Table of Contents

Description

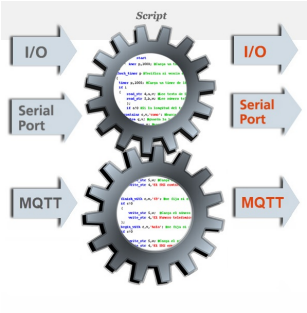
Some Exemys devices allow user-written programs to run on the device itself, making them more flexible and powerful.

The device will continue to operate normally while the script stored in its memory is running.

Operations that can be performed from a script

- **Mathematical** operations
- **Logical** operations
- **Timing** operations
- Reading the device's own inputs and Modbus variables
- Turning digital outputs on and off
- Interpreting data from the **serial port**
- **MQTT** publish/subscribe

(Some of these operations may or may not be available depending on the programmed device)



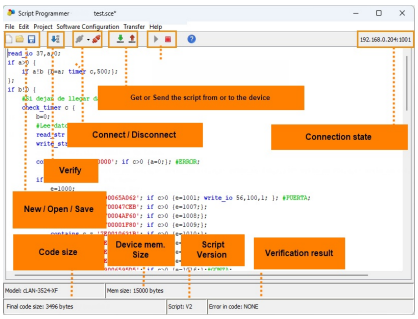
2026-05-05

Introduction

This program allows developing, compiling and transferring scripts to Exemys devices that support it.

Software Description

Below is a description of the main screen functions.



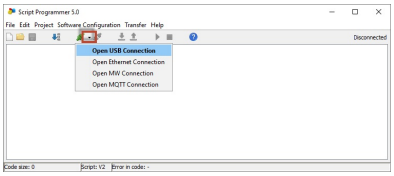
Device Connection

There are three ways to connect: via USB, via LAN/Ethernet, and via MW-XF depending on the device.

Device Connection - USB

First connect the device to the PC and make sure it is disconnected from ***"GRDconfig"***

Then expand the Connect button and choose ***"Open USB Connection"***

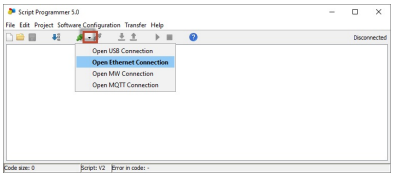


> ⚠ On some devices with a USB port, such as the RMS1-RM and wRemote-LoRa, you must disconnect and reconnect the device's USB port if you previously accessed the USB configuration console.

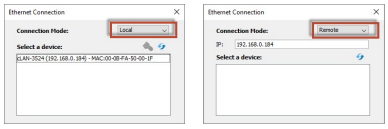
Device Connection - LAN/Ethernet

First connect the device to the same network as the PC and make sure it has a valid IP address configured as indicated in its user manual.

Then expand the Connect button and choose ***"Open Ethernet Connection"***

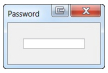


Doing so you should see the device(s) connected to your network, or you can choose the **"Remote"** mode to manually enter the device's IP address if it is on a different network.



Select the one you want to configure.

To connect you will need to enter the device password. It is the same one used to connect via MW.

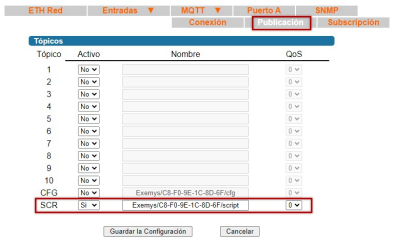


Device Connection via MQTT

Both the device and the Script Programmer must be connected to the same broker to send and receive scripts remotely via MQTT broker.

Device configuration

The SCR base topic used for publishing and subscribing must be configured. The device will append a text to the base as explained later.



Configuration topic - Device identification

The device will append the ClientID to the base configuration topic. The resulting subscribe and publish topics in the previous example will be:

Subscribe -> *Exemys/C8-F0-9E-1C-8D-6F/script/1/clientID*

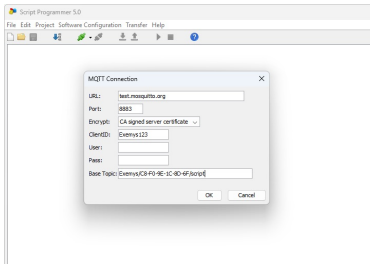
Publish -> *Exemys/C8-F0-9E-1C-8D-6F/script/0/clientID*

The clientID used by the device will be the one configured in the MQTT section.

ScriptProgrammer configuration

The broker parameters to connect to and the base topic to use for configuring remote devices must be loaded in the configurator.

To do this, go to *"Software Configuration -> MQTT Connection"*

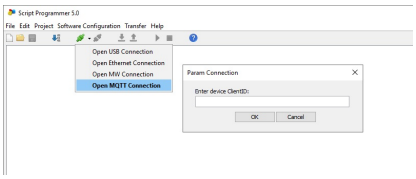


The *basic topic* must match the one configured on the devices.

Remote connection

Then expand the Connect button and choose ***"Open MQTT Connection"***

There you must enter the ClientID of the device to configure.



Once the connection is established, the "Download", "Upload", "Start" and "Stop" buttons can be used as if connected locally to the device.

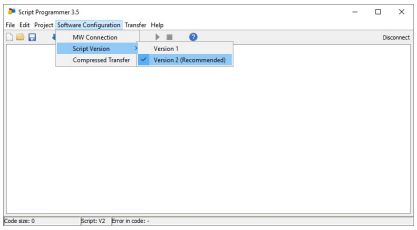
Script Versions 1, 2 and 3

In the ***"Project"*** menu, option ***"Properties"***, tab ***"Script"*** you can select between script version 1, 2 and 3.

Version 2 differs from version 1 in that it allows twice as many variables since they can be lowercase or uppercase.

All these devices work with version 2 or 3. From version 11.0 onwards they automatically switch to version 3. Programs written in version 3 are identical to those in version 2. They only take up less space in the device memory.

Version 2/3 differs from version 1 in that it allows twice as many variables since they can be lowercase or uppercase.

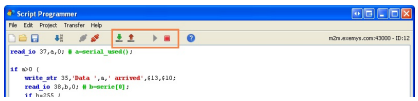


The script version will be used by Script Programmer in two situations: when **verifying** a script or when trying to **send it** to the device.

If when sending a script the selected version is not compatible with the target device, you will see a message indicating that error.

Upload and download of scripts

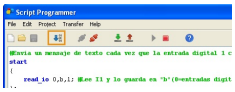
Once connected to the device, we can transfer and download scripts. We can also stop and start them.



Script editing

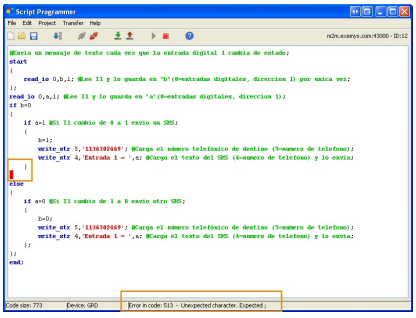
To develop a program, simply write the code in the editing panel. The environment provides autocomplete assistance for functions and highlights them to indicate correct syntax.

Once you finish writing the code, use the ***"Verify"*** button to compile the program and check for syntax errors.



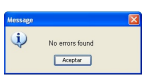
At compile time, the bottom panel will indicate whether there are errors. If there is an error, the line will be highlighted in red and the ***"Error in Code:"*** field will show the line number. If there are no errors, a message will appear showing ***"Error in Code: NONE"***.

The following image shows a program with an error; in this case a ***";"*** is missing.

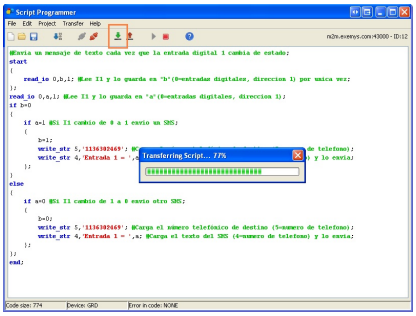


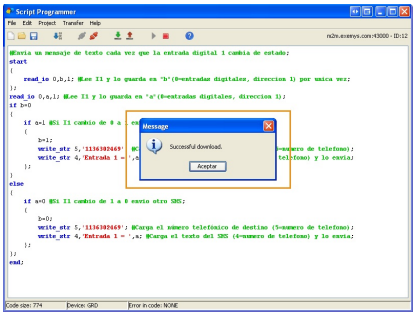
The semicolon is missing in the brace before the one marked in red; this is because the compiler indicates it found an unexpected character.

The following image shows a program without errors.



If there are no errors, we can transfer the program to the device by clicking ***"Download to device"***. A window will appear showing the download status and then a message indicating whether the download was successful or not.





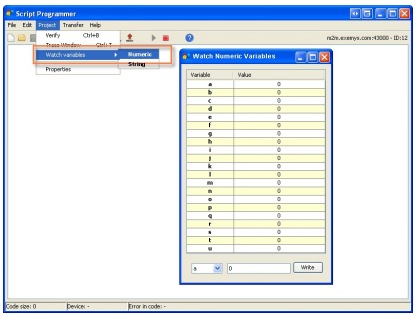
Script debugging

The Script Programmer has two tools for debugging written scripts.

Variable monitoring

With this tool you can view the value of numeric or text variables while the program is running. You can also modify the value of variables to simulate script working conditions.

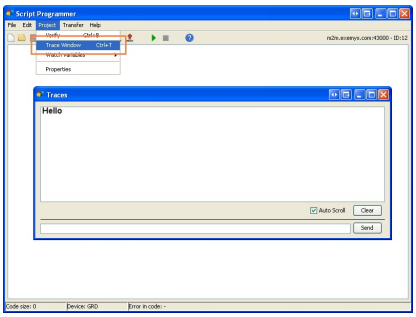
Once connected to the device, go to the **"Project"** menu, option **"Watch variables"** and then **"Numeric"** or **"String"** depending on the type of variable to monitor.



Transmission and reception of traces

With this tool you can send text from the script to the Script Programmer to follow the script's operation. Text can also be sent to simulate script working conditions.

Once connected to the device, go to the **"Project"** menu, option **"Trace Window"**



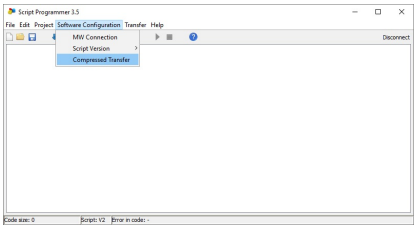
Script compression

The device has limited space for storing scripts. In almost all devices the maximum is 15,000 characters. There are some cases where it is limited to 5,000.

If this space is not sufficient for your application, you can use the script compression option. When enabled, the Script Programmer will remove all comments and tabs before sending the program.

If you want to keep the comments in your program, save a copy on your computer.

To enable compression, go to the **"Software Configuration"** menu, option **"Compressed Transfer"**



2026-05-05

Introduction

The ***Exemys Script*** programming language is a loop-type language, meaning it executes all code up to the last line and then starts again.

There are no statements to create loops within the program, so the program flow cannot be stopped at a specific code section. In this regard it resembles PLC programming, although its syntax is closer to C or Pascal.

Script Versions 1, 2 and 3

There are 3 script versions. Version 2 differs from version 1 in that it allows twice as many variables since they can be lowercase or uppercase. In version 1 variables are only written in lowercase. Version 3 saves between 20 tond 30% of the space occupied by the program in the device memory, allowing longer programs to be loaded. You can notice the difference by pressing the verify button.

Variables

There are 2 types of variables: numeric of type ***"long"*** and text of type ***"string"***.

It is not necessary to define variables since there is a fixed number.

In script version 2, numeric variables are 42, from ***"a"*** to ***"u"*** and from ***"A"*** to ***"U"***. Text variables are 10, from ***"v"*** to ***"z"*** and from ***"V"*** to ***"Z"***, with a maximum length of 100 characters each.

Since numeric variables are of type "long", any operation that yields a decimal result will be truncated.

The initial value of numeric variables is 0; for text variables it is an empty string.

Assign a value to a variable:

- Numeric variables:

```
a = 652;
```

- Text variables:

```
v = 'Hello'
```

Note that to assign text, it must be placed in single quotes.

String concatenation:

To concatenate variables, simply place them one after the other, separated by commas.

For example:

```
a = 20;
u = 'The temperature is: ';
v = ' °C';
```

If we want to form the string 'The temperature is 20 °C ' and store it in another variable we do the following:

```
w = u, a, v;
```

Another way to do it would be:

```
w = 'The temperature is: ', a,' °C';
```

Concatenation can only be done in text assignment to string variables and in the write_str function

Insertion of ASCII values in strings:

To insert an ASCII value in a string, the $ operator can be used. After the operator, the ASCII code in decimal must be specified. ASCII 0 is not allowed.

For example:

```
z = 'Hello',$13,$10;
```

ASCII value insertion can only be done in text assignment to string variables and in the write_str function

Variable Aliases

From version **6.1** of the Script Programmer, variable "aliases" can be created to make the code easier to read.

This code can be transferred to ALL Exemys devices that support Script Programmer, since before sending the code to the device, the alias is replaced by the corresponding variable.

Since aliases are sent as comments within the code, when reading the script loaded on the device, the variables will be replaced back by the aliases, unless the compression option was used, which removes comments.

It is ALWAYS recommended to keep a copy of the scripts loaded on the devices.

Example:

Before

```
read_io 2,a,1;
if a>b {
write_io 1,8,1;
};
end;
```

Now

```
#[a=SP1];
#[b=presion];
read_io 2,presion,1;
if presion>SP1 {
write_io 1,8,1;
};
end;
```

If you already have a written script, you can add comments indicating the aliases, send it to the device (uncompressed) and then read it back. When doing so, all variables with aliases will be replaced.

Arithmetic operators

Below are the operators that can be used in the script. Note that it only handles "long" type values, so any operation yielding decimal results will be truncated - only the integer part will be returned.

Operator	Meaning
=	Equals
^	Exponential
|	Or
&	And
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

For example:

```
a = 130;
b = a+5;
```

Resultado: La variable *b* vale 135.

Program structure

All instructions end with a semicolon ***";"***.

It is very important to keep in mind that the program runs in a loop: when the last statement is reached, it starts over. Because of this, there is a statement called ***"start"*** that runs only once when the program starts. Inside this block, initial values for variables or any other initialization can be placed.

Every program must end with the instruction ***"end;"***.

Below is the typical structure of a program:

```
start
{
    a = 10;
};
a = a + 1;
end;
```

This simple example shows how the ***"start"*** function is used to initialize variable ***"a"*** to 10 only once and then increment it by 1 continuously.
As we can see, all instructions after the ***"start"*** block and before ***"end;"*** are executed cyclically.

Comments:

To add a comment to the program, prepend the text with the ***"#"*** character and end it with ***";"***.

For example:

```
start
{
    a = 10;  #Initialize variable a = 10;
};
a = a + 1;    #Increment variable a;
end;
```

Flow control functions

Flow control specifies the order in which certain lines of code will be executed.
In our case there are 3 functions for flow control: ***"start"***, ***"if-else"*** and ***"end"***.

Function ***"start"***

Used to execute lines of code only once when the program starts.
The instructions to execute are placed between braces, and after the closing brace ***";"*** must be placed.

Syntax:

```
start
{
...;
...;
};
```

Function ***"if-else"***:

This function is used for decision-making. If the condition placed after ***"if"*** is valid, the instructions in the braces below will execute; otherwise the instructions inside ***"else"***.

The execution condition can use the following operators:

Operator	Function
=	Equal
!	Different
	Greater

>	thatn
<	Less than

The condition is written leaving a space after ***"if"***

Syntax:

Only ***"if"***:

```
if condition
{
   ...;
   ...;
};
```

After the closing brace of ***"if"*** a ***";"*** must be placed.

"if-else":

```
if condition
{
   ...;
   ...;
}
else
{
   ...;
   ...;
};
```

In this case, the ***";"*** is placed after the closing brace of ***"else"***, not after ***"if"*** .

Function ***"end"***

Used only to indicate the end of the program; after executing it, the program returns to the first line of code.

Syntax:

```
end;
```

Interface functions (sources/destinations)

These functions allow reading data from various sources and sending data to various destinations.

Function ***"read_io"***

Allows reading indexed parameters from various sources, such as values of digital input channels, digital output channels, analog inputs, etc.

The data "source" must be indicated with a number. If applicable, the "index" parameter specifies the position within that source.
The reading result is stored in a numeric variable.

Syntax:

```
read_io source,numeric_variable,index;
```

The available sources depend on the **device** where the script runs and the script version. New sources may be added in the future.
Refer to the sources/destinations section for more details.

Function ***"write_io"***

Allows writing indexed parameters to various destinations, such as values of digital output channels, pulse outputs, etc.

The "destination" where to write must be indicated with a number. If applicable, the "index" parameter specifies the position within that destination.

The "value" to write can be a number or a numeric variable.

Syntax:

```
write_io destination,index,value;
```

The available destinations depend on the **device** where the script runs and the script version. New destinations may be added in the future.
Refer to the sources/destinations section for more details.

Function ***"read_str"***:

Allows reading strings from various sources, such as data received from a serial port or by SMS. The ***"source"*** must be indicated with a number.

The reading result is stored in two variables: one of type string and one of type numeric, indicating the string length. If there is no data, this parameter will be 0.

Syntax:

```
read_str source,numeric_variable,string_variable;
```

The available sources depend on the **device** where the script runs and the script version. New sources may be added in the future.
Refer to the sources/destinations section for more details.

Function ***"write_str"***:

Allows writing strings to various "destinations", such as the serial port or SMS. The ***"destination"*** must be indicated with a number and the string to send.

Syntax:

```
write_str destination,string;.
```

The available destinations depend on the **device** where the script runs and the script version. New destinations may be added in the future.
Refer to the sources/destinations section for more details.

The string can be a variable or a string written in the function. **This function supports concatenation and inclusion of ASCII values.**

String functions

These functions allow performing various operations on string-type program variables.

Function ***"is_equal"***

Compares a string variable with a string (fixed text or string variable). Returns 0 if they differ and 1 if they are equal.

Syntax:

```
is_equal numeric_variable,string_variable,string;
```

Example:

```
v='Turn off';
is_equal c,v,'Turn off';
if c=1 {
        #The texts are equal;
};
```

Function ***"finish_with"***

Determines if a string variable ends with a particular string (fixed text or string variable). Returns 0 if false or 1 if true

Syntax:

```
finish_with numeric_variable,string_variable,string;
```

Example:

```
v='Turn off';
finish_with c,v,'off';
if c=1 {
        #The string stored in v ends with 'off';
};
```

Function ***"begin_with"***

Determines if a string variable begins with a particular string (fixed text or string variable). Returns 0 if false or 1 if true

Syntax:

```
begin_with numeric_variable,string_variable,string;
```

Example:

```
v='Turn off';
begin_with c,v,'Turn';
if c=1 {
        #The string stored in v starts with 'Turn';
};
```

Function ***"contains"***:

Determines if a string variable contains a particular string (fixed text or string variable). Returns 0 if not found, or the position where it was found if found.

Syntax:

```
contains numeric_variable,string_variable,string;
```

Example:

```
v='Turn off';
contains c,v,'off';
if c>0 {
        #The string stored in v contains the text 'off';
};
```

Function ***"upper"***

Converts all characters of a string variable to uppercase, storing the result in the same variable.

Syntax:

```
upper string_variable;
```

Example:

```
v='Off';
upper v;
#Now v equals 'OFF';
```

Function *"lower"*

Converts all characters of a string variable to lowercase, storing the result in the same variable.

Syntax:

```
lower string_variable;
```

Example:

```
v='Off';
lower v;
#Now v equals 'off';
```

Function *"strlen"*

Returns in a numeric variable the length of a *string* variable.

Syntax:

```
strlen numeric_variable,string_variable;
```

Example:

```
v='Apagar';
strlen c,v;
#Now c equals 6;
```

Function *"substr"*

Returns a part of a string variable within the same variable

Syntax:

```
substr start,end,string_variable;
```

Example:

```
v='APAGAR BOMBA';
substr 2,3,v;
#Now v equals 'PA';
```

Conversion functions

These functions convert variables from one type to another.

Function *"point"*

Places the decimal point on a numeric variable and converts it to a *string*.
Parameters are: the numeric variable, the string variable, and the number of decimal places.

Syntax:

```
point string_variable,numeric_variable,decimal_places;
```

Example:

```
c=123;
point v,c,1;
#Now v equals '12.3';
```

Function *"aton"*

Converts a number within a string to a numeric variable. Starts at the beginning of the string and ends where a non-numeric value or end of string is found.

Syntax:

```
aton numeric_variable,string_variable;
```

Example:

```
v='123 RPM';
aton c,v;
#Now c equals 123;
```

Funciones *"day","month","year","hs","min","sec"* y *"nday"*

These functions convert a ***time_stamp*** to day, month, year, hour, minute, second and day of week number, respectively.
As seen previously, the current date/time is read with the ***read_io*** type **7** function and stored in a numeric variable representing the number of seconds since 01/01/2000.

Syntax:

```
day day,timestamp;

month month,timestamp;

year year,timestamp;

hs hour,timestamp;

min minutes,timestamp;

sec seconds,timestamp;

nday day_of_week,timestamp;
```

The ***"nday"*** function returns the day of the week number starting from Sunday = 0.

Example:

```
read_io 7,e,0; #Read the current time into e;
day f,e;
month g,e;
year h,e;
hs i,e;
min j,e;
sec k,e;
#The current date and time is f/g/h i:j:k;
```

Mathematical and logical functions

These functions perform certain special mathematical operations.

Function *"neg"*

Used to negate a numeric variable. Negation is performed at the bit level.

Syntax:

```
neg resultado,inicio;
```

First place the variable in which to obtain the result, then the variable to negate.

Example:

```
a=32323; #7E43h
neg b,a;

# b vale 4294934972 (FFFF81BC);
```

Function *"sqtr"*

Performs the square root. Since Exemys Script only handles integer variables of type "long", the decimal part of the result will be truncated. To avoid this, it is recommended to multiply before performing the operation.

Syntax:

```
sqrt result,base;
```

Example:

```
a=225;
sqrt b,a;
#Now b equals 15;
```

Function *"scale"*

Allows scaling a variable using the equation of a line passing through 2 points.

Syntax:

```
scale resultado,inicio,x0,x1,y0,y1;
```

Example: Scale the 4-20mA signal from input AN1 to a value from 0 to 500

```
read_io 2,a,1; #a = AN1;
scale c,a,400,2000,0,500;
#Now c holds the scaled value;
```

Timer functions

These functions allow controlling program flow based on time intervals.

Functions *"timer"* and *"check_timer"*

These functions allow defining a time period and executing instructions when it elapses.
The usage is as follows: first, execute the ***"timer"*** function, passing a numeric variable and a time in milliseconds as parameters. Then query that variable with the ***"check_timer"*** function.

Syntax:

```
timer numeric_variable,time_in_milliseconds;
...
check_timer numeric_variable
{
    ...
    ...
```

```
};
```

As we can see, with the ***"timer"*** function we define the time period, and with ***"check_timer"*** we query the previously loaded variable. When the period elapses, the instructions placed between braces execute.
Keep in mind that once the period elapses, the condition will always be true, and if ***"check_timer"*** is called again, the instructions will execute again. The variable is normally reloaded inside the ***"check_timer"*** block.

Note: Timer functions should not be used in time-precision applications, as the timer may exhibit some jitter.

2026-05-05

Introduction

Using the read_io, write_io, read_str and write_str functions, multiple additional LRlink-LoRa functions can be accessed. This section of the manual lists the different sources/destinations and then explains their use by function.

Source/destination list

Source / Destination	Index	Value	R/W	Description	Function	Firmware
0	1 to 4	0/1	read_io	Digital input	I/O	10.0
1	1 to 20	0/1	read/write_io	Digital output		
2	1 to 8	400 to 2000 - 0 to 1000	read_io	Analog input		
3	1 to 4	0 to 999.999.999	read/write_io	Pulse counter		
4	1 to 2	400 to 2000 - 0 to 1000	read/write_io	Analog output		
5	4	1 -> COM A	write_io	Enables COM N in Script mode (in start, write_io 5,4,1; COM A script)	Serial	
402	12	0	write_io	Serial transmit buffer - Select and initialize	Serial Transmission	
405		0	write_io	Serial transmit buffer - Send binary data loaded with 403/404		
405		-	write_str	Serial transmit buffer - Load and send text (select buffer first)		
402	13	0 a 199	write_io	Serial receive buffer - Select and position	Serial Reception	
405		0 o N	write_io	Serial receive buffer - Empty into the buffer (0 = All)		
405	-	0 to 200	read_io	Serial receive buffer - Number of bytes (select buffer first)		
405		-	read_str	Receive text (select buffer first)		
11	0	2 -> Connected (9 en wRemote-LoRa)	read_io	Node connection status	LoRa	
5	6	0 / 1	write_io	Disable automatic report transmission	LoRa Reporte I/O	
17	0	0	write_io	Force DI report		
18	0	0	write_io	Force DO report		
19	0	0	write_io	Force AI report		
23	0	0	write_io	Force MB report		
402	8	0	write_io	LoRa transmit buffer - Select and initialize	LoRa Transmission	
405		0	write_io	LoRa transmit buffer - Send binary data loaded with 403/404		
405		0	read_io	Transmission status 0-Idle / 1-Sending / 2-Sent / 3-Failure		
402	9	0 a 199	write_io	LoRa receive buffer - Select and position	LoRa Reception	
405		0	write_io	LoRa receive buffer - Empty into the buffer (0 = All)		
403	1	10^N -2 -> 0.01 -1 -> 0.1 **0 -> 1** 1 -> 10 2 -> 100	write_io	Buffers - Configure exponent for floating-point values	Serial/LoRa	
	2	**0-> MSB** / 1 ->LSB **0-> MSW** / 1 ->LSW	write_io	Buffers - Configure byte/word order		
404	1	Number according to format, exponent, and order	read/write_io	Buffers - Unsigned 8-bit integer		
	2		read/write_io	Buffers - Signed 8-bit integer		
	3		read/write_io	Buffers - Unsigned 16-bit integer		
	4		read/write_io	Buffers - Signed 16-bit integer		
	6		read/write_io	Buffers - Signed 32-bit integer		
	7		read/write_io	Buffers - 32-bit floating-point value		
5	5	Minutos	write_io	Put to sleep for X minutes	Low Power	
35		-	read/write_str	Script Programmer traces	Traces	
21	1 to 20	0 to 2147483647	read/write_io	Non-volatile memory for numbers	Mem. Non-volatile	
121 to 125		-	read/write_str	Non-volatile memory for text 1 to 5		

2026-05-05

Input/output read/write operations

Source / Destination	Index	Value	R/W	Description	Function
0	1 to 4	0/1	read_io	Digital input	I/O
1	1 to 2	0/1	read_io/write_io	Digital output	
2	1 to 4	400 to 2000 - 0 to 1000	read_io	Analog input	
3	1 to 4	0 to 999.999.999	read/write_io	Pulse counter	
4	1 to 2	400 to 2000 - 0 to 1000	read/write_io	Analog output	

Sources 0 to 2 return the values of the different inputs/outputs. The **"index"** indicates the input number to read.

For analog signals, the value ranges from 400 to 2000 for inputs configured in current mode, and from 0 to 1000 for inputs in voltage mode.

For digital signals, the value will be 0 or 1 indicating off or on.

Example: Read the value of analog input 4 and store it in variable c

```
read_io 2,c,4;
```

Example: Set analog output 2 to 12 mA

```
write_io 4,2,1200;
```

2026-05-05

Serial Port - Text Mode

From the script, data can be sent and received in text mode.

To send and receive binary data, read "Serial Port - Binary Mode"

Source / Destination	Index	Value	R/W	Description	Function
5	4	1 -> COM A	write_io	Enables COM N in Script mode (in start, write_io 5,4,1; COM A script)	Serial
402	12	0	write_io	Serial transmit buffer - Select and initialize	Serial Transmission
405	-		write_str	Serial transmit buffer - Load and send text (select buffer first)	
402	13	0 a 199	write_io	Serial receive buffer - Select and position	Serial Reception
405		0 o N	write_io	Serial receive buffer - Empty into the buffer (0 = All)	
405	-	0 to 200	read_io	Serial receive buffer - Number of bytes (select buffer first)	
405	-		read_str	Receive text (select buffer first)	

Send example: Port initialization and transmission of text 'HOLA'

```
start {
  write_io 5,4,1; #Configure COM A in script mode;
};

if a=0 {
  a=1;
  write_io 402,12,0; #Select serial buffer;
  write_str 405,'HOLA'; #echo write;
};
end;
```

Receive example: Receive text characters until a 'CR' (ASCII 13) is found and store the text in string variable v

```
start {
  write_io 5,4,1; #Configure COM A in script mode;
};

write_io 402,13,0; #Select serial RX buffer;
read_io 405,b,0; #Received count;

if b!0 {
  read_str 405,c,z; #Read text in buffer;
  if c!0{
    y=$13; contains d,z,y; #Search for CR;
    if d!0{
      v=z; #Store the received text in v;
      write_io 402,13,0; #Select serial RX buffer;
      write_io 405,13,d; #delete everything read up to CR;
    };
  };
};
end;
```

2024-10-14

Serial Port - Binary Mode

From the script, data can be sent and received in binary mode.

To send and receive text, read "Serial Port - Text Mode"

Sources/destinations 402, 403, 404 and 405 used to access the buffer may be shared by other functions.

Source / Destination	Index	Value	R/W	Description	Function
5	4	1 -> COM A	write_io	Enables COM N in Script mode (in start, write_io 5,4,1; COM A script)	Serial
402	12	0	write_io	Serial transmit buffer - Select and initialize	Serial Transmission
405		0	write_io	Serial transmit buffer - Send binary data loaded with #03/404	
402	13	0 a 199	write_io	Serial receive buffer - Select and position	Serial Reception
405		0 o N	write_io	Serial receive buffer - Empty the buffer (0 = All)	
405	-	0 to 200	read_io	Serial receive buffer - Number of bytes (select buffer first)	
403	1	10^N -2 -> 0.01 -1 -> 0.1 **0 -> 1** 1 -> 10 2 -> 100	write_io	Buffers - Configure exponent for floating-point values	Binary buffer
	2	**0-> MSB** / 1 ->LSB **0-> MSW** / 1 ->LSW	write_io	Buffers - Configure byte/word order	
404	1	Number according to format, exponent, and order	read/write_io	Buffers - Unsigned 8-bit integer	
	2		read/write_io	Buffers - Signed 8-bit integer	
	3		read/write_io	Buffers - Unsigned 16-bit integer	
	4		read/write_io	Buffers - Signed 16-bit integer	
	6		read/write_io	Buffers - Signed 32-bit integer	
	7		read/write_io	Buffers - 32-bit floating-point value	

Send example: Port initialization and transmission of a binary frame with numbers in different formats.

```
start {
  write_io 5,4,1; #Configure COM A in script mode;
};

if a=0 {
  a=1;

  write_io 402,12,0; #Select binary buffer;

  write_io 403,2,0;#MSB;
  write_io 404,4,-3000; #SINT16;
  write_io 404,4,1000; #SINT16;
  write_io 403,2,0;#MSW;
  write_io 404,6,-70000; #SINT32;
  write_io 404,6,70000; #SINT32;
  write_io 403,2,1;#LSW;
  write_io 404,6,-70000; #SINT32;
  write_io 404,6,70000; #SINT32;
  write_io 403,2,1;#LSB;
  write_io 404,4,1000; #SINT16;
  write_io 403,2,0;#MSB;
  write_io 404,4,1000; #SINT16;

  write_io 403,2,0;#MSW;
  write_io 403,1,2;#10^2; #/100;
  write_io 404,7,123456; #FLOAT32;
  write_io 403,1,-2;#10^-2; #*100;
  write_io 404,7,123456; #FLOAT32;
  write_io 403,2,1;#LSW;
  write_io 403,1,2;#10^2; #/100;
  write_io 404,7,123456; #FLOAT32;
  write_io 403,1,-2;#10^-2; #*100;
  write_io 404,7,123456; #FLOAT32;

  write_io 405,12,0; #Send loaded data;
};
end;
```

These bytes are output on the serial port (expressed in hexadecimal)

F4 48 03 E8 **FF FE EE 90** 00 01 11 70 **EE 90 FF FE** 11 70 00 01 **E8 03** 03 E8 **44 9A 51 EC** 4C 3C 61 00 **51 EC 44 9A** 61 00 4B 3C

Receive example: Port initialization and reception of 39-byte frames with values in different formats. The transmitted data are shown below in hexadecimal.

F4 48 03 E8 **FF FE EE 90** 00 01 11 70 **EE 90 FF FE** 11 70 00 01 **E8 03** 03 E8 **44 9A 51 EC** 4C 3C 61 00 **51 EC 44 9A** 61 00 4B 3C

12 34 56 78 90

The received data are stored in different script variables. These are the results

B=-3000
C=1000
D=-70000
E=70000
F=-70000
G=70000
H=1000
I=1000
J=123456
K=123456
L=123456
M=123456

```
start {
  write_io 5,4,1; #Configure COM A in script mode;
};
write_io 402,13,1; #Select binary buffer;

read_io 405,A,0; #Read how many bytes are in the buffer;
if A>39{
  write_io 403,2,0;#MSB/MSW;
  write_io 402,13,1; read_io 404,B,4; #SINT16;
  write_io 402,13,3; read_io 404,C,4; #SINT16;
  write_io 402,13,5; read_io 404,D,6; #SINT32;
  write_io 402,13,9; read_io 404,E,6; #SINT32;
  write_io 403,2,1;#LSB/LSW;
  write_io 402,13,13; read_io 404,F,6; #SINT32;
  write_io 402,13,17; read_io 404,G,6; #SINT32;
  write_io 402,13,21; read_io 404,H,4; #SINT16;
  write_io 403,2,0;#MSB/MSW;
  write_io 402,13,23; read_io 404,I,4; #SINT16;
  write_io 402,13,25; write_io 403,1,2; read_io 404,J,7; #10^2 /100 FLOAT32;
  write_io 402,13,29; write_io 403,1,-2; read_io 404,K,7; #10^-2 *100 FLOAT32;
  write_io 403,2,1;#LSB/LSW;
  write_io 402,13,33; write_io 403,1,2; read_io 404,L,7; #10^2 /100 FLOAT32;
  write_io 402,13,37; write_io 403,1,-2; read_io 404,M,7; #10^-2 *100 FLOAT32;
  write_io 405,13,0; #Empty the entire reception buffer;
};
end;
```

2025-09-30

LoRa status and communication

From the script, input/output report transmission can be forced without waiting for the regular criteria.

Custom-formatted *payloads* can also be sent and received.

Connection status

Source 11 lets you know whether the device has already joined the configured gateway. 9 indicates connected.

Source / Destination	Index	Value	R/W	Description	Function
11	0	2 -> Connected (9 en wRemote-LoRa)	read_io	Node connection status	LoRa

Example: Read the connection status

```
read_io 11,g,0;
```

I/O report

If you want to send reports of I/O values, they can be forced from the script with destinations 17, 18, 19 and 23. The reported value is the I/O value itself.

Additionally, the automatic report sent by the device after performing a JOIN can be disabled.

Source / Destination	Index	Value	R/W	Description	Function
5	6	0 / 1	write_io	Disable configured report transmission	LoRa I/O reports
17	1 to 4	-	write_io	Force DI report	
18	1 to 4	-	write_io	Force DO report	
19	1 to 4	-	write_io	Force AI report	
20	1 to 4	-	write_io	Force AO report	
22	1 to 2	-	write_io	Force PI report	
23	1 to 10	-	write_io	Force MB report	

Example: Send a report of the analog inputs. (Reports the last measured value)

```
write_io 19,0,0;
```

Sending/Receiving custom LoRa *payloads*

From the script, custom-formatted *payloads* can be sent and received.

As a first step, it is recommended to disable automatic report transmission after JOIN with `write_io 5,6,1;`

Sources/destinations 402, 403, 404 and 405 used to access the buffer may be shared by other functions.

Source / Destination	Index	Value	R/W	Description	Function
5	6	0/ 1	write_io	Disable automatic report transmission	LoRa Transmission
402	8	0	write_io	LoRa transmit buffer - Select and initialize	
405		0	write_io	LoRa transmit buffer - Send binary data loaded with 403/404	
405		0	read_io	Transmission status 0-Idle / 1-Sending / 2-Sent / 3-Failure	
402	9	0 to 199	write_io	LoRa receive buffer - Select and position	LoRa Reception
405		0	write_io	LoRa receive buffer - Empty into the buffer (0 = All)	
403	1	10^N -2 -> 0.01 -1 -> 0.1 **0 -> 1** 1 -> 10 2 -> 100	write_io	Buffers - Configure exponent for floating-point values	LoRa
	2	**0-> MSB** / 1 ->LSB **0-> MSW** / 1 ->LSW	write_io	Buffers - Configure byte/word order	
404	1	Number according to format, exponent and order	read/write_io	Buffers - Unsigned 8-bit integer	
	2		read/write_io	Buffers - Signed 8-bit integer	
	3		read/write_io	Buffers - Unsigned 16-bit integer	
	4		read/write_io	Buffers - Signed 16-bit integer	
	6		read/write_io	Buffers - Signed 32-bit integer	
	7		read/write_io	Buffers - 32-bit floating-point value	

Binary-mode send example

It is possible to send custom payloads.

Before sending, the transmit buffer must be selected, then the data loaded with destinations 403 and 404, and finally transmission triggered. Then it can be checked whether the transmission was successful or not.

Example: All these steps are shown below.

```
read_io 11,a,0;#e=Connection status;
if a = 9 {
  write_io 402,8,1; #Buffer Tx LoRa;
  write_io 404,2,10;#Carga en buffer el numero 10 en formato entero de 8 bits con signo;
  write_io 403,2,0; #Configura orden MSB;
  write_io 404,4,23456;#Carga en buffer el numero 23456 en formato entero de 16 bits con signo y orden MSB;
  write_io 405,8,0; #Send LoRa buffer;
};

....

read_io 405,a,8; #d=transmission status;
if a = 2 {
  #Transmission exitoso ;
};
```

The transmitted payload will contain these bytes (expressed in hexadecimal)

0A `5B A0`

Binary-mode receive example

It waits for payloads sent from the gateway. Once received, it reads some of the bytes that compose it.

The payload sent from the gateway contains these bytes (expressed in hexadecimal)

0A `5B A0`

```
write_io 402,9,0; #Select LoRa RX buffer;
read_io 405,a,9; #Read number of received bytes;
if a!0 {
   write_str 35,'RX LORA: ',a;
   write_io 402,9,1; read_io 404,b,2; #b=frame's first by - signed int format;
   write_io 403,2,0;#MSB;
   write_io 402,9,2; read_io 404,c,4; #c=frame's bytes 2 and 3 - MSD 16 bit signed int;
   write_io 405,9,0; #Clear the receive buffer;
};
end;
```

After running the script, 'b' will be 10 tond 'c' will be 23456.

2026-05-05

Low Power

Source / Destination	Index	Value	R/W	Description	Function
5	5	Minutos	write_io	Put to sleep for X minutes	Low Power

Destination 5,5 allows putting the device into low power mode for X minutes.

Note that after that time, the device will operate as if it had just been powered on, resetting all script variables to 0.

While in low power mode all indicator LEDs will turn off.

If the USB port is connected to the PC, the device will not enter low power mode until it is disconnected.

The device can also be awakened by connecting the USB port.

2026-05-05

Transmission/Reception of messages to Script Programmer ("Traces")

Source /Destination	R/W	Description	Function
35	read/write_str	Script Programmer Traces	Traces

Destination 35 allows sending text to the "Traces" window in the Script Programmer. This is particularly useful when testing a new script.

There is a consideration regarding space and underscore characters. The space character will be replaced by an underscore before reading with *read_str 35*. The underscore will be replaced by a space after sending with *write_str 35*.

If the Script Programmer is not connected when writing to destination 35, the text will simply be lost but will not affect the operation of the program.

2024-04-23

Non-volatile memory read/write

The device can store text or numbers in non-volatile memory to preserve values even when the device is powered off.

Source / Destination	Index	Value	R/W	Description	Function
21	1 to 20	0 to 2147483647	read/write_io	Non-volatile memory for numbers	Mem. Non-volatile
121 to 125	-		read/write_str	Non-volatile memory for text 1 to 5	

For numbers

Source/destination 21 allows reading and writing numeric values in the device's non-volatile memory.

Example: Read the numeric value stored at position 15 of non-volatile memory into variable g

```
read_io 21,g,15;
```

For text

Sources/destinations 121 to 125 allow reading and writing up to 5 texts in the device's non-volatile memory.

Example: Write the word 'hello into the third position of non-volatile text memory.

```
write_str 123,'hello';
```

2026-05-05