

On/Off actuator

CT430900

# Programming manual



[www.besknx.com](http://www.besknx.com)

# Index

<b>1</b>	<b>DESCRIPCIÓN GENERAL.....</b>	<b>3</b>
<b>2</b>	<b>TECHNICAL INFORMATION.....</b>	<b>4</b>
<b>3</b>	<b>PROGRAMMING.....</b>	<b>5</b>
3.1	APPLICATION PROGRAM INFORMATION.....	5
3.2	INDIVIDUAL ADDRESS ASSIGNMENT .....	5
3.3	TYPE OF DEVICE.....	6
3.4	OUTPUTS OBJECTS.....	8
3.4.1	<i>Binary outputs table.....</i>	<i>8</i>
3.4.2	<i>Binary outputs description .....</i>	<i>9</i>
3.4.3	<i>Blind outputs table .....</i>	<i>10</i>
3.4.4	<i>Blind outputs description .....</i>	<i>11</i>
3.4.5	<i>Fan-coil outputs table .....</i>	<i>12</i>
3.4.6	<i>Fan-coil outputs description.....</i>	<i>13</i>
3.4.7	<i>Thermo-valve outputs table .....</i>	<i>13</i>
3.4.8	<i>Thermo-valve outputs description .....</i>	<i>15</i>
3.5	OUTPUTS PARAMETERS.....	16
3.5.1	<i>Binary outputs perameters .....</i>	<i>16</i>
3.5.2	<i>Blind outputs parameters.....</i>	<i>16</i>
3.5.3	<i>Fan-coil outputs parameters.....</i>	<i>17</i>
3.5.4	<i>Thermo-valve outputs parameters.....</i>	<i>17</i>
3.6	GENERAL PARAMETERS .....	18
3.6.1	<i>Inputs .....</i>	<i>18</i>
3.6.2	<i>Binary outputs.....</i>	<i>19</i>
3.6.3	<i>Blinds outputs.....</i>	<i>19</i>
3.6.4	<i>Scenes.....</i>	<i>19</i>
3.7	ARITHMETIC AND LOGIC UNIT.....	20
3.7.1	<i>Introduction.....</i>	<i>20</i>
3.7.2	<i>Type of blocks.....</i>	<i>21</i>
3.7.3	<i>Functions.....</i>	<i>22</i>
3.7.4	<i>Timers.....</i>	<i>23</i>
3.7.5	<i>Counters.....</i>	<i>26</i>
3.8	TEMPLATES .....	27
3.9	ADVANCED MODE.....	28
3.9.1	<i>Stairs light non retrigger .....</i>	<i>30</i>
3.9.2	<i>Stairs light retrigger .....</i>	<i>30</i>
3.9.3	<i>Intruder alarm .....</i>	<i>30</i>
3.9.4	<i>Delay on / delay off.....</i>	<i>31</i>
3.9.5	<i>Custom fan-coil .....</i>	<i>31</i>
3.10	PROGRAMMER MODE.....	33
3.10.1	<i>Scripts description.....</i>	<i>33</i>
3.10.2	<i>Editor .....</i>	<i>34</i>
3.10.3	<i>Programming language.....</i>	<i>35</i>
3.10.4	<i>Scripts de ejemplo.....</i>	<i>39</i>
3.11	PLUG-IN UPDATE .....	44
<b>4</b>	<b>INSTALLATION.....</b>	<b>45</b>

# 1 Descripción general

---

The Bes ref. CT430900 is an on/off actuator composed of 9 potential-free relay outputs (dry contact).

Its 9 outputs allow controlling 9 on / off electrical circuits or 4 blinds (2 outputs for one blind motor: up phase and down phase). Due to its high cut off capacity, this device is also recommended for capacitive loads, sockets, and electrical appliances control. The inputs can operate in different modes allowing to control binary outputs, blinds, fan-coil or thermo valves separately or simultaneously.

It incorporates an advanced Arithmetic and Logic Unit (UAL) that allows performing complex logic operation, timers programming, counters, etc. using internal results of operations or other external variables.

The cut off capacity of the relays is 16A @ 230Vac (potential free relay output). If necessary, insert a contactor to control higher power circuits.



General characteristics:

- 9 potential free relay outputs with a 16A @ 230Vac cut-off capacity.
- Each output can work independently or simultaneously in different modes (binary, blinds, fan-coils...).
- Easy and visual ALU (Arithmetic and Logic Unit) with timers, counters and any logic and arithmetic operation implementation.
- Advanced scripts execution that can be implemented by the programmer.

## 2 Technical information

---

Power supply	29V <sub>DC</sub> from KNX BUS
KNX current consumption	9mA from KNX BUS
Mounting	DIN rail
Size	6 DIN modules
Connections	BUS connection terminal KNX Screw terminals for outputs
Outputs	9 potential free relay output.
Outputs cut-off capacity	16A @ 230Vac
Environment temperature range	Operation: -10°C/55°C Storage: -30°C/60°C Transportation: -30°C/60°C
Regulation	According to the directives of electromagnetic compatibility and low voltage: EN 50090-2-2 / UNE-EN 61000-6-3:2007 / UNE-EN 61000-6-1:2007 / UNE-EN 61010-1.

## 3 Programming

---

### 3.1 Application program information

---

Application program: Ingenium (manufacturer) / Actuators (program name).

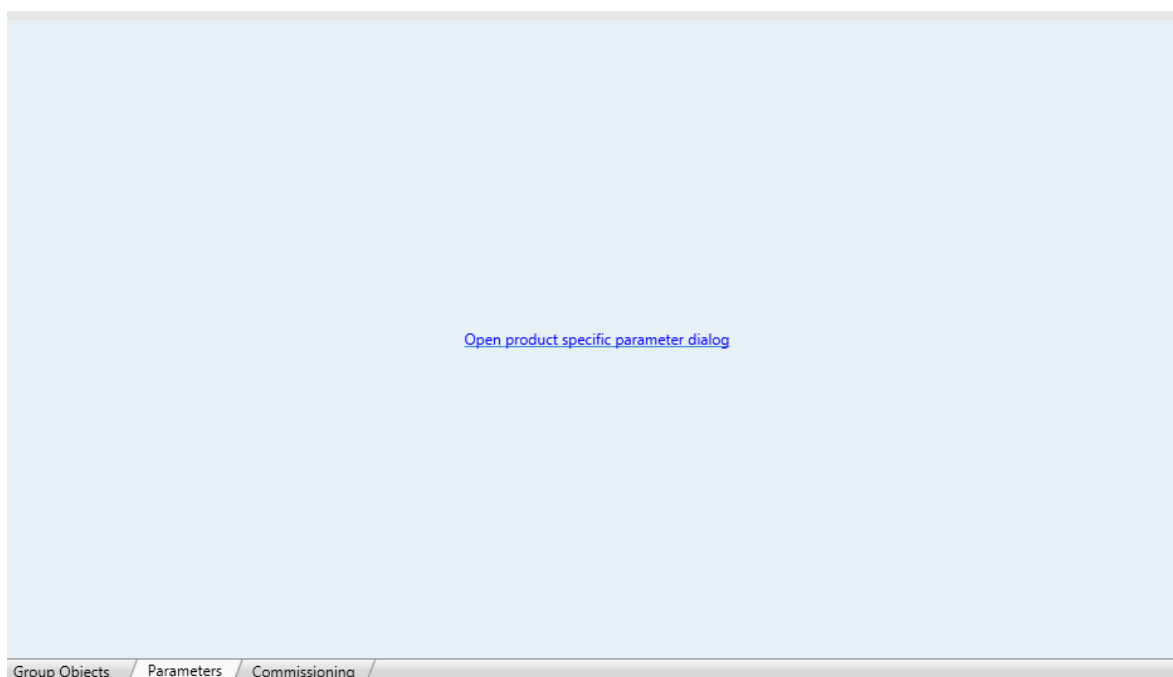
Catalogue version: v1.4

Maximum number of communication objects: 256.

Maximum number of assignments: 256.

ETS minimum required version: 4.1.8 (build 3614)

The parameters of the device are configured with a specific parameter dialog; do click on the link “open” from the parameters menu in the ETS to run it.



### 3.2 Individual address assignment

---

This actuator has a programming button for the KNX individual address assignment which is located on the front of the device.

A red led near the programming button lights up when it is pressed manually or if the device is set remotely to programming mode state.

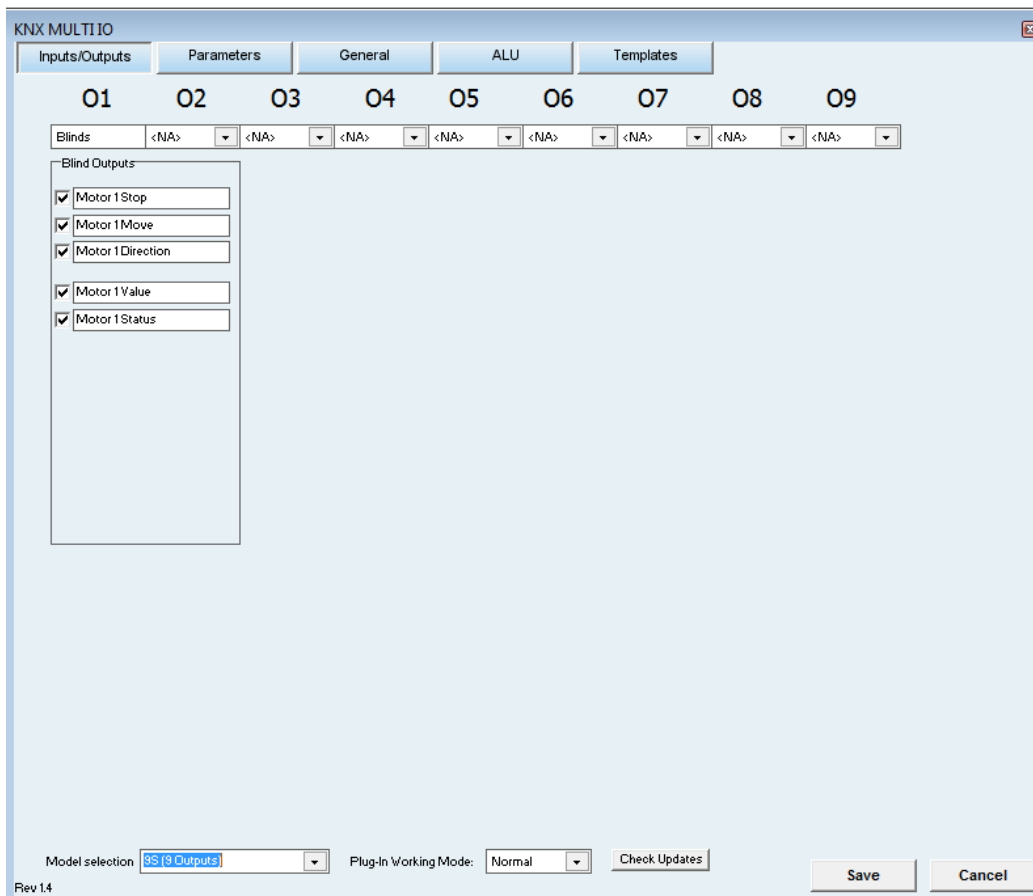
The led is automatically turned off if the ETS has assigned an individual address correctly or if the programming button is pressed again manually.

### 3.3 Type of device

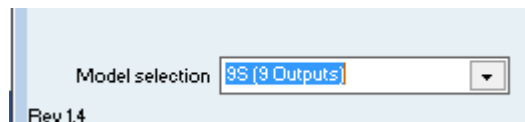
---

The parameters of the device are configured with a specific parameter dialog; do click on the link “open” from the parameters menu in the ETS to run it.

There are several tabs to configure different parameters depending on the type of the device selected; in this case the device that must be selected is the type “9 outputs”.

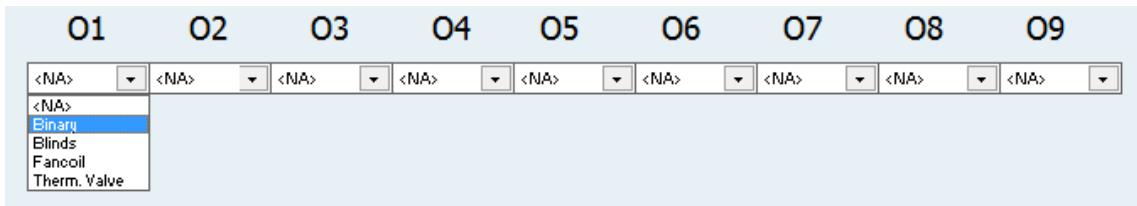


Use the selector at the bottom left corner of the main window to select the type of device to program



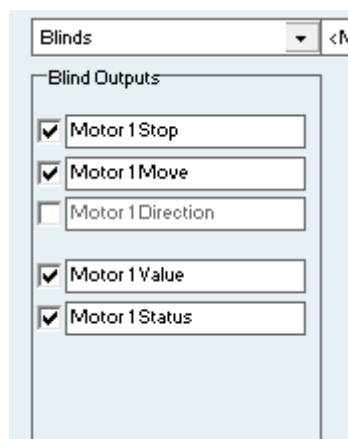
After that, a number of inputs and outputs appear depending on the model of the device selected. Each of these inputs and outputs can be configured to work in different modes independently and simultaneously.

Outputs can be programmed in binary, blinds, fan-coil or thermo-valve modes.



Depending on the type of output selected, more than one slot is occupied, for example, when selecting blinds, two outputs are reserved (odd output for the move up phase and even output for move down phase).

Once the type of output is selected, the communication objects associated to it appear below. The programmer can select whether to use or not any communication object by selecting them and also can edit the name that appears in the ETS for better recognition.



Default communication objects and names are explained next.

### 3.4 Outputs objects

#### 3.4.1 Binary outputs table

Objeto	Nombre   Función	Longitud	DPT	Flags				
				C	R	W	T	U
0	Output 1   Output 1 switching	1 bit	1.001	•		•		•
1	Output 1 status   Output 1 switching feedback	1 bit	1.001	•	•		•	
2	Output 2   Output 2 switching	1 bit	1.001	•		•		•
3	Output 2 status   Output 2 switching feedback	1 bit	1.001	•	•		•	
4	Output 3   Output 3 switching	1 bit	1.001	•		•		•
5	Output 3 status   Output 3 switching feedback	1 bit	1.001	•	•		•	
6	Output 4   Output 4 switching	1 bit	1.001	•		•		•
7	Output 4 status   Output 4 switching feedback	1 bit	1.001	•	•		•	
8	Output 5   Output 5 switching	1 bit	1.001	•		•		•
9	Output 5 status   Output 5 switching feedback	1 bit	1.001	•	•		•	
10	Output 6   Output 6 switching	1 bit	1.001	•		•		•
11	Output 6 status   Output 6 switching feedback	1 bit	1.001	•	•		•	
12	Output 7   Output 7 switching	1 bit	1.001	•		•		•
13	Output 7 status   Output 7 switching feedback	1 bit	1.001	•	•		•	
14	Output 8   Output 8 switching	1 bit	1.001	•		•		•
15	Output 8 status   Output 8 switching feedback	1 bit	1.001	•	•		•	
16	Output 9   Output 9 switching	1 bit	1.001	•		•		•
17	Output 9 status   Output 9 switching feedback	1 bit	1.001	•	•		•	
18	Output 1 bit script   Script recall	1 bit	1.001	•		•	•	•
19	Output 2 bit script   Script recall	1 bit	1.001	•		•	•	•
20	Output 3 bit script   Script recall	1 bit	1.001	•		•	•	•
21	Output 4 bit script   Script recall	1 bit	1.001	•		•	•	•
22	Output 5 bit script   Script recall	1 bit	1.001	•		•	•	•
23	Output 6 bit script   Script recall	1 bit	1.001	•		•	•	•
24	Output 7 bit script   Script recall	1 bit	1.001	•		•	•	•



25	Output 8 bit script   Script recall	1 bit	1.001	•	•	•	•
26	Output 9 bit script   Script recall	1 bit	1.001	•	•	•	•
128	Output 1 1-byte script   Script recall	1 byte	5.010	•	•	•	•
129	Output 2 1-byte script   Script recall	1 byte	5.010	•	•	•	•
130	Output 3 1-byte script   Script recall	1 byte	5.010	•	•	•	•
131	Output 4 1-byte script   Script recall	1 byte	5.010	•	•	•	•
132	Output 5 1-byte script   Script recall	1 byte	5.010	•	•	•	•
133	Output 6 1-byte script   Script recall	1 byte	5.010	•	•	•	•
134	Output 7 1-byte script   Script recall	1 byte	5.010	•	•	•	•
135	Output 8 1-byte script   Script recall	1 byte	5.010	•	•	•	•
136	Output 4 1-byte script   Script recall	1 byte	5.010	•	•	•	•
239	Output 1 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
240	Output 2 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
241	Output 3 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
242	Output 4 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
243	Output 5 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
244	Output 6 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
245	Output 7 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
246	Output 8 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
247	Output 9 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•

### 3.4.2 Binary outputs description

Name	Object X: Output X   Output X switching
Function	1-bit communication object for switching on and off an output.
Description	<p>When a “1” is received through this object the output is switched. When a “0” is received through this object the output is switched off.</p> <p>This is the normally open behaviour that depends on the parameter “mode”.The normally close behaviour is the opposite.</p> <p>By default, the status of an output is memorized when there is a power supply failure. If it is necessary to define a specific status, use a “power on script”.</p>

Name	Object X: Output X   Output X switching feedback
Function	1-bit communication object for feedback signalling of state of the output.
Description	When the output is off and receives a switch on telegram a “1” is sent through this object. When the output is on and receives a switch off telegram “0” is sent through this object.
Name	Object X: Output X   Output X script recall
Function	1-bit/1-byte/2-bytes communication objects for executing scripts.
Description	Advanced script programming.

### 3.4.3 Blind outputs table

Object	Name   Function	Length	DPT	Flags				
				C	R	W	T	U
0	Motor 1 move   Blind 1 move 0=up;1=down	1 bit	1.001	•		•		•
1	Motor 1 stop   Blind 1 stop 0=1=stop	1 bit	1.001	•		•		•
2	Motor 1 direction   current direction feedback	1 bit	1.001	•	•		•	
128	Motor 1 value   Blind 1 direct positioning	1 byte	5.001	•		•		•
129	Motor 1 status   Blind 1 position feedback	1 byte	5.001	•	•		•	
3	Motor 2 move   Blind 2 move 0=up;1=down	1 bit	1.001	•		•		•
4	Motor 2 stop   Blind 2 stop 0=1=stop	1 bit	1.001	•		•		•
5	Motor 2 direction   current direction feedback	1 bit	1.001	•	•		•	
130	Motor 2 value   Blind 2 direct positioning	1 byte	5.001	•		•		•
131	Motor 2 status   Blind 2 position feedback	1 byte	5.001	•	•		•	
6	Motor 3 move   Blind 3 move 0=up;1=down	1 bit	1.001	•		•		•
7	Motor 3 stop   Blind 3 stop 0=1=stop	1 bit	1.001	•		•		•
8	Motor 3 direction   current direction feedback	1 bit	1.001	•	•		•	
132	Motor 3 value   Blind 3 direct positioning	1 byte	5.001	•		•		•
133	Motor 3 status   Blind 3 position feedback	1 byte	5.001	•	•		•	
9	Motor 4 move   Blind 4 move 0=up;1=down	1 bit	1.001	•		•		•
10	Motor 4 stop   Blind 4 stop 0=1=stop	1 bit	1.001	•		•		•
11	Motor 4 direction   current direction feedback	1 bit	1.001	•	•		•	
134	Motor 4 value   Blind 4 direct positioning	1 byte	5.001	•		•		•
135	Motor 4 status   Blind 4 position feedback	1 byte	5.001	•	•		•	

12	Motor 1 bit script   Script recall	1 bit	1.001	•	•	•	•
13	Motor 2 bit script   Script recall	1 bit	1.001	•	•	•	•
14	Motor 3 bit script   Script recall	1 bit	1.001	•	•	•	•
15	Motor 4 bit script   Script recall	1 bit	1.001	•	•	•	•
136	Motor 1 1-byte script   Script recall	1 byte	5.010	•	•	•	•
137	Motor 2 1-byte script   Script recall	1 byte	5.010	•	•	•	•
138	Motor 3 1-byte script   Script recall	1 byte	5.010	•	•	•	•
139	Motor 4 1-byte script   Script recall	1 byte	5.010	•	•	•	•
239	Motor 1 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
240	Motor 2 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
241	Motor 3 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•
242	Motor 4 2-bytes script   Script recall	2 bytes	7.001	•	•	•	•

### 3.4.4 Blind outputs description

<b>Name</b>	<b>Object X: MotorX move   Blind X move (0=up;1=down)</b>
Function	1-bit communication object for moving up or down the blind.
Description	When a "1" is received through this object the blind motor moves up. When a "0" is received through this object the blind motor moves down.  Odd outputs (Z1, Z3, Z5 and Z7) must be connected to the up phase of the motor. Even outputs (Z2, Z4, Z6 and Z8) must be connected to the down phase of the motor. This order cannot be altered.
<b>Name</b>	<b>Object X: Motor X stop   Blind X stop (0=1=stop)</b>
Function	1-bit communication object for stop the blind movement.
Description	When any value is received through this object the blind motor stops moving.
<b>Name</b>	<b>Object X: Motor X direction   current direction feedback</b>
Function	1-bit communication object for feedback signalling of the current moving direction.
Description	When the blind motor starts moving up a telegram with value "0" is sent through this object. When the blind motor starts moving down a telegram with value "1" is sent through this object.  If the object is read while the motor is not moving a telegram with "0" is answered.

<b>Name</b>	<b>Object X: Motor X value   Blind X direct positioning</b>
Function	1-byte communication object for direct positioning of the blind.
Description	When a value is sent to this object the blind moves to the received position being 0 = completely close and 255 = completely open.
<b>Name</b>	<b>Object X: Motor X status   Blind X position feedback</b>
Function	1-byte communication object for feedback signalling of the position of the blind.
Description	When the blind motor stops the current position is sent through this object as feedback being 0 = completely close and 255 = completely open.  By default, the position of the blind is sent every second while it is moving. If the parameter "blind periodic notification" is deactivated, the position feedback is only sent when the motor stops.
<b>Name</b>	<b>Object X: Output X   Output X script recall</b>
Function	1-bit/1-byte/2-bytes communication objects for executing scripts.
Description	Advanced script programming.

### 3.4.5 Fan-coil outputs table

Object	Name   Function	Length	DPT	Flags				
				C	R	W	T	U
0	Fan 1 speed 1   Fan-Coil speed 1 setting	1 bit	1.001	●		●		●
1	Fan 1 speed 1 status   Fan-Coil speed 1 feedback	1 bit	1.001	●	●		●	
2	Fan 1 speed 2   Fan-Coil speed 2 setting	1 bit	1.001	●		●		●
3	Fan 1 speed 2 status   Fan-Coil speed 2 feedback	1 bit	1.001	●	●		●	
4	Fan 1 speed 3   Fan-Coil speed 3 setting	1 bit	1.001	●		●		●
5	Fan 1 speed 3 status   Fan-Coil speed 3 feedback	1 bit	1.001	●	●		●	
128	Fan 1 mode   Fan-Coil mode setting	1 byte	5.001	●		●		●
129	Fan 1 mode status   Fan-Coil mode feedback	1 byte	5.001	●	●		●	
6	Fan 1 bit script   Script recall	1 bit	1.001	●		●	●	●
130	Fan 1 byte script   Script recall	1 byte	5.010	●		●	●	●
239	Fan 1 2-bytes script   Script recall	2 bytes	7.001	●		●	●	●

### 3.4.6 Fan-coil outputs description

<b>Name</b>	<b>Object X: Fan X speed X   Fan-Coil speed X setting</b>
Function	1-bit communication object for switch the fan-coil to the corresponding speed.
Description	When a "1" is received through this object the fan-coil speed changes to the corresponding one. The other speeds are deactivated and a "0" is sent to the other speed objects for feedback.  The speeds of the fan-coil must be connected to the outputs as following: Z1=speed 1, Z2=speed 2 and Z3=speed 3. If it is necessary to change this configuration use a "custom fan-coil".
<b>Name</b>	<b>Object X: Fan X speed X status   Fan-Coil speed X feedback</b>
Function	1-bit communication object for feedback signalling of the current speed.
Description	When a speed is selected the status is sent through this object. A telegram with value "1" is sent through the object of the speed selected and also "0" is sent through the other speeds objects.
<b>Name</b>	<b>Object X: Fan X mode   Fan-Coil mode setting</b>
Function	1-byte communication object for direct speed selection.
Description	When a value is received through this object the fan-coil control compares it to the threshold levels configured and activates the corresponding speed.
<b>Name</b>	<b>Object X: Fan X mode status   Fan-Coil mode feedback</b>
Function	1-byte communication object for feedback signalling of the status.
Description	The current fan-coil speed value is sent through this object for feedback signalling with every change.
<b>Name</b>	<b>Object X: Fan X   Fan X script recall</b>
Function	1-bit/1-byte/2-bytes communication objects for executing scripts.
Description	Advanced script programming.

### 3.4.7 Thermo-valve outputs table

Object	Name   Function	Length	DPT	Flags				
				C	R	W	T	U
128	Valve 1 pwm  Valve 1 pwm control	1 byte	5.010	•	•	•	•	•
0	Valve 1 status   Valve 1 switching feedback	1 bit	1.001	•	•		•	
129	Valve 2 pwm  Valve 2 pwm control	1 byte	5.010	•	•	•	•	•
1	Valve 2 status   Valve 2 switching feedback	1 bit	1.001	•	•		•	
130	Valve 3 pwm  Valve 3 pwm control	1 byte	5.010	•	•	•	•	•
2	Valve 3 status   Valve 3 switching feedback	1 bit	1.001	•	•		•	
131	Valve 4 pwm  Valve 4 pwm control	1 byte	5.010	•	•	•	•	•

3	Valve 4 status   Valve 4 switching feedback	1 bit	1.001	•	•		•	
132	Valve 5 pwm  Valve 5 pwm control	1 byte	5.010	•	•	•	•	•
4	Valve 5 status   Valve 5 switching feedback	1 bit	1.001	•	•		•	
133	Valve 6 pwm  Valve 6 pwm control	1 byte	5.010	•	•	•	•	•
5	Valve 6 status   Valve 6 switching feedback	1 bit	1.001	•	•		•	
134	Valve 7 pwm  Valve 7 pwm control	1 byte	5.010	•	•	•	•	•
6	Valve 7 status   Valve 7 switching feedback	1 bit	1.001	•	•		•	
135	Valve 8 pwm  Valve 8 pwm control	1 byte	5.010	•	•	•	•	•
7	Valve 8 status   Valve 8 switching feedback	1 bit	1.001	•	•		•	
136	Valve 9 pwm  Valve 9 pwm control	1 byte	5.010	•	•	•	•	•
8	Valve 9 status   Valve 9 switching feedback	1 bit	1.001	•	•		•	
9	Valve 1 bit script   Script recall	1 bit	1.001	•		•	•	•
10	Valve 2 bit script   Script recall	1 bit	1.001	•		•	•	•
11	Valve 3 bit script   Script recall	1 bit	1.001	•		•	•	•
12	Valve 4 bit script   Script recall	1 bit	1.001	•		•	•	•
13	Valve 5 bit script   Script recall	1 bit	1.001	•		•	•	•
14	Valve 6 bit script   Script recall	1 bit	1.001	•		•	•	•
15	Valve 7 bit script   Script recall	1 bit	1.001	•		•	•	•
16	Valve 8 bit script   Script recall	1 bit	1.001	•		•	•	•
17	Valve 9 bit script   Script recall	1 bit	1.001	•		•	•	•
137	Valve 1 1-byte script   Script recall	1 byte	5.010	•		•	•	•
138	Valve 2 1-byte script   Script recall	1 byte	5.010	•		•	•	•
139	Valve 3 1-byte script   Script recall	1 byte	5.010	•		•	•	•
140	Valve 4 1-byte script   Script recall	1 byte	5.010	•		•	•	•
141	Valve 5 1-byte script   Script recall	1 byte	5.010	•		•	•	•
142	Valve 6 1-byte script   Script recall	1 byte	5.010	•		•	•	•
143	Valve 7 1-byte script   Script recall	1 byte	5.010	•		•	•	•
144	Valve 8 1-byte script   Script recall	1 byte	5.010	•		•	•	•
145	Valve 9 1-byte script   Script recall	1 byte	5.010	•		•	•	•
239	Valve 1 2-bytes script   Script recall	2 bytes	7.001	•		•	•	•

240	Valve 2 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
241	Valve 3 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
242	Valve 4 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
243	Valve 5 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
244	Valve 6 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
245	Valve 7 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
246	Valve 8 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●
247	Valve 9 2-bytes script   Script recall	2 bytes	7.001	●	●	●	●

- = Default configuration
- = Optional

### 3.4.8 Thermo-valve outputs description

---

<b>Name</b>	<b>Object X: Valve X pwm   Valve X pwm control</b>
Function	1-byte communication object for setting the duty cycle of the thermo-valve pwm output.
Description	The duty cycle of the pwm signal that controls the thermo-valve output is written by sending a value to this object.
<b>Name</b>	<b>Object X: Valve X status   Valve X switching feedback</b>
Function	1-byte communication object for switching feedback signalling.
Description	The thermo-valve output status is sent automatically through this communication object with every change.
<b>Name</b>	<b>Object X: Valve X   Valve X script recall</b>
Function	1-bit/1-byte/2-bytes communication objects for executing scripts.
Description	Advanced script programming.

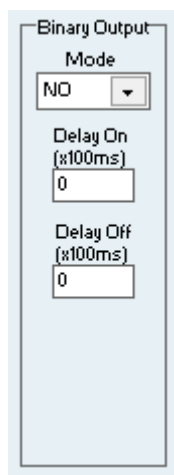
## 3.5 Outputs parameters

---

### 3.5.1 Binary outputs parameters

---

When an output is configured as an individual binary output the following parameters can be configured:



Binary Output  
Mode  
NO  
Delay On (x100ms)  
0  
Delay Off (x100ms)  
0

**Mode:** Normally open or normally closed. In normally open mode the output relay is controlled with the standard logic: 1 = close, 0 = open. In normally closed mode the output relay is controlled with the inverse logic: 1 = open, 0 = close.

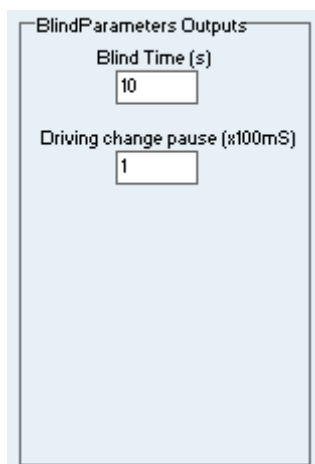
**Delay on (x100ms):** Time factor (base in 100 milliseconds) that the device waits before switching on the output (open or close contact depending on the mode).

**Delay off (x100ms):** Time factor (base in 100 milliseconds) that the device waits before switching off the output (open or close contact depending on the mode).

### 3.5.2 Blind outputs parameters

---

When outputs are configured as blind outputs the following parameters can be configured:



BlindParameters Outputs  
Blind Time (s)  
10  
Driving change pause (x100mS)  
1



**Blind time (s):** In this parameter it must be configured the time measured in seconds that the blind takes to raise up completely.

**Driving change pause:** This parameter is a factor (base in 100 milliseconds) for a dead time that the device waits before changing the direction of the blind while it is moving.

### 3.5.3 Fan-coil outputs parameters

---

When outputs are configured as fan-coil outputs the following parameters can be configured:

FancoilParameters Output	
Threshold Level1:	5
Threshold Level2:	85
Threshold Level3:	192
Output switch time (x100ms)	1

The received value through the fan-coil control communication object <<Fan X mode [1 byte]>> it is compared to these threshold levels by the device.

**Threshold level 1:** (from 0 to 255) if the fan-coil control value is lower than this threshold level the outputs of the fan-coil are switched off. If the control value is higher the Output O1 is switched on.

**Threshold level 2:** (from 0 to 255) if the fan-coil control value is lower than this threshold level the Output O1 is switched on. If the control value is higher the Output O1 is switched off and the Output 2 is switched on.

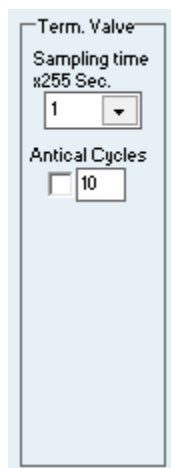
**Threshold level 3:** (from 0 to 255) if the fan-coil control value is lower than this threshold level the Output O2 is switched on. If the control value is higher the Output O2 is switched off and the Output 3 is switched on.

**Output switch time:** This parameter is a factor (base in 10 milliseconds) for a dead time that the device waits before changing from one speed to another while it is operating.

### 3.5.4 Thermo-valve outputs parameters

---

When outputs are configured as thermo valves outputs the following parameters can be configured:

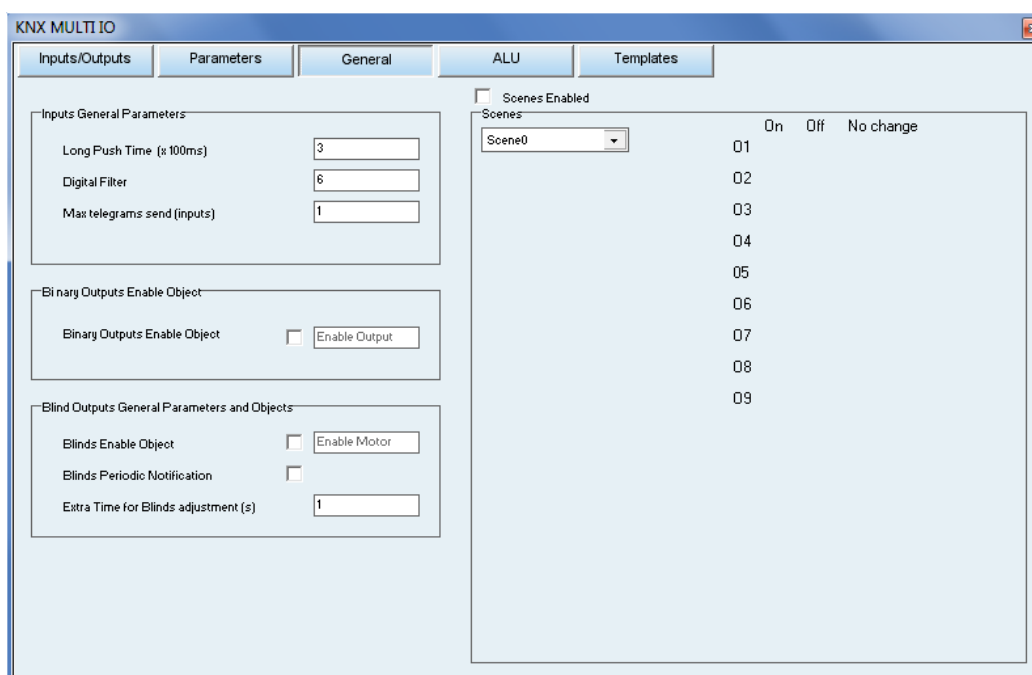


**Sampling time:** Time factor (base in 255 seconds) for the PWM signal generated by the output.

**Anti-lime cycles:** Activation or deactivation of the anti-lime function. When this function is activated, the device automatically closes the output for 5 seconds, according to the sampling time multiplied by the defined number of cycles.

### 3.6 General parameters

Here we can configure some general parameters of the device:



#### 3.6.1 Inputs

Not used in Ref. CT430900.

### 3.6.2 Binary outputs

---

**Binary outputs enable object:** Select if use or not an extra communication object that allows to enable or disable the control of binary outputs from knx bus commands (inputs are not enabled or disabled).

Polarity: 1 = control enabled / 0 = control disabled.

### 3.6.3 Blinds outputs

---

**Blinds outputs enable object:** Select if use or not an extra communication object that allows to enable or disable the control of blinds from knx bus commands (inputs are not enabled or disabled).

Polarity: 1 = control enabled / 0 = control disabled.

**Blinds periodic notification:** Activates or deactivates the periodic status notification of the blinds position while moving (time base 1 second).

**Extra time for blind adjustment:** Defines an additional time in seconds for complete blind position adjustment when it gets the upper or lower limit. The corresponding output remains closed an extra time measured in seconds.

### 3.6.4 Scenes

---

**Scenes enabled:** Activates or deactivates the communication object for scenes recall.

First select the scene value on the left, (this is the value that the corresponding communication object must receive to be executed) and then select the reaction of each output: on, off or no change (the final status of the output depends on the mode, NO or NC).

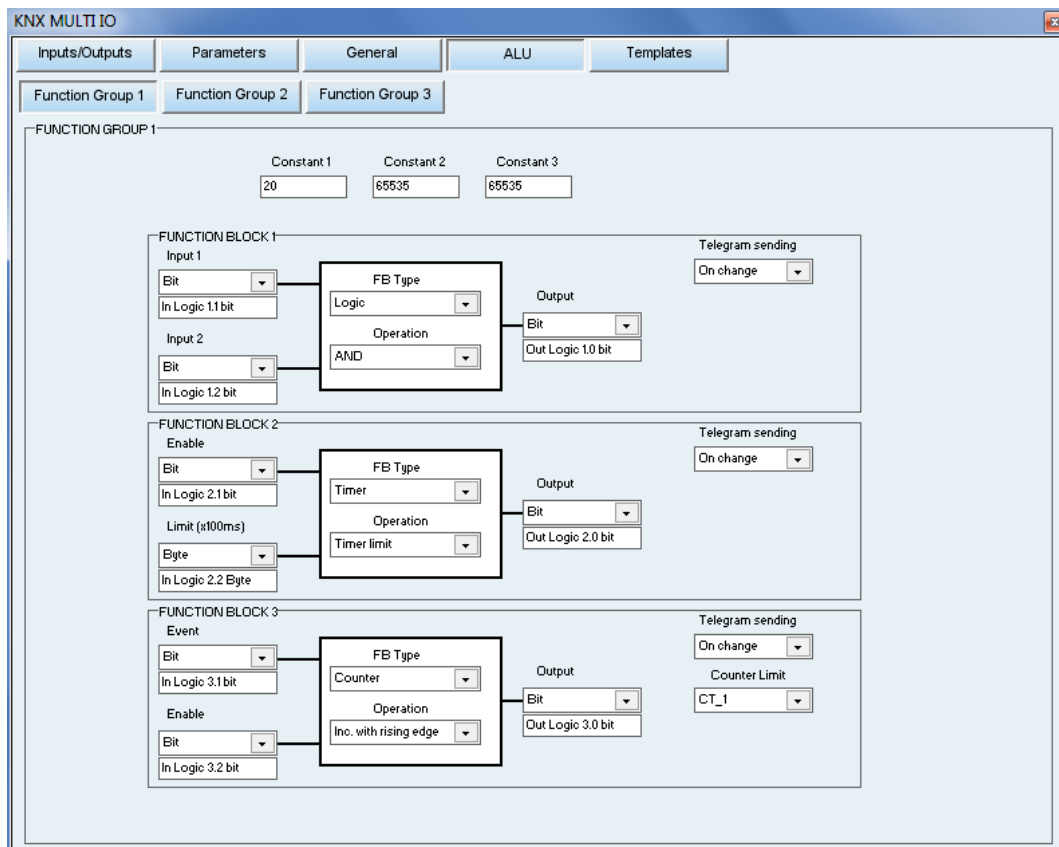
Scenes	On	Off	No change
Scene0	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

## 3.7 Arithmetic and logic unit

### 3.7.1 Introduction

The device incorporates an advanced Arithmetic and Logic Unit (UAL) that allows to do complex logic operation, timers programming, counters, etc. using internal or external variables with an intuitive and visual programming.

The ALU is composed of 3 function groups, and 3 function blocks in each group. A function block has 2 inputs and 1 output, with 3 communication objects each according to a data type that can be selected (bit, byte, 2 bytes).



The nomenclature used with the communication objects is the following:

**[in/out] [logic X.Y] [size]**

In/Out: Indicates if the object is an input or an output of the function block.

Logic X.Y: Being X the number of the function block (from 1 to 9) and Y the number of input (1,2) or output (0).

Size: Indicates the DPT or size of the communication object.

Below each input or output there is a case that indicates the name of the communication object. This name can also be edited by the programmer.

A function block can operate with values received from the bus through the input objects or with fixed parameters too, that are configured in the Constants cases above (there are 3 for each function group).

Constant 1	Constant 2	Constant 3
65535	65535	65535

Another possibility is the use of intermediate variables, named VARx. The advantage of this type of variables is to avoid sending telegrams to the bus when it is necessary to link the result of an operation from a block with the input of another one in the same application. There are up to 4 intermediate variables available for each function group. An intermediate variable of a function group cannot be used in other function group.

An important thing of this application is that a single device can host multiple UAL applications with up to 9 functions blocks so it is possible to perform complex operations linking outputs and inputs with intermediate variables or communication objects.

The output of any function block can have an active or passive sending. Use the following selector to decide if the result of the operation is sent with every change (active) or it is only read (passive).

Telegram sending

On change ▼

To program an operation, first select the type of block with the FB Type selector and the operation that it will implement. Then, select the type/size of inputs (constants, intermediate variables or communication objects) and finally the type/size of the output.

### 3.7.2 Type of blocks

---

It is possible to configure each of the three blocks with one of these three possible categories:

Logic and arithmetic functions:

- Logic type: AND, OR, XOR, NAND, NOR and NXOR operations
- Comparisons: equal, different, greater, greater or equal, less and less or equal.
- Arithmetic operations: addition, subtraction, multiplication and division.

Timers

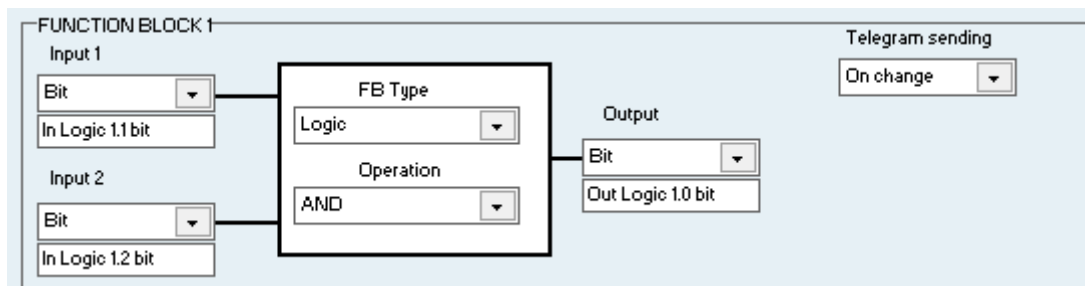
- Timers with limit
- PWM
- Cyclic timer

Counters

- Rising edge
- Falling edge
- Event counter 1 or different from 0
- Event counter 0

### 3.7.3 Functions

Go to the function block type selector and set “Logic” and the “operation” option.



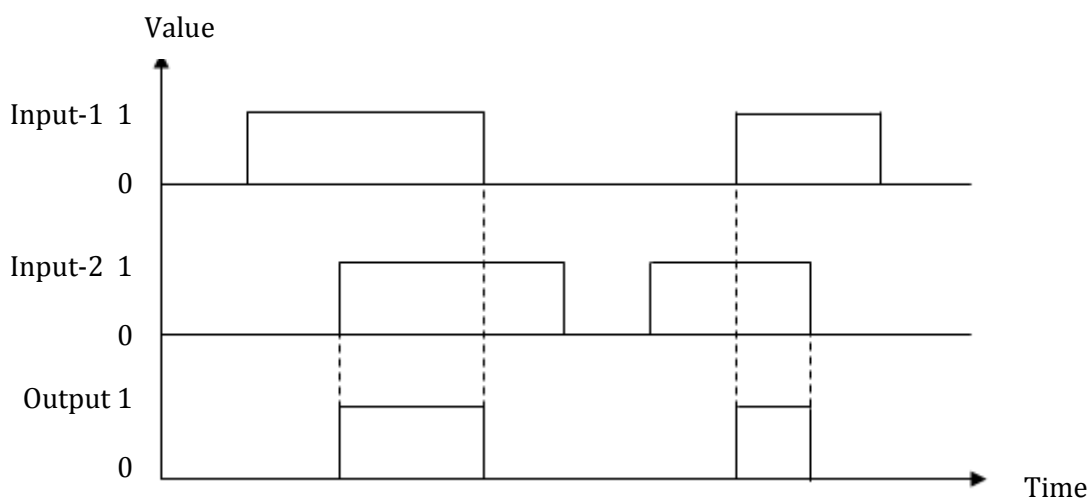
**Input 1 / Input 2** - Select where are the input values obtained from: they can be obtained from the bus by selecting the communication objects in the inputs lists (input 1 and input 2) according to data type needed (bit, byte or 2 bytes), and also a constant (CTEx) or an intermediate variable (VARx).

**Output** - In the same way as with inputs, in the outputs list can be selected the communication object according to the data type needed (bit, byte or 2 bytes), and also the result can be saved in the available intermediate variables (VARx). The output value is updated when any input object receive a telegram.

**Operation** - The option “Operation” allows to set the block behavior. In the case of comparison operations (=,!=,>,>=,<,<=), the value sent will be 1 if it is true and 0 if it is false. It is important to take into account that the output data type will be truncated if it has a smaller size that the input data type.

When the function block is programmed, any time an input receive a telegram the operation will be executed, sending the result to the output communication object or not according to the telegram sending selector (on change / only read).

Example: The previous picture shows how to program an AND logic operation with 2 bit inputs. When any value is received through the input communication objects (In Logic 1.1 bit and In Logic 1.2 bit) the function block calculates the operation and the result is sent through the output object (Out Logic 1.0 bit).

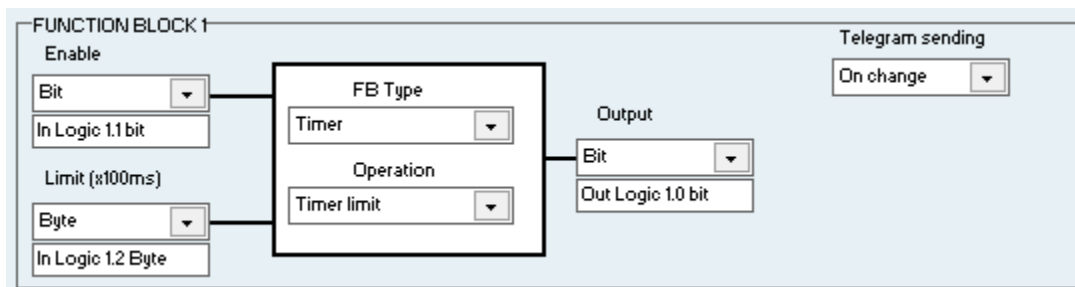


### 3.7.4 Timers

There are three types of timers that can be selected in the list “Type of timer”. The behavior of each type is explained next.

#### TIMER LIMIT

It sends a telegram to the bus or an intermediate variable when a limit set value is exceeded.

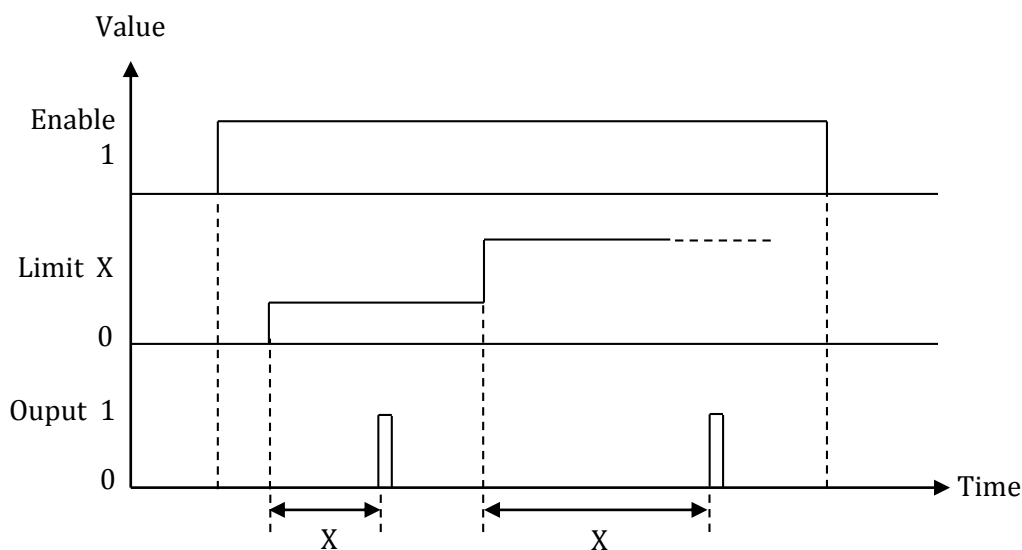


**Enable** - Enables/disables the timer function with 0/1 value. When a 1 is sent to the Enable input the timer is activated and the countdown starts with any value different from 0 received in its limit input. When a 0 is sent to the enable input, the timer is deactivated and the output sends a 0.

**Limit** - The timer starts the countdown with any value different from 0 in its limit value, that can be taken from a constant or from a bus communication object. When it finishes, the output telegram is sent and it does not start again until the reception of a new activation telegram. The limit value can have a size of 1 or 2 bytes when it is sent through the bus. It can be also obtained from a constant. The timer also stops the countdown when it receives a value of 0 and it restarts with any different value.

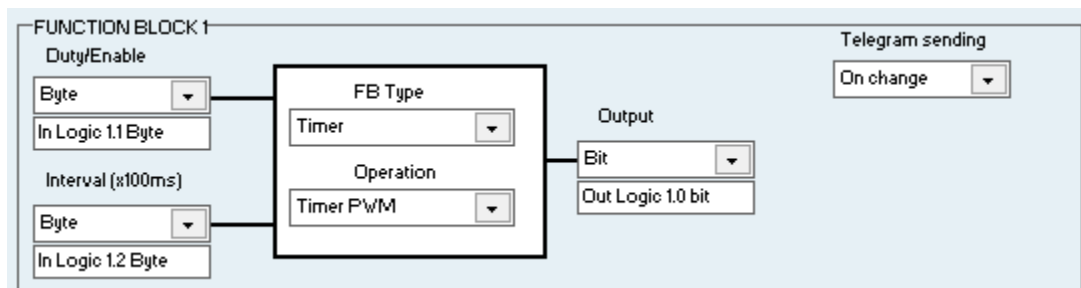
The value of the timer must be written in a 100ms scale, for example, the value must be 30 for an interval of 3 seconds.

**Output** - The output can have a size of 1 bit, 1 byte or 2 bytes, being the value sent 1 in any of the cases. It is also possible to send the event to an intermediate variable, that can be the input of another function block.



## TIMER PWM

This type of timer sends 1 and 0 telegrams alternatively during the interval programmed cyclically. The time between 1 and 0 depends on the duty value (from 1 to 10). A duty value of 0 deactivates the timer.

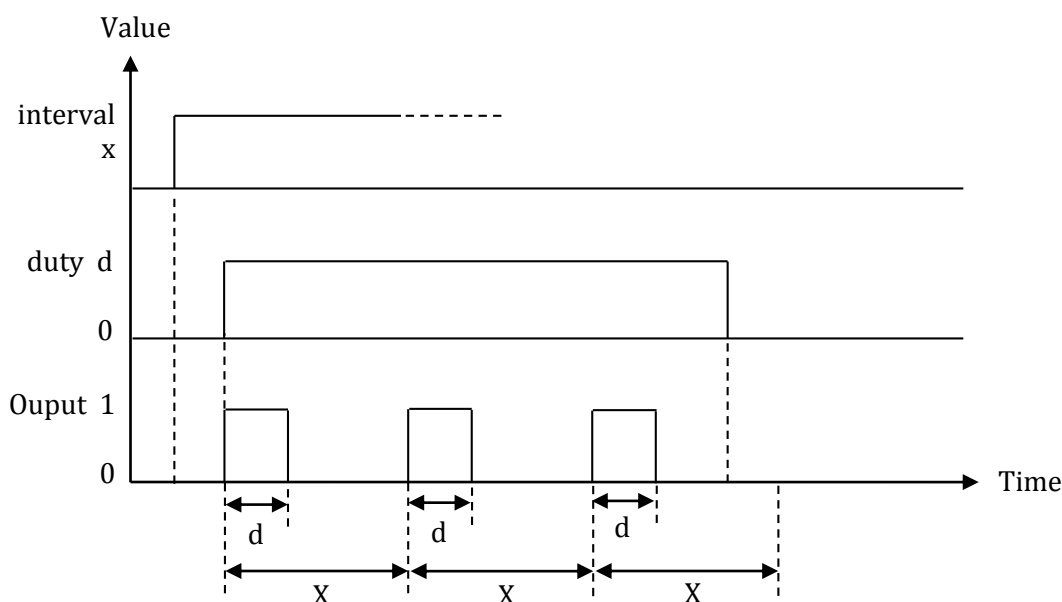


**Duty/Enable** - The duty cycle admits values of 1 byte or 2 bytes from 1 to 10. It allows to deactivate the timer when the value is 0. When the value is 10 it means a duty of 100% so the output is always activated.

**Interval** - It is the signal period of time. This value can be obtained from the bus (1 byte or 2 bytes), from a constant value or an intermediate variable.

The value of the timer must be written in a 100ms scale, for example, the value must be 30 for an interval of 3 seconds.

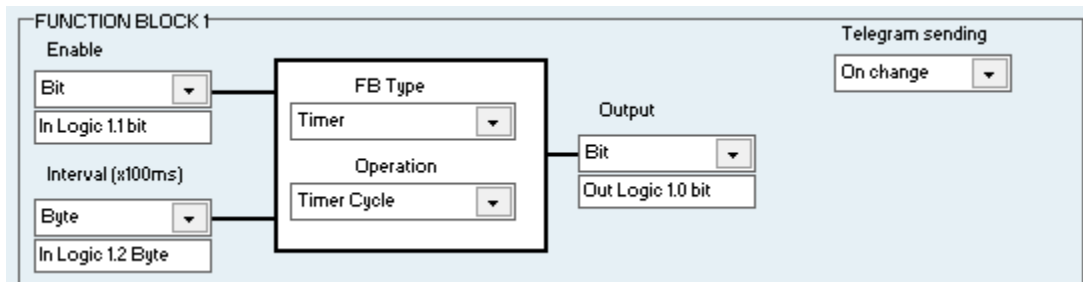
**Output** - The output can have a size of 1 bit, 1 byte or 2 bytes, being the value sent 1 in any of the cases. It is also possible to send the event to an intermediate variable which can be the input of another function block.





## TIMER CYCLE

This type of timer sends a telegram with value 1 cyclically when the time defined as interval is exceeded.

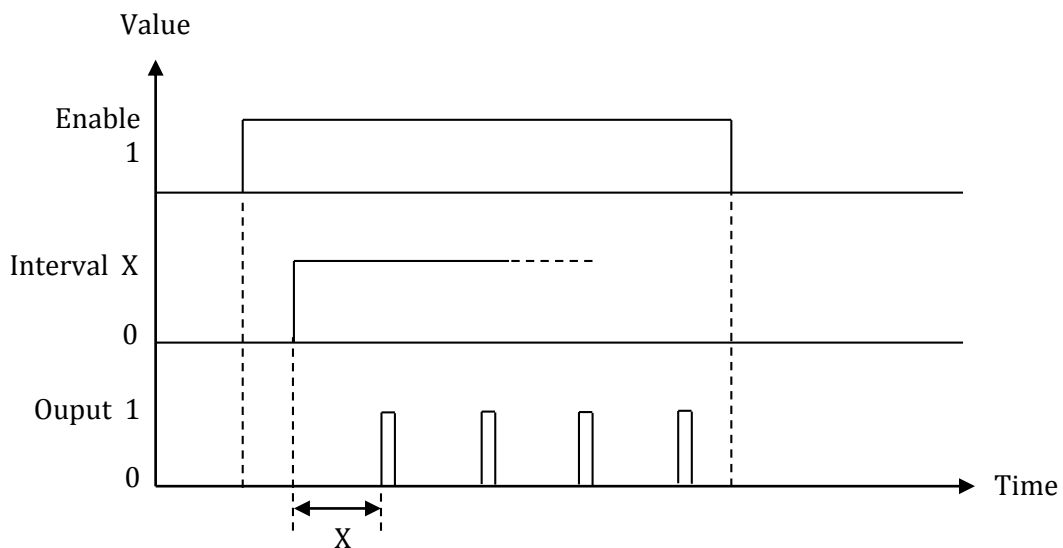


**Enable** - Allows to activate or deactivate the timer. This input can be associated to a bus communication object (1 bit, 1 byte or 2 bytes) or an intermediate variable.

**Interval** - It is the signal period of time. This value can be obtained from the bus (1 byte or 2 bytes), from a constant value or an intermediate variable.

The value of the timer must be written in a 100ms scale, for example, the value must be 30 for an interval of 3 seconds.

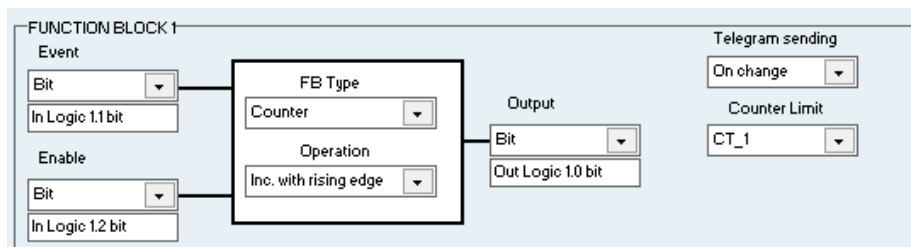
**Output** - The output can have a size of 1 bit, 1 byte or 2 bytes, being the value sent 1 in any of the cases. It is also possible to send the event to an intermediate variable which can be the input of another function block.



### 3.7.5 Counters

---

There are four types of counters that can be selected in the list “Type of counter” and are explained next.



**Counter Limit** – Allows to set the number of events over which the counter sends the finish telegram. This value can be obtained from a communication object (1 byte or 2 bytes), from a constant value or an intermediate variable. A value of 0 deactivates the counter.

**Event** – This is the input of the counter (indicates where are the events count from). It can be a communication object of 1 bit, 1 byte or 2 bytes or an intermediate variable.

**Output** – When the counter value exceeds the limit, a telegram with value 1 is sent. It can be a communication object of 1 bit, 1 byte or 2 bytes or an intermediate variable.

**Enable** – Overload in the input 2. The counter can be enabled or disabled with the bit in the input 2. At the same time, the limit value can be set with communication objects of 1 byte or 2 bytes.

#### RISING EDGE

---

When the input detects a rising edge (changes from 0 to 1) the counter increases its internal value. When the counter reaches the limit it sends a telegram to the bus with value 1. Then it returns to the initial disabled state.

#### FALLING EDGE

---

When the input detects a falling edge (changes from 1 or different from 0 to a value of 0) the counter increases its internal value. When the counter reaches the limit it sends a telegram to the bus with value 1. Then it returns to the initial disabled state.

#### INCREMENT WITH ONE

---

When the input receives a telegram with a value of 1 the counter increases its internal value. When the counter reaches the limit it sends a telegram to the bus with value 1. Then it returns to the initial disabled state.

#### INCREMENT WITH ZERO

---

When the input receives a telegram with a value of 0 the counter increases its internal value. When the counter reaches the limit it sends a telegram to the bus with value 1. Then it returns to the initial disabled state.

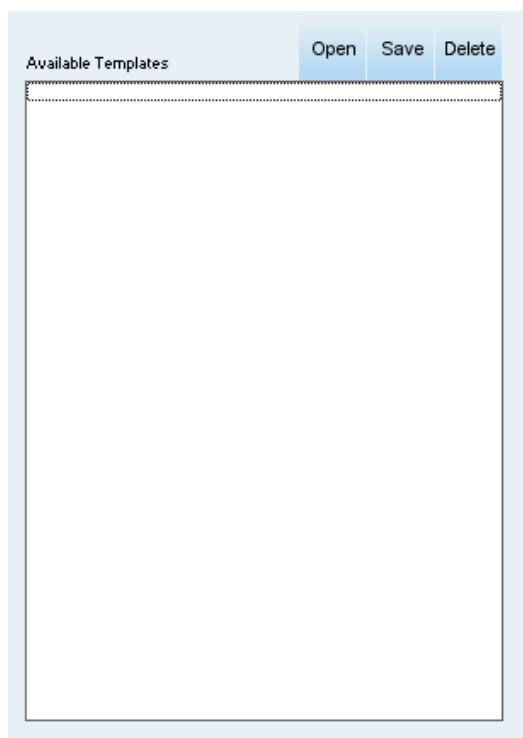
## 3.8 Templates

---

The 9S-K plug-in application allows saving or loading any configuration done by the programmer.

The default “copy/paste” or “transfer parameters and flags” functionalities included in the ETS4 are not supported when the device is programmed by an external application, but the 9S-K plug-in allows saving the whole parameterization of any device in order to use it with any other device or even other project.

Do click on the templates window to see every template included in the database.



By doing click on the save button every parameter configured, name edited, communication object and every information of the current programming is saved in a file.

Do click on any template and press open to load it or press delete to erase it from the database.

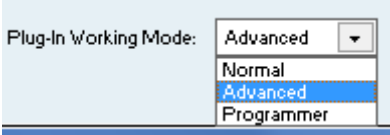


*The “copy/paste” and “transfer parameters and flags” functionalities included in the ETS4 are not supported when the device is programmed by an external application. To copy any configuration from a 9S-K to other device or project use a template.*

### 3.9 Advanced mode

---

This actuator incorporates the possibility of implementing complex operations or advanced functions allowing the programmer to run a simple mood or scenario or to develop his own advanced program execution that can be run from the bus, sending telegrams or receiving parameters, counting, operating, etc.



The advanced programming modes are only accessible by selecting the "advanced mode" or "programmer mode" from the combo box at the bottom of the plug-in window.



The advanced mode shows an extra window in the main menu above where the programmer can create advanced functions using a wizard. The functions created will be then available to be executed through communication objects as explained next.

**Advanced Mode Assistant:**

Advanced Function to Implement:

Launching Communication Object DPT:

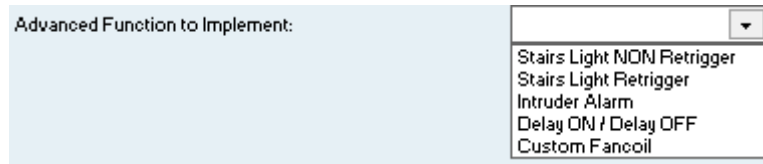
Communication Object Desired Name:

---

**Advanced Functions Implemented on Device:**

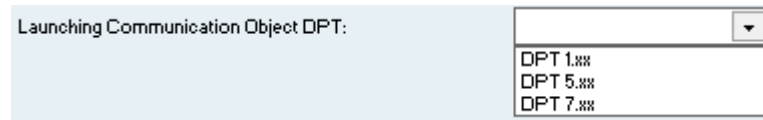
Object Name	Function	Output/Group Addr.	Parameter

To create an advanced function, first use the selector to decide which function will be implemented from the available functions:



A screenshot of a software interface showing a dropdown menu. The label 'Advanced Function to Implement:' is on the left. The dropdown list contains the following options: 'Stairs Light NON Retrigger', 'Stairs Light Retrigger', 'Intruder Alarm', 'Delay ON / Delay OFF', and 'Custom Fancoil'.

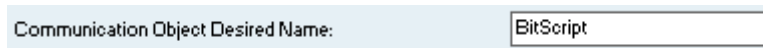
Then select the dpt of the function:



A screenshot of a software interface showing a dropdown menu. The label 'Launching Communication Object DPT:' is on the left. The dropdown list contains the following options: 'DPT 1.xxx', 'DPT 5.xxx', and 'DPT 7.xxx'.

This is the size of the communication object (dpt 1.\*, dpt 5.\*, dpt 7.\*) associated to the advanced function. A value sent to the advanced function communication object will execute it.

After that, the programmer can edit the name of the object for better recognition in the ETS.

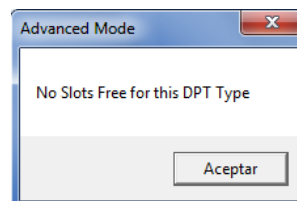


A screenshot of a software interface showing a text input field. The label 'Communication Object Desired Name:' is on the left. The text 'BitScript' is entered in the field.

When the type of advanced function, the size of its object and the name is configured, do click on the “advanced function parameters” button to continue with the settings of the specific parameters of the function.

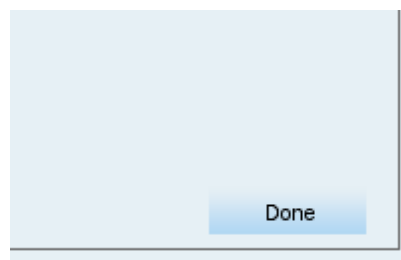


There is a limitation in the number of advanced functions that can be implemented in the device and also depends on the number of outputs configured. There are a 1-bit, a 1-byte and a 2-bytes functions available per output.



A “no slots free” message is shown if it is necessary to configure more outputs or if the number of advanced functions has been exceeded.

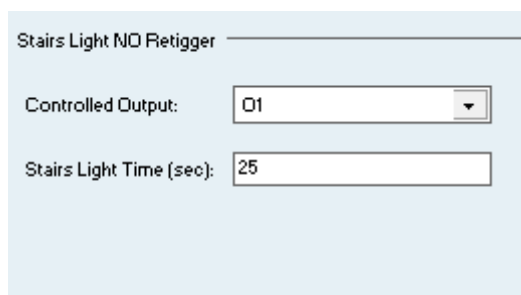
When the specific parameters of the advanced function are configured, do click on the button done.



### 3.9.1 Stairs light non retrigger

---

This advanced function allows programming a stair light by deciding which output it is controlled and the time that it is activated.



Stairs Light NO Retigger

Controlled Output:

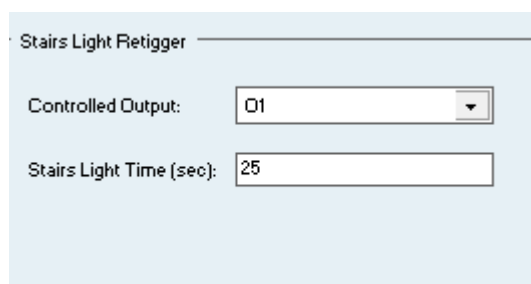
Stairs Light Time (sec):

By sending a “1” to the communication object associated to the function the output is activated. Then, the output is automatically deactivated after the time configured. If another “1” is received before this time the countdown do not start again.

### 3.9.2 Stairs light retrigger

---

This advanced function allows programming a stair light by deciding which output it is controlled and the time that it is activated.



Stairs Light Retigger

Controlled Output:

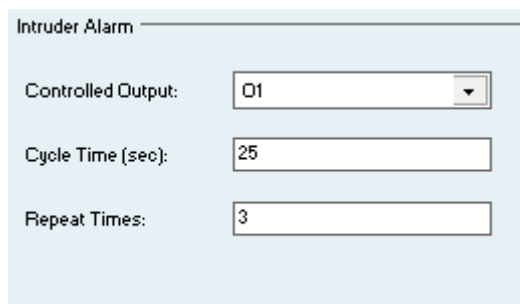
Stairs Light Time (sec):

By sending a “1” to the communication object associated to the function the output is activated. Then, the output is automatically deactivated after the time configured. If another “1” is received before this time the countdown starts again before deactivating it.

### 3.9.3 Intruder alarm

---

This advanced function allows programming a basic intruder alarm by deciding which output it is controlled, the time that it is activated and how many cycles are done.



Intruder Alarm

Controlled Output: O1

Cycle Time (sec): 25

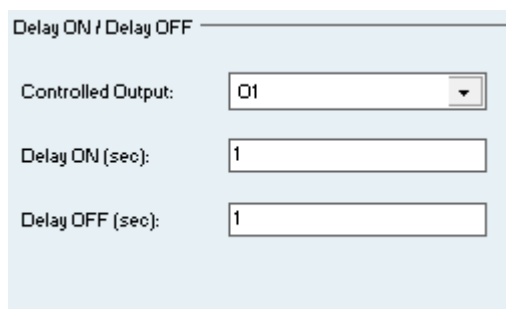
Repeat Times: 3

This function is focused to control a siren or any other output in an alarm system. By sending a “1” to the communication object associated to the function the output is activated the defined cycle time. Then, the output is automatically deactivated during another cycle time and the process is repeated the number of times configured.

### 3.9.4 Delay on / delay off

---

This advanced function allows programming on and off delays by deciding which output it is controlled and the times of each delay.



Delay ON / Delay OFF

Controlled Output: O1

Delay ON (sec): 1

Delay OFF (sec): 1

By sending a “1” to the communication object associated to the function the output is activated after the time configured as “delay on”. Then, when a “0” is received the output is deactivated after the time configured as “delay off”.

### 3.9.5 Custom fan-coil

---

This advanced function allows programming a customizable fan-coil control by deciding which output corresponds to each speed and defining the working thresholds.

Custom Fancoil with Hysteresis

V1 Output:  ▾

V2 Output:  ▾

V3 Output:  ▾

Switch Time (x50mS):

Working Thresholds:

V1 Thresholds (V1 if LowerThr < VALUE < UpperThr)

Lower Thr.

Upper Thr.

V2 Thresholds (V2 if LowerThr < VALUE < UpperThr)

Lower Thr.

Upper Thr.

V3 Thresholds (V3 if VALUE > LowerThr)

Lower Thr.

By sending a 1-byte or 2-bytes values to the communication object associated to the function the device decides which output (speed) must be activated according to the configured thresholds.

The advanced functions that have been already programmed are shown in the list on the left. Any function can be deleted by selecting it and pressing the button “delete function” at the bottom of the list.

Advanced Functions Implemented on Device:

Object Name	Function	Output/Group Addr.	Parameter
Output_1_stair_light	Stairs_Light_Retrigger:	O1:	25_Seconds:
Output_6_siren:	Intruder_Alarm:	O6:	30Sec_3Times:
Output_2_delays:	Delay_ON_Delay_OFF:	O2:	2Sec_ON_25Sec_OF

Delete Function



## 3.10 Programmer mode

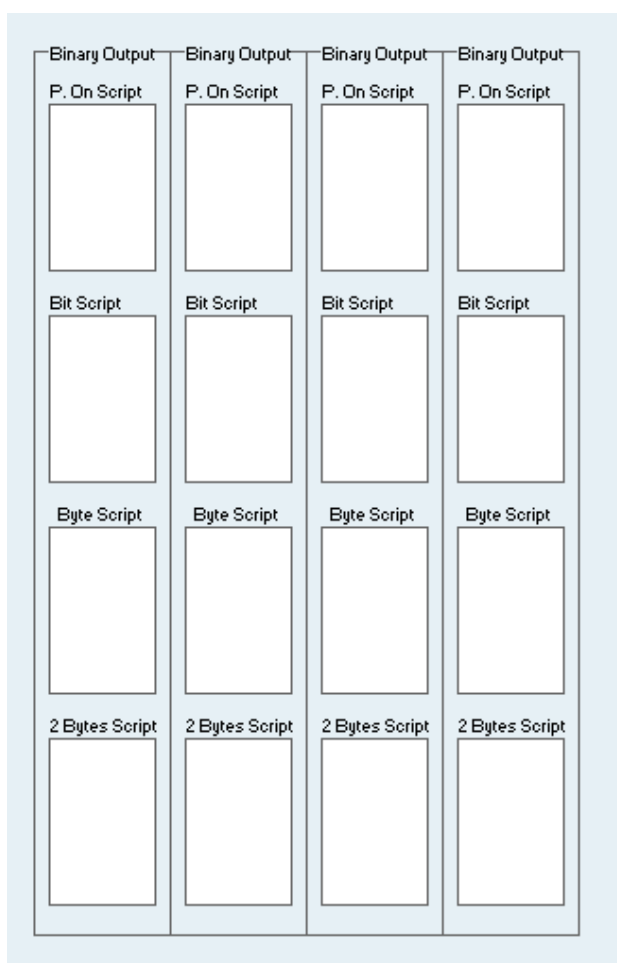
---

### 3.10.1 Scripts description

---

The actuator incorporates an advanced scripting programming method developed with its own programming language similar to other programming languages as C. The scripts allows the programmer to run a simple mood or scenario or to develop his own advanced program execution that can be run from the bus, sending telegrams or receiving parameters, counting, operating, etc.

There are 3 types of scripts according to the parameter that they can receive from the bus (through a group address associated): Bit scripts, byte scripts or 2bytes scripts. There is a fourth type of script that it is called power-on script and it is executed when the devices recovers the power supply.



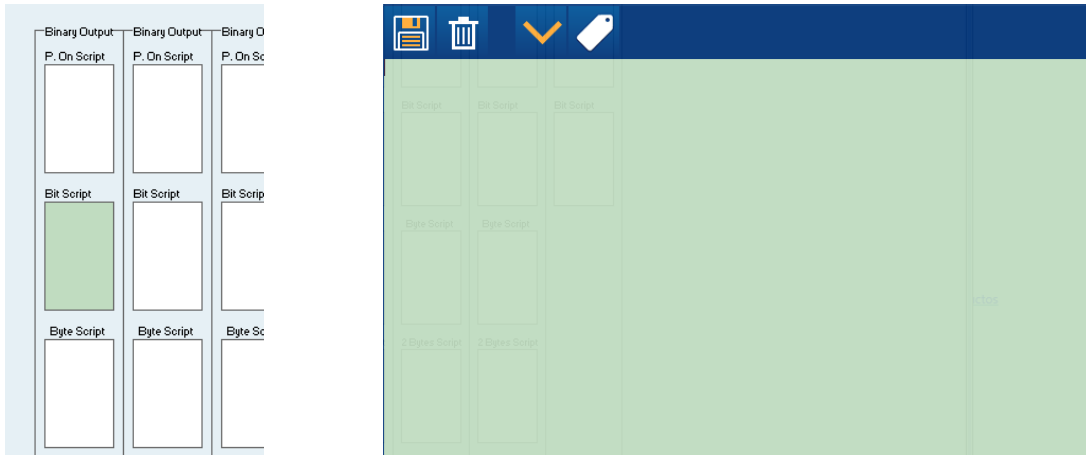
The scripts can be activated or deactivated from the inputs/outputs tab.



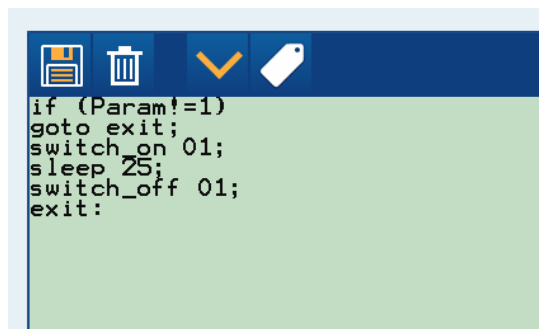
*Each script can have a maximum of 20Kb of size. When programming any script, a “script size out of bounds” message is shown if the maximum number of lines/commands is exceeded.*

### 3.10.2 Editor

When a script is selected, the editing window appears in front of the plug-in. The current working script is indicated in green.



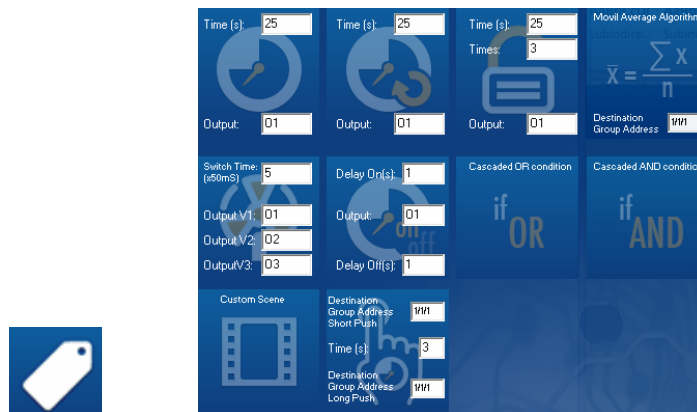
The programming of the script can be done manually according to the language that is explained next or can be done using the wizard. When the edition of the script has finished do click on the “save button” to save the current script or do click on the “bin” button to discard changes.



Use the “tick” button to check the correct spelling of the script. If no message is shown means that the script has been correctly written, if not, a message indicates the error.



The wizard includes some common examples of advanced functions that can be programmed into a script. Each of these functions have some parameters that can be configured with the wizard.



### 3.10.3 Programming language

---

The programming language used in the scripts is similar to other languages, for example, a semicolon indicates the end of every sentence. For better understanding each sentence can be written in a new line and we can use tabulations. There are several functions and reserved words that are explained next

#### PARÁMETROS RECIBIDOS POR EL BUS

---

##### **param**

It is the value received from the knx bus through the group address associated to the script. It can be a bit, a byte or 2bytes depending on the type of script.

It can be used in operations, functions and other commands, even between operators of different size, for example:

```
...  
var1=(param*2)+234;  
...
```

We can also assign to it a new value directly or as a result of an operation during the script execution, for example:

```
...  
var1=(param*2)+234;  
param=0;  
...
```

The parameter will hold the value received or assigned until the end of the script execution. When receiving a floating point 2bytes value, the real value taken by the param is the binary one. Floating point operations are not available.

#### INTERNAL VARIABLES

---

##### **varX**

There are up to 10 internal variables in each script that can be used in any operation, function and other commands, even between operators of different size. It is not necessary to declare or initialize them because with every script execution they take the value 0.

```
var1=(param*2)+234;  
var2=var2+var1;
```

In the same way as the parameter, we can assign to them a new value directly or as a result of an operation during the script execution. The variables hold the value assigned until the end of the script execution too.

The internal variables of a script can be used only in this script. If we need to send a variable from one script to another one it must be sent through the parameter.

## ARITHMETIC AND COMPARISON OPERATORS

---

There are two types of operators: arithmetic and comparison.

**Arithmetic:** = , + , - , \* , / (assignment, addition, subtraction, multiplication and division).

These arithmetic operators can be used in any sentence with variables, constant values and also the parameter. The result of an arithmetic operation can be assigned to the parameter or an internal variable by the operator “=”. Parenthesis can be used to group several operations as common arithmetic rules.

```
var1=5;  
var2=(param/2)+(var1*2);  
var3=var3+var2;
```

Notice that all arithmetic operations are done with integers, floating point operations are not possible.

**Comparison:** == , >= , <= , > , < , != (equal, major or equal, minor or equal, major, minor, different).

Comparison operators are used with the conditional function “if” as explained next.

## FUNCTIONS: GOTO AND IF


---

**Goto X;** Go to a line function

The “goto” function makes the program execution continue from a determined line that it is identified by a label “X” chosen by the programmer.

A label is defined with a combination of letters and/or numbers, but the first character must be a letter always, and the label finishes with “.”. For example:

```
...  
goto next;  
    var1=0;  
    ...  
    Var10=0;  
next:  
    var1=param+1;  
...
```



The program continues from the label and the sentences after the “goto” function are not executed


## If (A X B) Conditional function "if"

The conditional function "if" allows the programmer to execute or not one or more sentences of the script depending on the result of a comparison.

A and B are internal variables, constants or the parameter while X is a comparison operator.

If the comparison operation is true, the next line after the "if" is executed. If the comparison operation is false, the program jumps the sentence. For example:

```
If(param>100)
    var1=param;
If(param<=100)
    var1=0;
```



If the parameter it is not greater than 100  
the next assignation it is not executed


In the previous example, var1 will take the value of the parameter received only if it is greater than 100, if not var1 takes the value 0.

When the programmer needs to execute more than one sentence depending on a condition he can repeat the same "if" function, but it is better to use the opposite condition in combination with a "goto" function, for example:

```
If(param>100)
    var1=param;
If(param>100)
    var2=param*2;
If(param>100)
    var3=param*3;
If(param<=100)
    var1=0;
If(param<=100)
    var2=0;
If(param<=100)
    var3=0;
```

The previous script is the same than the following one using the opposite conditions:


```
If(param<=100)
    goto next;
    var1=param;
    var2=param*2;
    var3=param*3;
next;
If(param>100)
    goto next2;
    var1=0;
    var2=0;
    var3=0;
next2;
```



If the parameter it is not greater than 100  
the goto function is executed and the  
following sentences not

The programmer can use nested "if" functions to implement a composed condition. In the following example the script checks if the parameter value is between 5 and 10. If the condition is true the parameter is saved in the variable 1, if not the variable is 0.

```
If(param<5)
    If(param>10)
        var1=param;
var1=0;
```



If the parameter it is not lower than 5, the program jumps to the sentence var1=0.

When we use nested "if" functions, we can think that if the first "if" condition it is false, the following sentence it is not executed, so the second "if" condition it is not evaluated. It does not work like this.

In the previous example, the sentence that follows the first "if" function finishes in the first ",". This means that if the parameter it is not lower than 5 the program continues with var1=0. We can understand the script better if it is written like this:

```
If(param<5)
    If(param>10) var1=param;
var1=0;
```

Tabulations and carriage returns are not necessary; they are used for better understanding of the script.

## DIRECT COMMANDS

---

There are several direct commands that are executed with reserved words as explained next:

- Switch\_on X;** - Being X the name of the output from O1 to O16, it activates the corresponding relay.
- Switch\_off X;** - Being X the name of the output from O1 to O16, it deactivates the corresponding relay.
- Set X Y;** - Being X the name of the blind and Y the opening percentage.
- Sleep X;** - Being X the time in seconds from 0 to 65535. This command stops the execution of the script during the time configured.

## SENDING OF TELEGRAMS TO THE BUS

---

**Send\_telegram(X/X/X,dpt,value,command);**

- X/X/X - Is the group address form 0/0/1 to 31/7/255.
- dpt - Indicates the length of the data sent: dpt1, dpt5, dpt7 and dpt9.
- value - Is the value sent to the bus.
- command - Is the command of the telegram that can be: write or read.

```
...
send_telegram(1/1/10,dpt5,150,write);
send_telegram(1/2/15,dpt9,21.50,write);
...
```

### 3.10.4 Scripts de ejemplo

---

#### SIMPLE SWITCH OFF DELAY

---

**Description:** We need to switch on a light and automatically switch it off after a defined time. We will use a bit script that will be executed when receiving “1” or “0” through the associated group address. The following script is the easiest one that can be programmed.

**Bit Script:**

```
Switch_on OX;  
Sleep Y;  
Switch_off OX;
```

**Notes:** OX is the internal name of the output (from O1 to O16 depending on the type of actuator). If you need to switch on and off an output of any other device it must be used a “send\_telegram()” function. Y is the time measured in seconds that the light is on.

#### SWITCH OFF DELAY RETRIGGERABLE

---

**Description:** It is the same example as the previous one but now we need to retrigger the count with any “1” received through the associated group address.

**Bit Script:**

```
if(param==0)  
    goto exit;  
switch_on OX;  
wait:  
    if(param==1)  
        var1=0;  
    param=0;  
    if(var1>=Y)  
        goto exit;  
    var1=var1+1;  
    sleep 1;  
goto wait;  
exit:  
    switch_off OX;
```

**Notes:** OX is the internal name of the output (from O1 to O16 depending on the type of actuator). If you need to switch on and off an output of any other device it must be used a “send\_telegram()” function. Y is the time measured in seconds that the light is on. Tabulations are not necessary, they have been used for better understanding.

## SWITCH OFF DELAY WITH TIME AS PARAMETER

---

**Description:** It is the same example as above but now we send the time to count through the associated group address so we use a byte script.

### Byte script:

```
if(param==0)
    goto exit;
switch_on OX;
wait:
    if(var1>=param)
        goto exit;
    var1=var1+1;
    sleep 1;
goto wait;
exit:
    switch_off OX;
```

**Notes:** : OX is the internal name of the output (from O1 to O16 depending on the type of actuator). If you need to switch on and off an output of any other device it must be used a "send\_telegram()" function. The parameter received through the group address is saved in "param" and it is the time measured in seconds that the light is on. Tabulations are not necessary, they have been used for better understanding.

## SWITCH ON DELAY WITH TIME AS PARAMETER

---

**Description:** It is the same example as above but the delay is now for switching on.

### Byte script:

```
if(param==0)
    goto exit;
wait:
    if(var1>=param)
        goto exit;
    var1=var1+1;
    sleep 1;
goto wait;
exit:
    switch_on OX;
```

**Notes:** OX is the internal name of the output (from O1 to O16 depending on the type of actuator). If you need to switch on and off an output of any other device it must be used a "send\_telegram()" function. The parameter received through the group address is saved in "param" and it is the time measured in seconds that the light is off. Tabulations are not necessary, they have been used for better understanding.



## COUNTDOWN WITH TIME AS PARAMETER

---

**Description:** We will use a byte script to receive a byte value and start a countdown sending a telegram to the bus every second.

**Byte script:**

```
if(param==0)
    goto exit;
count:
    param=param-1;
    sleep 1;
    send_telegram(X/X/X,dpt7,param,write);
    if(param<=0)
        goto exit;
goto count;
exit:
```

**Notes:** X/X/X is the group address we want to send the countdown through. The parameter received through the group address is saved in “param” and it is the time measured in seconds for the countdown. Tabulations are not necessary, they have been used for better understanding.

## OR LOGIC WITH 3 VARIABLES

---

**Description:** We need to program an OR operation with 3 bits that we will activate with “1” or deactivate with “0” depending on the parameter received.

**Byte script:**

```
param=65535;
wait:
    if(param==65535)
        goto wait;
    if(param==0)
        var1=0;
    if(param==1)
        var1=1;
    if(param==2)
        var2=0;
    if(param==3)
        var2=1;
    if(param==4)
        var3=0;
    if(param==5)
        var3=1;
    var4=var1+var2+var3;
    send_telegram(X/X/X,dpt1,var4,write);
    param=65535;
goto wait;
```

**Notes:** X/X/X is the group address we want to send the result through. The parameter received through the group address is saved in "param" and it is processed. The result is formatted as a bit (dpt1) where any value different from "0" is "1". Tabulations are not necessary, they have been used for better understanding.

## ABSOLUTE VALUE SUBTRACTION

---

**Description:** We want to receive two parameters and calculate the subtraction in absolute value.

**Byte script:**

```
var1=param;
wait:
    if(param!=var1)
        goto wait;
var2=param;
if(var1>=var2)
    var3=var1-var2;
if(var1<var2)
    var3=var2-var1;
send_telegram(X/X/X,dpt7,var3,write);
```

**Notes:** : X/X/X is the group address we want to send the result through. The first parameter received through the group address is saved in "var1" and then we wait for the second parameter that we save in "var2". Tabulations are not necessary, they have been used for better understanding.

## "AND" LOGIC WITH MORE THAN ONE INSTRUCTION IN "IF" FUNCTION

---

**Description:** We want to activate or deactivate the internal outputs depending on the byte value received from 0 to 255. The first output is activated if the condition  $0 \leq \text{param} < 64$  is true, the second output if  $64 \leq \text{param} < 128$ , the third output if  $128 \leq \text{param} < 192$  and finally the fourth output if  $192 \leq \text{param} \leq 255$ . Also when we activate an output we deactivate the others. This means that we need to execute more than one instruction inside the "if" functions, so the trick for this is to write the opposite condition and use a "goto".

**Script:**

```
if(param<0)
    goto if1;
if(param>=64)
    goto if1;
    switch_off 02;
    switch_off 03;
    switch_off 04;
    switch_on 01;
if1:

if(param<64)
    goto if2;
if(param>=128)
```

```
        goto if2;
        switch_off O1;
        switch_off O3;
        switch_off O4;
        switch_on O2;
if2:

if(param<128)
    goto if3;
if(param>=192)
    goto if3;
    switch_off O1;
    switch_off O2;
    switch_off O4;
    switch_on O3;
if3:

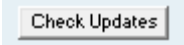
if(param<192)
    goto if4;
if(param>255)
    goto if4;
    switch_off O1;
    switch_off O2;
    switch_off O3;
    switch_on O4;
if4:
```

**Notes:** OX is the internal name of the output (from O1 to O16 depending on the type of actuator). If you need to switch on and off an output of any other device it must be used a "send\_telegram()" function. The parameter received through the group address is saved in "param" and it is processed. Tabulations are not necessary, they have been used for better understanding.

### 3.11 Plug-in update

---

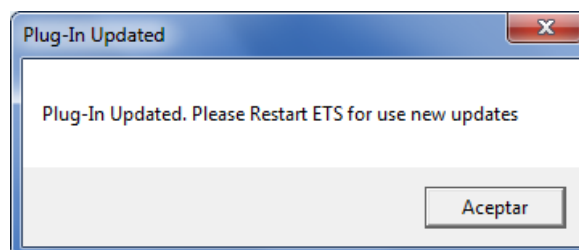
The programming application allows checking for updates through the internet and automatically



Check Updates

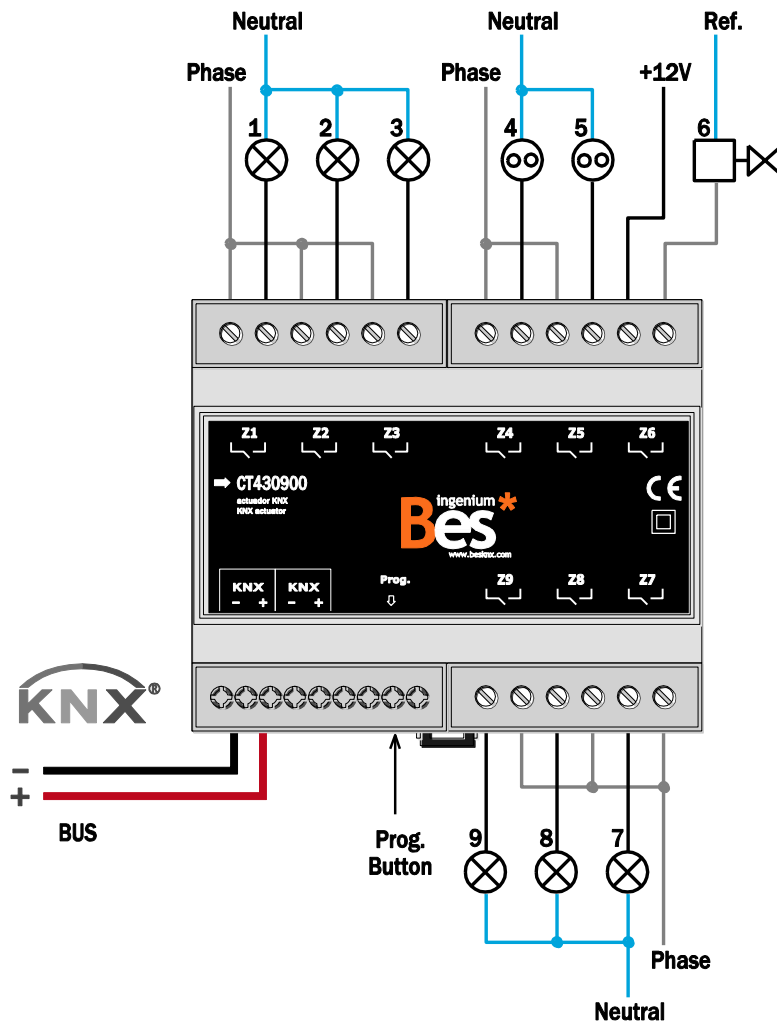
It is not necessary to re-import the device catalogue in the ETS4 or install any additional software, just do click on the button “check updates” at the bottom of the main window and the plugin will download and install the newest version from the Ingenium server if it is available.

Wait a minute while the application is downloaded. When it finishes the ETS4 must be closed in order to let the new plug-in be installed.



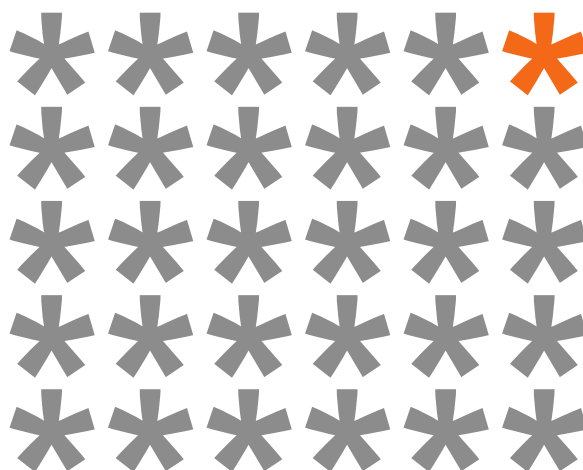
If there is not any new version available yet, the button “check updates” will show the message “no updates”.

## 4 Installation



Feed low voltage lines (BUS and inputs) in separate ducting to that of power (230V) and outputs to ensure there is enough insulation and avoid interferences.

Do not connect the main voltages (230V) or any other external voltages to any point of the BUS or inputs.



KNX products by **ingenium**



**Ingenium, Ingeniería y Domótica S.L.**

Parque Tecnológico de Asturias, Parcela 50

33428 Llanera, Asturias, Spain

T (+34) 985 757 195

tec@besknx.com

www.besknx.com

www.ingeniumsl.com

*Liability limitation: The present document is subject to changes or excepted errors. The contents are continuously checked to be according to the hardware and software but deviations cannot be completely excluded. Consequently any liability for this is not accepted. Please inform us of any suggestion. Every correction will be incorporated in new versions of this manual.*

Manual version: v1.1