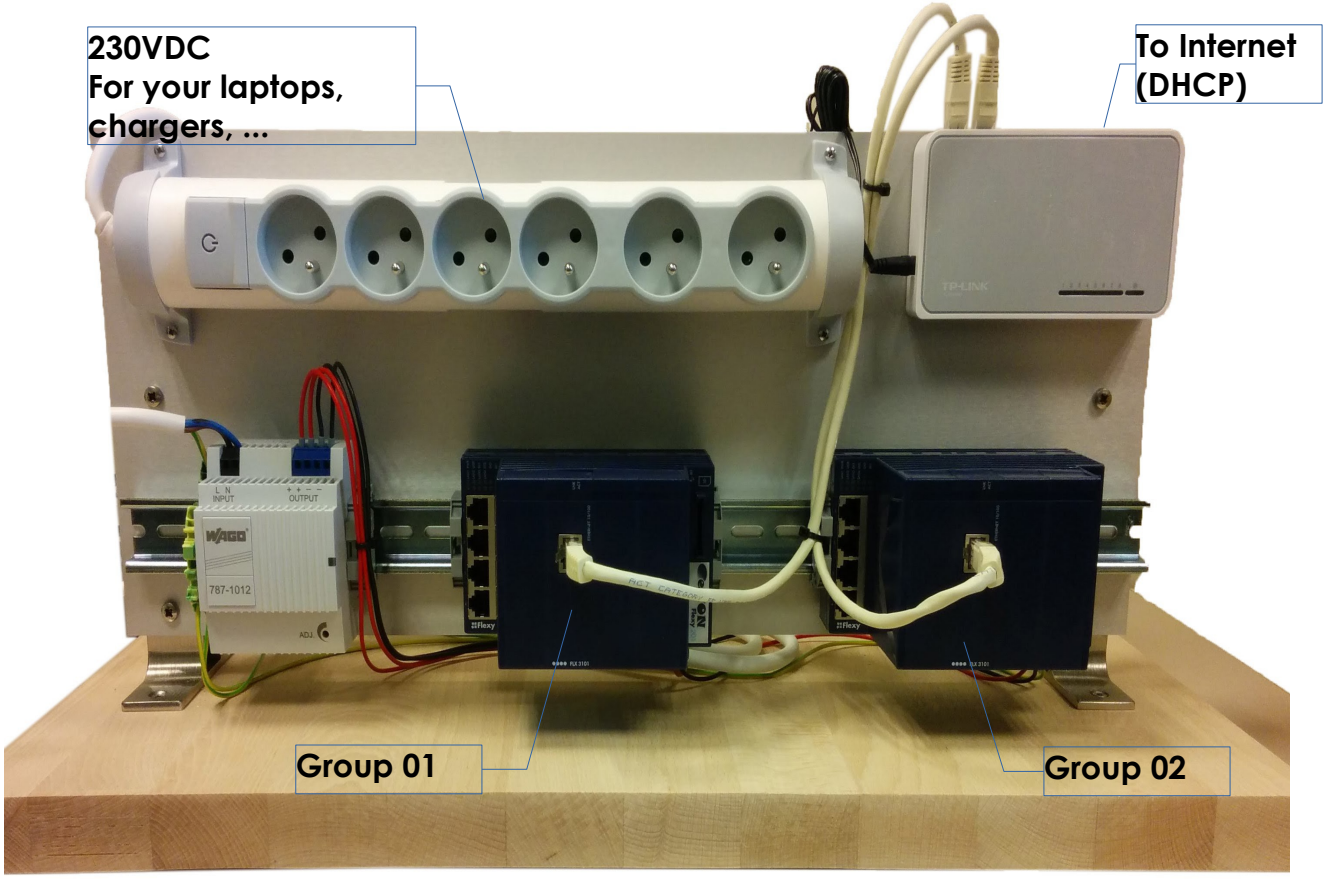
eWON
MACHINES CAN TALK

Flexthink!
Conference
for Developers

WORKSHOP

Custom data collection with BASIC

Toys presentation



Each table will be separated in two group and there is one eWON for each group.

The LAN IP address are written on it.

GROUP 1	GROUP 2
LAN IP: 10.20.1.1	LAN IP: 10.20.2.1
MASK: 255.255.0.0	MASK: 255.255.0.0
Login: adm Pwd: adm	Login: adm Pwd: adm
WAN IP: DHCP	WAN IP: DHCP

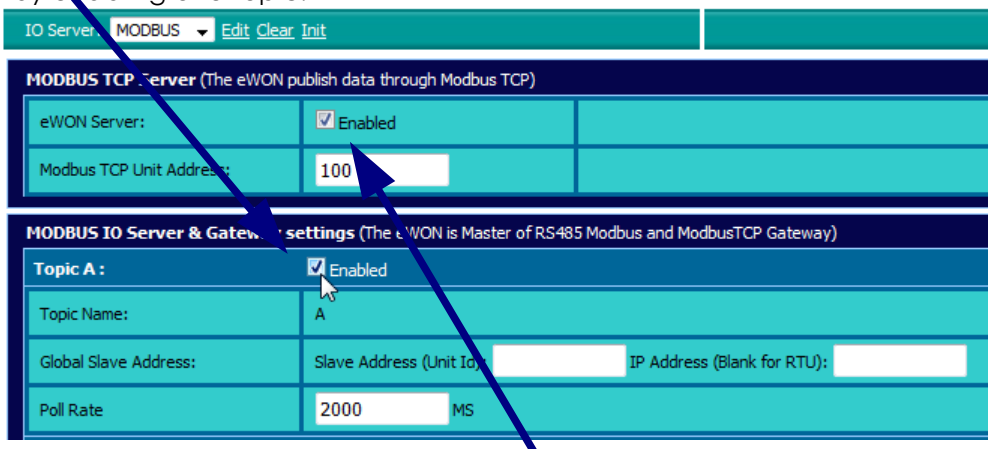
GROUP **X**	GROUP **Y**
LAN IP: 10.20.**X**.1	LAN IP: 10.20.**Y**.1
MASK: 255.255.0.0	MASK: 255.255.0.0
Login: adm Pwd: adm	Login: adm Pwd: adm
WAN IP: DHCP	WAN IP: DHCP

Exercise 1 : IOSend / IORcv

Purpose:
The phone number to call is stored in a char buffer on the PLC.
I want to read in one Modbus request this phone number.

eWON configuration – Modbus Client

As the IOSend / IORcv basic functions use an IOServer, you must configure it.
For this exercise, as we use ModbusTCP, simply starts the Modbus IOServer is enough.
You do that by enabling one topic.



eWON configuration – Modbus Server

For this exercise, your eWON will also play the PLC through the ModbusTCP Server

Server config:
For playing the PLC, your eWON requires to have the ModbusTCP server enabled.
It's enabled by default.

And some registers representing the char buffer published in ModbusTCP.
Every Tags on eWON can be published in ModbusTCP, then, simply define 10 MEM tags (representing 20 chars) on your eWON and configure each tags to publish on ModbusTCP.

Phone_01 : register 1000
Phone_02 : register 1001
…
Phone_10 : register 1009

Tag Name	Value	New Value		
phone_01	11059	11059	Update	[+3]
phone_02	12832	12832	Update	[2_]
phone_03	13879	13879	Update	[67]
phone_04	8248	8248	Update	[_8]
phone_05	14645	14645	Update	[95]
phone_06	8248	8248	Update	[_8]
phone_07	12336	12336	Update	[00]
phone_08	0	0	Update	
phone_09	0	0	Update	
phone_10	0	0	Update	

Your eWON is already configured with these tags.

Program.bas

The eWON is already programmed with some Basic codes showing how use the IOSend/IORcv functions... but not with the new BASIC2 features.
The purpose of this first exercise is to rework this little program to make it more *parametrisable* / *readable* by using new function declarations.

The code (version 1) is at the section

```
§ WkShop-Basic-ex1
```

```
23  WkShop-Basic-ex1
25  ReadPhone:
26    ReadFunction = 3
27    B$ = "100,10.2.100.147"
28    ReadRegAddr = 1000
29    ReadRegSize = 7
30
31  REM  ******************************************************
32  REM  ReadModbusRegister
33  REM  ReadFunction  is the Modbus function to use
34  REM        3 for HoldingRegister
35  REM        4 for InputRegister
36  REM  B$            is the Slave address of the device
37  REM  ReadRegAddr   is the register address to read
38  REM  ReadRegSize   is the size to read, in Word (16 bits)
39  REM  ******************************************************
40  ReadModbusRegister:
41    debug = 0
42    A$ = Chr$(INT(ReadFunction))
43    a% = ReadRegAddr - 1
44    a% = a% AND 65535
45    A$ = A$ + Chr$(a%/256)+Chr$(a% MOD 256)
46    a% = ReadRegSize
47    a% = a% AND 65535
48    A$ = A$ + Chr$(a%/256)+Chr$(a% MOD 256)
49    a% = Iosend "MODBUS",B$,A$
50
51  Wait_IO_End_1:
52    b% = Iorcv a%,1
53    If b%=-1 Then
54      Goto Wait_IO_End_1
55    Endif
56
57    B$ = Iorcv a%
58
59    If (B$="#ERR") Then
60      Print "Read ERROR"
61    else
62      //Z$ = B$( 3 To (2+(ReadRegSize*2)) )
63      Z$ = B$( 3 To )
64      Print "Read phone number"
65      Print "";Z$
66    Endif
67
68  End
69
```

Build the Modbus frame (A$) to send to the IOServer

Send the frame to IOServer

Wait the end of the IOSend

Read the IOServer answer

Error test
Answer is "#ERR" in case of error.

Extract the useful data from the answer

You can place your Basic2 code into the following empty section

```
§ WkShop-Basic-ex1-result
```

To test this (and your) program, you need to:

1. adapt the IP address (B$) to match your eWON IP address.

```
B$ = "100,10.2.100.147"
```

Ex: B$ = "100,10.20.17.1" if you are the Group17

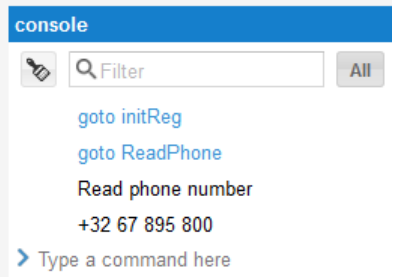
NB: the first value "100" is the modbus Slave address. It's by default 100 for the eWON modbus Server.

2. Set some values into the PLC (the phone tags)... if you talk currently the ASCII, set manually some values in Phone_xx tags... if not, call

```
GOTO InitReg
```

3. call the procedure

```
GOTO ReadPhone
```



NB: If you are not familiar with modbus protocol, use the debugger to follow step by step the execution of the program.

Playtime

For Beginner

Write the following code into your eWON, and execute it step by step

```
ReadModbusRegister:

  Addr$ = "100,10.2.100.147"
  Frame$ = Chr$(3)+Chr$(3)+Chr$(231)+Chr$(0)+Chr$(7)

  a% = Iosend "MODBUS",Addr$,Frame$

Wait_IO_End_1:
  b% = Iorcv a%,1
  If b%=-1 Then
    Goto Wait_IO_End_1
  Endif

  B$ = Iorcv a%

  If (B$="#ERR") Then
    Print "Read ERROR"
  Else
    Z$ = B$( 3 To )
    Print "Read phone number"
    Print "";Z$
  Endif

End
```

For Basic Friends

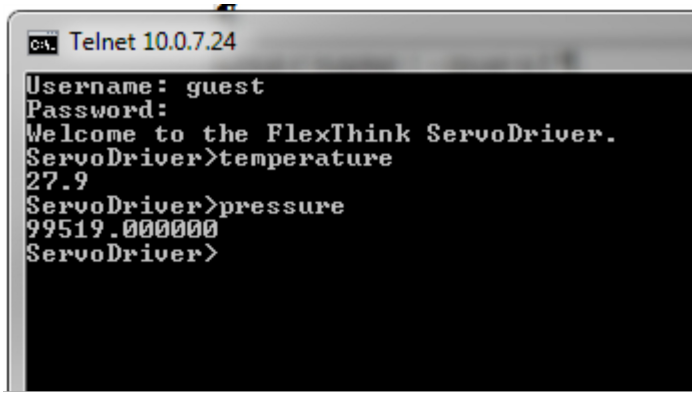
I propose to you to rework the program (on previous page) to make it more *parameterizable* / *readable* by using new function declarations.

Now, you can rework/debug the code of ReadPhone

You'll find technical info about IOSend / IORcv functions in appendices, as well as modbus frame format.

Exercise 2 : Telnet protocol implementation

Purpose:
On the Machine, there is a RaspberryPi handling the servo-motors and monitoring some data.
We can interrogate this RPi through a Telnet like protocol.

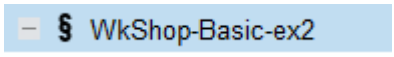


And we want the Flexy to retrieve the data from the RPi and put them into tags

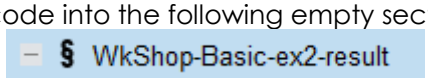
The eWON is already programmed with some Basic codes reading this Telnet protocol.
For that, you only need the using BASIC instructions : OPEN, PUT, GET and CLOSE

The purpose of this second exercise is to rework this little program to make it more *parametrisable* / *readable* with using new function declarations.

The code (version 1) is at the section



And you can place your Basic2 code into the following empty section



ServoDriver Telnet protocol

ServoDriver has a built-in server to answer some basic requests.
Open a tcp socket on port 8023.
You have to login first:

```
Username: guest
Password:
```

user: guest
password: guest
If user/password or if Number of connections (50) is exceeded, the telnet connection is closed.

Otherwise a welcome message is returned, followed by a prompt

```
Welcome to the FlexThink ServoDriver.
ServoDriver>
```

You can have the list of supported command by typing "help"

```
ServoDriver>help
Help on built in commands

? [<command>] - Display help
BYE - Exit the command shell
CURRENT - return the current consumed by servos
EXIT - Exit the command shell
HELP [<command>] - Display help
HISTORY - Display the command history
LOGOUT - Exit the command shell
POWER - return the global power consumed by servos
PRESSURE - return the current atmospheric pressure
QUIT - Exit the command shell
SERVO-COUNTERS <servo_ID> - get the number of change for servo <servo_ID>
TEMPERATURE - return the current on-board temperature
VOLTAGE - return the servo voltage
ServoDriver>
```

Then type de command to get the value…

```
ServoDriver>TEMPERATURE
33.0
ServoDriver>
```

The servo-counters command requires the id of the servo as parameters.
Each time the position of a servo changes, its counter is incremented.

```
ServoDriver>SERVO-COUNTERS 0
0
ServoDriver>SERVO-COUNTERS 1
8
```

Values are returned followed by a carriage return.
If something wrong happens, error message are preceded by "ERROR: "

```
ServoDriver>SERVO-COUNTERS 18
ERROR: Cannot get stats for servo 18
ServoDriver>
```

Program.bas (old Basic version)

```
Telnet:
  Clear : Cls

// ServoDriver protocol (telnet like)
  A$ = "10.0.7.24:8023"
  L$ = "guest"  // login
  P$ = "guest"  // password

SD_ErrorMsg$ = ""

  CLOSE 1
  A$ = "TCP:"+A$
  OPEN A$ FOR BINARY OUTPUT AS 1
Wait1:
  B$ = GET 1
  If (B$="#CLOSED#") Then GOTO Wait1

  delayToWait = 1000 : GOSUB Delay

  // read the prompt username
  A$ = GET 1,100
  p% = INSTR 1,A$,"Username:"
  If (p%=0) Then E$="Username prompt not found" : GOTO CloseTelnet  // error

  // send login
  A$ = L$ + CHR$(10)
  PUT 1,A$
  delayToWait = 500 : GOSUB Delay

  // read the prompt password
  A$ = GET 1,100
  p% = INSTR 1,A$,"Password:"
  If (p%=0) Then E$="Password prompt not found" : GOTO CloseTelnet  // error

  // send pwd
  A$ = P$ + CHR$(10)
  PUT 1,A$
  delayToWait = 1000 : GOSUB Delay

  // print Welcome message
  A$ = GET 1,100
  Print A$

  // now read some value from telnet protocol
  C$="CURRENT" : GOSUB GetCommandValue
  Servo_Current@ = commandValue
  Print "Current : ";commandValue

  C$="VOLTAGE" : GOSUB GetCommandValue
  Servo_Voltage@ = commandValue
  Print "Voltage : ";commandValue

  C$="PRESSURE" : GOSUB GetCommandValue
  Servo_Pressure@ = commandValue
  Print "Pressure : ";commandValue

CloseTelnet_B1:
  CLOSE 1
END

EndTelnet_B1:
Print "error with Telnet : ";E$
CLOSE1
END
```

```
243 // delayToWait = 1000 : GOSUB Delay
244 // delay to wait is in var 'delayToWait' in mSec
245 Delay:
246   s% = GETSYS PRG,"MSEC"
247 loopDelay:
248   For i%=1 To 1000
249   Next i%
250   t% = GETSYS PRG,"MSEC"
251   diff = t% - s%
252   If ( diff < delayToWait ) Then GOTO loopDelay
253 RETURN
254
255
256 // C$ equal command to send to Telnet
257 // E$ is the Global ERROR-MSG
258 GetCommandValue:
259   E$=""
260   A$ = C$ + CHR$(10)
261   PUT 1,A$
262   delayToWait = 100 : GOSUB Delay
263   A$ = GET 1,100
264
265   p% = INSTR 1,A$,"Unknown command"  // search for command error
266   If (p%<>0) Then E$=A$ : Print E$ :  RETURN
267
268   p% = INSTR 1,A$,CHR$(10)  // search for the <CR>  after the CMD echo
269   q% = INSTR (p%+1),A$,CHR$(10)  // search for the <CR>  after the VALUE echo
270   B$ = A$( (p%+1) TO (q%-1) )
271   commandValue = VAL(B$)
272
273 RETURN
```

Playtime

The purpose of this first exercise is to rework this little program to make it more *parameterizable* / *readable* by using new function declarations.

Now, you can rework/debug the code of Telnet.

Solutions

Exercise 1

```
ReadPhone_B2:
  PhoneNum$ = @readModbusRegister$("HOLDING","100,10.2.100.147",1000,7)
  Print "Read phone number"
  Print PhoneNum$
END

//*****************************************************************
//  readModbusRegister
//    PARAMS
//    $RegType$     :  is the Modbus register type, 'HOLDING' or 'INPUT'
//    $SlaveAddr$   :  is the Slave address of the device (ex: 100,10.0.0.53)
//    $RegAddr%     :  is the register address to read (first register is 1 ...not zero)
//    $RegSize%     :  is the size to read, in Word (16 bits)
//*****************************************************************
Function readModbusRegister$( $RegType$, $SlaveAddr$, $RegAddr%, $RegSize% )
  debug = 0
  If ($RegType$="HOLDING") Then
    $Frame$ = Chr$(3)
  Else
    $Frame$ = Chr$(4)
  Endif

  $RegAddr% = $RegAddr% - 1
  $RegAddr% = $RegAddr% AND 65535
  $Frame$ = $Frame$ + Chr$($RegAddr% / 256) + Chr$($RegAddr% MOD 256)
  $RegSize% = $RegSize% AND 65535
  $Frame$ = $Frame$ + Chr$($RegSize% / 256) + Chr$($RegSize% MOD 256)

  $Slot% = Iosend "MODBUS",$SlaveAddr$,$Frame$

  @waitIOSendEnd%( $Slot% )

  // read the buffer
  $Buffer$ = Iorcv $Slot%

  If ($Buffer$(1)=CHR$(131)) Then
    Print "Read ERROR"
    Print $Buffer$
  Endif

  $readModbusRegister$ = $Buffer$( 3 To )
EndFn

//*****************************************************************
Function waitIOSendEnd%( $SlotNum% )
  $Loop:
  $waitIOSendEnd% = Iorcv a%,1
  If $waitIOSendEnd%=-1 Then
    Goto $Loop
  Endif
EndFn

```

Exercise 2

```
282  TelnetB2:
283    Clear : Cls
284
285  // ServoDriver protocol (telnet like)
286    SD_Addr$ = "10.0.7.24"
287    SD_Port$ = "8023"
288
289    handle% = @openServoDriver%( 1, SD_Addr$, SD_Port$, "guest", "guest" )
290    If (handle%<>1) Then Print "Open failed" : GOTO EndTelnet : Endif
291
292    value = @getCommandValue(handle%, "CURRENT")
293    Servo_Current@ = value
294    Print "Current : ";value
295    value = @getCommandValue(handle%, "VOLTAGE")
296    Servo_Voltage@ = value
297    Print "Voltage : ";value
298    value = @getCommandValue(handle%, "PRESSURE")
299    Servo_Pressure@ = value
300    Print "Pressure : ";value
301
302  EndTelnet:
303    @closeServoDriver( handle% )
304
305  END
```

```
307  Function getCommandValue( $handle%, $cmd$)
308    SD_ErrorMsg$=""
309    @sendCmd($handle%, $cmd$)
310    @delay%(100)
311    $rep$ = @readBuffer$($handle%)
312    //print $rep$
313
314    $pos% = INSTR 1,$rep$,"Unknown command"  // search for command error
315    If ($pos%<>0) Then $getCommand=0 : SD_ErrorMsg$=$rep$ : RETURN
316
317    $CR$ = CHR$(10)
318    $pos1% = INSTR 1,$rep$,$CR$  // search for the <CR>  after the CMD echo
319    $pos2% = INSTR ($pos1%+1),$rep$,$CR$  // search for the <CR>  after the VALUE echo
320    $string$ = $rep$( ($pos1%+1) TO ($pos2%-1) )
321    $value = VAL($string$)
322
323    $getCommandValue = $value
324  EndFn
```

```
Function openServoDriver%( $handle%, $ipaddr$, $ipport$, $login$, $password$ )
  SD_ErrorMsg$ = ""

  CLOSE $handle%
  $addr$ = "TCP:"+$ipaddr$+":"+$ipport$
  OPEN $addr$ FOR BINARY OUTPUT AS $handle%
  @waitSocketReady( $handle% )

  @delay%(1000)
  $rep$ = @readBuffer$($handle%)
  $pos% = INSTR 1,$rep$,"Username:"
  If ($pos%=0) Then GOTO $CloseTelnet
  @sendCmd($handle%,$login$)
  @delay%(500)

  $rep$ = @readBuffer$($handle%)
  $pos% = INSTR 1,$rep$,"Password:"
  If ($pos%=0) Then GOTO $CloseTelnet
  @sendCmd($handle%,$password$)
  @delay%(1000)
  $rep$ = @readBuffer$($handle%)
  Print $rep$

  $openServoDriver% = $handle%
  RETURN

$CloseTelnet:
  CLOSE $handle%
  $openServoDriver% = 0
EndFn
```

```
Function closeServoDriver( $handle% )
  CLOSE $handle%
EndFn

Function sendCmd( $handle%, $cmd$ )
  $cmd$ = $cmd$ + CHR$(10)
  PUT $handle%,$cmd$
EndFn

Function waitSocketReady( $handle% )
$loop:
  $buf$ = GET $handle%
  If ($buf$="#CLOSED#") Then GOTO $loop
EndFn


Function readBuffer$( $handle% )
  $readBuffer$ = GET $handle%,100
EndFn

Function delay%( $mSec% )
  $start% = GETSYS PRG,"MSEC"
$loop:
  For $i%=1 To 1000
  Next $i%
  $now% = GETSYS PRG,"MSEC"
  $delay% = $now% - $start%
  If ( $delay% < $mSec% ) Then GOTO $loop
EndFn
```

Appendices

IOSend / IORcv : syntax

1.2.41 IORCV

Syntax [function]

IORCV S1

or

IORCV S1, I1

- **S1 is the STRING IOServerName**
- **I1 is an 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

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	slot 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

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
```

1.2.42 IOSEND

Syntax [function]

```
IOSEND S1, S2, S3
```

Purpose:

Sends a request by using the IO server's protocol.
See *"IORCV" on page 30* 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
```

Modbus protocol frame description

6.3 03 (0x03) Read Holding Registers

This function code is used to read the contents of a contiguous block of holding registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore registers numbered 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request

Function code	1 Byte	**0x03**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	**0x03**
Byte count	1 Byte	2 x **N***
Register value	**N*** x 2 Bytes	

***N** = Quantity of Registers

Error

Error code	1 Byte	**0x83**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read registers 108 – 110:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	03	Function	03
Starting Address Hi	00	Byte Count	06
Starting Address Lo	6B	Register value Hi (108)	02
No. of Registers Hi	00	Register value Lo (108)	2B
No. of Registers Lo	03	Register value Hi (109)	00
		Register value Lo (109)	00
		Register value Hi (110)	00
		Register value Lo (110)	64

The contents of register 108 are shown as the two byte values of 02 2B hex, or 555 decimal. The contents of registers 109–110 are 00 00 and 00 64 hex, or 0 and 100 decimal, respectively.