Table of Contents

Introduccion 1
Script Programmer 2
Lenguaje 13
- Introduccion 13
- Versiones 13
- Variables 13
- Alias de Variables 13
- Operadores aritmeticos 13
- Estructura de un programa 13
- Funciones de control de flujo 13
- Funciones de interfaz 13
- Funciones de string 13
- Funciones de conversion 13
- Funciones matematicas y logicas 13
- Funciones de temporizado 13

Fuentes-Destinos 24
- Listado de Fuentes/Destinos 24
- Parseo NMEA 26
- Esclavo Modbus interno 28
- Mensajes al Script Programmer ("Traces") 31
- Memoria no volatil 32

Descripción

El SGW1-MB-NM con soporte de programación de script (Firmware 10.4 o superior) permiten correr en el equipo programas escritos por el usuario, haciéndolos más flexibles y potentes.

El SGW1-MB-NM continuará funcionando normalmente mientras corre el script cargado en su memoria.

Operaciones que se pueden realizar desde un script

• Operaciones **Matemáticas**
• Operaciones **Lógicas**
• Operaciones de **Temporizado**
• Guardar valores en memoria no volátil
• Lectura e interpretación de otras sentencias NMEA

Como saber mas sobre programación de scripts

Instale el software de programación de scripts "Script Programmer" que puede descargar desde www.exemys.com

En los archivos de ayuda del Script Programmer o en el sitio web de Exemys encontrará instrucciones para aprender a programar los equipos.

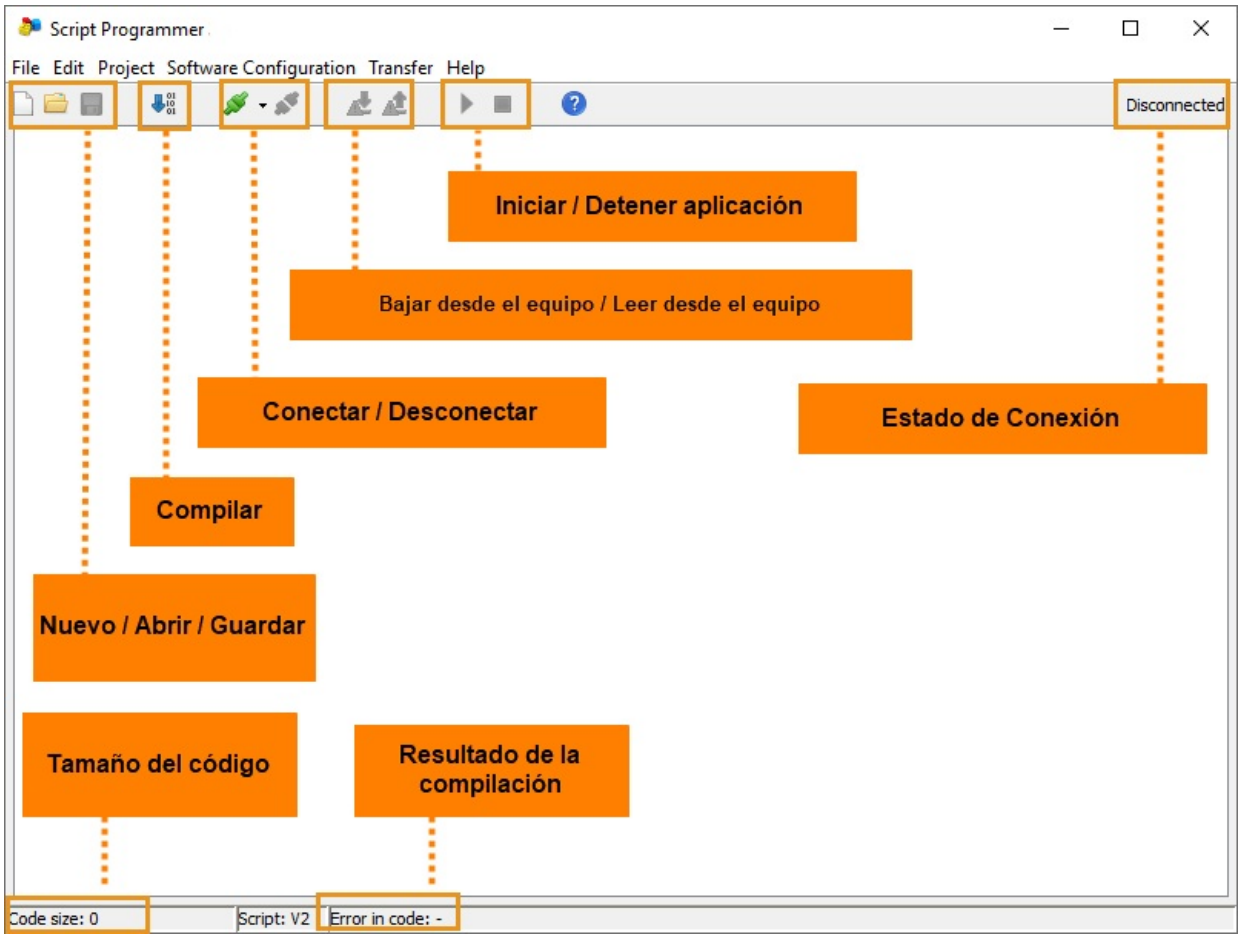
2026-04-16

Introducción

Este programa permite desarrollar, compilar y transferir scripts a equipos Exemys que lo soporten.

Descripción del Software

A continuación se ve una descripción de las funciones de la pantalla principal.



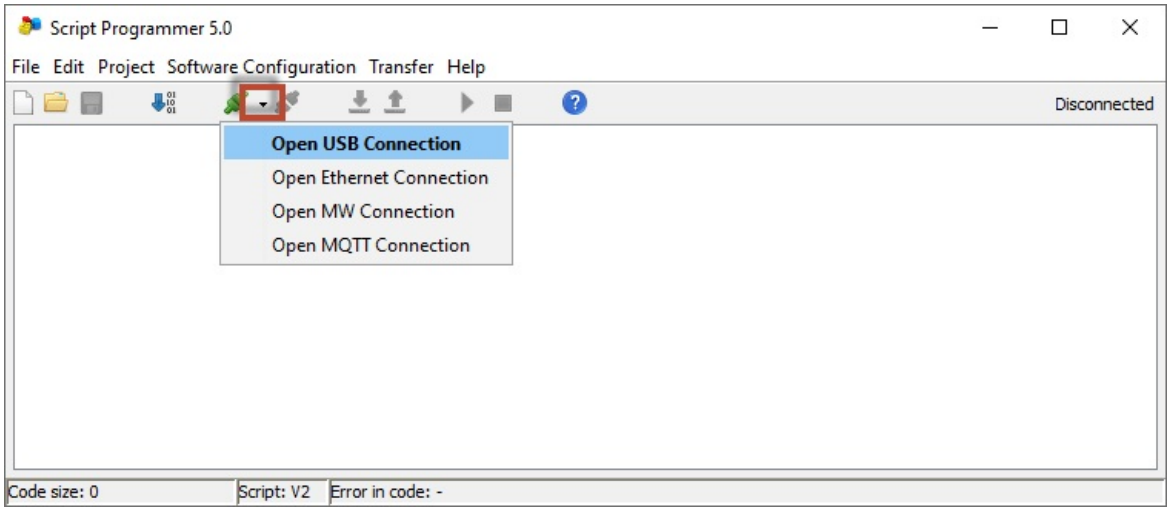
Conexión con el Equipo

Existen tres formas de conectarse, por USB, por LAN/Ethernet y via MW-XF dependiendo del equipo.

Conexión con el Equipo - USB

Primero debe conectar el equipo a la PC y asegurarse de que este desconectado del ***"GRDconfig"***

Luego desplegar el botón de Conectar y elegir ***"Open USB Connection"***

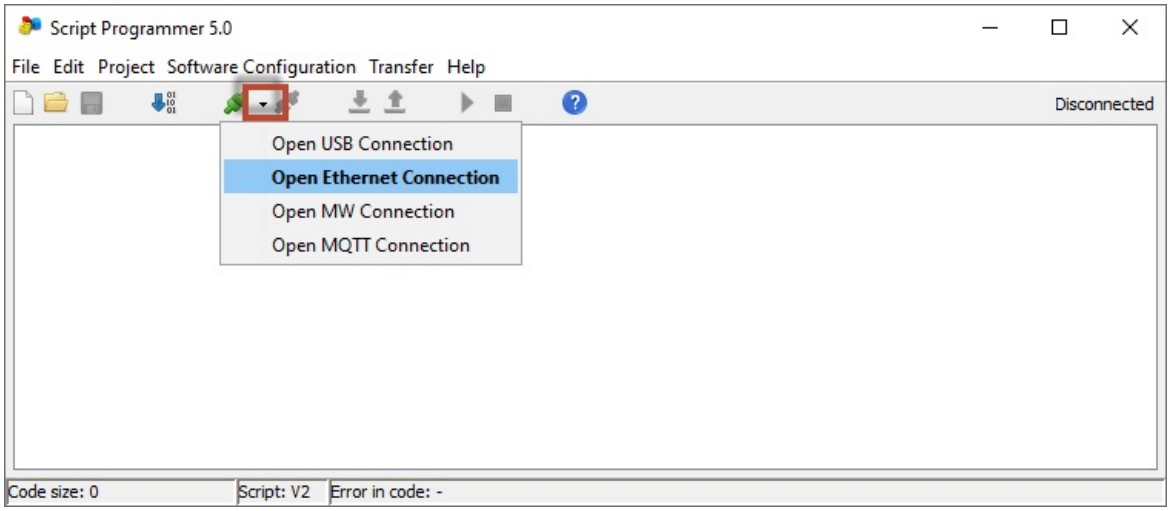


⚠	En algunos equipos con puerto USB, como el RMS1-RM y el wRemote-LoRa, debe desconectar y volver a conectar el puerto USB del equipo si previamente accedió a la consola de configuración USB.

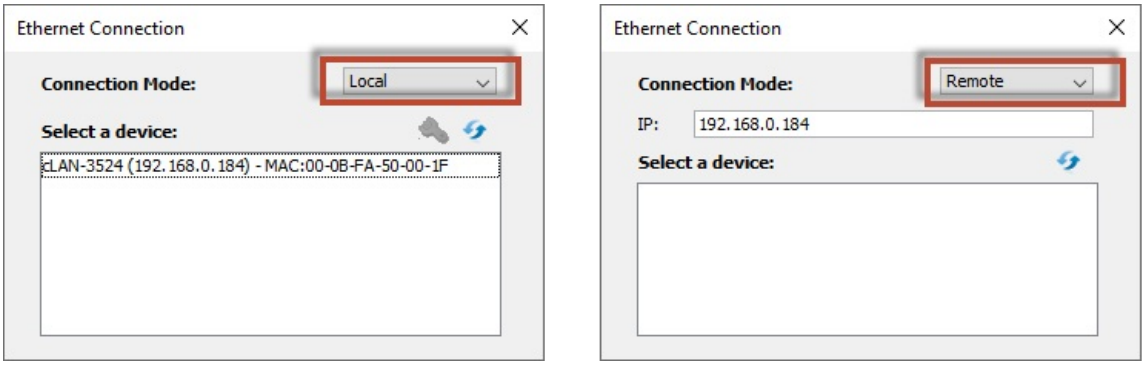
Conexión con el Equipo - LAN/Ethernet

Primero debe conectar el equipo a la misma red que la PC y asegurarse de que tenga configurada una dirección IP válida como se indica en su manual del usuario.

Luego desplegar el botón de Conectar y elegir ***"Open Ethernet Connection"***

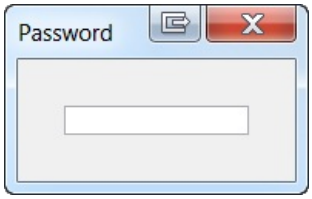


Al hacerlo deberá ver el/los equipo conectados a su red, o podrá elegir el modo **"Remote"** para indica manualmente la dirección IP del equipo si este se encuentra en otra red.



Seleccione el que quiera configurar.

Para conectarse deberá escribir la contraseña del equipo. Es la misma que usa para conectarse el MW.

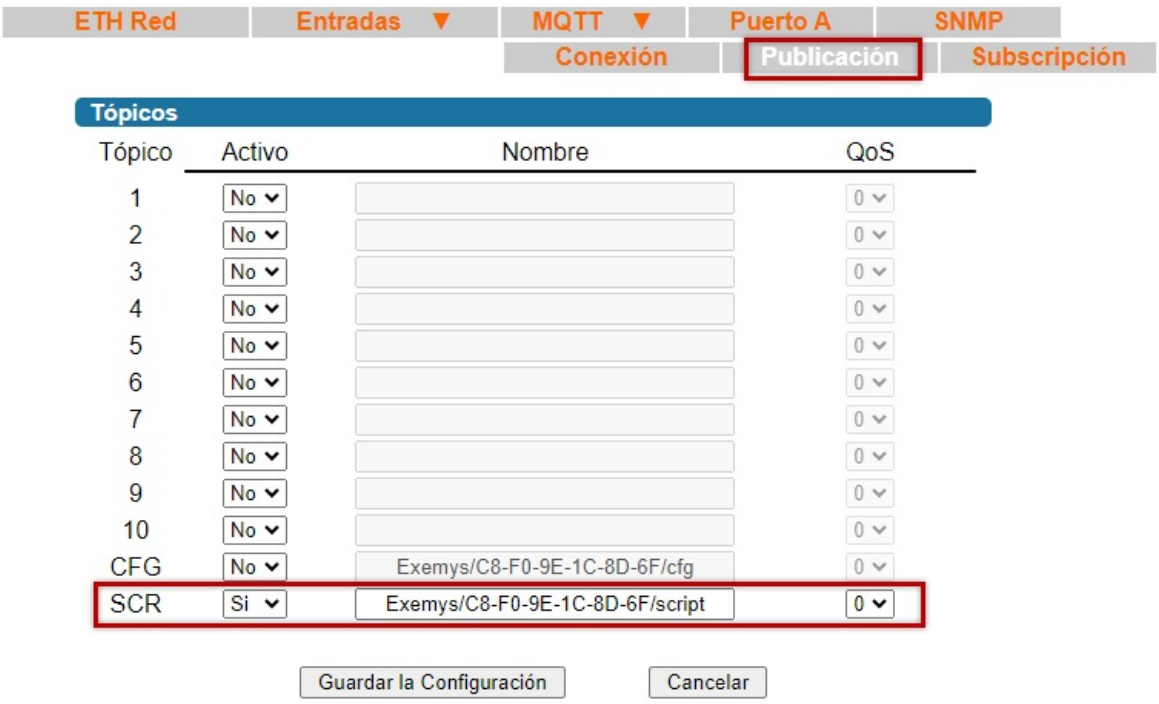


Conexión con el Equipo via MQTT

Tanto el equipo como el Script Programmer deben estar conectados al mismo broker para enviar y recibir scripts via broker MQTT de manera remota

Configuración en el equipo

Se debe configurar el tópico de base SCR usado para publicar y para subscribirse. Sobre la base el equipo agregará un texto como se explica mas adelante.



Tópico de configuración - Identificación de cada equipo

El equipo sumara el ClientID al tópico base de configuración. El tópico de suscripción y publicación en el ejemplo anterior resultante será el siguiente:

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

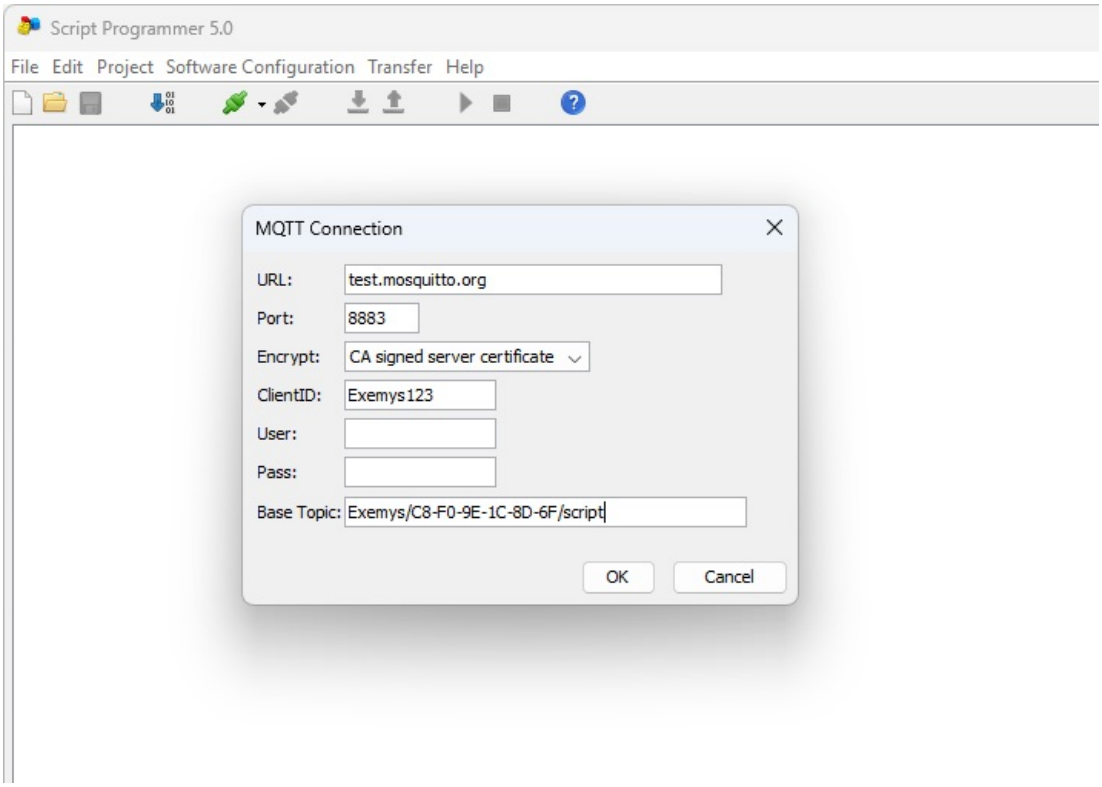
Publicar -> *Exemys/C8-F0-9E-1C-8D-6F/script/0/clientID*

El clientID usado por el equipo será el configurado en la sección MQTT.

Configuración en el ScriptProgrammer

Es necesario cargar en el configurador los parámetros del broker al cual nos queremos conectar y el tópico base a usar para configurar los equipos remotos.

Para ello hay que entrar a *"Software Configuration -> MQTT Connection"*

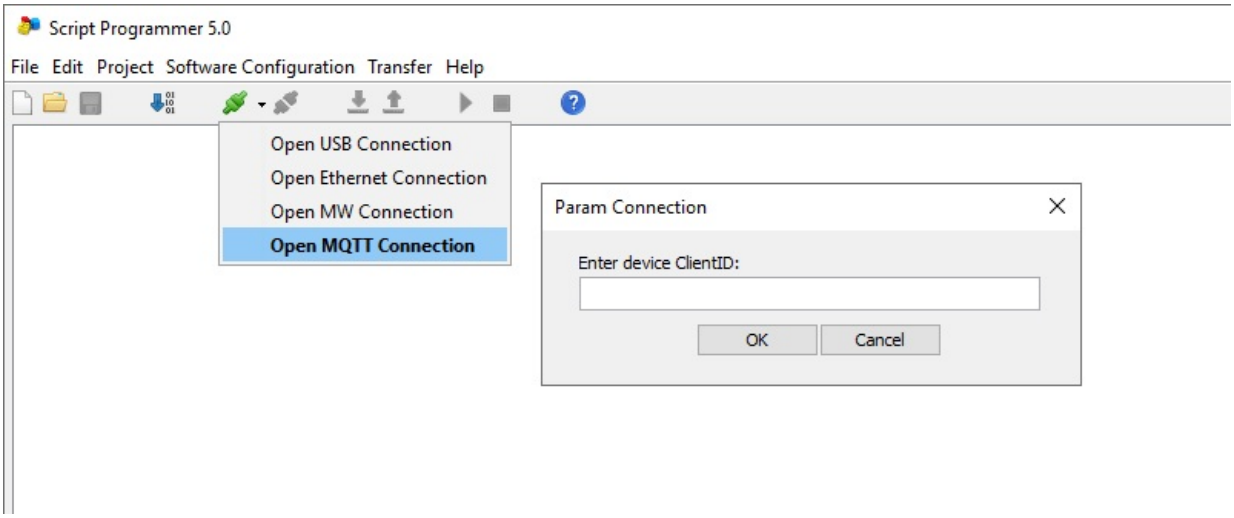


El *basic topic* debe coincidir con el configurado en los equipo.

Conexión remota

Luego desplegar el botón de Conectar y elegir ***"Open MQTT Connection"***

Allí se debe indicar el ClientID del equipo a configurar.



Una vez establecida la conexión se podrán usar los botones "Download", "Upload", "Start" y "Stop" como si estuviera conectado localmente al equipo.

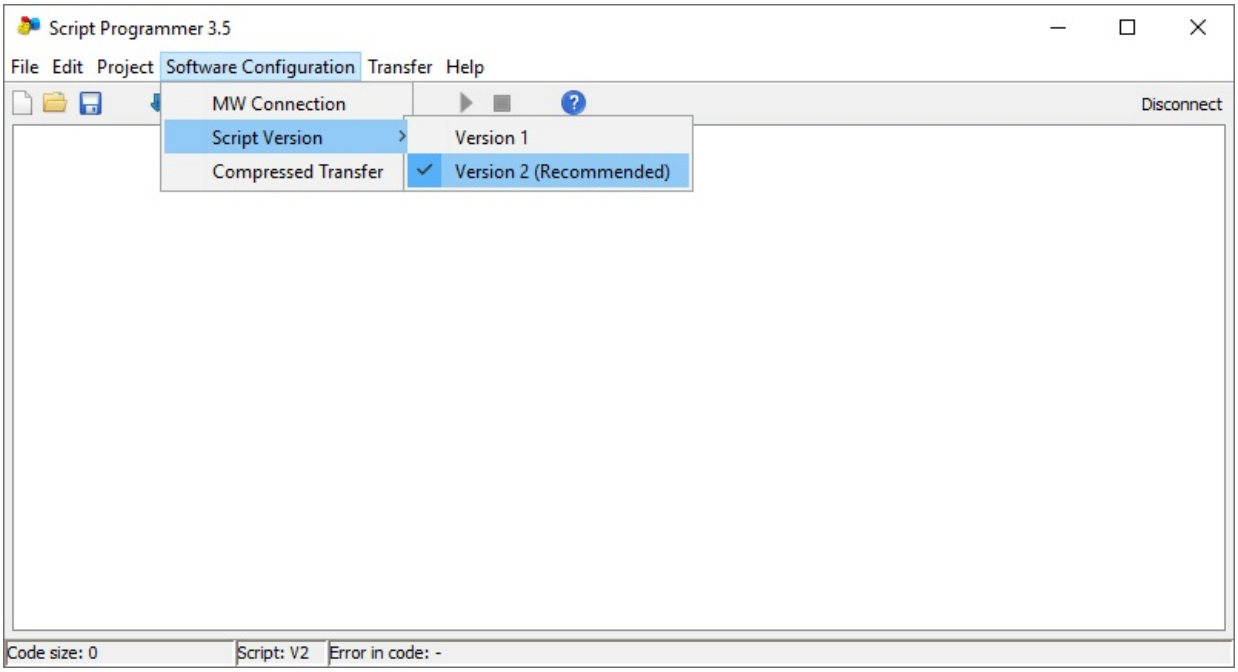
Versiones de Script 1, 2 y 3

En el menú ***"Project"***, opción ***"Properties"***, pestaña ***"Script"*** puede seleccionar entre la versión 1, 2 y 3 del script.

La versión 2 se diferencia de la 1 porque permite el doble de variables ya que pueden ir en minúscula o mayúscula.

Todos estos equipos funcionan con version con versión 2 o 3. Desde la versión 11.0 en adelante se pasan automáticamente a versión 3. Los programas escritos en versión 3 son idénticos a los de versión 2. Solo ocupan menos espacio en la memoria del equipo.

La versión 2/3 se diferencia de la 1 porque permite el doble de variables ya que pueden ir en minúscula o mayúscula.



La versión de script será usada por el Script Programmer en dos situaciones, cuando **verifique** un script o cuando trate de **enviarlo** al equipo.

Si al enviar un script la versión seleccionada no es compatible con el equipo destino verá un mensaje indicando ese error

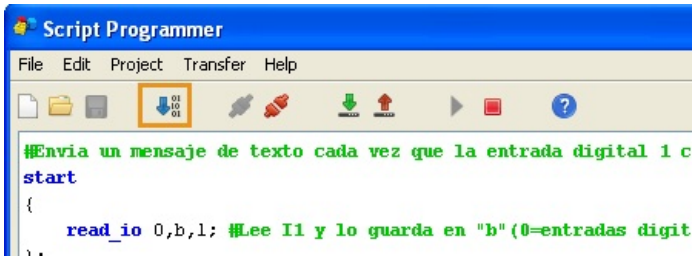
Carga y descarga de scripts

Una vez conectados al equipo podremos transferir y descargar los scripts.También de podra detener y poner en marcha.

Edición de scripts

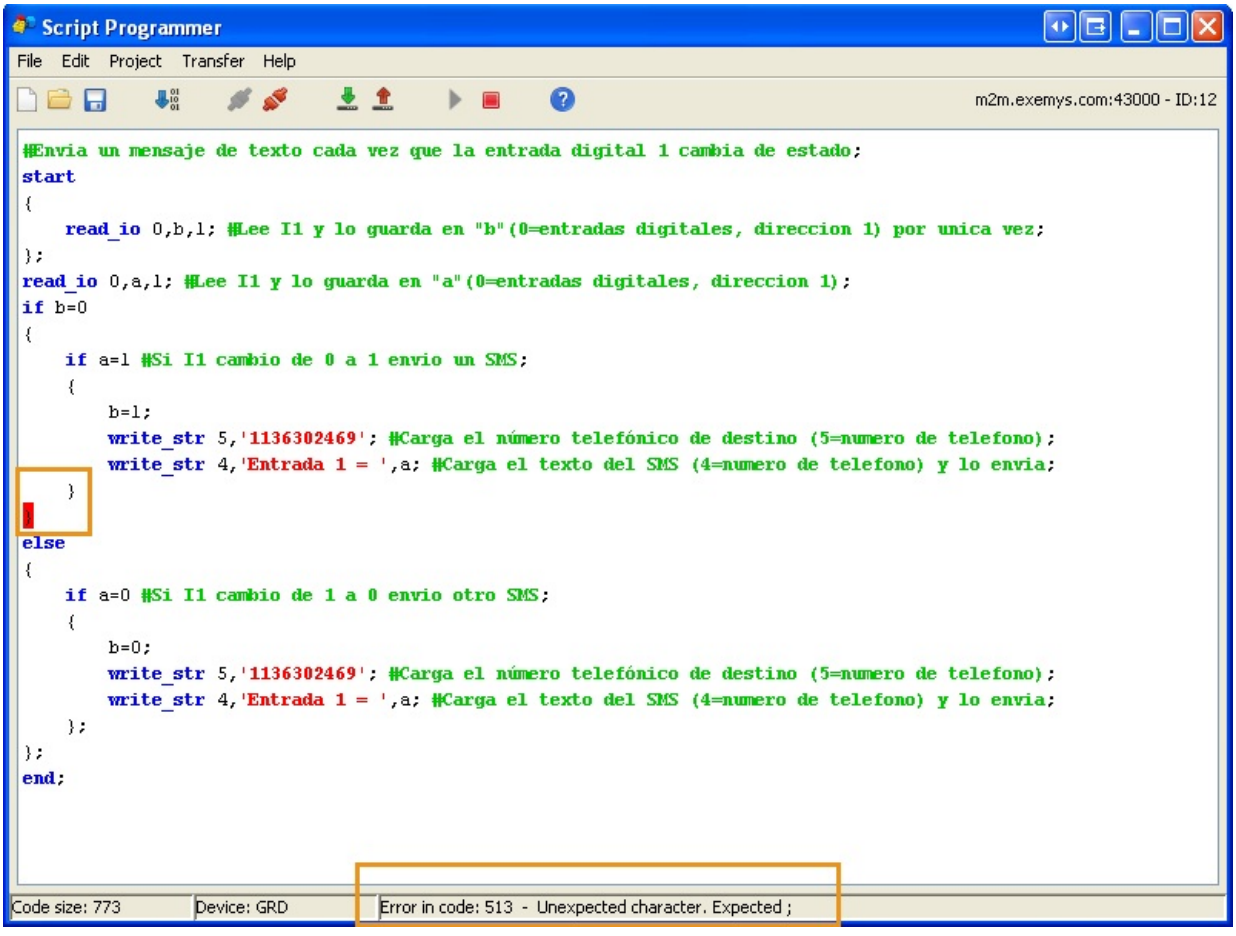
Para desarrollar un programa solo debemos escribir el código en el panel de edición, el entorno cuenta con ayuda en la escritura de las funciones y las resalta indicando su correcta escritura.

Una vez que terminamos de escribir el código con el botón ***"Verify"*** compilamos el programa y así podremos ver si este tiene algún error de sintaxis.



Al momento de compilar en la parte inferior nos indicará si el mismo tiene o no errores, si hay un error se marcará en rojo la línea en donde se encuentra y en la casilla ***"Error in Code:"*** nos dirá la línea, si no hay aparecerá un cartel indicándolo y en ***"Error in Code: NONE"***.

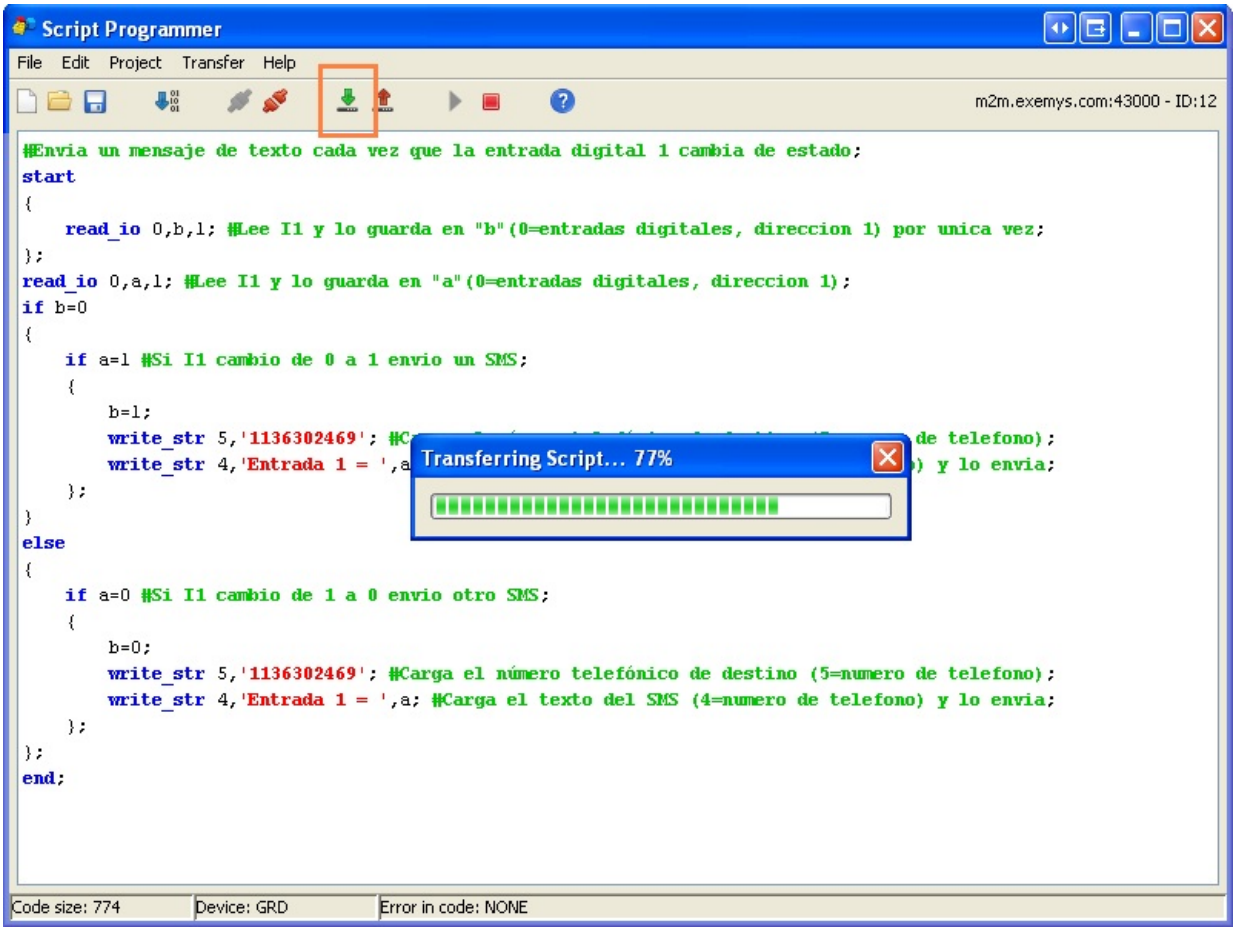
En la siguiente imagen vemos un programa con un error, en este caso falta un ***";"***.

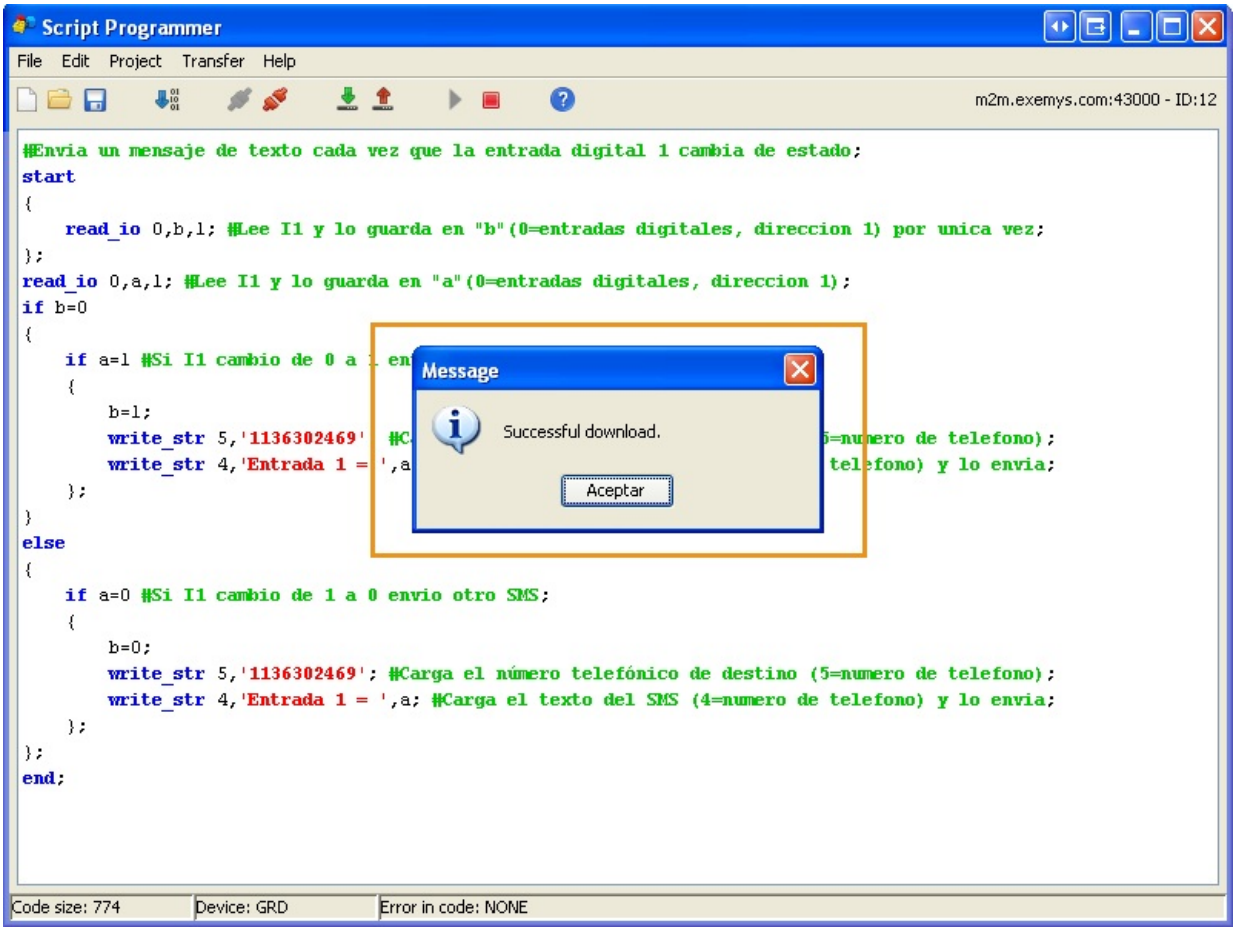


El punto y coma falta en la llave anterior a la que esta marcada en rojo, esto se debe a que el compilador nos indica que encontro un caracter que no es el esperado.

En la siguiente imagen vemos un programa sin errores.

Si no tiene errores podemos, transferir el programa al equipo haciendo clic en ***"Download to device".*** Aparecerá una ventana que nos indica el estado la descarga y luego un cartel que dice si la descarga fue exitosa o no.





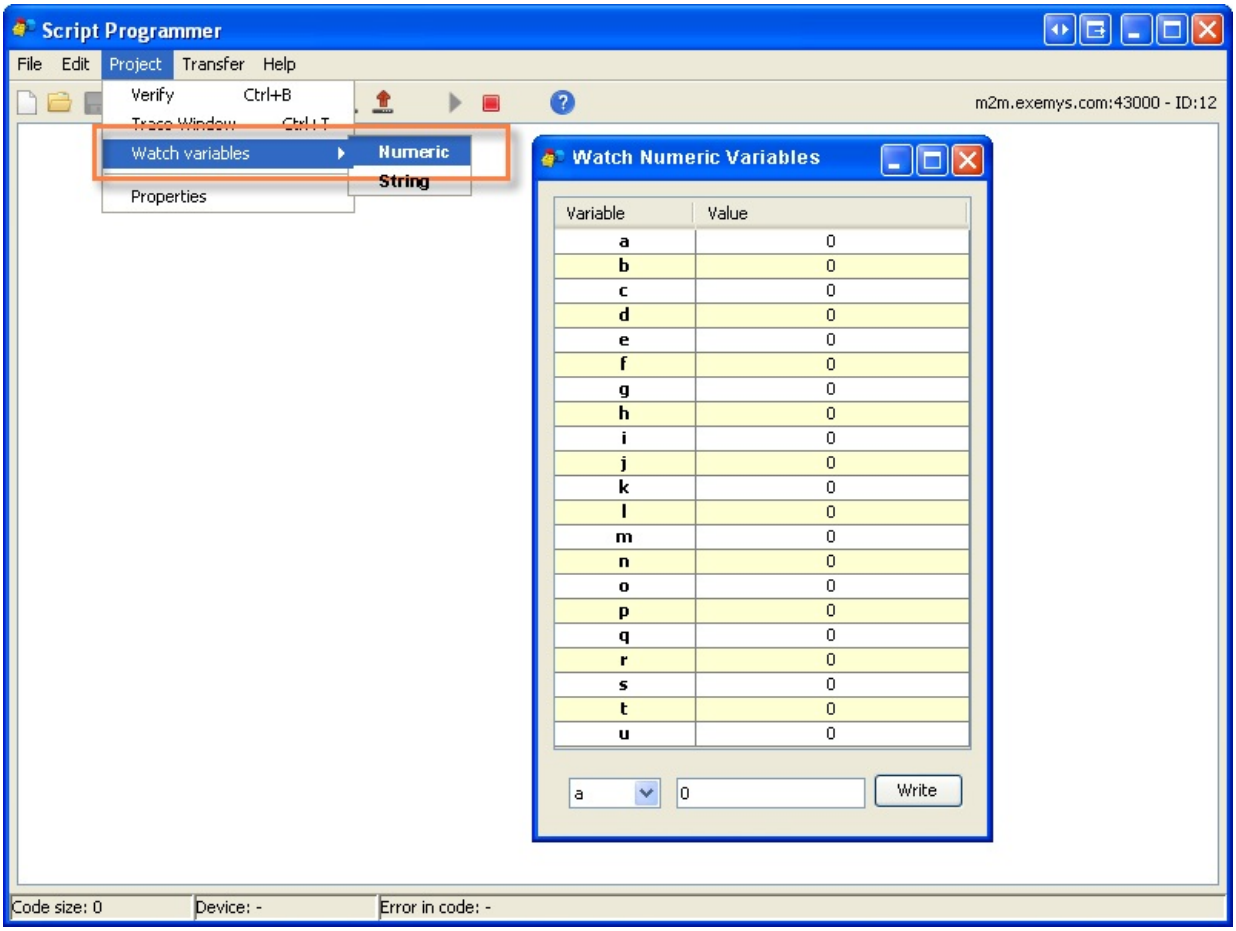
Depuración de scripts

El Script Programmer dispone de dos herramientas para la depuración de los scripts escritos.

Monitoreo de variables

Con esta herramienta puede ver el valor de las variables númericas o de tipo texto mientras el programa está corriendo. Tambíen puede modificar el valor de las variables para simular condiciones de trabajo del script.

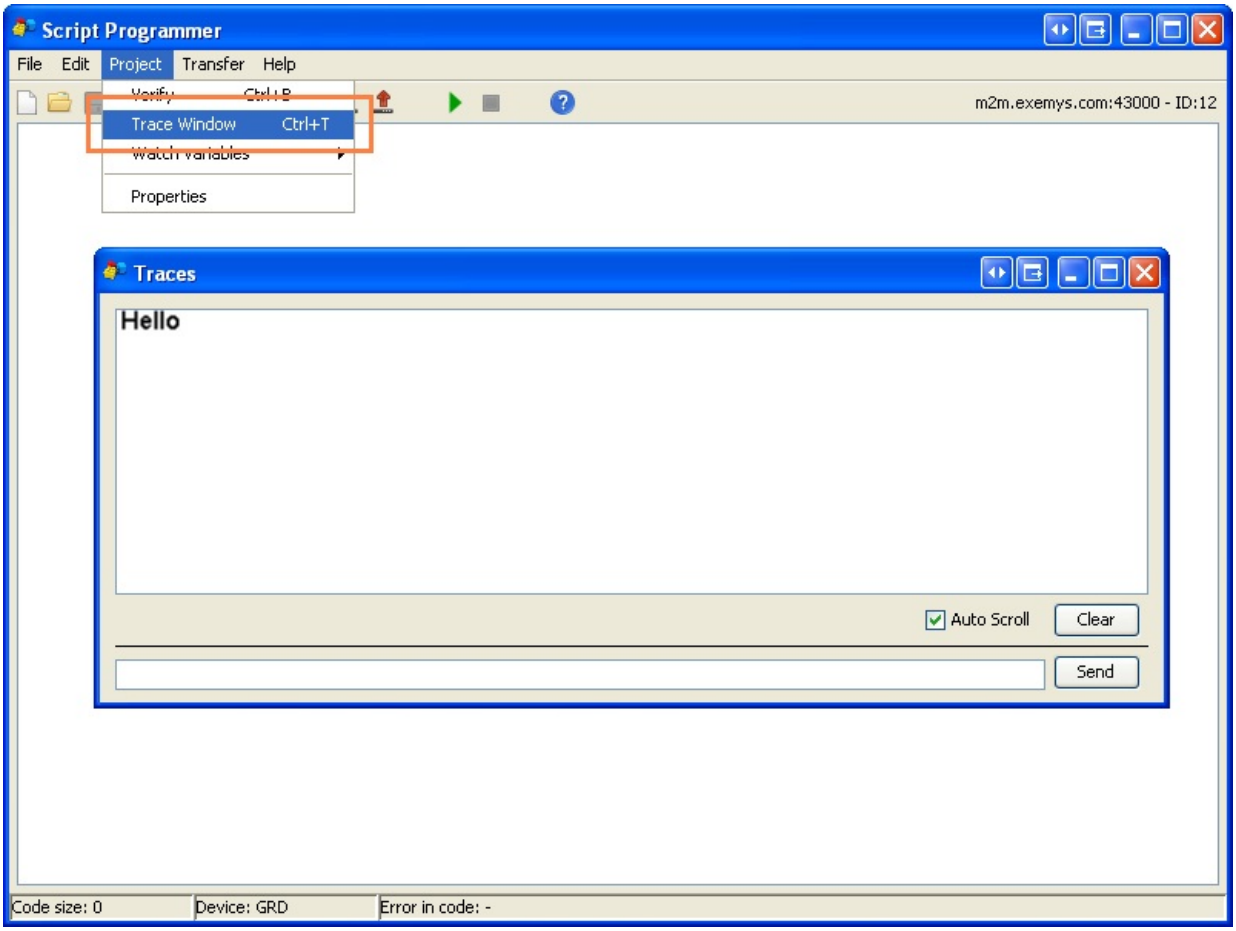
Una vez que este conectado al equipo vaya al menú **"Project",** opción **"Watch variables"** y luego **"Numeric"** o **"String"** según el tipo de variable a monitorea



Envío y recepción de trazas

Con esta herramienta puede enviar textos desde el script al Script Programmer para seguir el funcionamiento del script. Tambíen pueden enviar textos simular condiciones de trabajo del script.

Una vez que este conectado al equipo vaya al menú **"Project",** opción **"Trace Window"**



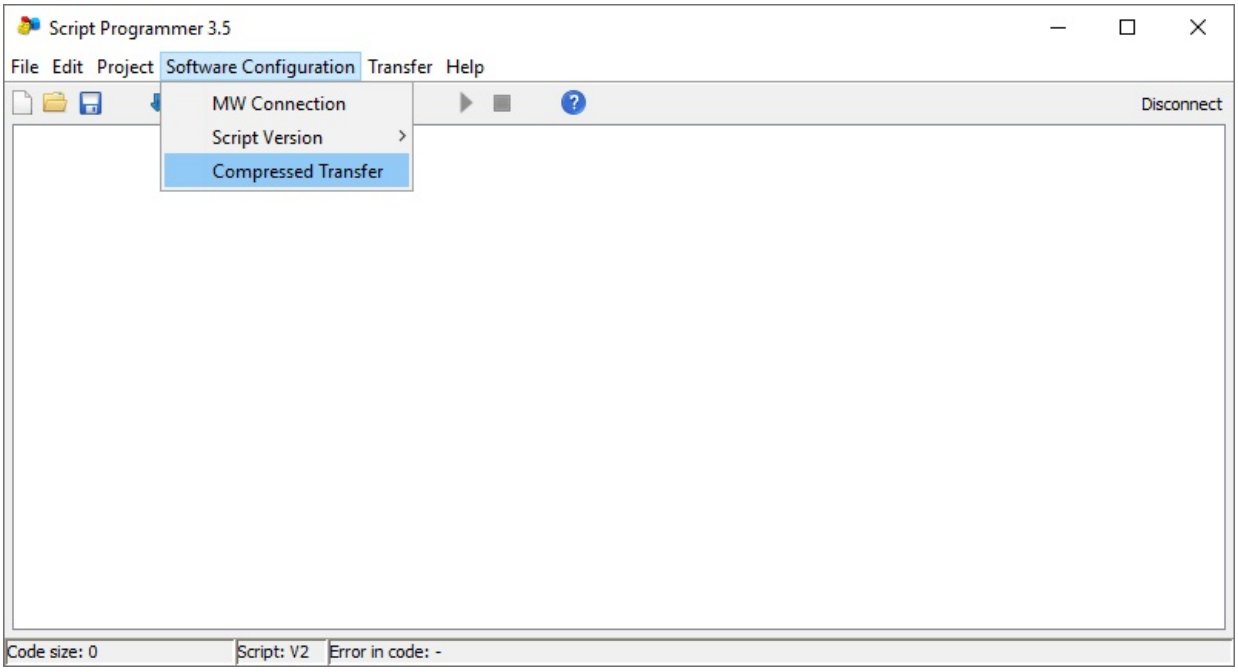
Compresión de scripts

El equipo tiene un espacio limitado para guardar el script. En casi todos los equipos máximo son 15.000 caracteres. Existen algunos casos en donde esta limitado a 5.000

Si este espacio no es suficiente para su aplicación puede usar la opción de compresión de scripts. Al hacerlo el Script Programmer quitará todos los comentarios y tabulaciones antes de enviar el programa.

Si quiere conservar los comentarios de su programa guarde una copia en su computadora.

Para habilitar la compresión vaya al menú **"Software Configuration "**, opción **"Compressed Transfer"**


Script Programmer 3.5
File Edit Project Software Configuration Transfer Help
MW Connection
Script Version
Compressed Transfer
Disconnect
Code size: 0
Script: V2
Error in code: -

2025-01-27

Introducción

El lenguaje de programación ***Exemys Script*** es de tipo bucle, esto quiere decir que se ejecuta todo el código hasta la última línea y luego vuelve a comenzar.

No hay sentencias para crear bucles dentro del programa, por lo que no se puede detener el flujo del programa en una sección de código. Se asemeja en este sentido a la programación de PLCs, aunque su sintaxis es mas cercana al C o a Pascal.

Versiones 1, 2 y 3 del script

Existe 3 versiones de script. La versión 2 se diferencia de la 1 porque permite el doble de variables ya que pueden ir en minúscula o mayúscula. En la versión 1 las variables solo se escriben en minúscula. La versión 3 ahorra entre un 20 y un 30% es espacio ocupado por el programa en la memoria del equipo, permitiendo así cargar programas mas largos. Puede notar la diferencia al presionar el botón de verificar.

Variables

Se dispone de 2 tipos de variables, numéricas del tipo ***"long"*** y de texto del tipo ***"string".***

No es necesario definir las variables ya que se tiene una cantidad fija.

En la **versión 2** de script las variable numéricas son 42, de la ***"a"*** a la ***"u"*** y de la ***"A"*** a la ***"U"*** Las de texto son 10, de la ***"v"*** a la ***"z"*** y de la ***"V"*** a la ***"Z"*** y con una longitud máxima de 100 caracteres cada una.

Como las variables numéricas son del tipo "long" cualquier operación que arroje un resultado con decimales se vera truncado.

El valor inicial de las variables numéricas es 0, en las de texto es un string vacío.

Asignar valor a una variable:

- Variables numéricas:

```
a = 652;
```

- Variables de texto:

```
v = 'Hola mundo'
```

Puede observarse que para cargar texto es necesario colocarlo entre comillas simples.

Concatenación de strings:

Para concatenar variables solamente debemos colocar una a continuación de la otra, separándolas por comas.

Por ejemplo:

```
a = 20;
u = 'La temperatura es de ';
v = ' °C';
```

Si queremos formar la frase 'La temperatura es de 20 °C ' y guardarla en otra variable hacemos lo siguiente:

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

Otra forma de hacerlo sería:

```
w = 'La temperatura es de ', a,' °C';
```

La concatenación solo puede hacerse en asignación de textos a variables string y en la función write_str

Inserción de valores ASCII en strings:

Si se quiere insertar un valor ASCII en string se puede usar el operador $. Después del operador se debe indicar al código ASCII en decimal. No se permite el ASCII 0.

Por ejemplo:

```
z = 'Hola mundo',$13,$10;
```

La inserción de valores ASCII solo puede hacerse en asignación de textos a variables string y en la función write_str

Alias de Variables

Desde la versión **6.1** del Script Programmer se pueden crear "alias" de variables para hacer mas facil la lectura del código.

Este código se puede transferir a TODOS los equipos de Exemys que soporten Script Programmer, ya que antes de envíar el código al equipo, se reemplazará el alias por la variable correspondiente.

Como los alias se envian como comentarios dentro del código, al leer el script cargado en el equipo, se volverán a reemplazar las variables por los alias, a no ser que se haya usado la opción de compresión, en donde se borran los comentarios.

Se recomiendo SIEMPRE mantener una copia de los scripts cargados en los equipos.

Ejemplo:

Antes

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

Ahora

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

Si tiene un script ya escrito, puede sumar los comentarios indicando los alias, enviarlo al equipo (sin comprimir) y luego leerlo. Al hacerlo, todas las variables que tengan alias serán reemplazadas.

Operadores aritméticos

A continuación veremos los operadores que se pueden utilizar en el script, debemos tener en cuenta que solo toma valores del tipo “long” por lo que cualquier operación que arroje resultados con decimales se verá truncada, o sea que solamente obtendremos como resultado la parte entera.

Operador	Significado
=	Igual
^	Exponencial

|	Or
&	And
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto

Por ejemplo:

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

Resultado: La variable *b* vale 135.

Estructura de un programa

Todas las instrucciones finalizan con punto y coma ***";"***.

Es muy importante tener en cuenta que el programa se ejecuta de manera que cuando se alcanza la ultima sentencia vuelve a comenzar, debido a esta particularidad disponemos de una sentencia llamada "***start"*** que se ejecuta solo una vez cuando el programa se inicia. Dentro de este bloque podremos cargar valores iniciales de variables o todo aquello que consideremos adecuado.

Todo programa debe terminar con la instrucción ***"end;"***.

A continuación veremos la estructura típica de un programa:

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

Este simple ejemplo muestra como se utiliza la función ***"start"*** para iniciar por única vez a la variable ***"a"*** con el valor 10 y luego incrementarse en 1 constantemente.
Como podemos ver todas las instrucciones que se encuentran después del bloque ***"start"*** y antes del ***"end;"*** se ejecutaran de manera cíclica.

Comentarios:

En caso de querer agregar un comentario al programa se de anteponer al texto el carácter ***"#"*** y finalizarlo con ***";"***.

Por ejemplo:

```
start
{
    a = 10;  #Inicia variable a = 10;
};
a = a + 1;    #Incrementa variable a;
end;
```

Funciones de Control de flujo

El control de flujo especifica el orden en que se van a llevar a cabo ciertas líneas del código.
En nuestro caso tenemos 3 funciones para realizar dicho control, el ***"start",*** el ***"if-else"*** y el ***"end"***.

Función ***"start"***

Se utiliza para ejecutar líneas de código por única vez en el momento que el programa se inicia.
Las instrucciones a ejecutar se colocan entre llaves y luego de la de cierre se debe colocar ***";"***.

Sintaxis:

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

Función ***"if-else"***:

Esta función se utiliza para la toma de decisiones, si la condición colocada luego del ***"if"*** es valida se ejecutaran las instrucciones que estén entre llaves debajo de este en caso contrario se ejecutaran las que estén dentro del ***"else"***.

La condición de ejecución puede utilizar los siguientes operadores:

Operador	Significado
=	Igual
!	Distinto
>	Mayor
<	Menor

La condición se escribe dejando un espacio después del ***"if"***

Sintaxis:

Solo ***"if"***:

```
if condición
{
    ...;
    ...;
};
```

Luego de la llave de cierre del ***"if"*** debe colocarse ***";"***.

"if-else":

```
if condición
{
    ...;
    ...;
}

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

En este caso el ***";"*** se coloca después de la llave de cierre del ***"else"*** y no después del ***"if"***.

Función ***"end"***

Solamente se utiliza para indicar el fin del programa luego de ejecutarla se vuelve a la primera línea del código.

Sintaxis:

```
end;
```

Funciones de interfaz (fuentes/destinos)

Estas funciones permiten tomar datos de distintas fuentes como también enviar datos a distintos destinos.

Función ***"read_io"***

Permite leer parámetros indexados de distintas fuentes, como los valores de canales de entradas digitales, salidas digitales, entradas analógicas, etc.

Se debe indicar la ***"fuente"*** del dato con un número. En caso de que corresponda, con el parámetro ***"índice"***, le decimos la posición dentro de esa fuente.
El resultado de la lectura se guarda en una variable numérica.

Sintaxis:

```
read_io fuente,variable_numerica,indice;
```

Las fuentes disponibles dependen del **equipo** donde se corre el script y la versión de script. En un futuro pueden agregarse nuevas fuentes.
Refiérase a la seeción fuentes/destinos para mas detalles.

Función ***"write_io"***

Permite escribir parámetros indexados en distintos destinos, como los valores de canales de salidas digitales, de pulsos, etc.

Se debe indicar el ***"destino"*** donde escribir con un número. En caso de que corresponda, con el parámetro ***"índice"***, le decimos la posición dentro de ese destino.

El ***"valor"*** a escribir puede ser un número o una variable numérica.

Sintaxis:

```
write_io destino,indice,valor;
```

Los destinos disponibles dependen del **equipo** donde se corre el script y la versión de script. En un futuro pueden agregarse nuevos destinos.
Refiérase a la seeción fuentes/destinos para mas detalles.

Función ***"read_str"***:

Permite leer strings originados en distintas fuentes, como lo son los datos recibidos por un puerto serie o por SMS. Para realizar esto se debe indicar la ***"fuente"*** con un número.

El resultado de la lectura se guarda en dos variables, una del tipo string y una del tipo numérica, donde se indica la longitud del string. Si no hay datos este parámetro dará 0.

Sintaxis:

```
read_str fuente,variable_numerica,variable_string;
```

Las fuentes disponibles dependen del **equipo** donde se corre el script y la versión de script. En un futuro pueden agregarse nuevas fuentes.
Refiérase a la seeción fuentes/destinos para mas detalles.

Función ***"write_str"***:

Permite escribir strings en distintos "destino", como el puerto serie o enviar SMS. Para realizar esto se debe indicar el ***"destino"*** con un número y el string a enviar.

Sintaxis:

```
write_str destino,string;.
```

Los destinos disponibles dependen del **equipo** donde se corre el script y la versión de script. En un futuro pueden agregarse nuevos destinos.
Refiérase a la seeción fuentes/destinos para mas detalles

El string puede ser una variable o un string escrito en la función. **Esta función permite concatenación e inclusión de valores ASCII.**

Funciones de string

Estas funciones permiten realizar distintas operaciones con las variables del programa del tipo string.

Función ***"is_equal"***

Compara una variable string con un string (texto fijo o variable string). Devuelve 0 si son distintos y 1 si son iguales.

Sintaxis:

```
is_equal variable_numerica,variable_string,string;
```

Ejemplo:

```
v='APAGAR BOMBA';
is_equal c,v,'APAGAR BOMBA';
if c=1 {
        #Los textos son iguales;
};
```

Función ***"finish_with"***

Determina si un variable string termina con un string en particular (texto fijo o variable string). Devuelve 0 si es falso o 1 si verdadero

Sintaxis:

```
finish_with variable_numerica,variable_string,string;
```

Ejemplo:

```
v='APAGAR BOMBA';
```

```
finish_with c,v,'BOMBA';
if c=1 {
        #El string guardado en v termina en 'BOMBA';
};
```

Función *"begin_with"*

Determina si un variable string comienza con un string en particular (texto fijo o variable string). Devuelve 0 si es falso o 1 si verdadero

Sintaxis:

```
begin_with variable_numerica,variable_string,string;
```

Ejemplo:

```
v='APAGAR BOMBA';
begin_with c,v,'APAGAR';
if c=1 {
        #El string guardado en v empieza con 'APAGAR';
};
```

Función *"contains"*:

Determina si una variable string contiene un string en particular (texto fijo o variable string). Devuelve 0 si no lo encuentra o la posición donde lo encontró si lo encuentra.

Sintaxis:

```
contains variable_numerica,variable_string,string;
```

Ejemplo:

```
v='APAGAR BOMBA';
contains c,v,'GAR';
if c>0 {
        #El string guardado en v contiene el texto 'GAR';
};
```

Función *"upper"*

Convierte todos los caracteres de una variable string a mayúsculas, guardando el resultado en la misma variable.

Sintaxis:

```
upper variable_string;
```

Ejemplo:

```
v='Apagar';
upper v;
#Ahora v es igual a 'APAGAR';
```

Función *"lower"*

Convierte todos los caracteres de una variable string a minúscula, guardando el resultado en la misma variable.

Sintaxis:

```
lower variable_string;
```

Ejemplo:

```
v='Apagar';
lower v;
#Ahora v es igual a 'apagar';
```

Función *"strlen"*

Devuelve en una variable numérica, la longitud de una variable *string*.

Sintaxis:

```
strlen variable_numerica,variable_string;
```

Ejemplo:

```
v='Apagar';
strlen c,v;
#Ahora c vale 6;
```

Función *"substr"*

Devuelve una parte de una variable string dentro de la misma variable

Sintaxis:

```
substr inicio,fin,variable_string;
```

Ejemplo:

```
v='APAGAR BOMBA';
substr 2,3,v;
#Ahora v vale 'PA';
```

Funciones de conversión

Estas funciones convierten variables de un tipo a otro.

Función *"point"*

Coloca el punto decimal a una variable numérica y la convierte a *string*.
Como parámetros se le pasa, la variable numérica, la variable string y la cantidad de decimales que queremos colocarle.

Sintaxis:

```
point variable_string,variable_numerica,decimales;
```

Ejemplo:

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

Función *"aton"*

Convierte un número dentro de string a una variable numérica. Empieza en el principio del string y

termina en donde encuentra un valor no numérico o el fin del string.

Sintaxis:

```
aton variable_numerica,variable_string;
```

Ejemplo:

```
v='123 RPM';
aton c,v;
#Ahora c vale 123;
```

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

Estas funciones convierten un ***time_stamp*** a día, mes, año, hora, minuto, segundo y número de día de la semana, respectivamente.
Como vimos anteriormente, el fecha/hora actual se lee con la función ***read_io*** tipo **7** y se guarda en una variable numérica que representa la cantidad de segundos desde el 01/01/2000.

Sintaxis:

```
day dia,timestamp;

month mes,timestamp;

year año,timestamp;

hs hora,timestamp;

min minutos,timestamp;

sec segundos,timestamp;

nday dia_semana,timestamp;
```

La función ***"nday"*** devuelve el número de día de la semana comenzando por el domingo = 0.

Ejemplo:

```
read_io 7,e,0; #Lee la hora actual en e;
day f,e;
month g,e;
year h,e;
hs i,e;
min j,e;
sec k,e;
#La fecha y hora actual es f/g/h i:j:k;
```

Funciones matemáticas y lógicas

Estas funciones realizan ciertas operaciones matemáticas especiales.

Funcion *"neg"*

Se utiliza para negar una variable numérica. La negación se produce a nivel de *bit*.

Sintaxis:

```
neg resultado,inicio;
```

Primero se coloca la variable en la cual se quiere obtener el resultado y luego la variable a negar.

Ejemplo:

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

# b vale 4294934972 (FFFF81BC);
```

Función *"sqtr"*

Realiza la raíz cuadrada. Como el Exemys Script solo maneja variables enteras de tipo "long", la parte decimal del resultado se verá truncada. Para evitar esto se recomienda multiplicar antes de realizar la operación.

Sintaxis:

```
sqrt resultado,inicio;
```

Ejemplo:

```
a=225;
sqrt b,a;
#Ahora b vale 15;
```

Función *"scale"*

Permite escalar una variable utilizando la ecuación de la recta que pasa por 2 puntos.

Sintaxis:

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

Ejemplo: Escalar la señal 4-20mA de la entrada AN1 en un valor de 0 a 500

```
read_io 2,a,1; #a = AN1;
scale c,a,400,2000,0,500;
#Ahora c tiene el valor escalado;
```

Funciones de temporizado

Estas funciones permiten controlar el flujo del programa en función de tiempos.

Funciones *"timer"* y *"check_timer"*

Con estas funciones podemos definir un periodo de tiempo y ejecutar instrucciones cuando este se cumpla.
La forma de uso es la siguiente. Primero ejecutamos la función ***"timer"*** a la cual le pasamos como parámetros una variable numérica y un tiempo en milisegundos. Luego se consulta esta variable con la función ***"check_timer"***.

Sintaxis:

```
timer variable_numerica,tiempo_en_milisegundos;
...
check_timer variable_numerica
{
    ...
    ...
};
```

Como podemos ver, con la función ***"timer"*** definimos el tiempo, y con ***"check_timer"*** consultamos la variable cargada anteriormente. Cuando se cumple el periodo, las instrucciones colocadas entre llaves se ejecutan.
Debemos tener en cuenta que una vez cumplido el periodo la condición siempre será verdadera, y si volvemos a realizar un ***"check_timer"*** se ejecutaran nuevamente las instrucciones. Normalmente se carga de vuelta la variable dentro del bloque "***check_timer"***

Nota: Las funciones de temporizado no deben usarse en aplicaciones de precisión de tiempo, ya que el temporizado puede presentar cierta dispersión.

2026-03-19

Introducción

Con las funciones read_io, write_io, read_str y write_str se puede acceder a múltiples funciones adicionales. En esta sección del manual se listan las distíntas fuentes/destinos y luego de explica su uso por función.

Listado de fuentes/destinos

	Se indica la versión de firmware desde la cual está disponilble una fuente/destino en caso que se haya agregedo a la versión inicial.

Fuente / Destino	Indice	Valor	R/W	Descripción	Función	Firmware
5	7	0/1	write_io	NMEA - Capturar todas las trama (en 0, solo las que no toma el equipo)	NMEA Configuración	10.5
	8	0/1	write_io	NMEA - Ignorar LRC (Para test)		10.5
402	21	0 / 1 a X	write_io	Buffer NMEA recepción - Selecciona (0) y posiciona (>1)	NMEA Recepció nNMEA	10.5
405		0 a 200	write_io	Buffer NMEA recepción - Limpia Buffer		10.5
405	-		read_str	Buffer NMEA - Recibir y parsear texto (antes seleccionar buffer)		10.5
405	-	0 a 200	read_io	Buffer NMEA recepción - Cantidad de bytes (antes seleccionar buffer)		10.5
404	6	-	read_io	NMEA - Entero o ASCII		10.5
	7		read_io	NMEA - Flotante (lo afecta expontente write_io 403,1)		10.5
5	2	0 / 1	write_io	Habilitar mapa Modbus interno	Esclavo Modbus	10.5
	3	1 a 247	write_io	ID Modbus interno, no puede ser igual al ID de la tabla interna		10.5
	4	1 a 64000	write_io	Corre el inicio del mapa Modbus interno (por defecto 1)		10.5
402	3	1 a 1000	write_io	Buffer Modbus - Selecciona y posiciona (Auto incremento para escritura)		10.5
403	1	10^N -2 -> 0.01 -1 -> 0.1 0 -> 1 1 -> 10 2 -> 100	write_io	Buffers - Configura exponente para flotantes		10.5
	2	0-> MSB / 1 ->LSB 0-> MSW / 1 ->LSW	write_io	Buffers - Configura orden bytes/words		10.5
	1		read/write_io	Buffers - Entero 8 bits sin signo		10.5

404	2	Numero según formato, exponente y orden	read/write_io	Buffers - Entero 8 bits con signo		10.5
	3		read/write_io	Buffers - Entero 16 bits sin signo		10.5
	4		read/write_io	Buffers - Entero 16 bits con signo		10.5
	6		read/write_io	Buffers - Entero 32 bits con signo		10.5
	7		read/write_io	Buffers - Flotante 32 bits		10.5
35	-		read/write_str	Traces del Script Programmer	Traces	10.5
21	1 a 20	0 a 2147483647	read/write_io	Memoria no volátil para números	Mem. No volatil	10.5
121 a 125	-		read/write_str	Memoria no volátil para texto 1 a 5		10.5

2026-04-16

Lecturas de valores de GPS

Fuente / Destino	Indice	Valor	R/W	Descripción	Función
5	7	0/1	write_io	NMEA - Capturar todas las trama (en 0, solo las que no toma el equipo)	NMEA Configuración
	8	0/1	write_io	NMEA - Ignorar LRC (Para test)	
402	21	0 / 1 a X	write_io	Buffer NMEA recepción - Selecciona (0) y posiciona (>1)	NMEA Recepció nNMEA
405		0 a 200	write_io	Buffer NMEA recepción - Limpia Buffer	
405	-		read_str	Buffer NMEA - Recibir y parsear texto (antes seleccionar buffer)	
405	-	0 a 200	read_io	Buffer NMEA recepción - Cantidad de bytes (antes seleccionar buffer)	
404	6	-	read_io	NMEA - Entero o ASCII	
	7		read_io	NMEA - Flotante (lo afecta expontente write_io 403,1)	

`write_io 5,7,1;` Fuerza a que todas las tramas NMEA lleguen al script. Incluso aquellas que el equipo puede interpretar normalmente

`write_io 5,8,1;` Hace que el equipo ignore el LRC. Sirve para probar tramas manualmente sin tener que calcular el LRC

`write_io 402,21,0;` Selecciona al buffer NMEA para luego procesarlo

`write_io 405,21,0;` Limpiar el buffer para preparase para recibir otra trama

`read_io 405` Permite saber se llego una trama indicando su largo en bytes

`read_str 405` Lee la trama que acaba de llegar y guardarla en una varible tipo string.

`read_io 404,6` Permiten leer el dato de una posición especifica de la sentencia NMEA recibida. Se puede leer como un entero si el número no tiene coma, o el valor ASCII si es una letra.

`read_io 404,7` Es igual que el 6 pero para valores con decimales. Como el script maneja solo enteros se puede multiplicar por un factor para tener de alguna manera los decimales. Por ejemplo `write_io 403,1,2` lo multiplica por 100.

Ejemplo completo: Lectura de datos de sentencia NMEA.

Si se envía la trama $VDCUR,A,2,0.5,150.3,1.2,W,150.3,1.1,B,N*2F

En 'g' quedará el ASCII 65 que representa la letra 'A'

En 'd' quedará el numero 2

En 'e' quedará el numero 15030 (150.3 x 100)

```
start{
# write_io 5,7,0; #Toma todas las sentencia NMEA;
 write_io 5,8,1; #IGNORAR CRC, solo para debug;
 write_io 5,2,1; #Habilitacion esclavo Modbus;
```

```
 write_io 5,3,10; #ID esclavo Modbus 10;
 write_io 5,4,1; #Direccion Inicial 1 (40001);

};

write_io 402,21,0; #Buffer NMEA;
read_io 405,b,0; #Hay en cola?;
if b > 0{
read_str 405,c,v; #Lee texto para luego parsearlo ;

 #Buffer NMEA PARSE 2 como entero/letra;
 write_io 402,21,2;
 read_io 404,g,6;

 #Buffer NMEA PARSE 3 como entero;
 write_io 402,21,3;
 read_io 404,d,6;

 #Buffer NMEA PARSE 5 como flotante;
 write_io 402,21,5;
 write_io 403,1,2; #x 100;
 read_io 404,e,7;

 write_io 405,21,0; #Limpio el buffer NMEA;

};

end;
```

2026-04-16

Esclavo Modbus Interno

El SGW1-MB-NM cuenta con grupos de registros Modbus. Uno es el estándar descripto en el manual del usuario.

El otro es dedicado al script y permite principalmente interpretar otras tramas NMEA y ponerlas a disposición del usuario en registros Modbus.

Este cuenta con una zona de memoria de 1000 palabras que pueden ser leidas o escritas desde el script.

Con la configuración de fábrica los registros van desde el Holding Register 1 al 1000 (40001 al 41000).

El destino 5 / índice 2 habilita el esclavo interno de scripts

El destino 5 / índice 3 indica que ID se usará para este esclavo. Deber ser diferentes a los configurados en la consola.

El destino 5 / índice 4 permite correr la dirección de inicio de este mapa. El valor por defecto es 1 (40001)

Las fuentes/destinos 403 y 404 usados para acceder al buffer pueden ser compartidos por otra funciones.

Despues de elegir el buffer y la posición con el destino 402, la dirección se autoincrementa a medida que se cargan datos.

Fuente / Destino	Indice	Valor	R/W	Descripción	Función
5	2	0 / 1	write_io	Habilitar mapa Modbus interno	Esclavo Modbus
	3	1 a 247	write_io	ID Modbus interno, no puede ser igual al ID de la tabla interna	
	4	1 a 64000	write_io	Corre el inicio del mapa Modbus interno (por defecto 1)	
402	3	1 a 1000	write_io	Buffer Modbus - Selecciona y posiciona (Auto incremento para escritura)	
403	1	10^N -2 -> 0.01 -1 -> 0.1 **0 -> 1** 1 -> 10 2 -> 100	write_io	Buffers - Configura exponente para flotantes	
	2	**0-> MSB** / 1 ->LSB **0-> MSW** / 1 ->LSW	write_io	Buffers - Configura orden bytes/words	
404	3	Numero según formato, exponente y orden.	read/write_io	Buffers - Entero 16 bits sin signo	
	4		read/write_io	Buffers - Entero 16 bits con signo	
	6		read/write_io	Buffers - Entero 32 bits con signo	
	7		read/write_io	Buffers - Flotante 32 bits	

Ejemplo de escritura: Habilitar el esclavo interno de scripts con el ID 10 y direccion inicial 1. Luego cargar los siguientes valores en registros Modbus desde el 40001

Registro Modbus	Valor (orden byte/word)	Valor hexadecimal
40001	-3000 (MSB)	F448

40002	1000 (MSB)	03E8
40003:4	-70000 (MSW)	FFFEEE90
40005:6	70000 (MSW)	00011170
40007:8	-70000 (LSB)	EE90FFFE
40009:10	70000 (LSB)	11700001
40011	1000 (LSB)	E803

```
start {
  write_io 5,2,1; #Habilitacion esclavo Modbus;
  write_io 5,3,10; #ID esclavo Modbus 10;
  write_io 5,4,1; #Direccion Inicial 1 (40001);
};

write_io 402,3,1; #Elige buffer Modbus y posiciona en direccion 1 (41001);

write_io 403,2,0;#MSB;
write_io 404,4,-3000; #40001;
write_io 404,4,1000; #40002;
write_io 403,2,0;#MSW;
write_io 404,6,-70000; #40003:4;
write_io 404,6,70000; #40005:6;
write_io 403,2,1;#LSW;
write_io 404,6,-70000; #40007:8;
write_io 404,6,70000; #40009:10;
write_io 403,2,1;#LSB;
write_io 404,4,1000; #40011;

end;
```

Ejemplo de lectura: Habilitar el esclavo interno de scripts con el ID 10 y direccion inicial 1. Leer los valores de los registros Modbus 40001 a 40011 y cargarlos en variables con distintos formatos.

Registro Modbus	Valor hexadecimal	Valor variable
40001	F448	a=-3000
40002	03E8	b=1000
40003:4	FFFEEE90	c=-70000
40005:6	00011170	d=70000
40007:8	EE90FFFE	e=-70000
40009:10	11700001	f=70000
40011	E803	g=1000

```
start {
  write_io 5,2,1; #Habilitacion esclavo Modbus;
  write_io 5,3,10; #ID esclavo Modbus 10;
  write_io 5,4,1; #Direccion Inicial 1 (40001);
};

write_io 402,3,1; #Elige buffer Modbus y posiciona en direccion 1 (41001);

write_io 403,2,0;#MSB;
write_io 402,3,1; read_io 404,a,4; #41001;
write_io 402,3,2; read_io 404,b,4; #41002;
write_io 403,2,0;#MSW;
write_io 402,3,3; read_io 404,c,6; #41003:4;
write_io 402,3,5; read_io 404,d,6; #41005:6;
write_io 403,2,1;#LSW;
write_io 402,3,7; read_io 404,e,6; #41007:8;
write_io 402,3,9; read_io 404,f,6; #41009:10;
write_io 403,2,1;#LSB;
write_io 402,3,11; read_io 404,g,4; #41011;
```

```
end;
```

Ejemplo completo: Lectura de datos de sentencia NMEA y mapeo del resultado en esclavo Modbus.

```
#$VDCUR,A,1,0.5,150.3,1.2,W,150.3,1.1,B,N*2F;

start{
# write_io 5,7,0; #Toma todas las sentencia NMEA;
 write_io 5,8,1; #IGNORAR CRC, solo para debug;
 write_io 5,2,1; #Habilitacion esclavo Modbus;
 write_io 5,3,10; #ID esclavo Modbus 10;
 write_io 5,4,1; #Direccion Inicial 1 (40001);

};

write_io 402,21,0; #Buffer NMEA;
read_io 405,b,0; #Hay en cola?;
if b > 0{
read_str 405,c,v; #Lee texto para luego parsearlo ;

 #Buffer NMEA PARSE 2 como entero/letra;
 write_io 402,21,2;
 read_io 404,g,6;

 #Buffer NMEA PARSE 3 como entero;
 write_io 402,21,3;
 read_io 404,d,6;

 #Buffer NMEA PARSE 5 como flotante;
 write_io 402,21,5;
 write_io 403,1,2; #x 100;
 read_io 404,e,7;

 write_io 405,21,0; #Limpio el buffer NMEA;

 write_io 402,3,1; #Elige buffer Modbus y posiciona primer registro del mapa
seleccionado con write_io 5,4,N;

 write_io 404,3,g; #40001=(UINT16)g;

 write_io 403,2,0; #MSB;
 write_io 404,6,d; #40002:3=(SINT32)d;

 write_io 403,2,0; #MSW;
 write_io 404,7,e; #40004:5=(FLOAT32)e;
};

end;
```

2026-04-16

Envio/Recepción de mensajes al Script Programmer ("Traces")

Fuente /Destino	R/W	Descripción	Función
35	read/write_str	Traces del Script Programmer	Traces

El destino 35 permite enviar texto a la ventana de "Traces" en el Script Programmer. Esto es particularmente util cuando se está probando un script nuevo.

Hay una consideración con respecto a los caracteres de espacio y guión bajo. El caracter de espacion será reemplazado por un guión bajo antes de leer con *read_str 35*. El guión bajo será reemplazado por un espacio luego de enviar con *write_str 35*.

Si el Script Programmer no se encuenta conectado al escribir en el destino 35 el texto simplemente se perderá pero no afectará al funcionamiento del programa.

2024-04-23

Lectura/Escritura de memoria no volátil

El equipo puede guardar texto o numeros en memoria no volátil para conservar los valores aunque el equipo se apague.

Fuente / Destino	Indice	Valor	R/W	Descripción	Función
21	1 a 20	0 a 2147483647	read/write_io	Memoria no volátil para números	Mem. No volatil
121 a 125	-		read/write_str	Memoria no volátil para texto 1 a 5	

Para números

La fuente/destino 21 permite leer y escribir hasta valóres numéricos en la memoria no volátil del equipo.

Ejemplo: Leer el valor numérico almacenado en al posición 15 de la memoria no volátil en la variable g

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

Para texto

Las fuentes/destinos 121 a 125 permiten leer y escribir hasta 5 textos en la memoria no volátil del equipo.

Ejemplo: Escribir la palabra 'hola' en la tercera posición de la memoria no volátil para textos.

```
write_str 123,'hola';
```

2024-04-23