

## Flow Computer Device Series enCore FC FC1, MC1

---

Manual  
Modbus AFB

---

## Contact

Elster GmbH (manufacturer)

Steinern Straße 19-21

55252 Mainz-Kastel/Germany

Telephone: +49 6134 605-0

E-mail: [info@elster.com](mailto:info@elster.com)

Website: [www.elster-instromet.com](http://www.elster-instromet.com)

Technical Support Flow Computers and Gas Analyzers

Telephone: +49 231 937110-88

E-mail: [ElsterSupport@Honeywell.com](mailto:ElsterSupport@Honeywell.com)

Website: [www.elster-instromet.com](http://www.elster-instromet.com)

# Contents

1	About these Instructions	5
2	Functional description	7
2.1	A summary of Modbus protocol variants	8
2.1.1	Modbus RTU/Modbus ASCII (serial)	10
2.1.2	Modbus TCP	10
2.2	Example scenarios	11
2.2.1	Data transmission via Modbus RTU/Modbus ASCII	11
2.2.2	Data transmission via Modbus TCP	12
3	Special feature of the AFB: Register and archive areas	13
3.1	Register areas / archive areas and registers – a comparison	14
3.2	Defining register areas or archive areas	16
3.3	Providing user-defined registers (register area)	23
3.4	Reading archives via Modbus (archive area)	24
3.5	Checking export registers on the device	27
4	Display and operation	30
4.1	Displays in overview	31
4.2	Displays in detail	32
5	FAQs	34
5.1	What is the difference between the normal and the expert modes in the Modbus AFB?	34
5.2	In normal mode, how can I parameterize a FC1 as a Modbus slave to COM2 in a few steps?	35
5.3	How do I synchronize the system time via Modbus? (Normal mode)	38
5.4	How can I resolve a Protocol error (message)?	40

6	Appendix	42
6.1	Nomenclature	42
6.2	Supported Modbus telegram types	43
6.3	Data description of the register formats	43
6.3.1	Byte order, word order and dword order	44
6.3.2	Supported data types	44
7	Bibliography	47
8	Index	48

# 1 About these Instructions

The enCore FC manual is modular. You will find an overview of the enCore/enSuite concept and the structure of the handbook, safety and risk instructions and the text labeling, refer to ⇒ the "Operating Instructions" of the enCore FC.

This volume describes the basic functionality and operation of the Modbus AFB.




## The Modbus AFB in the device series enCore FC

The Modbus AFB is available for all devices in the enCore Flow Computer series of devices (in short: enCore FC). The functions which support a specific device in detail depend on the device type, and are described in detail in the online help.

The significance of the individual parameters is extensively documented in the enSuite online help, which is why parameterization is only addressed as an example in this document. Parameters intended for special applications are also described in the online help.



## Accessing the online help

In enSuite you can access the general help via the menu item **Help** –  [Show online help](#). You can open context-sensitive help directly in the parametrization window from the desired branch by pressing **[F1]**.

This part of the documentation is intended at specialist personnel who are responsible for the service activities of the following tasks after the successful assembly of the device and installation of the current enSuite version on PC:

- adaptation of the device parametrization to the measuring task
- test of all data points and commissioning
- other service activities

The figures in these instructions are used to demonstrate what is being explained, so they may deviate depending on the configuration of the device and enSuite.

NOT UP-TO-DATE  
www.docuthek.com

## 2 Functional description

This manual describes the function and operation of the `Modbus AFB` in enCore devices. The AFB is based on the Modbus protocol, which enables data exchange with external devices.<sup>1</sup> As such, the AFB can be advantageously used in order to enable the exchanging of data with application-independent additional devices, such as station control technology and remote-control devices, in an application-specific manner.

The `Modbus AFB` maintains user-defined registers, which can be flexibly linked to enCore internal values during parameterization. You can freely parameterize Modbus addresses, data formats, byte order, and so on. You can also read archives via Modbus. Communication takes place via a TCP/IP network or a serial connection (point-to-point or bus).



### **Digression: Connection of intelligent measurement devices via Modbus scripts**

The Modbus protocol is also frequently used to connect measurement devices such as ultrasonic meters and process gas chromatographs ("intelligent measurement devices") to enCore devices. This is where manufacturers specify the specific details for Modbus communication and its evaluation as e.g. register assignment. Elster implements these manufacturer specifications and typical register assignments as scripts. The scripts for the most common intelligent measurement devices are already included in enCore devices in their delivery state.

For connecting a PGC or USM, select the necessary script when parameterizing for the corresponding serial or LAN interface. Any further Modbus parameterization is not necessary. A `Modbus AFB` is only needed for intelligent measuring devices in exceptional situations, namely only if a script does not provide all of the necessary registers.

---

<sup>1</sup> The implementation of the Modbus protocol is based on the respective Modbus specifications. (⇒ Appendix 7 [Bibliography](#), p. 44)

The connection of intelligent measurement devices, such as gas quality measurement devices or ultrasonic gas meters, via device-specific Modbus scripts, is not part of this manual.

⇒ See the "Basic System with SFBs" volume of the FC manual.

The Modbus protocol is a communication protocol that allows master/slave and/or client/server communication between devices connected in a network. This relates to an open protocol developed by Gould-Modicon in 1979 initially for serial communication between programmable logic controllers. Since 2007, the Modbus TCP version is part of the IEC 61158 standard.

enCore devices support the three variants of the Modbus protocol:

- Modbus ASCII
- Modbus RTU (**R**emote **T**erminal **U**nit)
- Modbus TCP

In general, each Modbus AFB enables you...

... in serial communication to respectively

- ... depict 1 Modbus slave ( $\cong$  1 device) with a 1 slave ID.
- ... depict 1 Modbus master ( $\cong$  1 device) which can address up to 10 slaves via their slave-ID.

... via TCP/IP to

- ... depict 1 Modbus server ( $\cong$  1 device) with IP address and port, to which up to 10 clients can have access at the same time.
- ... depict 1 Modbus client ( $\cong$  1 device) which can address a number of servers over their IP address and port.

## 2.1 A summary of Modbus protocol variants

The Modbus protocol is based on the master/slave or client/server architecture. It is a so-called single-master protocol, that is, the master or client controls the entire transmission and monitors, for example, any timeouts that might occur. Connected devices may only send telegrams if



requested by the master or client. Telegrams detected as erroneous are generally rejected and re-requested after a qualifying period.

For all protocol variants, the function code, start address and number of registers belong to the Modbus data. The data are found in one or more registers, each with a 16 bit length; you parameterize the assignment of the register with user data when commissioning.

The general settings of the serial and/or LAN interface of the enCore device are parameterized for the desired board in normal mode under **Start (I/O configuration)** and in expert mode in the **Basic System – I/O**. You parameterize the communication settings for the master and/or client and slave or server in the `Modbus AFB`.

⇒ The parameterization of the communication settings is not described in detail in the manual, as it is detailed in the online help.



#### Parameterizing a slave or server in normal mode

Note that in normal mode, in the case of serial communication, you only parameterize one (local) slave and in the case of TCP/IP, you can only parameterize one (local) Modbus server. In order to parameterize a Modbus master and/or a Modbus client, switch to expert mode.

⇒ [FAQ 5.1 What is the difference between the normal and the expert modes in the Modbus AFB?](#) (p. 34)

### 2.1.1 Modbus RTU/Modbus ASCII (serial)

The Modbus RTU and Modbus ASCII variants exhibit minor differences in framing. While Modbus ASCII uses the ASCII character set during transmission, transmission in the case of Modbus RTU is binary. Both use a serial asynchronous point-to-point data connection via RS232C or a multi-point connection via RS422 or RS485.

Each slave requires a unique ID (**Slave-ID**), through which the master can identify it on the bus – the ID zero (0) is reserved.

The serial communication settings for Modbus RTU and Modbus ASCII must be defined identically on the transmitter and receiver sides. This includes driver mode, baud rate, number of data bits (8 in the case of Modbus RTU, 7 with Modbus ASCII), parity and stop bits.

### 2.1.2 Modbus TCP

Modbus TCP uses a LAN interface of the CPU or ESER4 and the TCP/IP transmission protocol. The Modbus data are transmitted as a TCP/IP packet. By default, port 502 is reserved for the Modbus TCP server.

Each data exchange of a telegram takes place point-to-point wherein one of the two subscribers on the Modbus communication is a client and the other is a server. As the IP connection is disconnected after each transmission, a transmission can take place with another subscriber on the network afterwards.

The addressing of the Modbus devices takes place via the IP address of a device. Actually, a further ID such as the slave ID for Modbus RTU / Modbus ASCII is not required; however, it can be parameterized as an option.

## 2.2 Example scenarios

### 2.2.1 Data transmission via Modbus RTU/Modbus ASCII

The following example shows the addressing on a serial bus via Modbus RTU or Modbus ASCII. The prerequisite for the data exchange is that each slave has a unique slave ID which is stored in the master:

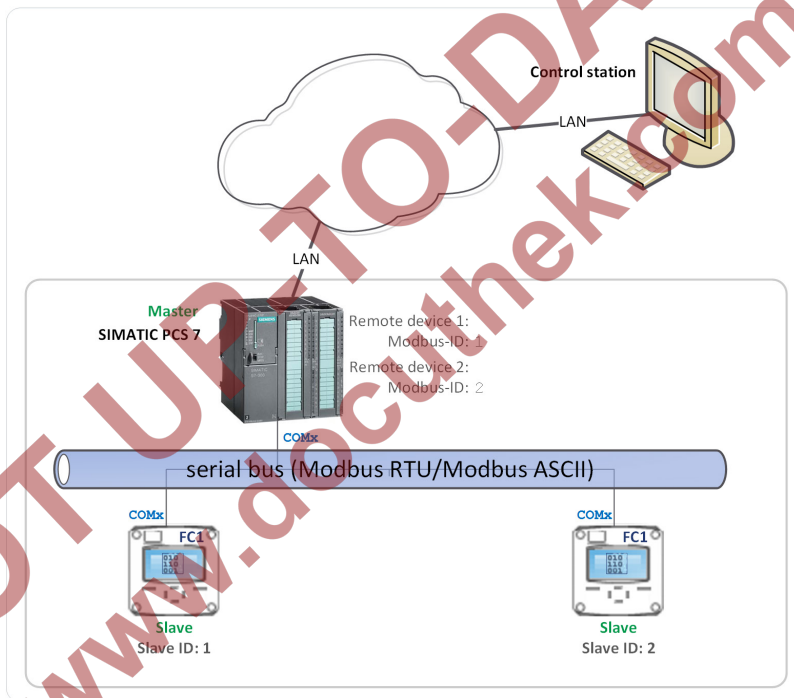


Fig. 2-1: Data transmission via Modbus RTU/Modbus ASCII – example

### 2.2.2 Data transmission via Modbus TCP

The following example shows the addressing in a TCP network via Modbus TCP. A client addresses a server via its (unique) IP address and port number. The prerequisite for the data exchange is that this information is stored on both the server and the client:

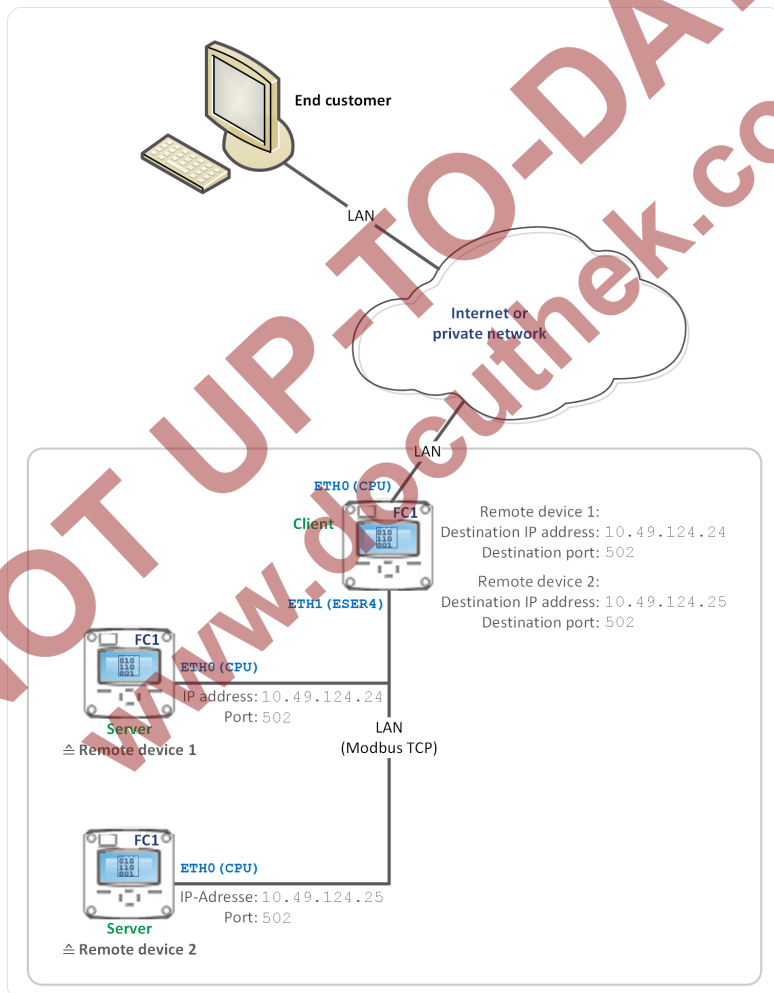


Fig. 2-2: Data transmission via Modbus TCP – example

### 3 Special feature of the AFB: Register and archive areas

The Modbus specification defines the data frame for data transmission, however it does not describe data formats. Therefore, in practice, there are many different formats in use.

In order to be flexible in the data description and at the same time to keep the parameterization as simple as possible, the Modbus AFB maintains register and archive areas in addition to the individual registers. Each register is thereby allocated precisely to one area. This has the advantage that all registers of an area inherit the general properties of this area (for example, the data formats).

Parameterization in the Modbus AFB always takes place in two stages:

(1) Define the register or archive area.

First, you create an area and define properties, which are valid for all registers in this area.

(2) Parameterize the register.

You then define the individual registers for this area and set the specific properties of the individual registers.



#### Register start address according to the Modbus specification with an offset of minus 1

Modbus uses the start address of a register to read it or to write. Make sure that the register numbers always start with 1, but that the start address in the case of Modbus is always 1 smaller than the register number. This means that in the Modbus telegram the start address is always the "Register number – 1". As such, the Modbus AFB behaves in accordance with the Modbus standard.

#### *For example:*

The current system time has the register number 4200 and correspondingly sends 4199 as the Modbus start address.

This chapter is divided into the following sections:

- ⇒ [3.1 Register areas / archive areas and registers – a comparison](#) (p. 14)
- ⇒ [3.2 Defining register areas or archive areas](#) (p. 16)
- ⇒ [3.3 Providing user-defined registers \(register area\)](#) (p. 23)
- ⇒ [3.4 Reading archives via Modbus \(archive area\)](#) (p. 24)



### 3.1 Register areas / archive areas and registers – a comparison






This section highlights the differences between register areas, archive areas and registers.



#### **Explanation of terms: Data Import and Data Export**

In order to avoid misunderstandings, the terms "writing" and "reading" of registers are avoided in this manual. This is because the meaning depends on the communication mode (master/client and/or slave/server). In order to display the direction of communication, we use the terms data import and data export:

-  **Data import**  
This is where the Modbus data are transmitted from an external device to the enCore device.
-  **Data export**  
This is where the Modbus data are transmitted from the enCore device to an external device.

Register area/Archive area	Register
<p><b>Number</b></p> <ul style="list-style-type: none"> <li>(Master/Client) up to 10 archive areas/register areas for each remote device</li> <li>(Slave/Server) up to 30 archive areas/register areas for each local device</li> </ul>	<p><b>Number</b></p> <ul style="list-style-type: none"> <li>overall 2,000 registers max. (beyond all register areas and archive areas)</li> </ul>
<p><b>2 types of areas</b></p> <ul style="list-style-type: none"> <li><b>Register area</b> <ul style="list-style-type: none"> <li>generic for user-defined registers</li> <li>both for local devices (slave/server) as well as remote devices (master/client)</li> <li>supports  Import register and  export register</li> </ul> </li> <li><b>Archive area</b> <ul style="list-style-type: none"> <li>specifically for reading archive groups of the <b>User archive AFB</b></li> <li>only for a local device (Slave/Server)</li> <li>supports only  export registers</li> </ul> </li> </ul>	<p><b>2 types of registers</b></p> <ul style="list-style-type: none"> <li> <b>Import register</b> A value is imported into the register via Modbus and mapped for further processing in the enCore device.</li> <li> <b>Export register</b> The enCore device stores a value in the export register and makes it available via Modbus.</li> </ul>
<p><b>Each area</b></p> <ul style="list-style-type: none"> <li>... can manage a variable number of registers – only restriction: The number of 2,000 registers should not be exceeded.</li> </ul>	<p><b>Each register</b></p> <ul style="list-style-type: none"> <li>... is always allocated to a register or an archive area, and inherits the general properties of its area and data type.</li> </ul>
<p><b>For each area</b></p> <ul style="list-style-type: none"> <li>... you define a general data description which all registers in this area inherit.</li> </ul>	<p><b>For each register</b></p> <ul style="list-style-type: none"> <li>... you define the special properties of the register <b>Name</b> and <b>register number</b>.</li> </ul>


Register area/Archive area	Register
<ul style="list-style-type: none"> <li>... you define the general data description with the following properties: <ul style="list-style-type: none"> <li><b>Register length:</b> 16 bit, 32 bit, 64 bit</li> <li><b>Byte order / Word order / Dword order:</b> HI/LO (Big-Endian), LO/Hi (Little-Endian)</li> <li><b>Data formats</b> for: measurements, counters and event counters, bit strings and status as well as time and date</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>... you define the data type specific properties, which can vary for the import and export register.</li> </ul> <p>Based on the example of an <b>import measurement:</b></p> <ul style="list-style-type: none"> <li><b>Physical unit</b></li> <li><b>Scaling factor</b> and <b>scaling offset</b></li> <li><b>Lower cut off</b> and <b>Upper cut off</b></li> </ul>
<p>For each register area...</p> <ul style="list-style-type: none"> <li>... you can specify an <b>Off limit handling</b> for measurements.</li> </ul>	
<p>For each archive area...</p> <ul style="list-style-type: none"> <li>... you can accurately map 1 archive group of the  User archive AFB.</li> <li>... you define whether the archive group is mapped on the Modbus register in a way that is address-based or register-based.</li> </ul>	

Table 3-1: Register and archive areas vs. individual registers – a comparison

## 3.2 Defining register areas or archive areas


















For each register area and archive area, you first define the general properties and data formats that are valid for all registers in this area.

Archive areas additionally have archive-specific properties. These are described in ⇒ chapter [3.4 Reading archives via Modbus \(archive area\)](#) (p. 24).






## Procedure in enSuite

### Prerequisites

- The communication settings in  **I/O configuration** and  **Modbus <x>** are parameterized.
- *Only for archive areas*  
The desired archive groups are already parameterized in the  **user archive AFB**.
- The communication settings in  **I/O configuration** and  **Modbus <x>** are parameterized.
- *for a local device (slave or server)*  
In the parameterization  the **Modbus AFB** folder is opened:  
 **<Device>** – [ **<Group>** –]  **Modbus [<x>]** –  
**Communication mode: Slave or server** –  **Local device** –  
 **Register areas**  
OR  
*for remote devices (master or client)*  
In the parameterization  the **Modbus AFB** folder is opened:  
 **<Device>** – [ **<Group>** –]  **Modbus [<x>]** –  
**Communication mode: Master or client** –  **Remote devices** –  
 **Remote device**.

To add a new area...

- ▶ ... switch to the **Parameter** tab.
- ▶ In the **Register areas**, add a new area using the  plus sign.
- ✓ In the parameterization window, a new folder  **Area <x>[: Register area]** has been added.
- ▶ In the folder  **Area <x>[: [Register area]** open the tab **Parameter**.
- ✓ This is where you define the general properties and data formats for this area.
- ▶ *Only for a local device (slave or server)*  
Using the drop-down list **Area <x>** you establish whether you define a **register area** or an **Archive area**.
- ✓ Additional parameters are blended in for the archive area only.

- With the **Register length** drop-down list, you establish whether the data is transferred in the form of **16 bit**, **32 bit** or **64 bit** registers. *Default* is **16 bit**.

In order to determine the position of the high- and low-order bytes for the data units byte (8 bit), word (16 bit), and dword (32 bit),...

- ... select the following entry in the drop-down lists **Byte order**, **Word order** and **Dword order**: ...
  - ... **HI/LO (big endian)**  
if the highest-value byte is on the start address.
  - ... **LO/HI (little endian) (default)**  
if the lowest-value byte is on the start address.



#### How to parameterize certain byte orders of a register or archive area

Order used  
(e.g. by a GC)

Parameterizing in enSuite:

1234

**Byte order** HI/LO (big endian)  
**Word order** HI/LO (big endian)  
**Dword order** HI/LO (big endian)

4321

**Byte order** LO/HI (little endian)  
**Word order** LO/HI (little endian)  
**Dword order** LO/HI (little endian)

2143

**Byte order** LO/HI (little endian)  
**Word order** HI/LO (big endian)  
**Dword order** HI/LO (big endian)

3412

**Byte order** HI/LO (big endian)  
**Word order** LO/HI (little endian)  
**Dword order** LO/HI (little endian)

- In the drop-down list **Register format (measurements)** you determine the data type with which measurements are mapped in registers:
  - **Word**  
as an integer with word length 16 bit and value range  $-2^{15} \dots 2^{15} - 1$

- **DWord**  
as an integer with word length 32 bit and value range  $-2^{31} \dots 2^{31} - 1$
  - **QWord**  
as an integer with word length 64 bit and value range  $-2^{63} \dots 2^{63} - 1$
  - **Float S5**  
as a floating-point number 32 bit (Siemens data type)
  - **Float** (default)  
as a normalized floating-point number 32 bit (single precision)  
according to IEEE 754
  - **Double**  
as a normalized floating-point number 64 bit (double precision)  
according to IEEE 754
- In the drop-down list **Register format (counter/event counter)** define the data type with which counters and event counters are mapped in registers:
- **Word**  
as an integer with word length 16 bit and value range  $-2^{15} \dots 2^{15} - 1$
  - **DWord** (default)  
as an integer with word length 32 bit and value range  $-2^{31} \dots 2^{31} - 1$
  - **QWord**  
as an integer with word length 64 bit and value range  $-2^{63} \dots 2^{63} - 1$
  - **Float S5**  
as a floating-point number 32 bit (Siemens data type)
  - **Float**  
as a normalized floating-point number 32 bit (single precision)  
based on IEEE 754
  - **Double**  
as a normalized floating-point number 64 bit (double precision)  
according to IEEE 754
  - **BCD8**  
as a binary coded decimal number 8 figures, word length 32 bit and  
value range  $0 \dots 10^9 - 1$

- **BCD16**

as a binary coded decimal number 16 figures, word length 64 bit and value range 0 ...  $10^{17} - 1$

► (since AFB version 03-12)

In the drop-down list **Register format for counters (fractional part)** define the data type with which the fractional part of counter values (these values are always  $< 1$ ) are mapped to registers:

- **Float**

as a normalized floating-point number 32 bit (single precision) based on IEEE 754

- **Double**

as a normalized floating-point number 64 bit (double precision) according to IEEE 754

► In the drop-down list **Register format (bit strings/status)** define the data type with which measurements are mapped in registers:

- **Word**

(default)

as an integer with word length 16 bit and value range  $-2^{15} \dots 2^{15} - 1$

- **DWord**

as an integer with word length 64 bit and value range  $-2^{63} \dots 2^{63} - 1$

- **QWord**

as an integer with word length 64 bit and value range  $-2^{63} \dots 2^{63} - 1$

- In the drop-down list **Register format (time/date)** define the data type with which measurements are mapped in registers:



#### Meaning of abbreviations for date and time

- ss  $\triangleq$  seconds (0 to 59)
- MM  $\triangleq$  minutes (0 to 59)
- hh  $\triangleq$  hours (0 to 23; wherein: 0 = midnight)
- WD  $\triangleq$  weekday (1 to 7; wherein: 1 = Sunday)
- dd  $\triangleq$  day (1 to 31)
- mm  $\triangleq$  month (1 to 12)
- yy  $\triangleq$  year starting from the 20yy (2000 to 2099)
- xx  $\triangleq$  (reserved)
- .  $\triangleq$  (reserved)
- 0  $\triangleq$  filler Null (0)

- **QWord**

as ssMMhhWDddmmyyxx with a register with a 64 bit word length

- **Word (mm,dd,yy,hh,MM)**

as mm, dd, yy, hh, MM with five registers, each with a 16 bit word length

- **Word (mm,dd,yy,hh,MM,ss)**

as mm, dd, yy, hh, MM, ss with six registers, each with a 16 bit word length

- **Word (yy,mm,dd,hh,MM,ss)**

as yy, mm, dd, hh, MM, ss with six registers, each with a 16 bit word length

- **Word (dd,mm,yy,hh,MM,ss)**

as dd, mm, yy, hh, MM, ss with six registers, each with a 16 bit word length

- **Float (mmddyy.0,hhMMss.0)**

as mmddyy.0, hhMMss.0 with two registers, each with a 32 bit word length

- **Float (yy,mm,dd,hh,MM,ss)**

as yy, mm, dd, hh, MM, ss with six registers, each with a 32 bit word length

- **Unix**

Number of seconds since 1.1.1970 00:00:00 with a register with a 32 bit word length – meaning of time (local or UTC) depending on the application

► *(Only for register area since AFB version 03-12)*

With the parameter **Import register type** you determine with which command the master reads data from the slave device. The AFB supports the following commands:

- **Holding register (0x03)**

The master reads the registers from the slave with function code 0x03 from the address range 40001 to 49999.

- **Input register (0x04)**

The master reads the registers from the slave with function code 0x04 from the address range 30001 to 39999.

► *(For measurements in register areas only)*

In the drop-down list **Off limits handling** you specify the behaviour in case the parameterized **Lower cut off** or **Upper cut off** limit is violated by a measurement value – this parameter is only evaluated if an upper and/or lower range limit is parameterized.

You have following options:

- **set to limit (default)**

Depending on whether the upper or the lower range limit is violated, the corresponding limit value is written into the register.

- **reject with error**

*(for import register only)* The value that violates the upper or lower limit is rejected. Instead, the error code 0x03 (invalid value) is sent via Modbus.



### Register assignment

In the case of enCore devices you define the register assignment with the **Register length** and **Register format <data type>** parameters. Application data are transmitted in one or more consecutive registers with the parametrized **Register length** according to the register's data format.

For example in the case of a register length of 16 bit, the transmission of 64 bit data takes place in 4 consecutive 16 bit registers.

The further procedure:

for register areas

⇒ [3.3 Providing user-defined registers \(register area\)](#) (p. 23)

for archive areas

⇒ [3.4 Reading archives via Modbus \(archive area\)](#) (p. 24)

## 3.3 Providing user-defined registers (register area)

With the exception of archive data, you can manage user-defined registers conveniently in one or more register areas, as required. You define the necessary import and export registers for each register area in a second step.














### Pre-assignment of the first Modbus list in the case of default parameterization

If you create a default application with (at least) one Modbus list using the wizard, the first list already contains a register area, which is pre-assigned with export registers for typical values of the flow conversion. These are suggestions from Elster. Adapt these registers to your requirements.

If you have created two lists using the wizard, the second **Modbus AFB** does not contain any predefined registers.

## Procedure in enSuite

### Prerequisites

- The register area is already defined.  
⇒ [3.2 Defining register areas or archive areas](#) (p. 16)
- *for a local device (slave or server)*  
In the parameterization  the Modbus AFB folder is opened:  
 **<Device>** –  **<Group>** –  **Modbus [<x>]** –  
**Communication mode: Slave or server** –  **Local device**  
OR  
*for remote devices (master or client)*  
In the parameterization  the Modbus AFB folder is opened:  
 **<Device>** –  **<Group>** –  **Modbus [<x>]** –  
**Communication mode: Master or client** –  **Remote devices** –  
 **Remote device <x>**

In order to edit a register list ...

- ... switch to the **Register list** tab.

On this tab, you can conveniently manage registers for the individual register areas. This is where you can add new import and export registers, move available registers, change or delete them.

For export registers of the measurement or counter type, enSuite provides a convenient opportunity to transmit a cyclical usage data or counter readings for each Modbus on the Interval tab.

⇒ The register types and register details that the AFB supports are detailed in the online help.

## 3.4 Reading archives via Modbus (archive area)

The archive area is optimized for editing archive groups of the **User archive AFB**. You can link an archive area to one or more archive groups of the **User archive AFB**.

Optionally you can read an archive channel based on an address or purposefully register-based:



- In the case of address-based access, the `Modbus AFB` maps an archive channel (or its most recent part) on the Modbus register 1:1. At the same time, you can parameterize the maximum archive depth with which archive channels are transmitted for each Modbus. In this method the AFB calculates how many Modbus registers an archive entry needs.
- In the case of register-based access, it is possible (as opposed to address-based mode) to specifically read an archive entry of the archive channel via an index (vs. a fixed parameterized number of the most recent archive entries). With consecutive inquiries (each with an altered index) it is possible, in this way, to read out the whole archive depth of an archive channel step by step without providing a Modbus register for the entire archive depth. In doing so one must ensure that the archive channel is not updated during the whole reading process. For example, by always starting the reading process of an hourly values archive, after the full hour.

### Procedure in enSuite

In order to link the archive area with an archive group of the  User archive AFB,...

- ▶ ... select the desired archive group in the drop-down list **Corresponding archive group**.

The AFB supports 2 different types which map the corresponding archive group on the Modbus register:

- ▶ In the drop-down list **Access mode**, select the entry...
  - ... **Address based**  
if you read a fixed parameterized number of the latest archive entries via Modbus.  
⇒ [A. parameterizing address based access](#) (p. 26)
  - ... **Register based**  
if you read archive entries of the archive channel in a targeted way via an index.  
⇒ [B. Parameterizing register-based access](#) (p. 26)

#### A. parameterizing address based access

- ▶ State the archive depth in the parameter **Number of archive records** – *default is 100.*
- According to the selected archive depth, the AFB uses a contiguous register area with ongoing register numbers for this archive channel. In doing so it takes account the register length and the data formats which you have parameterized for this archive area.
- ▶ Define the order in the parameter **Index method**:
  - **Push up**  
In the push up process, the most current archive entry in the reserved archive area is always transferred into the lowest register number and the oldest into the highest (assigned).
  - **Push down**  
In the push down process, the most current archive entry in the reserved archive area is always transferred into the highest register number and the oldest into the most recent (assigned).

#### B. Parameterizing register-based access

- ▶ In the parameter **Index register no.** of the index register enters – *default is 1.*
- ▶ In the parameter **Index register format** state the format of this register:
  - **Word/short** with a 16 bit word length
  - **DWord/int** with a 32 bit word length
- ▶ Define the order in the parameter **Index method**:
  - **Push up**  
In the push up process an index of zero (0) points to the oldest archive entry of the archive channel, an index of 1 to the second oldest, etc.
  - **Push down**  
In the push down process, it is the other way around. Here, an index of zero (0) points to the current archive entry, an index of 1 to the second most current one, etc.
  - **Sequence number**  
In this indexing process, you query an archive entry specifically via

its sequence number. Since the sequence number in an archive channel is unique, this reading process always delivers the same archive entry even upon repeated reading (as long as it is available in the archive channel).

- ✓ The general properties and data types of the archive area are defined.

In a second step, you define the necessary export register:

In order to edit a register list ...


- ... switch to the **Register list** tab.
  - ✓ On this tab, you can conveniently manage registers for the individual register areas. This is where you can add a new import and export register, move available registers, change or delete them.
- ⇒ The register types and register details that AFB supports are detailed in the online help.

### 3.5 Checking export registers on the device

With the AFB it is possible to check individual export registers directly in the device display – with the exception of the export registers of type **Export date and time**.

As soon as you log in to the device as a user with the user right **Change general system settings**<sup>2</sup>, you can activate or deactivate the test function.

---

<sup>2</sup> You can find the setting for the authorization in the parameterization under **<Device> – Basis System –  Users – Special user rights**.

## Procedure in enSuite

### Prerequisites

- You are logged in as a user with the user right **Change general system settings**.

To activate the test of an export register, ...

- ▶ ... switch to the detailed display of the desired export register via **Modbus – Main display**.
- In the **Modbus register** display, the **Test** line is displayed as a drop down list.
- ▶ Select the entry **on** in the drop-down list.
- The test mode is active and the value for the **Data object** can be edited. The text **Data object** is displayed in red font color during the test mode.

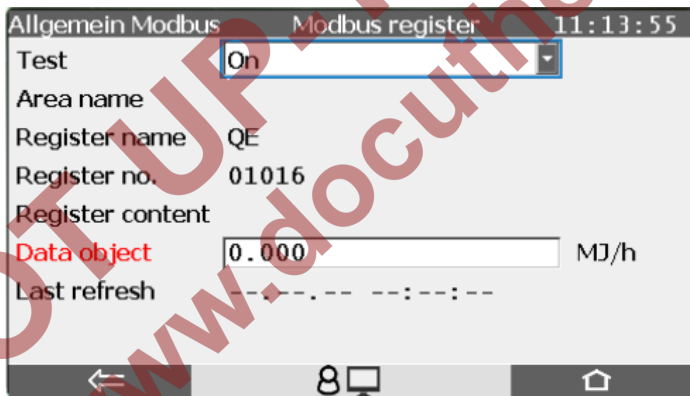


Fig. 3-1: Test function active for selected export register

- ▶ In the **Data object** field, enter the desired value that you want to simulate and confirm your entry with the button . The value ranges vary depending on the register type and assigned format.
- The specified value is provided via Modbus until you enter a new value or end the test function.



### Exit of test function

The test function ends in the following cases:

- As soon as you navigate to another page in the display, e.g. to simulate the value of another register.
- Logoff of the user, for example, due to the inactivity timeout.
- After a restart of the enCore device.

## 4 Display and operation

All register areas and archive areas are shown in the displays of Modbus AFB with the corresponding Modbus registers.



### Displays and navigation in the case of enCore FC devices

The general structure of the displays in the case of enCore FC devices and the basic navigation options are detailed in the "Operation Instructions" of the enCore FC in the chapter concerning display and navigation.

In general, in the operation of enCore FC devices, a distinction is made between hyperlinks and actions. While you use hyperlinks to navigate through the device's displays, you can use actions to perform specific functionalities. Hyperlinks and actions are highlighted in blue, both in the device and in the manual.

A list of the symbols and terms used in the following section can be found in the appendix (⇒ Appendix [6.1 Nomenclature](#), p. 42).

## 4.1 Displays in overview

The following figure shows the hierarchical structure and the navigation through the displays of the Modbus AFB:

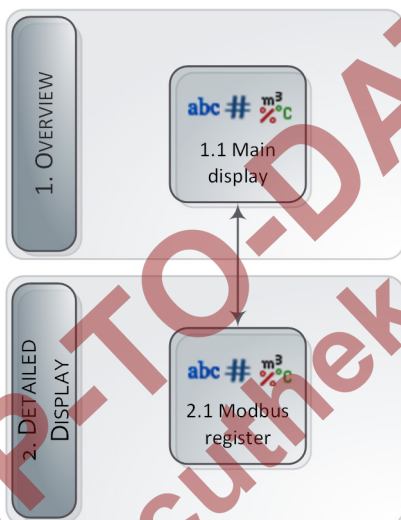


Fig. 4-1: Display – hierarchical structure

## 4.2 Displays in detail

The **Main display** of the **Modbus AFB** provides you with an overview of all (parameterized) Modbus areas of the **enCore** device and lists (in ascending order) all of the associated registers for the selected area. Optionally, you can switch to a detailed display for each register.



Modbus 3		Main display		14:51:09	
Modbus 3		TC:		225	
04806	900.000000	m³/h	/		
04808	000000001225.600	m³	/		
04810	09.02.18	14:51:08	/		
04814	3.716000	mol%	/		
04816	2.500000	bar	/		
04818	23.799999	°C	/		
04820	75895.898438	MJ/h	/		
04822	1698.856445	kg/h	/		

Fig. 4-2: Modbus **Main display** – example

In the example, the first Modbus registers of the register area **Q.Sonic plus** are displayed with their register numbers and the register content (decimal). To date, 84384 telegrams have been transmitted.

A character signals for each import or export register – in the example | (vertical line) – that the data of this register have been successfully transmitted. With the aid of the characters – (dash), \ (backslash), | (vertical line) and / (slash) a kind of turnstile is simulated that changes by a step each time the individual register is successfully transmitted.



## Overview display in detail

[1.1 Main display]		
<b>&lt;Drop-down list&gt;</b> contains all parameterized Modbus areas – for slave or server, only 1 unit is available	<b>TC: &lt;Counter&gt;</b> (Abbreviation <b>Telegram Counter</b> ) Telegram counter for the selected area	
<b>&lt;Reg. no.&gt;</b> Register number of the import register or export register [⇒ <b>2.1 Modbus register</b> ]	<b>&lt;Data object&gt;</b> Transmitted value formatted	→ \ → - → / A change of the character signals that the register has been successfully transmitted

[2.1 Modbus register]	
<b>Area name</b>	name of the register area (parameter <b>Name</b> )
<b>Register name</b>	name of the register (parameter <b>Name</b> )
<b>Register no.</b>	register number of the import register or the export register
<b>Register content</b>	transmitted value (hexadecimal)
<b>Data object</b>	transmitted value, formatted with unit
<b>Last refresh</b>	<date> <time>

## 5 FAQs

This chapter contains the most important settings and questions to help you with the tasks that occur on a regular basis:

- ⇒ [5.1 What is the difference between the normal and the expert modes in the Modbus AFB?](#) (p. 34)
- ⇒ [5.2 In normal mode, how can I parameterize a FC1 as a Modbus slave to COM2 in a few steps?](#) (p. 35)
- ⇒ [5.3 How do I synchronize the system time via Modbus?](#) (p. 38)
- ⇒ [5.4 How can I resolve a Protocol error \(message\)?](#) (p. 40)

### 5.1 What is the difference between the normal and the expert modes in the Modbus AFB?

#### Background

The normal mode serves to conveniently parameterize default applications. This is why not all parameters and export values are visible in normal mode, and not all editing functions are available. If necessary, you can switch to expert mode at any time, which offers *all* available parameters and functions. This is done via the menu item **Tools – Expert mode**. Please note that in case of changes you may not be able to switch back to normal mode.

#### Differences

With Modbus, the main difference between the normal and the expert mode is that in normal mode you can only parameterize enCore devices which take on the role of the slave and/or the server. However, in expert mode you establish for each instance of a `Modbus AFB` whether the enCore device is either the master or the client for the Modbus communication, or if it communicates as the slave or server.

The **Communication mode** parameter is only available in expert mode.

Another difference is the clear parameterization of Modbus lists in the normal mode by means of the three tabs **Settings**, **List of areas** and

**Register list.** The **Register list** is also available to you in expert mode. In expert mode, you parameterize with branches of the parameter tree as usual. As the parameterization of a Modbus list in the expert mode can be very branched and complex, we recommend that you only switch to the expert mode for the parameterization of a master and/or client.

5.2 In normal mode, how can I parameterize a FC1 as a Modbus slave to COM2 in a few steps?

**Background**

In normal mode, you can parameterize an enCore device as a Modbus slave in a few steps using the `Modbus AFB`. If you are creating a new parameterization for a default application, you already specify in the wizard whether no, one or two Modbus lists are available in normal mode. For each list, the wizard creates an instance of the `Modbus AFB` (**Modbus 1/Modbus 2**) in the parameterization.

If you use Modbus, a register area with export registers for typical values is predefined for a flow conversion in the Modbus list **Modbus 1**. In the case of two Modbus lists, the second list **Modbus 2** does not contain any predefined registers.

The following scenario exemplifies the parameterization in normal mode:

enCore device:	FC1 with default parameterization
Protocol type:	Modbus RTU
Interface:	COM 2 of the CPU
Transmission rate:	9600 baud
Transmission settings:	8 data bits, even parity, 1 stop bit
Slave ID:	2
Start of the register area:	3000
Export register:	base volume $V_{n,}$ base volume flow $Q_{n,}$ volume measurement alarm


Register length

16 bit

### Procedure in a few steps

In the following section, only the Modbus-relevant settings are described.


In the first step, you define the communication settings:

- ▶ Use the FC1 wizard to create a **Standard flow computer for gas** using the 1 Modbus list.
- ▶ Under **Start (I/O configuration)** mark the COM port **CH2** of the **CPU3** on the **I/O configuration** tab.
- ▶ In the drop-down list **CH2** in the bottom area, select the entry **COM-Port (single)**.
- ✓ The **Baud rate** has already been pre-set correctly to **9600** baud.
- ▶ Set the **Parity** to **Even**.
- ▶ In the parameter window open the folder  **Modbus 1** and switch to the tab **Settings**.
- ▶ Select the entry **serial** as **Interface**.
- ▶ Select the entry **CH2.COM-Port** as **Port**.
- ✓ **Modbus RTU** has already been correctly preset as **Protocol type**.
- ▶ Enter the value **2** as **Slave-ID**.


In the second step, enter the required export registers:

#### Procedure 1: Editing a pre-assigned register


By default, the list is already pre-assigned as **Register area** with a register length of **16 bit** on the tab **List of areas**.

- ▶ Switch to the **Register list** tab.
- ▶ Move the following export registers using the  arrow pointing upwards symbol as follows:
  - **V<sub>n</sub>** to position 1
  - **Q<sub>n</sub>** to position 2
  - **Volume measurement alarm** to position 3

In order to delete all other pre-assigned registers, ...

- ▶ ... mark the register in position 4, keep the [Shift] button pressed down and mark the last register in the list.
- ▶ By clicking on the  remove symbol, the highlighted registers will be deleted without any request for confirmation.

In order to assign successive register numbers, ...





- ▶ ... highlight the 3 export registers: **V<sub>n</sub>**, **Q<sub>n</sub>** and **Volume measurement alarm**.
- ▶ Click on the  number symbol and enter **3000** as a start number.
- ✓ The scenario has been parameterized.

## Procedure 2: Defining new registers


In order to delete all of the registers that have been pre-assigned by Elster, we use a small trick:

- ▶ Switch to the **List of register areas** tab.
- ▶ For the predefined **Area 1** select **Archive area** and then **Register area** once more.
- ✓ By default, the **Register length** is **16 bit**.

In order to define the new registers,...

- ▶ ...switch to the **Register list** tab.
- ▶ Open the export value window in the  **Stream1** folder.
- ▶ Pull the following export values into the register list using drag-and-drop:
  -  **Calculations.V<sub>n</sub>** to position 1
  -  **Calculations.Q<sub>n</sub>** to position 2
  -  **Volume measurement alarm** to position 3

In order to assign successive register numbers, ...

- ▶ ... highlight the 3 export registers: **V<sub>n</sub>**, **Q<sub>n</sub>** and **Volume measurement alarm**.
- ▶ Click on the  number symbol and enter the value **3000** as a start number.
- ✓ The scenario has been parameterized.

## 5.3 How do I synchronize the system time via Modbus? (Normal mode)

### Background

`Time Service`<sup>3</sup> is responsible for synchronizing or setting the device's internal system time. A prerequisite for this is that the desired time source(s) are stored in `Time Service` as import values in the parameters **Prim. external time source** and (optionally) **Sec. external time source**. In order to adjust the system time of a device, you can use NTP, Modbus or DSfG time as reliable time sources.

In order that an enCore device can adjust its system time to the Modbus time, in the first step, parameterize the corresponding import register in the `Modbus AFB` and in a second step, import the Modbus time in `Time Service` as an external time source.



We only recommend time adjustment via the Modbus time in cases where no external NTP source is available.

### Procedure in enSuite (in brief)


By way of an example, parameterization is shown using the following scenarios: A time server acts as the primary time source and provides its current time via Modbus.







#### Prerequisites


- The communication settings and the area of the Modbus list are already parameterized.

---



<sup>3</sup> The operation of the `Time Service` is described in detail in the FC manual "Basic System with SFBs".

In order to define the import register in the  **Modbus AFB** in a first step, ...


- ▶ Open the parameterization in normal mode.
- ▶ Switch to the **Register list** tab in the  **Modbus**.
- ▶ Mark the row *above* which the import register should be added.  
You can also subsequently move the item upwards with  and downwards with .
- ▶ Using , create a new import register.
- ▶ In the detail display, select the **Import date and time** entry in the **Register <x>** drop-down list.
- ▶ Using the **time mode** parameter, state whether the date and time of the import register is transmitted in **UTC** (**U**niversal **T**ime **C**oordinated) or in **local time**.
-  If the data are transmitted in **local time**, the enCore device first converts the date and time into UTC, before it provides the date and time in the export value  [**<Reg. No.>**]**<Name>: Import date and time.Time**.

By default, the device updates the export value  [**<Reg. No.>**]**<Name>: Import date and time.Time**, as soon as at least one value changes in the import register date and time.

In order to alternatively control the updating of this export value in a targeted manner via another register,...


- ▶ ... select the trigger register checkbox and enter the valid number of the import register.
-  Only when the value of the Trigger register changes from zero (0) to another value, is the export value  [**<Reg. No.>**] **<Name>: Import date and time.Time** updated.

In order to assign the Modbus time in **Time Service** in the second step, ...

- ▶ ... switch the folder **Time Service**, tab **Parameters**, section **General**.
- ▶ Select the parameterized Modbus register from the **Prim. external time source** drop-down list.
-  **Time Service** synchronizes the system time with the Modbus time, if a transfer is permitted according to the rules stored in Time Service.



## 5.4 How can I resolve a **Protocol error** (message)?

### Background

Masters (serial) or clients (TCP/IP) monitor the communication via Modbus. They establish a transmission error if either the Modbus communication is interrupted or the data import or data export is not successful for at least one register. In this case, the AFB generates a message with the same name  **Protocol error**. The message is displayed until all registers of this Modbus unit can be successfully imported or exported again.






Important parameters for error recognition include **Transaction timeout** (master) and **Error filter** (slave/server). In the case of **Transaction timeout** you determine a maximum time period for which a master waits for the answer from a slave before it generates a protocol error. In the case of the error filter, you specify for each slave/server the number of times the AFB tries to export or import a register (Round Robin method) before the master or client generates the message.

### Possible causes and remedies

-  The device is switched on, but is not reachable due to erroneous parameterization of the communication parameters.
- ▶ Ensure that the communication parameters in the  **Basic System** and Modbus AFB are coordinated between the master and slave and/or client and server.

OR


[only Slave/Server]

-  The error tolerance during data import or data export of a register is too low.
- ▶ On a test basis, increase the **Error filter** parameter under:
  -  **Modbus** –  **Communication mode: Master or client** –
  -  **Remote devices** –  **<Remote device x>**, tab **Parameter**, section **General**.



OR

[Master and slave]

-  At least one register is incorrectly or not at all parameterized on one of the two sides.
- ▶ Use the operation panel of the enCore device in the display belonging to the Modbus AFB and check which registers are not regularly refreshed. On both sides, check for errors in the respective parameterization concerning these register numbers.



OR

[Master and slave]

-  The cabling is defective or there is a cable breakage.
- ▶ Check the cabling and replace the damaged cable.

OR

[only master]

-  The remote device (slave or server) processes the telegrams slower than expected.
- ▶ On a test basis, increase the **Transaction timeout** of the master under:  
 **Modbus – Communication mode: Master or client**, tab **Parameter**, section **Interface**

# 6 Appendix

## 6.1 Nomenclature

The following symbols and names are used in the enCore FC device and in enSuite for measured data and calculated values in the context of Modbus AFB:

Symbol	Short form	Description
abc	<text>	parameterized register name or area name
#	<figure>	space holder for a figure, e.g. for a register number or a measurement
m <sup>3</sup> /h °C	<unit>	any unit, e.g. of a measurement

Table 6-1: Nomenclature

## 6.2 Supported Modbus telegram types

The Modbus protocol knows 18 different telegram types, of which enCore devices support the following – all other telegram types are either rejected or not evaluated.

Type	Description
0x03	("Read Holding Registers") Reads one or more associated registers of a remote device
0x10	("Write Multiple Registers") Changes one or more associated registers in a remote device
0x08	("Diagnostics") – only serial Provides diagnostic values for communication between a master and a slave
Additionally for a slave or server:	
0x04	("Read Input Registers") Reads one or more associated registers of a remote device
0x06	("Write Single Register") Changes a register in a remote device

Table 6-2: Supported telegram types

## 6.3 Data description of the register formats

The Modbus specification defines the general data frame for data transmission, however not the way in which data are mapped in the Modbus registers. Therefore, in practice, there are many different formats in use. The Modbus AFB describes different formats with the aid of register length, byte order and data type.

### 6.3.1 Byte order, word order and dword order

The byte order specifies the position of the high- and low-order bytes. In the AFB, you create the byte order for each register area for the data units byte (8 bits), word (16 bits) and dword (32 bit).

Byte order	Description
LO/HI	Low-Byte/High-Byte ("Little-Endian") ( <i>default</i> ) The high-order byte is located at the higher address.
HI/LO	High-Byte/Low-Byte ("Big-Endian") The high-order byte is located at the lower address.

### 6.3.2 Supported data types

**For measurements, counters/event counters, bit strings/status**

Name	Data type	Size
Word (short)	dual and hexadecimal numbers	16 bit
DWord/int	dual and hexadecimal numbers	32 bit
QWord (64-Bit)	dual and hexadecimal numbers	64 bit
Float S5	floating point number (Siemens data type)	32 bit
Float	normalized floating point number, single precision according to IEEE 754	32 bit
Double	normalized floating point number, double precision according to IEEE 754	64 bit
BCD8	binary coded decimal number	8 bit
BCD16	binary coded decimal number	16 bit

**For date and time:****Meaning of abbreviations for date and time**

- ss  $\triangleq$  seconds (0 to 59)
- MM  $\triangleq$  minutes (0 to 59)
- hh  $\triangleq$  hours (0 to 23; wherein: 0 = midnight)
- WD  $\triangleq$  weekday (1 to 7; wherein: 1 = Sunday)
- dd  $\triangleq$  day (1 to 31)
- mm  $\triangleq$  month (1 to 12)
- yy  $\triangleq$  year starting from the 20yy (2000 to 2099)
- xx  $\triangleq$  Validation code; ignored for import registers (only on the grounds of compatibility with the FC2000), and always has the value 1 in the case of export registers
- .  $\triangleq$  (reserved)
- 0  $\triangleq$  filler Null (0)

Name	Data type	Word length
<b>QWord</b>	ssMMhhWDddmmyyxx	1 register of 64 bit
<b>Word (mm,dd,yy,hh,MM)</b>	mm, dd, yy, hh, MM	5 registers of 16 bit each
<b>Word (mm,dd,yy,hh,MM,ss)</b>	mm, dd, yy, hh, MM, ss	6 registers of 16 bit each
<b>Word (yy,mm,dd,hh,MM,ss)</b>	yy, mm, dd, hh, MM, ss	6 registers of 16 bit each
<b>Word (dd,mm,yy,hh,MM,ss)</b>	dd, mm, yy, hh, MM, ss	6 registers of 16 bit each
<b>Float (mmddy.0,hhMMss.0)</b>	mmddy.0, hhMMss	2 registers of 32 bit each
<b>Float (yy,mm,dd,hh,MM,ss)</b>	yy, mm, dd, hh, MM, ss	6 registers of 32 bit each
<b>Unix</b>	number of seconds since 1.1.1970 00:00:00 – without stating the UTC	1 register of 32 bit

## 7 Bibliography

- MODBUS Application Protocol Specification V1.1b3 (Apr. 2012)
- MODBUS Over Serial Line Specification and Implementation Guide V1.02 (Dec. 2006)
- MODBUS Messaging on TCP/IP Implementation Guide V1.0b (Oct. 2006)

Publisher: Modbus Organization

Available at: [modbus.org/specs.php](http://modbus.org/specs.php)

## 8 Index

### A

An overview of the displays 31  
Appendix 42

### C

Causes of protocol errors 40

### D

Data description 43  
Data export 14  
Data import 14  
Data type  
    Bit strings/Status 44  
    Counter/Event counter 44  
    Date and time 45  
    Measurement 44  
Display and operation 30

### F

FAQs 34

### N

Navigation (enCore device) 30  
Nomenclature 42

### O

Online help  
    Access 5

### R

Register formats 43  
Resolving protocol errors 40

### S

Slave  
    Example parameterization 35

### T

Telegram type  
    0x03 43  
    0x04 43  
    0x06 43  
    0x08 43  
    0x10 43  
Telegram types 43  
Time adjustment 38

### U

Update system time  
    via Modbus 38