

# Table of Contents

Introduccion GRD-XF .....	1
Script Programmer GRD-XF .....	3
Lenguaje .....	9
Introduccion .....	9
Versiones .....	9
Variables .....	9
Alias de Variables .....	9
Operadores aritmeticos .....	9
Estructura de un programa .....	9
Funciones de control de flujo .....	9
Funciones de interfaz .....	9
Funciones de string .....	9
Funciones de conversion .....	9
Funciones matematicas y logicas .....	9
Funciones de temporizado .....	9
Fuentes-Destinos .....	20
Fuentes para "GRD-XF" .....	20
Canales de entrada/salida .....	24
Mensajes al Script Programmer ("Traces") .....	25
Memoria no volatil .....	26
Modbus - Lectura directa de consultas .....	27
GSM - Estados .....	28
MW - Estados de conexion .....	29
SMS - Envio/Recepcion y agenda telefonica. ....	31
Reportes - Forzado .....	33
Historicos - Generacion .....	34
Historicos - Acceso a memoria .....	35
Historicos - Bloqueo envio a MW .....	37
Reloj de tiempo real .....	38
Puerto serie - Modbus Texto .....	39
Puerto serie - Modbus Binario .....	39
Modem satelital .....	41
Calculo de checksums y CRCs .....	43
FTP - Cliente .....	44

## Descripción

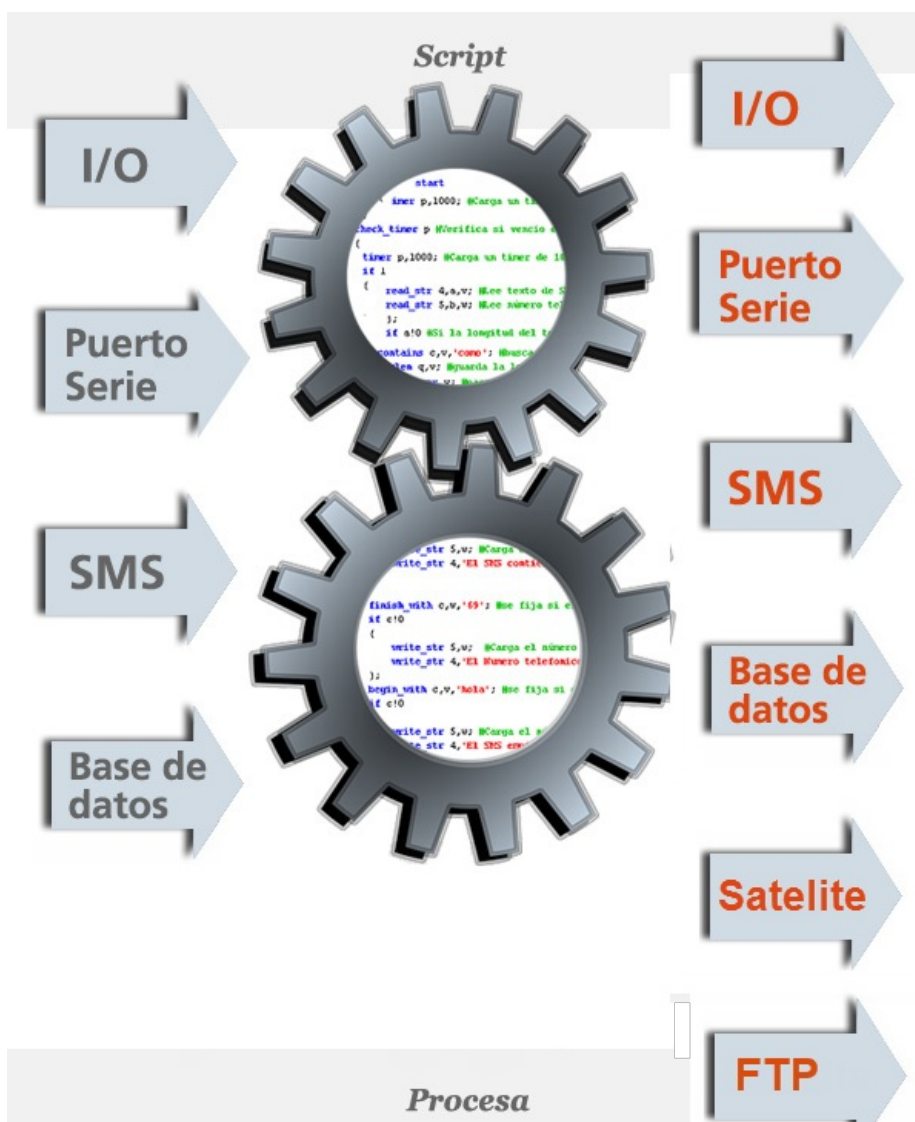
Los GRD-XF permiten correr en el equipo programas escritos por el usuario, haciéndolos más flexibles y potentes.

El GRD-XF continuará funcionando normalmente mientras corre el script cargado en su memoria.

A continuación se listan algunas de las funciones a realizar vía script. Consulte el resto de la documentación por funciones extras.

## Operaciones que se pueden realizar desde un script

- Operaciones **Matemáticas**
- Operaciones **Lógicas**
- Operaciones de **Temporizado**
- Lectura de las entradas propias del equipo y de variables Modbus
- Encendido y Apagado de salidas digitales
- Envío y recepción de **SMS**
- Envío por cliente **FTP**
- Interpretación de datos del **Puerto serie**
- Envío de datos por modem **Satelital** externo



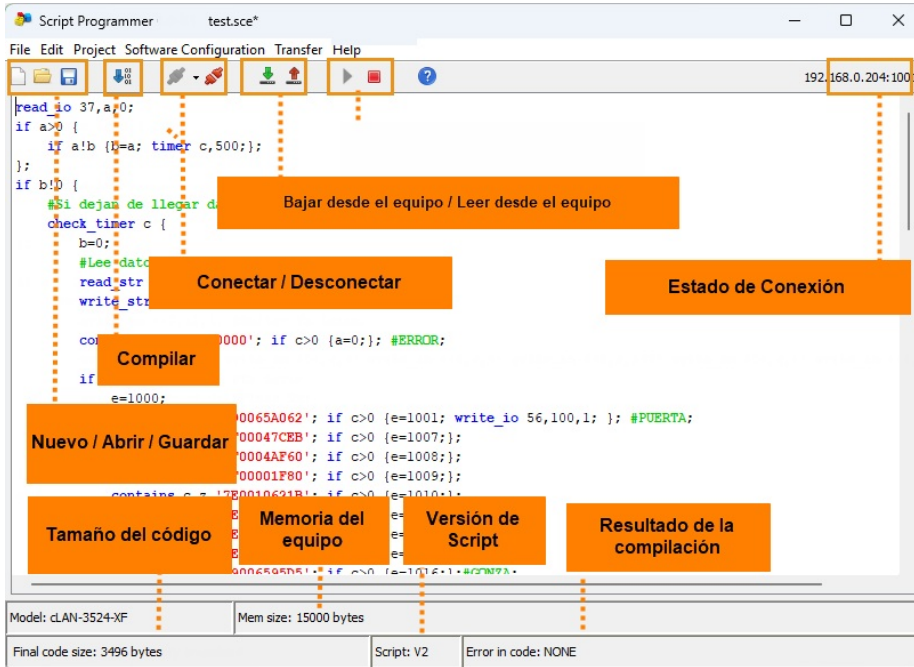
2025-12-22

**Introducción**

Este programa permite desarrollar, compilar y transferir el script al GRD-XF. Para utilizarlo, primero tenemos que asegurarnos de que el "GRDconfig" funciona correctamente y podemos comunicarnos con el GRD-XF.

**Descripción del Software**

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



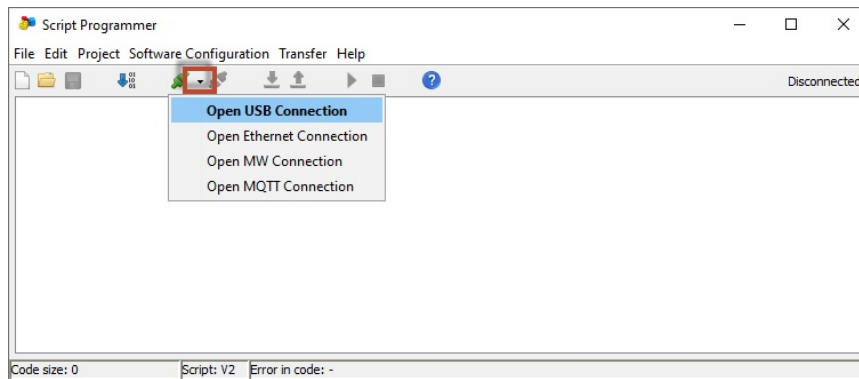
**Conexión con el Equipo**

Existen dos formas de conectarse, por USB y via MW-XF

**Conexión con el Equipo - GRD-XF por USB**

Primero debe conectar el GRD-XF a la PC y asegurarse de que este desconectado del "GRDconfig"

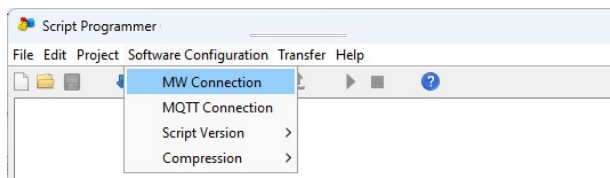
Luego desplegar el botón de Conectar y elegir "Open USB Connection"



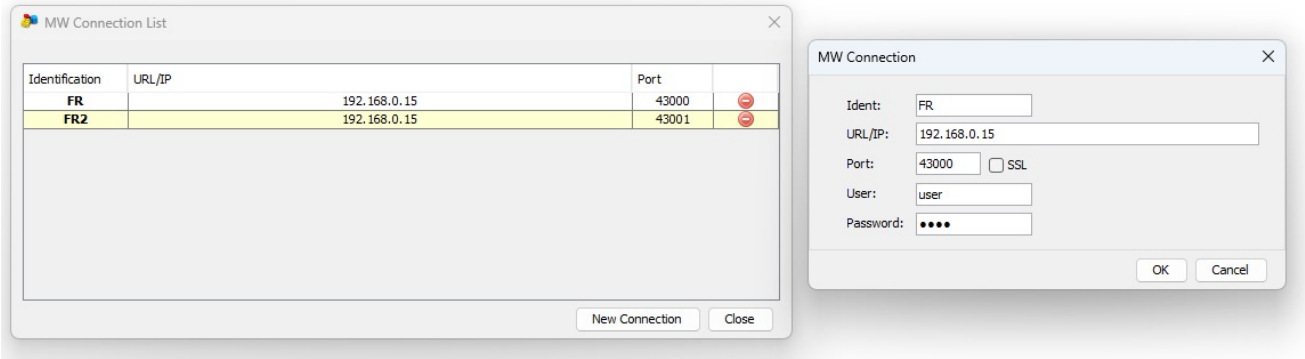
**Conexión con el Equipo - vía MW-XF**

El equipo debe estar conectado al Middleware para poder configurarlo en forma remota. **La versión del Middleware debe ser 4.2.0 o superior para soportar carga/descarga de scripts remota.**

Es necesario cargar en el configurador los parámetros de o los MW al cual/es nos queremos conectar para así tener acceso remoto a los GRDs que este tiene/n conectados, para ello hay que entrar a "Software Configuration -> MW Connection"

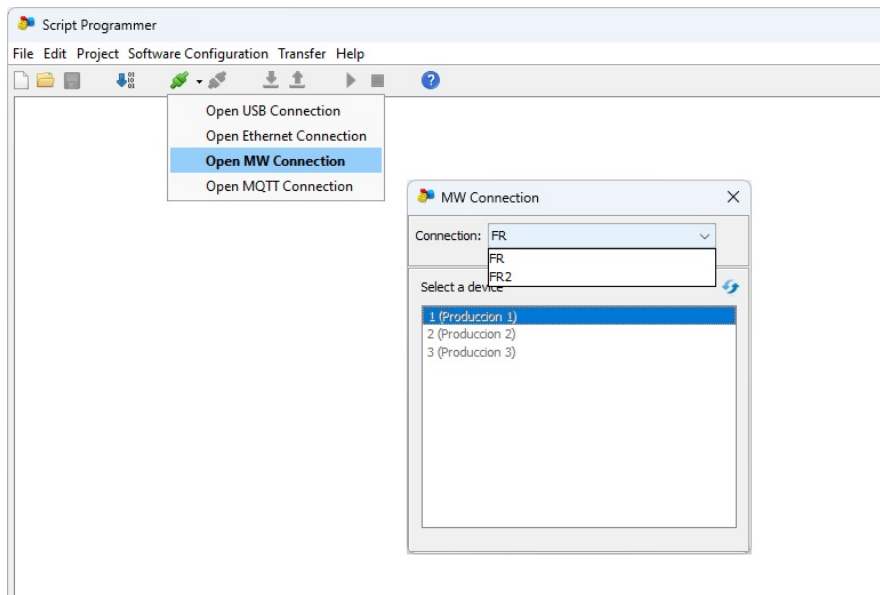


Y agregar tantas URL/IP, puerto y credenciales como sean necesarias presionando "New Connection"



Para conectarse a un equipo de manera remota seleccionar "Open MW Connection" en el botón de conexión. Luego de esto aparecerá una ventana listando las conexiones a MW configuradas y la lista de los equipos.

Nótese que hay equipos que están en color gris y otros en negro, los de color negro están disponibles y los grises no están conectados al MW en ese momento.

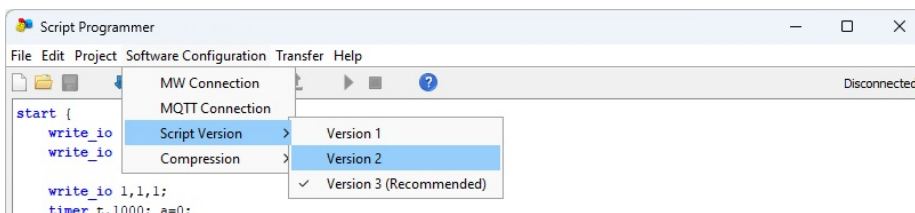


**Versiones de Script 1, 2 y 3**

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

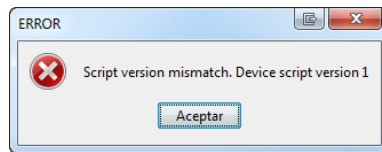
Solo los GRD-XF-2G y cLAN V1.x funcionan con versión 1. El resto funciona con versión 2, pero desde la versión 11.0 en adelante se pasan automáticamente a versión 3. Los programas escritos en versión 3 son idénticos a los de versión 2. Solo ocupan menos espacio en la memoria del equipo.

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



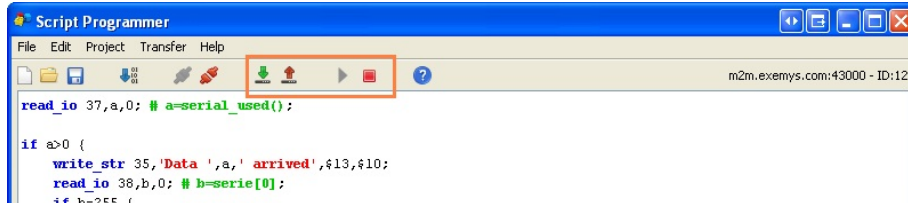
La versión de script será usada por el Script Programmer en dos situaciones, cuando **verifique** un script o cuando trate de **enviarlo** al equipo.

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

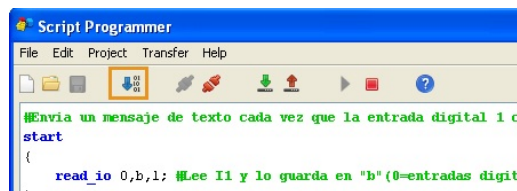
Una vez conectados al equipo podremos transferir y descargar los scripts. También de podra detener y poner en marcha.



### Edición de scripts

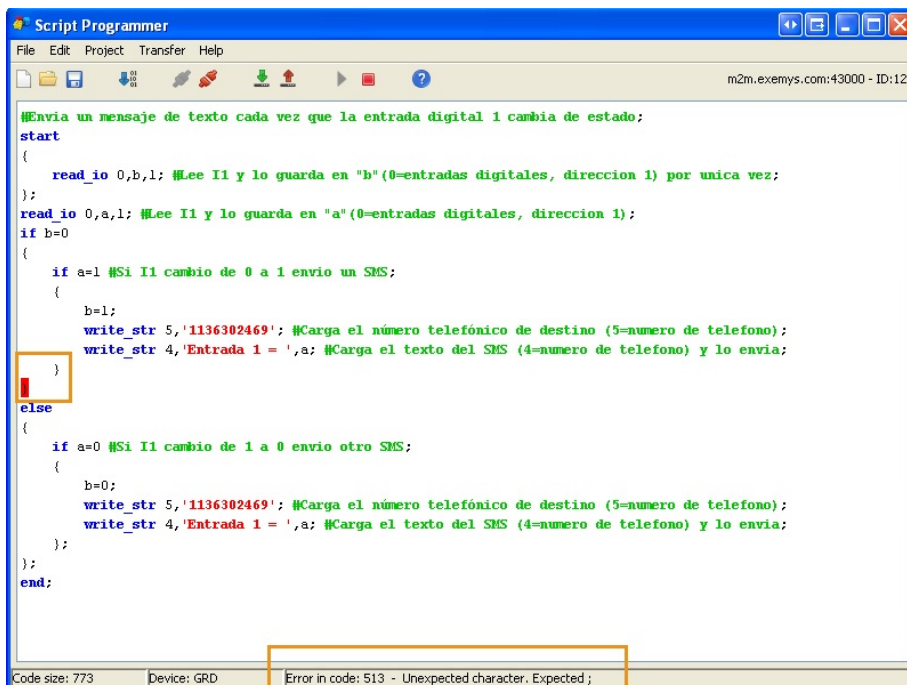
Para desarrollar un programa solo debemos escribir el código en el panel de edición, el entorno cuenta con ayuda en la escritura de las funciones y las resalta indicando su correcta escritura.

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 **“;”**.

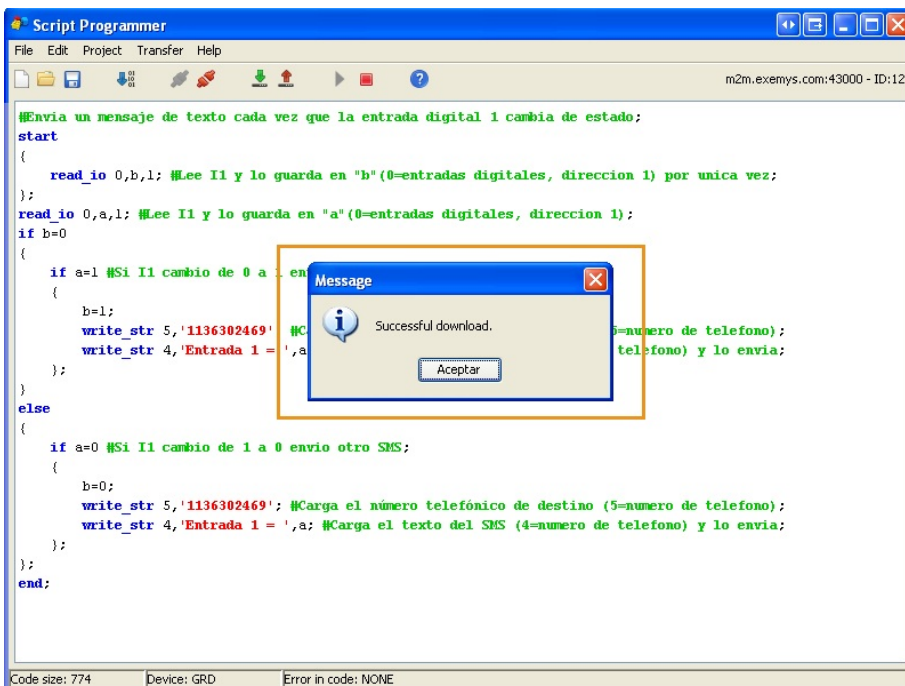
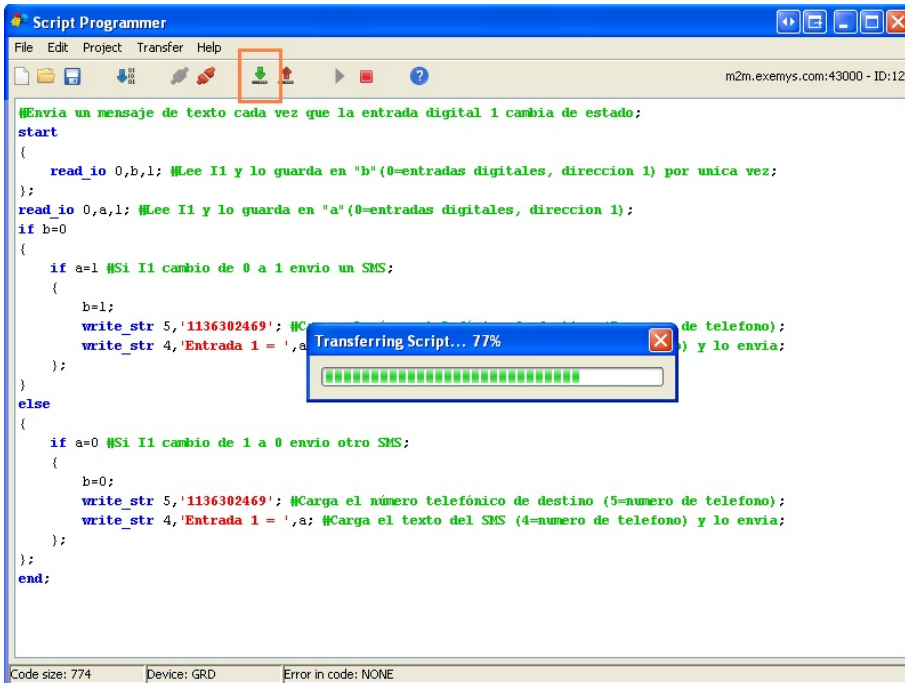


El punto y coma falta en la llave anterior a la que esta marcada en rojo, esto se debe a que el compilador nos indica que encontro un caracter que no es el esperado.

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.



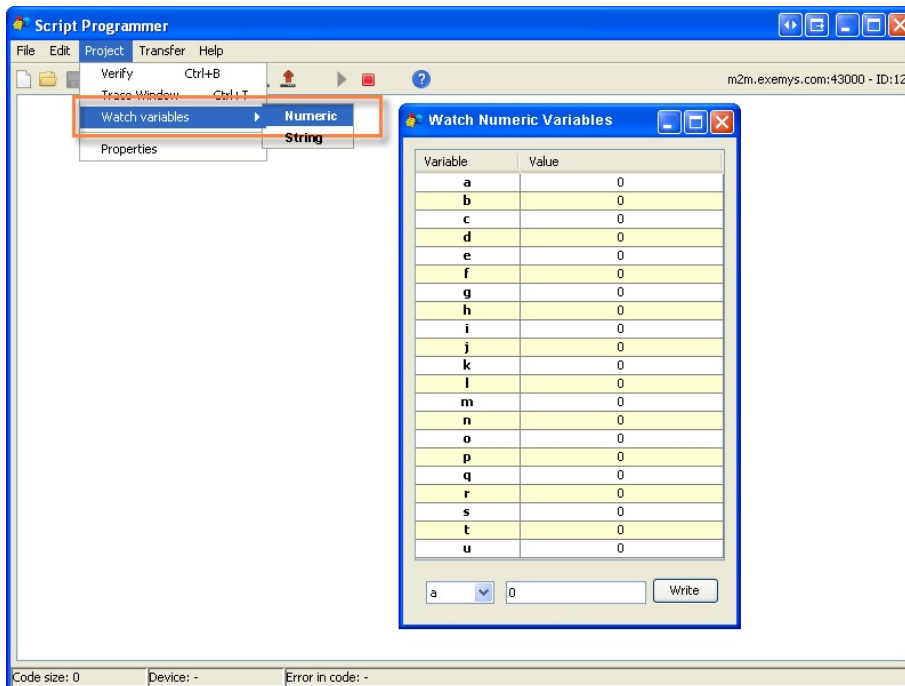
**Depuración de scripts**

El Script Programmer dispone de dos herramientas para la depuración de los scripts escritos. Los GRD deben tener la versión de firmware 5.2.0 o superior para soportar estas opciones. El cLAN siempre tiene estas opciones disponibles.

**Monitoreo de variables**

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

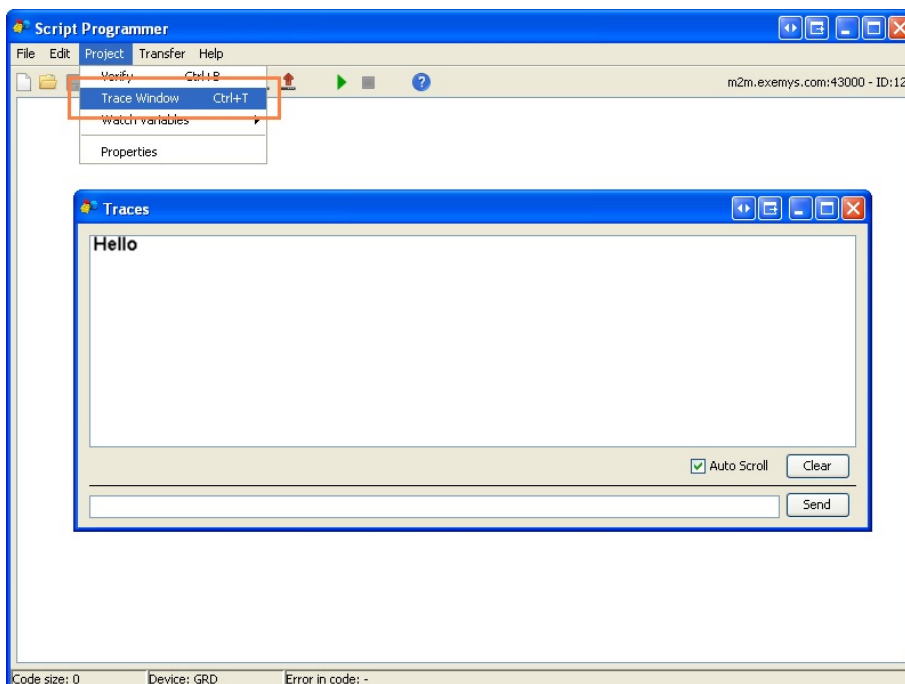
Una vez que este conectado al equipo vaya al menú **"Project"**, opción **"Watch variables"** y luego **"Numeric"** o **"String"** según el tipo de variable a monitorear



### Envío y recepción de trazas

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

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



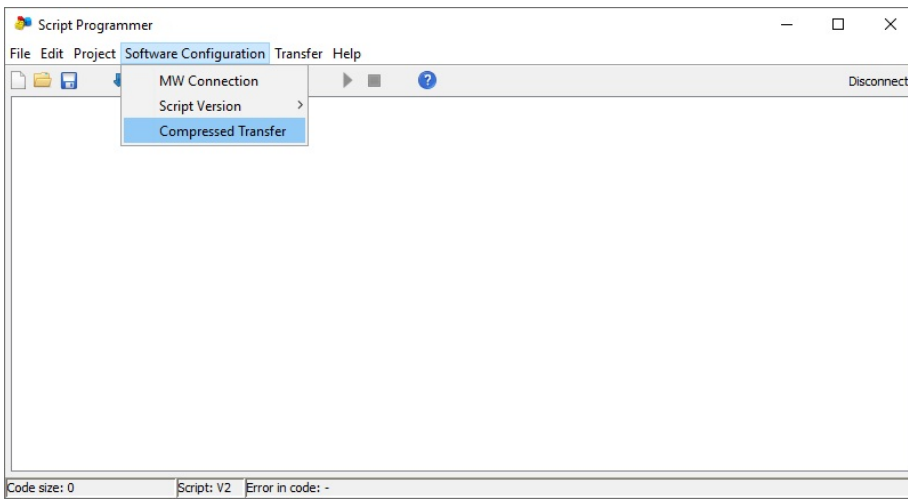
### Compresión de scripts

El equipo tiene un espacio limitado para guardar el script. El máximo son 15.000 caracteres.

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"**



2025-12-22

## 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.

Además de la lectura de este manual, recomendamos la lectura de scripts de ejemplo. Puede descargar ejemplos desde este link [www.exemys.com/EjemplosDeScriptEnGRD](http://www.exemys.com/EjemplosDeScriptEnGRD)

Todos estos ejemplos también funcionan en los cLAN, excepto lo que usan SMS.

## 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.

Los equipos GRD-XF-2G y cLAN V1.x funcionan con versión 1

Los equipos GRD-XF-3G y cLAN V2.x funcionan con versión 2 (hasta la versión 10.x) y la versión 3 desde 11.0 en adelante

Los GRD-MQ y cLAN MQ funcionan con versión 2 (hasta la versión 10.x) y la versión 3 desde 11.0 en adelante

La versión 3 es totalmente compatible con la 2 y se conmuta automáticamente a 3 en los equipos compatibles (V11.0 o superior).

Los programas escritos en versión 3 son idénticos a los de versión 2. Solo ocupan menos espacio en la memoria del equipo.

## 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 1** de script las variable numéricas son 21, de la **“a”** a la **“u”**. Las de texto son 5, de la **“v”** a la **“z”**, con una longitud máxima de 100 caracteres cada una.

En la **versión 2/3** 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.

Las variables numéricas, pueden “mapearse” para poder ser leídas de algún modo. En el GRD pueden vinculares de los canales de entradas y salidas.

**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 enviar el código al equipo, se reemplazará el alias por la variable correspondiente.

Como los alias se envían 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	Ahora
<pre>read_io 2,a,1; if a&gt;b { write_io 1,8,1; }; end;</pre>	<pre>#[a=SP1]; #[b=presion]; read_io 2,presion,1; if presion&gt;SP1 { write_io 1,8,1; }; end;</pre>

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 última 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 ejecutarán 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 ejecutarán las instrucciones que estén entre llaves debajo de este en caso contrario se ejecutarán 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

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 sección 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 sección fuentes/destinos para más 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 “*fuentes*” 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 sección fuentes/destinos para más 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 sección fuentes/destinos para más 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 **“sqr”**

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:

```
sqr resultado,inicio;
```

#### Ejemplo:

```
a=225;
sqr 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 ejecutarán 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 de los GRD-XF. En esta sección del manual se listan las distintas fuentes/destinos y luego de explica su uso por función.

## Listado de fuentes/destinos para GRD-XF

Se indica la versión desde la cual está disponible una fuente/destino en caso que se haya agregado a la versión inicial.

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función	2G	3G/4G
0	1 a 100	-	read_io	Canal de entrada digital (Ix)	Canales	Si	Si
1	1 a 100	-	read/write_io	Canal de salida digital (Ox)		Si	Si
2	1 a 100	-	read_io/write_io	Canal entrada analógica (ANx)		Si/5.1.3	Si
3	1 a 100	-	read/write_io	Canal de entrada de pulsos (PIx)		Si	Si
305	1 a 100	-	read/write_io	Memoria de canales		-	1.9
35	-	-	read/write_str	Traces del Script Programmer	Traces	5.2.0	Si
21	1 a 20	-	read/write_io	Memoria no volátil para números	Memoria No volatil	5.1.1	Si
121 a 125	-	-	read/write_str	Memoria no volátil para texto 1 a 5		5.2.2	Si
270	1 a 100	-	read_io	Lee el valor de la consulta Modbus MB-xxN	Modbus Master - Lectura directa consulta	-	1.8
271	1 a 100	-	read_io	Lee el estado de la consulta Modbus MB-xxN (0 Falla, 1 OK)		-	1.8
59	0	-	write_io	Cambia el multiplicador de read_io 23/36/72/68/69/73/74 de 1000 a otro valor (repito linea)	Modbus Master - Canales Float 32	5.2.5B3	Si
23	1 a 100	-	read_io	Canal P con query Modbus Float32. Devuelve parte entera del valor POR 1000. Ver write_io 59.		5.1.2	Si
36	1 a 100	-	read_io	Canal P con query Modbus Float32 con word invertidas. Devuelve parte entera del valor por 1000. Ver write_io 59.		5.2.0	Si
59	0	-	write_io	Cambia el multiplicador de read_io 23/36/72/68/69/73/74 de 1000 a otro valor (repito linea)	-	5.2.5B3	Si
9	0	Tabla	read_io	Estado de conexión GSM		Si	Si

280	0	0 / 1	read_io	Estado de roaming (0 no, 1 si)	GSM	-	1.8
10	0	Tabla	read_io	Estado de conexión GPRS	GPRS	Si	Si
22	0	0 / 1	read/write_io	Configuración del equipo la habilitación de conexión al MW (0 o 1)	MW	5.2.2	Si
11	0	Tabla	read_io	Estado de conexión MW		Si	Si
39	0	-	read_io	Cantidad de SMS en bandeja de salida	SMS	5.2.4	Si
4	-	-	read_str	Texto del SMS		Si	Si
5	-	-	read_str	Número telefónico del SMS		Si	Si
101 a 108	-	-	read_str	Nombres de la agenda telefonica 1 a 8		5.1.3	Si
111 a 118	-	-	read_str	Números de la agenda telefónica 1 a 8		5.1.3	Si
4	-	-	write_str	Texto del SMS		Si	Si
5	-	-	write_str	Número telefónico del SMS		Si	Si
17	1 a 100	0	write_io	De canal Dlx		Reportes, forzado	Si
18	1 a 100	0	write_io	De canal DOx	Si		Si
19	1 a 100	0	write_io	De canal ANx	Si		Si
20	1 a 100	0	write_io	De canal Plx	Si		Si
48	0	-	write_io	Deshabilitar el envío de registros históricos al MW (1 deshabilitado, 0 habilitado)	Historicos	5.2.2	Si
12	1 a 100	Valor	write_io	Por tiempo de canal ANx	Historicos, generacion	Si	Si
13	1 a 100	Valor	write_io	Por tiempo de canal Plx		Si	Si
14	1 a 100	Valor	write_io	De alarma máxima de canal ANx		Si	Si
15	1 a 100	Valor	write_io	De alarma mínima de canal ANx		Si	Si
16	1 a 100	Valor	write_io	De alarma normal de canal ANx		Si	Si
56	1 a 100	0/1	write_io	De cambio de canal Dlx		5.2.2	1.6
57	1 a 100	0/1	write_io	De cambio de canal DOx	5.2.2	1.6	
8	0	-	read_io	Cantidad de registros históricos no enviados almacenados en memoria	Historicos	Si	Si
60	-	-	write_io	Leer en memoria los datos de un registro específico de la memoria de registros (usar junto con read_io 61 a 65)		5.2.2	Si
61	0	-	read_io	'tipo de canal' de registro leído con write_io 60		5.2.2	Si
				'timestamp' de registro	Historicos		

62	0	-	read_io	Unestamp de registro leído con write_io 60	Historicos, acceso a memoria	5.2.2	Si
63	0	-	read_io	'tipo de historico' de registro leído con write_io 60		5.2.2	Si
64	0	-	read_io	'numero de canal' de registro leído con write_io 60		5.2.2	Si
65	0	-	read_io	'valor' de registro leído con write_io 60		5.2.2	Si
66	-	-	write_io	Borra los primeros N registros de la memoria de registros históricos		5.2.2	Si
7	0	-	read_io	Hora actual (segundos desde 1/1/2000)	Reloj	Si	Si
7	0	-	write_io	Puesta en hora actual (segundos desde 1/1/2000)		-	1.8
37	0	-	read_io	Cantidad de datos en el buffer de puerto serie (Los datos se borran del buffer con write_io 37)	Puerto serie	5.2.0	Si
37	0	-	write_io	Borrar los primeros N datos del buffer del puerto serie (usar junto con read_io 37 y read_io 38)		5.2.0	Si
38	0 a 199	0-255	read_io	Lee el valor binario de la posición indicada del buffer del puerto serie		5.2.0	Si
38	0	0-255	write_io	Enviar un byte al puerto serie		5.2.5B3	Si
6	-	-	read/write_str	Envío/Recepción del puerto Serie	Puerto serie, modo texto	Si	Si
32	0	-	write_io	Chequea si se enviaron datos al modem satelital (consume datos)	Satelite Iridium SBD	-	1.3
54	0	-	write_io	Inicia envío de registros históricos por modem satelital		5.2.2	Si
55	0	Tabla	read_io	Estado del envío por modem satelital.		5.2.2	Si
32	-	-	read_str	Recepcion de texto por SFIELD enviado por puerto serie transparente		-	1.3
29	-	-	write_str	Carga envío de string por modem satelital a puerto transparente (usar con write_io 31)		-	1.9
31	0	-	write_io	Dispara envío de string por modem satelital a puerto transparente (usar con write_str 29) MW 5.1.0	-	1.9	
50	-	-	write_str	Carga buffer de proceso (usar junto con read_str 51)	Parseo	5.2.0	Si
51	-	-	read_str	Lee buffer de proceso con agregado de inicio, fin y checksum de NMEA		5.2.0	Si

				(cargar antes el buffer de proceso con write_str 50)			
44	0	-	write_io	Iniciar conexión de cliente FTP	Cliente FTP	5.2.2	10.1
46	0	-	write_io	Cerrar archivo y terminar conexión de cliente FTP		5.2.2	10.1
47	0	Tabla	read_io	Estado de cliente FTP		5.2.2	10.1
40	-	-	write_str	Cargar URL para cliente FTP		5.2.2	10.1
41	-	-	write_str	Cargar usuario FTP		5.2.2	10.1
42	-	-	write_str	Cargar contraseña FTP		5.2.2	10.1
43	-	-	write_str	Cargar nombre de archivo para cliente FTP		5.2.2	10.1
45	-	-	write_str	Cargar línea de texto en archivo FTP y enviarla		5.2.2	10.1

2025-12-05

## Lectura/Escritura de canales de entrada/salida

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
0	1 a 100	-	read_io	Canal de entrada digital (Ix)	Canales
1	1 a 100	-	read/write_io	Canal de salida digital (Ox)	
2	1 a 100	-	read_io/write_io	Canal entrada analógica (ANx)	
3	1 a 100	-	read/write_io	Canal de entrada de pulsos (Plx)	
305	1 a 100	-	read/write_io	Memoria de canales	

Las fuentes 0 a 3 devuelven los valores de los distintos canales de entradas/salidas del equipo. El “**índice**” indica el numero de canal a leer.

**Ejemplo:** Leer el valor del canal de entradas analógicas 4 (AN4) y guardarlo en la variable c

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

El destino 0 permite activar las salidas del equipo. El “**índice**” indica el numero de canal a escribir.

**Ejemplo:** Apagar el canal de salida digital 3 (O3)

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

El destino 2 de **canales de entrada analógica** permiten escritura solo si están vinculados a consultas Modbus. El llamado a write\_io generará un comando es escritura Modbus

El destino 3 de **canales de pulsos** admiten los mismos valores que admita el equipo en esos canales (contadores físico o consulta Modbus de largo 2).

Las fuentes 23 y 36 permiten convertir registros Modbus Float 32 vinculados a canales de pulsos a valores enteros.

## Memoria de canales

Esta zona de memoria volatil con 100 posiciones puede funcionar como "Source" de todos los canales.

El valor de esta memoria puede leerse y escribirse con read\_io/write\_io 305

Esto permite liberar variables numéricas del script que antes se ocupaban para vincularlas a canales.

2025-12-05

## Envío/Recepción de mensajes al Script Programmer ("Traces")

Fuente/ Destino	Indice	Valor	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 (disponible desde Script Programmer V2.0) . 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 espacio 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 encuentra conectado al escribir en el destino 35 el texto simplemente se perderá pero no afectará al funcionamiento del programa.

2025-12-05

## Lectura/Escritura de memoria no volatil

### Para números

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
21	1 a 20	-	read/write_io	Memoria no volátil para números	Memoria No volatil

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

En equipos con firmware 5.1.1, no invocar a esta función permanentemente, solo cuando el valor cambia (se corre el riesgo de dañar la memoria).

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

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

### Para texto

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
121 a 125	-	-	read/write_str	Memoria no volátil para texto 1 a 5	Memoria No volatil

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

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

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

2025-12-05

## Lectura directa de consultas Modbus

<b>read_io</b>	<b>Descripción</b>	<b>Indice</b>
270	Lectura directa de consultas Modbus - Valor	1 a 100
271	Lectura directa de consultas Modbus - Estado	1 a 100

Las fuentes 270 y 271 permiten leer el valor de una consulta Modbus sin necesidad de previamente mapear esta consulta en un canal.

Esto permite no despediciar canales para usarlos como punto intermedio antes de procesar valores en el script.

La fuente 271 devuelve un 0 si hay falla en la comunicación Modbus o 1 si la comunicación no tiene errores

**Ejemplo:** Leer el valor de la consulta Modbus 4 y guardarlo en la variable c

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

2025-12-05

## Lectura de estados de GSM/GPRS en GRD

### Lectura de estados

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
9	0	Tabla	read_io	Estado de conexión GSM	GSM
280	0	0 / 1	read_io	Estado de roaming (0 no, 1 si)	
10	0	Tabla	read_io	Estado de conexión GPRS	GPRS

Desde las fuentes 9, 10 y 280 se puede conocer el estado de GSM, GPRS. Los posibles estados se listan a continuación

### Estados de GSM

#	Estado de GSM
0	OFF
1	ATTACHING
2	SIM NOT INSERTED
3	PIN REQUIERED
4	PIN ERROR
5	PIN OK
6	BLOQUED
7	<b>LOW SIGNAL</b>
8	ACCESS DENIED
9	<b>READY</b>

### Estados de GPRS

#	Estado de GPRS
0	OFF
1	WAIT GSM READY
2	ATTACHING
3	<b>CONNECTED</b>
4	ERROR
5	WAIT RECONNECTION

2025-12-05

## Lectura de estados de MW y configuración de conexión al MW

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
22	0	0 / 1	read/write_io	Configuración del equipo la habilitación de conexión al MW (0 o 1)	MW
11	0	Tabla	read_io	Estado de conexión MW	

## Lectura de estados

### Estados de Middleware

#	Estado de Middleware
0	OFF
1	WAIT GPRS READY (solo GRD)
2	CONNECTING
3	CONNECTION REFUSED
4	CONNECTION FAILED
5	HOST UNREACHABLE
6	HOST CLOSED CONNECTION (solo GRD)
7	<b>CONNECTED</b>
8	ERROR
9	WAIT RECONNECTION
10	DNS FAILURE
11	LOGGING IN

## Configuración de conexión con MW

La fuente/destino 22 permite configurar si el equipo se conecta o no al MW.

Tenga en cuenta que esta configuración se guarda en la memoria no volátil del equipo, y que una vez desconectado del MW no podrá acceder mas en forma remota.

## Deshabilitar el envío de registros históricos al MW

read_io / write_io	Descripción	Indice
48	Deshabilitar el envío de registros históricos al MW (1 deshabilitado, 0 habilitado)	0

La fuente/destino 48 permite deshabilitar el envío de registros históricos al MW.

Tenga en cuenta que esta configuración se guarda en la memoria no volátil del equipo, y que si desea

que el equipo vuelta a mandar históricos debe volver a habilitar el envío.

Esta función es util cuando se desea enviar registros históricos por algún medio alternativo.

**Ejemplo:** Deshabilitar el envío de registro históricos al MW.

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

*2025-12-05*

## Envio/Recepción de SMS. Agenda telefónica

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
39	0	-	read_io	Cantidad de SMS en bandeja de salida	SMS
4	-	-	read_str	Texto del SMS	
5	-	-	read_str	Número telefónico del SMS	
101 a 108	-	-	read_str	Nombres de la agenda telefonica 1 a 8	
111 a 118	-	-	read_str	Números de la agenda telefónica 1 a 8	
4	-	-	write_str	Texto del SMS	
5	-	-	write_str	Número telefónico del SMS	

## Recepción

Con las fuentes 4 y 5 se pueden recibir mensajes de texto. Para saber si llegó un mensaje se debe leer el texto del mensaje constantemente hasta que el largo del sea diferente de 0.

**Ejemplo:** Verificar si llega un mensaje de texto

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

## Envío

Los destinos 4 y 5 se pueden usar para enviar mensajes de texto. El mensaje se envia al escribir el texto en el destino 4. Previamente debe cargar el numero de destino en el destino 5.

Si se acaba de recibir un mensaje y lo quiere responder, puede escribir directamente la respuesta en el destino 4 sin escribir antes el número telefónico.

**Ejemplo:** Enviar un mensaje de texto

```
write_str 5,'1166041241'; #Carga el número telefónico destino;
write_str 4,'Hola'; #Carga el texto del SMS y lo envia;
```

Permite saber cuantos SMS están esperando para ser enviados.

## Agenda telefónica

Las fuentes 101 a 108 permiten leer los nombres de la agenda telefónica del GRD

Las fuentes 111 a 118 permiten leer los números telefónicos de la agenda telefónica del GRD

Estas fuentes resultan prácticas si se quiere cambiar el destinatario de un mensaje enviado desde el script sin tener que editar el programa.

**Ejemplo:** Leer el número telefónico en la posición 5 de la agenda del GRD

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

**2025-12-05**

## Forzado de reportes

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
17	1 a 100	0	write_io	De canal Dlx	Reportes, forzado
18	1 a 100	0	write_io	De canal DOx	
19	1 a 100	0	write_io	De canal ANx	
20	1 a 100	0	write_io	De canal Plx	

Los destinos 17 a 20 permiten forzar el envío de **reportes**, además de los que genera el equipo por si mismo. El valor del reporte es el que tenga el canal en ese momento. Se ignora el campo **valor**.

Se recomienda usar esta función con cuidado para no generar tráfico GPRS de manera permanente (solo GRD)

**Ejemplo:** Generar un reporte del canal AN3 cada 10 segundos con su valor actual

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

2025-12-05

## Historicos - Generación

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
12	1 a 100	Valor	write_io	Por tiempo de canal ANx	Historicos generacion
13	1 a 100	Valor	write_io	Por tiempo de canal Plx	
14	1 a 100	Valor	write_io	De alarma máxima de canal ANx	
15	1 a 100	Valor	write_io	De alarma mínima de canal ANx	
16	1 a 100	Valor	write_io	De alarma normal de canal ANx	
56	1 a 100	0/1	write_io	De cambio de canal Dlx	
57	1 a 100	0/1	write_io	De cambio de canal DOx	

Estos destinos permiten generar registros **históricos** desde el script, además de los que genera el equipo por si mismo. El valor del histórico se debe indicar en el campo *valor*.

Se recomienda usar esta función con cuidado para no generar registros históricos permanentemente.

**Ejemplo:** Generar un registro histórico por tiempo del canal AN2 cada 10 segundos con el valor 457

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

2025-12-05

## Histórico - Acceso a memoria de registros históricos

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
8	0	-	read_io	Cantidad de registros históricos no enviados almacenados en memoria	Historicos, acceso a memoria
60	-	-	write_io	Leer en memoria los datos de un registro específico de la memoria de registros (usar junto con read_io 61 a 65)	
61	0	-	read_io	'tipo de canal' de registro leído con write_io 60	
62	0	-	read_io	'timestamp' de registro leído con write_io 60	
63	0	-	read_io	'tipo de historico' de registro leído con write_io 60	
64	0	-	read_io	'numero de canal' de registro leído con write_io 60	
65	0	-	read_io	'valor' de registro leído con write_io 60	
66	-	-	write_io	Borra los primeros N registros de la memoria de registros históricos	

Las fuentes/destinos 60 a 66 permiten leer la memoria de registros históricos del equipo. Estas funciones se utilizan para poder enviar el contenido de esta memoria por un medio que no sea el envío al Middleware, por ejemplo, por FTP.

Antes de intentar leer un registro puede ver cuando registros hay en memoria usando *read\_io 8*. Luego puede invocar a *write\_io 60* para leer un registro particular y *read\_io 61 a 65* para obtener los valores de los campos del registro leído. Finalmente puede usar *write\_io 66* para borra el/los registros leídos.

**Ejemplo:** Leer los registro históricos de la memoria del equipo y enviarlos a la consola de "Traces" del Script Programmer

```
read_io 8,a,0;
if a>0
{
    write_io 60,0,0; #Lee el primer registro de la memoria (el 2do 0
indica la primera posicion);
    read_io 61,b,0; #Carga en b el tipo de canal;
    read_io 62,c,0; #Carga en c el timestamp;
    read_io 63,d,0; #Carga en d el tipo de historico;
    read_io 64,e,0; #Carga en e el numero de canal;
    read_io 65,f,0; #Carga en f el valor;
    w=b,'-',b,'-',c,'-',d,'-',e,'-',f,$13,$10;
    write_str 35,w; #Envia el texto del registro a la consola;
    write_io 66,0,1; #Borra el registro enviado de la memoria del
equipo;
};
```

Para un ejemplo mas completo del uso de esta funciones vea el ejemplo de uso de FTP.



*Si esta usando read\_io 8 / write\_io 66 para mantener la memoria con registros nuevos use la cantidad 90000 en vez de 100000.*



## Historicos - Deshabilitar el envio al MW

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
48	0	-	write_io	Deshabilitar el envio de registros históricos al MW (1 deshabilitado, 0 habilitado)	Historicos

La fuente/destino 48 permite deshabilitar el envío de registros históricos al MW.

Tenga en cuenta que esta configuración se guarda en la memoria no volatil del equipo, y que si desea que el equipo vuelva a mandar históricos debe volver a habilitar el envío.

Esta función es util cuando se desea enviar registros históricos por algún medio alternativo.

**Ejemplo:** Deshabilitar el envío de registro históricos al MW.

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

2025-12-05

## Lectura de reloj de tiempo real

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
7	0	-	read_io/write_io	Hora actual (segundos desde 1/1/2000)	Reloj

La fuente 7 permite leer la fecha/hora actual del equipo. El número puede ser convertido a texto usando las funciones de conversión.

**Ejemplo:** Obtener el mes actual en la variable g

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

En reciente versiones de firmware también se puede configurar la hora desde el script con `write_io 7`

```
2025-12-05
```

## Acceso al puerto serie en modo texto

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
6	-	-	read/write_str	Envio/Recepción del puerto Serie	Puerto serie, modo texto

La fuente/destino 6 permite recibir y enviar texto desde y hacia el puerto serie del equipo. Para saber si llegó texto al puerto serie se debe leer la fuente 6 constantemente hasta que el largo del sea diferente de 0.

Para enviar un texto simplemente escriba sobre el destino 6.

Para que funcione correctamente debe configurar el puerto serie en modo "Script"

Si quiere enviar caracteres binarios puede utilizar el operador \$

**Ejemplo:** Hacer "eco" del texto recibido por el puerto serie.

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

## Acceso al puerto serie en modo binario

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
37	0	-	read_io	Cantidad de datos en el buffer de puerto serie (Los datos se borran del buffer con write_io 37)	Puerto serie
37	0	-	write_io	Borrar los primeros N datos del buffer del puerto serie (usar junto con read_io 37 y read_io 38)	
38	0 a 199	0-255	read_io	Lee el valor binario de la posición indicada del buffer del puerto serie	
38	0	0-255	write_io	Enviar un byte al puerto serie	

La fuente/destino 37, mas la fuente 38 permiten interpretar datos binario binarios recibos en el puerto serie.

Para que funcione correctamente debe configurar el puerto serie en modo "Script"

Si quiere enviar datos 'binarios' puede usar el destino puede usar write\_str 6 junto con el operador \$

**Ejemplo:** Esperar a recibir mas de 2 bytes. Luego verificar si el tercer byte recibido es el binario 126. Finalmente borrar 3 bytes del buffer

```
read_io 37,a,0;
if a>2 {
    read_io 38,b,3;
    if b=126 {
        #El 3er byte recibido es el binario 126;
    };
    write_io 37,0,3;
};
```

## Segundo puerto serie en modo script

Desde la versión de firmware 11.3 ambos puertos series pueden funcionar en modo script. Las fuentes destino del puerto serie configurado en modo "Script Sec" son las mismas que el puerto principal pero sumando 1000.

Por ejemplo, para enviar datos en modo texto se usará `write_str 1006`

*2026-01-06*

## Modem satelital Iridium SBD

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
32	0	-	write_io	Chequea si se enviaron datos al modem satelital (consume datos)	Satelite Iridium SBD
54	0	-	write_io	Inicia envio de registros históricos por modem satelital	
55	0	Tabla	read_io	Estado del envío por modem satelital.	
32	-	-	read_str	Recepcion de texto por SFIELD enviado por puerto serie transparente	
29	-	-	write_str	Carga envio de string por modem satelital a puerto transparente (usar con write_io 31)	
31	0	-	write_io	Dispara envio de string por modem satelital a puerto transparente (usar con write_str 29) MW 5.1.0	

## Estado de envío

#	Estado del envío
-7	No envía. Conectado al MW
-6	Error enviando registros
-5	Inicializando modem
-4	Error en la configuración del puerto serie (Modo satelite y a 19200 bps)
-3	No hay registros para enviar
-2	Error al leer la memoria de registros
-1	Enviando registros
0	Listo para enviar
> 0	Cantidad de registros enviados

Si tiene conectado un modem SBD Iridium (EDGE/ITAS) al puerto serie del GRD/cLAN, puede enviar los registros históricos en su memoria al MW usando la red satelital de Iridium. En el manual del equipo puede ver mas detalles de la solución.

Se recomienda usar con discrecionalidad y conocer el costo del envio de datos por modem satelital.

El destino 54 permite iniciar el envio de los registros que el equipo tenga en memoria. Para iniciar el envio el modem debe estar en estado "Listo para enviar". El GRD/cLAN enviará todos los registros que pueda en un solo mensaje satelital.

Desde la versión 1.3 de GRD-XF-3G/4G y 2.2 de cLAN tambien se pueden recibir datos con el modem satelital. Los datos no llegan espontaneamente, el GRD/cLAN deb chequear si llegaron datos nuevos. Esto se realiza automaticamente cada vez que se envian datos. También se puede iniciar un chequeo de recepción de datos usando el write\_io 32. Tenga en cuenta que cada vez que se chequee, Iridium

genera un cargo. Para el envío de datos desde el MW hacia el GRD/cLAN se debe contratar y configurar en el MW el servicio de Iridium llamado "Static IP addresses for Mobile Terminated SBD"

Usando `read_str 32` se puede leer el texto enviado usando el modem satelital. Este texto se introduce al MW usando la conexión de puerto serie transparente.

También se pueden enviar strings a la conexión de puerto serie transparente del MW cargando el string con `write_str 29` y disparando el envío con `write_io 31`

Cada vez que se envían (`write_io 54`) o chequean datos (`write_io 32`) también se pueden recibir comandos desde el MW generados desde la tabla de escrituras en la base de datos. De esta manera se pueden modificar canales de salidas digitales, analógicos de consultas Modbus o canales vinculados a variables del script de manera remota.

Como ejemplo por favor mire el archivo *satelite.sce* que puede descargar junto a los otros ejemplos de uso de script [www.exemys.com/EjemplosDeScriptEnGRD](http://www.exemys.com/EjemplosDeScriptEnGRD)

2025-12-05

## Cálculo de checksums y CRCs

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función	2G	3G/4G
50	-	-	write_str	Carga buffer de proceso (usar junto con read_str 51)	Parseo	5.2.0	Si

El destino 50 permite cargar un string en el buffer de proceso para luego aplicar algún proceso al texto cargado

## Proceso de protocolo NMEA

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función	2G	3G/4G
51	-	-	read_str	Lee buffer de proceso con agregado de inicio, fin y checksum de NMEA (cargar antes el buffer de proceso con write_str 50)	Parseo	5.2.0	Si

La fuente 51 guardara en la variable la trama NMEA previamente cargada en el buffer de proceso con *write\_str 50*.

Esta fuente agrega a la trama original el inicio, el fin y el checksum de NMEA. Esto permite simular un "talker" NMEA con el equipo

**Ejemplo:** Enviar la sentencia NMEA *GPMWV,145.8,R,87.2,K,A* por el puerto serie del equipo, luego de agregarle el caracter de inicio, el de fin y el checksum

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

2025-12-05

## Cliente FTP

Fuente/ Destino	Indice	Valor	R/W	Descripción	Función
44	0	-	write_io	Iniciar conexión de cliente	Cliente FTP
46	0	-	write_io	Cerrar archivo y terminar conexión de cliente	
47	0	Tabla	read_io	Estado de cliente	
40	-	-	write_str	Cargar URL para cliente	
41	-	-	write_str	Cargar usuario	
42	-	-	write_str	Cargar contraseña	
43	-	-	write_str	Cargar nombre de archivo para cliente	
45	-	-	write_str	Cargar línea de texto en archivo y enviarla	

## Estados (read\_io 47)

#	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

Las fuentes/destinos listados en estas tablas permiten implementar un cliente FTP para la subida de archivos de texto a un servidor. Normalmente esta función será utilizada para enviar registros históricos de la memoria del equipo y en conjunto con las fuentes/destinos que permiten acceder a la memoria de registros históricos.

Para poder enviar datos por FTP el GRD debe estar registrado en la red GPRS (solo GRD).

Como ejemplo por favor mire el archivo *ftp.sce* que puede descargar junto a los otros ejemplos de uso de script [www.exemys.com/EjemplosDeScriptEnGRD](http://www.exemys.com/EjemplosDeScriptEnGRD)

2025-12-05