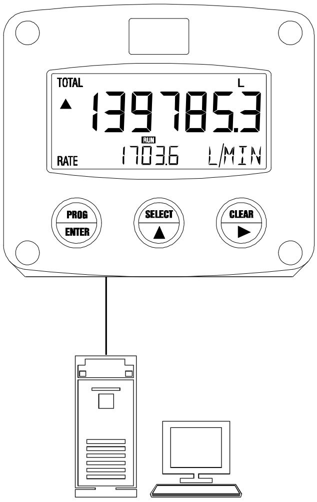
FLUIDWELL GENERAL MODBUS COMMUNICATION PROTOCOL



Manual	:	MODB100_v1302
Copyright 1998	:	Fluidwell bv - The Netherlands.

Warning: Any responsibility is lapsed if the instructions and procedures as described in this manual are not followed.

LIFE SUPPORT APPLICATIONS: The products FW/0110-series are not designed for use in life support appliances, devices, or systems where malfunction of the product can reasonably be expected to result in a personal injury. Customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify the manufacturer and supplier for any damages resulting from such improper use or sale.

CONTENTS:

1. INTRODUCTION

1.1. GENERAL

Fluidwell's communication protocol is compatible with MODBUS, used for Modicon Plc's e.g.. This protocol is described in "Protocol Reference Guide" (Modicon doc-nr. PI-MBUS-300 rev. C).

Fluidwell supports both modes of the above mentioned protocol: ASCII-mode and RTU-mode. These modes are described in chapter 2. and chapter 3. The selection between both modes is done at SETUP-level (100-series) or Parameter-level (300-series), just as setting the baud-rate, bus address and delay time.

The baudrate is selectable through values 1200, 2400, 4800, 9600. The bus address can be set from 1 up to and including 255.
Fluidwell equipment uses eight databits, no parity, a startbit and one stopbit. The least significant bit of every byte is send first.

This manual describes the general MODbus protocol implementation for all 100- and 300-series products. The product specific variables are described in the appendix of the operation manual.

1.2 DESIGN CHOICES IN PERSPECTIVE TO MODBUS SPECIFICATION

This communication package supports MODBUS functions “read holding registers” (03) and “preset multiple registers” (16). It should be noted that for these functions MODBUS specifies a register as being two bytes. Since some of the variables have a length which exceeds two bytes, a variable might need to be represented in more than one register. Therefore for reading or writing a variable its number (register number) and its length (number of registers) need to be supplied. The internal variable table may be visualised as a collection of small groups of registers, with each group representing a unique variable. This leads to the implication that it is not possible to read/write multiple variables in one request. Refer to chapter 5 for exact description of implemented MODBUS functions.

2. ASCII RECORD STRUCTURE

2.1. GENERAL

In ASCII-mode the databytes are ASCII-coded. This means that each (hexadecimal) databyte is split up into two nibbles; each nibble is send as its ASCII code afterwards. In this way every databyte is transferred as two ASCII-bytes.

Example:
The databyte 4Ah is transferred as 34h (= '4') and 41h (='A').

The advantage of this protocol-mode is that an ASCII-terminal can be used to "overhear" the dataline during debugging. A disadvantage is the quantity of data almost doubles.

The most significant nibble is send first.

msn	: most significant nibble ASCII coded
lsn	: least significant nibble ASCII coded

Following the example: msn = 34h and lsn = 41h.

ASCII BYTE-NR	**DATA BYTE-NR**	**Contents:**	**Meaning:**
00	00	":" (3Ah)	Start of record marker
01	01	address msn	Bus address product
02	01	address lsn	Bus address product
03	02	function msn	Function to be executed
04	02	function lsn	Function to be executed
05	03	msg-byte 0 msn	Data for/from the function
06	03	msg-byte 0 lsn	Data for/from the function
07	04	msg-byte 1 msn	Next byte
08	04	msg-byte 1 lsn	Next byte
//	//	etc.	//
05+2*x	03+x	msg-byte x msn	Last function data byte
06+2*x	03+x	msg-byte x lsn	Last function data byte
07+2*x	04+x	ERROR CHECK msn	LRC: LongitudinalRedundancy
08+2*x	04+x	ERROR CHECK lsn	Check: see below
09+2*x	05+x	CR (0Dh)	End of record marker 1
10+2*x	06+x	LF (0Ah)	End of record marker 2

Table 1: ASCII-mode.

Since each databyte is now represented as two ASCII-bytes, numbering of data-bytes in the record does not count up synchronously with the real number of bytes transferred over the line.

2.2. LRC - CHECKSUM

The checksum is calculated by adding all databytes in the record together from bus-address (data-byte 01) up to and including the last function data-byte (03+x) and take the two’s complement of the least significant byte, which means inverting the byte and adding 1. The calculated LRC-byte is now added to the record as its ASCII-coded 2-byte value.

The receiver must use the same kind of calculation and compare the result with the received checksum. A less complicated procedure is to calculate from the bus address up to AND INCLUDING the checksum (this means databytes 1 up to and including 04+x). If the record is not mutilated after receiving, the result must be 00.

All characters different as CR, LF, '0' - '9', ':', 'A' - 'F' point out a mutilated message.

3. RTU RECORD STRUCTURE

3.1. GENERAL

RTU BYTE-NR	**DATA BYTE-NR**	**Contents:**	**Meaning:**
		A time of at least 0.11 seconds between records is used as record separator!	
01	01	address	Bus address product
02	02	function	Function to be executed
03	03	msg-byte 0	Data for/from the function
04	04	msg-byte 1	Next byte
//	//	etc.	//
05+x	05+x	msg-byte x	Last function data byte
06+x	06+x	ERROR CHECK **lsb**	CRC: Cyclic Redundancy Check
07+x	07+x	ERROR CHECK **msb**	CRC: Cyclic Redundancy Check
		A time of at least 0.11 seconds between records is used as record separator!	

Table 2: RTU-mode.

In the table above, databyte-number 0 is NOT used by purpose to express the similarity with the ASCII-mode.

RTU-mode sends all databytes without conversion; time outs are used as record separators. Please notice that the time between two databytes WITHIN a record may not exceed 0.11 seconds! This time-to-wait is enlarged to two "PC-clockticks" (2/18.2 sec) compared with the original MODBUS specification, to get a more reliable and simplified time-out detection.

In RTU-mode the databyte-numbers in the record do count up synchronously with the real number of bytes transferred over the line, since each byte is transferred in its original form.

3.2. CRC-ERROR CHECK

The CRC-error check method uses the following polynomial:

X16 + X15 + X2 + X0

Description applied procedure:

1 Initialise a 16 bit-register with 0FFFFh (sixteen ones).
2 Put pointer to first byte (databyte no. 1 (address)).
3 Initialise a bit-counter to 8.
4 Put EXOR from LSbyte 16 bits-reg and pointed byte in LSbyte 16 bits-reg.
5 Shift the 16 bits-reg one position to the right; shift in a zero at the left side, save the shifted-out bit (LSbit).
6 Is the shifted-out bit "1", then EXOR the 16 bits-reg with A001h.
7 Is the shifted-out bit "0", then don't do anything at all.
8 Decrement bit-counter; if not all bits done, return to step 5.
9 Increment pointer to next byte, if not all bytes done, return to step 3.
10 Now, the sender adds the contents of the 16 bits register to the end of the message to be send.

WARNING: The CRC checksum bytes are in LSB-MSB order! (Contrary to the other words in the message which are in MSB-LSB order)

This CRC must be calculated from the bus-address (data-byte 01) up to and including the last data-byte (05+x).
The receiver can determine the CRC by following the same procedure, but now up to and including both checksum bytes (06+x/07+x). The now calculated checksum must equate zero, otherwise a mutilated message was received.
Extending the procedure to including the checksum bytes avoids the need for a compare of received and newly calculated checksum.

4. FUNCTION CODES

4.1. GENERAL

The table below contains an overview of the communication-functions. The second column contains the standard MODBUS description.
To be compatible with the several MODBUS-implementations in software-programs as SCADA e.g., only the Read en Write holding registers functions are used.
The third column shows the Fluidwell name. The last column indicates the paragraph which contains more detailed explanation of the function.

Due to the standard table-begin for all the Fluidwell equipment (see chapter 6), it is possible to detect the type of product, software-version etc. of each product present on the communication bus.

MODICON FUNCTION CODES		**FLUIDWELL IMPLEMENTATION**	**PAR.**
00	Not defined		
01	Read Coil(s) status		
02	Read input(s) status		
03	Read holding registers	Read table data	5.2
04	Read input registers		
05	Force single Coil		
06	Preset single register		
07	Read exception status		
08	Loopback diagnostic test		
09	Program (484 only)		
10	Poll prog. complete (484)		
11	Fetch event count.comm.		
12	Fetch comm. event log		
13	Program (184,384,484,584)		
14	Poll prog. (see 13)		
15	Force multiple coils		
16	Preset multiple reg's.	Write table data	5.3
17	Report slave id.		
18	Program 884 & micro 84		
19	Reset comm. link		
20	Read general ref. (584)		
21	Write general ref. (584)		
22-64	Reserv.f. exp. funct.		
65-72	Reserv.f. user funct.		
73-119	Illegal functions		
120-127	Reserv.f. intern. use		
128-255	Exception responses	Error response	5.4

Table 4: Function codes.

5. EXPLANATION FUNCTION CODES

5.1. GENERAL

In the diagrams below, following lay-out is used:

The upper line indicates the sequence number of each byte. These sequence numbers correspond with the databyte-numbers in chapter 2 and chapter 3.

The byte(s) 'CS' contains the CheckSum. In ASCII-mode this is a one-byte LRC (which will be represented in two bytes, just like all databytes in an ASCII record) and for RTU-mode this is a two byte CRC.

5.2. READ TABLE DATA

This function complies with the MODBUS specified function "Read holding registers" (03). It is used to read variables from the system. The system will respond to this type of message by sending either an acknowledge or an error response. Below you will find the record structures that are used to request and acknowledge this function.

The read request record looks like:

1	2	3	4	5	6	7	⇐ Databyte numbers
ADDRESS	FUNCTION	VARIABLE NUMBER		NUMBER OF WORDS		CS	
address	03t	msbyte	lsbyte	msbyte	lsbyte		

A read response record looks like:

1	2	3	4	5		⇐ Databyte numbers
ADDRESS	FUNCTION	BYTE-	VARIABLE 1		//	
address	03t	count n	msbyte	lsbyte	//	

	2+2*n	3+2*n	4+2*n	⇐ Databyte numbers
//	VARIABLE n		CS	
//	msbyte	lsbyte		

Address refers to the bus-address of the slave module that needs to execute the function. Function holds the MODBUS function code. Variable number tells the system which variable to read, a complete table of variable numbers can be found in chapter 6 and Appendix A.
These tables also show the number of bytes that need to be read to get the complete value. It is by no means possible to read more bytes than the real length of the variable as mentioned in this table. When such a request is made though, the system will respond with an error. This implies it is only possible to read one variable per request.

It should be noted that the read request record supplies the system with the number of words instead of the number of bytes to read. When variables with an odd number of bytes need to be read the number of words is rounded up (e.g. to read 1-byte bus address -> requested number of words is 1). This will result in an response which holds one extra byte, which will always be 0. The result is valid though, because this extra byte will always be the MS-byte of the returned data.

In contrary to reading too much bytes, it is possible to read a variable partially. When a request is made to read less bytes than the complete length of the variable, the system will return the requested number of bytes from the least significant part of the variable (e.g. datetime holds 6 bytes YMDHMS, when you request 2 words DHMS will be returned).

When the read request record is correctly received by the system and all data-members turn out to be valid, the system will return the read response record as mentioned above. The address and function bytes are an exact copy of the ones in the read request record, bytecount shows the number of bytes of data that the record holds. Of course as many bytes as requested are returned, so this number always equates 2*number_of_words.

In case any data-members in the read request record turn out to be invalid, the system will return an error response record. Paragraph 5.4 holds an exact description of all possible error records with an explanation of the possible causes. When the system does not return any record at all the cause might be found in a wrong bus-address, wrong checksum or a distortion in the communication process.

5.3. WRITE TABLE DATA

This function complies with the MODBUS specified function "Preset multiple registers" (16). It is used to write variables in the system. The system will respond to this type of message by sending either an acknowledge or an error response. Below you will find the record structures that are used to request and acknowledge this function.

It is not always possible to write a variable. Some variables are never writeable (e.g. serial number) and some others can become temporarily unavailable for writing (e.g. baudrate). The standard table for Fluidwell products (chapter 6) contains a column which shows the writeability of a variable. For the product specific variables mentioned in Appendix A goes that the ones that are also programmable on the unit itself (configuration variables) are unavailable for writing during local programming, to avoid conflicts between local and remote programming. Other, non-programmable, product specific variables are never writeable unless mentioned otherwise.

The write request record looks like:

1	2	3	4	5	6	7	← Databyte numbers
ADDRESS	FUNCTION	VARIABLE NUMBER		NUMBER OF WORDS		BYTE COUNT	
address	16t	msbyte	lsbyte	msbyte	lsbyte		

8	9	10	11	2n+6	2n+7	2n+8	← Databyte numbers
WORD 1		WORD 2		WORD n		CS	
msbyte	lsbyte	msbyte	lsbyte	msbyte	lsbyte		

An write response record looks like:

1	2	3	4	5	6	7	← Databyte numbers
ADDRESS	FUNCTION	VARIABLE NUMBER		NUMBER OF WORDS		CS	
address	16t	msbyte	lsbyte	msbyte	lsbyte		

Address, function, variable number and number of words are used the same way as with the read request record which was explained in the previous paragraph. Furthermore the bytecount should hold the number of bytes that need to be written to the given variable. The system will only accept a write request record when the bytecount equates the doubled value of the number of words (bytecount=2*number_of_words).

The number of words may never exceed the true length of the variable as given in the variable table. When variables with an odd number of bytes need to be written the number of words is rounded up (e.g. to write 1-byte bus address -> number of words is 1). This implies that the bytecount should always be an even number and therefore a dummy-byte needs to be added when variables of odd length are written. This dummy-byte should always be zero and must be added as the most significant byte in the data to write.

Example
We want to change the communication mode of the system. In the standard begin table for Fluidwell products we find that the variable number is 11 (0Bh). We also find that this variable is only one byte in length. Furthermore The MODBUS function code for writing is 16 (10h) and we assume the bus-address as being 01 for the system in question. We are willing to write the byte XX to this variable. The write request record should look like this:

01 10 00 0B 00 ***01 02 00 XX*** *CS*

The highlighted part shows we request 1 word, which leads to a bytecount of 2 and we add an extra '0'-dummy-byte at the most significant part of the data to write.

In contrary to writing too much bytes, it is possible to write a variable partially. When a request is made to write less bytes than the complete length of the variable, the system will overwrite the least significant part of the variable in question.

When the write request record is correctly received by the system and all data-members turn out to be valid, the system will return the write response record as mentioned above. This response record is an exact copy of the request record, only without the bytecount and the written databytes.

In case any data-members in the write request record turn out to be invalid, the system will return an error response record. Paragraph 5.4 holds an exact description of all possible error records with an explanation of the possible causes. When trying to write a variable from the "standard begin-table for Fluidwell products" that is not implemented on the system in question the write request will be confirmed, but of course this has no influence on the system whatsoever.
When the system does not return any record at all the cause might be found in a wrong bus-address, wrong checksum or a distortion in the communication process.

5.4 ERROR RESPONSE CODES

When a request is made and the system finds one of the data-members in the record to be invalid it will make this not acknowledge clear by sending an error response record.

1	2	3	4
ADDRESS	FUNCTION	ERROR	CS
address	OR 80h		

⇐ Databyte numbers

To let the master system know this is an error record the MODBUS function code will be changed. While the ms-bit of this code is normally always 0 it will be set to 1 (=OR 0x80). The master system can determine what went wrong by reading the error code on the third position of the error response record. In the table below you can find these error codes with an explanation of its meaning and the possible cause(s).

Error code 0 = reserved for future use

Error code 1 = ILLEGAL FUNCTION
Meaning: The function code received by the system does not represent an executable function.
Possible causes:
1. Requested function is not implemented on this system.
2. Function code does not represent a valid MODBUS function.

Error code 2 = ILLEGAL DATA ADDRESS
Meaning: The received data cannot be linked to a variable in the system
Possible causes:
1. Variable number does not represent a valid variable on this system.
2. Length of record does not comply with the expected length.
3. Number of words exceeds the real length of the variable.

Error code 3 = ILLEGAL DATA VALUE
Meaning: The data is not valid with this variable
Possible causes:
1. Value is not valid for this variable. It exceeds the value range of this variable (too low/high).
2. Dummy byte does not equate zero when writing an odd number of bytes.

Error code 4 = FAILURE IN ASSOCIATED DEVICE
Meaning: the message contains an illegal request, so it can not be processed
Possible causes:
1. The communication index is out of range when trying to access an indexed variable.

Error code 5: reserved for future use.

Error code 6 = SLAVE DEVICE BUSY
Meaning: The variable is not available for remote access
Possible causes:
1. Writing to a read-only variable.
2. System is currently being used in program-mode, so remote access is temporarily disabled for the variable that is being programmed.

When a function is requested and no response frame comes back within a short time (with regards to the Comdelay value!) the possible causes might be:
1. communication protocol is set to off,
2. message was mutilated due to a distortion on the line,
3. a hardware problem,
4. wrong bus-address,
5. baudrate of communicating devices differs (2400 <-> 9600),
6. communication mode of communicating devices differs (RTU <-> ASCII),
7. startbit, stopbit or number of databits of communicating devices differs,
8. wrong checksum.

6. STANDARD BEGIN-TABLE FLUIDWELL PRODUCTS

6.1. GENERAL

Not all functions are implemented in every Fluidwell product. This may vary per device and product-family. It is even possible that a product with the same product-number has different software and/or hardware.
The function-codes 3 and 16 (read and write table data) are implemented in every Fluidwell-product.
To make it possible to access any product (also future products) via this protocol, the table-begin as described below is mandatory. With this standard begin, communication software can conclude which other functions are implemented.

Nr.	VARIABLE NAME	LENGTH	NOT WRITABLE
00	Kind of product	2 bytes	never writable
01	Model number	2 bytes	never writable
02	Serial number product	4 bytes	never writable
03	Hardware options	2 bytes	never writable
04	Software options	2 bytes	never writable
05	Protocol options	2 bytes	never writable
06	Software version	2 bytes	never writable
07	Protocol version	2 bytes	never writable
08	Display languages	2 bytes	never writable
09	Bus address	1 byte	during local programming
10	Baud rate	2 bytes	during local programming
11	Communication mode	1 byte	during local programming
12	Comm. index use	2 bytes	always writable
13	Comm. index	1 byte	always writable
14	Comdelay	2 bytes	
15	general status	2 bytes	never writable
16	indexed status	1 byte	never writable
17	reserved		
18	reserved		
19	reserved		
20	Inputs		never writable
21	Outputs		never writable
22	Beep command	1 byte	always
23	Time and date	6 bytes	during local programming
24	Print command		not implemented
25	Reboot	2 bytes	always writable
26	reserved		
27	reserved		
28	reserved		
29	reserved		
30……..Further product dependant variables only!!			

Table 5: Standard begin-table Fluidwell products.

Nr.	VARIABLE EXPLANATION	
00	Kind of product	Batch controller, PC, printer etc.
01	Model number	Type of product
02	Serial number	Serial number binary coded
03	Hardware options	Extra hardware mounted, inputs / outputs e.g.
04	Software options	Extra software functionality, temperature compensation e.g.
05	Protocol options	Protocol extensions, also other protocol implementation e.g.
06	Software version	Product software version
07	Protocol version	Communication software version
08	Display languages	Code for the display languages available
09	Bus address	The communication address
10	Baud rate	Communication speed 1200=0, 2400=1, 4800=3, 9600=4
11	Communication mode	BUS_ASCII=0, BUS_RTU=1, OFF=2 Warning: can not be enabled remotely
12	Comm. index use	User options of the comm. index: 00 : no features 01 : auto increment after each usage 02 : auto decrement after each usage 04 : auto functions also during broadcast access
13	Comm. index	Pointer for indexed values
14	Comdelay	{ 0..9999 } In milliseconds!!
15	General status	Device status, general alarm etc. Meaning of bits [msb…lsb] for 100-series: 0x0001: PCF error 0x0002: EEPROM error 0x0004: data corrupted (communication) 0x0008: receive buffer full (communication)
16	Indexed status	status per liquid; busy, overrun, pause etc, new data
17	Reserved	
18	Reserved	
19	Reserved	
20	Inputs	See device description
21	Outputs	See device description
22	Beep command	One byte in units of 0.1 seconds.
23	Date time	Order used: year,month,day,hour,minutes,seconds All bytes binary coded.
24	Print command	Not implemented yet
25	Reboot	0xA50F (warm reboot) or 0x5AF0 (cold reboot). A warm reboot resets the unit, but all programmed variables are stored. A cold reboot resets the unit and all of its programmed variables. Note that also the accumulated total will be lost!
26	Reserved	
27	Reserved	
28	Reserved	
29	Reserved	
30……...Further product dependant variables only!!		

Table 6: Variable description.

6.2. INDEXED VARIABLES

Some Fluidwell products use an array of multiple variables, for example a linearisation table which holds multiple frequency-correction pairs or an array of multiple batch sizes. To make easy use of such an array, a communication index needs to be used.
The variable that will be returned depends of the value of this index. When a larger part of the array needs to be read or written it is possible to auto-increment or decrement the index, meaning that after each successful read/write the index will point to the next or previous variable in the array. This way complete arrays can be accessed fairly easily.
Note that if index=0 the first array member will be returned, so an array of 5 variables uses indices 0 through 4. Be aware that this index can only be used for variables which are marked as indexed in this manual, it is never possible to use the index to read/write multiple un-indexed variables!

Example
Assume we are communicating with a unit that uses linearisation and has a bus-address of 01. We are willing to read the complete linearisation table from it. Of course we can initialise the index, read the variable, increment the index, write the new index, read again and repeat this procedure until the whole table is read, however it is more easily to use the next steps:

1. *Initialise the communication index to the first array-member, we need to write:*
   01 10 00 0D 00 01 02 00 ***00*** *DF (LRC checksum! ASCII-mode only!)*
2. *Set the communication index use to auto-increment, we need to write:*
   01 10 00 0C 00 01 02 00 ***01*** *DF*
3. *Now we are ready to read the first array-member, for our example the linearisation table start at address 0400h and each entry contains 6 bytes:*
   01 03 ***04 00*** *00 03 F5*
4. *The index is now auto-incremented to 01, so now we only need to repeat step 3. until the whole table is read!*

For safety reasons the auto-increment/decrement function is disabled for Fluidwell products which are used in broadcast-mode (e.g. FW/0300 series), but can still be enabled if desired (see standard begin-table Fluidwell products, variable 12)

7. GENERAL REMARKS

7.1. GENERAL

All numeric variables are binary coded. There are variables with different lengths:

byte	: 8 bits
word	: 16 bits
double word	: 32 bits

When transmitting multibyte variables, the msb is transmitted first.
In tables, the length of variables is given in bytes.
Several on/off variables use "0" for off-position. A value unequal "0" indicates that the concerning variable is "on".

Variables that indicate a decimal position contain as value the number of digits behind the decimal point. This corresponds with the negative value of the exponent from 10 from the number to be multiplied with the concerning variable.

Example:
K-factor = 1031, decimal position K-factor = 2.
This means that the K-factor has two digits behind the decimal point; the K-factor is 10.31
This corresponds with the multiplying with 0.01 = 10E-2. The last 2 corresponds (but without minus sign) with decimal position 2.

Time variables - if no other time unit indicated - use a time base of 0.1 second.
If the programmed overrun-time in a batch controller is 6.5 seconds for example, then the "communication-value" will be 65.

Some variables are always write protected such as the product and serial number. It is always allowed to read a variable.

INDEX:

NOTES:

NOTES: