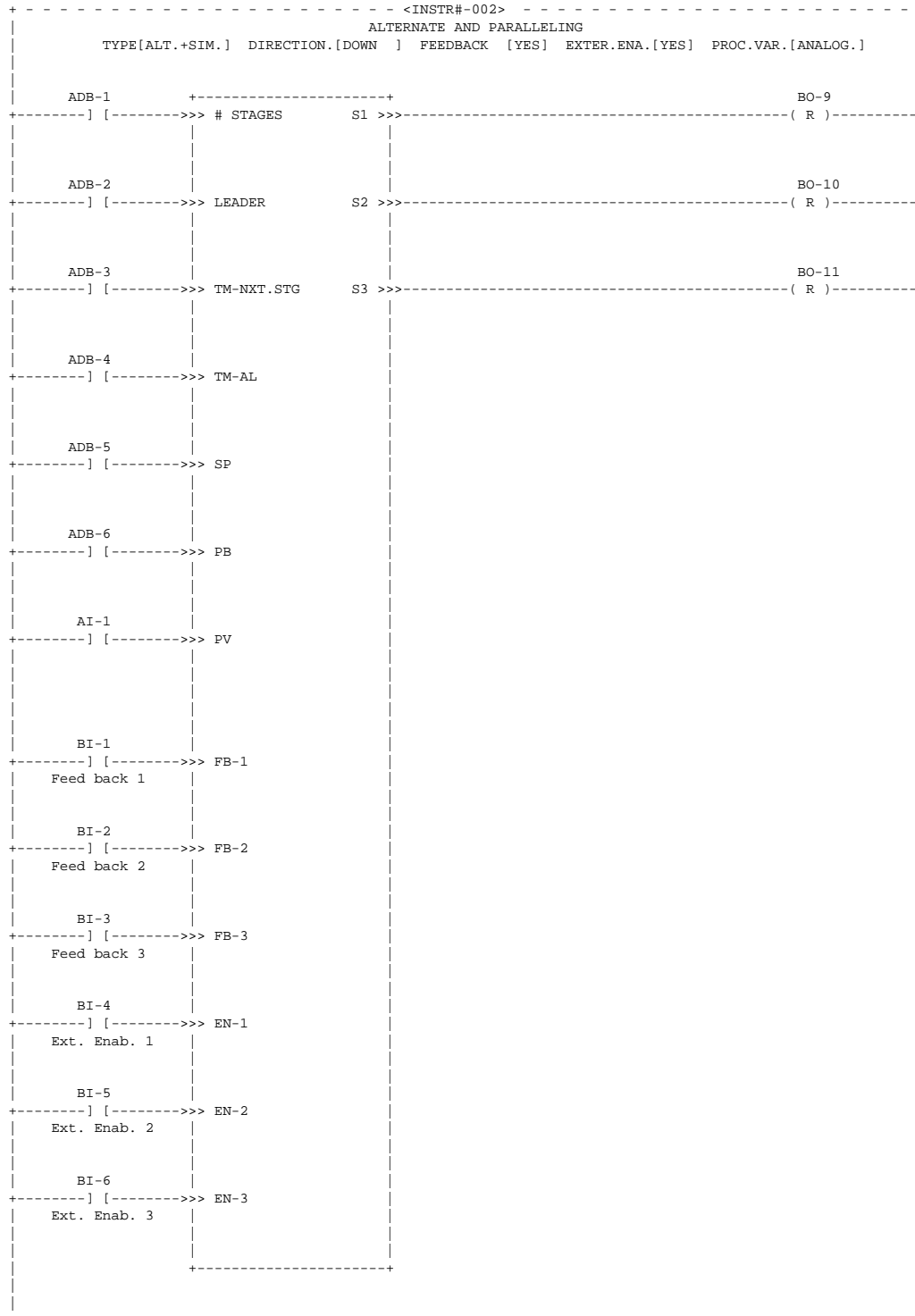
If the STATUS FEEDBACK option is selected, the ALARM TIMER will start to run when the pump is started. Before the Alarm Timer expires, the State Feedback signal has to become active (1), in case this doesn't happen. When the Alarm Timer expires, it will turn OFF that pump and start the next pump in the sequence.

The Status Feedback input doesn't necessarily have to be a binary input, it can be any register. This gives a lot of flexibility as the Status Feedback register can be the result of logic functions.

If the option EXTERNAL STAGE ENABLE is selected, the corresponding stage will only be started if the corresponding stage input is active (1).

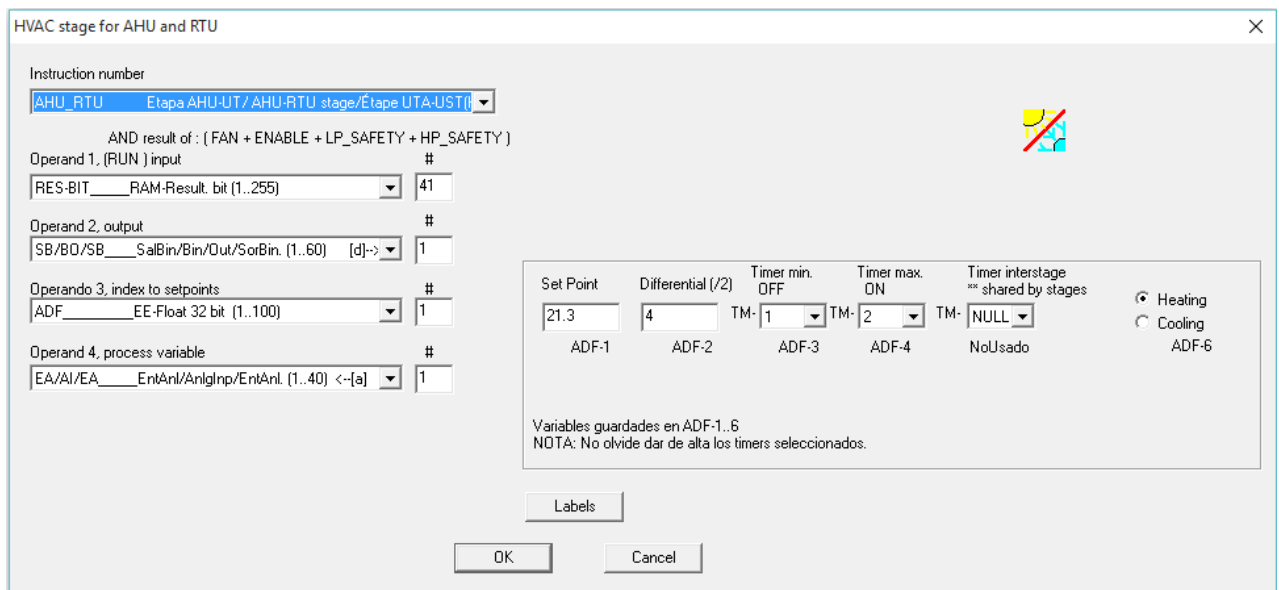
When the ALTERNATING + PARALLELING mode is selected, when the first pump starts, the NEXT STAGE TIMER will also start running. Once the time has expired, and if the PROCESS VARIABLE hasn't reached the SET POINT value, the module will start the next pump in the sequence, and will repeat again the timer process, if again the timer expires and the pressure hasn't reached the SET POINT level, it will start the third pump or as many as stages are programmed. Once the process value reaches the set point, it will turn all the pumps OFF.

The following picture is a Ladder diagram, as generated for the Alternate & Paralleling instruction of the example described on the previous page.



3.15 AHU and RTU staging, Create easily Simple or Complex HVAC sequences

This block allows the creation of stages for HVAC machines such as: Air handling Units (AHU) and/or Roof Top Units (RTU), by joining several instructions. A strategy to control HVAC equipment with multiple stages can be created.



HVAC stage for AHU and RTU

Instruction number
 AHU_RTU Etapa AHU-UT / AHU-RTU stage / Étape UTA-UST (i)

AND result of : (FAN + ENABLE + LP_SAFETY + HP_SAFETY)

Operand 1, (RUN) input #
 RES-BIT RAM-Result. bit (1..255) 41

Operand 2, output #
 SB/BO/SB SalBin/Bin/Out/SorBin. (1..60) [d]-> 1

Operando 3, index to setpoints #
 ADF EE-Float 32 bit (1..100) 1

Operand 4, process variable #
 EA/AI/EA EntAnl/Anlglnp/EntAnl. (1..40) <-[a] 1

Set Point	Differential (/2)	Timer min. OFF	Timer max. ON	Timer interstage *** shared by stages	
21.3	4	TM- 1	TM- 2	TM- NULL	<input checked="" type="radio"/> Heating
ADF-1	ADF-2	ADF-3	ADF-4	NoUsado	<input type="radio"/> Cooling ADF-6

Variables guardades en ADF-1..6
 NOTA: No olvide dar de alta los timers seleccionados.

Labels

OK Cancel

- The **operand 1** (RUN) indicates if this stage is operational or not. If it is desired to be always operational, the operand value must be ≥ 1 , and can be a constant.
- The **operand 2** (OUTPUT) is the bit that turns the stage ON / OFF.
- The **operand 3** (SETPOINTS) is an index into variables of type EEPROM where the operating parameters of this block will be stored, these parameters are the following:
 - Set point, normally it is a temperature set point.
 - Differential, it is the proportional band for a stage to be turned ON or OFF with hysteresis, for example if the **SP = 25** and the **PB = 3**, for cooling, the stage will be on ON if the process variable is ≥ 26.5 ($SP + BP/2$) and on OFF if the process variable ≤ 23.5 ($SP - BP/2$), this logic is reversed for heating.

- **The timer for minimum time OFF**, is optional and it is an index to the Timer that keeps the count of the minimum time that this stage should be on OFF, once that it changes from ON to OFF.
- **The timer for minimum time ON**, is optional and it is an index to the Timer that keeps the count of the minimum time that this stage should be on ON, once that it changes from OFF to ON.
- **The timer for inter-stages**, is optional and it is an index to the timer that keeps the count between any stage change, to avoid that multiple stages are set ON or OFF simultaneously. This timer will be shared for all the stages of a same machine.
- **Heating or cooling**, is a flag that indicates if this stage is of heating or cooling, and invert the comparison logic between the process variable and the set point +/- the differential.
- The **operand 4 (process variable)** tells which is the variable used to control the stage.

The following example is a typical program to control an AHU with stages using these control blocks on an OpenBAS-HV-XN10P controller, some lines have been removed for clarity.

```

+-----< OpenBAS >-----+
+-----< INSTR#-001 >-----+
+-----SECTION DIVIDER LABEL-----+
+-----+-----+
+-----UMA 2-ET-----+
+-----+-----+
+-----< INSTR#-002 >-----+
+-----TIMER 1 SEC.-----+
+-----+-----+
+-----RES_BIT-25-----+-----TM-5-----+
+-----] [----->>> !RUN/LOAD TM >>>-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----ADI-5-----+-----TM = 1 SEC.-----+-----T._intere stage-----+
+-----] [----->>> K-----+-----+
+-----K_t_inter stage +-----+
+-----< INSTR#-003 >-----+
+-----AND-----+
+-----RES_BIT-1-----RES_BIT-2-----BI-1-----BI-2-----RES_BIT-41-----+
+-----] [-----] [-----] [-----] [-----] [-----+-----+-----+
+-----Arr Fan H-----Hab Et-1-----Prot_Baja-----Prot_Alta-----Etapa 1 R-----+
+-----orario-----x horario-----Pres 1-----_Pres_1-----UN-----+
+-----< INSTR#-004 >-----+
+-----RES_BIT-1-----RES_BIT-3-----BI-3-----BI-4-----RES_BIT-42-----+
+-----] [-----] [-----] [-----] [-----] [-----+-----+-----+
+-----Arr Fan H-----Hab Et-2_-----Prot_Baja-----Prot_Alta-----Etapa 2_R-----+
+-----orario-----x_horario-----_Pres_2-----_Pres_2-----UN-----+
+-----< INSTR#-005 >-----+
+-----ASSIGN OUTPUTS-----+
+-----RES_BIT-1-----+-----BO-3-----+
+-----] [-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----Arr Fan H-----+-----Ventilado-----+
+-----orario-----+-----r UMA-----+
+-----< INSTR#-006 >-----+
+-----CALLS SUBROUTINE-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+----->>> SUBR. INSTR.-----= 010-----+
+-----| [ STAGE-1 ]-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----< INSTR#-007 >-----+
+-----CALLS SUBROUTINE-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+----->>> SUBR. INSTR.-----= 015-----+
+-----| [ STAGE-2 ]-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----END-----+
+-----< OpenBAS >-----+
+-----< (program continues) >-----+
  
```

In the program on last page we can see the following relevant instructions:

- The instruction #1 is just a label.
- In instruction #2, the common inter-stage timer TM-5 is created,
- In instructions #3 and 4, protections for high and low pressure that enable the stages 1 and 2 respectively, that are in series with the schedule bits to turn ON the main fan and the corresponding stages during a period of the day.
- The instruction #5 starts the air-handler fan.
- The instructions #6 and 7 call subroutines for stages 1 and 2 respectively.
- The instruction #8 is the END of the program.

Following is the listing of the call graph of the whole program, where these relationships are shown:

```
*****
Jump and subroutine call graph
Last used instruction number = (19)
*****
#      Instruction      Notes
===      =====
001      [SECTIO_LABEL]  Name:    [UMA 2-ET ]
002      [TIMER         ]
003      [AND           ]
004      [AND           ]
005      [OUT_ASSIGN    ]
006      [SUB_CALL      ] ===( Subroutine call in block    [ 10] )===> [STAGE-1  ]
007      [SUB_CALL      ] ===( Subroutine call in block    [ 15] )===> [STAGE-2  ]
008      [PROGRAM_END   ] ***( Last PLC instruction      )***
009      [PROGRAM_END   ] ***( Last PLC instruction      )***

-----
[ SUBROUTINE START ]    Name: [STAGE-1  ]
-----
010      [SECTIO_LABEL]  Name:    [STAGE-1  ]
011      [AHU_STAGE     ]
012      [TIMER         ]
013      [TIMER         ]
014      [SUB_END       ] <===( Subroutine ends          )===

-----
[ SUBROUTINE START ]    Name: [STAGE-2  ]
-----
015      [SECTIO_LABEL]  Name:    [STAGE-2  ]
016      [AHU_STAGE     ]
017      [TIMER         ]
018      [TIMER         ]
019      [SUB_END       ] <===( Subroutine ends          )===
```

The listings on the next pages describe the subroutines in detail.

Now comes the Ladder diagram for subroutine for stage ONE.

- Instruction ten is just a label used for documenting, so this name appears in the subroutine call (See instruction six in the previous page).
- The instruction number eleven is the stage one configuration function of the AHU, which is set for cooling, and you can see all of their configuration parameters below.
- Instructions twelve and thirteen define the minimum OFF time timers and minimum ON timers that should be set up, that were selected in the stage configuration as TM-1 and TM-2 respectively.
- Instruction fourteen indicates the **END of the subroutine**.

```

<INSTR#-010>
SECTION DIVIDER LABEL
+=====+
+-----+ STAGE-1 +-----+
+=====+
<INSTR#-011>
AHU / RTU STAGE (HVAC)

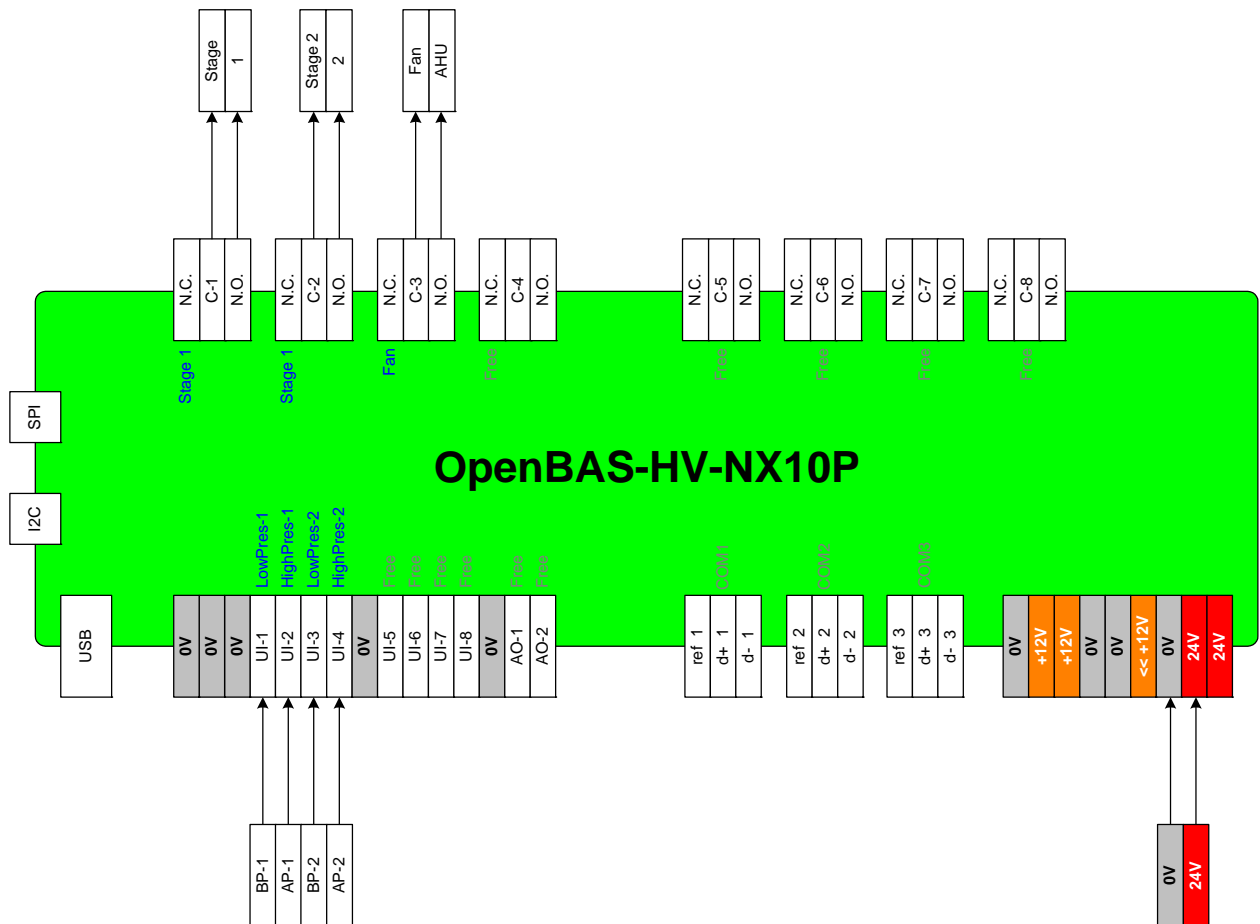
RES_BIT-41 +-----+ BO-1
+-----] [----->>> RUN OUTPUT. >>> ( R )
RUN stage 1 |-----+ Output Stage 1
| (ON) VP >= SP+DIF/2
| (OFF) VP <= SP-DIF/2
AI-1 | (Invert. for heat.)
+-----] [----->>> VP
Temperature
ADF-1 |
+-----] [----->>> SP
SP stage_1
ADF-2 |
+-----] [----->>> DIF (/2)
DIFF stage_1
ADF-3 |
+-----] [----->>> TM MIN. OFF
Idx tm off_1
ADF-4 |
+-----] [----->>> TM MIN. ON
Idx tm on_1
ADF-5 | (shared by all stg.)
+-----] [----->>> TM INTERSTAGES **
Idx tm interStg_2
ADF-6 |
+-----] [----->>> Heat=0/Cool=1
! Heat/Cool +-----+
<INSTR#-012>
TIMER 1 SEC. TM-1
+-----] [----->>> !RUN/LOAD TM >>> ( T )
TM = 1 SEC. T._min_OFF_1
ADI-1 |
+-----] [----->>> K
K t off 1 +-----+
<INSTR#-013>
TIMER 1 SEC. TM-2
+-----] [----->>> !RUN/LOAD TM >>> ( T )
TM = 1 SEC. T._min_ON_1
ADI-2 |
+-----] [----->>> K
K_t_on_1 +-----+
<INSTR#-014>
END
---(program continues)-----< OpenBAS >-----

```

- Block fifteen is just a label used for documenting, so this name appears in the subroutine call (See block seven in the previous page).
- Block sixteen is the function block of stage two of the AHU, which is configured for cooling, and you can see all their configuration parameters below.
- Blocks seventeen and eighteen define the minimum OFF time timer and minimum ON timer that should be set up, that were selected on the stage configuration as TM-3 and TM-4 respectively.
- Block nineteen indicates the **END of the subroutine**.

140

The connection diagram for an AHU control with two stages is shown below:



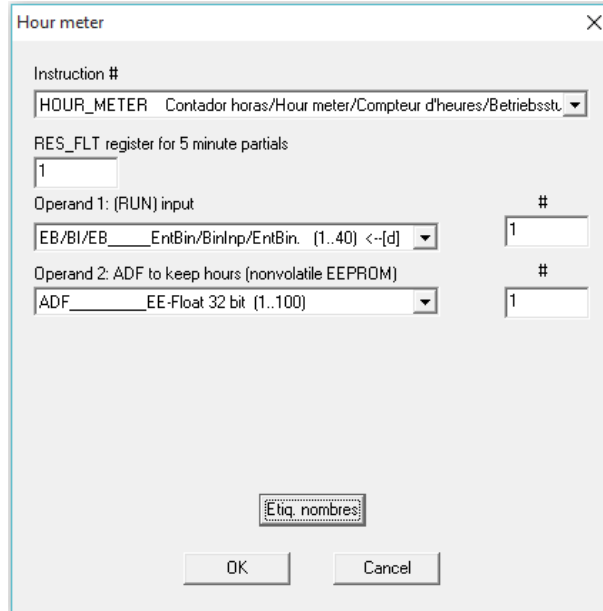
3.16 Hour Counter, Create an hour counter

This instruction can keep a totalized count of hours of operation; it is used for example to keep hours of use of:

- Pumps, Engines, Compressors, Open doors, Out or range, etc.

On next page the dialog to define an Hour Counter instruction is shown.

Dialog to set up an Hour Counter instruction.



The dialog box is titled "Hour meter". It contains the following fields:

- Instruction #:** A dropdown menu showing "HOUR_METER Contador horas/Hour meter/Compteur d'heures/Betriebsst".
- RES_FLT register for 5 minute partials:** A text box containing the value "1".
- Operand 1: (RUN) input:** A dropdown menu showing "EB/BI/EB EntBin/BinInp/EntBin. (1..40) <-[d]" and a small text box containing "1".
- Operand 2: ADF to keep hours (nonvolatile EEPROM):** A dropdown menu showing "ADF EE-Float 32 bit (1..100)" and a small text box containing "1".
- Etq. nombres:** A button.
- OK** and **Cancel** buttons at the bottom.

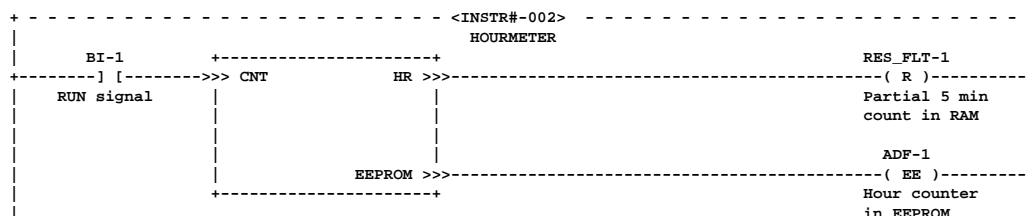
First set register RES_FLT that will keep the partial count of operation hours every 5 minutes.
 Operand 1, is the source of the hours counter, will increase the count every time its value is ONE.
 Operand 2, is the register of type **ADF** that will contain the hour meter register in EEPROM.

NOTE:

The value stored in the (Operand-2) **ADF** in the EEPROM is updated with the RAM value stored in register **RES_FLT** (second field) every 5 minutes only, because the EEPROM has a life of 1,000,000 writing cycles.

It is advised to disable the option of: **CLEAR REGISTERS RES_FLT** in the option of **START RAM @ POWERON**, to avoid that in case of power failure or control reset, this partial count of 5 minutes is lost.
 See section 6.16.8 Set up what happens to Memory and I/O on Power On

The following is the Ladder diagram of an Hour Counter instruction



3.18 Special User Programs Link your interface to user “C language” created Instructions

When the need comes that a very complex or repetitive automation task has to be implemented, there is no substitute for a high level language. This is where the C Language comes to the rescue.

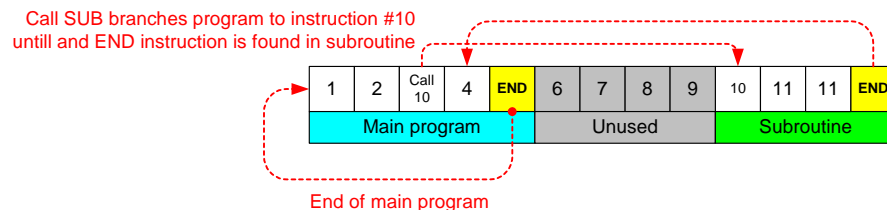
See section 7.14 Advanced C Programming for repetitive and advanced users, for more information on special instructions and how to set up the Special instruction.

3.19 Program End Insert END instructions to terminate PLC or exit subroutine calls

When the PLC comes from the factory, all 400 instructions contain the END instruction, so as you move forward in the program, those END instructions will be replaced by active instructions.

When the PLC encounters an END instruction in the main program it resets the instruction counter to 1 and starts executing from the beginning of the PLC program again.

When calling a subroutine that might be above the END instruction in the main program that reverts to instruction #1 again, the END instruction found inside the subroutine only marks the END of the subroutine and not the end of the program, this is depicted in the image below.



When saving a PLC program to a disk file, the last active END instruction is searched including any subroutine called above the main program’s END instruction.

Only if the STOP WHEN FINDING FIRST VALID INSTRUCTION check box is disabled all 400 instructions, even if they contain nothing, will be saved to file. See section 7.9 Save and Restore your PLC Ladder Logic Program to and from Disk in your PC, for more information.



CANADA - Main Office
25 Interchange Way
Vaughan, ON L4K 5W3
Tel: (888) 660-4655
(905) 660-4655
Fax: (905) 660-4113

U.S.A.
4575 Witmer Industrial Estates
Niagara Falls, NY 14305
Tel: (888) 660-4655
(905) 660-4655
Fax: (905) 660-4113

TECHNICAL SUPPORT
North America
Tel: (888) Mircom5
(888) 647-2665
International
Tel: (905) 647-2665

© MGC 2017
Printed in Canada
Subject to change without prior notice
www.mircomgroup.com