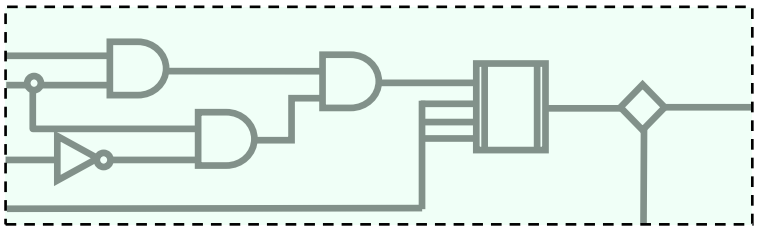
Zennio
Passion for KNX innovation



# Logical Functions (X10)

## 10x Logical-Mathematical Functions Module

USER MANUAL

User Manual Version: 1.c

www.zennio.com

## Contents

# DOCUMENT UPDATES

| Version | Changes | Page(s) |
|---|---|---|
| 1.c | General revision of texts and format. | - |
| 1.b | General revision of texts and format. | - |

# 1 INTRODUCTION

## 1.1 LOGICAL FUNCTIONS MODULE

A variety of Zennio devices (as the actuators of the ACTinBOX and MAXinBOX families, or the KES Energy Saver) incorporate a logical functions module, which makes them capable of performing **mathematical** and **binary logic** operations with data received from the KNX bus, as well as of sending the results through 1-bit, 1-byte or 2-byte specific communication objects provided for such purpose.

The operands of these functions may be of the following types:

- **Communication objects** received through the KNX bus.
- **Internal variables** containing partial results from previous operations.
- **Constant values**, defined by parameter in ETS.

Depending on how many independent functions can be configured, the incorporated module may be one of the following:

- **"X5" module**: up to five different and independent functions can be configured, each of which may consist itself in as many as four successive and interrelated operations.
- **"X10" module**: up to ten different and independent functions can be configured, each of which may consist itself in as many as four successive and interrelated operations.

Please refer to the user manual of the specific Zennio device in order to confirm the module (X5 or X10) it incorporates.

**Note**: *hereafter, this user manual will focus on the X10 module. For specific information about the X5 module, please refer to the corresponding user manual, available at the Zennio website*.

# 2 CONFIGURATION

## 2.1 GENERAL APPROACH

The X10 Logical Functions module permits enabling and configuring up to ten independent logical functions, the behaviour of which is typically divided into three stages:

- **Call**: the first step to make the function execute consists in *calling* it. With that aim, one or more communication objects can be configured, so that whenever they update their values from the KNX bus, they will automatically trigger the execution of the function.

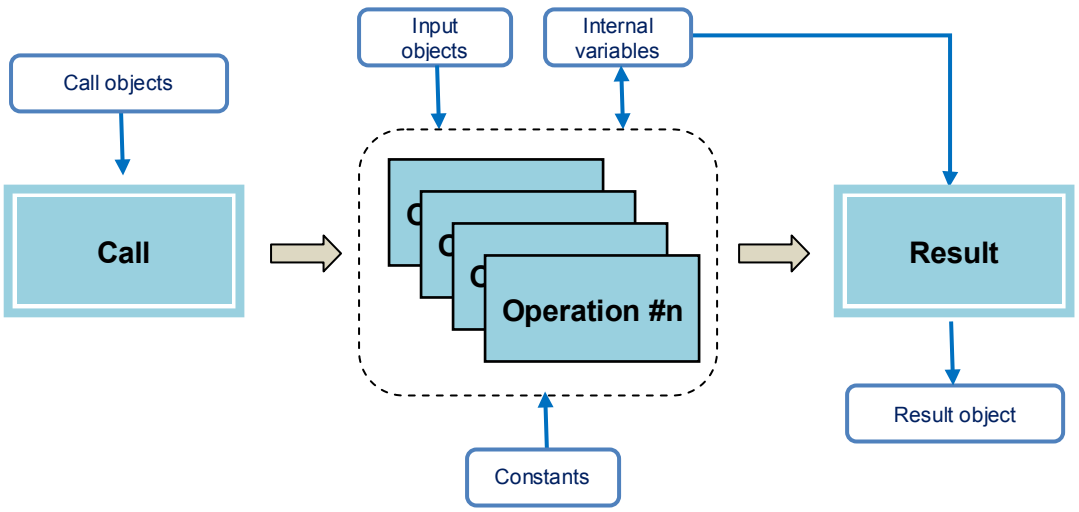
- **Operations**: triggering the function will itself initiate the execution of up to four mathematical or binary operations. The following needs to be configured for each of them:

  - **Operation type**: desired action (addition, subtraction, negation, etc.).

  - **Operands**: values to operate on. They can be input communication objects, internal variables containing the result of previous operations, or constants predefined in ETS.

  - **Result**: internal variable where to store the result of the operation.

- **Result**: it is necessary to set which internal variable contains the global result of the function, so its value will be sent through the corresponding communication object once the execution of all the operations ends.



## 2.2 CALL OBJECTS

For each function, the integrators will have at their disposal up to eight call objects (with a size of one bit, one byte or two bytes), each of which will trigger the function as soon as it receives a value from the bus. These objects do not necessarily need to be used as operators as well.

## 2.3 OPERATIONS

Each logical function consists in the execution of up to four consecutive operations. The available operations can be grouped as follows:

- **Logical**: ID, NOT, AND, OR, XOR, NAND, NOR and XNOR.
- **Arithmetical**: ID, add, subtract, multiplication, division, maximum and minimum.
- **Comparison**: higher, higher or equal, lower, lower or equal, equal, unequal.
- **Conversion**: *cast* operations to convert a certain operand from one size to another (e.g., to convert a 1-bit value into a 1-byte value).

The X10 module can operate in the following value ranges (for either communication objects, internal variables with intermediate results, or numerical constants defined by parameter in ETS):

- Binary values: **0** and **1**.
- Unsigned integers (one byte): **0 – 255**.
- Unsigned integers (two bytes): **0 – 65535**.
- Floating-point decimal values (two bytes): **0.00 – 120.00**.

For further information in relation to these operations, the size transformations and the truncation of values, please refer to *ANNEX I: Operations Reference*.

## 2.4 INPUT OBJECTS

Multiple specific objects can be enabled to use them with the logical functions:

- Up to 32 one-bit objects,
- Up to 16 one-byte objects,
- Up to 16 two-byte objects.

The value of the above objects may act, for example, as operands for the operations of the enabled functions.

## 2.5 INTERNAL VARIABLES

Additionally, the integrator will have the following at their disposal:

- 32 one-bit internal variables,
- 16 one-byte internal variables,
- 16 two-byte internal variables.

All of them may be used to temporarily store intermediate results, which themselves will be available as input values for later operations.

## 2.6 RESULT OBJECTS

Every logical function comes with a specific object (one-bit, one-byte or two-bytes size, depending on the parameterisation) through which the final value of a certain internal variable (which needs to be set by parameter) will be sent to the bus, as the result of the sequence of operations that make up the function.

The integrator has the chance to set whether this sending should happen every time the function is executed, or periodically, or only in case the function throws a result that differs from that of the previous execution. On the other hand, the results sent can be restricted, so the KNX bus is only notified when the result meets a certain restriction or range of values. Finally, it is also possible to configure in ETS a certain delay for the transmission of the result.

## 2.7 "GATE" OBJECTS

In addition to the functionality explained above, an option is provided to enable/disable each function independently in runtime, by writing a "0" or a "1" (configurable) to its particular *gate* object, which needs to be one of those already mentioned in 2.4. While disabled, the function will ignore any values received through the call objects, thus preventing the execution of the operations in any case.

# 3 ETS PARAMETERISATION

## 3.1 GENERAL SCREEN

The general screen of the X10 Logical Functions module contains the options shown in the figure below (please note that minor differences may be found from one device to another).

**Note**: *the general parameter tab of the logical functions module may not show in ETS by default – it may be necessary to enable it from the General parameter tab of the device itself. Please refer to the user manual of the actual device for more details*.

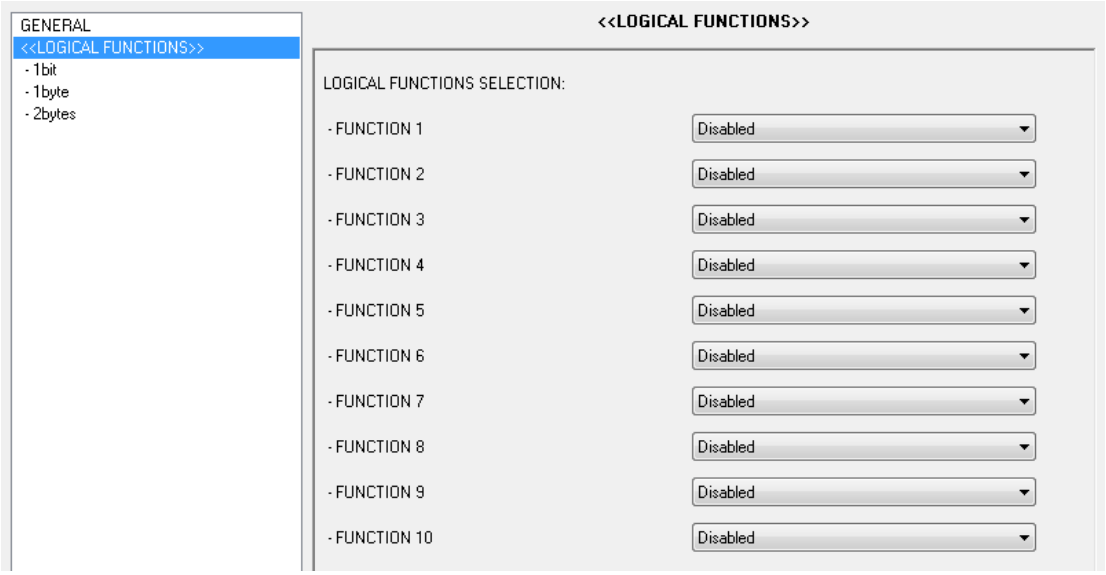



**Figure 1.** General Parameter Tab of the X10 Logical Functions Module.

As illustrated, none of the ten functions comes enabled by default. As they become enabled by the integrator, additional tabs will be included into the tab list on the left.

The next sections cover the purpose of every tab and of the parameters they contain.

## 3.2 1bit, 1byte, 2bytes

These three tabs, which do show in the tab list at any time, let the integrator enable (one by one) the input communication objects (with a size of one bit, one byte or two bytes) that may be required by the functions, being then possible to make use of them

as operands, call objects, etc. These objects are named according to the following pattern:

- "**[FL] (1 bit) Data Entry *n***",
- "**[FL] (1 byte) Data Entry *n***",
- "**[FL] (2 bytes) Data Entry *n***".

As stated in section 2.4, up to 32 one-bit objects, 16 one-byte objects and 16 two-byte objects can be enabled. None of them comes enabled by default.

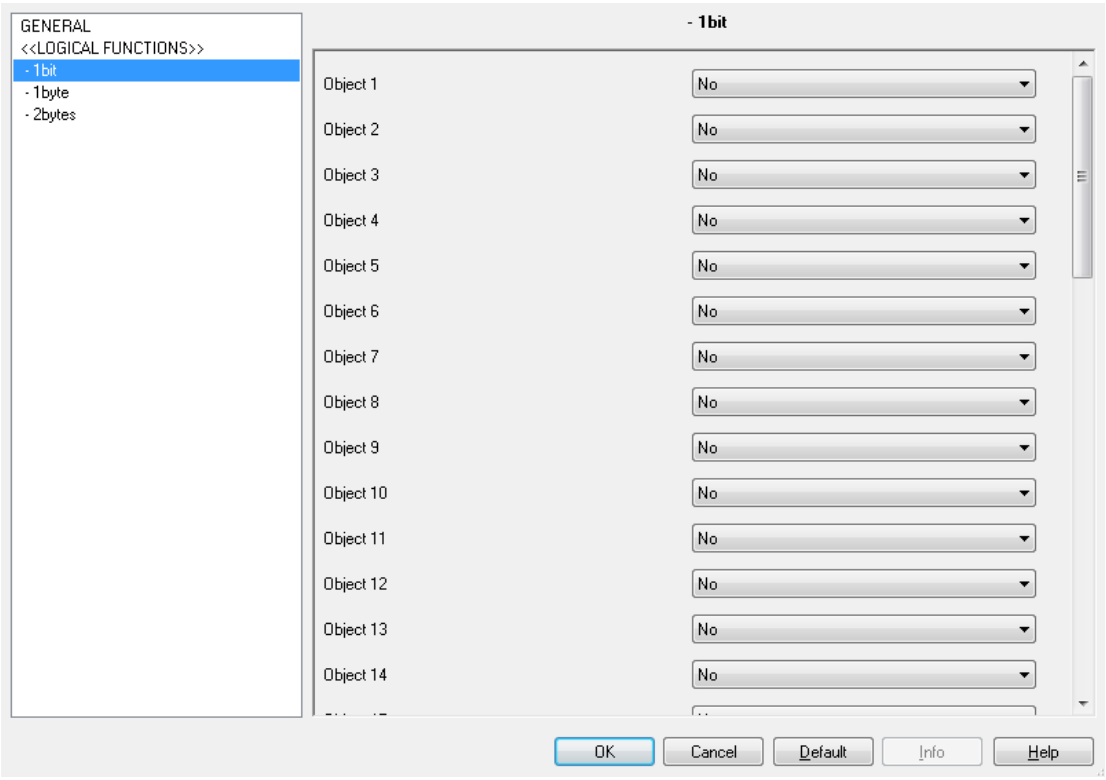



**Figure 2.** Enabling the (one-bit) Input Objects.

## 3.3 FUNCTION n

A specific tab will be included into the tab list on the left per enabled function (see 3.1), being itself divided into three more tabs.

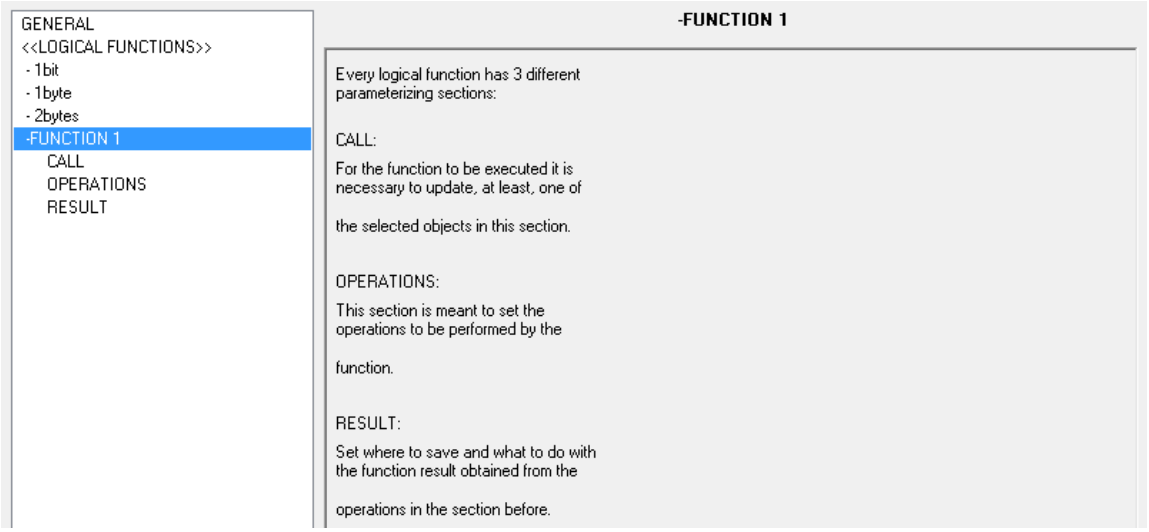



**Figure 3.** Function 1

### 3.3.1 CALL

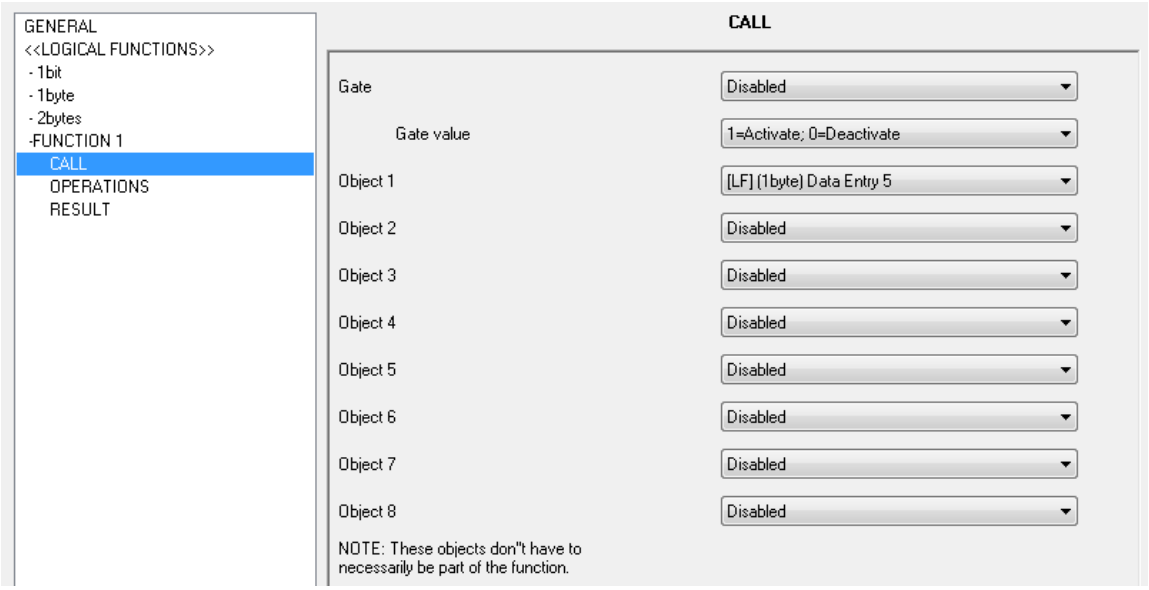



**Figure 4.** "Call" Tab

This section lets selecting as many as **eight objects** (which need to be specifically enabled; see section 3.2) to work as call objects. The call objects will be responsible for triggering the execution of the function whenever one of them receives a value from the bus. Of course, setting one particular object as the call object of multiple functions will make them trigger together whenever the object is written a value.

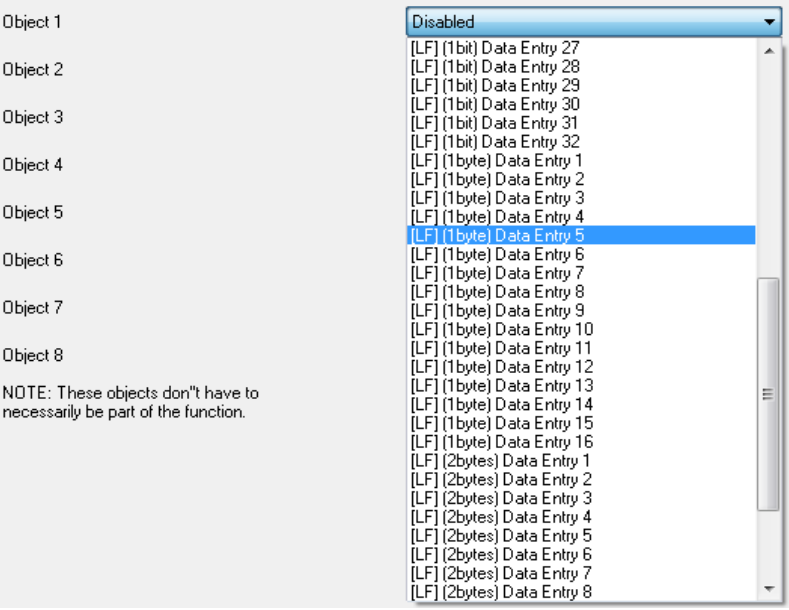



**Figure 5.** Selecting the Call Objects

Moreover, the "**Gate**" parameter in this window permits selecting which of the active objects (see section 3.2) should act as the enable/disable object of the function (i.e., the "gate" object), and the reaction to the two possible values (through the "**Gate value**" parameter): "[0=Activate; 1=Deactivate]" and "[1=Activate; 0=Deactivate]"), as explained in section 2.7. It is possible to discard the gate functionality by setting the "**Gate**" parameter to "Disable" (which is the option selected by default).

**Notes**:

- *After a download into the device, all the gate objects are initialised with the value corresponding to "deactivated" ("0" or "1", depending on the parameterisation), while after a bus failure, they will always recover the values they had prior to the failure.*
- *If the gate object of a function is itself part of that (or any other) function, for example as an operand or as a result, special care should be taken in relation to the values it acquires, as they may unexpectedly enable/disable the function.*

### 3.3.2 OPERATIONS

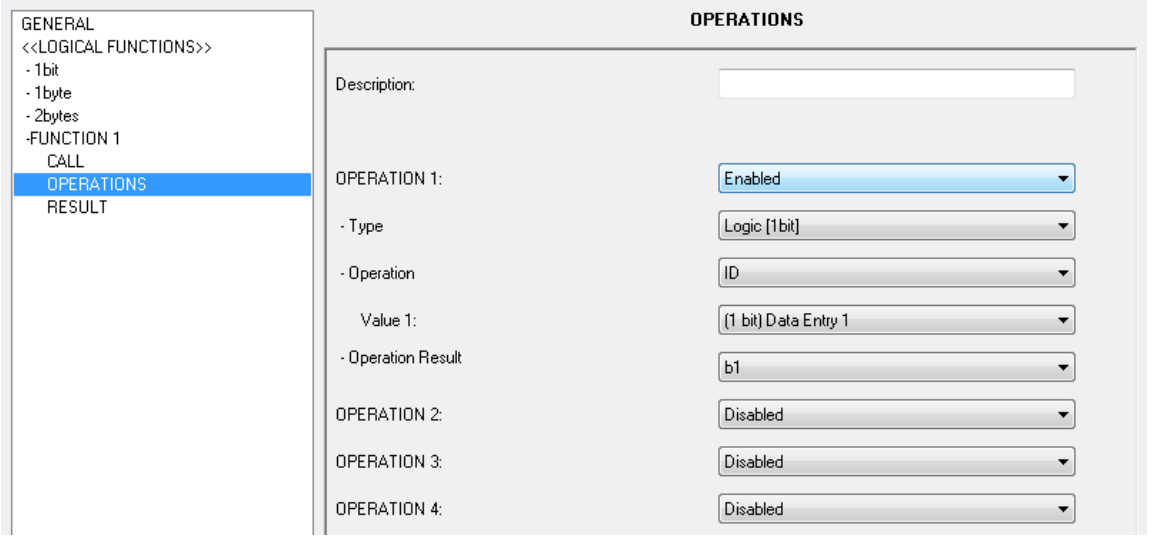



**Figure 6.** Operations

The purpose of this section is to define the different operations that will compose the function, by means of the following parameters:

- **Description**: sets a brief description (up to 100 characters) for the logical function. This field simply helps to identify the function, and does not have practical implications – it is provided for the convenience of the integrator.

- **Operation "i"**: enables or disables the operation number "i" (1-4). Every operation enabled, for its part, will offer the following parameters:

  - **Type**: sets the type of the operation (logical, arithmetical, comparison or conversion) and the size of the involved operands (one bit, one-byte unsigned integer, two-byte unsigned integer, two-byte floating point). See section 2.3.

  - **Operation**: sets the particular action executed by operation number "i". Depending on the selected operation type (logical, arithmetical, comparison or conversion), this parameter will display different options. For further information, please refer to ANNEX I: Operations Reference.

  - **Operand "j"**: depending on the selection in the above parameter, one or more additional parameters named "Operand j" will show up, thus letting the user set the input values (operands) of the operation. These may be communication objects, internal variables or constant values.

- **Operation Result**: sets the internal variable where the result of the operation will be stored. This partial result may be configured afterwards as the final function result or take part in later operations as an operand, if desired.

**Note**: *all the logical functions share together the same set of internal variables. This means that, for example, if function no. 1 stores a partial result into the variable "n1" and afterwards function no. 2 reads that variable (to perform an operation on its value), it will find the value that was written there by function no. 1.*

### 3.3.3 RESULT

From this section it is possible to establish which internal value should be considered as the one that contains the final result of the function, so that after executing all the operations that make up the function, the value of that variable will be reported to the bus through the "**[FL] Function *n* RESULT (size)**" object, where "size" will depend on the configuration.

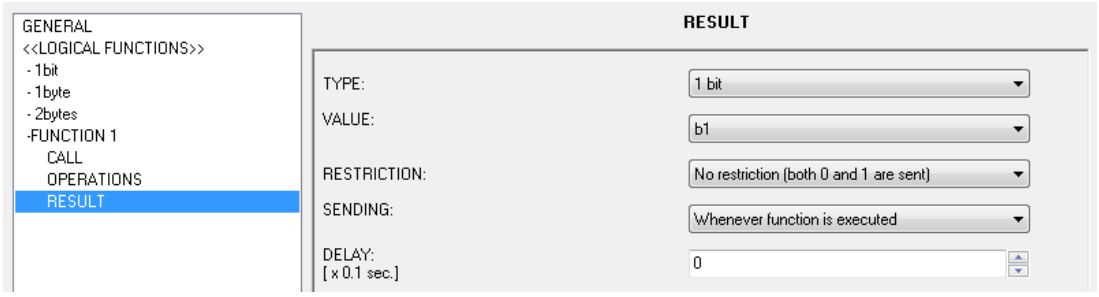



**Figure 7.** Result

- **Type**: sets the size of the function result. The available options are "1 bit", "1 byte", "2 bytes (unsigned integer)" and "2 bytes (floating point)".

- **Value**: sets the particular internal variable whose value will be sent to the bus, through the function result object, after performing the operations.

- **Restriction**: sets restrictions over the results to be sent to the KNX bus. The available options are:

  - **One bit**:

    - No restriction (both 0 and 1 are sent),
    - Only 0 is sent,
    - Only 1 is sent.

- **One byte / Two bytes (unsigned integer) / Two bytes (floating point)**:

  - No restriction,
  - Only send values equal to reference,
  - Only send values not equal to reference,
  - Only send values lower than reference,
  - Only send values higher than reference.

  The reference value should be specified through the "**Reference Value**" parameter, and may take values between 0 and 255 (in the case of one-byte unsigned integers), between 0 and 65535 (in the case of two-byte unsigned integers) and between 0 and 1200 tenths, i.e., between 0 and 120.0 (in the case of two-byte floating point values).

- **Sending**: this field defines under what circumstances the result of the function should be sent to the KNX bus.

  - Whenever Function is Executed.

  - Result is Different from Last Sent: the result of the function will only be sent to the bus when it differs from the value previously sent.

    **Note**: *temporarily disabling a function through its "gate" object has no effect over the restrictions: once enabled again, the function will still remember the last result sent*.

  - Periodical: the result object will be sent (updated each time) repeatedly to the bus, every certain time once the function is triggered for the first time, depending on the "**Cycle Time**" parameter (from 0 to 65535 seconds).

    Bear in mind that after sending the result for the first time, the object will only be re-sent when the period expires, thus the **responses to successive calls** to the function may not be immediate: the cycle time is not restored every time the function is called again (although it will do in case of a power interruption).

    **Note**: *if an order to disable the function is received through its gate object while a periodical sending was running, this sending will be interrupted. It will only resume after enabling and calling/triggering the function again*.

- **Delay**: sets a delay time (between 0 and 65535 seconds) to be counted after the execution of the function and before sending the result to the bus. If the result is preferred to be sent immediately, this should be set to "0".

  The fact that the delay applies to the sending of the result should be clear: the function will still execute after the call, no matter if the result is afterwards transmitted after a delay, and without resulting affected by any changes in the value of the operands that may take place during such delay time.

  **Note**: *if an order to disable the function is received through its gate object during the delay time, the transmission of the result will be cancelled and will not take place until the function is enabled and called/triggered again*.

# ANNEX I: OPERATIONS REFERENCE

## OPERATIONS IN BOOLEAN LOGIC (1 BIT)

- **ID (identity)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 1 |

- **AND (logical conjunction)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- **OR (logical disjunction)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

- **XOR (exclusive OR)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- **NOT (negation)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 1 |
| 1 | - | 0 |

- **NAND (negated AND)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- **NOR (negated OR)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

- **NXOR (negated XOR)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## ARITHMETIC OPERATIONS

- **ID (identity)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | - | *v1* |

- **ADDITION**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *v1 + v2* |

- **SUBTRACTION**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *v1 - v2* |

- **MULTIPLICATION**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *v1 * v2* |

- **DIVISION**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *v1 / v2* |

- **MAXIMUM**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *max(v1, v2)* |

- **MINIMUM**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *min(v1, v2)* |

**Note**: *it is advisable to read the Additional Remarks for further information about specific cases and overflows*.

## COMPARISON OPERATIONS

- **HIGHER**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *1 ↔ v1 > v2.*<br>*0 in any other case.* |

- **HIGHER OR EQUAL**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *1 ↔ v1 ≥ v2.*<br>*0 in any other case.* |

- **LOWER**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *1 ↔ v1 < v2.*<br>*0 in any other case.* |

- **LOWER OR EQUAL**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *1 ↔ v1 ≤ v2.*<br>*0 in any other case.* |

- **EQUAL**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *1 ↔ v1 = v2.*<br>*0 in any other case.* |

- **NOT EQUAL**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| *v1* | *v2* | *1 ↔ v1 ≠ v2.*<br>*0 in any other case.* |

**Note**: *it is advisable to read the Additional Remarks for further information about specific cases and overflows*.

## CONVERSION OPERATIONS

### Conversion to one bit

- **1 byte → 1 bit**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1-255 | - | 1 |

- **2 bytes (unsigned integer) → 1 bit**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 - 65535 | - | 1 |

- **2 bytes (floating point) → 1 bit**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| ≤ 0,00 | - | 0 |
| 0.01 – 120.00 | - | 1 |

### Conversion to one byte

- **1 bit → 1 byte**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 1 |

- **2 bytes (unsigned integer) → 1 byte**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 1 |
| … | … | … |
| 255 | - | 255 |
| 256 - 65535 | - | 255 |

- **2 bytes (floating point) → 1 byte**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| ≤ 0.00 | | 0 |
| 0.01 | - | 0 |
| … | … | … |
| 0.10 | - | 1 |
| 0.11 | - | 1 |
| … | … | … |
| 25.49 | - | 254 |
| ≥ 25.50 | | 255 |

### Conversion to two bytes (unsigned integer)

- **1 bit → 2 bytes (unsigned integer)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 1 |

- **1 byte → 2 bytes (unsigned integer)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 1 |
| … | … | … |
| 255 | - | 255 |

- **2 bytes (floating point) → 2 bytes (unsigned integer)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| ≤ 0.00 | - | 0 |
| 0.01 | - | 0 |
| … | … | … |
| 0.10 | - | 1 |
| 0.11 | - | 1 |
| … | … | … |
| 119.98 | … | 1199 |
| 119.99 | - | 1199 |
| ≥ 120.00 | - | 1200 |

### Conversion to two bytes (floating point)

- **1 bit → 2 bytes (floating point)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0,00 |
| 1 | - | 0.10 |

- **1 byte → 2 bytes (floating point)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 1 |
| … | … | … |
| 254 | - | 25.40 |
| 255 | - | 25.50 |

- **2 bytes (unsigned integer) → 2 bytes (floating point)**

| *1st Operand* | *2nd Operand* | *Result* |
|---|---|---|
| 0 | - | 0 |
| 1 | - | 0 |
| … | … | … |
| 1199 | - | 119.90 |
| ≥ 1200 | - | 120.00 |

## ADDITIONAL REMARKS

As already stated, the X10 logical functions module can operate on the following data types:

- Binary: **0** and **1**.
- Unsigned integers (one byte): **0 – 255**.
- Unsigned integers (two bytes): **0 – 65535**.
- Floating-point decimal numbers (two bytes): **0.00 – 120.00**.

These operands may be communication objects, internal variables where the partial results are temporarily stored, or even, for certain operations, numerical constants set by parameter in ETS.

On the other hand, it is important to bear in mind the following remarks:

- **Overflows in arithmetic operations** are dealt with by returning the value of the limit being surpassed. For instance, a one-byte addition between the values 250 and 10 will return 255 and not 260, as 255 is the limit of the range permitted for one byte.
- Floating-point operations **always truncate decimal values to tenths of a unit** in their operandos. For example, assuming that the "[FL] (2 bytes) Data Entry 1" object has been given a value of 2.37, the result of running an "ID" (identity) operation on it will be 2.30. Analogously, the result of adding an object with value 0.09 to an object with value 1.27 will be 1.20 (i.e., the result of 0.0 + 1.2), and not 1.36, nor 1.30, nor 1.40.
- Moreover, floating-point operations **will always truncate to zero any negative input or result values**. For example, assuming that the "[FL] (2 bytes) Data Entry 1" object has been given a value of -5.00, the result from adding it to "[FL] (2 bytes) Data Entry 2") when the latter has a value of +10.00 will be 10.00, while the result of subtracting the latter from the former will be 0.00, not -15.00 nor -10.00.

- **Power failures** in the bus do not imply the loss of the values of the objects nor of the internal variables: after the power recovery, they will still maintain their previous values.

- **The Multiply and Divide operations are only intended to operate on an object** (for instance, a temperature object) **and a numerical constant** (configurable in ETS), **or on an internal variable and a numerical constant**. Mathematically incorrect or unexpected results may be sometimes thrown due to overflows and other restrictions, if one object/variable is multiplied or divided by another object/variable, although such parameterisation is possible.

- **Divisions by zero** do not return a result.

**Join and send us your inquiries about Zennio devices:**

**http://zennioenglish.zendesk.com**

**Zennio Avance y Tecnología S.L.**
C/ Río Jarama, 132. Nave P-8.11
45007 Toledo (Spain).

*Tel. +34 925 232 002.*
*Fax. +34 925 337 310.*
*www.zennio.com*
*info@*zennio*.com*