



# Logical Functions X10

Zennio module

of 10 logical functions



## INDEX

1. Introduction.....	3
1.1. Logical functions module .....	3
1.2. Zennio devices with X10 module .....	4
2. Configuration.....	5
2.1. Overall working .....	5
2.2. Entry objects .....	6
2.3. Internal variables.....	6
2.4. Gate object.....	6
2.5. Call Objects .....	7
2.6. Operations .....	7
2.7. Result.....	7
3. ETS Parameterization.....	8
3.1. General window .....	8
3.2. Call .....	9
3.2.1. Gate object.....	9
3.2.2. Call objects.....	10
3.3. Operations .....	11
3.3.1. Operations type.....	12
3.4. Result .....	13
3.5. Communication objects .....	15
Annex I. Logical operations .....	16
Annex II. Conversion operations .....	18

# 1. INTRODUCTION

---



## 1.1. LOGICAL FUNCTIONS MODULE

Some Zennio devices, like the actuators, have implemented a logical functions module that is meant to perform binary logic operations with incoming data from the KNX bus to send the result through other communication objects, of different lengths (1-bit, 1-byte and 2-bytes), specifically enabled for this aim.

These logical functions work with two different types of data:

- From the **KNX bus**, through special communication objects enabled for these functions.
- From **Internal variables**, to store the intermediate partial operation results.

There are two logical functions modules:

-  **5 multi operational logical functions module (X5)**. Up to 5 different logical functions can be enabled, independent of each other, which can carry out up to 4 operations each.
-  **10 multi operational logical functions module (X10)**. Up to 10 different logical functions can be enabled, independent of each other, which can carry out up to 4 operations each.

In the current document, the 10 logical functions module (X10) is explained in detail.

## 1.2. ZENNIO DEVICES WITH X10 MODULE

In the table 1.1 it is shown the Zennio devices (device name and application program version) that currently have incorporated the 10 logical functions module (X10) explained in the present documentation. This table will be updated every time the number of versions and devices on which this module is implemented increases.

Device	Application Program Version
MAXinBOX 16	1.0
MAXinBOX 16	2.0
ACTinBOX MAX6	3.0
ACTinBOX QUATRO	2.0

Table 1.1. Zennio devices with logical functions module X10

**Note:** For more detailed information about the X10 module parameterization of Zennio devices, please consult the "ETS Parameterization", in the section 3 of this document.

## 2. CONFIGURATION

A serial of general concepts concerning the logical functions module X10 are explained below.

### 2.1. OVERALL WORKING

The working of the logical functions module is summarized in the following points:

- 🌐 **CALL.** The first step is calling the function to be executed. For this aim, one or several communication objects will be enabled (up to a total of 8). Every time their value is updated, the corresponding logical function will be executed.
- 🌐 **OPERATIONS.** Every logical function can carry out up to 4 different operations. For all of them, one must define the type of operation, the operands (values to operate with, which may be communication objects or internal variables) and the internal variable where the partial result of every operations will be stored.
- 🌐 **RESULT.** Finally, the result of the functions is stored in an internal variable and sent to the KNX bus through a communication object. It is possible to configure different options for this sending.

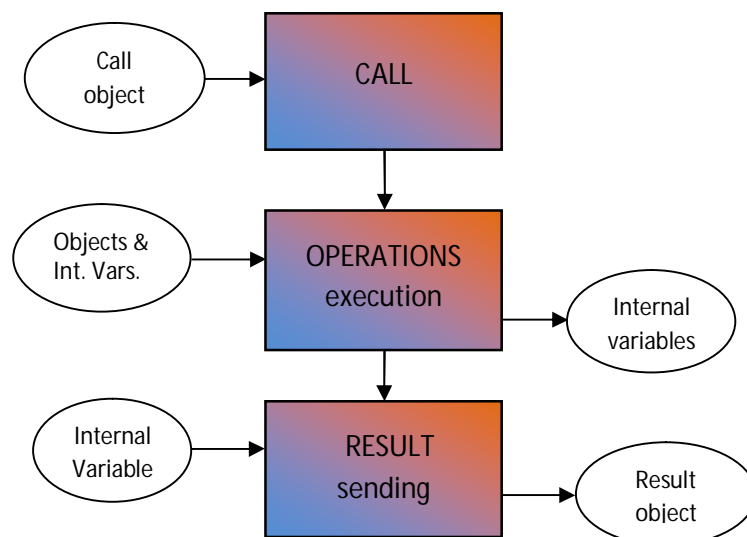





Figure 2.1. Overall working of logical functions module

## 2.2. ENTRY OBJECTS

The communication objects used in logical functions can be 1-bit, 1-byte or 2-bytes length. One needs to enable, one to one, all the necessary objects of every type to be used as entry data in the binary logic operations to be carried out.




For the logical functions module X10, the following objects can be enabled:

-  **32 communication objects of 1-bit.**
-  **16 communication objects of 1-byte.**
-  **16 communication objects of 2-bytes.**

## 2.3. INTERNAL VARIABLES

These are variables of the logical functions module, which are handled internally and that enable storing the intermediate partial operations results.

The logical functions module X10 has the following internal variables:

-  **32 variables** to store **1 bit** results [b1 ... b32]
-  **16 variables** to store **1 byte** results [n1 ... n16]
-  **16 variables** to store **2 byte** results [x1 ... x16]

## 2.4. GATE OBJECT

For every of the 10 logical functions, one of the 32 1-bit communication objects available can be used as **gate** for the function. This gate object allows activating or deactivating the execution of the logical function when it receives the corresponding binary value ("0" or "1", as parameterized).

See section 3.2 for further information about this object.

## 2.5. CALL OBJECTS

The calling objects are those that trigger the function execution when they update their value.

Up to 8 objects (of different type, from the established data entry objects) can be configured for every enabled logical function.





It is not necessary that the objects in charge of triggering the function are included in it; i. e., they may not be used as operands of the function.

## 2.6. OPERATIONS

The operations are a set of steps that allow getting several results from the established data entry objects.

The logical functions module X10 allows enabling and configuring up to 4 different operations for every function.

These operations may be of 4 types:

-  **Logic** (AND/OR/NOT/...)
-  **Arithmetic** (Add/Divide/Maximum/...)
-  **Comparison** (Higher/Lower/Unequal/...)
-  **Conversion** (to 1 bit /to 1 byte /to 2 bytes)

The options associated to every type of operation will be explained in detail in section 3.3.1.

## 2.7. RESULT

The results are the values that every function will send to the bus once all the defined operations have been carried out.

It is possible to set when and how the sending of those results to the KNX bus is carried out, by parameterizing a periodical sending, or with a certain delay, for instance.

They can be of three types: 1-bit, 1-byte or 2-bytes results.

### 3. ETS PARAMETERIZATION

Next the logical functions module X10 configuration in ETS is shown:

The screenshots are taken from the MAXinBOX 16 actuator (version 2.0) but all the options, parameters and aspect of the logical functions module configuration window in ETS are the same for all Zennio devices that incorporate this X10 module (see Table 1.1).

#### 3.1. GENERAL WINDOW

In the application program of the devices with the X10 module, there is an exclusive window to configure it:

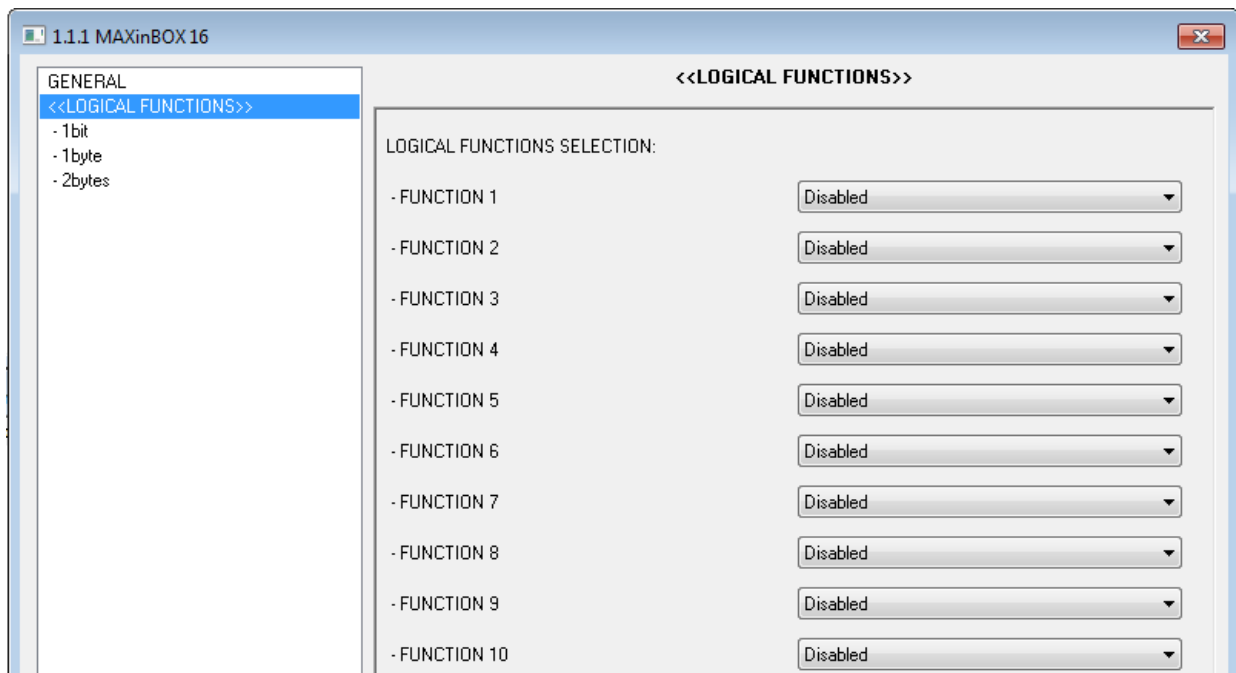


Figure 3.1. Logical functions general window

As it can be seen in the Figure 3.1, the 10 logical functions are disabled by default.

In the "1bit", "1byte" and "2bytes" flanges it is possible to enable, one to one, the communication objects of several lengths to be used as data entry objects for the functions.

As seen in the Configuration section, up to 32 1-bit objects, 16 1-byte objects and 16 2-bytes objects can be enabled.



When enabling the logical functions, several windows will appear to configure the Call, Operation and Result options. Next, these will be explained in detail.

### 3.2. CALL

Every enabled logical function has the section **Call**, to select the objects that will be in charge of calling the function, and to enable the use of a gate object for the function.

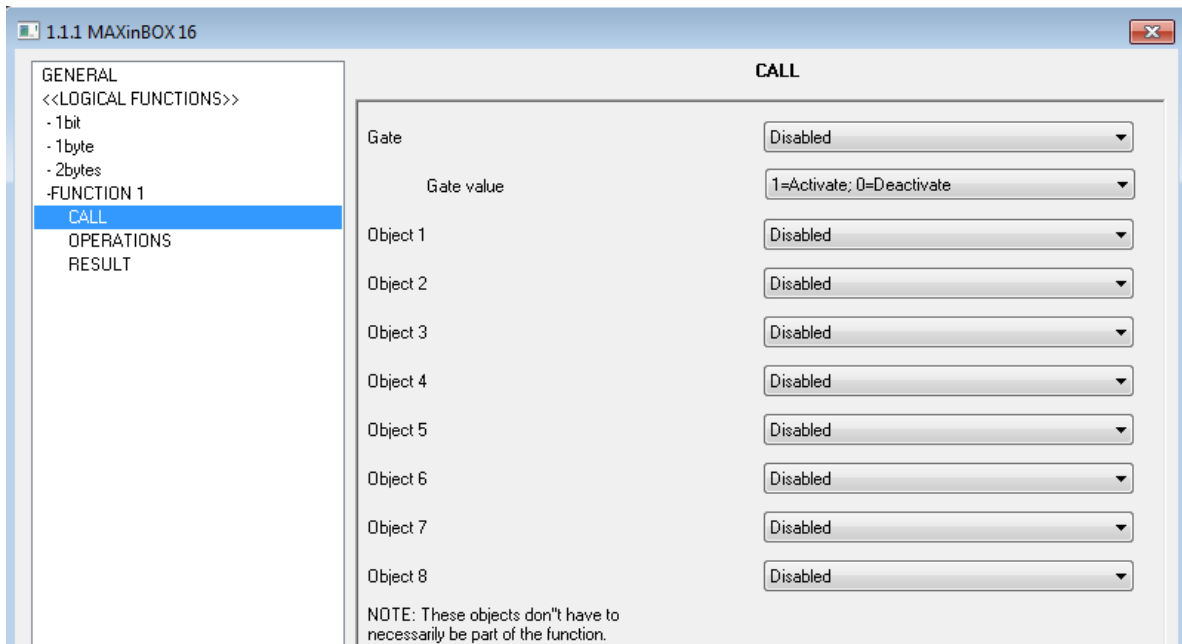
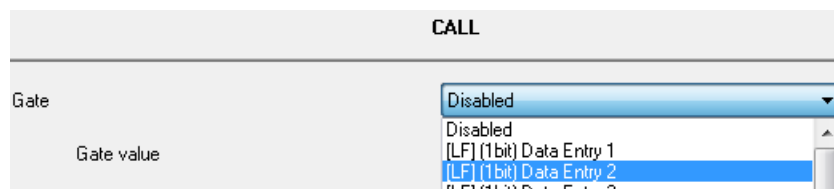


Figure 3.2. Call window

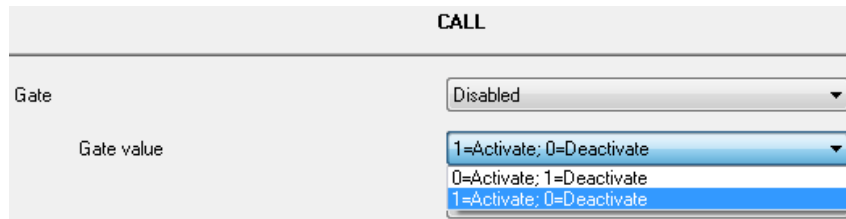
#### 3.2.1. GATE OBJECT

If one wants to control the functions execution through an object, it is necessary to enable the **gate object**, selecting in the drop-down menu one of the 32 1-bit available objects.



It is possible to enable one gate object per function, up to a total of 10 (equal or different). If no gate object is needed, one must select the option "Disabled"; in this case, the logical functions will be executed when they are called by the corresponding call objects, since they will be always ready to be executed.

The gate value to activate/deactivate the logical function must be set. The available options are: [0=Activate; 1=Deactivate] and [1=Activate; 0=Deactivate].



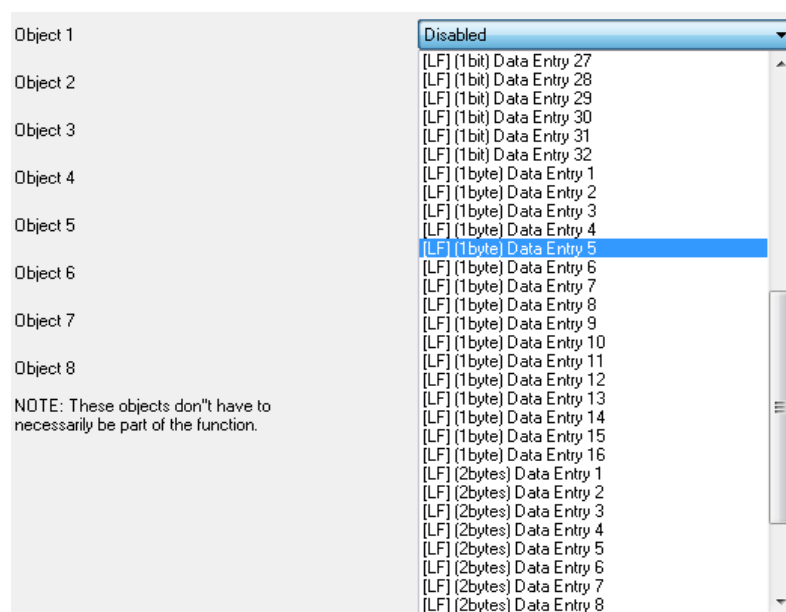
When the corresponding function is called, it will be executed (provided that the gate object has the corresponding activation value). When the gate object has the opposite value (deactivation value), the execution of the function will be disabled (the function cannot be called until the gate object has the activation value)

**Note I:** After downloading the application program, the gate object will be disabled (value "0" or "1", as parameterized). After a bus power failure, it keeps the value it had before.

**Note II:** Take into account that if one selects the same gate object as one used as data entry object of the function, pay attention to its update value, since it may enable/disable the function at a non-desired moment.

### 3.2.2. CALL OBJECTS

The call objects are those whose update triggers the logical function execution. Up to 8 different call objects can be selected, from the 64 available.



For the function to be executed, at least one of the enabled objects in this section must update its value and send it to the KNX bus. In case the gate object has been enabled, it must also receive the activation value for the function to be executed when it is called.

### 3.3. OPERATIONS

In this section, the operations of the function are defined. For every enabled logical function, it is possible to configure up to 4 different operations.

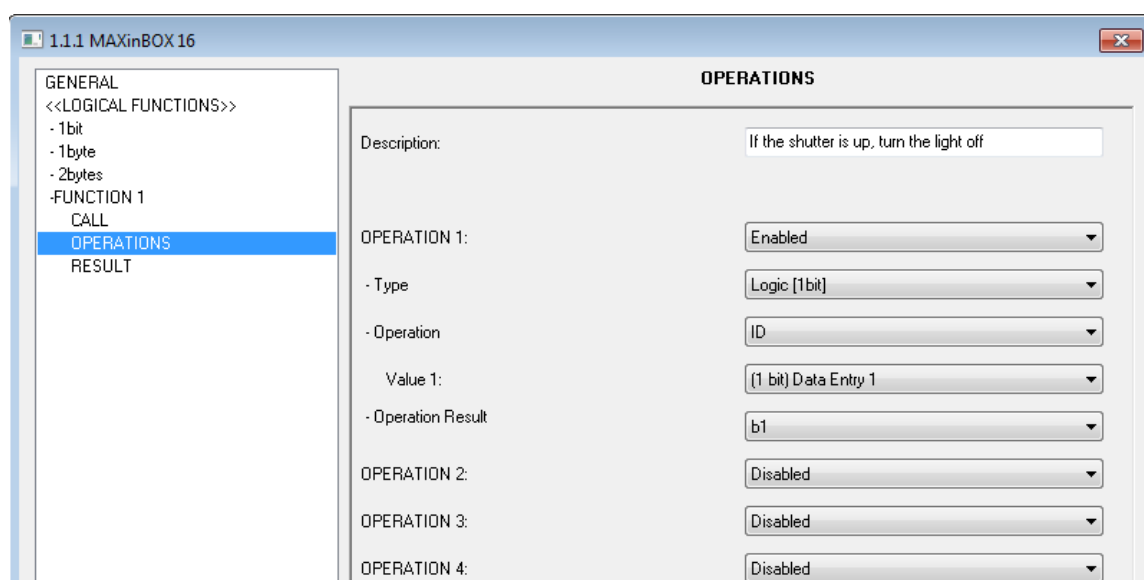






Figure 3.3. Operations window

In the field "**Description**" it is possible to write a short description (100 characters) of the logical functions: what is its aim, which operations carries out, its result, etc.

To parameterized an operation, it must enabled previously. When enabling it, the following options will appear:


-  **Type:** the logical function can carry out 4 different operation types. See section 3.3.1 to know the operation types available.
-  **Operation:** depending on the selected type, different operations can be carried out. They must be selected here. Consequently, one or two operands (values) will be enabled. These may be: communication objects (data entry objects), internal variables or constant values.
-  **Operation result:** to select the internal variable, of the corresponding size, where the partial result of the operation will be saved.

### 3.3.1. OPERATIONS TYPE

 **Logic:** this type of operations works with 1-bit values. The logical operations available are: **ID**, **AND**, **OR**, **XOR**, **NOT**, **NAND**, **NOR** and **NXOR** (see [Annex I. Logical Operations](#), to better know the logical operations that are carried out).

All of them work with two different operands (Value 1 and Value 2), except **ID** and **NOT**, which only work with Value 1. The values, as mentioned before, can be chosen from the available 32 1-bit objects (all of them are shown, but only the ones previously enabled as data entry objects are valid) and the 32 1-bit internal variables (b1...b32).

The operation result is a 1-bit value that can be stored in any of the 32 available 1-bit internal variables (b1...b32).


 **Arithmetic [1 byte/2 bytes (unsigned integer)/2 bytes (floating point)]:** depending on the chosen type, these operations will work with 1-byte or 2-bytes (unsigned integer or floating point) values. The arithmetic operations available are: **ID**, **ADD**, **SUBTRACT**, **MULTIPLY**, **DIVIDE**, **MAXIMUM** and **MINIMUM**.

All of them work with two different operands (Value 1 and Value 2), except **ID**, which only work with Value 1. These values can be chosen from the available 16 1-byte/2-bytes objects (the ones previously enabled as data entry objects), the 16 1-byte internal variables (n1...n16), the 16 2-bytes internal variables (x1...x16) or a constant value defined by parameter (value between 0 and 255, for arithmetic 1-byte; between 0 and 65535 for 2-bytes unsigned integer and between 0 and 120.0 for 2-bytes floating point).

The operation result will be 1-byte or 2-bytes (depending on the chosen operation) and can be stored in any of the 16 1-byte internal variables (n1...n16) or in any of the 2-bytes internal variables (x1...x16).

**Note I:** *If the result in the 2-bytes arithmetic operations exceeds the allowed range, this will be converted to the corresponding limit in the range.*


**Note II:** *Dividing by 0 does not send anything to the KNX bus.*

 **Comparison [1 byte/2 bytes (unsigned integer)/2 bytes (floating point)]:** depending on the chosen type, these operations will work with 1-byte or 2-bytes (unsigned integer or floating point) values. The logical operations available are: **HIGHER**, **HIGHER OR EQUAL**, **LOWER**, **LOWER OR EQUAL**, **UNEQUAL** and **EQUAL**.

All of them work with two operands (Value 1 and Value 2), which can be chosen from the available 16 1-byte/2-bytes objects (the ones previously enabled as data entry objects), the 16

1-byte internal variables (n1...n16), the 16 2-bytes internal variables (x1...x16) or a constant value defined by parameter (value between 0 and 255, for arithmetic 1-byte; between 0 and 65535 for 2-bytes unsigned integer and between 0 and 120.0 for 2-bytes floating point).

The operation result will be 1-bit (value "1" if the comparison is fulfilled and "0" if not). This result can be stores in any of the 32 1-bit internal variables (b1...b32).

 **Conversion [1 byte/2 bytes (unsigned integer)/2 bytes (floating point)]:** to convert he communication objects between formats. All the available conversions are explained in [Annex II. Conversion operations](#) in the current documentation. All of them work with a single operand (Value 1), which can be chosen from the communication objects enabled as data entries or any of the internal variables, depending on the type of conversion. The result type will also depend on the conversion type.

### 3.4. RESULT

This section is meant to determine where to store and what to do with the result obtained in the previous operations.

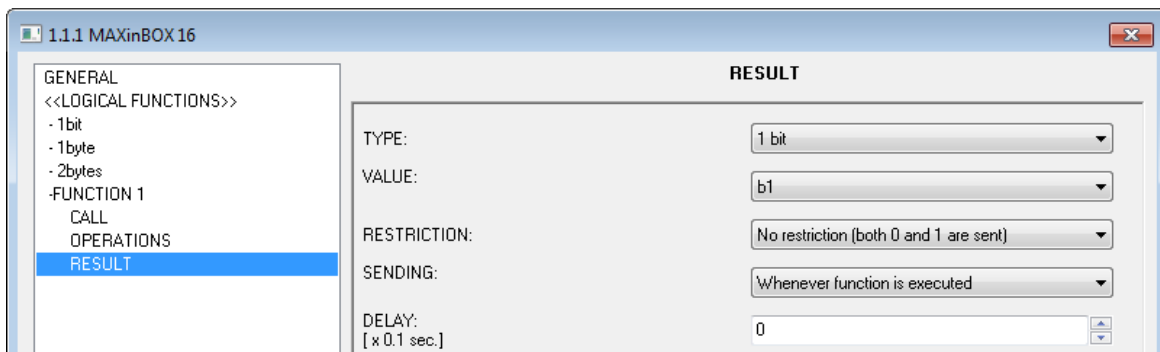





Figure 3.4. Result window


These configuration options are shown:

-  **Type:** choose among 1-bit, 1-byte, 2-bytes (unsigned integer or floating point) for the final result.
-  **Value:** set the internal variable (function of the result type) where the final result will be stored. This value will be sent to the KNX bus.
-  **Restriction:** the sending of the function result can be restricted, i.e., the function will decide whether to send the result to the bus or not, depending on the result value. For 1-bit


results, the sending of the final result can be restricted to just sending the value "1" or the value "0". For 1-byte and 2-bytes results, the following restrictions may be applied:

- **Values equal to reference one**
- **Values not equal to reference one**
- **Values higher than reference one**
- **Values lower than reference one**

Just below, a new field will appear where to set the desired reference value. For 1-byte results, it is possible to set a value between [0-255]; for 2-bytes (unsigned integer) results, a value between [0-65535] and for 2-bytes (floating point) results, a value between [0-1200, in tenths].

 **Sending:** set the conditions to send the result to the KNX bus, choosing among the following:

- **Whenever function is executed.** This parameter is related to section "Call" (see section 3.2), since the function will be executed every time any of the enabled objects in the the section "Call" is updated.
- **Result is different from last sent.** The result will be sent to the KNX bus every time the final result in the operations changes its value.
- **Periodical sending.** The result will be periodically sent every "x" seconds (defined in the field "Cycle time", shown when selecting this option).


 **Delay:** time to pass before sending the result to the KNX bus. If no delay is needed, set a 0 in this field.

Take into account the following considerations:


- If the gate object receives the value parameterized for disabling the function ("0" or "1", see section 3.2) during a periodical sending of the result, this sending will be interrupted and will not be sent until the function is executed again.
- If this disabling value is receives while waiting for a result with delay, the sending of the result will be interrupted and the function will not executed until receiving the enabling value through the gate object and the function is called again.

### 3.5. COMMUNICATION OBJECTS

The communication objects for the logical functions module can be of two types:

 **Data:** data coming from the KNX bus, with which the operations work. These data are named in ETS as follows:

➤ **[LF] ("size") Data Entry "Y":** where "size" can be "1-bit", "1-byte" or "2-bytes" and "Y" is the datum number, i.e., from 1 to 32 for 1-bit objects and from 1 to 16 for 1-byte and 2-bytes objects.

 **Results:** these are the function results. One result per function. These data are named in ETS as follows:

➤ **[LF] Function "X" RESULT ("size"):** where "size" can be "1-bit", "1-byte" or "2-bytes" and "X" is the number of the corresponding function, value from 1 to 10.

The concrete number associated to the communication objects related to logical functions is different in all the Zennio devices that have the X10 module. To know these numbers, please consult the corresponding user manual, the Annex "Communication objects". These manuals are available at: <http://www.zennio.com>.

# ANNEX I. LOGICAL OPERATIONS

This Annex is meant to describe the different logical operations that the Zennio logical functions modules can carry out (See section 3.3.1 for further information on parameterization).

## ID (Identity)

Value 1	Result
0	0
1	1

## AND (Conjunction)

Value 1	Value 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

## OR (Disjunction)

Value 1	Value 2	Result
0	0	0
0	1	1
1	0	1
1	1	1


## XOR (Exclusive OR)

Value 1	Value 2	Result
0	0	0
0	1	1
1	0	1
1	1	0


## NOT (Negation)

Value 1	Result
0	1
1	0




 **NAND (Not AND)**

Value 1	Value 2	Result
0	0	1
0	1	1
1	0	1
1	1	0

 **NOR (Not OR)**

Value 1	Value 2	Result
0	0	1
0	1	0
1	0	0
1	1	0

 **NXOR (Not XOR)**

Value 1	Value 2	Result
0	0	1
0	1	0
1	0	0
1	1	1

## ANNEX II. CONVERSION OPERATIONS

This Annex is meant to describe the different conversion operations that the Zennio logical functions modules can carry out (See section 3.3.1 for further information on parameterization).

### Conversion [1 bit → 1 byte]

Value 1	Result
0	00000000
1	00000001

### Conversion [1 bit → 2 bytes (unsigned integer)]

Value 1	Result
0	\$00 00
1	\$00 01

### Conversion [1 bit → 2 bytes (floating point)]


Value 1	Result
0	0
1	0.1

### Conversion [1 byte → 1 bit]


Value 1	Result
0	0
1-255	1

### Conversion [1 byte → 2 bytes (unsigned integer)]


Value 1	Result
0	\$00 00
1	\$00 00
...	...
255	\$00 FF

 **Conversion [1 byte → 2 bytes (floating point)]**

Value 1	Result
0	0
1	0.1
...	...
255	25.5

 **Conversion [2 bytes (unsigned integer) →1 bit]**

Value 1	Result
0	0
1-65535	1

 **Conversion [2 bytes (unsigned integer) →1 byte]**

Value 1	Result
0	0
1	1
...	...
255	255
>255	255

 **Conversion [2 bytes (unsigned integer) →2 bytes (floating point)]**

Value 1	Result
0	0
1	0.1
...	...
1200	120.0
>1200	120.0

 **Conversion [2 bytes (floating point) →1 bit]**

Value 1	Result
0	0
0.1-120.0	1

 **Conversion [2 bytes (floating point) →1 byte]**

Value 1	Result
0	0
0.1-25.5	1-255
...	...
>25.5	255

 **Conversion [2 bytes (floating point) →2 bytes (unsigned integer)]**

Value 1	Result
0	0
0.1	1
...	...
120.0	1200



**BECOME USER!**

<http://zennioenglish.zendesk.com>

**TECHNICAL SUPPORT**