

AnyBus[®] Communicator Protocol Appendix **Modbus**

DOC. ABC-APPENDIX-MB Rev.0.91

Revision notes

Revision	Date	Description	Responsible
0.9	2001-06-20	Created	Edk/MaB
0.91	2001-07-30	Updated numbering	Edk

Preface

The data and illustrations found in this manual are not binding. We reserve the right to modify our products in line with our policy of continuous product development. The information in this manual is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks AB.

HMS Industrial Networks AB assumes no responsibility for any errors that may appear in this document.

The product and technology described in this document is patent pending in the following countries:
USA, Canada, Japan, Belgium, Denmark, Finland, France, Greece, Ireland, Italy, Luxemburg, Monaco, Netherlands, Portugal, Switzerland, Lichtenstein, Spain, United Kingdom, Sweden, Germany and Austria.

ANYBUS® is a registered trademark of HMS Industrial Networks AB.
All other trademarks are the property of their respective holders.

About the AnyBus Communicator Modbus Appendix

This fieldbus appendix contains fieldbus specific information about the Modbus protocol for the AnyBus Communicator. For more information about the AnyBus Communicator, please refer to the AnyBus Communicator User Manual, DOC. ABC-UM.

If technical support is required, please contact the AnyBus Support Centre:

Europe (Sweden)

Phone: +46 (0) 35 - 17 29 20

E-mail: support@hms.se

Germany

Phone: +49-721-96472-0

E-mail: ge-support@hms-networks.com

North America

Phone: +1-773-404-2271

Toll Free: 888-8-ANYBUS

E-mail: us-support@hms-networks.com

Japan

Phone: +81-45-478-5340

E-mail: jp-support@hms-networks.com

Conditions for trademark use

Please contact HMS for further information.

Related documents

Document name	Author	Document ID	Revision
AnyBus Communicator User Manual	Edk/MaB	ABC-UM	0.91
Modbus Protocol Reference Guide	Modicon	PI-MBUS-300	J

Abbreviations

Important abbreviations used in this manual:

Abbreviation	Description
AB-C	AnyBus Communicator
ABcCon	AnyBus Communicator Configuration Software

1 Appendix for Modbus

1.1 Introduction

When configured for Modbus protocols, the AnyBus Communicator supports Modbus RTU, Modbus ASCII and Modbus Generic. Some basic knowledge is needed, as to understand how to use the Anybus-C for configuration of the Modbus network.

The Modbus standard was created by Modicon for communication between controllers and other devices. The transactions on the Modbus network are of master/slave type, and are named “query” and “response”. One single master sends the queries. All transactions on the network have got a frame structure where one part is common for both Modbus RTU and Modbus ASCII. This is illustrated in figure 1.



Figure 1: Modbus frame layout

The main difference between Modbus RTU and Modbus ASCII is that in RTU all hexadecimal values are represented with one byte and in ASCII they are represented with two bytes. Another difference is the start and stop signs that envelope the frame.

Modbus RTU

Each byte in the Modbus RTU message represents a hexadecimal value between 0 and 255. The frame looks exactly like the one mentioned earlier and there is always an interval of 3.5 silent characters between the frames. CRC (Cyclical Redundancy Check) is used for error checking. Multiple Modbus transactions would appear on the physical interface like this:

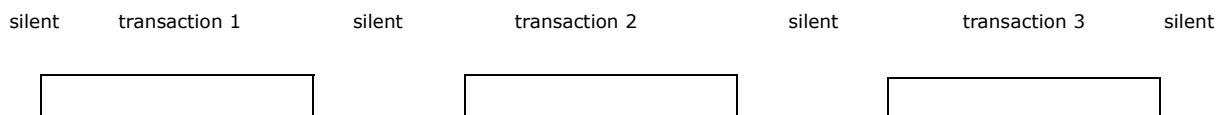


Figure 2: Modbus RTU transactions

Modbus ASCII

Each byte in the Modbus ASCII message represents one hexadecimal digit i.e. 0-9, A-F. This means that two bytes are used to represent each hexadecimal value (0x00-0xFF). For example, the value 0x2A is represented like this: 1st byte: "2" (0x32), 2nd byte: "A" (0x41). Start and stop characters are added to the frame from Figure 1. A colon, ":", is used as start character and "CR""LF" are used as stop characters. Longitudinal Redundancy Check (LRC) is used for error checking. A complete Modbus ASCII transaction would appear on the physical interface like this:

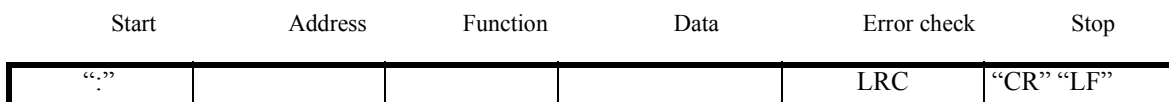


Figure 3: Modbus ASCII transactions

1.2 AB-C as Modbus master

When the AB-C acts as master on the Modbus network it uses a scan-list for communication with the different slaves on the network as described in the "Sub-network, Master" section. The scan-list is defined using AbcCon. When Modbus protocol is selected you add pre-defined modbus commands to the scan-list simply by selecting them from a list. The next chapter describes all the supported Modbus commands and what you need to think about when using them in the AB-C.

What makes the AB-C Modbus specific is the way transactions are used in Modbus commands and what the transactions consist of. One important issue in the AB-C is that the data must be of a pre-defined length. The example below shows how this works.

Basically, if we exclude the start character and stop character in the Modbus frame, each Modbus frame consists of two one-byte objects, one data object, maybe some more one- or two-byte objects and one error-check object. Lets take an example using Modbus RTU. Example. Read Holding Register (0x03) to node 0x05.

This command is built up like this:

Query:

Modbus frame	Address	Function	Data		Error check
Frame contents	0x05	0x03	"Starting Address"	"No. of points"	CRC
AB-C frame	"One byte object"	"One byte object"	Two byte object	Two byte object	"Error check object"

Table 1: Query

What you as a user must do here is to enter "Starting address" and "No. of points". These two parameters are represented as two-byte objects that you enter values into using AbcCon. This command will ask the slave for the same registers every time the command is sent and unless something goes wrong, the slave will answer with the same amount of data every time.

Response:

Modbus frame	Address	Function	Data		Error check
Frame contents	0x05	0x03	"Byte count"	"Data"	CRC
AB-C frame	"One byte object"	"One byte object"	One byte object	"Data Object"	"Error check object"

Table 2: Response

In the response the data section in the Modbus frame needs to be filled in. Here the data section is represented by a one-byte object (byte count) and a data object (data). To match the query you need to add a value into the one-byte object that is two times the No. of points value you entered in the query. Furthermore, the data object needs a starting address and a length where the length should match the one-byte object (byte count).

As you can see the requested data is always of the same length and therefore the data object in the response is also always of the same length. Should less bytes than specified arrive then the response is considered to have an error and a re-transmission of the query will occur if this command is configured for re-transmission. The same handling is done if more data than expected arrives.

1.3 Modbus commands

The following tables list all Modbus commands that are supported by the AB-C. For each command there is an explanation about what actions you as user need to take on the query and response.

Code	Name
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
07	Read Exception Status
11	Fetch Comm. Event Ctr
12	Fetch Comm. Event Log
15	Force Multiple Coils
16	Preset Multiple Registers
17	Report Slave ID
20	Read General Reference
21	Write General Reference
22	Mask Write 4X Register
23	Read/Write 4X Register
24	Read FIFO Queue

Table 3: Modbus Commands

1 Read Coil Status		
Query	Starting Address	2 byte value where you enter the first address of the requested coils.
	No. of Points	2 byte value where you enter the number of coils to read.
Response	Byte Count	1 byte value where you enter the number of expected data bytes.
	Data	Data object where you enter the length of the received data and the destination address.

Table 4: Read Coil Status

2 Read Input Status		
Query	Starting Address	2 byte value where you enter the first address of the requested discrete inputs.
	No. of Points	2 byte value where you enter the number of inputs to read.
Response	Byte Count	1 byte value where you enter the number of expected data bytes.
	Data	Data object where you enter the length of the received data and the destination address.

Table 5: Read Input Status

3 Read Holding Registers		
Query	Starting Address	2 byte value where you enter the first address of the requested registers.
	No. of Points	2 byte value where you enter the number of registers to read.
Response	Byte Count	1 byte value where you enter the number of expected data bytes.
	Data	Data object where you enter the length of the received data and the destination address.

Table 6: Read Holding Registers

4 Read Input Registers		
Query	Starting Address	2 byte value where you enter the first address of the requested registers.
	No. of Points	2 byte value where you enter the number of registers to read.
Response	Byte Count	1 byte value where you enter the number of expected data bytes.
	Data	Data object where you enter the length of the received data and the destination address.

Table 7: Read Input Registers

5 Force Single Coil		
Query	Data	Data object with 4 bytes. The fieldbus master should enter Coil Address and Force Data in these four bytes.
Response	Data	Data object with 4 bytes. The slave returns Coil Address and Forced Data in these 4 bytes.

Table 8: Force Single Coil

6 Preset Single Register		
Query	Data	Data object with 4 bytes. The fieldbus master should enter Register Address and Preset Data in these four bytes.
Response	Data	Data object with 4 bytes. The slave returns Register Address and Preset Data in these 4 bytes.

Table 9: Preset Single register

7 Read Exception Status		
Query	-	-
Response	Data	Data object with 1 byte. The slave returns the 8 Exception Status Coils in this byte.

Table 10: Read Exception Status

11 Fetch Comm. Event Counter		
Query	-	-
Response	Data	Data object with 4 bytes. The slave returns Status and Event Count in these 4 bytes.

Table 11: Fetch Comm. Event Counter

12 Fetch Comm. Event Log		
Query	-	-
Response	Byte Count	1 byte value where you enter the number of expected data bytes.
	Data	Data object with 6-70 bytes. The slave returns Status, Event Count, Message Count and the Event Log in these bytes.

Table 12: Fetch Comm. Event Log

15 Force Multiple Coils		
Query	Coil Address	2-byte value where you enter the reference of the first coil to be forced.
	Quantity of Coils	2-byte value where you enter the number of coils to force.
	Byte Count	1-byte value where you enter the number of data bytes.
	Data	Data object where you enter the length of the data to send and the source address. The fieldbus master should enter the force data in these bytes.
Response	Data	Data object with 4 bytes. The slave returns Coil Address and Quantity of Coils forced in these bytes.

Table 13: Force Multiple Coils

16 Preset Multiple Registers		
Query	Starting Address	2 byte value where you enter the address of the first register to be preset.
	No. of Registers	2 byte value where you enter the number of registers to preset.
	Byte Count	1 byte value where you enter the number of data bytes.
	Data	Data object where you enter the length of the data to send and the source address. The fieldbus master should enter the preset data in these bytes.
Response	Data	Data object with 4 bytes. The slave returns Starting Address and No. of Registers preset in these bytes.

Table 14: Preset Multiple Registers

17 Report Slave ID		
Query	-	-
Response	Byte Count	1 byte value where you enter the number of expected data bytes.
	Data	Data object where you enter the length of the recieved data and the destination address. The slave returns slave ID, Run Indicator Status and Additional Data in these bytes.

Table 15: Report Slave ID

20 Read General Reference		
Query	-	-
Response	-	-

Table 16: Read General reference

21 Write General reference		
Query	-	-
Response	-	-

Table 17: Write General Reference

22 Mask Write 4X Registers		
Query	Data	Data object where you enter the length of the data to send and the source address. The fieldbus master should enter the Reference Address, AND mask and OR mask in these bytes.
Response	Data	Data object where you enter the number of expected data bytes and the destination address. The slave returns Reference Address, AND mask and OR mask in these bytes.

Table 18: Mask Write 4X Registers

23 Read/Write 4X Registers		
Query	-	-
Response	-	-

Table 19: Read/Write 4X Registers

24 read FIFO Queue		
Query	-	-
Response	-	-

Table 20: Read FIFO Queue

This page is intentionally left blank.



If you have any comments about this documentation, please take a few minutes to fill out this form, and let us know about your opinions. These comments will help us improve our work, and make us aware of what customers of our products may find good, faulty or even missing.

Document title and revision: _____

Your name and company: _____

Phone: _____

E-mail: _____

Comments:

Text and illustrations:

What information is missing or unclear?:

Other comments:

Send your comments to:

*HMS Industrial Networks AB
Support Department
Pilefeltsgatan 93-95
302 50 Halmstad
SWEDEN*

You may also mail or fax your comments:

*E-mail: support@hms-networks.com
Fax: +46 (0)35 172909*