eWON
MACHINES CAN TALK

Reference Guide

RG 006 / Rev. 1.3

Programming Reference Guide

This document explains all there is to know about Basic Scripting combined with an eWON

support.ewon.biz

Table of Contents

1. BASIC language definition

1.1. Introduction

The program of the eWON is based on syntax close to the BASIC, with many specific extensions.

This document is an evolution of the RG-002-0-EN-(Programming Reference Guide) but can only be applied on Flexy devices running with a firmware strictly higher than v8.1s4.

All other eWONs (eWON CD, eWON Flexy with firmware <= v8.1s4) need to refer to the RG-002-0-EN-(Programming Reference Guide).

BASIC Scripting is possible on the eWON thanks to the Basic IDE that can be found via the Configuration > Basic IDE link.

1.2. Program flow

It is very important to understand how the eWON executes its program!

There's a difference between the storing and the execution of the program within the eWON: the eWON has a program task that extracts BASIC requests from a queue and executes the requests.

A request can be:

- A single command
  example: MyVar=1
- A branch to a label
  example: goto MyLabel
- A list of commands (program block)

In the first case, the command is executed then the BASIC task is ready again for the next request.

In the second case, the BASIC task goes to label *MyLabel* and the program executes until the END command is encountered or until an error occurs.

Suppose the eWON has no program, and you create:

- An Init Section:

```
CLS
myVar = 0
```

- A Cyclic Section:

```
FOR V% = 0 TO 10
    myVar = myVar + 1
NEXT V%
PRINT myVar
```

- A custom Section called *myNewSection*:

```
myNewSection:
myVar = 0
PRINT "myVar is reset"
```

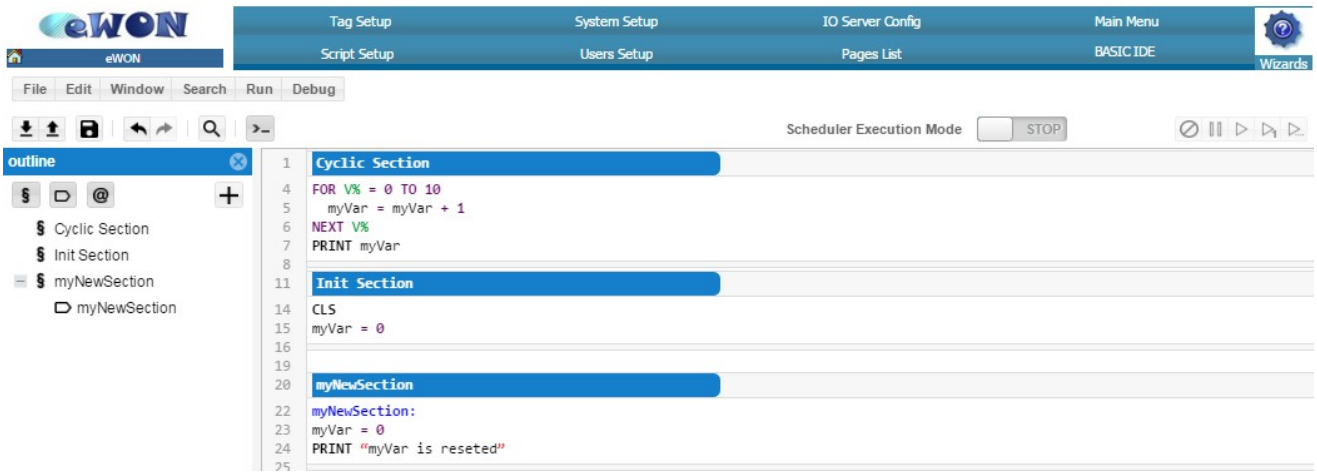

Illustration 1: BASIC code of the introduction

If you download the corresponding program.bas file using an FTP client, you will obtain the following program:

```
rem --- eWON start section: MY NEW SECTION
rem --- eWON user (start)
MyNewSection:
    MyVar = 0
    PRINT "MyVar is Reset"
rem --- eWON user (end)
End
rem --- eWON end section: MY NEW SECTION
rem --- eWON start section: Cyclic Section
ewon_cyclic_section:
rem --- eWON user (start)
```

```
FOR V%=0 to 10
    MyVar=MyVar+1
NEXT V%
PRINT MyVar
rem --- eWON user (end)
End
rem --- eWON end section: Cyclic Section
rem --- eWON start section: Init Section
rem --- eWON user (start)
ewon_init_section:
    CLS
    MyVar = 0
rem --- eWON user (end)
end
rem --- eWON end section: Init Section
```

As you can see, the code you have entered is present, but the eWON has added some remarks and labels in order to allow edition and to provide program flow control.

For each section in the editor, the eWON has added an END statement at the end to prevent the program from continuing to the next section. The example also shows that any label is global to the whole program and should not be duplicated.

We can also see here that there is not correlation between the section name and the label used in that section.

The section name is only a way to organize program listing during edition in the eWON.It can contain spaces while the program labels can't.

When the program starts (click RUN from the web site for example), the eWON posts 2 commands in the Queue:

Queue pos	Content	Type
...		
3		
2	goto ewon_cyclic_section	CYCLIC_SECTION
1	goto ewon_init_section	INIT_SECTION

Table 1: BASIC Queue - 1

The eWON BASIC task will read the request in the queue that has the lowest index and will execute it until an END is found or until an error occurs.

The first command is "GOTO ewon_init_section". The following lines will be executed:

```
ewon_init_section:
rem --- eWON user (start)
CLS
myVar = 0
rem --- eWON user (end)
END
```

The END command on the last line will end the program and the BASIC task will check in the queue for a new request:

Queue pos	Content	Type
...		
3		
2		
1	goto ewon_cyclic_section	CYCLIC_SECTION

Table 2: BASIC Queue - 2

The first available command is "*goto ewon_cyclic_section*", it will also be executed until the END is found. When this END is executed the BASIC task will detect that the section it has just executed was a CYCLIC_SECTION and it will post a new "*goto ewon_cyclic_section*" request in the queue.

This is how the program is continuously executed forever while the BASIC is in RUN mode.

There are a number of actions that can be programmed to occur upon event, like ONTIMER:

```
TSET 1,10
ONTIMER 1,"goto MyLabel"
```

Suppose you add the above lines in the INIT SECTION, it will start timer 1 with an interval of 10 seconds and program a "*goto MyLabel*" request when timer 1 ellapses.

What actually happens when the ONTIMER occurs is that the eWON posts the "*goto MyLabel*" request in the BASIC queue.

Queue pos	Content	Type
...		
3		
2	goto MyLabel	
1	goto ewon_cyclic_section	CYCLIC_SECTION

Table 3: BASIC Queue - 3

When the CYCLIC SECTION will be finished, the timer request will be extracted from the queue and then executed. If the CYCLIC SECTION takes a long time to execute, then the time can elapse more than once during its execution, this could lead to more timer action to be posted in the queue:

Queue pos	Content	Type
...		
5		
4	goto MyLabel	
3	goto MyLabel	
2	goto MyLabel	
1	goto ewon_cyclic_section	CYCLIC_SECTION

Table 4: BASIC Queue - 4

The BASIC queue can hold more than 100 requests, but if TIMER goes too fast or if other events like ONCHANGE are used the queue can overflow, in that case an error is logged in the events file and requests are dropped.

You can also see that the ONTIMER request is not executed with the exact precision of a timer, depending on the current load of the BASIC when the timer elapses.

When an ASP block has to be executed for the delivery of a WEB page to a client, the ASP block is also put in the queue

As an example, if ASP block contains the following lines:

Queue pos	Content	Type
...		
3	FromWebVar = Var1! PRINT #0;TIME$	
2	goto MyLabel	

Queue pos	Content	Type
1	goto ewon_cyclic_section	CYCLIC_SECTION

Table 5: BASIC Queue - 5

If a request in the queue contains more than 1 BASIC line, what actually happens is the following:

- The block is appended to the end of the program as a temporary section:

```
ewon_one_shot_section:
fromWebVar = Var1
PRINT #0;TIME$
END
```

  The temporary label is called (goto ewon_one_shot_section).
- When the execution is done, the temporary section is deleted from the program.

As a consequence, we have the following:

- Any global variable, or label can be used in REMOTE.BAS or ASP blocks; you can call subroutines in your ASP blocks and share common variables with the program.
- If a section is being executed when the ASP section is posted, all the requests in the queue must first be executed. This may have an impact on the responsiveness of the WEB site when ASP is used.
- When using ASP you would better group your blocks to avoid posting too many different requests in the queue. By doing this you will reduce queue extraction and BASIC context switches.
- If a big amount of ASP request (or long ASP request) is posted to the BASIC by the WEB server, it may slow down normal execution of the BASIC.
- Sections are never interrupted by other sections: this is always true, when a program sequence is written, it will never be broken by another execution (of timer or WEB request or anything else).

1.2.1. Character String

A character string can contain any set of characters. When creating an alphanumeric string with a quoted string the ‘ or " delimiter can be used

```
"abcd"
’abdc’
"abc‘def’ ghi "
```

The examples above show 3 valid quotes strings.

A character string can be stored either in an alphanumeric type variable, or in an alphanumeric variable array.

1.2.2. Command

A command is an instruction that has 0 or several comma (,) separated parameters.

There are 2 exceptions to the comma separator: PRINT and PUT.

```
GOTO Label
PRINT
CLS
SETSYS TAG, "name","Power"
SETSYS TAG,"SAVE"
```

1.2.3. Integer

An integer is a number between -2147483648 and +2147483647. This number can be stored in an integer variable. When a parameter of integer type is specified for a function or a command and the variable past is of real type, the eWON automatically converts the real value to an integer value.

When the expected value is of integer type and the past value is a character string, the eWON generates an error.

1.2.4. Real

A Real number is a number in floating point representation of which value is between -3.4028236 10E38 and +3.4028234 10E38. Value of this type can be stored in a variable of real type or in an array of reals.

A Real number has approximately 7 significant digits. This means that conversion of a number with more than 7 significant digits to real will lead to a lost of precision.

When a function expects a real number and an integer is passed, the eWON automatically converts the integer into a real value. If the function waits for a real and a character string is passed, the eWON generates an error.

eWON uses IEEE 754 single precision representation (32 bits).

So the fraction is coded on 23 bits, which represents about 7 significants digits. But in the ViewIO page the values are only displayed with 6 digits. If you use the Tag in Basic Scripting you will find the 7 significant digits.

1.2.5. Alphanumeric character

An alphanumeric character is one of the ASCII characters. Each ASCII character has a numerical representation between 0 and 255.

The ASCII basic function returns the ASCII code of a character, and the CHR$ function converts the ASCII code to a string of 1 character.

1.2.6. Function

A function is a BASIC command having 0 or several parameters and returning a result that can be of integer, real or string type.

```
ASCII "HOP"
GETSYS TAG,"NAME"
PI
```

1.2.6.1. Function declaration

To declare a function, you need two keywords:

- **FUNCTION**
  It is used to start the function definition and is followed on he same line by the function name
- **ENDFN**
  It is used to end the function definition

- **Example**

```
FUNCTION my_function // function definition begins
    PRINT "my_string"
ENDFN
```

1.2.6.2. Function return value

You specify the function return value using the following function name convention:

- If your function returns an integer: Function my_function%
- If your function returns a string: Function my_function$
- If your function returns a float: Function my_function

To specify the return value of a function, an implicit variable is created automatically based on your function name. When your function exits, the return value is the last value of this variable.

```
FUNCTION my_function
    $my_function = 1
    $my_function = $my_function + 1
    PRINT "my_string"
```

```
ENDFN
```

This example prints “my_string” in the console but the return value will be 2.

1.2.6.3. Keyword “return” inside functions

The keyword “return” can be used at any place inside a function in order to end it.

```
FUNCTION my_function
    IF (global_var%=1) THEN
          $my_function = 1.0
          RETURN
    ENDIF
    $my_function = 0.0
ENDFN
```

The current value of the “RETURN” ($FunctionName) will be returned just as if we reached the ENDFN.

1.2.6.4. Function parameters

Parameters can be defined and applied to a function. These parameters need to be typed (same way as functions).

Properties of these parameters:

- Parameters are put between parenthesis and separated by a coma.
- Parameters are, by default, passed by value.
- Parameters type is deduced by the naming convention:
  - '$' at the end for string
  - '%' at the end for integer
  - nothing at the end for float
- Parameters are local variables in the function scope.
- These function parameters don't exist outside the function.
- To clarify this distinction with standard variables: every parameter variable begins with '$' in the declaration and inside the function. This allows you to manipulate global and local variable with the same name without messing up.

  - **Example**

```
FUNCTION my_function($param1, $param2%, $param3$)
    $my_function = $param2% + $param1 + 1
ENDFN
PRINT @my_function(3, 3, "3")
```

1.2.6.5. Function call

To call a function, the '@' character precedes the function name and the parameters values are put between parenthesis. If there is no parameters, parenthesis can be omitted.

```
FUNCTION my_function($param1)
    PRINT "call of [my_function] with param [";$param1;"]"
ENDFN

FUNCTION my_function2()
    PRINT "my_function2()"
ENDFN

FUNCTION my_function3
    PRINT "my_function3()"
ENDFN

@my_function(3)
@my_function2 // call of a function without parenthesis nor parameters
@my_function3() // call of a function without parameters
```

Pay attention to float and integer parameters, if a float is given as an integer parameter (or the opposite), an implicit cast will occur.

```
FUNCTION my_function($param1%)
    PRINT "call of [my_function] with param [";$param1;"]"
ENDFN

@my_function(3) // OK
@my_function(3.4) // KO
```

1.2.6.6. Passing arguments by reference

By default the parameters are passed by value.

This means that side effects can't be executed. But sometimes, side effects are useful (i.e: a function that returns 3 values).

If the parameter is preceded by '@', they will be passed by reference. It can then be used

as a normal parameter inside the function.

The only difference compared to a normal parameter (passed by value) is that all changes made inside the function will be visible outside this function.

- **Example**

```
FUNCTION my_function(@$param1,@$param2,@$param3$)
    $param1 = $param1 * 2
    $param2 = $param2 * 2
    $param3$ = "my_function_string"
ENDFN
v1 = 1.5
v2% = 2
v3$ = "my_string"
@my_function(v1, v2%, v3$)
PRINT v1 // Prints 3.00
PRINT v2% // Prints 4
PRINT v3$ // Prints my_function_string
```

1.2.6.7. Recursive function call

A function can be called inside an already existing function.

- **Example**

```
FUNCTION exp($x, $n)
    IF ($n = 1) then
          $exp = $x
    ELSE
          IF ($n mod 2 = 0) THEN
                $exp = @exp($x * $x, $n / 2)
          ELSE
                $exp = $x * @exp($x * $x, ($n - 1) / 2)
          ENDIF
    ENDIF
ENDFN

PRINT @exp(3, 3)
```

1.2.7. Label

To use the GOTO and GOSUB commands, you need to define LABELS.

A label is a name beginning a line and ended by a colon ':'. The label must not have any space character.

The GOTO/GOSUB instruction use the Label name **without** the colon as parameter.

```
MyLabel
GOTO "MyLabel"
```

1.2.7.1. Local label

Sometimes, it's useful to have labels only inside a function to ease the flow control, but without polluting the name spaces of the program.

To solve this, local labels can be defined in functions.

- **Syntax**

$S1:

- *S1 is the label name*

A '*GOTO*' can now be used inside the function to move to this label. Outside the function, this label doesn't exist.

- **Example**

```
FUNCTION test_label():
    $a% = 1
    GOTO $exit
    $a% = 2
    $exit:
    $test_label = $a%
ENDFN
PRINT @test_label() // Prints 1
```

1.2.8. Operators priority

When these operations appear in expressions, they have the following priority:

- Bracket terms: maximum priority
- All functions except NOT and – (inversion)
- Inversion of sign -
- *, /, ^, MOD (modulo function)
- +, -
- =, >, <, <=, >=, <>
- NOT, BNOT • AND, OR, XOR: minimum priority

The expressions are ordered by decreasing order of priority.

- Note -

^ operator is the Power operator
*i.e.: 2^4 = 2*2*2*2*

- **See also**

"NOT" on page 69, "BNOT" on page 27, "AND" on page 25, "OR" on page 84, "XOR" on page 110, "MOD" on page 68

1.2.9. Type of Variables

Variables typed as Integer or as String can be defined with a long name. Long name variable are also applicable on Array (ex. : DIM arrayOfString(25,80)

Variable names are case insensitive (myint% and MyInt% are the same variable).

1.2.9.1. Integer variable

- **Syntax**

abcdef%

- The name of the variable is followed by the "%" letter to indicate that it is an integer variable. An integer variable can contain a number of type integer.

- **Example**

```
my_variable% = 1 // unlimited number of variables
DIM arrayOfString(25,80) // unlimited number of array of strings DIM A(25,80)
DIM arrayOfFloat(25,80) // unlimited number of array of floats
```

1.2.9.2. Real variable

- **Syntax**

abcdef

- abcdef is the name of the variable that can contain up to 200 characters.

Variable names are not case sensitive: AbCdEf and ABCDEF represent the same variable.

The variable name can only contain the letters "a" to "z"; all other characters are not allowed.

The variable name can contain alphabetical characters, numbers and "_" underscore, but name must begin with an alphabetical character.

A real variable can contain a real number.

Others characters are not allowed.

```
MyVar = 12.3 (valid)
My Var = 12.3 (invalid)
My_Var = 12.3 (valid)
Var1 = 12.3 (valid)
1Var = 12.3 (invalid)
```

1.2.9.3. Alphanumeric string

- **Syntax**

abcdef$

The name of the variable is followed by the "$" letter to indicate that it is a string. A string can contain any number of characters. Its size is modified each time the content of the variable is modified.

It is possible to address parts of a string with the TO keyword:

A$(4 TO 10)	returns a string with chars 4 to 10
A$(4 TO)	returns a string with character 4 to end of string
A$(4 TO LEN(A$))	same result

1.2.9.4. Characters arrays

The number of dimensions is only limited by the memory size of the BASIC.

When the DIM command is called, the array is created and replaces any other DIM or variable existing with the same name. To erase an array you can either use the CLEAR command that erases all variables, or change the dimension of the array to 1 element with another call to DIM if you don't want to clear everything but need to release memory.

An array of which name is a$(E1,E2,E3) and an alphanumeric variable of which name is a$ can exist simultaneously. A characters array contains E1*E2*E3 *... characters.

- **Syntax [Command]**

DIM a$(E1 [, E2 [, E3 [,....]]])

a$ is the name of the variable array created, its name only contains one active character of "a" until "z". E1 is the number of characters for the first dimension. E2, E3, E4 are optional and are present if the array must have 2, 3, 4,...dimensions.

DIM A$(10,20)	
DIM Z(6)	

A$(4, 3 TO 3)	same kind of access with arrays

1.2.9.5. Real arrays

When the DIM command is called, the array is created and replaces any existing array with the same name. To erase an array you can either use the CLEAR command that erases all variables, or bring back the dimension of the array to 1 element if you don't want to clear everything but need to release memory.

In order to assign a value, type a(x, y, z)=value.

An array of which name is a(E1,E2,E3) and a real variable of which name is a CAN exist simultaneously. A real array contains E1*E2*E3 *... reals.

- **Syntax [Command]**

DIM a(E1 [, E2 [, E3 [,....]]])

a is the name of the array variable created, its name contains one character from "a" to "z".

E1 is the number of real for the first dimension. E2, E3, E4 are optional and are present if the array must have 2, 3, 4,… dimensions.

The number of dimensions is only limited by the BASIC memory size.

```
DIM d(5,5)
d(1,6)=6
```

1.2.9.6. Local Variables

Local variables are used to define variables visible only in the function scope.

The local variable needs to be preceding by the '$' character inside the function.

- **Example**

```
FUNCTION a()
    $b = 3 // local variable b
    $a = $b + 3
ENDFN

exec:
print @a() // here, @a() exists, but not $b.
```

1.2.10. TagName variable

- **Syntax**

TagName@

TagName is the name of the Tag. Adding the '@' after the Tag name allows direct access to the Tag value. This syntax can be used for reading or writing to the Tag.

- **Example**

```
Tag1@ = 25.3
Tag2@ = Tag1@
IF (Tag3@>20.0) THEN …
```

Only in some cases it is useful to use the GETIO or SETIO commands in order to build the Tag name in the program (to perform some repetitive operations or if a Tag name begins with a number, it cannot be accessed in Basic using the @ syntax, instead the GETIO, SETIO function must be used).

```
FOR i%=1 to 10
    A$ = "Tag"+STR$(i%)
    SETIO A$,i%
NEXT i%
```

1.2.11. Tag Access

All the Basic functions accessing Tags could reference the tag by its name, by its Index or by its ID.

Method	Parameter	Example	Example explanation
Tag name access	Tagname String	SETIO "TAG1",23.5	Set the value 23.5 in the Tag named TAG1
Index access	Negative Integer (or 0)	SETIO -2,23.5	Set the value 23.5 in the Tag at the INDEX 2 (the third entry in the var_lst.txt)
TagId access	Positive Integer (>0)	SETIO 2,23.5	Set the value 23.5 in the Tag with the ID=2

Table 6: Tag Access methods

If there are 6 Tags defined in the config, each Tag can be accessed by its index (-0 to -5) or by its ID (the first item of a Tag definition when reloading the config.txt file, the ID of a Tag is never reused during the live of a configuration until the eWON is formatted) or finally by its name.

1.2.12. Limitations of the BASIC

- The eWON BASIC script is limited by the memory reserved for it (128 k). Users have to share this memory space between the code and the data.

2. List of Keywords

The commands and functions used to program the eWON are listed below in alphabetical order.

The following commands or functions are available for any firmware version except specifically notified otherwise.

2.1. Syntax convention

In the following keyword usage description, the following convention is used to represent the parameters:

Parameter	Type
E1, E2	Integer
S1, S2	String
CA	Character (if string passed, first char is evaluated)

Table 7: BASIC keywords syntax convention

2.1.1. # (bit extraction operator)

- **Syntax [function]**

E1 # E2

- E1= integer word
- E2 = bit position (0 to 31)

- **Purpose**

The # function is used to extract a bit from an integer variable (and only from an integer).

- **Example**

```
i%=5 :Rem Binary 0101
a%=i%#0 :Rem a%=1
b%=i%#1 :Rem b%=0
c%=i%#2 :Rem c%=1
```

2.1.2. // (comment)

- **Syntax [command]**

// free text

- **Purpose**

This command enables the insertion of a line of comment in the program. The interpreter does not consider the line.

- **Example**

```
PRINT a%
// we can put whatever comment we want here
a%=2: REM Set a% to 2
```

- **See also**

"REM" on page 93

2.1.3. ABS

- **Syntax [function]**

ABS E1

- **Purpose**

The function returns the absolute value of E1. E1 can be a value or a Tag name. See also "Operators priority" on page 8. If the value is negative, you have to add use ().

- **Example**

```
ABS (-10.4)
```

This would return : 10.4

2.1.4. ALMACK

- **Syntax [function]**

ALMACK TagRef, S2

- TagRef is the Tag reference (TagName, ID or -Index) See Tag Access on page 10

- S2 is the UserName of the user that will acknowledge the alarm. If this field is the empty field "", then the "adm" login is assumed for acknowledgment.

- **Purpose**

The function acknowledges the alarm status of a given Tag. ALMACK returns error "Operation failed (28)" if the tag is not in alarm.

- **Example**

```
ALMACK "MyTag", "TheMighty"
```

2.1.5. ALSTAT

- **Syntax [function]**

ALSTAT TagRef

- TagRef is the Tag reference (TagName, ID or -Index) See Tag Access on page 10

- **Purpose**

Returns the S1 Tag alarm status. The returned values are:

Parameter	Type
0	No alarm
1	Pretrigger: no active alarm but physical signal active
2	In alarm
3	In alarm but acknowledged
4	Returns to normal but not acknowledged

Table 8: Values returned by the ALSTAT command

- **Example**

```
a% = ALSTAT "MyLittleTag"
```

2.1.6. AND

- **Syntax [Operator]**

E1 AND E2

- **Purpose**

Do a bit-wise AND between E1 and E2. Also have a look at the priority of the operators.

- **Example**

```
1 AND 2
```

Returns 0

```
2 AND 2
```

Returns 2

```
3 AND 1
```

Returns 1

```
MyFirstTag@ AND 3
```

Keeps first 2 bits.

- **See also**

"Operators priority" on page 17, "OR" on page 84, "XOR" on page 110

2.1.7. ASCII26

- **Syntax [function]**

ASCII CA

- **Purpose**

The function returns the ASCII code of the first character of the chain CA. If the chain is empty, the function returns 0.

- **Example**

```
a% = ASCII "HOP"
```

Returns the ASCII code of the character H

- **See also**

“CHR$” on page 28

2.1.8. BIN$

- **Syntax [function]**

BIN$ E1

- **Purpose**

The function returns a string of 32 characters that represents the binary value of E1. It does not work on negative values.

- **Example**

```
A$= BIN$ 5
```

REM A$ is worth " 00000000000000000000000000000101 " after this affectation

- **See also**

“HEX$” on page 56

2.1.9. BNOT

- **Syntax [function]**

BNOT E1

- **Purpose**

This function returns the "bitwise negation" or one's complement of the integer E1.

- **Example**

```
a%=5
b%=BNOT a%
```

```
print BIN$(b%)
```

Will display 11111111111111111111111111111010

- **See also**

"Operators priority" on page 17

2.1.10. CFGSAVE

- **Syntax [Command]**

CFGSAVE

- **Purpose**

Writes the eWON configuration to flash. This command is necessary after SETSYS command on SYS, TAG or USER records because using SETSYS will modify the configuration in memory. The modification will be effective as soon as the SETSYS XXX,"save" (where XXX stands for SYS, USER or TAG), but the config is not saved to the eWON flash file system.

- **See also**

"GETSYS, SETSYS" on page 47

2.1.11. CHR$

- **Syntax [Function]**

CHR$ E1

- **Purpose**

The function returns a character string with only one character corresponding to the ASCII code of E1. E1 must be contained in the 0..255 range.

- **Example**

```
A$= CHR$ 48
```

A$ is worth "0" after this affectation

```
B$=CHR$(getio(MyTag))
```

If MyTag=32, then B$ will hold one space.

- **See also**

"ASCII130" on page 26

2.1.12. CLEAR

- **Syntax [Command]**

CLEAR

- **Purpose**

Erases all variables from the eWON. All DIM are erased. This command cannot be canceled.

2.1.13. CLOSE

- **Syntax [Command]**

CLOSE I1

- I1 is the file number (1-8)

- **Purpose**

This command closes the file with file number I1. If the file is opened for write, it is actually written to the flash file system. The function can be called even if the file is not opened.

- **See also**

"EOF" on page 33, "GET" on page 41, "OPEN" on page 79, "PUT" on page 87

2.1.14. CLS

- **Syntax [Command]**

CLS

- **Purpose**

This command erases the virtual screen of the eWON, visible in the Script control page.

- **See also**

"PRINT - AT" on page 85

2.1.15. DAY

- **Syntax [Function]**

DAY E1 / S1

- E1 is a date in integer format (number of seconds since 1/1/1970)
- S1 is a date in String format ("18/09/2003 15:45:30")

- **Purpose**

This function returns an integer corresponding to the value of the day of the month (1--31) that matches a defined time variable. REM: Do not call the function with a float variable of value (or this would result to error "invalid parameter").

- **Example**

```
a$ = TIME$
a% = DAY a$
```

```
b% = getsys prg,"TIMESEC"
a% = DAY b%
```

- **See also**

“DOW” on page 31, “DOY” on page 32, “MONTH” on page 68, “WOY” on page 109

2.1.16. DEC

- **Syntax [Function]**

DEC S1

- S1 is the string to convert from HEX to DEC

- **Purpose**

This function returns an integer corresponding to the hexadecimal value of parameter. The string is not case sensitive (a023fc = A023FC). The string can be of any length.

- **Example**

```
A$= HEX$(1234)
I% = DEC(A$)
```

Now, I% = 1234

- **See also**

"HEX$" on page 56

2.1.17. DIM

- **Purpose**

The DIM function permits to create variables of array type. Two types of array are available: the characters arrays and the real arrays.

See also:

"Type of Variables" on page 18

2.1.18. DMSYNC

- **Syntax [Function]**

DMSYNC

- **Purpose**

The command has no parameter and will trigger a Data Management synchronisation. If the Data Management has been configured on the eWON, this command will send the historical data to the Data Management system.

2.1.19. DOW

- **Syntax [Function]**

DOW E1 / S1

- E1 is a date in integer format (number of seconds since 1/1/1970)
- S1 is a date in String format ("18/09/2003 15:45:30")

- **Purpose**

This function returns an integer corresponding to the value of the day of the week (0--6; Sunday = 0) that matches a defined time variable. REM: Do not call the function with a float variable of value (or this would result to error "invalid parameter").

- **Example**

```
a$ = TIME$
a% = DOW a$
```

```
b% = getsys prg,"TIMESEC"
a% = DOW b%
```

- **See also**

"DAY" on page 30, "DOY" on page 32, "MONTH" on page 68, "WOY" on page 109

2.1.20. DOY

- **Syntax [Function]**

DOY E1 / S1

- E1 is a date in integer format (number of seconds since 1/1/1970)
- S1 is a date in String format ("18/09/2003 15:45:30")

- **Purpose**

This function returns an integer corresponding to the value of the current day in the year (0-365) that matches a defined time variable. REM: Do not call the function with a float variable of value (or this would result to error "invalid parameter").

- **Example**

```
a$ = TIME$
a% = DOY a$
```

```
b% = getsys prg,"TIMESEC"
a% = DOY b%
```

- **See also**

"DAY" on page 30, "DOW" on page 31, "MONTH" on page 68, "WOY" on page 109

2.1.21. DYNDNS

- **Syntax**

DYNDNS

- **Purpose**

The command has no parameter and asks a NO-IP dynamic PPP IP address update to the Dynamic DNS server you have set in Publish IP address Configuration page. It will be used to synchronize a Dynamic DNS server such as No-IP with the eWON PPP IP address.

2.1.22. END

- **Syntax [Command]**

END

- **Purpose**

Indicates the end of the program. This command can also be used to stop the execution of a section. If the program is in RUN mode, this command will suspend the execution until another section is ready to run (ONCHANGE, CYCLIC etc.).

- **Example**

```
PRINT " START "
END
PRINT " SUB "
```

- **See also**

"HALT" on page 55

2.1.23. EOF

- **Syntax [function]**

EOF E1

- E1 is a number (1-8) corresponding to a /usr file or an ExportBlocDescriptor.

- **Purpose**

Returns 1 when end of file is reached. EOF always returns 1 for files opened for write. EOF works only with OPEN "file:..." or OPEN "exp:..." FileStream.

- **Example**

```
PRINT "open file"
    OPEN "file:/usr/myfile.txt" FOR TEXT INPUT AS 1

ReadNext:
IF EOF 1 THEN GOTO ReadDone
A$ = GET 1
PRINT A$
GOTO ReadNext

ReadDone:
PRINT "close file"
CLOSE 1
```

- **See also**

"CLOSE" on page 29, "GET" on page 41, "OPEN" on page 79, "PUT" on page 87

2.1.24. ERASE

- **Syntax [Command]**

ERASE Filename | Keyword

- **Purpose**

Erase the specified file in the /usr directory. That means this command will not work for a different directory than the "/usr" directory. Omitting "/usr/" before the filename will result to a syntax error. The file and directory names are case sensitive.

- **Example**

```
ERASE "/usr/myfile.shtm"
```

In order to erase some root files, some special keywords have been added.

Keyword	Description	Since Firmware
ERASE "#ircall"	To erase the ircall.bin file, then all historical logged data	5.7
ERASE "#events"	To erase the events.txt file, the diagnostics file.	5.7
ERASE "#hst_alm"	To erase the hst_alm.txt file,	5.7

Keyword	Description	Since Firmware
	the alarms historical file.	
ERASE "#usr"	To erase (and format) completely the "/usr" directory/partition	6.2
ERASE "#sys"	To erase (and format) completely the "/sys" directory/partition	6.2

Table 9: Special keywords for ERASE command

- **See also**

"RENAME" on page 93

2.1.25. FCNV

- **Syntax [function]**

FCNV S1,EType[,ESize,SFormat]

- S1 is the string to be converted.
- EType is the parameter determining the type of conversion.
- ESize is the size of the string to convert (can be shorter than the entire S1).
- SFormat is the format specifier for the conversion.

- **Purpose**

Converts a string to a number (float or integer). The return value can be an IEEE float, an Integer, a CRC16, a LRC. The type of conversion is determined by the EType parameter.

Etype value	Conversion type
1	convert string (MSB first) to Float
2	convert string (LSB first) to Float
5	compute the CRC16 on string and return an Integer
6	compute the LRC on string and return an Integer
10	convert string (MSB first) to Integer
11	convert string (LSB first) to Integer
20	convert string to a Float using a SFormat specifier
30	convert string to an Interger using a SFormat specifier

Etype value	Conversion type
40	convert time as string into time as Integer

Table 10: Etype of FCNV command

- **See also**

"SFMT" on page 98

2.1.25.1. Convert from an IEEE float representation

The IEEE float representation use four bytes (32 bits).

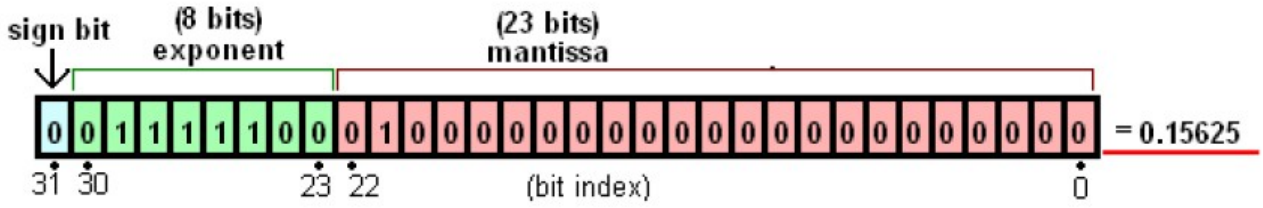

Illustration 2: Conversion to an IEEE float

The string could be LSB first:

```
A$ = SFMT FloatNum, 1
```

This will convert FloatNum to a string holding the IEEE representation with MSB first

```
A$(1) = MSB (Exponent+ Sign)
…
A$(4) = LSB (Mantissa LSB)
```

or MSB first:

```
A$ = SFMT FloatNum, 2
```

This will convert FloatNum to a string holding the IEEE representation with LSB first

```
A$(1) = LSB (Mantissa LSB)
```

```
...
A$(4) = MSB (Exponent+ Sign)
```

- **Example**

```
ieee = -63.456
A$ = SFMT ieee,1
rem a$(1)=194 a$(2)=125 a$(3)=210 as(4)=242

A$ = SFMT ieee,2
rem a$(1)=242 a$(2)=210 a$(3)=125 as(4)=194
```

2.1.25.2. Compute CRC16 of a string

Compute the Cyclic Redundancy Check (CRC) of the string.

CRC-16 uses the Polynomical 0x8005 ($x^{16} + x^{15} + x^2 + 1$) with an init value of 0xFFFF

- **Example**

```
A$="My string"
c% = FCNV A$,5
print c% : rem c% = 51608
```

2.1.25.3. Compute LRC of a string

Compute the Longitudinal Redundancy Check (LRC) of the string.

The LRC computation is the sum of all bytes modulo 256.

- **Example**

```
A$="My string"
c% = FCNV A$,6
print c% : rem c% = 125
```

2.1.25.4. Convert from an Integer representation

Convert a string containing several bytes (1 to 4) in an Integer value.

The integer representation could be LSB (Least Significant Byte) first or MSB (Most Significant Byte) first.

The ESize parameter is required!

It is the size of the string to convert (it can be 1, 2, 3 or 4).

```
FCNV A$, 10,4
```

This will convert A$(1 to 4) to an integer representation with MSB first

```
A$(1) = MSB
…
A$(4) = LSB
```

```
FCNV A$, 10,2
```

This will convert A$(1 to 2) to an integer representation with MSB first

```
A$(1) = MSB
…
A$(2) = LSB
```

```
FCNV A$, 11,4
```

This will convert A$(1 to 4) to an integer representation with LSB first

```
A$(1) = LSB
…
A$(4) = MSB
```

```
FCNV A$, 11,2
```

This will convert A$(1 to 2) to an integer representation with LSB first

```
A$(1) = LSB
…
A$(2) = MSB
```

- **Example**

```
A$=CHR$(1)+CHR$(4)+CHR$(2)+CHR$(0)
a%=FCNV A$,10,2
b%=FCNV A$,11,2
PRINT a% : rem a% = 260
PRINT b% : rem b% = 1025

c%=FCNV A$,10,3
PRINT c% : rem c% = 66562
c%=FCNV A$,10,4
PRINT c% : rem c% = 17039872
```

2.1.25.5. Convert string to a Float using a SFormat specifier

Convert a string with a float number (ex: A$="153.24") to a Float variable using a Format specifier.

- The **ESize parameter is required**.
  It is the size of the string to convert (use 0 to convert the whole string).
- The **SFormat parameter is required**.
  t is the format specifier string and must be "%f".

- **Example**

```
float_0 = FCNV "14.2115",20,0,"%f"
float_1 = FCNV "14.2115",20,4,"%f"
    rem float_0==14.2115 float_1==14.2
float_2 = FCNV "-142.1e3",20,0,"%f"
```

2.1.25.6. Convert string to an Interger using a SFormat specifier

Convert a string with an integer number (ex: A$="154" or A$="F0E1") to an Integer variable using a Format specifier.

- The **ESize parameter is required**.
  It is the size of the string to convert (use 0 to convert the whole string).
- The **SFormat parameter is require**d.
  It is the format specifier string and can be:
  - "%d" if the string holds a decimal number.
  - "%o" if the string holds an octal number.
  - "%x" if the string holds an hexadecimal bumber.

- **Example**

```
a% = FCNV "1564",30,0,"%d" : rem a%=1564
a% = FCNV "1564",30,2,"%d" : rem a%=15
a% = FCNV "FE",30,0,"%x" : rem a%=254
a% = FCNV "11",30,0,"%o" : rem a%=9
```

2.1.25.7. Convert time as string into time as Integer

Convert a String holding a time in the format "dd/mm/yyyy hh:mm:ss" (ex: "28/02/2007 16:48:22") into an Integer holding the number of seconds since 01/01/1970 00:00:00.

- Important -

Float value have not enough precision to hold the big numbers used to represent seconds since 01/01/1970, this leads to lost of precision during time conversion.

- **Example**

```
a% = FCNV "24/04/2007 12:00:00",40 : rem a%=1177416000
a% = FCNV "01/01/1980 00:00:00",40 : rem a%=315532800
```

- **See also**

"TIME$" on page 104

2.1.26. FOR NEXT STEP

- **Syntax**

FOR a% = E1 TO E2 STEP E3
NEXT a%

- a% is an integer variable used as a counter.
- E1, E2, E3 are integer values/variables

- **Purpose**

The instructions between the lines containing the FOR and the NEXT are executed until a% = E2. The loop is always executed at least 1 time, even if E1 is greater than E2.

During first loop execution, a% equals E1.

FOR and NEXT cannot be on the same line of program.

Do not exit the FOR/NEXT loop by a GOTO statement because, in this case, after a certain

number of executions, the memory of the eWON is full.

- **Example**

```
FOR a%=10 TO 20 STEP 2
PRINT a%
NEXT a%
```

2.1.27. GET

The GET command works completely differently if the file is opened in Binary mode or in Text mode.

The file syntax has been extended in version 3 of the eWON to allow access to the serial port and to TCP and UDP socket.

The command description describes operation for

- /usr (Text and Binary modes)
- COM (always binary)
- TCP-UDP (always binary)

2.1.27.1. /usr in Binary mode

- **Syntax [function]**

GET E1, E2/S1

- E1 is the file number (1-8)
- E2 is the number of bytes to read from the file

Or

- If S1 is used, the function returns file specific information.

S1 Value	Return information
"SIZE"	Total file size

Table 11: Value /usr in Binary Mode

- **Purpose**

Returns a string of char with the data read. Moves the file read pointer to the character following the last character read (or to end of file).

- Get 1, 1 will return max 1 char
- Get 1, 5000 will return max 5000 char

- Get 1 without param is equivalent to Get 1,2048

- **Example**

```
OPEN "file:/usr/myfile.bin" FOR BINARY INPUT AS 1
A$ = GET 1,10
REM read 10 bytes
PRINT A$
CLOSE 1
```

2.1.27.2. /usr in Text mode

- **Syntax [function]**

GET E1[, E2]

- E1 is the file number (1-4)
- E2 optional: buffer size.
  When a data is read from the file, it must be read in a buffer to be interpreted. The buffer must be able to hold at least the whole item and the CRLF at the end of the line if the item is the last of the line.
  The default buffer size is 1000 bytes, if your file contains items that may be bigger than 1000 bytes, you should specify this parameter, and otherwise you only have to specify the E1 parameter: file number.

- **Purpose**

Returns a STRING or a FLOAT according to the data read from the file. If the data read is surrounded with quotes, it is returned as a STRING, if the data read is not surrounded with quotes, it is returned as a FLOAT.

The function never returns an INTEGER. The function moves the file read pointer to the next item. For string items, the ' quote or " quote can be used. The separator between items is the ';' character. When a CRLF (CHR$(13)+CHR$(10)) is found it is also skipped.

- **Example**

```
REM file content
123;"ABC"
1.345;"HOP"
DIM A$(2,20)
DIM A(2)
OPEN "/myfile.txt"
FOR TEXT INPUT AS 1
I%=1

ReadNext:
    IF EOF 1 THEN GOTO ReadDone
    A(I%) = GET 1
    A$(I%) = GET 1
```

```
    I% = I%+1
    GOTO ReadNext
ReadDone:
CLOSE 1
```

2.1.27.3. COM – Binary mode

- **Syntax [Function]**

GET E1,E2
CLOSE

- E1: File number
- E2: maximum number of bytes to read from the serial port.

- **Purpose**

Returns a string with the data read from the serial port buffer.

If there are no data to read from the buffer the returned string is empty.

If E2 is specified and the buffer contains more than E2 bytes, the function returns with E2 bytes.

If E2 is specified and the buffer contains less than E2 bytes, then the function returns with the content of the buffer.

The function always returns immediately.

- Note -

Attempting to USE a serial port used by an IO server is not allowed and returns an error.

- **Example**

```
OPEN "COM:2,... AS 1"
A$=GET 1,100
CLOSE 1
```

2.1.27.4. TCP/UDP in Binary mode

- **Syntax [function]**

GET E1, E2

- E1 is the file number returned by OPEN.
- E2: maximum number of bytes to read from the socket.

- **Purpose**

Returns a string with the data read from the TCP/UDP socket. If there are no data to read from the buffer the returned string is empty.

If E2 is specified and the buffer contains more than E2 bytes, the function returns with E2 bytes.

If E2 is specified and the buffer contains less than E2 bytes, then the function returns with the content of the buffer. If the other party has closed the socket or if the socket is in error at the TCP/IP stack level, the function exits with error (See "ONERROR" on page 74)

The function always returns immediately.

- **See also**

"CLOSE" on page 29, "EOF" on page 33, "OPEN" on page 79, "PUT" on page 87

2.1.28. GETFTP

- **Syntax [function]**

GETFTP S1, S2 [,S3]

- S1 is the name of the source file (to retrieve on the FTP server)
- S2 is the name of the destination file (to write on the eWON)
- S3 (optional) is the FTP server connection parameters.
  If S3 is unused, the FTPServer parameters from the General config page will be used.

- **Purpose**

Retrieves a file on an FTP server and copy it on the eWON.

The source filename can include a path (built with "/" or "\" depending of the FTP server). As the destination filename is on the eWON, you must begin by a "/" and can include a path built with "/".

The S3 parameters is as follow:

- [user:password@]servername[:port][,option1]

The option1 parameters is to force PassiveMode, put a value 1 as option1 parameter. If omitted, option1=0, then eWON will connect in ActiveMode.

This command posts a scheduled action request for a GETFTP generation.

When the function returns, the GETSYS PRG,"ACTIONID" returns the ID of the scheduled action and allows tracking this action. It is also possible to program an ONSTATUS action that

will be called when the action is finished (with or without success).

- **Example**

```
GETFTP "server_file_name.txt","/usr/ewon_file_name.txt"
GETFTP "server_file.txt","/usr/ewon_file.txt","user:pwd@ServerTP.com:21,1"
GETFTP "inst_val.txt","/inst_val.txt"
```

- **See also**

"ONSTATUS" on page 77, "GETSYS, SETSYS" on page 47, "PUTFTP" on page 89

2.1.29. GETHTTP

- **Syntax [function]**

GETHTTP S1,S2,S3[,S4]

- S1: Connexion Parameter
  with the format : [user:password@]servername[:port]
- S2: file name to assign on the eWON
  with the format : file name path
- S3: URI of the file on the HTTP
  with the format : server absolute path of the file to be downloaded
- S4 (optional): "PROXY"

- **Purpose**

The GETHTTP command submit a HTTP GET request. It allows the download of a file (one per GETHTTP command) using its URI.

When the function returns, the GETSYS PRG, returns the ID of the scheduled action and allows tracking of this action. It is also possible to program an ONSTATUS action that will be called when the action is finished (with or without success).

When "PROXY" is added at the end of the command, the eWON will perform the GETHTTP through a Proxy server. The eWON will use the Proxy server parameters configured in System Setup / Communication / VPN Global.

- **Example**

Download without HTTP basic authentification:

```
GETHTTP "10.0.100.206","/usr/filename1.txt","/filename1.txt"
```

When no port is specifed, HTTP port is 80.

Download with basic authentification and configured HTTP port:

```
GETHTTP "adm1:adm2@www.ewon.biz:89","/usr/filename1.txt","/filename1.txt"
```

HTTP server is supposed to listen on port 89 at address www.ewon.biz (adm1 is used as login. adm2 is used as password).

Download without HTTP basic authentification through Proxy serveur:

```
GETHTTP "10.0.100.206","/usr/filename1.txt","/filename1.txt","PROXY"
```

- **See also**

"ONSTATUS" on page 77, "GETSYS, SETSYS" on page 47, "PUTHTTP" on page 90

2.1.30. GETIO

- **Syntax [function]**

GETIO S1

- S1 is theTagRef which refers to the Tag reference (TagName, ID or -Index)
  See Tag Access on page 21

Returns the value of the S1 Tag. This value is a FLOAT.

- **Example**

```
A = GETIO "MyTag"
```

```
A = GETIO 12 : rem if TagID =12
```

This function is equivalent to A = MyTag@

- Important -

The MYTAG Basic variable is distinct than the memory Tag "MYTAG".

- **See also:**

"SETIO" on page 97

2.1.31. GETSYS, SETSYS

The GetSys and SetSys function are used to get or set some special parameters of the eWON.

There are 5 types of parameters:

Group	Description
PRG	Program parameters like the time in milliseconds or the type of action that started the program
SYS	Edition of the eWON system parameters
COM	Edition of the eWON communication parameters
USER	Edition of the eWON users list
TAG	Edition of the eWON Tag list
INF	Information about eWON (debug counter,...)

Table 12: GETSYS & SETSYS parameters

Each group has a number of fields that can be read of written.

2.1.31.1. PRG

Field Name	Op.	Type	Description
ACTIONID	R/W	I	After execution of a scheduled action like: SendSMS SendMail PutFTP SENDTRAP TCP/UDP Connect (see OPEN command), the ACTIONID returns the ID of the action just executed. When the ONACTION event is executed, this ACTIONID is stored in EVTINFO. Writing in this field is useful to read the current value of an action.
ACTIONSTAT	RO	I	Current status of the action with ActionID given by ACTIONID. If ACTIONSTAT must be checked, ACTIONID must be initialized first. Possible values of ACTIONSTAT are: • -1: in progress • -2: ID not found • 0: done with success • >0: finished with error = error code The eWON maintains a list with the status of the last 20 scheduled actions executed. When more actions are

Field Name	Op.	Type	Description
			executed, the older status is erased and its ACTIONSTAT may return –2, meaning it is not available anymore
EVTINFO	RO	I	The value of this field is updated before executing the ONXXXXX (ONSTATUS, ONERROR, etc.), see the different ONXXXXX function for the meaning of the EVTINFO parameter
TIMESEC	RO	I	Returns the time elapsed since 1/1/1970 in seconds. (Useful for computing time differences) Warning: when you assign this value to a float variable the number is too big and rounding will occur. You should use an integer variable (ex: a%) to store this value
MSEC	RO	I	Time in MSEC since eWON has booted Max value is 134217727 then it wraps to 0
RUNSRC	RO	I	When program is started, the source of the execution is given by this parameter: • 1: Started from the Web site 'Script Control' window • 2: Started by the FTP server because program has been updated • 3: A 'GO' command has been executed from the script • 4: Automatic program start at eWON boot
PPPIP	R/W	S/I	This parameter returns the string corresponding to the current PPP IP address. When the eWON is offline, the value returned is "0.0.0.0". When the eWON is online the value returned is the dotted IP address allocated for the PPP connection. The parameter can be written in order to disconnect the eWON. The only value accepted when writing in this parameter is 0 (*setsys prg, "PPPIP", 0*)
WANIP	RO	S	This parameter returns the string corresponding to the current WAN IP address. When the eWON is offline, the value returned is "0.0.0.0". If no-ip has been called, then WANIP returns the IP returned by no-ip, otherwise the actual physical WAN IP address (PPP or Ethernet) is returned. REM1: if getsys prg,"WANIP" is called in a ONWAN event it is likely that if a no-ip request is scheduled through publish ip address, it is not yet finished when the ONWAN is called. REM2: getsys prg,"WANIP" returns the same value as getsys prg,"PPPIP" if no-ip is not (yet) called and WAN interface is on modem, BUT writing to WANIP does not close the WAN connection like writing to PPPIP may close the PPP

Field Name	Op.	Type	Description
			connection.
VPNIP	RO	S	Currently allocated VPN ip address. If eWON is not connected to VPN this is 0.0.0.0
TRFWD	R/W	S	Transparent forwarding IP address. The parameter can be used to write or read the routing parameter. The parameter is only active when the PPP connection is established
SERNUM	R/W	S	Returns a string with the eWON serial number string
PRIOH	R/W	I	Used for changing the script priority
PRION	R/W	I	Used for changing the script priority
PRIOT	R/W	I	Used for changing the script priority
RESUMENEXT	R/W	I	Controls the OnError action. Possible values are a combination of: • 1: Resume Next mechanism is enabled • 4: Do not execute ONERROR • 8: Do not show error on virtual screen This parameter is useful when testing the existence of a variable, file or other. Example: Testing the existence of a file can be done by opening it and see if it generated an error. The Error result is accessible through LSTERR
LSTERR	R/W	I	Contains the code from the last Basic error that occurred (-1: no error). See "BASIC Error Codes" on page 113 The LSTERR is automatically cleared (value -1) when an end of section is reached (instruction END) You can also write the value -1 on LSTERR to clear the error (SETSYS PRG,"LSTERR",-1).
NBTAGS	RO	I	Returns the number of tags defined in eWON.
SCHRST	W	I	Clear all pending scheduled actions (except the action currently 'in progress'). Write only with the value 1 SETSYS PRG,"SCHRST",1 When Scheduled Actions are cleared, they have the status 'Action Canceled' (value 21613)
MDMRST	W	I	Force an Hardware Modem Reset SETSYS PRG,"MDMRST",1
ADSLRST	W	I	Force an Hardware ADSL Modem Reset SETSYS PRG,"ADSLRST",1

Table 13: PRG group fields

- Note -

Op. stands for Operation and contains 3 values : RO = Read Only, R/W = Read & Write and W = Write

Type may contain 3 values : I = Integer, R = Real and S = String

2.1.31.2. SYS

The fields edited with this group are the one found in the config.txt file under the section System. The fields are described in the section "Error: Reference source not found".

2.1.31.3. COM

The fields edited with this group are the one found in the comcfg.txt. The fields are described in the section "Error: Reference source not found". It is possible to tune the modem detection too. See COM Section on page 118.

2.1.31.4. INF

This group holds all information data about eWON. All these fields are Read Only. The fields displayed from this group are the one found in the estat.htm file.

2.1.31.5. TAG

The fields edited with this group are the one found in the config.txt file under the section TagList. The fields are described in the section "Error: Reference source not found".

See TAG Section on page 121.

2.1.31.6. USER

The fields edited with this group are the one found in the config.txt file under the section UserList. The fields are described in the section "Error: Reference source not found".

See User Section on page 125

2.1.31.7. Procedure

- A block must be loaded for edition with the SETSYS command and a special field called "load" (*SETSYS TAG,"load",XXXXXXX*). According to the source, this block will be either the eWON system configuration, the eWON COM configuration, a Tag configuration or a user configuration.
- Then each field of this configuration can be accessed by the GETSYS or SETSYS commands. This edition works on the record loaded values but does not actually

affect the configuration.

- When edition is finished, the SETSYS command is called with a special field called "save" and the edited block is saved (this is only necessary if the record has changed). At this time, the record edited content is checked and the configuration is updated and applied.
- The CFGSAVE command can be called to save the updated configuration to flash.

2.1.31.7.1. Recognized field values per group

The fields values are the same fields as those returned by the FTP get config.txt command.

- **Syntax**

GETSYS SSS, S1
SETSYS SSS, S1, S2 / E2

- SSS is the source block: PRG, SYS, TAG, USR - This parameter must be typed as is (it could not be replaced by a string)! • S1 is the field name you want to read or modify.
- S1 can be the action "load" or "save"
- S2 /E2 is the value to assign to the field, of which type depends on the field itself

- **Example**

```
A% = GETSYS PRG, "TIMESEC"
REM Suppose Tag_1 exists and is memory Tag
SETSYS TAG,"load","Tag_1"
A$ = GETSYS TAG,"Name" : REM A$="Tag_1"
SETSYS TAG,"ETO","ewon_actl@ewon.biz" : REM EmailTo field of Tag_1
SETSYS TAG,"save" : REM save data in the config => update Tag_1
SETSYS TAG,"Name","Tag_2"
SETSYS TAG,"save" : REM update or create Tag_2 with Tag_1 cfg
```

- **See also**

“CFGSAVE” on page 28, “Error: Reference source not found” on page Error: Reference source not found

2.1.31.7.2. TAG Load

The TAG load case is particular because is allows to load a Tag defined by its name, its ID or its Index.

If there are 6 Tags defined in the config, each Tag can be accessed by its index (0 to 5), its ID (the first item of a Tag definition when reloading the config.txt file, the ID of a Tag is never reused during the live of a configuration until the eWON is formatted) or finally by its name.

Method	XXX Param.	Example	Ex. Explanation
Tag name access	Tagname	Setsys Tag,"load","MyTagName"	Loads Tag with name MyTagName
Index access	-Index	Setsys Tag,"load",- 4	Loads Tag with index 4
TagId access	Id	Setsys Tag,"load",50	Loads Tag with id 50

Table 14: SETSYS TAG, "load" examples

- **See also**

Tag Access on page 21

2.1.31.7.3. Extended syntax to access IOServer lists of parameters

- **Generic Syntax**

GETSYS SYS,"ParamName:SubParamName"
SETSYS SYS,"ParamName:SubParamName","NewValue"

- ParamName is the name of the whole field (form the config.txt file).
- SubParamName is the sub-parameter (inside the ParamName) that you want to read or modify.
- NewValue is the value to assign to the field.

- **Specific IOServer Syntax**

GETSYS SYS,"IOSrvData[IOServerName]:SubParamName"
SETSYS SYS,"IOSrvData[IOServerName]:SubParamName","NewValue"

- IOServerName is the name of the IOServer you want to edit (form the config.txt file).
- SubParamName is the sub-parameter (inside the IOSrvData[...]) that you want to read or modify.
- NewValue is the value to assign to the field.

- **Purpose**

Allows an easy access to sub-parameters contained in a parameter String (since firmware 5.6s2).

Previously, to modify the IOSrvData2 parameter from the example below, you have to handle the whole string.

```
...
IOSrv0:EWON
```

```
IOSrv1:MODBUS
IOSrv2:UNITE
IOSrv3:
IOSrv4:
IOSrv5:
IOSrv6:
IOSrv7:
IOSrv8:
IOSrv9:
IOSrvData0:MinInterval:10¶MaxInterval:268435455¶ReverseCount:0
IOSrvData1:ComPortNum:1
IOSrvData2:EnabledA:1¶PeriodA:1000¶GlobAddrA:0,254,0¶EnabledB:0¶EnabledC:0¶Co
mPortNum:1¶Baudrate:19200¶Parity:2¶HWMode:1¶TwoStop: 0¶UVER2:1¶DisDefTr:0
IOSrvData3:
IOSrvData4:
IOSrvData5:
IOSrvData6:
IOSrvData7:
```

Now you can access directly the sub-parameter.

- **Example**

with Generic Syntax:

```
SETSYS SYS,"load"
A$ = GETSYS SYS,"IOSrvData2:GlobAddrA"
SETSYS SYS,"IOSrvData2:GlobAddrA","0,254,0"
```

with Specific IOServer Syntax:

```
SETSYS SYS,"load"
A$ = GETSYS SYS,"IOSrvData[UNITE]:GlobAddrA"
SETSYS SYS,"IOSrvData[UNITE]:GlobAddrA","0,254,0"
```

2.1.32. GO

- **Syntax [Command]**

GO

- **Purpose**

Start program execution (RUN). This is equivalent to clicking RUN in the script control window.

This command is mainly useful for remote eWON operation through the use of REMOTE.BAS FTP transfer.

- **See also**

"HALT" on page 55, "REBOOT" on page 92

2.1.33. GOSUB RETURN

- **Syntax**

GOSUB Label
Label:
Expression
RETURN

- **Purpose**

When the GOSUB line is executed, the program continues at "Label" line. Then the program executes up to the RETURN line. The RETURN command modifies the program pointer to the line immediately following the GOSUB Line.

- Important -

If the GOSUB line contains instruction after the GOSUB command, they won't be executed on return.

It is possible to create a new section containing the Label. Sections are useful in order to divide the program into smaller code snippets and help the reader to get a clear view of the software.

At the end of every section there is an invisible END but jumps are possible from section to section.

- **Example**

```
GOSUB NL3
PRINT " End "
END
NL3 : PRINT " Beginning "
```

```
RETURN
REM Display " Beginning " then " End "
GOSUB NL3 :print "Never"
PRINT " End "
END
NL3 : PRINT " Beginning "
RETURN
REM Display " Beginning " then " End " => "Never" is never printed
```

2.1.34. GOTO

- **Syntax [Command]**

GOTO Label

- **Purpose**

The execution of the program continues to the line indicated by Label. The Label statement cannot be empty

The GOTO command also allows starting the program without erasing all variables.

A string can be passed in a GOTO command.

- **Example**

```
GOTO Label
Print " Hop "
REM the program continues at line Label (Hop is not printed)
Label:
…
A$ = "my_label"
GOTO A$
```

2.1.35. HALT

- **Syntax [Command]**

HALT

- **Purpose**

Stops program execution. This is equivalent to clicking STOP' in the script control window. This command is mainly useful for remote eWON operation through the use of REMOTE.BAS FTP

transfer.

- **See also**

"GO" on page 53, "REBOOT" on page 92

2.1.36. HEX$

- **Syntax [function]**

HEX$ E1

- **Purpose**

The function returns a chain of 8 characters that represents the hexadecimal value of the E1 number.

- **Example**

```
a$= HEX$ 255
REM A$ is worth " 000000FF " after this affectation
```

- **See also**

"BIN$" on page 27

2.1.37. HTTPX

The Basic Script implemented in the eWON is capable of dealing with HTTP(S) request & response.

2.1.37.1. REQUESTHTTPX

- **Syntax**

REQUESTHTTPX http[s]://S1, S2 [, S3 [, S4 [, S5 [, S6 [, S7]]]]]

- **S1 is the Server**
  It is the URL of the targeted request. For example : "192.168.0.10" or "www.example.com"
  It is also part of the URL that constitute the Query string. For example: "service" or "12345/control?axis=x&val=1"
- **S2 is the Method**
  It's the REST API HTTP verb. This can be "Get", "Post", "Put", "Patch", "Delete",

"Options", "Head", "Purge"

- **S3 are the Headers** (optional)
  The headers the eWON sends through the request. For example "ContentType=application/json&XRequest =test"
- **S4 is the Post-Data** (optional)
  The POST data can be either:
  - separated by an ampersand "&" using the traditional QUERYSTRING format [FieldName1=ValueNa me1] [&FieldNameX=ValueNa meX]*
  - raw data

  For example: "firstname=jack&lastname=nicholson" or "{\"myData\":21}"
- **S5 is the File-Data** (optional)
  String for FILE data separated by an ampersand using the traditional QUERYSTRING format and having each value corresponding to an eWON ExportBlockDescriptor: [FieldName1=ExportBlockDescriptor1]
  For example: "pictures[]=[$dtEV$fnevents.txt]&pictures[]=[$dtCF$fnconfig.txt]"
- **S6 is the File-Answer** (optional)
  The file name inside /usr/ folder where the answer needs to be stored.
  For example: "/usr/myfile"
- **S7 is the Proxy** (optional)
  If the request should use a Proxy or not. Accepts "PROXY" or "" as value.

When File-Answer is empty or not specified :

- the result of the request is saved in a buffer in the memory. The information can then be retrieved with the RESPONSEHTTPX command (cf. Infra).
- there are three buffers: each buffer can handle a response body of max. 64KB. An HTTP request error is produced if the response body is bigger than the max. size allowed.

Whenever the POST-DATA field is specified without any FILE-DATA information, the default content type header (if not specified in the HEADER field) is

```
'Content-Type: application/x-www-form-urlencoded; charset=ISO-8859-1'
```

When using a "multipart/form-data" content type, it isn't possible to set the boundary.

```
"Content-Type:multipart/form-data; boundary=--------myseparator"
```

This would not be supported.

2.1.37.2. RESPONSEHTTPX

- **Syntax**

RESPONSEHTTPX S1 [, S2]

- **S1 is the Parameter**
  Sending the info to retrieve. For example: "HEADER", "STATUSCODE", "RESPONSE_BODY"
- **S2 is the Specific-Header** (optional)
  Set a specific header to retrieve. This works only when S1 is set to "HEADER"

RESPONSEHTTPX is used to retrieve the information from a previous REQUESTHTTPX command.

Use the ACTIONID (parameter from the GETSYS PRG) to specify the request:

- **RESPONSEHTTPX "HEADER"**
  returns all server headers with the format "HeaderName: value" separated by CR+LF (ascii 13dec then 10dec).
- **RESPONSEHTTPX "HEADER", "Specific-Header"**
  returns only "Specific-Header: value"(or an empty string if not found")
- **RESPONSEHTTPX "STATUSCODE"**
  returns the request status code ("200", "404", etc) as a string
- **RESPONSEHTTPX "RESPONSE-BODY"**
  returns the response body as a string that can contain NULL characters.

- **Example**

```
request:
REQUESTHTTPX "http://www.example.com/coucou.php","GET"
actionID% = GETSYS PRG,"ACTIONID"
PRINT "request actionid is "; actionID%
END

onEvent:
eventId% = GETSYS PRG,"EVTINFO"
IF (eventId% = actionID%) THEN
  SETSYS PRG,"ACTIONID",eventId%
  stat% = GETSYS PRG,"ACTIONSTAT"
  IF (stat% = 0) THEN
    GOTO response
  ELSE
    PRINT "Error (ERROR = "+Str$(stat%) + ")"
  ENDIF
ENDIF
END
```

```
response:
a$ = RESPONSEHTTPX "STATUSCODE"
PRINT "***status: "; a$
a$ = RESPONSEHTTPX "HEADER"
PRINT "***all headers: "; a$
a$ = RESPONSEHTTPX "HEADER", "Server"
PRINT "***server header: "; a$
a$ = RESPONSEHTTPX "RESPONSEBODY"
IF (Len(a$) < 1000) THEN
  PRINT "***response body: "; a$
Else
  PRINT "***response body size: "; Len(a$)
ENDIF
END
```

2.1.38. IF, THEN, ELSE, ENDIF

This sequence of commands now supports two different syntaxes: the short IF syntax and the long IF syntax.

2.1.38.1. Short IF Syntax

- **Syntax**

IF N THEN EXPRESSION1 [ELSE EXPRESSION2 [ENDIF]]

- **Purpose**

The condition is the result of an operation returning an N integer.

- If N is 0, the condition is considered as false and the eWON executes the following line or to the ELSE "expression2" if present.
- If N is different from 0, the condition is considered as true and the eWON executes "expression1".
  - If more than one instruction has to be executed, separate them with ':'.
  - If N is an expression or a test, use ().

The ELSE Expresion2 is optional and the finishing ENDIF is also optional.

- Important -

The short IF syntax is used as soon as an item is found after the THEN statement. Even putting a REM statement on the IF N THEN line will make the eWON consider it as a short IF statement.

2.1.38.2. Long IF Syntax

- **Syntax**

IF N THEN
    Expression1
ELSE
    Expression2
ENDIF

The ELSE Expression2 is optional but ENDIF is mandatory.

You can mix short and long IF syntax in your code, but don't forget that anything typed after the THEN statement will lead to a short IF syntax interpretation.

- **Example**

```
IF (a<10) THEN PRINT"A is lower than 10": SETIO"MyTag",1
```

```
IF (a<10) THEN
    PRINT"A is lower than 10": MyTag@ = 1
ELSE
    PRINT"A is bigger than 10": MyTag@ = 0
ENDIF
```

2.1.39. INSTR

- **Syntax [Function]**

INSTR I1,S1,S2

- I1 is the index in the string to search (valid value goes from 1 to LEN S1)
- S1 is the string to be search for S2
- S2 is the string to search for in S1

- **Purpose**

The function returns an integer equal to the position of string S2 in string S1.

- If string S2 is found, the function return a value from 1 to the length of S1 (The returned index is 1 based).
- If string S2 is not contained in S1, the function returns 0.

The I1 parameter should be 1 to search the whole S1 string.

If I1 is >0 then string S1 is searched starting at offset I1. The value returned is still referenced to

the beginning of S1.

- **Example**

```
INSTR 1, "AAABBC","BB" = 4
```

```
INSTR 3, "AAABBC","BB" = 4
```

- Note -

As internally, the INSTR function uses the character zero (0x00) as delimiter, you cannot search for character zero with INSTR. B$=CHR$(0) : A% = INSTR 1,A$,B$ will always return 1, whichever a zero character is present or not.

2.1.40. INT

- **Syntax [function]**

INT F1

- **Purpose**

Extract the integer part of the number. There is no rounding operation.

- **Example**

```
A = INT(10.95)
REM A equals 10.00
```

```
A% = 10.95
REM A equals 10 --- automatic type conversion
```

2.1.41. IOMOD

- **Syntax [function]**

IOMOD TagRef

- TagRef is the Tag reference (TagName, ID or -Index)
  See Tag Access on page 21

- **Purpose**

Returns '1' if the TagRef Tag value has been modified in the eWON since the last call to the IOMOD command.

The call to this function resets Tag change internal flag to 0. i.e. if the variable doesn't change anymore, the next call to IOMOD will return 0. You can achieve an equivalent behavior with the use of ONCHANGE event handler.

- **Example**

```
a% = IOMOD " MYTAG "
IF a% THEN PRINT " mytag has changed "
```

- **See also**

“ONCHANGE” on page 71

2.1.42. IORCV

- **Syntax [function]**

IORCV S1[,I1]

- S1 is the STRING IOServerName parameter
- I1 is an optional additional parameter (= 0 OR = 1 OR = -1)

- **Purpose**

The IOSEND and IORCV functions must be used together. They are used to send/receive custom IOServer requests.

These functions can only be used if IOServer is configured.
Use IORCV function for reading IO server response to an IOSEND request.

- Note -

There are three transmission slots available, using IORCV allows you to free them before the three slots are busy. Requests are interlaced with gateway requests sent to the IO server and with normal IO server polling operations.

- **First Case**

```
a$ = IORCV a%
a$ = IORCV a%,0
```

Returns the result or the status of the Request.

a% holds the request number and is the result of the IOSEND command.

a$="XXXXXXXXX"	where XXXXXXXXX is the result of the request
a$="#FREE"	slot a% is free
a$="#RUN"	slot a% is in progress
a$="#ERR"	slot a% is done with error

Table 15: First case, I = 0 for IORCV command

If the request is done (all cases except "#RUN"), the slot is always freed after the "IORCV a %" or "IORCV a%,0".

- **Second Case**

```
a$ = IORCV a%,-1
```

Same as for "a$=IORCV a%,0", but the slot is not freed if a request is done.

- **Third Case**

```
b% = IORCV a%,1
```

Returns the status of the IORCV command in INTEGER format. The slot is not freed by this parameter.

The returned status can contain the following values:

b%= -2	slot a% is free
b% = -1	slot a% is in progress
b% = 0	slot a% is done with success
b% > 0	slot a% is done with error
b% < -2	lot a% is done with error - code type: warning. Such warning codes mean "Read failed" on the serial link. These warnings are flagged as internal and thus are not added in the event log. The warning codes can be very long; ie. -536893114

Table 16: Third case, I = 1 for IORCV command

- **Example**

```
TestIO:
A$ = chr$(4)+chr$(0)+chr$(0)+chr$(0)+chr$(1) : rem create modbus command
rem initiate the modbus request on slave 21
a% = IOSEND "MODBUS","21",A$

Wait_IO_End:
b% = IORCV a%,1 : rem read the status
IF b%=-1 THEN
    GOTO Wait_IO_End : rem if idle then loop
ENDIF

B$ = IORCV a% : rem read the result and free the slot
PRINT LEN(B$)
PRINT B$ END
```

- **See also**

"IOSEND" on page 64

2.1.43. IOSEND

- **Syntax [function]**

IOSEND S1, S2, S3

- **Purpose**

Sends a request by using the IO server's protocol.

See "IORCV" on page 62 for an example of how this function must be used.

Parameters are:

- STRING IOServerName: IO Server name as it appears in the Tag configuration page
- STRING Address: Slave address as described in the eWON User manual for each IO server section
- STRING IoCommand: Array of bytes with a protocol command, the content depends on the IO server.

Returns a request number (slot) that must be used in IORCV for reading the response to the request.

- Note -

The request result is read by using the IORCV function and uses a polling mechanism. That means that you need to use IORCV in order to check with the request received with IOSEND that the slot is free.

There are three transmission slots available, using IORCV allows you to free them before the three slots are busy. Requests are interlaced with gateway requests sent to the IO server and with normal IO server polling operations.

- **Example**

```
a% = IOSEND IOServerName,Address,IoCommand
```

- **See also**

"IORCV" on page 62

2.1.44. LEN

- **Syntax [function]**

LEN S1

- **Purpose**

The function returns the number of characters in a string.

- **Example**

```
a$= "Hop "
A% = LEN A$
```

```
REM a% equal 3
```

2.1.45. LOGEVENT

- **Syntax [command]**

LOGEVENT S1 [,S2]

- S1 is the phrase to log
- S2 is the type of logging. This parameter is optional and can take the following ranges of values:

Range of values	Description
0 … 99	Error
-99 … -1	Warning
100 … 199	Trace

Table 17: LOGEVENT – Range of values

If the logging level is not specified, it is considered to be an error.

- **Purpose**

Appends an event to the log file. The current time is automatically used for event logging.

- **Example**

```
logevent "Save this in log", 120
REM Would append 978353046;"01/01/2001 12:44:06";"Save this in log" to the
log file.
```

2.1.46. LOGIO

- **Syntax [command]**

LOGIO TagRef

TagRef is the Tag reference (TagName, ID or -Index) See Tag Access on page 10

- **Purpose**

Force historical logging of TagRef Tag.

The Tag must have historical logging enabled (not available on all eWON's versions).

The point is logged at the time the LOGIO command is issued with its current value.

- Note -

If the Tag is configured for historical logging with logging dead band equal to –1 and time interval equal to 0, no point will be logged automatically and it is possible to program a purely scripted logging.

- **Example**

```
LOGIO "mytag"
```

2.1.47. LTRIM

- **Syntax[Command]**

LTRIM S1

- S1 is a string.

- **Purpose**

LTRIM returns a copy of a string with the leftmost spaces removed.

- **Example**

```
b$ = LTRIM a$
```

- **See also**

"RTRIM" on page 94

2.1.48. MEMORY

- **Syntax**

MEMORY S1

- S1 can be one of the 3 next values:
  - "PROG" returns the free memory of the program zone
  - "VAR" returns the free memory of the variable zone
  - "TOT" returns "PROG" + "VAR"

2.1.49. MOD

- **Syntax [Operator]**

E1 MOD E2

- **Purpose**

Compute the remainder of the division of E1 by E2

- **Example**

```
1 MOD 2
REM returns 1
2 MOD 2
REM returns 0
```

- **See also**

"Operators priority" on page 17

2.1.50. MONTH

- **Syntax [Function]**

MONTH E1

- E1 is a date in integer format (number of seconds since 1/1/1970)
- S1 is a date in String format ("18/09/2003 15:45:30")

- **Purpose**

This function returns an integer corresponding to the value of the month (1--12) that matches a defined time variable.

- Important -

Do not call the function with a float variable of value (or this would result to error "invalid parameter").

- **Example**

```
a$ = TIME$
a% = MONTH a$
```

```
b% = getsys prg,"TIMESEC"
a% = MONTH b%
```

- **See also**

"DAY" on page 30 "DOW" on page 31, "DOY" on page 32, "WOY" on page 109

2.1.51. NOT

- **Syntax [function]**

NOT E1

- **Purpose**

The function returns '1' if E1 is equal to '0' otherwise the function returns 0.

- **Example**

```
IF NOT a% THEN PRINT " A% is worth 0 "
```

- **See also**

"Operators priority" on page 17

2.1.52. NTPSync

- **Syntax [function]**

NtpSync

- **Purpose**

Posts a request for clock resynchronization (even if this feature is disabled in the configuration).

2.1.53. ONxxxxxx

There are some ONxxxxxx commands listed below.

These commands are used to register a BASIC action to perform in case of special conditions. For every ONxxxxxx command, the action to execute is a string that is used as a BASIC command line.

When the condition occurs, the command is queued in an execution queue and is

executed when its turn comes.

These functions are:

Onxxxx command	Description
ONTIMER	Executed when one of the timers expires
ONCHANCE	Executed when a Tag changes (value or configuration)
ONALARM	Executed when a Tag alarm state changes
ONERROR	Executed when an error occurs during BASIC execution
ONSTATUS	Executed when a scheduled action is finished (success or failure)
ONSMS	Executed when a SMS is received (only for eWON with GSM/GPRS Modem)
ONPPP	Executed when the PPP connection goes online or offline
ONVPN	Executed when the VPN goes connected or disconnected
ONWAN	Executed when the WAN goes connected or disconnected

Table 18: The various "ONXXXX" functions

When the command line programmed is executed, a special parameter is set in SETSYS PRG,"EVTINFO". The value of the parameter depends on the ONxxxxxx function and it can be checked with the GETSYS command.

- Important -

For all ONxxxx command, if the last parameter is omitted, the action is canceled.

- **Example**

```
ONTIMER 1
REM will cancel any action programmed on TIMER 1.
```

- **See also**

"GETSYS, SETSYS" on page 47, ONxxxxx (following chapters)

2.1.53.1. ONALARM

- **Syntax [command]**

ONALARM TagRef,S2

- TagRef is the Tag reference (TagName, ID or -Index)
  See Tag Access on page 21
- S2 is the command line to execute in case of alarm state change.

- **Purpose**

Executes the S2 command line when alarm state on TagRef Tag changes. The EVTINFO parameter (see GETSYS page 24) is set to the Tag ID when command is called.

- Note -

*ONALARM will execute the command when the alarm status gets the value "2" (or above), that means that ONALARM **does not detect** the "pre trigger" status (value=1).*

- **Example**

```
ONALARM "MyTag","goto MyTagAlarm"
```

- **See also**

"ALSTAT" on page 25, "GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69, "ONCHANGE" on page 71

2.1.53.2. ONCHANGE

- **Syntax [command]**

ONCHANGE TagRef,S2

- TagRef is the Tag reference (TagName, ID or -Index)
  See Tag Access on page 10
- S2 is the command line to execute in case of value change.

- **Purpose**

Executes S2 command line when the TagRef Tag changes (value or configuration).

The EVTINFO parameter (see "GETSYS, SETSYS" on page 24) is set to the Tag ID when

command is called.

- Note -

The ONCHANGE is triggered when:
the value of the Tag changes
the configuration of the tag is updated.

- **Example**

```
ONCHANGE "MyTag","goto MyTagChange"
```

- **See also**

"IOMOD" on page 62, "GETSYS, SETSYS" on page47, "ONxxxxxx" on page 69

2.1.53.3. ONDATE

- **Syntax [command]**

ONDATE I1,S1,S2

- I1 is the planner entry index to set (from 1 to 10).
- S1 is the Timer Interval string.
- S2 is the Basic command(s) to execute at the specified interval.

ONDATE I1

- I1 is the planner entry index to delete (from 1 to 10).

- **Purpose**

The ONDATE function allows you to defined 10 plannified tasks.

Available since Firmware 5.7.

All ONDATE entries are deleted automatically when the program is stopped by the RUN/STOP link.

2.1.53.3.1. Timer Interval settings

The syntax of the S2 parameter is the following: **mm hh dd MMM DDD**

Field	Settings
mm	This is the Minute parameter. A number between 0 to 59

Field	Settings
hh	This is the Hour parameter. A number between 0 to 23
dd	This is the Day parameter. A number between 1 to 31
MMM	This is the Month parameter. A number between 1 to 12 Or the month name abbreviation in english (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec)
DDD	This is the Day Of Week parameter. A number between 1 to 7 with 1=monday, 2=tuesday, ..., 7=sunday Or the day name abreviation in english (mon, tue, wed, thu, fri, sat, sun)

Table 19: ONDATE – Timer Interval syntax

- Important -

These 5 parameters are all required! When used together, the dd and DDD parameters make an OR operation (every dd of the month or DDD).

In addition, there are some operators to specify multiple date/time.

Field	Settings
*	The * (asterisk) operator specifies all possible values for a field from Table 14. For example, an * in the hh time field would be equivalent to 'every hour'.
,	The , (comma) operator specifies a list of values. For example: "1,3,4,7,8" (space inside the list must not be used)
-	The - (dash) operator specifies a range of values. For example: "1-6", which is equivalent to "1,2,3,4,5,6".
/	The / (slash) operator (called "step"), which can be used to skip a given number of values. For example, "*/3" in the hour time field is equivalent to "0,3,6,9,12,15,18,21".

Table 20: ONDATE – Timer Interval Operators

- **Example**

Field	Settings
ONDATE 1,"* * * * *","GOTO MyFunc"	will do "GOTO MyFunc" every minutes.

Field	Settings
ONDATE 1,"0 * * * *","GOTO MyFunc"	will do "GOTO MyFunc" every hour.
ONDATE 1,"0 0 * * *","GOTO MyFunc"	will do "GOTO MyFunc" on every day at midnight (00:00).
ONDATE 1,"*/15 * * * *","GOTO MyFunc"	will do "GOTO MyFunc" every 15 minutes.
ONDATE 1,"15 7 1 1 *","GOTO MyFunc"	will do "GOTO MyFunc" at 7:15, the first of january. Could also be written like '15 7 1 jan *'
ONDATE 1,"15 8 * * 1","GOTO MyFunc"	will do "GOTO MyFunc" at 8:15, each monday. Could also be written also like '15 8 * * mon'
ONDATE 1,"0 8-18 * * 1-5","GOTO MyFunc"	will do "GOTO MyFunc" at every hour between 8:00 and 18:00 on every working day (monday to friday)
ONDATE 1,"0 6,7,18,19 * * *","GOTO MyFunc"	will do "GOTO MyFunc" at 6, 7, 18 and 19 o'clock on every day.
ONDATE 1,"* * 13 * fri","GOTO MyFunc"	will do "GOTO MyFunc" at every minutes on each friday OR the 13th of the month (and not only on the friday 13th).
ONDATE 1	will delete the plannified entry 1

Table 21: Task Planner – Timer examples

- **See also**

"TSET" on page 105, "ONTIMER" on page 77

2.1.53.4. ONERROR

- **Syntax [command]**

ONERROR S1

- S1 is the command line to execute when an error occurs during program execution .

- **Purpose**

The EVTINFO parameter (See GETSYS, SETSYS on page 24) is set to the code of the error.

- **Example**

```
ONERROR "goto TrapError"
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69

2.1.53.5. ONPPP

- **Syntax [command]**

ONPPP S1

- S1 is the command line to execute when the PPP connection goes online or offline.

- **Purpose**

The EVTINFO parameter (see GETSYS page 24) is set to one of the following values:

EVTINFO Value	Situation
1	The PPP connection has gone ONLINE
2	The PPP has gone OFFLINE

Table 22: ONPPP – EVTINFO values

- **Example**

```
ONPPP "goto PppAction"
END PppAction:
I%=GETSYS PRG,"EVTINFO"
IF I%=1 THEN
    PRINT "Online with address ";GETSYS PRG,"PPPIP"
ELSE
    PRINT "PPP Going offline"
ENDIF
END
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69

2.1.53.6. ONSMS

- **Syntax [command]**

ONSMS S1

- S1 is the command line to execute when eWON receives a SMS.

- **Purpose**

A typical use of the ONSMS syntax is allowing eWON to send a read SMS receipt to the SMS sender

You can read the received SMS with GETSYS PRG function with

- smsRead:
  - hold 1 if there is a new SMS (reading smsRead load the other parameters)
  - hold 0 if the SMS queue is empty
- smsFrom: String holding the phone number of the sender
- smsDate: String holding the Date of SMS reception
- smsMsg: String holding the SMS message

- **Example**

```
InitSection:
ONSMS "Goto Hsms"
END

Hsms:
a% = getsys prg,"SmsRead"
IF (a%<>0) then
    s% = s%+1
    print "SMS Nr: ";s%
    f$ = getsys prg,"smsfrom"
    print "From: ";f$
    print getsys prg,"smsdate"
    a$ = getsys prg,"smsmsg"
    print "Message: ";a$
    b$ = f$+",gsm,0"
    c$ = "Received message: "+a$
    sendsms b$,c$
    goto HSms
ENDIF
END
```

2.1.53.7. ONSTATUS

- **Syntax [command]**

ONSTATUS S1

- S1 is the command line to execute when a scheduled action is finished.

- **Purpose**

The EVTINFO parameter (see GETSYS page 24) is set to the ACTIONID of the finished action when command is called. This function can be used to track success or failure of scheduled actions.

- **Example**

```
ONSTATUS "goto Status"
```

- **See also**

"GETSYS, SETSYS" on page 47, "GETSYS, SETSYS" on page 47, "PUTFTP" on page 89, "SENDMAIL" on page 94, "SENDSMS" on page 95, "SENDTRAP" on page 96

2.1.53.8. ONTIMER

- **Syntax [command]**

ONTIMER E1,S1

- E1 is the timer number (see TSET page 105)
- S1 is the command line to execute when timer expires.

- **Purpose**

Executes S1 command line when E1 expires.

The EVTINFO parameter (see GETSYS, SETSYS page 47) is set to the timer number when command is called.

- **Example**

```
ONTIMER 1,"goto Timer1"
ONTIMER 1, "LOGIO 'mytag' "
```

- **See also**

"GETSYS, SETSYS" on page47, "ONxxxxxx" on page 69 "TSET" on page 105

2.1.53.9. ONVPN

- **Syntax [command]**

ONVPN S1

- S1 is the command line to execute when the VPN connection status change (at connection or at disconnection).

- **Purpose**

The EVTINFO parameter (see GETSYS page 24) is set to one of the following values:

EVTINFO Value	Situation
1	The VPN connection has gone ONLINE
2	The VPN has gone OFFLINE

Table 23: ONVPN – EVTINFO values

- **Example**

```
ONVPN "goto VPN_Action"
END
VPN_Action:
I%=GETSYS PRG,"EVTINFO"
IF I%=1 THEN
    PRINT "VPN Online"
ELSE
    PRINT "VPN Going offline"
ENDIF
END
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69

2.1.53.10. ONWAN

- **Syntax [command]**

ONWAN S1

- S1 is the command line to execute when the WAN connection status change (at connection or at disconnection).

- **Purpose**

The EVTINFO parameter (see GETSYS page 24) is set to one of the following values:

EVTINFO Value	Situation
1	The WAN connection has gone ONLINE
2	The WAN has gone OFFLINE

Table 24: ONWAN – EVTINFO values

- **Example**

```
ONWAN "goto WAN_Action"
END
WAN_Action:
I%=GETSYS PRG,"EVTINFO"
IF I%=1 THEN
    PRINT "WAN Online with address";GETSYS PRG,"WANIP"
ELSE
    PRINT "WAN Going offline"
ENDIF
END
```

- **See also**

"GETSYS, SETSYS" on page47, "ONxxxxxx" on page 69

2.1.54. OPEN

2.1.54.1. Introduction to file management

Files accessed in BASIC can be of 4 different types:

- Files from the /usr directory
- Serial communication link
- TCP or UDP socket
- Export Block Descriptor

2.1.54.2. OPEN general syntax

There are two different modes of operation for the file access:

- BINARY mode: file is read by blocks of bytes

- TEXT mode: files are read or written as CSV files

See the "GET" on page 41 and "PUT" on page 87 commands for a detailed difference between the BINARY and TEXT mode outputs.

There are 3 operation types:

Parameter Value	Description
INPUT	The file must exist. It is opened for a read only operation. The file pointer is set to the beginning of the file
OUTPUT	The Path must exist. If the file exists it is erased first. The file is opened for write only operation
APPEND	The Path must exist. The file doesn't have to exist. If the file does not exist, it is created (like with OUTPUT type). If the file exists, it is opened and the write pointer is located at the end of the file. The file is opened for write only operation

Table 25: OPEN read & write operations parameters

When binary mode is used, the data written to the file are strings of characters that are considered as stream of bytes.

The GET command returns the amount of bytes requested. When Text mode is used, the operation is completely different: the PUT operation is more like a PRINT command directed to file, the data are formatted as text, and each data is separated by a ';' in the output file (strings are exported between quotes).

The GET command works like a READ command, the file is read sequentially and each GET returns one of the ';' separated element, the type of the data returned depends on the type of data read. In both modes, files are read sequentially until end of file is reached. End of file can be tested with the EOF function.

The eWON user flash file system allows up to 8 files to be simultaneously opened for read (even twice the same file), and 1 file opened for write.

If a file is opened for read it cannot be opened for write at the same time (and vice versa).

Running the program will close any previously opened files (not GOTO).

2.1.54.3. Different File/stream types

2.1.54.3.1. FILE open /usr

- **Syntax [command]**

OPEN S1 FOR BINARY|TEXT INPUT|OUTPUT|APPEND AS E1

- E1 is the file number. After the OPEN operation, the file is referenced by its file number

and not by its file name. There are 8 file numbers available. Once a file number is assigned to a file, it is allocated to that file, until the CLOSE command is issued.

- S1 describes the access to a file that is located on eWON directories. S1 must respect the following syntax:
  - "file:/directory/filename"
    This allows to read or write files in the /usr directories. You will not be able to access the files in the root (virtual files like config.txt) with this command.

Example	Comment
OPEN "file:/sys/test.dat" FOR BINARY INPUT AS 1 A$=GET 1,4 CLOSE 1	Opens file 1 Reads 4 bytes

Table 26: OPEN – different file type example 1

  - If S1 does not begin by "file:", "tcp", "com", or "exp", then the file will be considered as being part of the /usr directory.
    The following syntax is the old one (version 3) and is kept for compatibility purpose.

Example	Comment
OPEN "test.dat" FOR BINARY INPUT AS 1 A$=GET 1,4 CLOSE 1	Open the /usr/test.dat file Reads 4 bytes

Table 27: OPEN – different file type example 2

2.1.54.3.2. TCP or UDP stream open Syntax [command]

OPEN S1 FOR BINARY INPUT|OUTPUT AS E1

- S1 must respect the following syntax:
- "tcp:Address:dest_Port[,TimeOut]"
  "udp:Address:dest_Port[:src_Port][,TimeOut]"
  - Address can be a dotted IP address like 10.0.0.1 or a valid resolvable internet name like ftp.ewon.be
  - dest_Port must be a valid port number from 1 to 65535.
  - src_Port (optional)
    If defined, the return port will be forced to the src_Port value (works only with UDP protocol).
    If not defined, the return port is allocated automatically by the eWON TCP/IP stack.
  - TimeOut (optional) is the number of seconds eWON will wait to decide if the OPEN command failed (default : 75 sec)

- E1 is the file number. After the OPEN operation, the file is referenced by its file number and not by its file name. There are 8 file numbers available. Once a file number is assigned to a file, it is allocated to that file, until the CLOSE command is issued.

- Note -

This command works only with BINARY

- Important -

For scheduled action: when the OPEN command is used to open a TCP connection, the command returns before the connection is actually opened. A scheduled action is posted because opening the socket may require a dial out or take up to more than a minute, and the BASIC cannot be stopped during that time.

In order to know if the connection is established, the user has 2 options:

- Check the scheduled action status by checking the PRG,ACTIONSTAT (See GETSYS, SETSYS on page 24).
- Read the file with GET: as long as the file is not actually opened, the function returns #CLOSED#.
  When the function stops sending #CLOSED# the file can be read and written for socket operations.

- **Example**

```
OPEN "tcp:10.0.0.1:25" FOR BINARY OUTPUT AS 1
PUT 1,CHR$(13)+CHR$(10)
A$=GET 1
CLOSE 1
```

Opens socket to 10.0.0.1 port 25 for read/write access. Write a CRLF then read response.

2.1.54.3.3. COM port open Syntax [command]

OPEN S1 FOR BINARY INPUT|OUTPUT AS E1

- S1 will be as follows: **com:n,b,dpsh**
  - where n is 1 to 4 (the port number, 1 is Front panel serial port, 2 is Modem Port)
  - where b is the baud rate
  - where d is the number of bits "7" or "8"

- where p is the parity: "e","o" or "n"
- where s is the number of stop bit "1" or "2"
- where h is the handshaking "h": half duplex, "r": yes Rts/Cts, "n": No

- This command will open the serial port to port 1 to 4 with the given line parameters.
- E1 is the file number. After the OPEN operation, the file is referenced by its file number and not by its file name. There are 8 file numbers available. Once a file number is assigned to a file it is allocated to that file until the CLOSE command is issued.

- Note -

This command works only with BINARY. Both INPUT and OUTPUT modes allow to both Read and Write on the COM port.

Attempting to USE a serial port used by an IO server is not allowed and returns an error.

- **Example**

```
OPEN "com:1,9600,8n1n" FOR BINARY OUTPUT AS 3
```

Opens the COM1 (Serial port 1) with parameters: speed 9600, bit 8, parity none, stop bit 1 and handshaking no.
The file number used is 3. You are in Binary mode and you can read and write.

2.1.54.3.4. EXP export bloc descriptor open Syntax [command]

OPEN S1 FOR TEXT|BINARY INPUT AS E1

- S1 will be as follows: "exp:XXXXX", where XXXXX is an Export Block Descriptor.
- E1 is the file number. After the OPEN operation, the file is referenced by its file number and not by its file name. There are 8 file numbers available. Once a file number is assigned to a file it is allocated to that file until the CLOSE command is issued. When the export block has been read (or not if you close before end) you must call CLOSE to release memory.

- Important -

You cannot use the PUT command with a EXP: file

- **Example**

```
OPEN "exp:$dtAR $ftT" FOR TEXT INPUT AS 1
Loop:
A$ = Get 1
PRINT A$
IF A$ <> "" THEN GOTO Loop
CLOSE 1
```

In this case the "a$ = get 1" can be called until it returns an empty string to read the content of the Export Block Descriptor; the data are then read by blocks of maximum 2048 bytes. If you want to reduce or increase that size, you can call "a$ = get 1,y", where y is the maximum number of bytes you want the function to return (do not put y=0).

```
OPEN "exp:$dtUF $ftT $fn/myfile.txt" FOR TEXT INPUT AS 1
A$ = Get 1
PRINT A$
CLOSE 1
```

- **See also**

“CLOSE” on page 29 “EOF” on page 33, “GET” on page 41, “PUT” on page 87

2.1.55. OR

- **Syntax [Operator]**

E1 OR E2

- **Purpose**

Does a bit-by-bit OR between the 2 integers E1 and E2.

Pay attention that:

- When executed on float elements (float constant or float variable), the OR functions returns the logical OR operation.
- When executed on integer elements (integer constant or integer variable - like i%), the OR function returns the bitwise OR operation
- This is NOT true for AND and XOR
- This is historical and is left for compatibility with existing programs

- **Example**

```
1 OR 2 REM returns 3
2 OR 2 REM returns 2
3 OR 1 REM returns 3
```

- **Logical OR**

```
var1=0.0
var2=0.0
ORResult = var1 OR var2
Print ORResult
rem ORResult = 0.0
```

```
var1=0.0
var2=12.0
ORResult = var1 OR var2
Print ORResult
rem ORResult = 1.0
```

- **See also**

"Operators priority" on page 17, "AND" on page 25, "XOR" on page 110

2.1.56. PI

- **Syntax [function]**

PI

- **Purpose**

The function returns 3.14159265

2.1.57. PRINT - AT

- **Syntax [Command]**

PRINT CA

- This command displays the text CA followed by a new line.

PRINT CA;

- This command displays the text CA without a new line.

PRINT AT E1, E2 CA

- This command displays the text CA at the E1 column and at the E2 line.

PRINT CA1;CA2[;CA3...]

- Display the CA1, CA2 text etc. one following the other (don't pass to next line).

- **Purpose**

The eWON has a virtual "screen" that can be used in order to inspect the content of values while the program is running, or in order to debug an expression…

- **Example**

```
PRINT " HOP1 "; HOP2 "
```

- **See also**

“CLS” on page 29

2.1.58. PRINT

- **Syntax [Command]**

PRINT #x,CA

- With x defined as follows:
  - 0 = User's web page
  - 1 = Virtual screen
- CA is described in the PRINT - AT section.

- **Purpose**

The PRINT command sends output to the virtual screen.

With the PRINT # command, output can be routed to another destination. When running ASP code, the print command can be used to build the content of the page sent to the user.

If you print to Web page, the Print command add a "
" at the end of line to pass to the next line.

If you don't want to pass to next line, you need to add a ";" (semicolon) at your Print such as : PRINT A$;

Example

```
PRINT #0,A$ REM sends A$ to the user's web page
PRINT #1,A$ REM works like PRINT A$ by sending to the virtual screen.
```

2.1.59. PUT

The put command works completely differently if the file is opened in Binary mode or in Text mode. The file must be opened for OUTPUT or for APPEND operation (APPEND for /usr files only).

- COM, TCP-UDP, /usr

The file syntax has been extended in version 3 of the eWON to allow access to the serial port and to TCP and UDP socket. The command description describes operation for /usr (Text and Binary modes), COM (always binary) and TCP-UDP (always binary)

2.1.59.1. File Syntax[Command] – Binary mode

PUT E1, S1[;S2…]

- E1 is file number (1-8)
- S1 is the string of char to append to the file. The number of bytes written depends on the length of the string.
- S2…: (optional) additional data to write.
  The length of a BASIC line limits the number of items.

- Important -

The delimiter between the file number and the first item is a ',' but the separator between the first item and the optional next item is a ';'. This is close to the PRINT syntax.

- **Example**

```
OPEN "/myfile.bin" FOR BINARY OUTPUT AS 1
PUT 1,"ABCDEF";"GHIJKLMN"
CLOSE 1
REM Now reopens and append
OPEN "/myfile.bin" FOR BINARY APPEND AS 1
PUT 1,"OPQRSTUVWXYZ"
CLOSE 1
```

2.1.59.2. File Syntax[Command] – Text mode

PUT E1, V1[;V2…][;]

- E1 is file number (1-8)
- V1 is an element of type STRING, INTEGER or FLOAT
- V2…(optional): additional data to write (STRING, INTEGER or FLOAT)

The data are converted to text before being written to file. If data is of STRING type it is written between quotes ("), otherwise not. A ';' is inserted between each data written to the file.

If the PUT command line ends with a ';', the sequence of data can continue on another BASIC line. If the PUT command line ends without the ';' character, the line is considered as finished and a CRLF (CHR$(13)+CHR$(10)) is added at the end of the file.

- **Example**

```
OPEN "/myfile.txt" FOR TEXT OUTPUT AS 1
PUT 1,123;"ABC";
PUT 1,"DEF"
PUT 1,345.7;"YYY";"ZZZ"
CLOSE 1
```

The above code produces this file:

```
123;"ABC";"DEF"
345.7;"YYY";"ZZZ"
```

There is a CRLF at the end of the last line. By writing *PUT 1,345.7;"YYY";"ZZZ";* you would have avoided that.

2.1.59.3. COM Syntax[Command] – Binary mode

PUT 1, S1

- S1: string of data to write to serial port

- **Purpose**

Writes the S1 string to the serial port. The function returns only after all the data have been actually sent.

- Important -

The string can contain any byte by using the CHR$ function.
Serial port cannot be used by an IO server in the same time, or it would result to a "IO Error".

2.1.59.4. TCP/UDP Syntax[Command] – Binary mode

PUT E1, S1

- E1: is the file number returned by the OPEN function.
- S1: string of data to write to the socket.

- **Purpose**

Writes the S1 string to the socket

The function returns only after all the data have been actually transferred to the stack.

- Important -

The socket must be opened. The OPEN command returns immediately but generates a scheduled action. The PUT command will generate an IO error until the socket is actually opened (See OPEN on page 40).

When data are transferred to the TCP/IP stack, it does not mean that the data have been received by the socket end point. It may take minutes before the data are considered as undeliverable and the socket is put in error mode.

The string can contain any byte by using the CHR$ function.

- **See also**

"CLOSE" on page 29, "EOF" on page 33, "GET" on page 41, "OPEN" on page 79.

2.1.60. PUTFTP

- **Syntax[command]**

PUTFTP S1,S2 [,S3]

- S1 is the destination file name (to write on the FTPServer)
- S2 is the file content (String)
  This content may be an EXPORT_BLOCK_DESCRIPTOR content.
  See also chapter "Export block descriptor" in the General User Guide.

- S3 (optional) is the FTP server connection parameters.
  If S3 is unused, the FTPServer parameters from the General config page will be used.

- **Purpose**

Put a file on a FTP server, content of the file is either a string or an Export_Bloc_Descriptor.

The S3 parameters is as follow: [user:password@]servername[:port][,option1]

- The option1 parameters is to force PassiveMode, put a value 1 as option1 parameter.
  If omitted, option1=0, then eWON will connect in ActiveMode.

This command posts a scheduled action request for a PUTFTP generation.

When the function returns, the GETSYS PRG,"ACTIONID" returns the ID of the scheduled action and allows tracking this action. It is also possible to program an ONSTATUS action that will be called when the action is finished (with or without success).

- **Example**

```
REM Post a file containing a custom text
PUTFTP "/ewon1/MyFile.txt","this is the text of the file"

REM Post a file containing the event log
PUTFTP "/ewon1/events.txt","[$dtEV]"

REM Post on a defined FTP server, a file with the Histo logging of
Temperature tag
PUTFTP
"/ewon1/Temperature.txt","[dtHL$ftT$tnTemperature]","user:pwd@FTPserver.com"
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69, "ONSTATUS" on page 77.

2.1.61. PUTHTTP

- **Syntax [Command]**

PUTHTTP S1,S2,S3,S4,S5 [,S6]

- S1: Connexion Parameter
  with the format [User:Password@]ServerName[:Port]
- S2: URI of the action (absolute path of the request URI)
- S3: Text fields
  with the format [FieldName1=ValueName1][&FieldNameX=ValueNameX]*
- S4: File fields
  with the format [FieldName1=ExportBlockDescriptor1]

[&FieldNameX=ExportBlockDescriptorX]*

- S5: Error String
- S6 (Optional): "PROXY"

What you should take note of:

- In the preceding syntax description the square brackets are used to define an optional section for a given parameter. The * is used to indicate that the preceding optional section may be repeated 0 to n times.
- All the parameters are mandatory. If you don't need to post Text fields, just write an empty string for the S3 parameter
- The HTTP Server response sent back will be checked against the Error String. If the Response contains the Error String the command will finish without success.
- Spaces in Text fields and File fields strings are not allowed except inside export block descriptors (inside the EBD brackets).
- One fieldname=valuename section in the text field parameter may not exceed 7500 bytes (Otherwise action will finish without success). This limitation does not apply for the file fields.

- **Purpose**

The PUTHTTP command submit an HTTP form to a Web server (like you do when you answer a Web form).

The submitted forms may contain text fields and file fields.

The HTTP method used is the POST method (multipart/form-data). Content Type of the file fields is always application/octet-stream. Files to upload are defined using the Export Block descriptor syntax (See also Export Block descriptor section in the eWON reference guide).

When the function returns, the GETSYS PRG, returns the ID of the scheduled action and allows tracking of this action. It is also possible to program an ONSTATUS action that will be called when the action is finished (with or without success).

When "PROXY" is added at the end of the command, the eWON will perform the PUTHTTP through a Proxy server. The eWON will use the Proxy server parameters configured in System Setup / Communication / VPN Global.

- Important -

The posting method used (chunked packets) is only correctly handled on IIS 6.0 and Apache Webservers. Posting on IIS 5 doesn't work (Windows XP). Chuncked packets are not applied when the "PROXY" parameter is used because most Proxy servers do not accept them.

If PUTHTTP is used with the "PROXY" parameter, then eWON creates a temporary file named "puthttp.proxy" in the /usr directory to store the data locally before sending it towards the server via the Proxy.

- **Example**

Textfields form without HTTP basic authentfication:

```
PUTHTTP
"10.0.5.33","/textfields.php","firstname=jack&lastname=nicholson","","failed"
```

When file fields are not needed an empty string is used for parameter S4. When no port is specified HTPP port 80 is used.

Text fields with basic authentification and configured HTTP port:

```
PUTHTTP
"adm1:adm2@www.ewon.biz:89","/textfields.php","fname=jack&lname=nicholson",""
,"failed"
```

HTTP server is supposed to listen on port 89 at address www.ewon.biz adm1 is used as login and adm2 is used as password.

Text fields + file fields:

```
PUTHTTP "10.0.5.33", "/upload.php","firstname=bob&lastname=nicholson",
"pictures[]=[$dtEV $fnevents.txt]&pictures[]=[$dt CF$fnconfig.txt]","failed"
```

$fn (file name) directive is optional but when not used $dtCF will be used as destination file name

Textfields form without HTTP basic authentfication through a Proxy server:

```
PUTHTTP "10.0.5.33","/
textfields.php","firstname=jack&lastname=nicholson","","failed","PROXY"
```

- **See also**

"ONSTATUS" on page 77, "GETSYS, SETSYS" on page 47, "GETHTTP" on page 45, see Export Block Descriptor on General Reference Manual

2.1.62. REBOOT

- **Syntax [Command]**

REBOOT

- **Purpose**

This Basic keyword provides a very easy way to reboot eWON.

A typical use of this command is by simply entering it into a file you name "remote.bas" then saving locally and uploading this file on the eWON FTP site to replace the existing remote.bas file. eWON then directly reboots.

2.1.63. REM

- **Syntax [command]**

REM free text

- **Purpose**

This command enables the insertion of a line of comment in the program. The interpreter does not consider the line.

- **Example**

```
PRINT a%
REM we can put whatever comment we want here
a%=2: REM Set a% to 2
```

- **See also**

“// (comment)” on page 24

2.1.64. RENAME

- **Syntax [Command]**

RENAME S1,S2

- S1, S2 are string

- **Purpose**

Change the name of file S1 to S2. The command only works in the “/usr” directory. Omitting "/usr/" before the filename will result to a I/O error.

The file and directory names are case sensitive. The directory must exist before the call of the function. There is no automatic directory creation.

- **Example**

```
RENAME "/usr/OldName.txt","/usr/NewName.txt"
```

- **See also**

"ERASE" on page 34

2.1.65. RTRIM

- **Syntax[Command]**

RTRIM S1

- S1 is a string

- **Purpose**

RTRIM returns a copy of a string with the rightmost spaces removed.

- **Example**

```
b$ = RTRIM a$
```

- **See also**

"LTRIM" on page 67

2.1.66. SENDMAIL

- **Syntax[command]**

SENDMAIL S1,S2,S3,S4

- S1 is the E-mail address of the recipients (TO). Multiple recipients can be entered separated by ';'.
- S2 is the E-mail address of the recipient Carbon Copies (CC). Multiple recipients can be entered separated by ';'.
- S3 is the subject of the message.
- S4 is the content of the message.

- **Purpose**

This command posts a scheduled action request for an Email generation. When the function returns, the GETSYS PRG,"ACTIONID" returns the ID of the scheduled action and allows tracking this action.

It is also possible to program an ONSTATUS action that will be called when the action is finished (with or without success).

The S4 message content follows a special syntax that allows sending attachments and inserting Export data inside the content itself (See also chapter "Export block descriptor" in the General User Guide).

The content field (S4) syntax can content any number of [EXPORT_BLOCK_DESCRIPTOR], these blocks will be replaced by their actual content.

- **Example**

```
S4 = "Event Log data [$dtEV] And a real time table: [$dtRL $ftT $tnMyTag]"
Rem will generate an Email with [$dtEV] and [$dtRL…] replaced by the actual
data.
```

If instead of putting [EXPORT_BLOCK_DESCRIPTOR] you put &[EXPORT_BLOCK_DESCRIPTOR], then the same data is attached to the Email.

The position in the S4 field where the &[..] is placed does not matter, the attachment &[…] descriptor will NOT appear in the content itself, but will produce the given attachment.

- **Example**

```
M$ = "Event Log data are attached to this mail &[$dtEV]"
Rem will generate an Email with "Event Log data are attached to this mail "
as content and an attachment with the events log.

SENDMAIL "ewon@actl.be", "", "Subject", "Message"
SENDMAIL "ewon@actl.be", "", "Subject", M$
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69, "ONSTATUS" on page 77.

2.1.67. SENDSMS

- **Syntax[command]**

SENDSMS S1,S2

- S1 is the SMS recipients list.
  Please refer to chapter "SMS on alarm configuration" in the General User Guide, for syntax of this field.
- S2 is the content of the message (maximum 140 characters).

- **Purpose**

This command posts a scheduled action request for an SMS generation.

When the function returns, the GETSYS PRG,"ACTIONID" returns the ID of the scheduled action and allows tracking this action. It is also possible to program an ONSTATUS action that will be called when the action is finished (with or without success).

- **Example**

```
REM send an SMS to 2 recipients.
D$ = "0407886633,ucp,0475161622,proximus"
D$ = D$ + ";" + "0407886634,ucp,0475161622,proximus"
SENDSMS D$, "Message from eWON"
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69, "ONSTATUS" on page 77.

2.1.68. SENDTRAP

- **Syntax[command]**

SENDTRAP I1,S1

- I1 is the first trap parameter (INTEGER)
- S1 is the second trap parameter (STRING)

- **Purpose**

This command posts a scheduled action request for an SNMP TRAP generation.

The first parameter is sent on OID .1.3.6.1.4.1.8284.2.1.4.2

The second parameter is sent in OID .1.3.6.1.4.1.8284.2.1.4.1

```
-
-- Script information
-- ewonScript OBJECT IDENTIFIER ::= { prodEwon 4 }

scpUserNotif OBJECT-TYPE
SYNTAX DisplayString (SIZE (0..255))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
```

```
"This is the text of the last trap sent by the Script"
::= { ewonScript 1 }

scpUserNotifI
OBJECT-TYPE SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This is a free parameters for script generated traps"
::= { ewonScript 2 }
```

When the function returns, the GETSYS PRG,"ACTIONID" returns the ID of the scheduled action and allows tracking this action. It is also possible to program an ONSTATUS action that will be called when the action is finished (with or without success).

- **Example**

```
REM send a trap with NotifI = 10 and Notif = Trap message
SENDTRAP 10,"Trap message"
```

- **See also**

"GETSYS, SETSYS" on page 47, "ONxxxxxx" on page 69, "ONSTATUS" on page 77.

2.1.69. SETIO

- **Syntax [command]**

SETIO TagRef, F1

- TagRef is the Tag reference (TagName, ID or -Index) See Tag Access on page 10
- F1 is the value to give to the Tag.

- **Purpose**

Modifies the value of a Tag. The Tag must be writable (not for the read-only Tags).

- Note -

In many cases this function is efficiently replaced by the TagName@ syntax. For example SETIO "MyTag", 10.2 is equivalent to MyTag@=10.2

- **Example**

```
SETIO "MYTAG", 10.123
```

2.1.70. SETTIME

- **Syntax [Command]**

SETTIME S1

- S1 is the new date / time to set.
- S1 can contain only the time. In that case the date is not modified.
- S1 can contain only a date. In that case the time is set to 00:00:00

- **Purpose**

Updates the eWON's real time clock.

- Note -

An event is generated in the events log.

- **Example**

```
REM The following are valid time updates
SETTIME "1/1/2000": REM Time is set to 01/01/2000 00:00:00
SETTIME "01/12/2000 12:00": REM Time is set to 01/12/2000 12:00:00
PRINT TIME$: REM suppose it returns "15/01/2000 07:38:04"
SETTIME "12:00": REM Time is set to 15/01/2000 12:00:00
```

- **See also**

"TIME$" on page 104

2.1.71. SFMT

- **Syntax [Command]**

SFMT Item,EType[,ESize,SFormat]

- Item is the number (Integer or Float) to format into string.
- EType is the parameter determining the type of conversion.

- ESize is the size of the output string as formated.
- SFormat is the format specifier for the conversion.

- **Purpose**

Converts a number (float or integer) to a formated string. The type of conversion is determined by the EType parameter.

Etype value	Conversion Type
1	Convert float number to string (MSB first)
2	Convert float number to string (LSB first)
10	Convert integer to string (MSB first)
11	Convert integer to string (LSB first)
20	Format float number using a SFormat specifier
30	Format integer number using a SFormat specifier
40	Format time as Integer into time as String

Table 28: SFMT function -Etype value and conversion

If ESize is equal to 0 (or negative) with a SFormat present, then ESize is the size of the output string as formated.

If ESize is positive, SFMT will produce a string of ESize bytes.

- **See also**

"FCNV" on page 35

2.1.71.1. Convert float to IEEE float representation

The IEEE float representation use four bytes (32 bits).

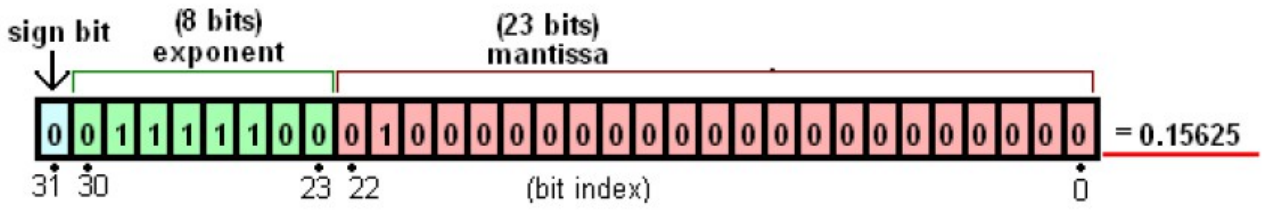

Illustration 3: Conversion to an IEEE float

EType = 1 or 2

The string could be LSB first:

```
A$ = SFMT FloatNum, 1
```

This will convert FloatNum to a string holding the IEEE representation with MSB first

```
A$(1) = MSB (Exponent+ Sign)
…
A$(4) = LSB (Mantissa LSB)
```

or MSB first:

```
A$ = SFMT FloatNum, 2
```

This will convert FloatNum to a string holding the IEEE representation with LSB first

```
A$(1) = LSB (Mantissa LSB)
...
A$(4) = MSB (Exponent+ Sign)
```

- **Example**

```
ieee = -63.456
A$ = SFMT ieee,1
rem a$(1)=194 a$(2)=125 a$(3)=210 as(4)=242

A$ = SFMT ieee,2
rem a$(1)=242 a$(2)=210 a$(3)=125 as(4)=194
```

2.1.71.2. Convert integer to string

Convert an integer value to a string holding the bytes array representation of this integer.

This representation can be MSB (Most Significant Byte) first or LSB (Least Significant Byte) first.

EType = 10 or 11

The **ESize parameter is required**. It is the size of the returned string (it can be 1, 2, 3 or 4)

- **Example**

```
a% = 1534
A$ = SFMT a%,10,4
rem a$(1)=0 a$(2)=0 a$(3)=5 as(4)=254

A$ = SFMT a%,11,4
rem a$(1)=254 a$(2)=5 a$(3)=0 as(4)=0
```

2.1.71.3. Convert a float to a string using a SFormat specifier

Convert a float number (MyVal=164.25) to a String using a Format specifier.

EType = 20

The **ESize parameter is required**. It is the size of the returned string (use 0 to let eWON set the length).

The **SFormat parameter is required**. It is the format specifier string and is like "%f" or "%.5g".

The syntax for the float format specifier is "**%[flags][width][.precision]type**":

Type (required)	**'f', 'F'** : Print a float in normal (fixed-point) notation. **'e', 'E'** : Print a float in standard form ([-]d.ddd e[+/-]ddd). **'g', 'G'** : Print a float in either normal or exponential notation, whichever is more appropriate for its magnitude. 'g' uses lower-case letters. 'G' uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
Flags (optional)	**'+'** : always denote the sign '+' or '-' of a number (the default is to omit the sign for positive numbers). **'0'** : use 0 to left pad the number.
Width (optional)	**number** : set the length of the whole string for padding. Only needed when flag 0 is used.
.precision (optional)	**number** : the decimal portion of the output will be expressed in at least number digits.

Table 29: SFMT – Conversion from float to string using Sformat - 1

- **Example**

```
MyVal = 164.25
A$ = SFMT MyVal,20,0,"%f"
rem a$="164.250000"
A$ = SFMT MyVal,20,0,"%012.3f"
```

```
rem a$="00000164.250"
A$ = SFMT MyVal,20,0,"%e"
rem a$ = "1.642500e+02"
```

2.1.71.4. Convert an integer to a string using a SFormat specifier

Convert an integer number (a% = 1935) to a String using a Format specifier.

EType = 30

The **ESize parameter is required**. It is the size of the returned string (use 0 to let eWON set the length).

The **SFormat parameter is required**. It is the format specifier string and is like "%d" or "%o".

The syntax for the float format specifier is "**%[flags][width]type**".

Type (required)	**'d'** : convert into integer notation. **'o'** : convert into Octal notation. **'x' or 'X'** : convert into Hexadecimal notation (lowercase or uppercase)
Flags (optional)	**'+'** : always denote the sign '+' or '-' of a number (the default is to omit the sign for positive numbers). **'0'** : use 0 to left pad the number.
Width (optional)	**number** : set the length of the whole string for padding. Only needed when flag 0 is used.

Table 30: SFMT – Conversion from float to string using Sformat - 2

- **Example**

```
a% = 2568
A$ = SFMT a%,30,0,"%010d"
rem a$="0000002568"
A$ = SFMT a%,30,0,"%o"
rem a$="5010" OCTAL notation
A$ = SFMT a%,30,0,"%X" rem a$ = "A08"
```

2.1.71.5. Convert time as Integer into time as String

Convert an Integer holding the number of seconds since 01/01/1970 00:00:00 into a String holding a time in the format “dd/mm/yyyy hh:mm:ss” (ex: “28/02/2007 16:48:22”).

EType = 40

SFMT TimeAsInt,40

The TimeAsInt must be an Integer. If not, the function will return a syntax error.

If a float parameter is passed, it must be converted to an integer value first (See INT on page 61)

- Important -

Float value have not enough precision to hold the big numbers used to represent seconds since 1/1/1970, this leads to lost of precision during time conversion.

- **Example**

```
A$ = SFMT 0,40
rem a$="01/01/1970 00:00:00"
a% = 1000000000
A$ = SFMT a%,40
rem a$="09/09/2001 01:46:40"
```

2.1.72. SGN

- **Syntax [function]**

SGN F1

- **Purpose**

Returns the sign of F1:

- If F1 is > 0, the function returns 1.
- If F1 = 0, the function returns 0.
- If F1 is < 0, the function returns -1.

- **Example**

```
SGN (-10)
REM returns -1
SGN (-10.6)
REM returns -1
SGN 10
REM returns 1
```

2.1.73. SQRT

- **Syntax [function]**

SQRT F1

- **Purpose**

Returns the square root of F1.

- **Example**

```
SQRT 16 :REM returns 4
```

2.1.74. STR$

- **Syntax [function]**

STR$ F1/E1

- **Purpose**

The function returns the character string related to an E1 or F1 number.

- **Example**

```
a%=48
a$= STR$ a%
REM A$ is worth " 48 " after this affectation
```

- **See also**

"VAL" on page 106

2.1.75. TIME$

- **Syntax[function]**

TIME$

- **Purpose**

Returns the string with the current date and time. The output format is "dd/mm/yyyy hh:mm:ss" (ex: "25/10/2004 15:45:55")

The number of characters in the returned string is constant.

- Note -

The GETSYS command provides a mean to return the current time as a number of seconds since 1/1/1970.
The SFMT and FCNV functions allow you to convert between TimeString and TimeInteger.

- **Example**

```
PRINT TIME$
```

- **See also**

"SETTIME" on page 98, See "FCNV" on page 35, See "SFMT" on page 98

2.1.76. TGET

- **Syntax[function]**

TGET E1

- E1 is the number of the timer (1 to 4).

- **Purpose**

Returns N (>0) if the timer expires and then resets the value (N is the number of times the timer has expired). Returns '0' if the timer did not expired since the last call to TGET.

- **Example**

```
REM timer 1 minute
TSET 1,60
Label1:
IF NOT TGET 1 THEN GOTO LABEL1
```

- **See also**

"ONTIMER" on page 77, "TSET" on page 105.

2.1.77. TSET

- **Syntax[Command]**

TSET E1, E2

- E1 is the number of the timer (1 to 4).
- E2 is the value in seconds of the timer.

- **Purpose**

Initializes the timer E1 at an E2 time base (in second). The timer is read by TGET.

- **Example**

```
REM timer 1 minute
TSET 1,60
Label1:
IF NOT TGET 1 THEN GOTO LABEL1
```

To stop a timer, you must put the value 0:

```
TSET 1,0
```

- **See also**

"ONTIMER" on page 77, "TGET" on page 105.

2.1.78. TYPE$

- **Syntax**

TYPE$(S1)

- S1 is the name of the variable. The returned value will be "String", "Float" or "Integer"

2.1.79. VAL

- **Syntax [function]**

VAL S1

- **Purpose**

The function evaluates the character string and returns the corresponding expression.

- Note -

VAL is a function that usually takes an expression and returns a Real after expression

evaluation. This VAL function can also evaluate an expression that returns a string.

- **Example**

```
a$= "12"
a% = VAL (" 10"+ a$)
REM a% equal 1012
a$="abc"
b$="efg"
c$=val("a$+b$")
REM c$ equal "abcefg"
```

- **See also**

"STR$" on page104.

2.1.80. WAIT

- **Syntax [function]**

WAIT N1,S[,N2]

- N1 is the File number to wait on.
- S is the operation to execute (max 255 char)
- N2 is the timeout in sec (if omitted, the default is 60 sec)

- **Purpose**

The WAIT command is used to monitor events on files.

Currently the events monitored are:

- Data received on TCP and UDP socket

Wait for data available on N1 (or TimeOut) and then execute the S operation (ex: "goto DataReceived").

The WAIT function will register a request to wait for the event, it will not block until the event occurs.

When the WAIT function calls the operation, it will preset the EVTINFO (see Getsys PRG,"EvtInfo"), with the result of the operation:

EVTINFO	Signification
> 0	The event occurred and read can follow: • =1: Read is pending • =2: Ready for Write • =3: Ready for Write and Read is pending **Important** : If Read is pending, then the A$=Get N1 function will be used, in case the Get function returns an empty string, it means that there is an error on the socket (either the socket was closed by the other party or the socket is not writable), in that case, the file should be closed because it is not more valid.
-1	The wait operation was aborted because of an error on the file monitored (for example the file was closed).
-2	The condition was not met during the wait operation (TimeOut).

Table 31: VAL – EVTINFO values

You can have a maximum of 4 WAIT command pending at the same time.

If a WAIT command is pending on a file and another WAIT command is issued on the same file, an "IO Error" error will occur.

- **Example**

The following example concerns TCP socket. It connects to a server running the ECHO protocol.

```
Tw:
Cls
Close 1
OPEN "tcp:10.0.100.1:7" FOR BINARY OUTPUT AS 1
o%=0

wo:
a% = Getsys Prg,"actionstat"
IF a%=-1 Then Goto wo
Put 1,"msg_start"
Wait 1,"goto rx_data"
End

rx_data:
a%=Getsys Prg,"evtinfo"
IF (a%>0) Then
    Print "info:";a%
    a$=Get 1
    Print a$
```

```
    Put 1,"abc"+Str$(o%)
    o%=o%+1
    Wait 1,"goto rx_data"
ELSE
Print "error:";a%
ENDIF
```

2.1.81. WOY

- **Syntax [Function]**

WOY E1 / S1

- E1 is a date in integer format (number of seconds since 1/1/1970)
- S1 is a date in String format ("18/09/2003 15:45:30")

- **Purpose**

This function returns an integer corresponding to the ISO8601 Week-Of-Year number that matches a specified time variable.

- Note -

Do not call the function with a float variable of value (or this would result to error "invalid parameter").

- **Example**

```
a$ = TIME$
a% = WOY a$
```

```
b% = getsys prg,"TIMESEC"
a% = WOY b%
```

- **See also**

“DAY” on page 30 “DOW” on page 31, “DOY” on page 32, “MONTH” on page 68

2.1.82. WriteEBD

This command is available starting firmware v12.

- **Syntax [Function]**

WriteEBD S1, S2

- S1 is an Export Block Descriptor (EBD) in a String format
- S2 is the file path the EBD content will be streamed in

This command return an action ID.

- **Purpose**

This command streams an Export Block Descriptor (EBD) to the filesystem using a scheduled action.

- **Example**

```
WriteEBD "$dtEV", "/usr/myEvent.txt"
```

2.1.83. XOR

- **Syntax [Operator]**

E1 XOR E2

- **Purpose**

This command returns the bitwise XOR comparison of E1 and E2.

a XOR b returns 1 if a if true or if b is true, but NOT IF both of them are true.

- **Example**

```
1 XOR 2 returns 3
2 XOR 2 returns 0
```

- **See also**

"Operators priority" on page 68, "AND" on page 25, "OR" on page 84.

3. Debug a BASIC program

With the BASIC IDE comes an integrated console.

This means that you can perform the debugging directly within your code.

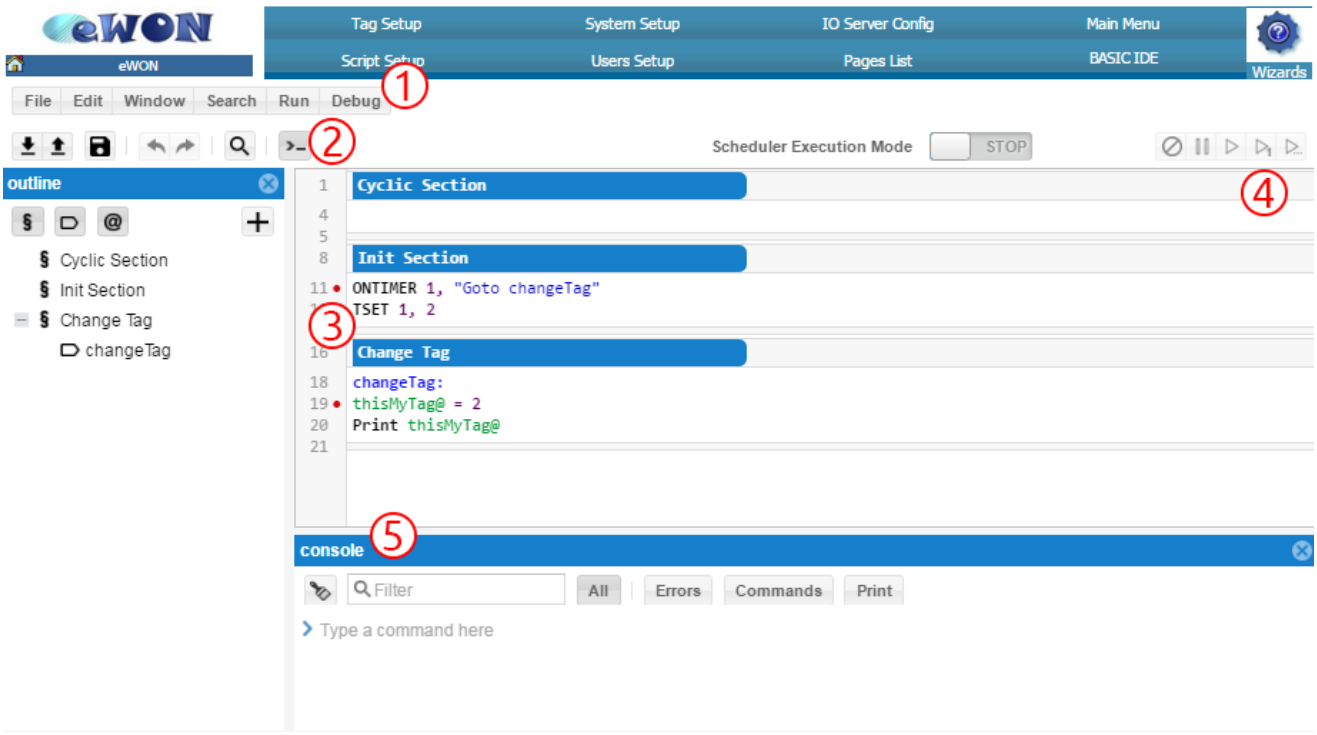

Illustration 4: BASIC IDE window

In the above picture, you'll find :

Number 1

This is the general menu for the debugging. You'll be able to

- Pause, Continue and Abort
- Perform step by step action
- Remove all breakpoints

Number 2

This icon shows / hides the console frame (Number 5)

Number 3

Manually point out the line you want the debugger to stop on.

Number 4

Control the flow of the BASIC script. Thanks to this flow menu, you will be able to Play/Resume, Pause, Perform step by step action.

Number 5

The console frame allows you to perform more advanced actions such as:

- sort the different types of log (error, command or print)
- see the result of functions, commands, …
- manually trigger actions such as functions, label, ...

4. BASIC Error Codes

These codes are returned in ONERROR:

Error code	Error name
0	syntax error
1	'(or)' expected
2	no expression present
3	'=' expected
4	not a variable
5	invalid parameter
6	duplicate label
7	undefined label
8	THEN expected
9	TO expected
10	too many nested FOR loops
11	NEXT without FOR
12	too many nested GOSUBs
13	RETURN without GOSUB
14	Out of memory
15	invalid var name
16	variable not found
17	unknown operator
18	mixed string&num operation
19	Dim index error
20	',' expected
21	Number expected
22	Invalid assignment
23	Quote too long
24	Var or keyword too long
25	No more data
26	reenter timer
27	label not found

Error code	Error name
28	Operation failed
29	ENDIF expected
30	ENDIF without IF
31	ELSE without IF
32	Math error
33	IO Error
34	End of file
35	val in val

Table 32: Basic Error Codes

5. Configuration fields

This section describes the fields found in the config.txt file.

All the fields are readable and writable using GETSYS and SETSYS(unless otherwise specified). The file is separated in the several sections (System, UserList and TagList).

One of the three sections must first be loaded with the "SETSYS SYS, xxx" command, where xxx is one of SYS, USER or TAG.

- **Example**

```
Setting the eWON Identification parameter and printing the Information
(parameter = Identification, Information)
SETSYS SYS, "LOAD"
SETSYS SYS, "Identification", "10.0.0.53"
PRINT GETSYS SYS, "Information"
SETSYS SYS, "SAVE"
```

5.1. SYS Config

The following table describes the fields accessible from the system configuration.

The last column gives the ewon configuration web page where the parameter appears. The web pages are found under Configuration.

Name	Description	Web page
Identification	Identification of the eWON (appears on the logon web page logon below the logo)	System Setup/General/General/Identification
Information	Complementary information about the eWON	System Setup/General/General/Identification
SmtpServerPort	SMTP server port	System Setup/General/General/
SmtpServerAddr	SMPT server address	System Setup/General/General/
SmtpUserName	SMTP user name	System Setup/General/General/
AlRetrigInt	Interval after which an alarm will be re triggered if the condition is still true (only for alarms that have not been	System Setup/General/General/
NtpEnable	1 if NTP service is enabled, 0 otherwise	System

Name	Description	Web page
		Setup/General/General/
NtpServerAddr	NTP Server address as a chain of char	System Setup/General/General/
NtpServerPort	NTP server port	System Setup/General/General/
NtpInterval	Interval between NTP connections	System Setup/General/General/
PrgAutorun	1 if script starts at eWON boot time. See script control page	Script Setup
FormatRequest	1 if a format has been requested, 0 otherwise	System Setup/General/General/
MbsBaudRate	Modbus baud rate. 0 if disabled, positive value otherwise	IO Server Config > Modbus
Mbs2StopBit	1 if Modbus IO server uses two stop bits, 0 if it uses 1 stop bit	IO Server Config > Modbus
MbsParity	0 for none, 1 for even, 2 for odd	IO Server Config > Modbus
MbsReplyTO	Modbus reply time out	IO Server Config > Modbus
MbsPR(x)	X = = 1..3, Modbus topic 1..3 polling rate (expressed in Msec)	IO Server Config > Modbus
TimeZoneOffset	Time zone (expressed in seconds)	System Setup/General/General
MbsAddress	Modbus address	IO Server Config > Modbus
MbsSlaveEn	1 if Modbus slave mode enabled	IO Server Config > Modbus
DecSeparator	Decimal separator: 44 = "," 46 = "."	
Page(x)	x = 1..11, User Page as defined in the Page Lists config page	Pages List
IOSrv(x)	x = 0..9	
IOSrvData(x)	x = 0..9	"Corresponding IO Server Config"
SecureUsr	1 to Enable user security pages	System Setup/General/General
HomePage	User defined home page	System Setup/General/General
MbsSMB(x)	x=1..3, Modbus Topic x	
MbsSIP(x)	x=1..3, Modbus Topic x IP address	IO Server Config □ Modbus

Name	Description	Web page
FtpServerPort	FTP Server port	System Setup/General/General
FtpServerAddr	FTP server address	System Setup/General/General
FtpUserName	FTP login name	System Setup/General/General
FtpPassword	FTP password	System Setup/General/General
SmtpAllowB64		
MbsEn(x)	x=1..3, Modbus Topic x enabled (1 if enabled, 0 otherwise)	IO Server Config □ Modbus
FTPC_SDTO		
FTPC_SCTO		
FTPC_ACTO		
FTPC_RDTO		
DNS_SRTO		
SnmpCom(x)	x=1..5, SNMP Community x	System Setup/General/SNMP
SnmpR	x=1..5, SNMP Community x Read enabled	System Setup/General/SNMP
SnmpW	x=1..5, SNMP Community x Write enabled	System Setup/General/SNMP
SnmpAlwAll	Accepts SNMP packet from any host (1 = enabled)	System Setup/General/SNMP
SnmpHIp(x)	x=1..5, SNMP Host x IP Address	System Setup/General/SNMP
SnmpHCom(x)	x=1..5, SNMP Host x Community	System Setup/General/SNMP
SnmpHTrap(x)	x=1..5, SNMP Host x Trap enabled	System Setup/General/SNMP
SnmpHAlw(x)	x=1..5, SNMP Host x access allowed	System Setup/General/SNMP
MbsBits*	Modbus Bits (7 or 8)	N/A
AlMaxTry	Number of times an action is retried in case of error	System Setup/General/General
AlRetryInt	Interval between action trials in case of error	System Setup/General/General

Table 33: System Configuration fields

The MbsBits parameter specifies how to Modbus IO Server will read the bytes. The possible values are 7 and 8 bits/byte.

5.2. COM Section

This section describes the fields found in the comcfg.txt file. All the fields are readable and writable (unless otherwise specified) using GETSYS and SETSYS with the COM parameter.

- **Example**

Setting the First ISP phone number (parameter = PPPClPhone1) to number 0123456789:

```
SETSYS COM, "LOAD"
SETSYS COM, "PPPClPhone1","0123456789"
SETSYS COM, "SAVE"
```

The following table describes the fields accessible from the communication configuration. The last column gives the ewon configuration web page where the parameter appears.

The web pages are found under System Setup.

Name	Description	Web page
EthIp	Ethernet IP address	Communication/Ethernet
EthMask	Ethernet IP Mask	Communication/Ethernet
EthGW	Ethernet Gateway	Communication/Ethernet
ModemInitStr	Modem Initialization String	Communication/Modem
PPPServerIP	PPP Server IP Address	Communication/Dial UP (PPP)
PPPServerMask	PPP Server IP Mask	Communication/Dial UP (PPP)
PPPServerGW	PPP Server Gateway	Communication/Dial UP (PPP)
PPPClientIp	PPP Client IP Address	Communication/Dial UP (PPP)
PPPClCompress	Enable PPP Client Compression (enabled =1)	Communication/Dial UP (PPP)
PPPClPhone1	ISP1 Phone number	Communication/Dial UP (PPP)
PPPClUserName1	ISP1 User Name	Communication/Dial UP (PPP)
PPPClPassword1	ISP1 Password	Communication/Dial UP (PPP)
PIN	PIN code (for GSM modem usage only)	Communication/Modem

Name	Description	Web page
RTEnIpFwrd	Enable IP forwarding (1 = enabled)	Communication/Router (Filter)
DialInOut	Enable Dial in / out / both (1 / 2 / 3)	Communication/Dial UP (PPP)
InEqualOut	Enable Usage of dial in connection to dial out (enabled = 1)	Communication/Dial UP (PPP)
DialTO	Dial out timeout	Communication/Dial UP (PPP)
ClIdle	Client mode idle timeout before hang up	Communication/Dial UP (PPP)
SrvIdle	Server mode idle timeout before hang up	Communication/Dial UP (PPP)
EthDns1	Ethernet DNS 1 IP Address	Communication/Ethernet
EthDns2	Ethernet DNS 2 IP Address	Communication/Ethernet
PPPSrvCompress	Enable PPP Server compression (enabled = 1)	Communication/Dial UP (PPP)
PPPClNeedChap	Enable CHAP authentication requirement (enabled = 1)	Communication/Dial UP (PPP)
PPPClPhone2	ISP2 Phone number	Communication/Dial UP (PPP)
PPPClUserName2	ISP2 User Name	Communication/Dial UP (PPP)
PPPClPassword2	ISP2 Password	Communication/Dial UP (PPP)
CallAlloc	Allocated Budget	Communication/Dial UP (PPP)
CallAllocRst	Budget Reset Period	Communication/Dial UP (PPP)
CBEnabled	Enable callback (enabled = 1)	Communication/Callback
CBDelay	Delay after rings before callback (in seconds)	Communication/Callback
CBIdleTime	Callback mode idle timeout before hang up	Communication/Callback
CBPubEMail	Email address where to send the IP address when callback	Communication/Callback
CBDDnsType	Dynamic DNS Type	Communication/Callback

Name	Description	Web page
CBDDnsUName	Dynamic DNS User Name	Communication/Callback
CBDDnsPass	Dynamic DNS Password	Communication/Callback
CBDDnsHName	Dynamic DNS Host Name	Communication/Callback
CBDDnsDName	Dynamic DNS Domain Name	Communication/Callback
CBType	Callback type (0 = Callback on ring, 1 = Callback on User's Request)	Communication/Callback
CBNbRing	Minimal number of rings to detect callback	Communication/Callback
CBTo	ISP to use when calling back (1 / 2)	
RTEnTransFw	Enable transparent forwarding (enabled = 1)	Communication/Router (Filter)
RTEnAuthRt	Enable user authentication when forwarding (enabled = 1)	Communication/Router (Filter)
RTEnableNat	Enable Network Address Translation (enabled = 1)	Communication/Router (Filter)
ModDetCnt	Number of time the eWON tries to detects the modem in case of error (default = 1)	N/A
ModExpType*	Expected modem type (default = -1)	N/A
ModFrcType*	Forced modem type (default = -1)	N/A
SSAM**	Server Access Selection Mode	N/A
CBNbRingOH	Number of rings more than the minimal for callback	Communication/Callback

Table 34: Communication Configuration fields

* The following table describes the possible modem type values

Type	Value
Not used	-1
No modem	0
14400 baud	1
33600 baud	2
56600	3
ISDN	4

Type	Value
Unknown	5
Wavecom Wismo Q2403 GMS/GPRS	0X83 Modem type can be found in the eWON Information page you open by clicking on the eWON Logo. In the above case: "Internal BIBAND GSM (131)"

Table 35: Modem type values

** The following table describes the SSAM possible values:

Description	Value
The last server that worked will be used for next call	-1
Return to first	0 (default)
Always use server 1	1
Always use server 2	2

Table 36: SSAM values

5.3. TAG Section

This section describes the configuration fields for a single Tag. The fields are readable and writable using GETSYS and SETSYS with the TAG parameters and the Tag name.

- **Example**

Printing the alarm status and setting the value (to 45) of the Tag "testTag".

```
SETSYS TAG, "LOAD", "testTag"
PRINT GETSYS TAG, "alstat"
SETSYS TAG, "TAGVALUE", 45
SETSYS TAG,"DoSetVal",1
SETSYS TAG, "SAVE"
```

The following table describes the fields accessible from the Tag configuration. The web pages are found under Tag Setup/Tag Name.

Name	Description
Id	Tag id. Not editable through the web page (only for program usage)
Name	Tag name
Description	Tag description

Name	Description
ServerName	IO Server the Tag gets the value from
TopicName	Topic the Tag takes its basic configuration from
Address	Tag address
Coef	Tag value multiplier coefficient
Offset	Tag value offset
LogEnabled	Enabled Tag value logging (enabled = 1)
AlEnabled	Enable Tag alarm (enabled = 1)
Type	0 = Boolean, 1 = analog
AlBool	Boolean Tag alarm level
MemTag	Is memory Tag (1 = memory Tag, 0 = other)
MbsTcpEnabled	Modbus TCP Enable
MbsTcpFloat	Consider as float value (2 subsequent registers)
SnmpEnabled	Enable SNMP (enabled = 1)
RTLogEnabled	Enable real time logging (enabled = 1)
AlAutoAck	Enable alarm auto-acknowledging (enabled = 1)
ForceRO	Force read-only Tag
SnmpOID	SNMP OID
AlHint	Alarm hint
AlHigh	Alarm high level (warning level)
AlLow	Alarm low level (warning level)
AlTimeDB	Alarm interval deadband
AlLevelDB	Alarm level deadband
PageId	Page the Tag is published on
RTLogWindow	Real-time logging time span
RTLogTimer	Real-time logging interval
LogDB	Historical logging deadband
LogTimer	Historical logging interval
AlLoLo	Alarm low-low level (danger level)
AlHiHi	Alarm high-high level (danger level)
MbsTcpRegister	Enabled access to the Tag as a Modbus

Name	Description	
	register (enabled = 1)	
MbsTcpCoef	Tag value Modbus TCP publishing multiplier coefficient	
MbsTcpOffset	Tag value Modbus TCP publishing offset	
EEN*	Enable Email	alarm notification config
ETO	Email alarm recipient(s) (coma separated)	alarm notification config
ECC	Email alarm carbon-copy recipient(s)	alarm notification config
ESU	Email alarm subject	alarm notification config
EAT	Email alarm attachment (as Export Block Descriptor)	alarm notification config
ESH	Enable Email sent as SMS (enabled = A)	alarm notification config
SEN*	Enable SMS	alarm notification config
STO	SMS alarm recipient	alarm notification config
SSU	SMS alarm subject	alarm notification config
TEN*	Enable trap (SNMP)	alarm notification config
TSU*	Trap (SNMP) subject	alarm notification config
FEN*	Enable FTP	alarm notification config
FFN	FTP destination file name	alarm notification config
FCO	FTP file content (as Export Block Descriptor)	alarm notification config
AlStat	Alarm status (0 = no alarm, 1 = in alarm)	View IO page
ChangeTime	(ReadOnly) Last change time	View IO page
TagValue	(ReadOnly) Tag current value	View IO page
TagQuality	(ReadOnly) Quality of the Tag	View IO page
AlType	(ReadOnly) Alarm Status of the Tag	View IO page
DoDelete	(WriteOnly) Delete the Tag (0 = do not delete, 1 = delete)	
DoAck	(WriteOnly) Acknowledge the Tag (0 = do not acknowledge, 1 = acknowledge)	
DoSetVal	(WriteOnly) Set to 1 to be able to modify TagValue	

Table 37: TAG Configuration fields

5.3.1. Send on alarm notification patterns

In the table below are listed the different pattern values you will find in the in the ":TagList" section from the config.txt file, in the EEN, SEN, TEN and FEN columns, depending on the way you configure the send on alarm action for the Tag (that means, depending on which alarm status will trigger the send on alarm action):

ALM	ACK	RTN	END	Values
				0
x				8
	x			16
		x		32
			x	2

Table 38: Pattern for TagList parameter from config.txt

If you activate several of the send on alarm actions checkboxes, the result of the value will be an addition of selected fields' values.

- **Example**

If you activate "ALM" and "END" to trigger an SMS sending, the value of the "SEN" field will be 10.

5.3.2. Setting a Tag value, deleting a Tag and acknowledging an alarm

A Tag value can be set using the following sequence (shown for a Tag MM1):

```
SETSYS TAG, "LOAD", "MM1"
SETSYS TAG, "TAGVALUE", 1234
SETSYS TAG, "DoSetVal",1
SETSYS TAG, "SAVE"
```

There are other ways to change a Tag's value. Examples:

```
MM1@ = 1234
setio "MM1",1234
```

Let's the MM1 Tag is in alarm state. It is then possible to acknowledge its alarm with the following command:

```
SETSYS TAG, "LOAD", "MM1"
SETSYS TAG, "DoAck", 1
SETSYS TAG, "SAVE"
```

It is possible to delete a Tag with:

```
SETSYS TAG, "LOAD", "MM1"
SETSYS TAG, "DoDelete", 1
SETSYS TAG, "SAVE"
CFGSAVE : REM Writes configuration to flash
```

Add Tag

```
SETSYS TAG, "LOAD", "MM1"
SETSYS TAG, "Name", "New_TagName"
SETSYS TAG, "Address", "New_address"
SETSYS TAG, "SAVE"
CFGSAVE : REM Writes configuration to flash
```

- Note -

The fields that are not specified will be taken over from the "MM1" tag

5.4. User Section

This section describes the configuration fields for a single user. The fields are readable and writable using GETSYS and SETSYS with the USER parameters and the user name.

The following table describes the fields accessible from the User configuration. The web pages are found under **Users Setup/[The name from the User]**.

Name	Description
Id	User Id (only for programs simplicity)
FirstName	User first name
Lastname	User last name

Name	Description
Login	User login
Password	User password
Information	User information
Right	Combination of bits for user rights on Tags (acknowledge, view, write, …)
EMA	User Email address
SMS	User SMS number
AccessPage	The page the user is allowed to access
AccessDir	The directory (and subdirectories) the user is allowed to access
CBEn	Allow the user to use callback (allowed = 1)
CBMode	Callback phone number is: 0 = mandatory, 1 = user defined
CBPhNum	Callback phone number
DoDelete	Delete the User (0 = do not delete, 1 = delete)

Table 39: USERS Configuration fields

- **Example**

Change_password:

```
SETSYS USER, "LOAD", "pierre"
SETSYS USER, "password", "new_password"
SETSYS USER, "SAVE"
CFGSAVE : REM Writes configuration to flash
```

Add_user:

```
SETSYS USER,"LOAD","username" : REM The "username" must be an existing user.
The same access rights will be applied on the new user
SETSYS USER,"login","new_username"
SETSYS USER,"password","new_password"
SETSYS USER, "SAVE" CFGSAVE : REM Writes configuration to flash
```

Delete_user:

```
SETSYS USER,"LOAD","username"
SETSYS USER,"DoDelete",1
SETSYS USER, "SAVE"
CFGSAVE : REM Writes configuration to flash
```

Illustration Index

Index of Tables

Revision

Revision History

Revision Level	Date	Description
1.0	24/05/2016	Original new version of RG-002: Programming Reference Guide
1.1	05/12/2016	Correction Error in 2.1.76 TGET & 2.1.77 TSET
1.2	13/03/2017	Changed: RequestHttpx syntax Changed: WriteEBD
1.3	12/06/2017	Changed: Chapter 1.2 : program.bas content

Document build number: 26