Indice

SCIENCE GATE
Your Automation Partner

SCIGATE AUTOMATION (S) PTE LTD

No 1 Bukit Batok Street 22 #01-01 Singapore 659592
Tel: (65) 6561 0488 Fax: (65) 6561 0588
Email: sales@scigate.com.sg Web: https://scigate.com.sg/

Business Hours: Monday - Friday 8:30AM - 6:15PM

Script Programming

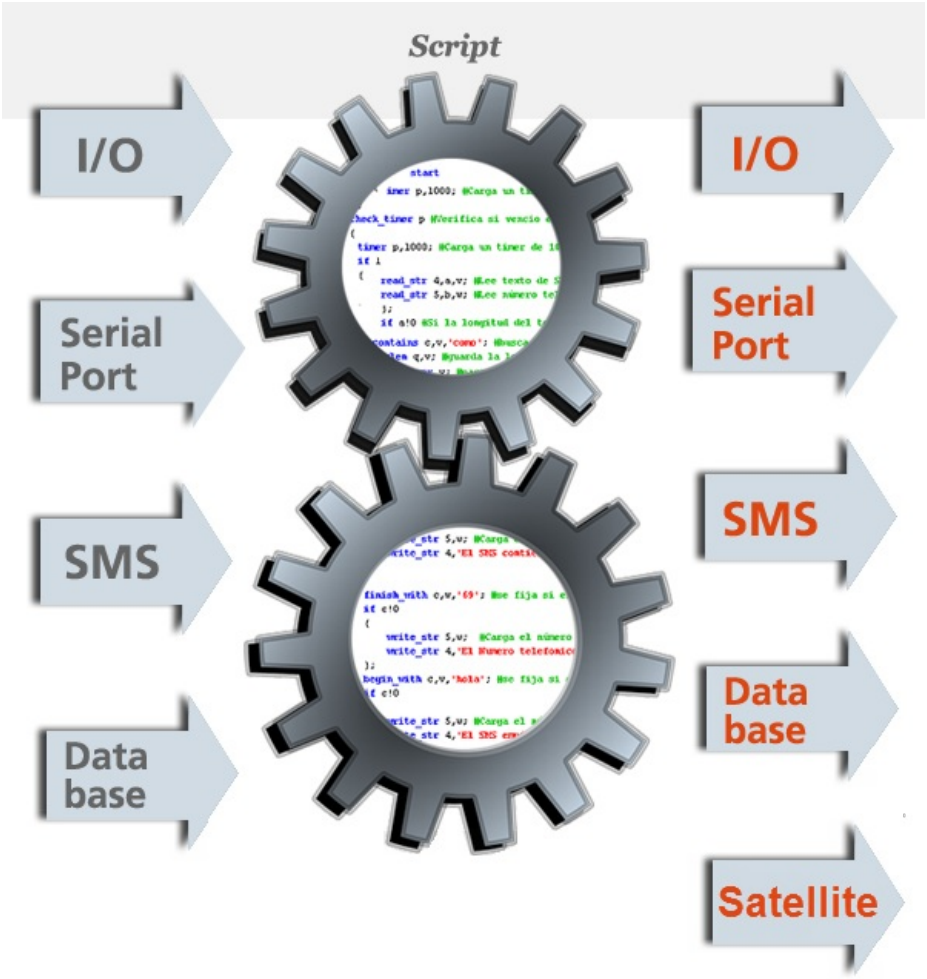
Description

The GRD/cLAN with script programming support allows you to run user written scripts on the device, making it more powerful and flexible.

The GRD/cLAN will continue working normally while the script is running.

Script Features

- **Math** operations
- **Logic** operations
- **Timing** functions
- Physical and Modbus channels readings
- Digital outputs control
- **SMS** sending and receiving (GRD only)
- **Serial port** data parsing
- Sending and receiving data using external **Satellite** modem

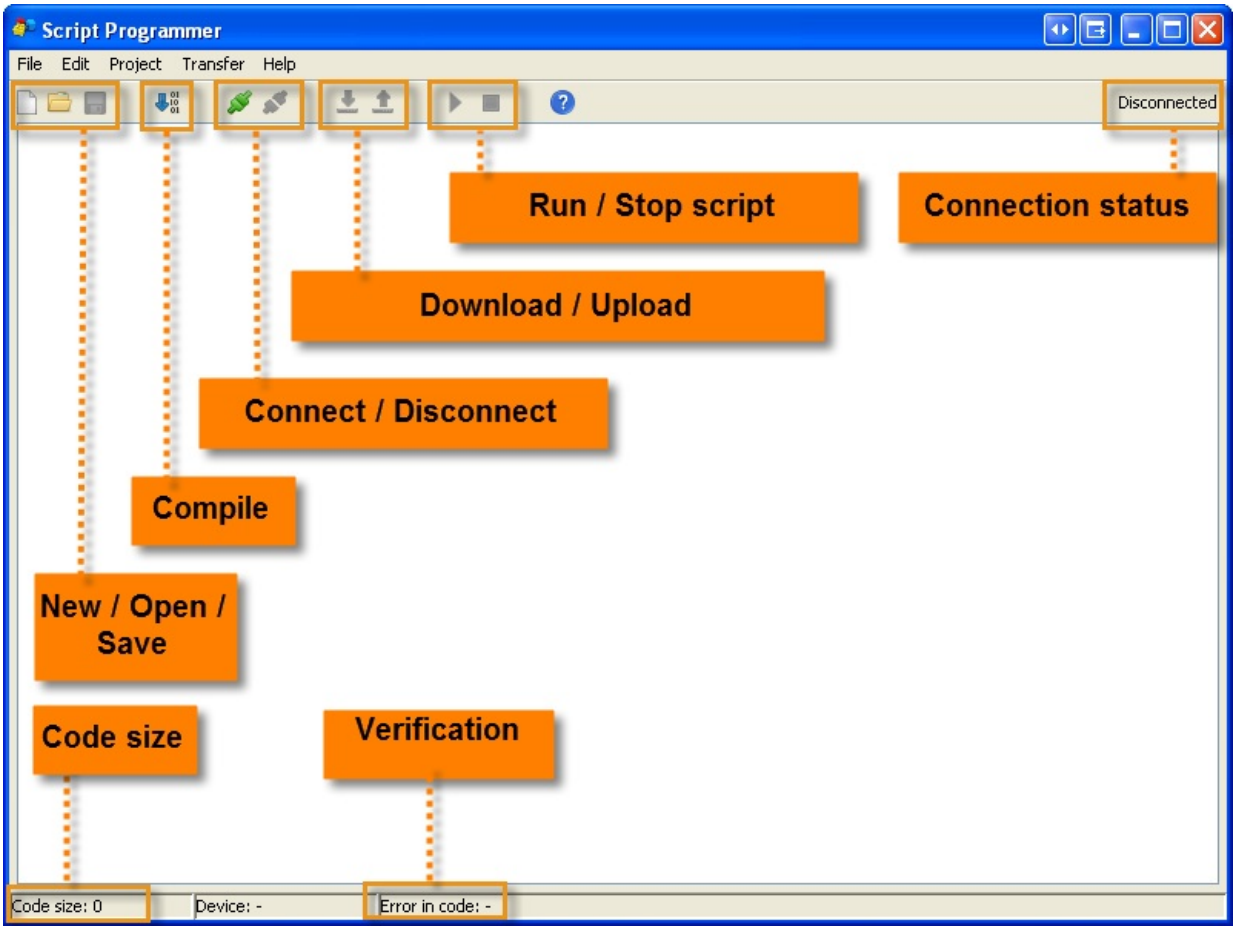


2017-06-14

Introduction

This software is used to write, compile and download the user' scripts to the GRD/cLAN. Before using it please check that the GRDconfig software is able to talk to the device.

Software description



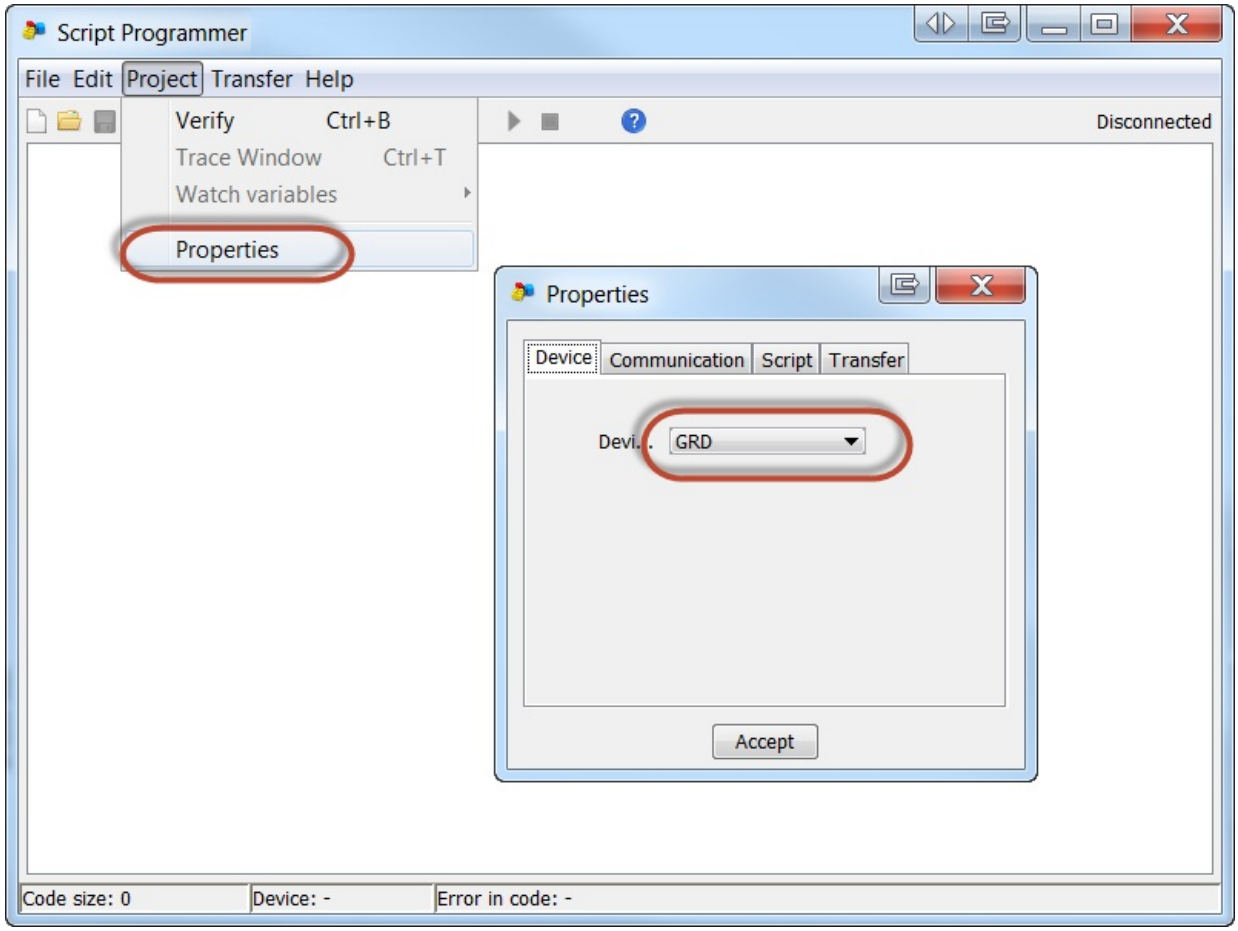
Connecting to the device

There are two ways to connect the GRDconfig to the device. Locally (By USB on the GRD, by LAN/Ethernet on the cLAN) and remotely (through the Middleware)

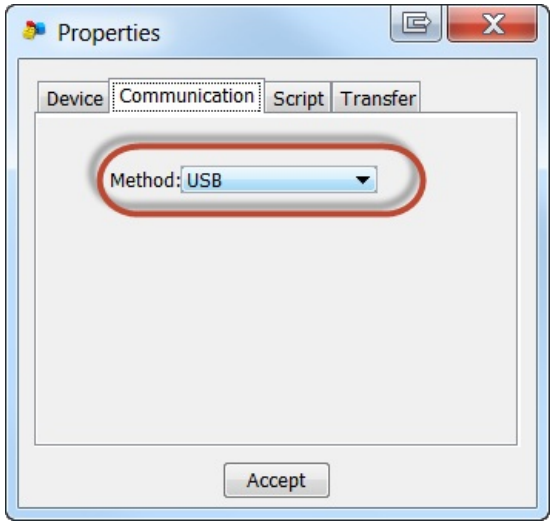
Connecting to the device - GRD by USB

The USB driver must be installed on order to do it using the USB port.

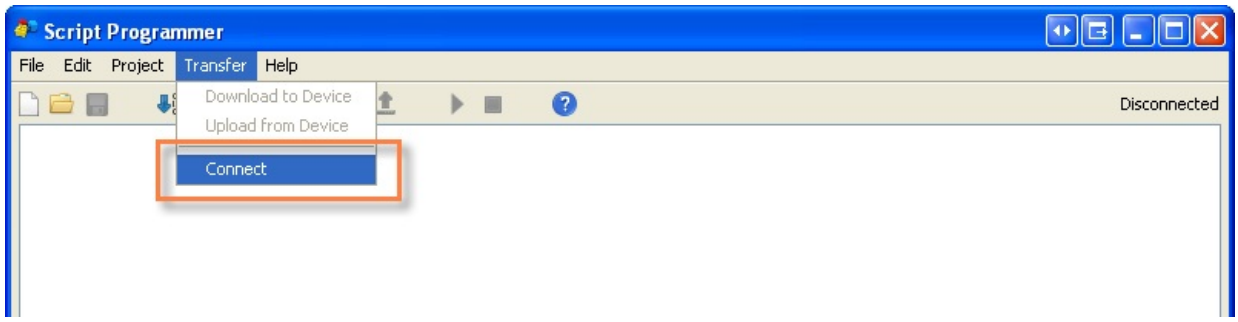
Open the ***"Project"***menu, option ***"Properties"*** and choose "GRD" on the "Device" tab.



The choose the ***"Communication"*** tab, choose USB and press accept



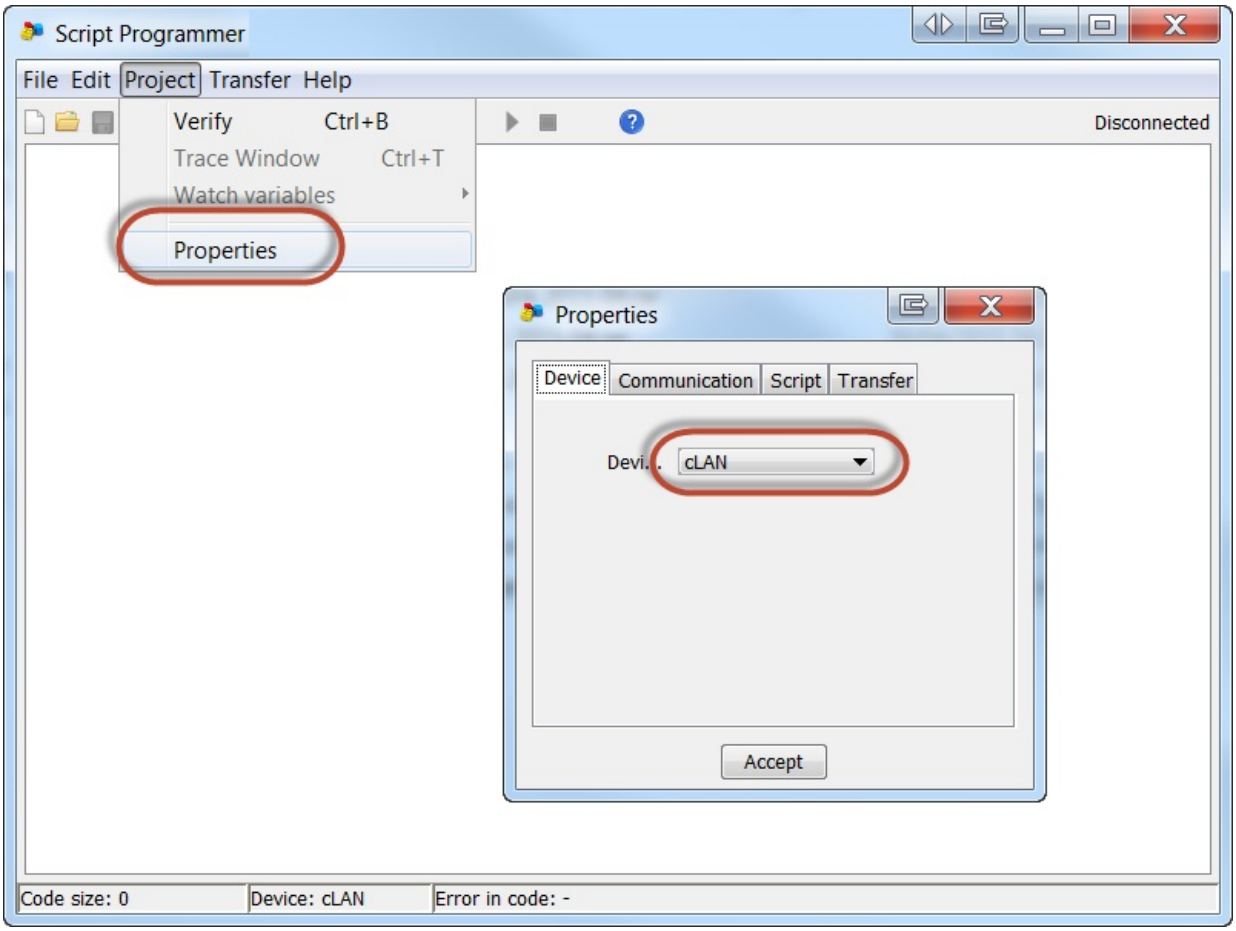
Then go to the ***"Transfer"*** menu and click ***"Connect"***.



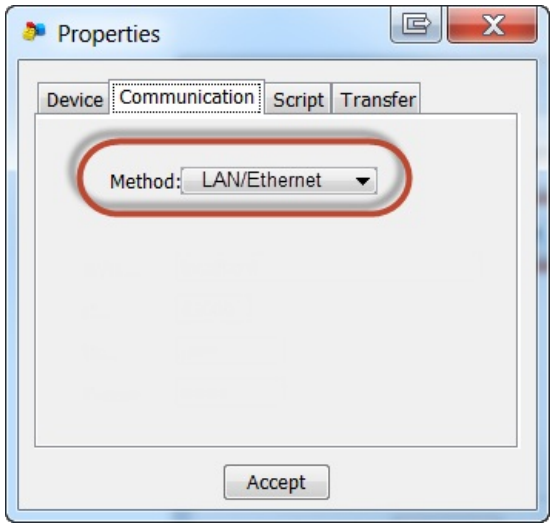
Connecting to the device - cLAN by LAN/Ethernet

The cLAN must be connected to the same network of your computer. Check that it has a valid IP address as described later on these manual.

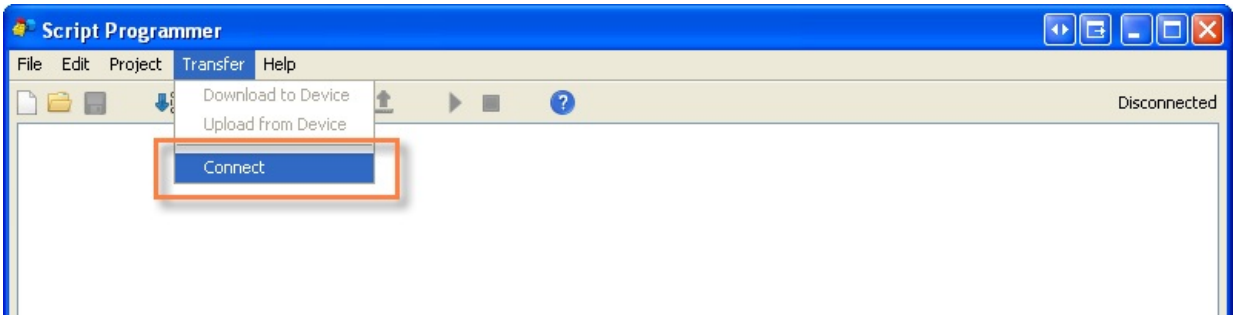
Open then ***"Project"*** menu, opion ***"Properties",*** and choose "cLAN" on the "Device" tab



Choose "LAN/Ethernet" on the ***"Communication"*** tab and press Accept

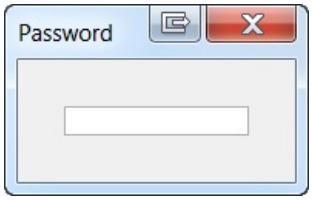


Go to the ***"Transfer"*** menu and click ***"Connect"***



After doing it you will see a list with the cLANs connected to your network. Choose the one you want to configure.

You will be asked to type a password. It's the same password the cLAN will use to establish a connection to the MW.

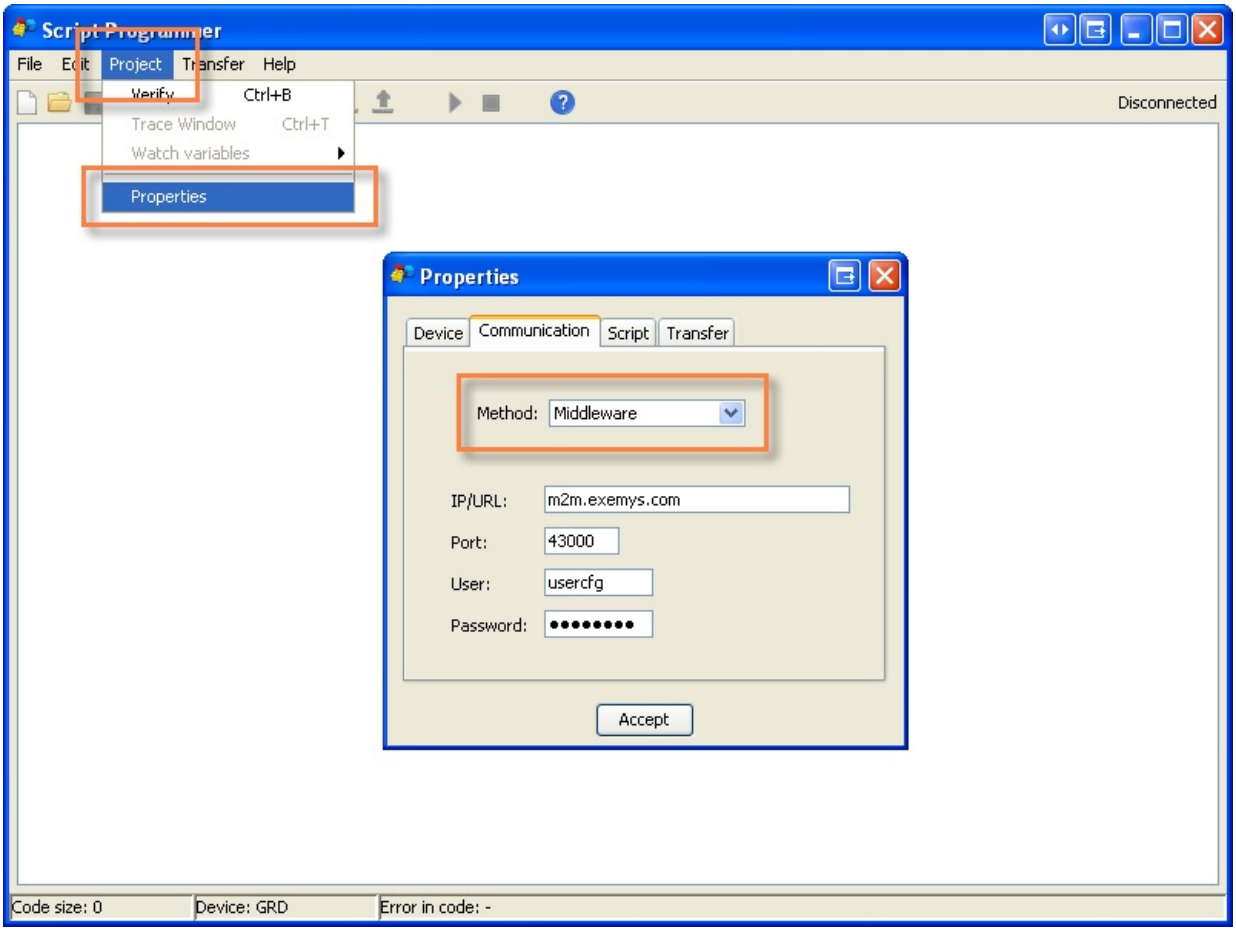


Connecting to the device - cLAN-XF / GRD-XF remotely (XF models only)

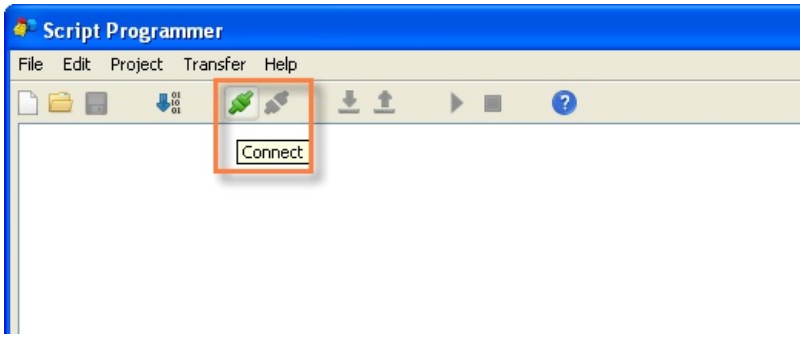
The device must be connected to the MW to configure it remotely.

The Middleware version must be 4.2.0 or higher to support scripts download/upload.

If you are going to use the Middleware you must set up the MW's IP address/URL, port, user and password for remote configuration. To do it, go to the **"Project"** menu**,** then to **"Properties"**, select the **"Communication"** tab and choose "Middleware" on the **"Method"** combo box.



Then click on the connect button at any time.



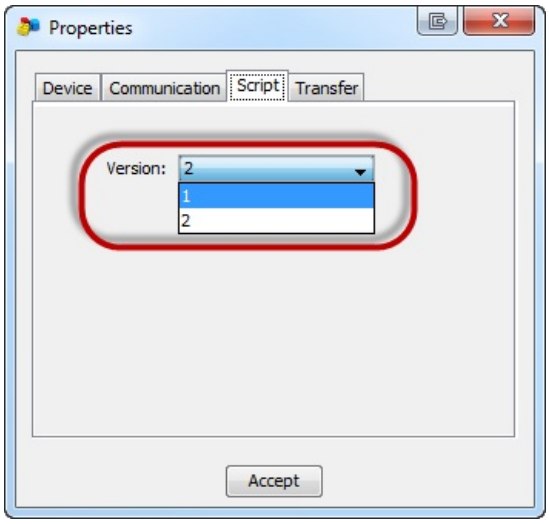
Then you will have to click on the device that you want to configure.

Script Versions 1 and 2

In the menu ***"Project"***, option ***"Properties"***, tab ***"Script"*** you can chose between script versions 1 and 2.

GRD-2G and cLAN V1.x use version 1. GRD-3G y cLAN V2.0+ use version 2.

Version 2 doubles the variables quantity, it allows upper and lower case variables. Version 1 only allows lower case variables.

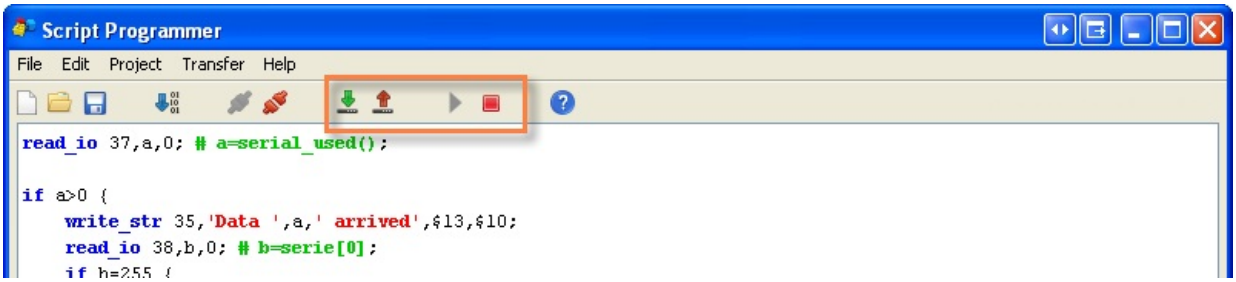


The choosen version will be use in two situations. While verifyng the script or before sending it to the device.

If the script version is not compatible with the device you will see this error message.

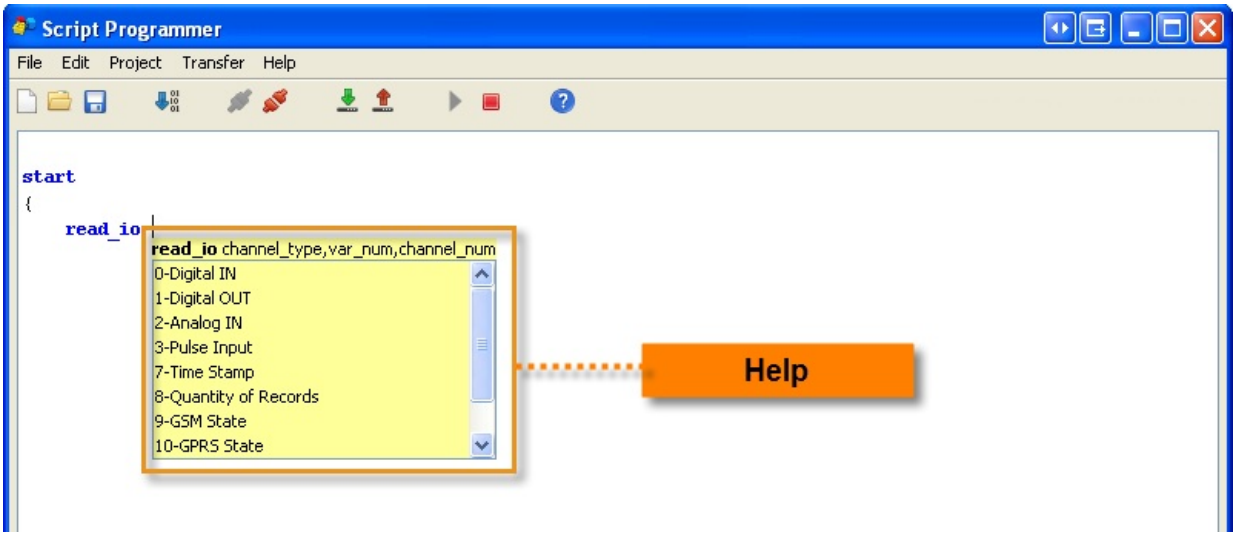
Upload/Download Script

Once the link is established you can send or received scripts. Before downloading a new script it will be checked.

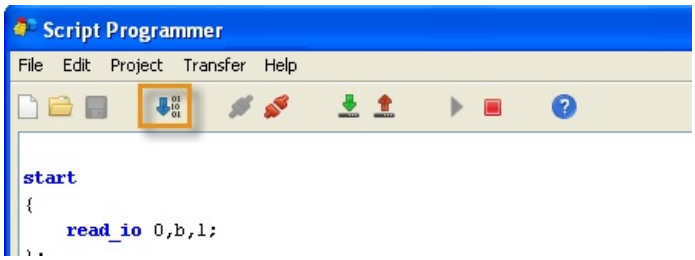


Editing scripts

To write a script you must type the code on the edition text area. Contextual help will be displayed on some functions. .

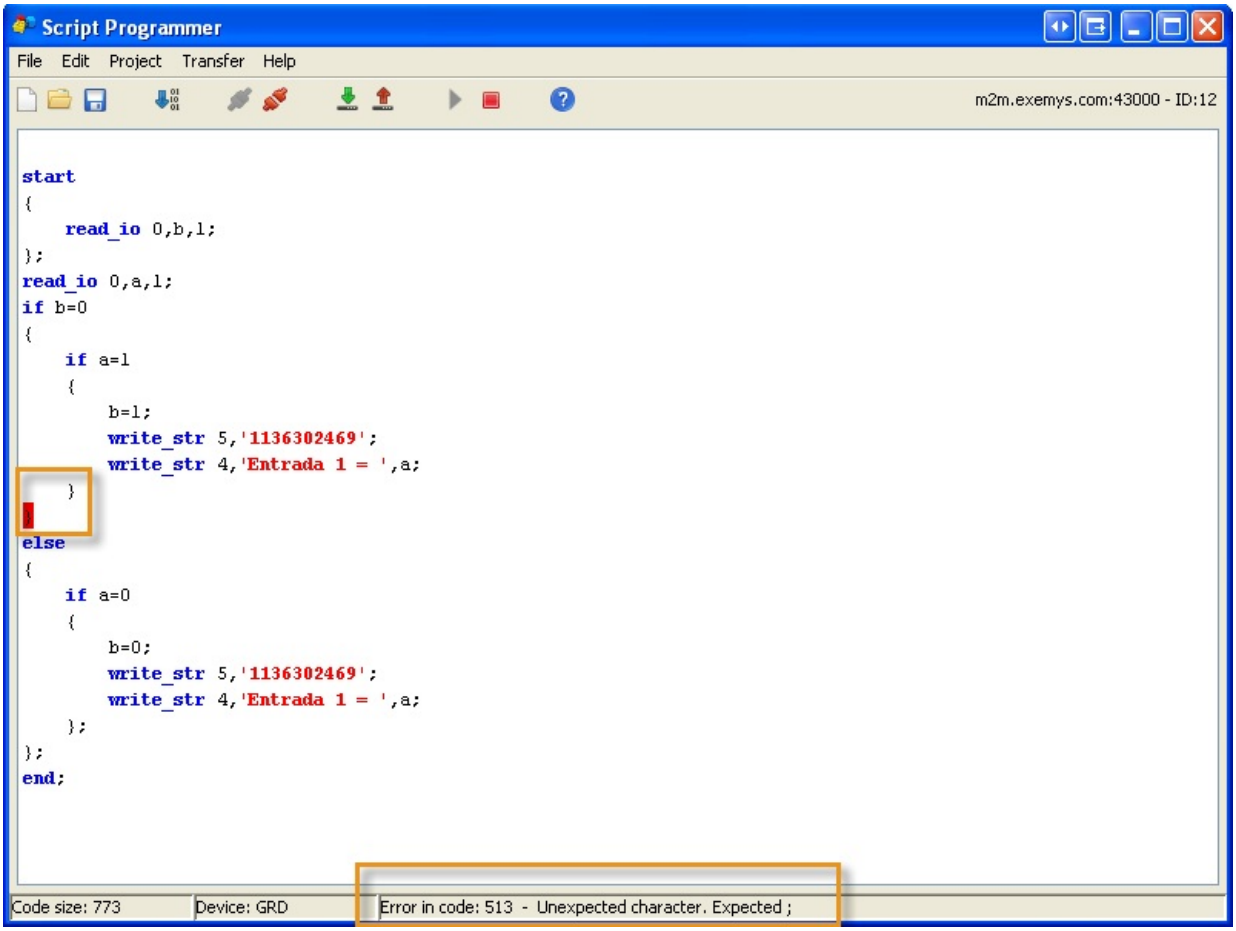


To check if there are errors on the code you can press the ***"Verify"*** anytime.



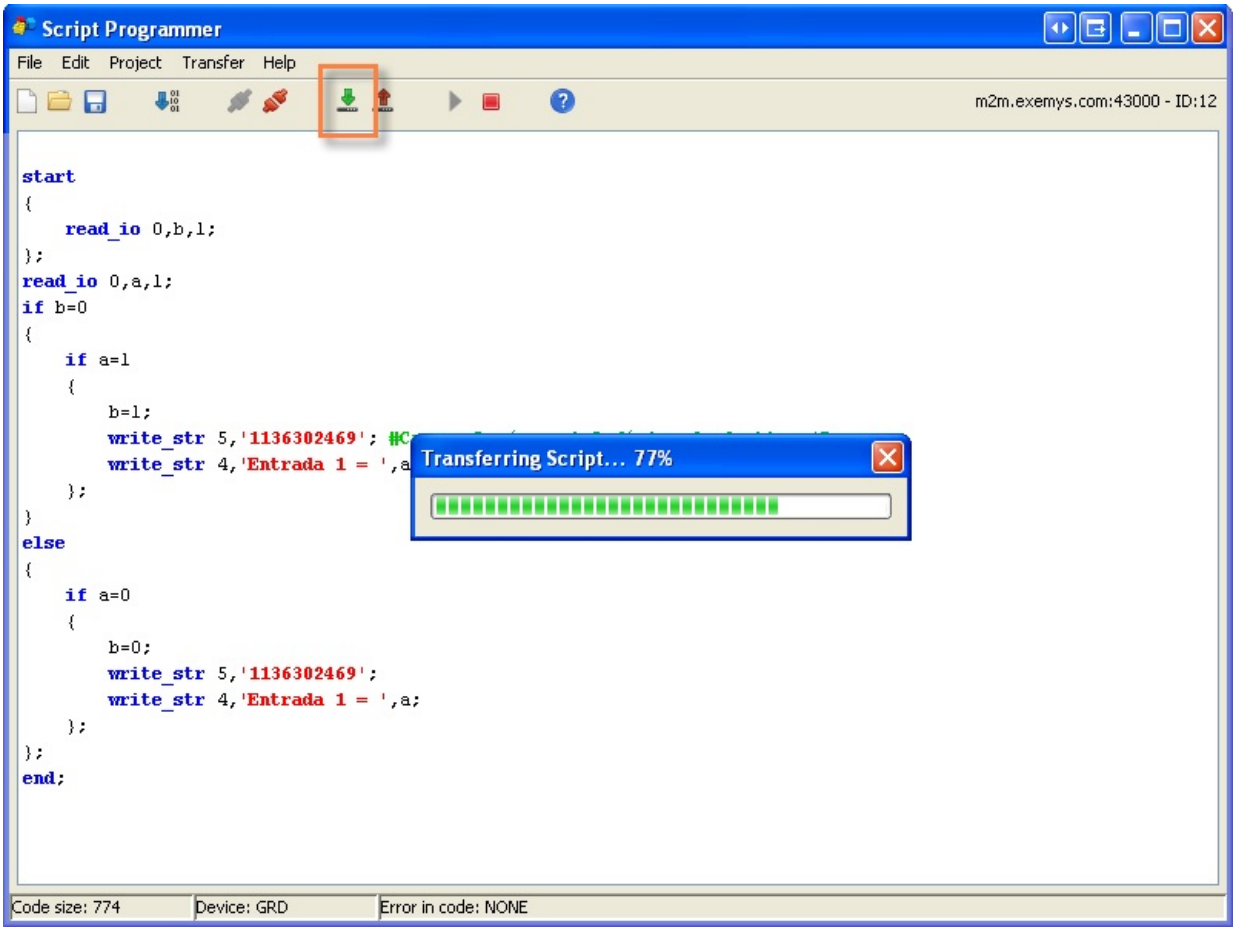
If the software detects an error it will mark it with a red square and will show you the line on the bottom

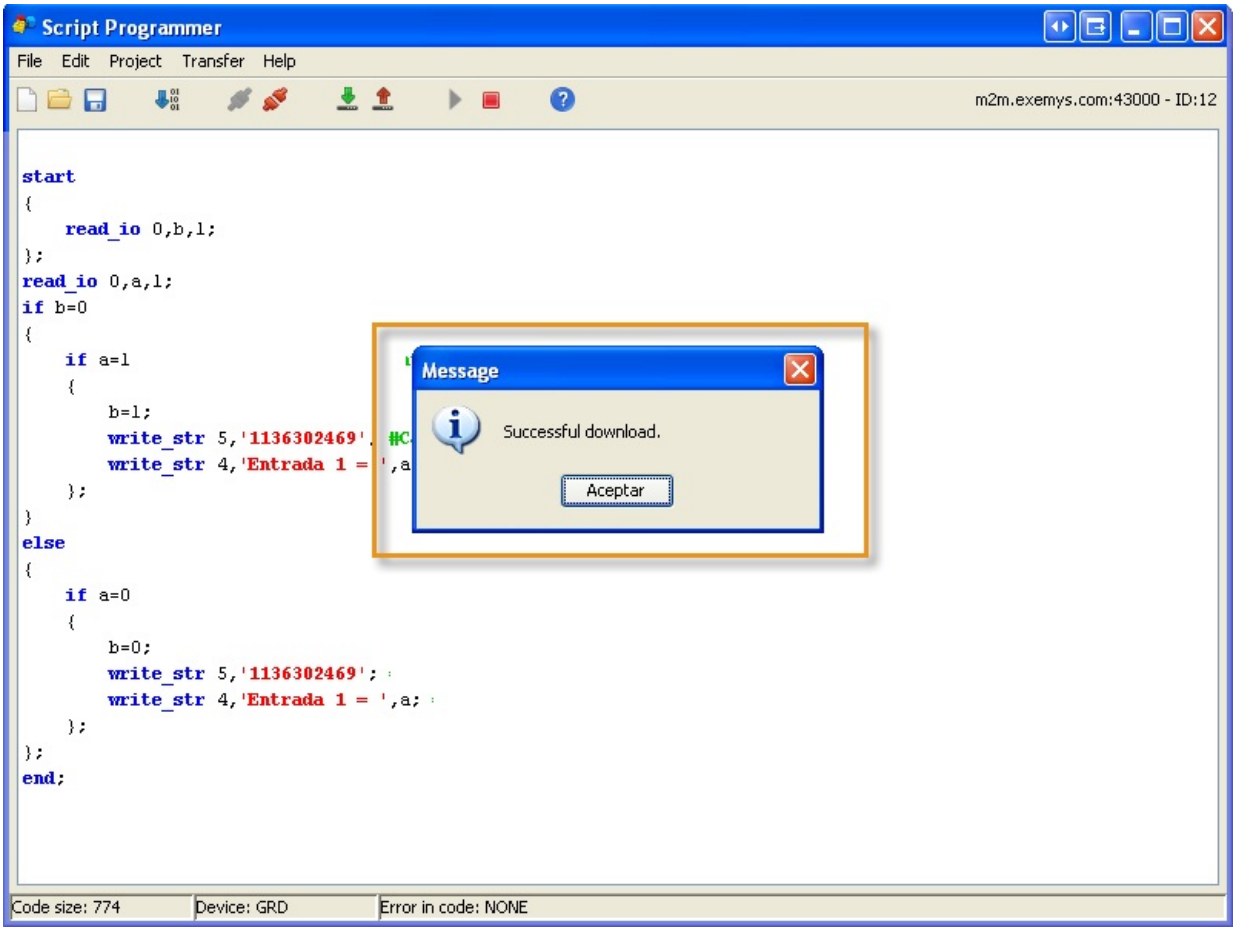
On this next example we can see a missing ***";"***



If no errors are found you will see the next pop-up window.

Once the script is verified you can send it to GRD by clicking on ***"Download to device".***





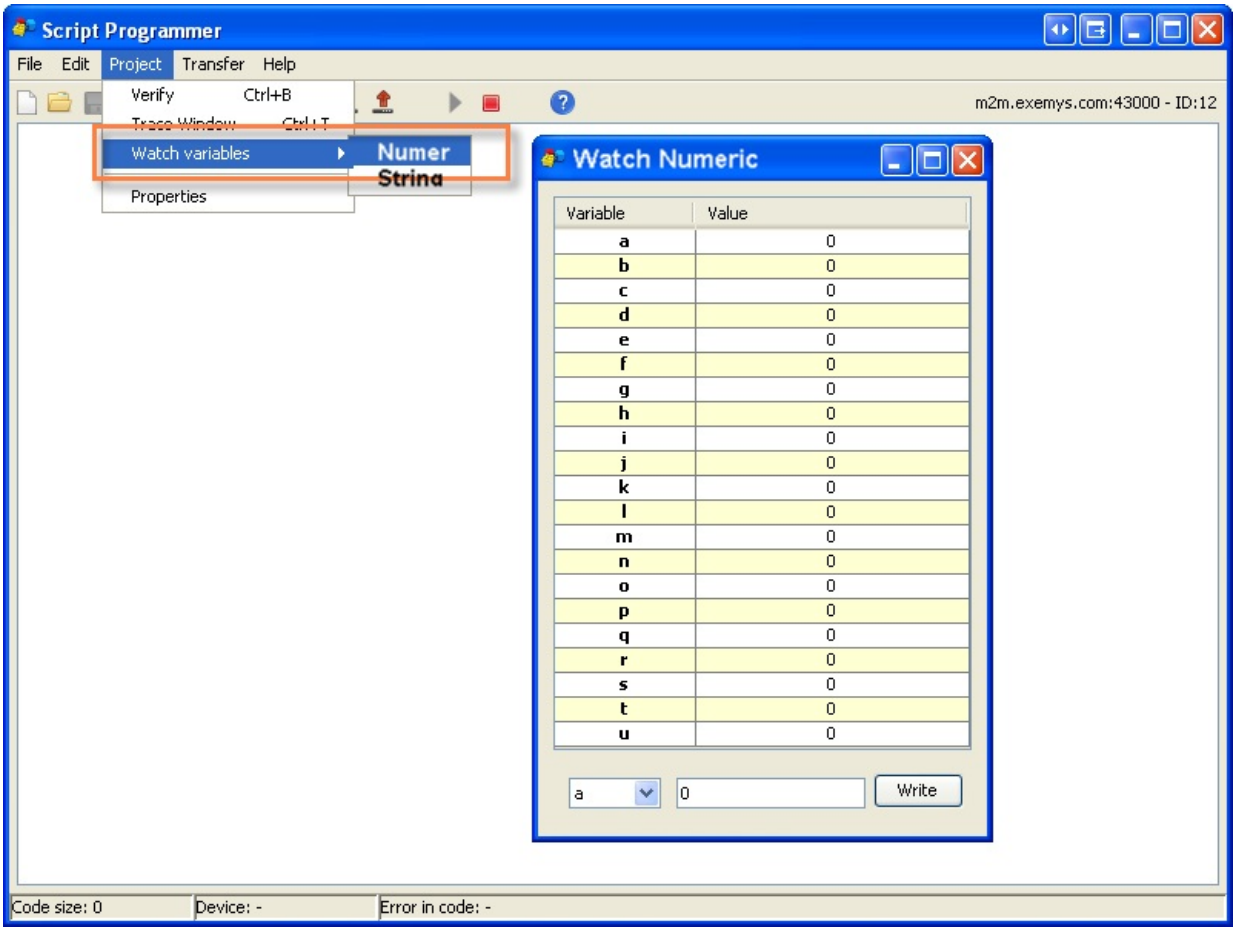
Script debugging

The Script Programmer has two methods that will help you to debug your application. The GRD's firmware must be 5.2.0 or higher to support these options.

Variable watch

This tool will let you check the numeric and string values. You will also be able to edit these values in run time.

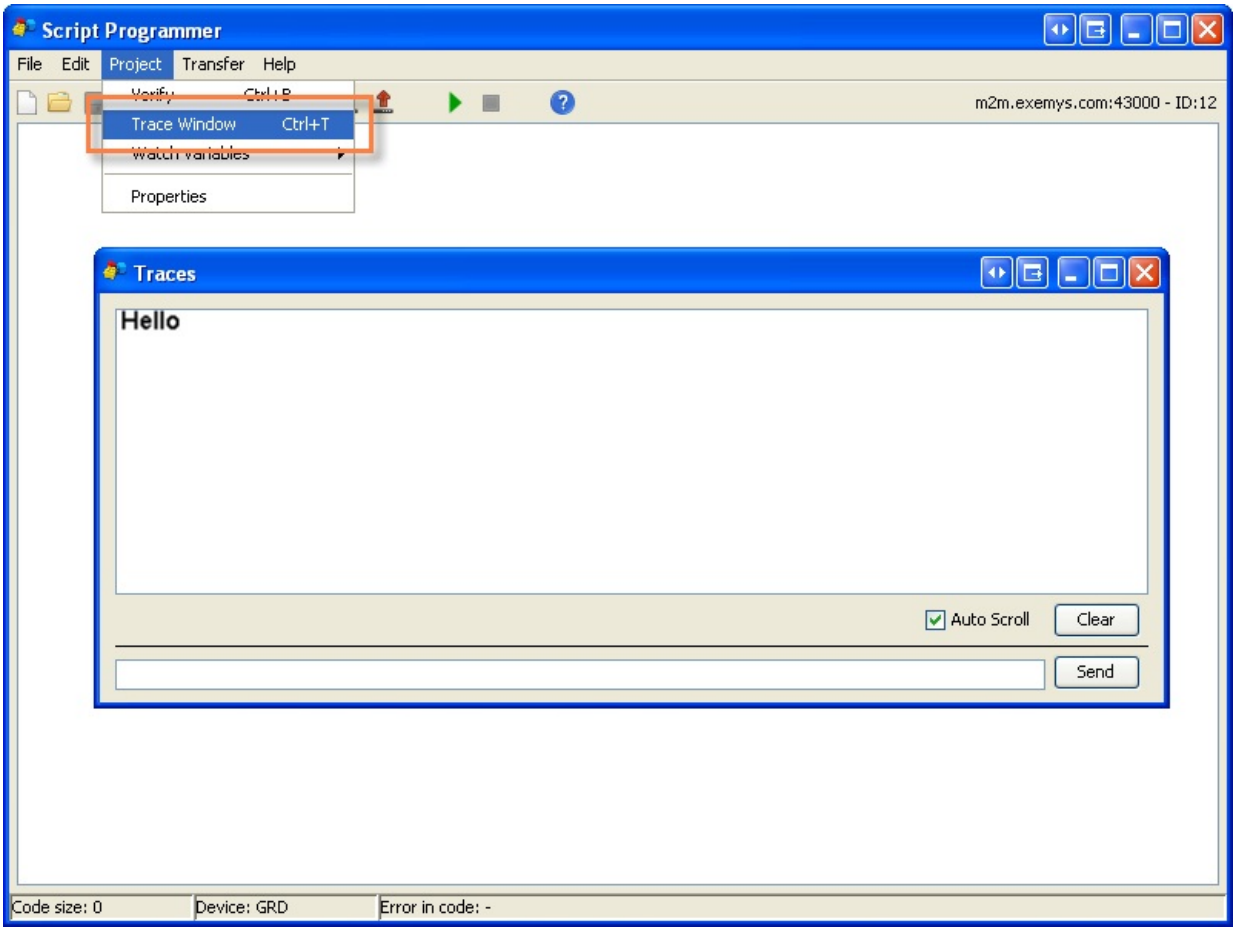
Once the GRD is connected to the **"Project"** menu**, "Watch variables"** option and then select **"Numeric"** or **"String"**



Trace Window

This tool will display messages sent from the script in a pop-up window. You will also be able to send texts to the script to simulate different working conditions.

Once the GRD is connected to the **"Project"** menu**, "Trace Window "**option



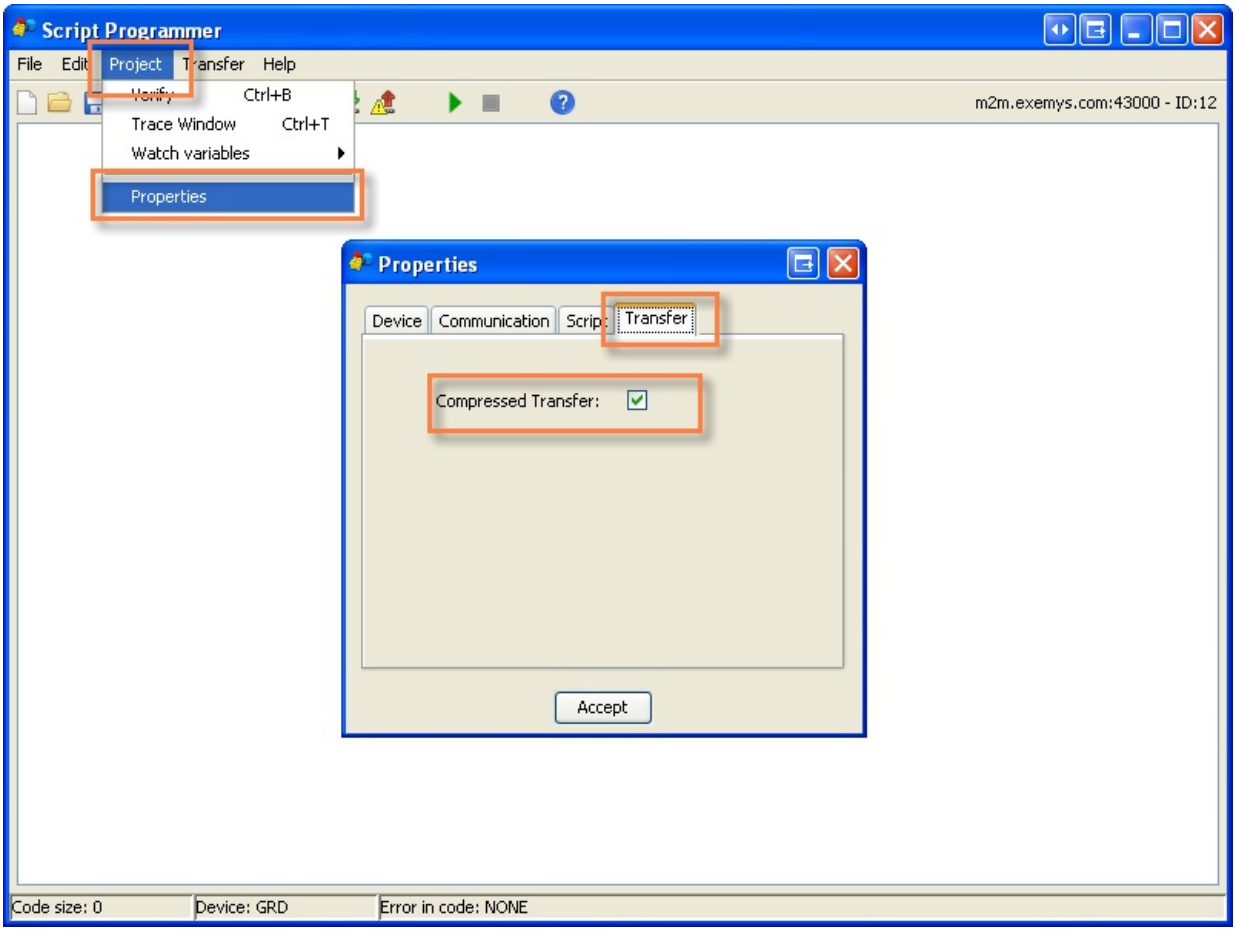
Script Compression

The script maximum size supported by the GRD is 20000 characters.

If this space is not enough for your application you can compress the script before sending it to the GRD. This will remove all your comments on the code and the extra spaces and tabs.

If you want to keep these comments you have to save a copy of the script in your computer.

To enable the compression go to the **"Project"** menu, **"Properties"** option, **"Transfer"** tab and check the **"Compressed Transfer"** check box.


Script Programmer
File
Edit
Project
Transfer
Help
Verify
Ctrl+B
Trace Window
Ctrl+T
Watch variables
Properties
m2m.exemys.com:43000 - ID:12
Properties
Device
Communication
Script
Transfer
Compressed Transfer:
Accept
Code size: 0
Device: GRD
Error in code: NONE

2020-11-25

Introduction

The Exemys script programming language runs in a loop. This means that it will run until the last program line and start from the beginning again.

Loop functions are not avaible. So, the program flow can't be stopped or looped. It runs like a ladder program in a PLC, but its syntaxis is similar to C language.

We suggest not only to read this manual but to read the examples to better understand how to write a script. You can download the examples from here www.exemys.com/GRDscriptsExamples

Script using the SMS feature will only work on the GRD.

Script Versions 1 and 2

There are two version of the script. Version 2 doubles the variables quantity, it allows upper and lower case variables. Version 1 only allows lower case variables.

GRD-2G and cLAN V1.x use version 1

GRD-3G and cLAN V2.0+ use version 2

All GRD-MQ and cLAN-MQ use version 2.

Variables

There are two variable types. Numeric variables and String variables.

It's not necesary to define variables.

In **Version 1** there are 21 numeric variables, from ***"a"*** to ***"u"***. And there are 5 string variables from ***"v"*** to ***"z"***.

In **Version 2** there are 42 numeric variables, from ***"a"*** to ***"u"*** and from ***"A"*** to ***"U"***. And there are 10 string variables from ***"v"*** to ***"z"*** and from ***"V"*** to ***"Z"***.

String variable's maximum length is 100 characters.

Numeric variables are signed integer type, and their value range goes from −2,147,483,648 to 2,147,483,647. If a math operation gives a result with decimals, it will be truncated to the integer part.

Initial value es 0 for numeric variables and empty for strings.

Numeric variables can be mapped into GRD's I/O channels to send reports or create historical records based on its values.

Assigning a value to a variable:

- Numeric variables:

```
a = 652;
```

- String variables :

```
v = 'Hello world'
```

String concatenation:

To concatenate two or more strings use the comma operator.

Example:

```
a = 20;
u = 'Temperature ';
v = ' °F';

w = u, a, v;
```

The result will be `'Temperature 20 °F'`

Another way to do the same is:

```
w = 'Temperature ', a,' °F';
```

String concatenation can only be done on string variables value assignment and *write_str* function

Assigning ASCII values to a string variable:

To assign ASCII values use the $ operator. After the operator type the ASCII value on decimal notation. ACII value zero is not allowed.

Example:

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

ASCII values assignment can only be done on string variables value assignment and *write_str* function

Arithmetic operators

Operator	Description
=	Assignment
^	Exponential
|	Bitwise Or
&	Bitwise And
+	Addition
-	Subtraccion
*	Multiplication
/	Division
%	Modulo

Example:

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

Result: *b* variable value is 135.

Program structure

Al instruction must end with the ***";"*** symbol.

The program runs in a loop. This means that it will run until the last program line and start from the beginning again.

The script last instruction must be ***"end;"***

```
a = a + 1;
end;
```

On this example "a" variable will be incremented constantly. Its initial value is 0.

On-line comments:

If you wish to add a comment line you must use the ***"#"***. On-line comments must also en with the ***";"*** symbol.

Flow control functions

"start" function

It marks a block that will be executed only once. It must be written at the beginning of the script.

Syntax:

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

Example:

```
start
{
    a = 10;  #a initial value is 10;
};
a = a + 1;   #a is incremented by 1 constantly;
end;
```

"if-else" function

The script will decide the script execution flow based on condition. If the condition is true the code in the block next to the "***if"*** instruction will be executed. You can add also a code block that will be executed if the condition is no true.

The condition operators are the following ones:

Operator	Description
=	Equals to
!	Not equal to
>	Greater than
<	Less than

Syntax:

Single ***"if"***:

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

A ***";"*** symbol is required to close the block.

"if-else":

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

else
{
   ...;
   ...;
};
```

The***";"*** is only requiered on the ***"else"*** block.

"end" function

This function is used to mark the end of the program. When the interpreter finds this line it will jump to the first line of the script.

Syntax:

```
end;
```

Interface functions

"read_io" funcion

With ***read_io*** you can get values from different sources like I/O channels, the real time clock, etc

The ***"source"*** is indicated with a number. Some sources will require an index number to point an address inside that source.

The result of this function will be loaded in the indicated numeric variable.

Syntax:

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

Available sources may change depending on the device where you are running the script and the script version. New sources can be added in the future.
Browse "Sources-Destinations" section for the currently available ones.

"write_io" function

With ***write_io*** you can set values in different destinations, like digital output channels, pulse channels, etc.

The ***"destination"*** is indicated with a number. Some destinations will require an index number to point an address inside that destination.

The value to be written can be a number or a numeric variable.

Syntax:

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

Available destinations may change depending on the device where you are running the script and the script version. New destinations can be added in the future.

Browse "Sources-Destinations" section for the currently available ones.

"read_str" function

With ***read_str*** you can get incoming strings from different sources like the serial port or a SMS.

The ***"source"*** is indicated with a number.

The result of this function will be loaded in the indicated string and numeric variables. The numeric variable will contain the string length. If the value is 0 it means that there isn't a new incoming string from that source.

Syntax:

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

Available sources may change depending on the device where you are running the script and the script version. New sources can be added in the future.

Browse "Sources-Destinations" section for the currently available ones.

"write_str" function

With ***write_str*** you can send strings to different destinations, like an SMS or the serial port. The ***"destination"*** is indicated with a number.

Syntax:

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

Available destinations may change depending on the device where you are running the script and the script version. New destinations can be added in the future.

Browse "Sources-Destinations" section for the currently available ones.

The string can be a variable string or a text typed between single quotes. **This function support string concatenation and including ASCII values.**

String functions

"is_equal" function

Compares one string variables with a text (variable string or a text typed between single quotes). The

numeric variable will contain the result, 1 if they are equal or 0 if they are different.

Syntax:

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

Example:

```
v='PUMP RUN';
is_equal c,v,'PUMP RUN';
if c=1 {
        #texts are equal;
};
```

"finish_with" function

Compares the end of one string variables with a text (variable string or a text typed between single quotes). The numeric variable will contain the result, 1 if they match or 0 if they don't.

Syntax:

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

Example:

```
v='PUMP RUN';
finish_with c,v,'RUN';
if c=1 {
        #The string ends with 'RUN';
};
```

"begin_with" function

Compares the beginning of one string variables with a text (variable string or a text typed between single quotes). The numeric variable will contain the result, 1 if they match or 0 if they don't.

Syntax:

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

Example:

```
v='PUMP RUN';
end_with c,v,'RUN';
if c=1 {
        #The string ends with 'RUN';
};
```

"contains" function

Determines if one string (fixed text or string variable) is contained by a string variable. The numeric variable will contain the position where the string if found or 0 if its not contained.

Syntax:

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

Example:

```
v='PUMP RUN';
contains c,v,'MP';
```

```
if c>0 {
        #The variable v contains the text 'MP' ;
};
```

"upper" function

Converts all character is one string variable to uppercase.

Syntax:

```
upper string_variable;
```

Example:

```
v='Turn ON';
upper v;
#v equals 'TURN ON';
```

"lower" function

Converts all character is one string variable to lowercase.

Syntax:

```
lower string_variable;
```

Example:

```
v='Turn ON';
upper v;
#v equals 'turn on';
```

"strlen" function

Gets the string length and stores it on a numeric variable.

Syntax:

```
strlen numeric_variable,string_variable;
```

Example:

```
v='PUMP RUN';
strlen c,v;
#c equals 8 ;
```

"substr" function

Returns part of a string within the same string variable

Syntax:

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

```
v='PUMP RUN';
substr 2,3,v;
#v equals 'UMP';
```

Conversion functions

"point" function

Converts a numeric variable to string and places a decimal point on a fixed position.

Syntax:

```
point string_variable,numeric_variable,decimals;
```

Example:

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

"aton" function

Converts number inside a string variable to a numeric variable. It starts at the beginning of the string and ends where it finds a non-numeric character or reaches the end of the string.

Syntax:

```
aton numeric_variable,string_variable;
```

Example:

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

"day","month","year","hs","min","sec" and *"nday"* functions

These functions will convert a ***time_stamp*** to day, month, year, hour, minute, seconds or day of the week.

Current data/time can be read using ***read_io*** with source #7.

Syntax:

```
day day,timestamp;

month mont,timestamp;

year year,timestamp;

hs hour,timestamp;

min minutes,timestamp;

sec seconds,timestamp;

nday dayoftheweek,timestamp;
```

"nday" function will return the day of the week number starting with Sunday=0s.

Example:

```
read_io 7,e,0; #Reads current time and date into e;
day f,e;
month g,e;
year h,e;
hs i,e;
```

```
min j,e;
sec k,e;
#The current time and date is f/g/h i:j:k;
```

Mathematical and logic functions

"neg" function

It will invert the value of a numeric variable bitwise.

Syntax:

```
neg result,initialvalue;
```

Example:

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

# b equals 4294934972 (FFFF81BC);
```

"sqtr" function

Calculates the square root of a numeric variable. As numeric values are integers the fractional part will be truncated. Multiply the number before calculation if you need higher precision.

Syntax:

```
sqrt result,initialvalue;
```

Example:

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

"scale" function

Scales a number using the two point form of the linear equation.

Syntax:

```
scale result,initialvalue,x0,x1,y0,y1;
```

Example: Scale a 4-20mA signal on input AN1 to a number between 0 and 500

```
read_io 2,a,1; #a = AN1
scale c,a,400,2000,0,500;
#c equals scaled number
```

Timming functions

This functions will allow you to control the program flow using timers.

"timer" and *"check_timer"* function

Use ***"timer"*** to store on a numeric variable the time you want to wait (in milliseconds)

Use ***"check_timer"*** to check if the time has expired or not.

Syntax:

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

Once the time has expired the code inside the *check_timer* block will be executed. This code will be executed on every program loop until the timer is loaded again. Typically you will be reloaded the timer inside the *check_timer* block.

Note: The timing functions are no recommend on applications where precision timing is required because timers can have some dispersion.

2020-11-25

Introduction

Functions read_io, write_io, read_str and write_str can be used to gain access to additional features. On this section the different sources and destinations per function are listed. Then they are grouped by feature.

Sources/destinations list

"read_io" sources

Source	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
0	Digital input channel (Ix)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
1	Digital output channel (Ox)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
2	Analog input channel (ANx)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
3	Pulse input channel (PIx)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
23	Pulse input channel mapped to Modbus query reading Float 32 number. Returns the integer part of the value times 1000 (PIx)	1 to 100	5.1.2	Yes	Yes	Yes	Yes	Yes
36	Pulse input channel mapped to Modbus query reading Float 32 number with bytes swapped. Returns the integer part of the value times 1000 (PIx)	1 to 100	5.2.0	Yes	Yes	Yes	Yes	Yes
305	Channels memory	0	-	-	1.9	2.8	Yes	Yes
7	Current time (seconds since 1/1/2000)	0	Yes	Yes	Yes	Yes	Yes	Yes
8	Records in historical records memory	0	Yes	Yes	Yes	Yes	Yes	Yes
9	GSM link state (see table below)	0	Yes	-	Yes	-	Yes	-
10	GPRS link state (see table below)	0	Yes	-	Yes	-	Yes	-
11	Middleware link state (see table below)	0	Yes	Yes	Yes	Yes	-	-i
21	Numbers non-volatile memory (read)	1 a 20	5.1.1	Yes	Yes	Yes	Yes	Yes
22	MW link enabled configuration	0	5.2.2	Yes	Yes	Yes	-	-
37	Number of bytes stored in serial port buffer (to delete these bytes use write_io 37)	0	5.2.0	Yes	Yes	Yes	Yes	Yes
38	Binary value of one byte of the serial port buffer	1 to 100	5.2.0	Yes	Yes	Yes	Yes	Yes
39	SMS pending to be sent	0	5.2.4	Yes	Yes	Yes	Yes	-
48	Disable sending historical records to the MW (1 disabled, 0 enabled)	0	5.2.0	Yes	Yes	Yes	Yes	Yes
47	FTP client state	0	5.2.2	Yes	-	Yes	-	Yes
55	Satellite modem state	0	5.2.2	Yes	Yes	Yes	-	-
61	Get historical record memory channel type, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
62	Get historical record memory timestamp, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
63	Get historical record memory historical type, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
64	Get historical record memory channel number, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
65	Get historical record memory value, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
75	*UDP socket reception state (1 = ready)*	*0*	-	-	-	*2.2*	-	*Yes*
76	*UDP socket transmition state (1 = ready)*	*0*	-	-	-	*2.2*	-	*Yes*
77	*UDP socket binary read. Bytes received*	*0*	-	-	-	*2.2*	-	*Yes*
78	*UDP socket binary read. Read position*	*0 a 100*	-	-	-	*2.2*	-	*Yes*
81	*HTTP client. Answer length*	*0*	-	-	-	*2.2*	-	*Yes*
82	*HTTP client. State*	*0*	-	-	-	*2.2*	-	*Yes*
95	*SMTP client. State*	*0*	-	-	-	*2.2*	-	*Yes*
195	*POP client. State*	*0*	-	-	-	*2.2*	-	*Yes*
270	Get direct Modbus query value	1 to 100	-	-	1.8	-	Yes	-
271	Get direct Modbus query state	1 to 100	-	-	1.8	-	Yes	-
280	Roamming state (0=no, 1=yes)	0	-	-	1.8	-	Yes	-
1000	MQTT-Messages pending to be read	0	-	-	-	-	Yes	Yes
1001	MQTT-Broker link state (1=connected)	0	-	-	-	-	Yes	Yes

"write_io" destinations

Destination	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
1	Digital output channel (Ox)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
2	Analog input channel (ANx, Modbus only)	1 to 100	5.1.3	Yes	Yes	Yes	Yes	Yes
3	Pulse input channel (PIx)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
305	Channels memory	0	-	-	1.9	2.8	Yes	Yes
7	Set real time clock (seconds since 1/1/2000)	0	-	-	1.8	2.6	Yes	Yes
56	Create a digital input channel "by change" historical record	1 to 100	5.2.2	Yes	1.6	Yes	Yes	Yes
57	Create a digital output channel "by change" historical record	1 to 100	5.2.2	Yes	1.6	Yes	Yes	Yes
12	Create an analog input channel "by time" historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
14	Create an analog input channel "by alarm" maximum value historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
15	Create an analog input channel "by alarm" minimum value historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
16	Create an analog input channel "by alarm" normal value historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
13	Create a pulse input channel "by time" historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
17	Force a digital input (Ix) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
18	Force a digital output (Ox) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
19	Force an analog input (ANx) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes

20	Force an pulse input (PIx) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
21	Numbers non-volatile memory (write)	1 a 20	Yes	Yes	Yes	Yes	Yes	Yes
22	MW link enabled configuration (0 o 1)	0	5.2.2	Yes	Yes	Yes	-	-
48	Disable sending historical records to the MW (1 disabled, 0 enabled)	0	5.2.2	Yes	Yes	Yes	Yes	Yes
37	Delete N bytes from the serial port buffer (use together with read_io 37 and read_io 38)	0	5.2.0	Yes	Yes	Yes	Yes	Yes
38	Send a byte to the serial port (Binary value)	0	5.2.5	-	Yes	Yes	Yes	Yes
60	Read specific register from historical records memory (use together with read_io 61 to 65)	-	5.2.2	Yes	Yes	Yes	Yes	Yes
66	Delete the first N registers from the historical records memory	-	5.2.2	Yes	Yes	Yes	Yes	Yes
54	Initiate sending historical records using satellite modem	0	5.2.2	Yes	Yes	Yes	-	-
32	*Satellite. Begin reception check.*	*0*	-	-	*1.3*	*2.2*	-	-
59	Change the multiplier for read_io 23 and 36 (1000 default value)	0	5.2.5	-	Yes	Yes	Yes	Yes
44	Initiate FTP client connection	0	5.2.2	Yes	-	Yes	-	Yes
46	Finish FTP client connection	0	5.2.2	Yes	-	Yes	-	Yes
77	*UDP Client. Send N bytes previously stored on the UDP buffer*	*0*	-	-	-	*2.2*	-	*Yes*
78	*UDP Client. Load UDP buffer*	*0 a 100*	-	-	-	*2.2*	-	*Yes*
82	*HTTP Client. Begin connection check*	*0*	-	-	-	*2.2*	-	*Yes*
195	*POP Client. Begin connection check*	*0*	-	-	-	*2.2*	-	*Yes*

"read_str" sources

Source	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
4	SMS text	Yes	-	Yes	-	Yes	-
5	SMS sender's phone number	Yes	-	Yes	-	Yes	-
6	Serial port	Yes	Yes	Yes	Yes	Yes	Yes
32	*Satellite. String received from MW's transparent port.*	-	-	*1.3*	*2.2*	-	-
35	Script Programmer's trace window	5.2.0	Yes	Yes	Yes	Yes	Yes
51	Reads process buffer adding NMEA start, end and checksum bytes (load process buffer with write_str 50 before using it)	5.2.0	Yes	Yes	Yes	Yes	Yes
77	*UDP Socket. Get last string received.*	-	-	-	*2.2*	-	*Yes*
81	*HTTP Client.. Get last string received.*	-	-	-	*2.2*	-	*Yes*
193	*POP client. Get sender's email address.*	-	-	-	*2.2*	-	*Yes*
194	*POP client. Get subject*	-	-	-	*2.2*	-	*Yes*
195	*POP client. Get body. Load next message in queue.*	-	-	-	*2.2*	-	*Yes*
101 a 108	Phone book names 1 to 8	5.1.3	-	Yes	-	Yes	-
111 a 118	Phone book telephone number 1 to 8	5.1.3	-	Yes	-	Yes	-
121 a 125	Strings non-volatile memory 1 to 5 (read)	5.2.2	Yes	Yes	Yes	Yes	Yes
1000	MQTT. Get first message in queue.	-	-	-	-	Yes	Yes

"write_str" destinations

Destination	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
4	SMS text (send order)	Yes	-	Yes	-	Yes	-
5	SMS recipient's phone number	Yes	-	Yes	-	Yes	-
6	Serial port	Yes	Yes	Yes	Yes	Yes	Yes
35	Script Programmer's trace window	5.2.0	Yes	Yes	Yes	Yes	Yes
50	Process buffer (use together with read_str 51)	5.2.0	Yes	Yes	Yes	Yes	Yes
121 a 125	Strings non-volatile memory 1 to 5 (write)	5.2.2	Yes	Yes	Yes	Yes	Yes
40	Load FTP Client URL	5.2.2	Yes	-	Yes	-	Yes
41	Load FTP Client use	5.2.2	Yes	-	Yes	-	Yes
42	Load FTP Client password	5.2.2	Yes	-	Yes	-	Yes
43	Load FTP Client file name	5.2.2	Yes	-	Yes	-	Yes
45	Load FTP file text line and send	5.2.2	Yes	-	Yes	-	Yes
75	*UDP socket. Initialize socket and set listen port*	-	-	-	*2.2*	-	*Yes*
76	*UDP socket. Set remote IP address and port*	-	-	-	*2.2*	-	*Yes*
77	*UDP socket. Send string*	-	-	-	*2.2*	-	*Yes*
80	*HTTP client. Ser URL and port*	-	-	-	*2.2*	-	*Yes*
84	*HTTP client. Set path/file name*	-	-	-	*2.8*	-	*Yes*
81	*HTTP client. Set GET query string (xx=123&yy=456…)*	-	-	-	*2.2*	-	*Yes*
83	*HTTP client. Set value to 'data' field on the GET query string (alternative to write_str 81)*	-	-	-	*2.2*	-	*Yes*
89	*SMTP client. Set URL and port*	-	-	-	*2.2*	-	*Yes*
90	*SMTP client. Set sender's email address*	-	-	-	*2.2*	-	*Yes*
91	*SMTP client. Set user name*	-	-	-	*2.2*	-	*Yes*
92	*SMTP client. Set password*	-	-	-	*2.2*	-	*Yes*
93	*SMTP client. Set recipient's email address*	-	-	-	*2.2*	-	*Yes*
94	*SMTP client. Set subject*	-	-	-	*2.2*	-	*Yes*
95	*SMTP client. Set body and begin sending email*	-	-	-	*2.2*	-	*Yes*
189	*POP client. Set URL and port*	-	-	-	*2.2*	-	*Yes*
191	*POP client. Set user name*	-	-	-	*2.2*	-	*Yes*
192	*POP client. Set password*	-	-	-	*2.2*	-	*Yes*
1001	MQTT-Load topic to publish	-	-	-	-	Yes	Yes

1002	MQTT-Load payload to publish and begin publishing	-	-	-	-	Yes	Yes

Read/Write Input/Output channels

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
0	Digital input channel (Ix)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
1	Digital output channel (Ox)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
2	Analog input channel (ANx)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
3	Pulse input channel (PIx)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
23	Pulse input channel mapped to Modbus query reading Float 32 number. Returns the integer part of the value times 1000 (PIx)	1 to 100	5.1.2	Yes	Yes	Yes	Yes	Yes
36	Pulse input channel mapped to Modbus query reading Float 32 number with bytes swapped. Returns the integer part of the value times 1000 (PIx)	1 to 100	5.2.0	Yes	Yes	Yes	Yes	Yes

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
1	Digital output channel (Ox)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
2	Analog input channel (ANx, Modbus only)	1 to 100	5.1.3	Yes	Yes	Yes	Yes	Yes
3	Pulse input channel (PIx)	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
59	Change the multiplier for read_io 23 and 36 (1000 default value)	0	5.2.5	-	Yes	Yes	Yes	Yes

Sources 0 to 3 will return the value of the different Input/Output channels. Use the **index** to point to a particular channel address.

Example: Read analog input channel #4 (AN4) value and save it in variable c

```
read_io 3,c,4;
```

Destination 0 allows you to change the value of the digital output channels. Use the **index** to point to a particular channel address.

Example: Turn digital output channel 3 off (O3)

```
write_io 1,3,0;
```

Destination 2 allows you to change the value of **analog input channels** linked to a **Modbus** query. Calling write_io will force a Modbus write command.

Destination 3 will support the same values the source linked to that channels supports (physical counters or Modbus querys with 2 registers length)

Sources 23 and 36 will convert Modbus 32bit floating point queries linked to pulse channels to integer values.

Channels memory

read_io / write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
305	Channels memory	0	-	-	1.9	2.8	Si	Si

This volatile memory area with 100 positions can be used a Source for all the I/O channels.

It can accesed using `read_io/write_io 305`

This allows the user to free script variables used to map data on the I/O channels.

Read direct Modbus query value

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
270	Get direct Modbus Query - value	1 to 100	-	-	1.8	-	Yes	-
271	Get direct Modbus Query - state	1 to 100	-	-	1.8	-	Yes	-

Sources 270 y 271 can be used to get the Modbus Query result (value and state) without the need of mapping it in a channel.

Source 271 will return 0 if the master is not able to get the value from the slave, or 1 if it can.

Example: Get the value from Modbus query 4 and save it in variable c

```
read_io 270,c,4;
```

Real time clock reading

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
7	Current time (seconds since 1/1/2000)	0	Yes	Yes	Yes	Yes	Yes	Yes

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
7	Set real time clock (seconds since 1/1/2000)	0	-	-	1.8	2.6	Yes	Yes

Source 7 will read the current date and time of the real time clock. This number can be converted to a friendly format using the convertion functions.

Example: Get the current month and store in the variable g

```
read_io 7,e,0;
month g,e;
```

Non-volatile memory access

For numbers

read_io / write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
21	Numbers non-volatile memory	1 a 20	5.1.1	Yes	Yes	Yes	Yes	Yes

The source/destination 21 will allow you to read and write numbers from and into the non-volatile memory

In devices with firmware version 5.1.1 do not invoke this function permanently, only when you need to change the value (the non-volatile memory can be damage)

Example: Read the number stored in position 15 of the non-volatile memory and store it in the variable g

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

For strings

read_str / write_str	Descripción	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
121 a 125	Strings non-volatile memory 1 to 5	5.2.2	Yes	Yes	Yes	Yes	Yes

Sources/Destinations 121 to 125 will allow read and write up to 5 strings from/into the non-volatile memory

Example: Write the string 'hello' into the 3rd position of the strings non-volatile memory

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

Read GSM/GPRS/MW state and MW link configuration

State read

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
9	GSM link state (see table below)	0	Yes	-	Yes	-	Yes	-
10	GPRS link state (see table below)	0	Yes	-	Yes	-	Yes	-
11	Middleware link state (see table below)	0	Yes	Yes	Yes	Yes	-	-
280	Roamming state (0=no, 1=yes)	0	-	-	1.8	-	Yes	-

Sources 9,10,11 and 280 can be used to read the GSM, GPRS and MW link state. Below you'll find the possible values for each one.

GSM state (GRD only)

#	GSM state
0	OFF
1	ATTACHING
2	SIM NOT INSERTED
3	PIN REQUIERED
4	PIN ERROR
5	PIN OK
6	BLOQUED
7	**LOW SIGNAL**
8	ACCESS DENIED
9	**READY**

GPRS state (GRD only)

#	GPRS state
0	OFF
1	WAIT GSM READY
2	ATTACHING
3	**CONNECTED**
4	ERROR
5	WAIT RECONNECTION

Middleware state

#	Middleware state
0	OFF
1	WAIT GPRS READY
2	-
3	CONNECTION REFUSED
4	CONNECTION FAILED
5	HOST UNREACHABLE
6	HOST CLOSED CONNECTION
7	**CONNECTED**
8	ERROR
9	WAIT RECONNECTION
10	DNS FAILURE
11	LOGGING IN (Only GRD 5.1.2)

Middleware link configuration

read_io / write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
22	MW link enabled configuration	0	5.2.2	Yes	Yes	Yes	-	-

Source/Destination 22 will allow you to enable/disable the MW link configuration.

Have in mind that this setting is stored in the device's configuration and that once the device is disconnected from the MW you won't be able to configure it remotely. If you are working with the GRD you can recover the link by using SMS commands.

Stop sending historical records to the Middleware

read_io / write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
48	Disable sending historical records to the MW (1 disabled, 0 enabled)	0	5.2.0	Yes	Yes	Yes	-	-

Source/Destination 48 will allow you to disable/enable the device sending historical records to the MW.

Have in mind that this setting is stored in the device's configuration. If you want the device to send records again to the MW you must enable this feature again.

This feature is useful if you are planning to read the historical records using and alternative method.

Example: Disable the device sending historical records to the MW.

```
write_io 48,0,1;;
```

Receiving/Sending SMS. Phonebook (GRD only)

Receiving

read_str	Description	GRD-XF-2G	GRD-XF-3G	GRD-MQ
4	SMS text	Yes	Yes	Yes
5	SMS sender's phone number	Yes	Yes	Yes

Sources 4 and 5 can be used to receive SMS. To check if there's an incoming SMS read source 4 until the length is greater than 0.

Example: Check if there's and incoming SMS

```
if 1 {
     read_str 4,a,v;
     read_str 5,b,w;
};
if a!0 {
     #Incomming SMS;
};
```

Sending

write_str	Description	GRD-XF-2G	GRD-XF-3G	GRD-MQ
4	SMS text (send order)	Yes	Yes	Yes
5	SMS recipient's phone number	Yes	Yes	Yes

Destinations 4 and 5 can be used to send SMS. First write the recipients phone number on destination 5. Then write the text to send on destination 4.

If you want to answer and incoming SMS you can write the text on destination 4 without writing the phone number,

Example: Send an SMS

```
write_str 5,'1166041241'; #Writes the recipients phone number;
write_str 4,'Hello'; #Writes the text and sends the SMS;
```

read_io	Description	GRD-XF-2G	GRD-XF-3G	GRD-MQ
39	SMS pending to be sent	5.2.4	Yes	Yes

Phonebook

read_str	Description	GRD-XF-2G	GRD-XF-3G	GRD-MQ
101 a 108	Phone book names 1 to 8	5.1.3	Yes	Yes
111 a 118	Phone book telephone number 1 to 8	5.1.3	Yes	Yes

Sources 101 to 108 will allow you to read the names on the GRDs phonebook

Sources 111 to 118 will allow you to read the phone numbers on the GRDs phonebook

These sources are useful if you want to edit an SMS recipient without having to edit the script but editing the GRD configuration only.

Example: Read phone number #5 on the phonebook and store it in variable v

```
read_str 115,a,v;
```

Sending/Receiving messages to the Script Programmer ("Traces")

read_str							

/ write_str	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
35	Script Programmer's trace window	5.2.0	Yes	Yes	Yes	Yes	Yes

Destination 35 will allow you to send messages that will be displayed on the Script Programmer's "Trace" window (available since Script Programmer V2.0).

There is a consideration about the underscore and spaces character. The space character will be replaced by the underscore character before reading with *read_str 35*. The underscore character will be replaced by the space character after sending with *write_str 35*.

If the Script Programmer is not connected to the device the text will be lost but will not affect the script performance.

Serial port in text mode

read_str / write_io	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
6	Serial port	Yes	Yes	Yes	Yes	Yes	Yes

Source/Destination 6 allows you sending and receiving strings to and from the serial port. To check if data has arrived to the serial port read source 6 until lenght is larger than 0

To send strings just write to destination 6

For this feature to work properly you must configure the serial port in "Script" mode

To send binary characters use the $ operator.

Example: Send an echo of the strings received on the serial port.

```
read_str 6,a,v;
if a!0 {
    write_str 6,v;
};
```

Serial port in binary mode

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
37	Number of bytes stored in serial port buffer (to delete these bytes use write_io 37)	0	5.2.0	Yes	Yes	Yes	Yes	Yes
38	Binary value of one byte of the serial port buffer	1 to 100	5.2.0	Yes	Yes	Yes	Yes	Yes

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
37	Delete N bytes from the serial port buffer (use together with read_io 37 and read_io 38)	0	5.2.0	Yes	Yes	Yes	Yes	Yes
38	Send a byte to the serial port (Binary value)	0	5.2.5	-	Yes	Yes	Yes	Yes

Source/Destination 37 and Source 38 allow you to get and parse binary data received at the serial port.

For this feature to work properly you must configure the serial port in "Script" mode

To send binary characters use write_str 6 and the $ operator.

Example: Wait until receiving 3 or more characters. Check if the third one is binary 126. Delete 3 bytes from the seriel port buffer.

```
read_io 37,a,0;
if a>2 {
    read_io 38,b,3;
    if b=126 {
        #The third byte is binary 126;
    };
    write_io 37,0,3;
};
```

Creating historical records

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
56	Create a digital input channel "by change" historical record	1 to 100	5.2.2	Yes	1.6	Yes	Yes	Yes
57	Create a digital output channel "by change" historical record	1 to 100	5.2.2	Yes	1.6	Yes	Yes	Yes
12	Create an analog input channel "by time" historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
14	Create an analog input channel "by alarm" maximum value historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
15	Create an analog input channel "by alarm" minimum value historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
16	Create an analog input channel "by alarm" normal value historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
13	Create a pulse input channel "by time" historical record	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes

These destinations allow creating historical records from the script besides the regular historical records.
The value of the record must be loaded in the *value* field.

Use with care to avoid generating historical records constantly.

Example: Create a "by time" record for AN2 channel every 10 seconds with the value 457

```
check_timer t
{
    timer t,10000;
    write_io 12,2,457;
};
```

Force sending reports

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
17	Force a digital input (Ix) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
18	Force a digital output (Ox) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
19	Force an analog input (ANx) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes
20	Force an pulse input (PIx) channel report	1 to 100	Yes	Yes	Yes	Yes	Yes	Yes

These destinations allow forcing sending reports from the script besides the regular reports. **The reported value will be the one current value of the channel. The *value* field will be ignored.**

Use with care to avoid generating GPRS traffic constantly (GRD only)

Example: Force a report for AN3 channel with its current value every 10 seconds

```
check_timer t
{
    timer t,10000;
    write_io 19,3,0;
};
```

Historical records memory access

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
8	Records in historical records memory	0	Yes	Yes	Yes	Yes	Yes	Yes
61	Get historical record memory channel type, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
62	Get historical record memory timestamp, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
63	Get historical record memory historical type, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
64	Get historical record memory channel number, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes
65	Get historical record memory value, use with write_io 60	0	5.2.2	Yes	Yes	Yes	Yes	Yes

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
60	Read specific register from historical records memory (use together with read_io 61 to 65)	0	5.2.2	Yes	Yes	Yes	Yes	Yes
66	Delete the first N registers from the historical records memory	-	5.2.2	Yes	Yes	Yes	Yes	Yes

These sources/destinations allow you to gain access to the historical records memory. The will be used in applications where you need to send this records using an alternative way (not using the MW).

Before reading a register you must check how many records are in the memory using *read_io 8*. Then you can invoke *write_io 60* to read a particular register and *read_io 61* to *65* to get the fields of the read register.

Finally use *write_io 66* to delete the read records.

Example: Read all the records and send them to Script Programmer traces window.

```
read_io 8,a,0;
if a>0
{
    write_io 60,0,1 ; #Reads first record in memory;
    read_io 61,b,0; #Loads in b the channel type;
    read_io 62,c,0; #Loads in c the timestamp;
    read_io 63,d,0; #Loads in d the historical type;
    read_io 64,e,0; #Loads in e the channel type;
    read_io 65,f,0; #Loads in f the value;
    w=b,'-',b,'-',c,'-',d,'-',e,'-',f,$13,$10;
    write_str 35,w; #Sends the record to the traces window;
    write_io 66,0,1; #Deletes the first record in memory;
};
```

Satellite Modem

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
55	Satellite modem state	0	5.2.2	Yes	Yes	Yes	-	-

#	Send state
-7	Not sending while connected to the MW
-6	Error while sending records
-5	Initializing modem
-4	Serial port configuration erro (Satellite modem and 19200 bps)
-3	Historical records memory empty
-2	Error while reading historical records memory
-1	Sending records
0	Ready to send
> 0	Records sent:

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ

54	Initiate sending historical records using satellite modem	0	5.2.2	Yes	Yes	Yes	-	-

If you connect an SBD Iridium modem (EDGE/ITAS) to the GRD/cLAN serial port, you can send historical records to the MW using the Iridium satellite network. Read the device user's manual for more details.

Please check the satellite modem costs before using it.

Destination 54 initiates the device to send historical records in its memory using the satellite modem. The modem must be "ready to send" to start doing it. The GRD/cLAN will send all the records it can on a single satellite message.

Please check the *satellite.sce* example that you can download from here
www.exemys.com/GRDscriptsExamples

FTP Client

read_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
47	FTP client state	0	5.2.2	Yes	-	Yes	-	Yes

#	Estado de cliente FTP
0	**IDLE**
2	CONNECTING
3	CONNECTION FAIL
7	ERROR
8	SENDING FILE
9	WAIT READY TO SEND
10	**READY TO SEND**

write_io	Description	Index	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
44	Initiate FTP client connection	0	5.2.2	Yes	-	Yes	-	Yes
46	Finish FTP client connection	0	5.2.2	Yes	-	Yes	-	Yes

write_str	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
40	Load FTP Client URL	5.2.2	Yes	-	Yes	-	Yes
41	Load FTP Client use	5.2.2	Yes	-	Yes	-	Yes
42	Load FTP Client password	5.2.2	Yes	-	Yes	-	Yes
43	Load FTP Client file name	5.2.2	Yes	-	Yes	-	Yes
45	Load FTP file text line and send	5.2.2	Yes	-	Yes	-	Yes

These sources/destinations allow you to implement and FTP client to upload text files into an FTP server. You will usually use this feature to send the historical records in text format.

To send data using the FTP client the GRD must be attached to the GPRS network (GRD only)

Please check the *ftp.sce* example that you can download from here
www.exemys.com/GRDscriptsExamples

CRC and checksums calculation

write_str	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
50	Process buffer (use together with read_str 51)	5.2.0	Yes	Yes	Yes	Yes	Yes

Use this destination to load the process buffer with a string

NMEA protocol

read_str	Description	GRD-XF-2G	cLAN-XF V1	GRD-XF-3G	cLAN-XF V2	GRD-MQ	cLAN-MQ
51	Reads process buffer adding NMEA start, end and checksum bytes (load process buffer with write_str 50 before using it)	5.2.0	Yes	Yes	Yes	Yes	Yes

Source 51 will load the NMEA sentences previously loaded in the process buffer, adding the start/end characters and the NMEA checksum.

Use this source to implement an **NMEA talker**

Example: Send NMEA sentence *GPMWV,145.8,R,87.2,K,A* to the serial port after adding the start/end characters and the NMEA checksum.

```
write_str 50,'GPMWV,145.8,R,87.2,K,A';
read_str 51,a,w;
write_str 6,w;
```

MQTT Link state and publish (GRD-MQ and cLAN-MQ)

read_io	Description	Index	GRD-MQ	cLAN-MQ
1001	MQTT-Broker link state (1=connected)	0	Yes	Yes

You can publish MQTT messages from the script

write_str	Descripción	GRD-MQ	cLAN-MQ
1001	MQTT-Load topic to publish	Yes	Yes
1002	MQTT-Load payload to publish and begin publishing	Yes	Yes

Example:

```
read_io 1001,h,0;
if h=1 {
   write_str 1001,'v1/devices/me/telemetry';
   write_str 1002,'{A1:52}';
};
```

MQTT subscription messages reception (GRD-MQ and cLAN-MQ)

You can get the messages linked to the topics subscribed using the GRD config software

It's possible to subscribe up to 10 topics.

Received messages are queued an must be read one by one.

read_io	Descripción	Indice	GRD-MQ	cLAN-MQ
1000	MQTT-Messages pending to be read	0	Yes	Yes

read_str	Descripción	GRD-MQ	cLAN-MQ
1000	MQTT. Get first message in queue.	Yes	Yes

Example:

```
read_io 1000,b,0; b
if b!0 {
   read_str 1000,c,z;
};
```

z will hold the payload

2020-11-26