

Network Interface Appendix

Anybus[®] CompactCom 30 Modbus RTU

Doc.Id. HMSI-27-339
Rev. 2.03

Important User Information

This document is intended to provide a good understanding of the functionality offered by Modbus RTU. The document only describes the features that are specific to the Anybus CompactCom 30 Modbus RTU. For general information regarding the Anybus CompactCom, consult the Anybus CompactCom design guides.

The reader of this document is expected to be familiar with high level software design, and communication systems in general. The use of advanced Modbus RTU-specific functionality may require in-depth knowledge in Modbus RTU networking internals and/or information from the official Modbus RTU specifications. In such cases, the people responsible for the implementation of this product should either obtain the Modbus RTU specification to gain sufficient knowledge or limit their implementation in such a way that this is not necessary.

Liability

Every care has been taken in the preparation of this manual. Please inform HMS Industrial Networks AB of any inaccuracies or omissions. The data and illustrations found in this document are not binding. We, HMS Industrial Networks AB, reserve the right to modify our products in line with our policy of continuous product development. The information in this document 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.

There are many applications of this product. Those responsible for the use of this device must ensure that all the necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

HMS Industrial Networks AB will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features, timing, or functional side effects found outside the documented scope of this product. The effects caused by any direct or indirect use of such aspects of the product are undefined, and may include e.g. compatibility issues and stability issues.

The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, HMS Industrial Networks AB cannot assume responsibility for actual use based on these examples and illustrations.

Intellectual Property Rights

HMS Industrial Networks AB has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the US and other countries.

Trademark Acknowledgements

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

Warning:	This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.
ESD Note:	This product contains ESD (Electrostatic Discharge) sensitive parts that may be damaged if ESD control procedures are not followed. Static control precautions are required when handling the product. Failure to observe this may cause damage to the product.

Table of Contents

	Important User Information	
	<i>Liability</i>	1
	<i>Intellectual Property Rights</i>	1
	<i>Trademark Acknowledgements</i>	1
Preface	About This Document	
	Related Documents	1
	Document History	1
	Conventions & Terminology	2
	Support.....	2
Chapter 1	About the Anybus CompactCom Modbus RTU	
	General	3
	Features	3
Chapter 2	Tutorial	
	Introduction	4
	Fieldbus Conformance Notes	4
Chapter 3	Basic Operation	
	General Information	5
	<i>Software Requirements</i>	5
	<i>Network Data Conversion</i>	6
	Communication Settings	7
	Diagnostics	7
	Data Exchange.....	8
	<i>Application Data (ADIs)</i>	8
	<i>Process Data</i>	9

Chapter 4 **Modbus Register Implementation**

Holding Registers (4x)	10
Input Registers (3x)	10
Coils (0x)	10
Discrete Inputs (1x)	10

Chapter 5 **Modbus Functions**

Read Coils	12
Read Discrete Inputs	12
Read holding Registers	13
Read Input Registers	13
Write Single Coil	13
Write Single Register	14
Diagnostics	14
Write Multiple Coils	14
Write Multiple Registers	16
Report Slave ID	16
Read/Write Multiple Registers	16
Read Device Identification	16
Exchange Process Data	17
Send ABCC Message	17

Chapter 6 **Anybus Module Objects**

General Information	18
Anybus Object (01h)	19
Diagnostic Object (02h)	20
Network Object (03h)	21
Network Configuration Object (04h)	22
<i>Multilingual Strings</i>	24

Chapter 7 Network Specific Host Application Objects

General Information	25
Modbus Host Object (FAh)	26

Appendix A Categorization of Functionality

Basic.....	28
Extended.....	28
Advanced	28

Appendix B Implementation Details

SUP-Bit Definition.....	29
Anybus State Machine	29
Application Watchdog Timeout Handling	29

Appendix C Modbus Message Forwarding

Appendix D Technical Specification

Front View.....	31
Protective Earth (PE) Requirements	32
Power Supply	32
Environmental Specification	32
EMC Compliance.....	32

Appendix E Timing & Performance

General Information	33
Process Data.....	34
<i>Overview</i>	34
<i>Anybus Read Process Data Delay (Anybus Delay)</i>	34
<i>Anybus Write Process Data Delay (Anybus Delay)</i>	34
<i>Network System Read Process Data Delay (Network System Delay)</i>	35
<i>Network System Write Process Data Delay (Network System Delay)</i>	35

P. About This Document

For more information, documentation etc., please visit the HMS website, 'www.anybus.com'.

P.1 Related Documents

Document	Author
Anybus CompactCom 30 Software Design Guide	HMS
Anybus CompactCom 30 Hardware Design GuideHMSI-27-339	HMS
Anybus CompactCom Software Driver User Guide	HMS
Modbus application specification	MODBUS-IDA
Modbus over serial line specification and implementation guide	Modbus.org

P.2 Document History

Summary of Recent Changes (2.02 ... 2.03)

Change	Page(s)
Updated support information	2
Front view information moved to Technical Specification	31
Clarified information for Communication LED	31
Clarified table for Modbus Interface	31
Updated Modbus functions with No of Registers where applicable	Chapter 5

Revision List

Revision	Date	Author(s)	Chapter(s)	Description
1.00	2005-09-14	PeP	-	First official version
1.01	2005-10-18	PeP	-	Misc. minor corrections
1.10	2005-12-08	PeP	5, 6	Updated Modbus functionality & added multilingual information
1.12	2008-01-14	PeP	1, 5, 7, 6, D	Minor update
2.00	2010-10-14	KeL	All	Change of concept
2.01	2011-02-09	KeL	P, 3, 7	Minor updates
2.02	2011-08-26	KaD	3	Minor updates
2.03	2016-01-08	KeL	P, 5, D	minor updates

P.3 Conventions & Terminology

The following conventions are used throughout this manual:

- Numbered lists provide sequential steps
- Bulleted lists provide information, not procedural steps
- The terms ‘Anybus’ or ‘module’ refers to the Anybus CompactCom module.
- The terms ‘host’ or ‘host application’ refers to the device that hosts the Anybus module.
- Hexadecimal values are written in the format NNNNh or 0xNNNN, where NNNN is the hexadecimal value.
- A byte always consists of 8 bits.

P.4 Support

For general contact information and support, please refer to the contact and support pages at www.anybus.com.

1. About the Anybus CompactCom Modbus RTU

1.1 General

The Anybus CompactCom Modbus RTU communication module provides instant Modbus RTU connectivity via the patented Anybus CompactCom host interface. Any device that supports this standard can take advantage of the features offered by the module, allowing seamless network integration regardless of network type.

This product conforms to all aspects of the host interface for Active modules defined in the Anybus CompactCom Hardware- and Software Design Guides, making it fully interchangeable with any other device following that specification. Generally, no additional network related software support is needed, however in order to take advantage of advanced network specific functionality, a certain degree of dedicated software support may be necessary.

1.2 Features

- Galvanically isolated bus electronics
- Customizable Identity Information
- Supports all common baud rates up to 115200bps
- RTU (8bit) and ASCII (7bit) support
- Modbus message forwarding
- Diagnostic support

2. Tutorial

2.1 Introduction

This chapter is a complement to the Anybus CompactCom Implementation Tutorial. The ABCC tutorial describes and explains a simple example of an implementation with Anybus CompactCom. This chapter includes network specific settings that are needed for a host application to be up and running and possible to certify for use on Modbus RTU networks.

2.2 Fieldbus Conformance Notes

- This product is pre-certified for network compliance. While this is done to ensure that the final product can be certified, it does not necessarily mean that the final product will not require re-certification. For further information, please contact HMS.

3. Basic Operation

3.1 General Information

3.1.1 Software Requirements

No additional network support code needs to be written in order to support the Anybus CompactCom Modbus RTU, however due to the nature of the Modbus RTU networking system certain restrictions must be taken into account:

- Modbus RTU is acyclic by nature. Process Data can however still be accessed slightly faster due to less host communication overhead.
- Modbus RTU in itself does not impose any particular timing demands when it comes to acyclic requests (i.e. requests towards instances in the Application Data Object), however it is generally recommended to process and respond to such requests within a reasonable time period (exactly what this means in practice depends on the implementation and the actual installation).
- The Anybus State Machine indicates 'PROCESS_ACTIVE' as long as the Modbus RTU communication takes place within a specified timeout period, after which the module shifts to 'WAIT_PROCESS'. By default, this timeout is disabled, causing the module to stay in 'PROCESS_ACTIVE' after the first received Modbus request.
- Anybus state 'IDLE' is managed through a dedicated entry in the Modbus register map.
- The total number of ADIs that can be represented on the network depends on their size. By default, ADIs with instance numbers 1...4063 can be accessed from the network, each with a size of up to 32 bytes.
- ADI Names, types and similar attributes cannot be accessed from the network.
- A network write access of an ADI mapped to process data will result in a corresponding write access of the process data buffer of the Anybus CompactCom Modbus RTU. Such access will therefore not result in a Set_Attribute command towards the application.
- A network read access of an ADI, even if it is mapped to process data, will result in a corresponding Get_Attribute command towards the application.
- No support for network reset requests
- During runtime, up to 5 diagnostic instances can be created by the host application. An additional 6th instance may be created in the event of a major fault.
- The use of advanced Modbus RTU-specific functionality may require in-depth knowledge in Modbus RTU networking internals and/or information from the official Modbus RTU specification. In such cases, the people responsible for the implementation of this product is expected either to obtain these specifications to gain sufficient knowledge or limit their implementation in such a way that this is not necessary.

For in depth information regarding the Anybus CompactCom software interface, consult the general Anybus CompactCom 30 Software Design Guide.

3.1.2 Network Data Conversion

All network data (ADIs and Process Data) will be converted to/from the Modbus network as follows:

Anybus Type	Description	Conversion Details
BOOL	Boolean	Single BOOLS and arrays of BOOLS are converted to bit fields and padded with zeroes to the closest 16bit multiple. This is done in order to make it possible to access Booleans using the Modbus discrete and coil functions.
SINT8	Signed 8bit integer	Padded with zeroes to fill a 16bit register. Arrays of SINT8 are packed and padded with zeroes to the closest 16bit multiple.
SINT16	Signed 16bit integer	Occupies one 16bit Modbus register
SINT32	Signed 32bit integer	Occupies two 16bit Modbus registers
SINT64	Signed 64bit integer	Occupies four 16bit Modbus registers
UINT8	Unsigned 8bit integer	Padded with zeroes to fill a 16bit register. Arrays of UINT8 are packed and padded with zeroes to the closest 16bit multiple.
UINT16	Unsigned 16bit register	Occupies one 16bit Modbus register
UINT32	Unsigned 32bit register	Occupies two 16bit Modbus registers
UINT64	Unsigned 64bit integer	Occupies four 16bit Modbus registers
CHAR	Character	Padded with zeroes to fill a 16bit register. Arrays of CHAR are packed and padded with zeroes to the closest 16bit multiple.
ENUM	Enumeration	Padded with zeroes to fill a 16bit register. Arrays of 8bit ENUM are packed and padded with zeroes to the closest 16bit multiple.
FLOAT	Floating point/real number	Occupies two 16bit Modbus registers

See also “Process Data” on page 9.

3.2 Communication Settings

As with other Anybus CompactCom products, network related communication settings are grouped in the Network Configuration Object (04h).

In the case of Modbus, this includes...

- **Device Address¹**

On Modbus, each device on the network must be assigned a unique node address. The Device Address specifies which address to use for the module.

- **Communication Settings (Baudrate, Parity & no. of Stop bits)¹**

The module supports all common baud rates up to 115.2kbps; odd/even/no parity; 1 or 2 stop bits. These settings must be set to match the network, or else no communication can take place.

- **Modbus Mode (RTU/ASCII)¹**

The module supports 7bit ASCII and 8bit RTU communication. This setting must be set to match the network, or no communication can take place.

- **Process Active Timeout**

This value specifies how long the module shall stay in the 'PROCESS_ACTIVE'-state after receiving a Modbus request. Note that this affects the SUP-bit, see "SUP-Bit Definition" on page 29.

The parameters in the Network Configuration Object (04h) are also available from the network as dedicated entries in the Modbus register map, see "Holding Registers (4x)" on page 10.

3.3 Diagnostics

Each instance within the Diagnostic Object (02h) is represented on the network as a dedicated entry in the Modbus register map (see "Input Registers (3x)" on page 10). Note that since each entry corresponds *directly* to a specific diagnostic instance, it is possible to have "empty" diagnostic entries in the register map (such entries return zeroes upon reading).

Note that this functionality must not be confused with the Modbus diagnostic functionality (Function Code 8), see "Diagnostics" on page 14.

1. These settings require a reset in order to have effect; Any changes made during runtime will cause the module to indicate an error on the Device Status LED (see "Diagnostic support" on page 3). It will then continue operating using the previous settings until reset.

3.4 Data Exchange

3.4.1 Application Data (ADIs)

As mentioned previously, the total number of ADIs that can be represented on the network depends on their size. By default, ADIs with instance numbers 1...4063 can be accessed from the network, each with a size of up to 32 bytes. It is possible to alter this ratio by implementing attribute #9 ('Number of ADI indexing bits') in the Modbus Object (FAh).

Example 1 (Default settings)

In this example, attribute #9 in the Modbus Host Object (FAh) is set to its default value (04h).

Holding Register #	ADI No.
0210h... 021Fh	1
0220h... 022Fh	2
0230h... 023Fh	3
0240h... 024Fh	4
...	...
FFE0h... FFEFh	4062
FFF0h... FFFFh	4063

Each ADI is represented using 16 Modbus registers, which means that in theory up to 32 bytes of an ADI can be accessed from the network.

Example 2 (Customized implementation)

In this example, attribute #9 in the Modbus Host Object (FAh) is set to 05h.

Holding Register #	ADI No.
0210h... 022Fh	1
0230h... 024Fh	2
0250h... 026Fh	3
0270h... 028Fh	4
...	...
FFB0h... FFCFh	2030
FFD0h... FFEFh	2031

Each ADI is represented using 32 Modbus registers, which means that in theory up to 64 bytes of an ADI can be accessed from the network.

Note: ADIs must be written as a whole, i.e. it is not permitted to write to parts of a larger ADI. Reading ADIs can be performed without limitations, however.

3.4.2 Process Data

Modbus RTU is acyclic by nature and does not feature a dedicated cyclic data channel in the same sense as many other networks. The response time for registers associated with Process Data will however still be significantly faster due to less host communication overhead.

512 Holding registers and 256 Input registers are reserved for Process Data. This ensures that all combinations of data types fits into Modbus register space. Just as regular ADIs, the Process Data is converted to a format suitable for Modbus.

Example:

Application Process Data		Corresponding Modbus Register Layout			Comment
Byte Offset	Data Type	Register Offset	High Byte	Low Byte	
0	UINT8	0		UINT8	Padded to a full 16bit register
1	UINT8	1		UINT8	Padded to a full 16bit register
2	ENUM	2		ENUM	Padded to a full 16bit register
3	SINT16	3	SINT16		-
4					
5	BOOL	4			Converted to a bit field and padded to a full 16bit register
6	UINT32	5	UINT32		Two 16 bit registers
7					
8		6			
9					
10	BOOL ^[3]	7			Packed to a bit field and padded to a full 16bit register
11					
12					
13	SINT8 ^[3]	8	SINT8 ^[1]	SINT8 ^[0]	Padded to two 16bit registers
14		9		SINT8 ^[2]	
15					
16...37	BOOL ^[23]	10	b15...0		Packed to a bit field and padded to two 16bit registers
		11		b22...b16	

Note: The example above assumes that the Process Data is accessed as Holding Registers or Input Registers. For more information, see below.

As seen in the Modbus register map, Process Data can be accessed on a bit by bit basis (Coils & Discrete Inputs) - *or* - as 16bit entities (Holding Registers & Input Registers). For example, reading Discrete Inputs 0000h-000Fh will return the same data as reading Input Register 0000h or Holding Register 0100h.

See also...

- “Network Data Conversion” on page 6
- “Modbus Register Implementation” on page 10

Note: Writing to the Write Process Data register area has no effect, and reading unused register locations will return zeroes.

4. Modbus Register Implementation

4.1 Holding Registers (4x)

Range	Contents	Notes
0000h...00FFh	Read Process Data	See "Process Data" on page 9
0100h...01FFh	Write Process Data	
0200h	Node Address	These registers corresponds to the settings in the Network Configuration Object (04h), see "Network Configuration Object (04h)" on page 22.
0201h	Communication Settings	
0202h	RTU/ASCII Mode	
0203h	Process Active Timeout	
0204h	Enter/Exit Idle Mode ^a	0: Not Idle, >0: Idle
0205h...020Fh	(reserved)	-
0210h...FFFFh	ADI No. 1....nn	See "Application Data (ADIs)" on page 8

a. See B-29 "Anybus State Machine"

4.2 Input Registers (3x)

Range	Contents	Notes
0000h...00FFh	Write Process Data	See "Process Data" on page 9
0100h	Diagnostic Event Count	Number of pending diagnostic events
0101h	Diagnostic Event #1	These registers corresponds to instances in the Diagnostic Object (02h), see "Modbus Host Object (FAh)" on page 26.
0102h	Diagnostic Event #2	
0103h	Diagnostic Event #3	
0104h	Diagnostic Event #4	
0105h	Diagnostic Event #5	High byte = Severity Low byte = Event Code
0106h	Diagnostic Event #6	
0107h...FFFFh	(reserved)	-

4.3 Coils (0x)

Range	Contents	Notes
0000h...0FFFh	Read Process Data	See "Process Data" on page 9

4.4 Discrete Inputs (1x)

Range	Contents	Notes
0000h...0FFFh	Write Process Data	See "Process Data" on page 9

5. Modbus Functions

The following Modbus functions are implemented in the module:

#	Function	Page
1	Read Coils	12
2	Read Discrete Inputs	12
3	Read holding Registers	13
4	Read Input Registers	13
5	Write Single Coil	13
6	Write Single Register	14
8	Diagnostics	14
15	Write Multiple Coils	14
16	Write Multiple Registers	16
17	Report Slave ID	16
23	Read/Write Multiple Registers	16
43	Read Device Identification	16
68	Exchange Process Data	17
69	Send ABCC Message	17

Supported Exception Codes

Code	Name	Description
0x01	Illegal function	The function code in the query is not supported
0x02	Illegal data address	The data address received in the query is outside the initialized memory area
0x03	Illegal data value	The data in the request is illegal

Note: If Modbus message forwarding has been enabled in the Modbus Object (FAh), all Modbus messages except function 8 (Diagnostics) will be forwarded to the host application. For more information, see “Modbus Message Forwarding” on page 30.

5.1 Read Coils

Function Code: 1
Register Type: 0x (Coils)
No of Registers: 1 to 2000 (0x7D0)

Details

This function is mapped to the Read Process data part of the Holding Registers as follows:

Coil #	Holding Register #	Bit #
0000h	0000h	0
0001h		1
0002h		2
0003h		3
...		...
000Fh		15
0010h	0001h	0
0011h		1
0012h		2
0013h		3
...		...
001Fh		15
...
0FF0h	00FFh	0
0FF1h		1
0FF2h		2
0FF3h		3
...		...
0FFFh		15

5.2 Read Discrete Inputs

Function Code: 2
Register Type: 1x (Discrete Inputs)
No of Registers: 1 to 2000 (0x7D0)

Details

This function is mapped to the Write Process data part of the Input Registers; the mapping is otherwise identical to that of ‘read coils’ function described above.

5.3 Read holding Registers

Function Code: 3
 Register Type: 4x (Holding Registers)
 No of Registers: 1 to 125 (0x7D)

Details

Mapped to Read- and Write Process Data, ADIs, and configuration registers. It is allowed to read parts of a larger ABCC data type; it is also allowed to read multiple ADIs using a single request.

5.4 Read Input Registers

Function Code: 4
 Register Type: 3x (Input Registers)
 No of Registers: 1 to 125 (0x007D)

Details

Mapped to Write Process Data and diagnostic registers.

5.5 Write Single Coil

Function Code: 5
 Register Type: 0x (Coils)

Details

This function is mapped to the Read Process data part of the Holding Registers as follows:

Coil #	Holding Register #	Bit #
0000h	0000h	0
0001h		1
0002h		2
0003h		3
...		...
000Fh		15
0010h	0001h	0
0011h		1
0012h		2
0013h		3
...		...
001Fh		15
...

Coil #	Holding Register #	Bit #
0FF0h	00FFh	0
0FF1h		1
0FF2h		2
0FF3h		3
...		...
0FFFh		15

5.6 Write Single Register

Function Code: 6
 Register Type: 4x (Holding Inputs)

Details

Mapped to Read- and Write Process Data, ADIs and configuration registers. ADIs must be written as a whole, however the Process Data area accepts writes of any size.

5.7 Diagnostics

Function Code: 8
 Register Type: -

Details

Supported diagnostic sub functions are listed below.

#	Diagnostic Function	Comments
0	Return Query Data	Request data is looped back.
1	Restart Communications Option	Only resets the network interface; host interface remains unaffected.
4	Force Listen Only Mode	The module stays in the 'WAIT_PROCESS' state when in listen-only mode.
10	Clear Counters and Diagnostic Register	Clears counters, but there is no diagnostic register.
11	Return Bus Message Count	-
12	Return Bus Communication Error Count	-
13	Return Bus Exception Error Count	-
14	Return Slave Message Count	-
15	Return Slave No Response Count	-
16	Return Slave NAK Count	-
17	Return Slave Busy Count	-
18	Return Bus Character Overrun Count	Both character and message drops are counted using this counter.
20	Clear Overrun Counter and Flag	-

Note: 'Diagnostic register'-related sub functions are not implemented due to their Modicon-specific nature.

5.8 Write Multiple Coils

Function Code: 15
Register Type: 0x (Coils)
No of Registers: 1 to 1968 (0x7B0)

Details

This function is mapped to the Read Process data part of the Holding Registers; the mapping is identical to that of 'read coils' function described above.

5.9 Write Multiple Registers

Function Code: 16
 Register Type: 4x (Holding Registers)
 No of Registers: 1 to 123 (0x7B)

Details

Mapped to Read- and Write Process Data, ADIs and configuration registers.

Note: ADIs must be written as a whole, but the Process Data area accepts writes of any size.

5.10 Report Slave ID

Function Code: 17
 Register Type: -

Details

Requests the slave ID attribute from the Modbus Object (FAh). If this attribute is not implemented, the module will return 'unsupported FC' to the originator of the request.

5.11 Read/Write Multiple Registers

Function Code: 23
 Register Type: 4x (Holding Registers)
 No of Registers (read): 1 to 125 (0x7D)
 No of Registers (write): 1 to 121 (0x79)

Details

Mapped to read and write process data, ADIs and configuration registers.

Note 1: ADIs must be written as a whole, but the process data area accepts writes of any size.

Note 2: It is allowed to read parts of larger data types, and to read multiple ADIs using a single request.

5.12 Read Device Identification

Function Code: 43 (Subcode 14)
 Register Type: -

Details

Basic and regular device identification objects are supported according to the Modbus specification. Extended device identification objects are not supported.

Identification strings are extracted from the host application via the Modbus Object (FAh). If this object is not implemented, the default identification strings will be returned.

5.13 Exchange Process Data

Function Code: 68
Register Type: -

Details

This is an Anybus-specific function code which can be used for efficient Process Data exchange.

- **Request Format:**

The size of the 'Process Data Read'-subfield can be up to 252 bytes. Longer requests will be rejected by the module.

Addr	68	Process Data Read	CRC
------	----	-------------------	-----

- **Response Format:**

The size of the 'Process Data Write'-subfield can be up to 252 bytes. Larger responses will be truncated.

Addr	68	Process Data Write	CRC
------	----	--------------------	-----

5.14 Send ABCC Message

Function Code: 69
Register Type: -

Details

This is an Anybus-specific function code which can be used to access both objects in both the Anybus module and the host application.

- **Request Format:**

The format of the 'Message'-subfield is thoroughly described in the general Anybus CompactCom 30 Software Design Guide.

Addr	69	Message (Command)	CRC
------	----	-------------------	-----

- **Response Format:**

The format of the 'Message'-subfield is thoroughly described in the general Anybus CompactCom 30 Software Design Guide.

Addr	69	Message (Response)	CRC
------	----	--------------------	-----

6. Anybus Module Objects

6.1 General Information

This chapter specifies the Anybus Module Object implementation in the module.

Standard Objects:

- “Anybus Object (01h)” on page 19
- “Diagnostic Object (02h)” on page 20
- “Network Object (03h)” on page 21
- “Network Configuration Object (04h)” on page 22

Network Specific Objects:

(none)

6.2 Anybus Object (01h)

Category

Basic

Object Description

This object assembles all common Anybus data, and is described thoroughly in the general Anybus CompactCom 30 Software Design Guide.

Supported Commands

Object: Get_Attribute
 Instance: Get_Attribute
 Set_Attribute
 Get_Enum_String

Object Attributes (Instance #0)

(Consult the general Anybus CompactCom 30 Software Design Guide for further information.)

Instance Attributes (Instance #1)

Basic

#	Name	Access	Type	Value
1	Module type	Get	UINT16	0401h (Standard Anybus CompactCom)
2... 11	-	-	-	Consult the general Anybus CompactCom 30 Software Design Guide for further information.
12	LED colors	Get	struct of: UINT8 (LED1A) UINT8 (LED1B) UINT8 (LED2A) UINT8 (LED2B)	<u>Value:Color:</u> 01h Yellow 02h Red 01h Green 02h Red
13... 15	-	-	-	Consult the general Anybus CompactCom 30 Software Design Guide for further information.

6.3 Diagnostic Object (02h)

Category

Basic

Object Description

This object provides a standardised way of handling host application events & diagnostics, and is thoroughly described in the general Anybus CompactCom 30 Software Design Guide.

Supported Commands

Object: Get_Attribute
 Create
 Delete

Instance: Get_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1... 4	-	-	-	Consult the general Anybus CompactCom 30 Software Design Guide for further information.
11	Max no. of instances	Get	UINT16	5 + 1

Instance Attributes

Basic

#	Name	Access	Type	Value
1	Severity	Get	UINT8	Consult the general Anybus CompactCom 30 Software Design Guide for further information.
2	Event Code	Get	UINT8	Consult the general Anybus CompactCom 30 Software Design Guide for further information.

Each instance created in this object is represented as a dedicated entry in the Modbus register map.

See also...

- “Holding Registers (4x)” on page 10.

6.4 Network Object (03h)

Category

Basic

Object Description

For more information regarding this object, consult the general Anybus CompactCom 30 Software Design Guide.

Supported Commands

Object: Get_Attribute

Instance: Get_Attribute
 Set_Attribute
 Get_Enum_String
 Map_ADI_Write_Area
 Map_ADI_Read_Area

Object Attributes (Instance #0)

(Consult the general Anybus CompactCom 30 Software Design Guide for further information.)

Instance Attributes (Instance #1)

Basic

#	Name	Access	Type	Value
1	Network type	Get	UINT16	0045h
2	Network type string	Get	Array of CHAR	'Modbus RTU'
3	Data format	Get	ENUM	00h (LSB first)
4	Parameter data support	Get	BOOL	True
5	Write process data size	Get	UINT16	Current write process data size (in bytes) Updated on every successful Map_ADI_Write_Area ^a
6	Read process data size	Get	UINT16	Current read process data size (in bytes) Updated on every successful Map_ADI_Read_Area ^a
7	Exception Information	Get	UINT8	(No network specific exception information available)
8... 10	-	-	-	Consult the general Anybus CompactCom 30 Software Design Guide for further information.

a. Consult the general Anybus CompactCom 30 Software Design Guide for further information.

6.5 Network Configuration Object (04h)

Category

Extended

Object Description

This object holds network specific configuration parameters that may be set by the end user. A reset command (factory default) issued towards this object will result in all instances being set to their default values.

See also...

- 3-5 “General Information”

Supported Commands

Object:	Get_Attribute Reset
Instance:	Get_Attribute Set_Attribute Get_Enum_String

Object Attributes (Instance #0)

(Consult the general Anybus CompactCom 30 Software Design Guide for further information.)

Instance Attributes (Instance #1, ‘Device Address’)

This instance holds the actual Modbus node address.

Extended

#	Name	Access	Type	Description
1	Name ^a	Get	Array of CHAR	‘Address’
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (get/set/shared access)
5	Value ^b	Get/Set	UINT8	1...247: Modbus node address value

a. Multilingual, see 6-24 “Multilingual Strings”.

b. It is strongly recommended to use this attribute

Communication Settings (Instance #2, 'Comm Settings')

This instance holds the network communication settings, i.e. baud rate, parity, stop bits etc.

Extended

#	Name	Access	Type	Description																																																																																
1	Name ^a	Get	Array of CHAR	'Comm Settings'																																																																																
2	Data type	Get	UINT8	08h (= ENUM)																																																																																
3	Number of elements	Get	UINT8	01h (one element)																																																																																
4	Descriptor	Get	UINT8	07h (read/write/shared access)																																																																																
5	Value ^b	Get/Set	ENUM	<table><tr><th>b7</th><th>b6</th><th>b5</th><th>b4</th><th>b3</th><th>b2</th><th>b1</th><th>b0</th><th>Contents</th></tr><tr><td colspan="8" rowspan="4"></td><td>0</td><td>0</td><td>Even parity, 1 stop bit (default)</td></tr><tr><td>0</td><td>1</td><td>Odd parity, 1 stop bit</td></tr><tr><td>1</td><td>0</td><td>No parity, 2 stop bits</td></tr><tr><td>1</td><td>1</td><td>No parity, 1 stop bit</td></tr><tr><td colspan="8" rowspan="8"></td><td>0</td><td>0</td><td>1200bps</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>2400bps</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>4800bps</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>9600bps</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>19200bps (default)</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>38400bps</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>57600bps</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>76800bps</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>115200bps</td></tr></table>	b7	b6	b5	b4	b3	b2	b1	b0	Contents									0	0	Even parity, 1 stop bit (default)	0	1	Odd parity, 1 stop bit	1	0	No parity, 2 stop bits	1	1	No parity, 1 stop bit									0	0	1200bps	0	0	0	1	2400bps	0	0	1	0	4800bps	0	0	1	1	9600bps	0	1	0	0	19200bps (default)	0	1	0	1	38400bps	0	1	1	0	57600bps	0	1	1	1	76800bps	1	0	0	0	115200bps
b7	b6	b5	b4	b3	b2	b1	b0	Contents																																																																												
								0	0	Even parity, 1 stop bit (default)																																																																										
								0	1	Odd parity, 1 stop bit																																																																										
								1	0	No parity, 2 stop bits																																																																										
								1	1	No parity, 1 stop bit																																																																										
								0	0	1200bps																																																																										
								0	0	0	1	2400bps																																																																								
								0	0	1	0	4800bps																																																																								
								0	0	1	1	9600bps																																																																								
								0	1	0	0	19200bps (default)																																																																								
								0	1	0	1	38400bps																																																																								
								0	1	1	0	57600bps																																																																								
								0	1	1	1	76800bps																																																																								
1	0	0	0	115200bps																																																																																

a. Multilingual, see 6-24 "Multilingual Strings".

b. It is strongly recommended to use this attribute

Enumeration Strings

Enumeration strings are based on the current settings as follows: <baudrate><parity><stopbits>

- **<baudrate>**
Baudrate in ASCII, e.g. '2400' or '115200'
- **<parity>**
Single letter representing the parity setting: 'e' (even parity), 'o' (odd parity) or 'n' (no parity)
- **<stopbits>**
Single digit representing the number of stop bits, i.e. '1' (one stop bit) or '2' (two stop bits).

Example:

Baudrate:19200

Parity: Even

Stop bits:1

Enumeration String:'19200e1'

RTU/ASCII Mode (Instance #3, 'Mode')

This parameter specifies which communication protocol to use, i.e. Modbus RTU or Modbus ASCII.

Extended

#	Name	Access	Type	Description
1	Name ^a	Get	Array of CHAR	'Mode'
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (get/set/shared access)
5	Value ^b	Get/Set	ENUM	0:'RTU (8bits)' (default) 1:'ASCII (7bits)'

a. Multilingual, see 6-24 "Multilingual Strings".

b. It is strongly recommended to use this attribute

Process Active Timeout (Instance #4, 'Timeout')

This instance specifies the process active timeout in milliseconds.

Extended

#	Name	Access	Type	Description
1	Name ^a	Get	Array of CHAR	'Timeout'
2	Data type	Get	UINT8	05h (= UINT16)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (get/set/shared access)
5	Value ^b	Get/Set	UINT16	Process active timeout in milliseconds (default = 0).

a. Multilingual, see 6-24 "Multilingual Strings".

b. It is strongly recommended to use this attribute

See also...

- "Software Requirements" on page 5
- "Anybus State Machine" on page 29

6.5.1 Multilingual Strings

The instance names in this object are multilingual and are translated based on the current language settings as follows:

Instance	English	German	Spanish	Italian	French
1	Address	Adresse	Dirección	Indirizzo	Adresse
2	Comm Settings	Komm Einstell	Comu Config	Imp Comunic	Comm Config
3	Mode	Modus	Modo	Modalità	Mode
4	Timeout	Timeout	Timeout	Timeout	Timeout

7. Network Specific Host Application Objects

7.1 General Information

This chapter specifies the host application object implementation in the module. The objects listed here may optionally be implemented within the host application firmware to expand the Modbus RTU implementation.

Standard Objects:

- Application Object (see Anybus CompactCom 30 Software Design Guide)
- Application Data Object (see Anybus CompactCom 30 Software Design Guide)

Network Specific Objects:

- “Modbus Host Object (FAh)” on page 26

7.2 Modbus Host Object (FAh)

Category

Extended, advanced

Object Description

This object implements Modbus specific features in the host application.

The implementation of this object is optional; the host application can support none, some, or all of the attributes specified below. The module will attempt to retrieve the values of these attributes during start-up; if an attribute is not implemented in the host application, simply respond with an error message (06h, “Invalid CmdExt[0]”). In such case, the module will use its default value.

If the module attempts to retrieve a value of an attribute not listed below, respond with an error message (06h, “Invalid CmdExt[0]”).

Supported Commands

Object:	Get Attribute Process_modbus_message (see “Modbus Message Forwarding” on page 30)
Instance:	Get Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	'Modbus'
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	0001h
4	Highest instance no.	Get	UINT16	0001h

Instance Attributes (Instance #1)

Extended

#	Name	Access	Type	Default Value ^a	Comment
1	Vendor name	Get	Array of CHAR	'HMS'	These settings will be returned in the response to the Modbus request 'Read Device Identification', see "Read Device Identification" on page 16.
2	Product Code	Get	Array of CHAR	'ABCC-RTU'	
3	Major Minor Revision	Get	Array of CHAR	(firmware rev.)	
4	Vendor URL	Get	Array of CHAR	-	
5	Product name	Get	Array of CHAR	-	The maximum allowed length of each string is 244 bytes; strings exceeding this length will be truncated.
6	Model name	Get	Array of CHAR	-	
7	User Application Name	Get	Array of CHAR	-	
8	Device ID	Get	Array of UINT8	-	The value of this attribute is returned in the response to the Modbus function 'Report Slave ID', see "Report Slave ID" on page 16 (If not implemented, the module will respond with 'Unsupported FC'). The maximum allowed length of this attribute is 251 bytes.
9	No. of ADI indexing bits	Get	UINT8	04h	Range: 0... 7 Specifies how ADIs are represented on Modbus RTU. For more information, see "Application Data (ADIs)" on page 8.

a. If an attribute is not implemented, this value will be used instead (with the exception of attribute #8 'Device ID').

Advanced

#	Name	Access	Type	Default Value ^a	Comment
10	Enable Modbus Message forwarding	Get	BOOL	False	See "Modbus Message Forwarding" on page 30

a. If an attribute is not implemented, this value will be used instead (with the exception of attribute #8 'Device ID').

A. Categorization of Functionality

The objects, including attributes and services, of the Anybus CompactCom and the application are divided into three categories: basic, advanced and extended.

A.1 Basic

This category includes objects, attributes and services that are mandatory to implement or to use. They will be enough for starting up the Anybus CompactCom and sending/receiving data with the chosen network protocol. The basic functions of the industrial network are used.

Additional objects etc, that will make it possible to certify the product also belong to this category.

A.2 Extended

Use of the objects in this category extends the functionality of the application. Access is given to the more specific characteristics of the industrial network, not only the basic moving of data to and from the network. Extra value is given to the application.

A.3 Advanced

The objects, attributes and services that belong to this group offer specialized and/or seldom used functionality. Most of the available network functionality is enabled and accessible. Access to the specification of the industrial network is normally required.

B. Implementation Details

B.1 SUP-Bit Definition

The supervised bit (SUP) indicates that the network participation is supervised by another network device. In the case of Modbus, this means that the SUP-bit will be set when the module is in PROCESS_ACTIVE state and the Process Active Timeout is set to a value other than zero.

B.2 Anybus State Machine

The table below describes how the Anybus State Machine relates to the Modbus network.

Anybus State	Implementation	Comment
WAIT_PROCESS	Awaiting Modbus requests	-
ERROR	-	-
PROCESS_ACTIVE	A Modbus request addressed to this node has been received within the last 'Process Active Timeout' time.	If no timeout value is specified, the module will stay in this state after the first received Modbus request.
IDLE	This state can be entered as desired by writing to Holding Register no. 0204h (See "Holding Registers (4x)" on page 10).	This is a network specific implementation which can be used as desired to put the module in Idle state.
EXCEPTION	Further Modbus requests will be ignored	-

B.3 Application Watchdog Timeout Handling

Upon detection of an application watchdog timeout, the module will cease network participation and shift to state 'EXCEPTION'. No other network specific actions are performed.

C. Modbus Message Forwarding

Category

Advanced

Description

If Modbus message forwarding has been enabled (Modbus Host Object (FAh), Instance #1, attribute #12), all Modbus requests with the exception of function code #8 (Diagnostics) will be forwarded to the host application.

For this purpose, the Modbus Host Object implements a command called `Process_Modbus_Message` (Command code 10h), which is used to tunnel Modbus requests to the host application.

- Command Message Layout**

The following message will be sent by the module to the host application upon receiving a Modbus request.

Field	Contents								Notes
	b7	b6	b5	b4	b3	b2	b1	b0	
Source ID	(Source ID)								Selected by the module
Dest. Object	FCh								Destination Object = Modbus Host Object
Dest. Instance (lsb)	00h								Destination Instance = Object Instance
Dest. Instance (msb)	00h								
E	0								This is not an error message
C		1							This is a command
Command Number			10h						Process_Modbus_Message
Message Data Size	(size of MsgData[0...n])								-
CmdExt[0]									(reserved, ignore)
CmdExt[1]									
MsgData[0...n]	Modbus message (Request)								Request message frame, excluding CRC

- Response Message Layout**

The host application must to respond to the request as described below.

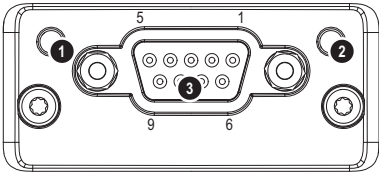
Field	Contents								Notes
	b7	b6	b5	b4	b3	b2	b1	b0	
Source ID	(Source ID)								(Selected by the module)
Dest. Object	FCh								Object = Modbus Host Object
Dest. Instance (lsb)	00h								Instance = Object Instance
Dest. Instance (msb)	00h								
E	0								This is not an error message
C		0							This is a response
Command Number			10h						Process_Modbus_Message
Message Data Size	(size of MsgData[0...n])								-
CmdExt[0]									(reserved, set to zero)
CmdExt[1]									
MsgData[0...n] ^{ab}	Modbus message (response)								Response message frame, excluding CRC

a. The response data size must not exceed 254 bytes.

b. If there is no data in the response, no response will be sent to the originator of the Modbus request.

D. Technical Specification

D.1 Front View

#	Item	
1	Communication LED	
2	Device Status LED	
3	Modbus Interface	

Communication LED

LED State	Description
Off	No power - <i>or</i> - no traffic
Yellow	This LED will flash during correct reception and transmission (20 ms on, 40 ms off)
Red	A fatal error has occurred

Device Status LED

LED State	Description
Off	Initializing - <i>or</i> - no power
Green	Module initialized, no error
Red	Internal error - <i>or</i> - major unrecoverable fault
Red, single flash	Communication fault or configuration error Case 1: Invalid settings in Network Configuration Object. Case 2: Settings in Network Configuration Object has been changed during runtime (i.e. the settings does not match the currently used configuration)
Red, double flash	Application diagnostics available

Modbus Interface

The Modbus interface is galvanically isolated, and provides both RS-232 and RS-485.

Pin	Direction	Signal	Comment
Housing	-	PE	Protective Earth
1	-	GND	Bus polarization, ground (isolated)
2	Output ^a	5V	Bus polarization power +5V DC (isolated)
3	Input	PMC	Connect to pin #2 for RS-232 operation. Leave unconnected for RS-485 operation.
4	-	-	-
5	Bidirectional	B-Line	RS-485 B-Line (+)
6	-	-	-
7	Input	Rx	RS-232 Data Receive
8	Output	Tx	RS-232 Data Transmit
9	Bidirectional	A-Line	RS-485 A-Line (-)

a. Any current drawn from this pin will affect the total power consumption. See also "Power Consumption" on page 32.

D.2 Protective Earth (PE) Requirements

In order to ensure proper EMC behaviour, the module must be properly connected to protective earth via the PE pad / PE mechanism described in the general Anybus CompactCom 30 Hardware Design Guide.

HMS Industrial Networks does not guarantee proper EMC behaviour unless these PE requirements are fulfilled.

D.3 Power Supply

Supply Voltage

The module requires a regulated 3.3V power source as specified in the general Anybus CompactCom 30 Hardware Design Guide.

Power Consumption

The Anybus CompactCom Modbus RTU is designed to fulfil the requirements of a Class A module. For more information about the power consumption classification used on the Anybus CompactCom platform, consult the general Anybus CompactCom 30 Hardware Design Guide.

The current hardware design consumes up to 210mA¹².

Note: It is strongly advised to design the power supply in the host application based on the power consumption classifications described in the general Anybus CompactCom 30 Hardware Design Guide, and not on the exact power requirements of a single product.

D.4 Environmental Specification

Consult the Anybus CompactCom 30 Hardware Design Guide for further information.

D.5 EMC Compliance

Consult the Anybus CompactCom 30 Hardware Design Guide for further information.

-
1. Note that in line with HMS policy of continuous product development, we reserve the right to change the exact power requirements of this product without prior notification. Note however that in any case, the Anybus CompactCom Modbus RTU will remain as a Class A module.
 2. This value is valid under the condition that no current is being drawn from bus connector pin 2 (+5V output for bus polarization; see “Modbus Interface” on page 31).

E. Timing & Performance

E.1 General Information

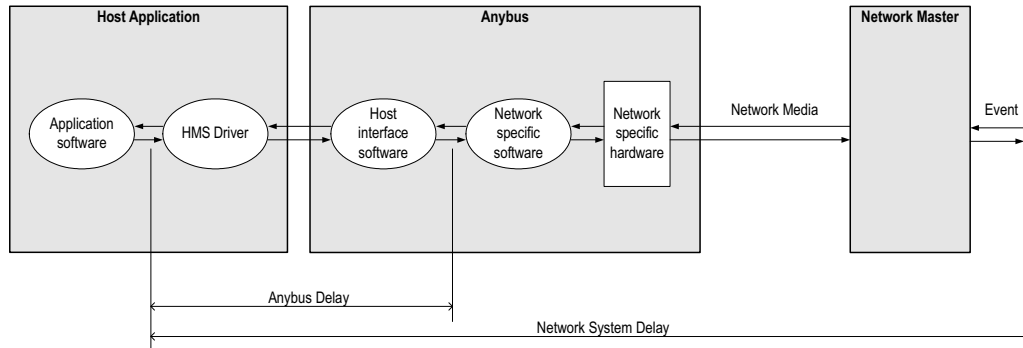
This chapter specifies timing and performance parameters that are verified and documented for the Anybus CompactCom 30 Modbus RTU.

The following timing aspects are measured:

Category	Parameters	Page
Startup Delay	T1, T2	Please consult the Anybus CompactCom 30 Software Design Guide, App. B.
NW_INIT Delay	T3	
Telegram Delay	T4	
Command Delay	T5	
Anybus Read Process Data Delay (Anybus Delay)	T6, T7, T8	
Anybus Write Process Data Delay (Anybus Delay)	T12, T13, T14	
Network System Read Process Data Delay (Network System Delay)	T9, T10, T11	35
Network System Write Process Data Delay (Network System Delay)	T15, T16, T17	35

E.2 Process Data

E.2.1 Overview



E.2.2 Anybus Read Process Data Delay (Anybus Delay)

The Read Process Data Delay (labelled ‘Anybus delay’ in the figure above) is defined as the time measured from just before new data is buffered and available to the Anybus host interface software, to when the data is available to the host application (just after the new data has been read from the driver).

Please consult the Anybus CompactCom 30 Software Design Guide, Appendix B, for more information.

E.2.3 Anybus Write Process Data Delay (Anybus Delay)

The Write Process Data Delay (labelled ‘Anybus delay’ in the figure) is defined as the time measured from the point the data is available from the host application (just before the data is written from the host application to the driver), to the point where the new data has been forwarded to the network buffer by the Anybus host interface software.

Please consult the Anybus CompactCom 30 Software Design Guide, Appendix B, for more information.

E.2.4 Network System Read Process Data Delay (Network System Delay)

The Network System Read Process Data Delay (labelled 'Network System Delay' in the figure), is defined as the time measured from the point where an event is generated at the network master to when the corresponding data is available to the host application (just after the corresponding data has been read from the driver).

Parameter	Description	Typ.	Max.	Unit.
T9	Network System Read Process Data delay, 8 ADIs (single UINT8)	3.68	3.80	ms
T10	Network System Read Process Data delay, 16 ADIs (single UINT8)	5.20	5.36	ms
T11	Network System Read Process Data delay, 32 ADIs (single UINT8)	8.28	8.48	ms

Conditions:

Parameter	Conditions
Application CPU	-
Timer system call interval	1 ms
Driver call interval	0.2... 0.3 ms
No. of ADIs (single UINT8) mapped to Process Data in each direction.	8, 16 and 32
Communication	Parallel
Telegram types during measurement period	Process Data only
Bus load, no. of nodes, baud rate etc.	Normal

E.2.5 Network System Write Process Data Delay (Network System Delay)

The Network System Write Process Data Delay (labelled 'Network System Delay' in the figure), is defined as the time measured from the time after the new data is available from the host application (just before the data is written to the driver) to when this data generates a corresponding event at the network master.

Parameter	Description	Avg.	Max.	Unit.
T15	Network System Write Process Data delay, 8 ADIs (single UINT8)	Anybus write process data delay + modbus scan interval. The modbus scan interval is typically 10 ms or more, which makes the Anybus write process data delay negligible.		
T16	Network System Write Process Data delay, 16 ADIs (single UINT8)			
T17	Network System Write Process Data delay, 32 ADIs (single UINT8)			

Conditions: as in "Network System Read Process Data Delay (Network System Delay)" on page 35.