WEIGHT INDICATOR
MICROCONTROLLER

E-AF

MODBUS

Communication protocol

E-AF_01.00_12.01_EN_MODBUS

INDEX

1. GENERALITIES

The Modbus communication protocol defines the structure of the messages and the communication mode between a "master" device which manages the system and one or more "slave" devices which respond to the interrogations of the master (master-slave technique).
This protocol defines how the transmitter and receiver are identified, the data switch mode, the communication launch and interruption modes and the error detection mode.

Only the master can start a transaction (**Query**) while the other devices (the slaves) respond by supplying the data requested to the master or carrying out the actions requested in the query. The master can either address single slaves or transmit a broadcast message to all. The slaves respond with a message (**Response**) to the queries which are individually addressed, but do not transmit any answer to the master if there are broadcast messages.
A transaction can therefore have the following structures:

- Single question to a slave / Single answer from the slave
- Single broadcast message to all the slaves / No answer from the slaves

2. SYMBOLS

In the manual:

msb= most significant bit

MSB= most significant byte

lsb= less significant bit

LSB= less significant byte

3. SELECTION OF THE "MODBUS" SERIAL COMMUNICATION MODE

To select the Modbus communication protocol mode one should enter in the *SET-UP ENVIRONMENT* of the instrument:

Input in the Setup environment

- Turn on the indicator, press **TARE** while the software version is displayed (the display shows the "LANG" menu).
- Select the "SEtuP" step (using the ▼ key) and press ENTER;
- Select the "SEriAL" step (using the ▼ key) and press ENTER;
- Select the "CoM PC" step (using the ▼ key) and press ENTER;
- Select the "bAud" step (using the ▼ key) and press ENTER; select the desired baud rate and press ENTER;
- Select the "Add.485" step (using the ▼ key) and press ENTER; insert the machine code of the scale and press ENTER;
- Select the "ProtoC" step (using the ▼ key) and press ENTER; select the Modbus protocol and press ENTER;
- Press various times the C key until the display shows the "SAVE?" message.
- Press ENTER to confirm the changes made.

See the instrument's technical manual for further details.

4. RTU (BINARY) MODBUS TRANSMISSION MODE

Each byte (8 bit) in a message has 2 hexadecimal characters of 4 bits.
The main advantage of this mode, in comparison to the ASCII, is its greater density of characters which allow for the transmission of higher volume of data equal to the baud rate.

5. DESCRIPTION OF THE COMPONENTS AND MESSAGE FORMAT

In the RTU transmission modes, a Modbus message is put by the transmitting device inside a frame, which has a known beginning and end point. This allows for the receiving devices to locate the beginning of the message, read the address part and determine which device it is addressed to (or all the devices, if the message is broadcast) and to know when the message is complete. In this way the incomplete messages can be detected and consequently indicated as errors.
The format of the messages, for the master as well as the slave, includes:

- The **address of the device** with which the master has established the transaction (the address 0 corresponds to a broadcast message transmitted to all the slave devices).
- The **function code** which defines the requested action.
- The **data** which must be transmitted.
- The **error check** made up according to the CRC algorithm.

These fields are described in detail in the following paragraphs. For the Query and Response there is:

Query:
The *function code* tells the addressed slave device which action must be made. The *data* bytes contain some additional information which the slave needs in order to execute the function. The *error check* field gives the slave a method in order to confirm the integrity of the message contents.

Response:

- *If the slave gives a normal answer:*
  The *function code* is the echo of the query function code. The *data* bytes contain the data retrieved from the slave, like the registers' values or the states.
- *If a slave locates an error (format, parity, or in the CRC) or it is unable to execute the requested action:*
  The master message is considered non valid and rejected and consequently the action will not be executed; a Response in which the *function code* is changed in order to indicate that is an error response and the *data* bytes contain a code which describes the error.

5.1 FRAME FORMAT IN RTU MODE

In the RTU mode the messages start with a silent interval that lasts at least a period equal to 3,5 times the time period of a character (T1-T2-T3-T4 time interval, see Figure 1). The devices monitor continuously the transmission bus, also during the silent intervals. When the first field (the address) is received, each device decodes it in order to verify whether the device is addressed.
For each field the characters which may be transmitted are all the decimal values from 0 to 255.
After the last transmitted character there will be a silent interval equal to at least 3,5 times the time period of a character, indicating the end of the message; after this a new message can start.
The entire frame must be transmitted as a continuous stream. If there is a silent interval greater than the time period of 1,5 characters before the completion of the frame, the receiving device considers the incomplete message as ended and assumes that the next byte is the address field of a new message.
In the same way, if a new message starts before a time period equal to 3,5 characters following a previous message, the receiving device considers it a continuation of the previous message. This causes an error, and consequently the value in the final field of the CRC will not be valid, due to the combination of the two messages.

A typical message frame is shown in the following figure:

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1-T2-T3-T4	8 BITS	8 BITS	N * 8 BITS	16 BITS	T1-T2-T3-T4

Figure 1

5.2 THE DEVICE ADDRESS

As mentioned above, the Modbus transactions always involve the master, which manages the line, and a slave at a time (except for the broadcast messages). In order to identify the message consignee, the numeric address of the selected slave device (one byte: eight bits for the RTU) is transmitted as the first field of the frame. Each slave will therefore be assigned a different address number so that it can be identified. When the slave transmits its answer, its address is entered in the response's field address in order that the master knows which slave is responding.

Valid addresses for the slave devices are within a range from 0 to 247, in particular:

0 ⇨ *broadcast* address (all the slave devices)
1 ⇨ minimum possible address for the slave
247 ⇨ maximum possible address for the slave

5.3 THE FUNCTION CODE

The field of the frame function code of a message contains eight bits (RTU). Valid codes are within the range from 1 to 255 decimals.
When a message is transmitted from a master to a slave device the function code field indicates to the slave what kind of action should be executed (for example the reading of the *Input Registers*, etc.).
When a slave responds to the master, it uses the function code field in order to indicate either a normal response (without errors) or a type of error which has already taken place (called *exception* responses). For a normal response, the slave simply echoes the original function code, while for an exception response it gives back a code which is equivalent to the original function code, but with the most significant bit set at the 1 logic value.
Besides the modification of the function code for an exception response, the slave enters a single code within the data field of the response message, in order to tell the master which type of error has taken place or the reason for the exception.

5.4 THE DATA

The data field is made by using groups of two hexadecimal digits, in the range from 00 to FF Hex. This can be made by a RTU character, in accordance with the network's serial transmission mode.
The field data of the messages transmitted from the master to the slave devices contains additional information which the slave must use in order to carry out the action defined by the function code.

5.5 THE ERROR CHECK

The contents of the error check field depend on the Modbus transmission mode because there are two distinct error check methods. In particular:

The communication strings are checked by a CRC (Cyclical Redundancy Check) algorithm, see Section 7 for further details.

The error check field contains 16 bits (implemented as 2 bytes of 8 bits), which are the result of the calculation of a CRC algorithm executed on the contents of the message.
This field is the last of the message and the first byte is the one of the low order and is followed by one of the high order, which is the last one of the frame.

5.6 EXAMPLE OF THE MESSAGE COMPONENTS IN RTU

The following tables show an example of the fields inside a Modbus message, for a Query as well as for a normal Response; in both cases the fields' content is shown in hexadecimals and how the message is organized (framed) in RTU mode.

Query: **"Read Input Registers"** to the 01 Slave Device address, for the reading of the contents of 3 registers starting from register n°8.

Field Name	**Example (Hex)**	**RTU: 8-bit field**
Heading		None
Slave Address	*01*	0000 0001
Function	04	0000 0100
Start Address (HIGH)	00	0000 0000
Start Address (LOW)	08	0000 1000
Number of Registers (HIGH)	00	0000 0000
Number of Registers (LOW)	03	0000 0011
Error Check		CRC (16 bits)
Terminator		None
Nr. of total bytes		8

Response:

Field Name	**Example (Hex)**	**RTU: 8-bit field**
Heading		None
Slave Address	01	0000 0001
Function	04	0000 0100
Number of bytes	06	0000 0110
Data (HIGH)	02	0000 0010
Data (LOW)	2B	0010 1011
Data (HIGH)	00	0000 0000
Data (LOW)	00	0000 0000
Data (HIGH)	00	0000 0000
Data (LOW)	63	0110 0011
Error Check		CRC (16 bits)
Terminator		None
Nr. of total bytes		11

In the Response of the Slave there is the Function Echo indicating that it's a normal type of answer.
The "Number of Bytes" field specifies how many groups of 8-bit data are given back, in other words, the number of bytes of the "Data" fields is shown for the RTU.
For example the 63 Hex value is transmitted as a 8-bit byte (01100011).

6. MODBUS FUNCTIONS

Each function is exposed in detail in the following pages and is made up of a **QUERY** (Master request → Instrument) and a **RESPONSE** (Instrument response → Master).

NOTE:

Each character is an Hexadecimal type of character (made up of 4 bits).

- With **0x** or **Hex** before a number it means that it has to do with a hexadecimal value.

6.1 LIST OF THE SUPPORTED FUNCTIONS

In the following table there are the active Modbus functions for the instrument.

Function Code	Description
01 (0x01)	READ COILS STATUS
03 (0x03)	READ HOLDING REGISTERS
04 (0x04)	READ INPUT REGISTERS
05 (0x02)	PRESET SINGLE COIL
06 (0x06)	PRESET SINGLE REGISTER
06 (0x06)	PRESET SINGLE REGISTER
16 (0x10)	PRESET MULTIPLE REGISTERS

Table 1

In the parentheses there are the hexadecimal values.

6.2 LIST OF THE TRANSMISSION STRINGS

In the following paragraphs the functions (shown in Table 8) supported by the instrument are described in detail; for this purpose one uses the following classification for the message fields:

- **Address:** 1 byte for the instrument address (slave).
- **Function:** Code or Number of the function to be executed.
- **Number of bytes:** represents the number of bytes which make up a datum.
- **Error Check (CRC):** for the error check, in the RTU and in the ASCII transmission modes it's always 2 bytes. See section 7 for further details.

Other fields for the message frames are described in detail in the various single functions.

NOTE: the following registers are defined, on which the functions operate:

- N°16 Input Registers (or "Input Registers"): written by the Instrument → read by the Master
- N°16 Output Registers (or "Holding Registers"): written by the Master → read by the Instrument

The input and output registers are fully described in section 9.

6.3 FUNCTION 1, 3 and 4: READ COILS STATUS / HOLDING / INPUT REGISTERS (01, 03 and 04 Hex)

It reads the contents of the slave instrument's registers (which the instrument will write); the broadcast communication is not supported.

Query:
One specify the registers / coils data area to read (*Function*), the Initial Register (*1st Register Address*) from which the reading starts and the Number of Registers which must be read (*Nr. of Registers*). The registers are addressed from 0: in this way the registers from 1 to 16 are addressed as 0 to 15.

Address	Function	Address 1st Register		Nr. of Registers		Error Check
		High	Low	High	Low	
A	**04**	*00*	*08*	*00*	*01*	CRC

Response:
The response message is made up of 2 bytes for each read register, with the binary content aligned on the right in each byte. For each register the first byte contains the most significant bits and the second byte contains the least significant bits.

Address	Function	Address 1st Register		Nr. of Registers		Error Check
		High	Low	High	Low	
A	**04**	02		*00*	*0A*	CRC

Example: A = *01*;
- *in the Query*: 1st Register address = *00 08*; Number of Registers = *00 01*
- *in the Response*: 1st Register = *00 0A*

6.4 FUNCTION 5 and 6: PRESET COIL / SINGLE REGISTER (05 and 06 Hex)

It allows to set a single register (which the instrument or slave goes to read) to a determined value.
The broadcast transmission of this command is allowed and in which one can set the same register in all the connected slaves.

Query:
One specifies the Register / Coil data area to write (Function), the Register Address which must be set (*Register Address*) and the relative Value (*Register Value*). The registers are addressed starting from 0: in this way the registers from 1 to 16 are addressed as 0 to 15.

Address	Function	Address 1st Register		Nr. of Registers		Error Check
		High	Low	High	Low	
A	**06**	*00*	*01*	*00*	*03*	CRC

Response:
It is the echo of the Query.

Address	Function	Address 1st Register		Nr. of Registers		Error Check
		High	Low	High	Low	
A	**06**	*00*	*01*	*00*	*03*	CRC

Example: A = *01*;
- *in the Query*: Register Address = *00 01*; Register Value = *00 03*
- *in the Response*: Register Address = *00 01*; Register Value = *00 03*

6.5 FUNCTION 15 and 16: PRESET MULTIPLE COILS / REGISTERS (0F and 10 Hex)

Allows to set various registers (which the instrument or slave goes to read) to a determined value.

Query:

One specifies the registers / coils data area to write (*Function*). Here is specified the address of the First Register which must be set (*1st Register address*), the Number of Registers to be written (*Nr. of Registers*) starting from the first, the number of bytes transmitted for the values of the registers (2 bytes for each register) or *Nr. of Bytes* and then the values to be assigned to the registers (*1st Register value* of 2 bytes, *2nd Register Value, etc.*) starting from the first one addressed.

Address	Function	1st Register Address High Low	Nr. of Registers High Low	Nr. of bytes	1st Register Value High Low	2nd Register Value High Low	Error Check
A	**10**	*00 00*	*00 02*	*04*	*00 00*	*00 00*	CRC

Response:

The response includes the identification of the modified registers (*1st Register address* and *Nr. of Registers*).

Address	Function	Address 1st Register High Low	Nr. of Registers High Low	Error Check
A	**10**	*00 00*	*00 02*	CRC

Example: A = *01*;

- *in the Query*: 1st Register Address = *00 00*; Nr. of Registers = *00 02*; Nr. of bytes = *04*; 1st Register Value = *00 00*; 2nd Register Value = *00 00;*
- *in the Response*: 1st Register Address = *00 00*; Nr. or registers = *00 02;*

7. ERROR CHECK METHODS

The Modbus serial communication uses two error check types:

Check <u>on the character</u> or parity (even or uneven), can be applied *optionally* to each character
Check <u>on the frame</u> (CRC algorithm), applied to the entire message.

Both the check on the character as well as the one on the frame of the message is created in the Master and applied to the contents of the message before the transmission. The Slave checks each character and the entire frame of the message during the reception.

7.1 PARITY CHECK

It is possible to configure the parity check in the following way:

n ⇨ no parity (none)
E ⇨ even parity (Even)

This determines how the parity is set in each character.

7.2 CRC ALGORITHM: CYCLICAL REDUNDANCY CHECK (RTU MODE)

In the RTU transmission mode, the messages include an error check field based on a CRC method, which checks the contents of the entire message and is applied without any regard to any parity method used for the single characters. The CRC field is made up of 2 bytes (containing a binary value of 16 bits) and is calculated from the transmitting device, which puts the CRC at the end of the message. The receiving device calculates again the CRC during the reception of the message, and compares the calculated value with the actual value received in the CRC field. If the two values are not the same an error has taken place.

7.3 A PROCEDURE FOR CREATING THE CRC IS THE FOLLOWING

1. Loading a 16-bit register with FFFF Hex (all 1). This register is called *Register CRC*
2. OR-exclusive with the first byte of the message and the least significant byte of the *CRC Register* at 16 bit.
   The result is inserted in the *CRC register.*
3. The *CRC Register* is shifted of 1 bit to the right (towards the LSB), a 0 is inserted in the place of the MSB. The LSB is extracted and examined.
4. If LSB = 0 ⟶ Step 3 is to be repeated. (another shift)
   If LSB = 1 ⟶ The OR-ex is made between the *CRC Register* and the A001 Hex (1010 0000 0000 0001) polynomial value.
5. Steps 3. and 4. are repeated until 8 shifts have been made; after this a byte of 8 bits have been completely processed.
6. Steps 2 to 5 are repeated for the next byte of 8 bits of the message.
   One continues until all the bytes are processed.
7. The final contents of the *CRC Register* are the <u>CRC Value</u>.
8. When the CRC is inserted within the message, its bytes (high and low) must be exchanged as described below.

7.4 PLACING OF THE CRC IN THE MESSAGE

When the 16 bits of the CRC (2 bytes) are transmitted in the message, the least significant byte must be transmitted first, followed by the most significant byte.

For example, if the CRC value is 1241 Hex (0001 0010 0100 0001):

Addr	Func	Data	Data	Data	Data	Data	**CRC**	**CRC**
							LOW 41	**12**

Fig. 5*:* *Sequence of the CRC bytes.*

7.5 EXAMPLE IN C LANGUAGE IN GENERATING THE CRC

A functioning example for the creation of the CRC in the C language is shown below.

NOTE: *The function creates internally the swapping of the high and low bytes of the CRC. The bytes are already exchanged in the CRC value which is given back by the function, which can then be placed directly in the message for the transmission.*

The function uses two arguments:

`unsigned char` **`*puchMsg;`** → A pointer to the message buffer which contains the binary data to be used for creating the CRC

`unsigned short` **`usDataLen;`** → The quantity of bytes in the message buffer

The function gives back the CRC value as an *unsigned short*.

CRC generation function

```
unsigned short CRC16(unsigned char *puchMsg,
                     unsigned short usDataLen
                     )

{
    unsigned short CRC;
    int i, n;
    unsigned short usPolynomial = 0xA001;
    unsigned short usInitialReminder = 0xFFFF;
    CRC = usInitialReminder;//initialisation CRC
    for (i = 0; i < usDataLen; i++)//for each byte of the message it executes the division module-2 for the
    polynomial
    {
        CRC = CRC ^ puchMsg[i];//XOR byte low CRC with byte message under exam
        for (n = 0; n < 8; n++)// division module-2 at bit
        {
            if (CRC & 0x0001) //least significant bit CRC 1
            {
                CRC = CRC >> 1; //shift to the right of the CRC
                CRC = CRC ^ usPolynomial; //XOR CRC with polynomial
            }
            else//bit least significant CRC 0
            CRC = CRC >> 1; //shift to the right of the CRC
        }
    }
CRC = (CRC << 8) | (CRC >> 8); //switch of least significant and most significant byte
return CRC;
}
```

8. MODBUS EXCEPTIONS

In a normal response (Normal Response) the Slave device echoes the Function Code of the Query, putting it in the Response Function field. All the function codes have their own most significant bit (MSB) set at 0 (values less than 80 Hex). In an exception response (Exception Response) the slave sets the MSB of the Function Code at 1 (equivalent to summing the value 80 Hex to the normal response code) in order to signal that an anomaly has taken place, and the Exception Code is given back in the Data Field, in order to indicate the type of exception.

8.1 LIST OF THE DETECTED EXCEPTIONS

Active Modbus exceptions

Code	Exception	Description
01	*Illegal Function*	The ***Function Code*** received by the Query is not supported or not valid
02	*Illegal Data Address*	The **Data Address** received in the Query is not an address supported by the Slave Device or is not valid
03	*Illegal data Value*	A **Value in the Data field** of the Query is not a value acceptable by the Slave device or is not valid
06	*Slave Device Busy*	The **Slave is busy** in processing a command which requires a lot of time. The Master can transmit again the message later, when the Slave is free

Table 2

9. DATA AREAS

There are 3 data areas, "Input", "Holding" and "Coils", defined this way due to the master's point of view: while the "Input" area is read by this device, the "Holding" and "Coils" ones are written.
The first 2 areas are organised in registers on which the Modbus protocol functions perform.
All the numeric values have the Big Endian format (the 1st byte is the most significant one) for the Data Input Area and the Data Output Area, while these have the Little Endian format (the 1st byte is the least significant one) for the SETUP area.

9.1 INPUT REGISTERS DATA AREA

The input data area is read by the master (is therefore written by the instrument) and is made up of registers (input registers), of 2 bytes.

Table 9.1: Modbus Input Registers

Setup section

Register	Value
30001 (0)	Software version

- Format of the Software Version

The software version is in the BCD format. The first byte of the register is the software release converted in the BCD format and the second byte is the sub release converted in the BCD format.
Example: the software version is 02.11 is read 0000001000010001

30002	Configured channels
30003	Scale 1 capacity (High)
30004	Scale 1 capacity (Low)
30005	Scale 1 division
30006	Scale 1 decimals
30007	Scale 1 unit **(1)**
30008	Scale 2 capacity (High)
30009	Scale 2 capacity (Low)
30010	Scale 2 division
30011	Scale 2 decimals
30012	Scale 2 unit **(1)**
30013	Scale 3 capacity (High)
30014	Scale 3 capacity (Low)
30015	Scale 3 division
30016	Scale 3 decimals
30017	Scale 3 unit **(1)**
30018	Scale 4 capacity (High)
30019	Scale 4 capacity (Low)
30020	Scale 4 division
30021	Scale 4 decimals
30022	Scale 4 unit **(1)**
30023	Remote Scale capacity (High)
30024	Remote Scale capacity (Low)
30025	Remote Scale division
30026	Remote Scale decimals
30027	Remote Scale unit **(1)**
30028	Archive decimals
30029	Archive unit **(1)**

([1]) **NOTE**: ***Significance of the numeric value in the Unit of Measure field:***

0 → Grams
1 → Kilograms
2 → Tons
3 → Pounds

Status section

Register	Value
30101 (100)	Gross weight (High)
30102	Gross weight (Low)
30103	Net weight (High)
30104	Net weight (Low)

- Format of the GROSS WEIGHT and NET WEIGHT value

Whole in absolute value (without decimals)

Example: if 3 decimals are set, the value 3,000 is read 3000
If 2 decimals are set, the value 3,00 is read 300

30105	Input status register
30106	Command status register
30107	Output status register
30108	Scale state

- Format of the Input Status Register value

See 9.1.1 section

- Format of the Command Status Register value

See 9.1.3 section

- Format of the Output Status Register value

See 9.1.2 section

30109	Present scale
30110	Present channel ADC value (High)
30111	Present channel ADC value (Low)
30112	Present channel mV value, 3 decimals (so it's µV)
30113	High bit: sum mode (1 active, 0 not active); low bit: used channels

Application section

Register	Value			
	EAF03, EAF08, EAF09	**EAF02**	**EAF04**	**EAF05**
30201	Platform 1 weight (High)	APW decimals	Density (High)	Currency decimals
30202	Platform 1 weight (Low)	APW unit	Density (Low)	Currency decimals
30203	Platform 2 weight (High)	Pcs decimals	Gross Volume (High)	Amount (Word 3)
30204	Platform 2 weight (Low)	Pcs (High)	Gross Volume (Low)	Amount (Word 2)
30205	Platform 3 weight (High) for AF08	Pcs (Low)	Net Volume (High)	Amount (Word 1)
30206	Platform 3 weight (Low) for AF08	APW (High)	Net Volume (Low)	Amount (Word 0)
30207	Platform 4 weight (High) for AF08	APW (Low)		Price (High)
30208	Platform 4 weight (Low) for AF08			Price (Low)

Alibi memory section (for EAF08)

Register	Value
30301	Gross weight (High)
30302	Gross weight (Low)
30303	Tare (High)
30304	Tare (Low)
30305	ID (High)
30306	ID (Low)
30307	Alibi memory status (rewrite: 8 bits, scale number: 3 bits, manual tare flag: 1 bit)

9.1.1 <u>Input Status Register</u> (Table 9.1.1)

It is Input Register number 104; two bytes defined in the following way:

Bit	Description	Bit meaning	
		0	1
(LSB)			
0	Net Weight Polarity	+	--
1	Gross Weight Polarity	+	--
2	Weight Stability	NO	YES
3	Underload Condition	NO	YES
4	Overload Condition	NO	YES
5	Entered Tare Condition	NO	YES
6	Manual Tare Condition	NO	YES
7	Gross ZERO zone	Out of Zone 0	In Zone 0
(MSB)			
8	Input 1	DISABLED	ENABLED
9	Input 2	DISABLED	ENABLED
10	Input 3	DISABLED	ENABLED
11	Input 4	DISABLED	ENABLED
12	Input 5	DISABLED	ENABLED
13	Input 6	DISABLED	ENABLED
14	Input 7	DISABLED	ENABLED
15	Input 8	DISABLED	ENABLED

9.1.2 <u>Output Status Register</u> (Table 9.1.2)

It is Input Register number 106; two bytes defined in the following way:

Bit	Description	Bit meaning	
		0	1
(LSB)			
0	RELAY 1	NOT EXCITED	EXCITED
1	RELAY 2	NOT EXCITED	EXCITED
2	RELAY 3	NOT EXCITED	EXCITED
3	RELAY 4	NOT EXCITED	EXCITED
4	RELAY 5	NOT EXCITED	EXCITED
5	RELAY 6	NOT EXCITED	EXCITED
6	RELAY 7	NOT EXCITED	EXCITED
7	RELAY 8	NOT EXCITED	EXCITED
(MSB)			
8	RELAY 9	NOT EXCITED	EXCITED
9	RELAY 10	NOT EXCITED	EXCITED

10	RELAY 11	NOT EXCITED	EXCITED
11	RELAY 12	NOT EXCITED	EXCITED
12	RELAY 13	NOT EXCITED	EXCITED
13	RELAY 14	NOT EXCITED	EXCITED
14	RELAY 15	NOT EXCITED	EXCITED
15	RELAY 16	NOT EXCITED	EXCITED

9.1.3 Command Status Register

It is Input Register number 105; two bytes defined in the following way:

High Byte → ***Last received command*** (see Table 5.2.1)
Low Byte: low nibble → ***Counting of processed commands*** (module 16)
high nibble → ***Result of last received command***

In which the ***Result of the last received command*** can assume the following values:

- OK = 0 Corrected and executed command
- ExceptionCommandWrong = 1 Incorrect command
- ExceptionCommandData = 2 Data in the incorrect command
- ExceptionCommandNotAllowed = 3 Command not allowed
- ExceptionNoCommand = 4 Inexistent command

9.2 HOLDING REGISTERS DATA AREA

The "Holding" data area is written by the master (is therefore read by the instrument) and is made up of registers (holding registers), of 2 bytes.

Table 9.2: Modbus Holding Registers

Commands section

Register	Value
40001 (0)	Command register
40002	Parameter 1 (High)
40003	Parameter 1 (Low)

2 words Set-points section

Register	Value
40101 (100)	Set-point 1 ON value (High)
40102	Set-point 1 ON value (Low)
40103	Set-point 2 ON value (High)
40104	Set-point 2 ON value (Low)
40105	Set-point 3 ON value (High)
40106	Set-point 3 ON value (Low)
40107	Set-point 4 ON value (High)
40108	Set-point 4 ON value (Low)
40109	Set-point 5 ON value (High)
40110	Set-point 5 ON value (Low)
40111	Set-point 6 ON value (High)
40112	Set-point 6 ON value (Low)
40113	Set-point 7 ON value (High)
40114	Set-point 7 ON value (Low)
40115	Set-point 8 ON value (High)
40116	Set-point 8 ON value (Low)
40117	Set-point 9 ON value (High)

40118	Set-point 9 ON value (Low)
40119	Set-point 10 ON value (High)
40120	Set-point 10 ON value (Low)
40121	Set-point 11 ON value (High)
40122	Set-point 11 ON value (Low)
40123	Set-point 12 ON value (High)
40124	Set-point 12 ON value (Low)
40125	Set-point 13 ON value (High)
40126	Set-point 13 ON value (Low)
40127	Set-point 14 ON value (High)
40128	Set-point 14 ON value (Low)
40129	Set-point 15 ON value (High)
40130	Set-point 15 ON value (Low)
40131	Set-point 16 ON value (High)
40132	Set-point 16 ON value (Low)
40133	Set-point 1 OFF value (High)
40134	Set-point 1 OFF value (Low)
40135	Set-point 2 OFF value (High)
40136	Set-point 2 OFF value (Low)
40137	Set-point 3 OFF value (High)
40138	Set-point 3 OFF value (Low)
40139	Set-point 4 OFF value (High)
40140	Set-point 4 OFF value (Low)
40141	Set-point 5 OFF value (High)
40142	Set-point 5 OFF value (Low)
40143	Set-point 6 OFF value (High)
40144	Set-point 6 OFF value (Low)
40145	Set-point 7 OFF value (High)
40146	Set-point 7 OFF value (Low)
40147	Set-point 8 OFF value (High)
40148	Set-point 8 OFF value (Low)
40149	Set-point 9 OFF value (High)
40150	Set-point 9 OFF value (Low)
40151	Set-point 10 OFF value (High)
40152	Set-point 10 OFF value (Low)
40153	Set-point 11 OFF value (High)
40154	Set-point 11 OFF value (Low)
40155	Set-point 12 OFF value (High)
40156	Set-point 12 OFF value (Low)
40157	Set-point 13 OFF value (High)
40158	Set-point 13 OFF value (Low)
40159	Set-point 14 OFF value (High)
40160	Set-point 14 OFF value (Low)
40161	Set-point 15 OFF value (High)
40162	Set-point 15 OFF value (Low)
40163	Set-point 16 OFF value (High)
40164	Set-point 16 OFF value (Low)

1 word Set-points section (low part)

Register	Value
40201 (200)	Set-point 1 ON value
40202	Set-point 2 ON value
40203	Set-point 3 ON value
40204	Set-point 4 ON value

40205	Set-point 5 ON value
40206	Set-point 6 ON value
40207	Set-point 7 ON value
40208	Set-point 8 ON value
40209	Set-point 9 ON value
40210	Set-point 10 ON value
40211	Set-point 11 ON value
40212	Set-point 12 ON value
40213	Set-point 13 ON value
40214	Set-point 14 ON value
40215	Set-point 15 ON value
40216	Set-point 16 ON value
40217	Set-point 1 OFF value
40218	Set-point 2 OFF value
40219	Set-point 3 OFF value
40220	Set-point 4 OFF value
40221	Set-point 5 OFF value
40222	Set-point 6 OFF value
40223	Set-point 7 OFF value
40224	Set-point 8 OFF value
40225	Set-point 9 OFF value
40226	Set-point 10 OFF value
40227	Set-point 11 OFF value
40228	Set-point 12 OFF value
40229	Set-point 13 OFF value
40230	Set-point 14 OFF value
40231	Set-point 15 OFF value
40232	Set-point 16 OFF value

Scale total section

Register	Value
40301 (300)	Net Partial Total or Entered Partial Total for EAF03, EAF09 (Word 3)
40302	Net Partial Total or Entered Partial Total for EAF03, EAF09 (Word 2)
40303	Net Partial Total or Entered Partial Total for EAF03, EAF09 (Word 1)
40304	Net Partial Total or Entered Partial Total for EAF03, EAF09 (Word 0)
40305	Gross Partial Total or Exited Partial Total for EAF03, EAF09 (Word 3)
40306	Gross Partial Total or Exited Partial Total for EAF03, EAF09 (Word 2)
40307	Gross Partial Total or Exited Partial Total for EAF03, EAF09 (Word 1)
40308	Gross Partial Total or Exited Partial Total for EAF03, EAF09 (Word 0)
40309	Weighs Partial Total (Word 3)
40310	Weighs Partial Total (Word 2)
40311	Weighs Partial Total (Word 1)
40312	Weighs Partial Total (Word 0)
40313	Net General Total or Entered General Total for EAF03, EAF09 (Word 3)
40314	Net General Total or Entered General Total for EAF03, EAF09 (Word 2)
40315	Net General Total or Entered General Total for EAF03, EAF09 (Word 1)
40316	Net General Total or Entered General Total for EAF03, EAF09 (Word 0)
40317	Gross General Total or Exited General Total for EAF03, EAF09 (Word 3)
40318	Gross General Total or Exited General Total for EAF03, EAF09 (Word 2)
40319	Gross General Total or Exited General Total for EAF03, EAF09 (Word 1)
40320	Gross General Total or Exited General Total for EAF03, EAF09 (Word 0)
40321	Weighs General Total (Word 3)
40322	Weighs General Total (Word 2)
40323	Weighs General Total (Word 1)

40324	Weighs General Total (Word 0)
40325	Net Grand Total or Entered Grand Total for EAF03, EAF09 (Word 3)
40326	Net Grand Total or Entered Grand Total for EAF03, EAF09 (Word 2)
40327	Net Grand Total or Entered Grand Total for EAF03, EAF09 (Word 1)
40328	Net Grand Total or Entered Grand Total for EAF03, EAF09 (Word 0)
40329	Gross Grand Total or Exited Grand Total for EAF03, EAF09 (Word 3)
40330	Gross Grand Total or Exited Grand Total for EAF03, EAF09 (Word 2)
40331	Gross Grand Total or Exited Grand Total for EAF03, EAF09 (Word 1)
40332	Gross Grand Total or Exited Grand Total for EAF03, EAF09 (Word 0)
40333	Weighs Grand Total (Word 3)
40334	Weighs Grand Total (Word 2)
40335	Weighs Grand Total (Word 1)
40336	Weighs Grand Total (Word 0)
40337	Net First Archive Total or Entered First Archive Total for EAF03, EAF09 (Word 3)
40338	Net First Archive Total or Entered First Archive Total for EAF03, EAF09 (Word 2)
40339	Net First Archive Total or Entered First Archive Total for EAF03, EAF09 (Word 1)
40340	Net First Archive Total or Entered First Archive Total for EAF03, EAF09 (Word 0)
40341	Gross First Archive Total or Exited First Archive Total for EAF03, EAF09 (Word 3)
40342	Gross First Archive Total or Exited First Archive Total for EAF03, EAF09 (Word 2)
40343	Gross First Archive Total or Exited First Archive Total for EAF03, EAF09 (Word 1)
40344	Gross First Archive Total or Exited First Archive Total for EAF03, EAF09 (Word 0)
40345	Weighs First Archive Total (Word 3)
40346	Weighs First Archive Total (Word 2)
40347	Weighs First Archive Total (Word 1)
40348	Weighs First Archive Total (Word 0)
40349	Net Second Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 3)
40350	Net Second Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 2)
40351	Net Second Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 1)
40352	Net Second Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 0)
40353	Gross Second Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 3)
40354	Gross Second Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 2)
40355	Gross Second Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 1)
40356	Gross Second Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 0)
40357	Weighs Second Archive Total (Word 3)
40358	Weighs Second Archive Total (Word 2)
40359	Weighs Second Archive Total (Word 1)
40360	Weighs Second Archive Total (Word 0)
Not present in EAF01, EAF02, EAF	
40361	Net Third Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 3)
40362	Net Third Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 2)
40363	Net Third Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 1)
40364	Net Third Archive Total or Entered Second Archive Total for EAF03, EAF09 (Word 0)
40365	Gross Third Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 3)
40366	Gross Third Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 2)
40367	Gross Third Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 1)
40368	Gross Third Archive Total or Exited Second Archive Total for EAF03, EAF09 (Word 0)
40369	Weighs Third Archive Total (Word 3)
40370	Weighs Third Archive Total (Word 2)
40371	Weighs Third Archive Total (Word 1)
40372	Weighs Third Archive Total (Word 0)

- Format of the GROSS TOTAL and NET TOTAL value
Whole in absolute value (without decimals)
Example: if 3 archive decimals are set, the value 3,000 is read 3000
If 2 archive decimals are set, the value 3,00 is read 300

Archive Record section

Register	Value
40401 (400)	First Archive Selected Record
40402	Second Archive Selected Record
40403	Third Archive Selected Record

In which:

Firmware	First archive	Second archive	Third archive
EAF01	Articles	Customers	
EAF02	Articles	Customers	
EAF03	Customers	Materials	Vehicles
EAF04	Articles		
EAF05	Products	Customers	Ingredients
EAF08	Database		
EAF09	Customers	Materials	Vehicles

NOTE: The value 65535 identifies the deselected record.

9.2.1 Command Register

It is the Output Register number 0. It is made up of two bytes and can take on the following values, which correspond to implemented commands described in table 9.2.1.

Execution of a Command
The execution of a command happens when the contents of the Command Register vary (therefore to repeat the last command one should first set the Command register at the NO COMMAND value, and then at the command value).

Table 9.2.1: Command Register

Command	Value	Description	Parameters
NO_COMMAND	0 (0x0000)	No command	None
ZERO_REQUEST	1 (0x0001)	Zero scale	None
TARE_REQUEST	2 (0x0002)	Auto tare	None
TAREMAN_REQUEST	3 (0x0003)	Preset tare	Param. 1 = tare value **([3])**
REMOTE_SCALE_REQUEST	4 (0x0004)	Remote scale switch	None
CHANNEL_1_REQUEST	5 (0x0005)	Scale 1 switch	None
CHANNEL_2_REQUEST	6 (0x0006)	Scale 2 switch	None
CHANNEL_3_REQUEST	7 (0x0007)	Scale 3 switch	None
CHANNEL_4_REQUEST	8 (0x0008)	Scale 4 switch	None
SUM_REQUEST	9 (0x0009)	Sum mode switch	None
READ_ALIBI (for EAF08)	10 (0x0010)	Read alibi memory	Param.1 = rewrite, param. 2 = ID
WRITE_ALIBI (for EAF08)	11 (0x0011)	Write alibi memory	None

([3]) NOTE: *Format of the Parameter 1 and Parameter 2 values:*
→ *For the MANUAL TARE (only Param1):*

Example: if 3 decimals are set, in order to enter the value 3,000 → one should write 3000
If 2 decimals are set, in order to enter the value 3,00 → one should write 300

9.3 COILS STATUS DATA AREA

The "Coils status" data area is written by the master (is therefore read by the instrument) and is made up of coils of 1 bit.

Table 9.3: Modbus Coils Status

I°Coil.	Coils Status	Bit meaning	
		0	**1**
00001 (0)	Digital output 1 **([4])**	Not active	Active
00002 (1)	Digital output 2 **([4])**	Not active	Active
00003 (2)	Digital output 3 **([4])**	Not active	Active
00004 (3)	Digital output 4 **([4])**	Not active	Active
00005 (4)	Digital output 5 **([4])**	Not active	Active
00006 (5)	Digital output 6 **([4])**	Not active	Active
00007 (6)	Digital output 7 **([4])**	Not active	Active
00008 (7)	Digital output 8 **([4])**	Not active	Active
00009 (8)	Digital output 9 **([4])**	Not active	Active
00010 (9)	Digital output 10 **([4])**	Not active	Active
00011 (10)	Digital output 11 **([4])**	Not active	Active
00012 (11)	Digital output 12 **([4])**	Not active	Active
00013 (12)	Digital output 13 **([4])**	Not active	Active
00014 (13)	Digital output 14 **([4])**	Not active	Active
00015 (14)	Digital output 15 **([4])**	Not active	Active
00016 (15)	Digital output 16 **([4])**	Not active	Active

([4]) NOTE: Writing allowed if the related output has no associated function.