Anybus® CompactCom™ 30

SOFTWARE DESIGN GUIDE

HMSI-168-97 3.1 en-US ENGLISH

Important User Information

Liability

Intellectual Property Rights

Table of Contents

Page

1 Preface

1.1 About this Document

This document is intended to provide a good understanding of the Anybus CompactCom platform. It does not cover any of the network specific features offered by the Anybus CompactCom 30 products; this information is available in the appropriate Network Guide.

The reader of this document is expected to be familiar with high level software design and industrial network communication systems in general. For additional information, documentation, support etc., please visit the support website at www.anybus.com/support.

1.2 Related Documents

Document	Author	Document ID
Anybus CompactCom M30 Hardware Design Guide	HMS	HMSI-168-31
Anybus CompactCom B30 Design Guide	HMS	HMSI-27-242
Anybus CompactCom Host Application Implementation Guide	HMS	HMSI-27-334
Anybus CompactCom 30 Network Guides (separate document for each supported fieldbus or network system)	HMS	
Anybus CompactCom Implementation Tutorial	HMS	HMSI-168-106
Anybus CompactCom 30 Drive Profile Design Guides	HMS	

1.3 Document History

Version	Date	Description
1.00 - 2.09		See information in earlier versions of document
3.0	2017-08-22	Moved from FM to DOX General update
3.1	2019-03-01	Rebranded Minor corrections

1.4 Document Conventions

Ordered lists are used for instructions that must be carried out in sequence:

1. First do this
2. Then do this

Unordered (bulleted) lists are used for:

- Itemized information
- Instructions that can be carried out in any order

...and for action-result type instructions:

► This action...

→ leads to this result

Bold typeface indicates interactive parts such as connectors and switches on the hardware, or menus and buttons in a graphical user interface.

```
Monospaced text is used to indicate program code and other
kinds of data input/output such as configuration scripts.
```

This is a cross-reference within this document: *Document Conventions, p. 6*

This is an external link (URL): www.hms-networks.com

ⓘ *This is additional information which may facilitate installation and/or operation.*

! This instruction must be followed to avoid a risk of reduced functionality and/or damage to the equipment, or to avoid a network security risk.

⚠ **Caution**
This instruction must be followed to avoid a risk of personal injury.

⚠ **WARNING**
This instruction must be followed to avoid a risk of death or serious injury.

1.5 Document Specific Conventions

- The terms “Anybus” or “module” refers to the Anybus CompactCom module.
- The terms “host” or “host application” refer to the device that hosts the Anybus.
- Hexadecimal values are written in the format NNNNh or 0xNNNN, where NNNN is the hexadecimal value.
- Intel byte order is assumed unless otherwise stated.
- Object Instance equals Instance #0.
- Object Attributes resides in the Object Instance.
- The terms “Anybus implementation” and “Anybus version” generally refers to the implementation in the Anybus module, i.e. network type and internal firmware revision.
- Unless something is clearly stated to be optional, it shall be considered mandatory.

- When writing, fields declared as “reserved” shall be set to zero.
- When reading, fields bits declared as “reserved” shall be ignored.
- Fields which are declared as “reserved” must not be used for undocumented purposes.
- A byte always consists of 8 bits.
- A word always consists of 16 bits.

1.6 Trademark Information

- Anybus® is a registered trademark of HMS Industrial Networks.
- EtherNet/IP is a trademark of ODVA, Inc.
- DeviceNet is a trademark of ODVA, Inc.
- EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

All other trademarks are the property of their respective holders.

2 About the Anybus CompactCom 30

2.1 General Information

The Anybus CompactCom 30 network communication device is a high performance, low cost communication solution for industrial field devices. Typical applications are Frequency Inverters, PLCs, HMIs, Visualization Devices, Instruments, Scales, Robotics and Intelligent measuring devices.

The Anybus CompactCom 30 software interface is designed to be network protocol independent, allowing the host application to support all major networking systems using the same software driver, without loss of functionality.

To provide flexibility and room for expansion, an object oriented addressing scheme is used between the host application and the Anybus CompactCom. This allows for a very high level of integration, since the host communication protocol enables the Anybus CompactCom to retrieve information directly from the host application using explicit object requests rather than memory mapped data.

This document does not cover Passive Anybus CompactCom modules. For further information, consult the separate network interface appendices for these products.

2.2 Features

- Host interface is network protocol independent
- Multilingual support
- High level of integration
- Cyclic and Acyclic data
- Optional support for advanced network specific features
- Serial or Parallel operation
- Interrupt operation (optional)
- Free source level (C-language) software driver available

3 Software Introduction

3.1 Background

The primary function of an industrial network interface is to exchange information with other devices on the network. Traditionally, this has mostly been a matter of exchanging cyclic I/O and making it available to the host device via two memory buffers.

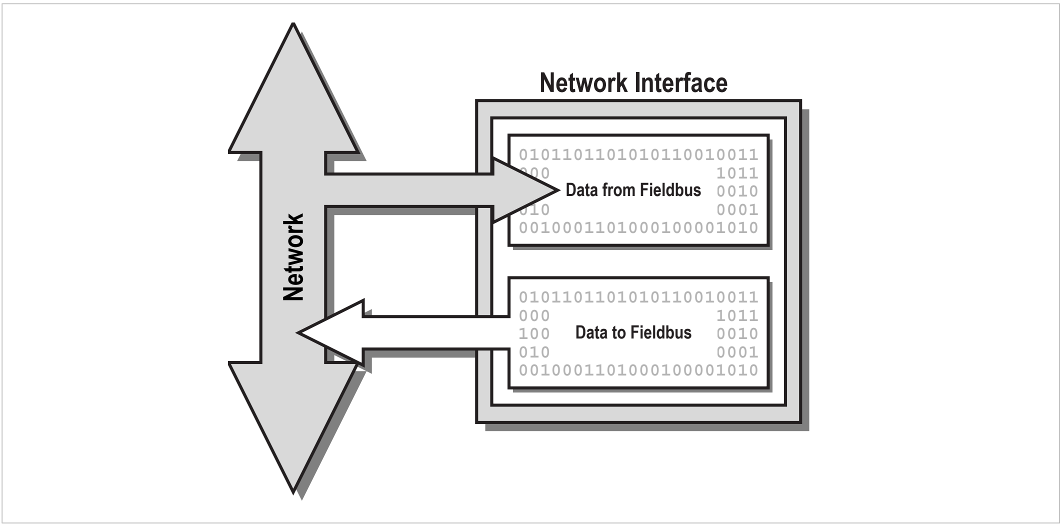


Fig. 1

As demand for higher level network functionality increases, the typical role of a network interface has evolved towards including acyclical data management, alarm handling, diagnostics etc.

Generally, the way this is implemented differs fundamentally between different networking systems. This means that supporting and actually taking advantage of this new functionality is becoming increasingly complex, if not impossible, without implementing dedicated software support for each network.

By utilizing modern object oriented technology, the Anybus CompactCom provides a simple and effective way of supporting most networking systems, as well as taking advantage of advanced network functionality, without having to write separate software drivers for each network. Acyclic requests are translated in a uniform manner, and dedicated objects provide diagnostic and alarm handling according to each network standard.

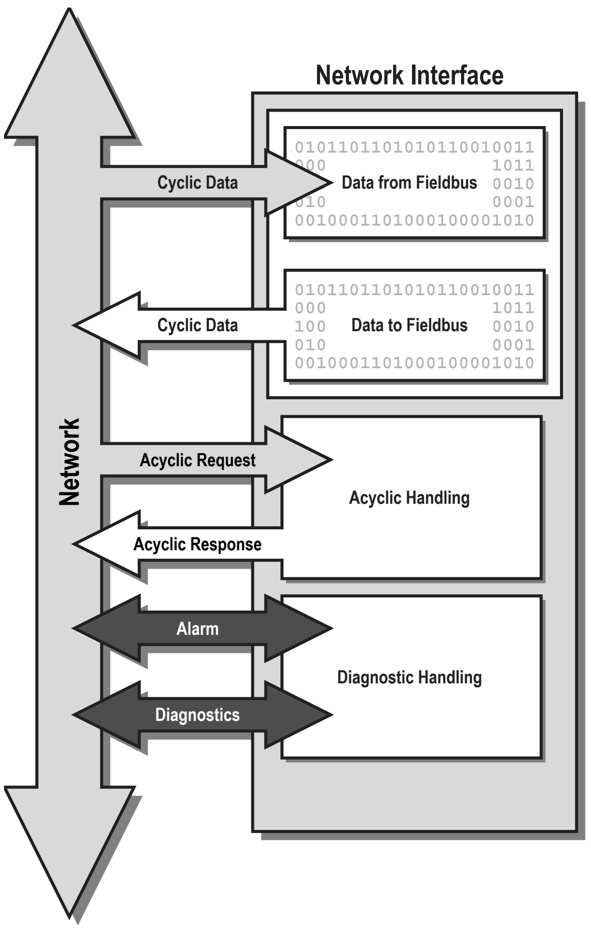


Fig. 2

3.2 The Object Model

3.2.1 Basics

To provide a flexible and logical addressing scheme for both the host application and the Anybus module, the software interface is structured in an object structured manner. While this approach may appear confusing at first, it is nothing more than a way of categorizing and addressing information.

Related information and services are grouped into entities called ‘Objects’. Each object can hold subentities called ‘Instances’, which in turn may contain a number of fields called ‘Attributes’. Attributes typically represents information or settings associated with the Object. Depending on the object in question, Instances may either be static or created dynamically during runtime.

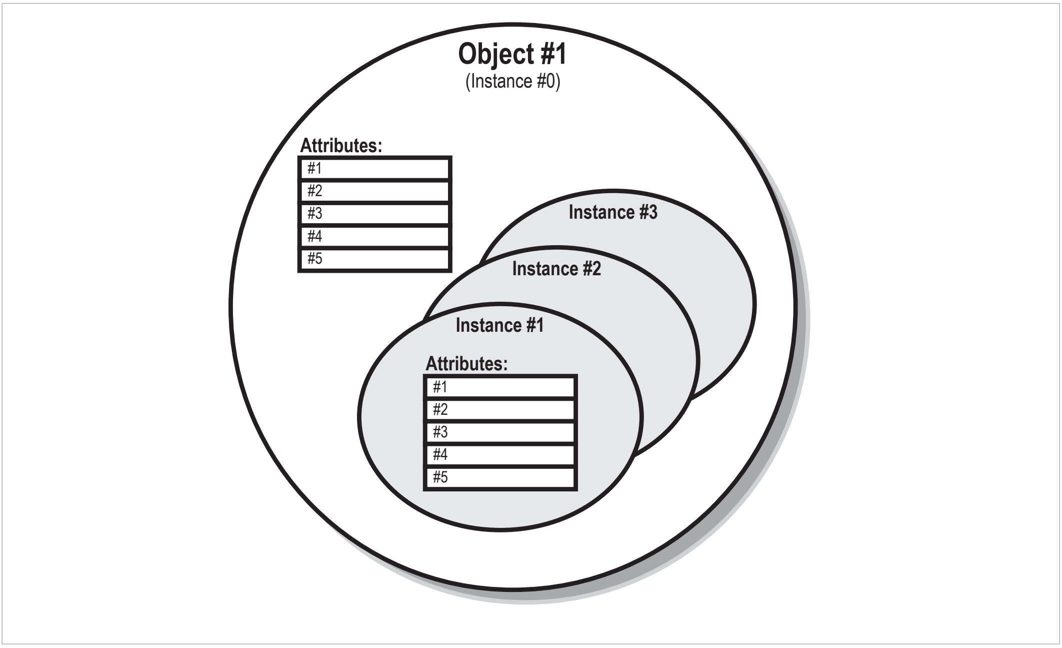


Fig. 3

3.2.2 Addressing Scheme

Objects, and their contents, are accessed using Object Messaging. Each object message is tagged with an object number, instance, and attribute, specifying the location of the data or setting associated with the message.

> **!** This addressing scheme applies to both directions; i.e. just like the Anybus module, the host application must be capable of interpreting incoming object requests and route them to the appropriate software routines.

Example:

The module features an object called the “Anybus Object”, which groups common settings relating the Anybus module itself.

In this object, instance #1 contains an attribute called ‘“Firmware version”’ (attribute #2). To retrieve the firmware revision of the module, the host simply issues a **Get Attribute** request to object #1 (Anybus Object), Instance #1, Attribute #2 (Firmware version).

3.2.3 Object Categories

Based on their physical location, objects are grouped into two distinct categories:

Anybus Module Objects	These objects are part of the Anybus firmware, and typically controls the behavior of the module and its actions on the network.
Host Application Objects	These objects are located in the host application firmware, and may be accessed by the Anybus module. This means that the host application must implement proper handling of incoming object requests.

3.2.4 Standard Object Implementation

The standard object implementation has been designed to cover the needs of all major networking systems, which means that it is generally enough to implement support for these objects in order to get sufficient functionality regardless of network type.

Optionally, support for network specific objects can be implemented to gain access to advanced network specific functionality. Such objects are described separately in each network guide.

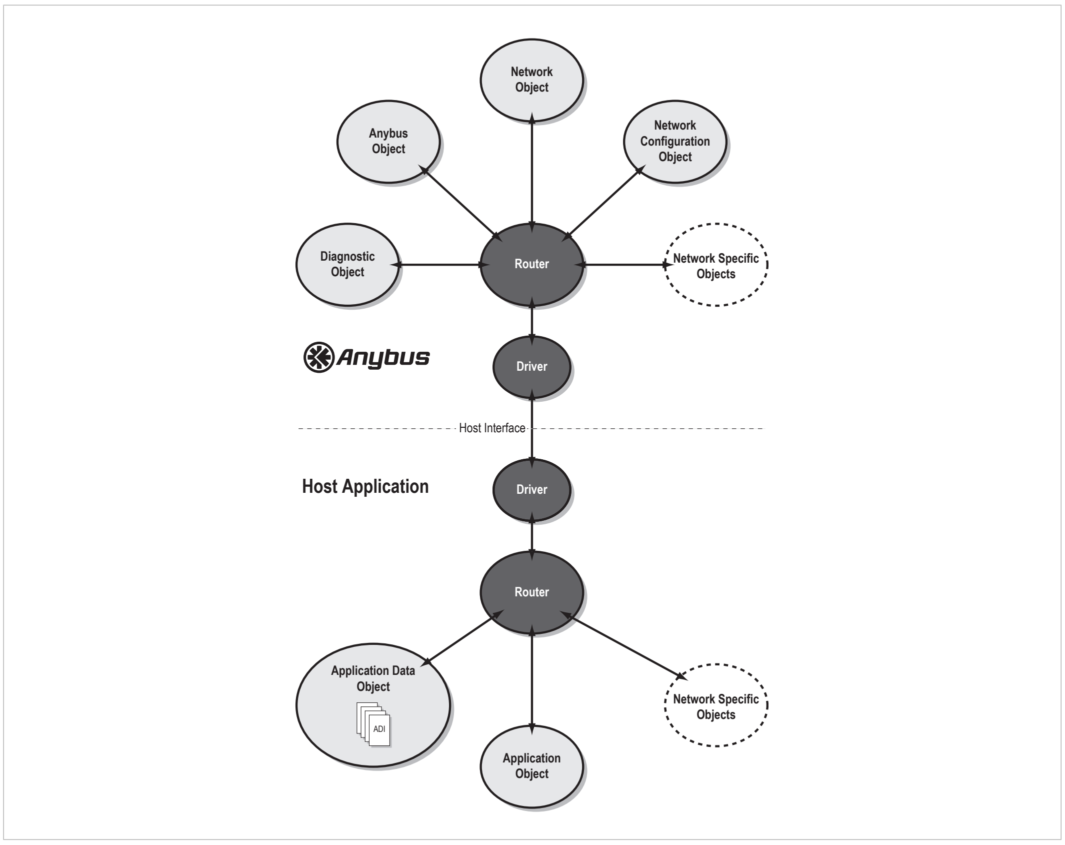


Fig. 4

Anybus Module Objects

The following objects are implemented in the standard Anybus CompactCom 30 firmware:

- Anybus Object
- Diagnostic Object
- Network Object
- Network Configuration Object
- Network Specific Objects

Exactly how much support that needs to be implemented for these objects depends on the requirements of the host application.

See also...

- *Anybus Module Objects, p. 48*

Host Application Objects

The following objects can be implemented in the host application.

- Application Data Object (Mandatory)
- Application Object (Optional)
- Network Specific Objects (Optional)

It is mandatory to implement the Application Data Object and the Application Object. The exact implementation however depends heavily on the requirements of the host application. Implementation of the Application Object is optional, but highly recommended.

See also...

- *Host Application Objects, p. 64*

3.3 Network Data Exchange

Data that is to be exchanged on the network is grouped in the Application Data Object. This object shall be implemented in the host application firmware, and each instance within it (from now on referred to as "ADI", i.e. Application Data Instance) represents a block of network data.

ADIs are normally associated with acyclic parameters on the network side. For example, on DeviceNet and EtherNet/IP, ADIs are represented through a dedicated vendor specific CIP object, while on PROFIBUS, ADIs are accessed through acyclic DP-V1 read/write services. On EtherCAT and other protocols that are based on the CANopen Object Dictionary, ADIs are mapped to PDOs, defined in the object dictionary.

ADIs can also be mapped as Process Data, either by the host application or from the network (where applicable). Process Data is exchanged through a dedicated data channel in the Anybus CompactCom host protocol, and is normally associated with fast cyclical network I/O. The exact representation of Process Data is highly network specific; for example on PROFIBUS, Process Data correlates to IO data.

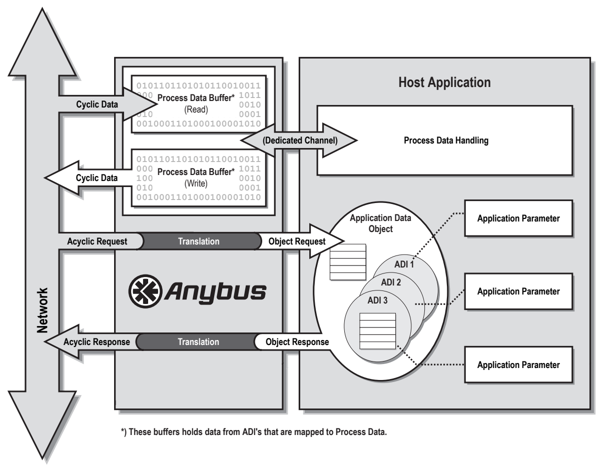


Fig. 5

Each ADI may be tagged with a name, data type, range and default value, all of which may be accessed from the network (if supported by the network in question). This allows higher level network devices (e.g. network masters, configuration tools etc.) to recognize acyclic parameters by their actual name and type (when applicable), simplifying the network configuration process.

Some networking systems allows both cyclic and acyclic access to the same parameter. In the case of the Anybus CompactCom 30, this means that an ADI may be accessed via explicit object requests and Process Data simultaneously. The Anybus module makes no attempt to synchronize this data in any way; if necessary, the host application must implement the necessary mechanisms to handle this scenario.

The Anybus interface uses little endian memory addressing. This means that the byte order is from the least significant byte (LSB) to the most significant byte (MSB). The Anybus will however handle ADI values transparently according to the actual network representation (indicated to the application during initialization). The application driver is responsible for byte swap if required. Use of this approach is decided because of the following reasons:

- The Anybus can not hold information about the data type of all ADIs due to memory limitations and start-up time demands.
- The alternative to read the data type prior to every parameter write or read request would be too time consuming.

See also...

- *Process Data Subfield, p. 21*
- *The Anybus State Machine, p. 32*
- *Network Object (03h), p. 58*
- *Application Data Object (FEh), p. 66*

3.4 Diagnostics

The Anybus CompactCom 30 features a dedicated object for host related diagnostics. To report a diagnostic event, the host application shall create an instance within this object. When the event has been resolved, the host simply removes the diagnostic instance again.

Each event is tagged with an Event Code, which specifies the nature of the event, and a Severity Code, which specifies the severity of the event. The actual representation of this information is highly network specific.

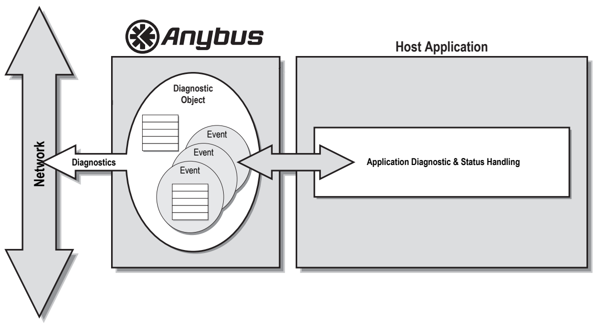


Fig. 6

See also...

- *Diagnostic Object (02h), p. 54*

3.5 Multilingual Support

Where applicable, the Anybus CompactCom 30 supports multiple languages. This mainly affects instance names and enumeration strings, and is based on the current language setting in the Anybus Object. Note that this also applies to Host Application Objects, which means that the host application should be capable of changing the language of enumeration strings etc. accordingly.

When applicable, the Anybus CompactCom 30 forwards change-of-language-requests from the network to the Application Object. It is then up to the host application to grant or reject the request, either causing the module to change its language settings or to reject the original network request.

Supported languages:

- English (default)
- German
- Spanish
- Italian
- French

See also...

- *Application Object (FFh), p. 75*

4 Host Communication Layer

4.1 General Information

4.1.1 Basic Principles

On its lowest level, the host interface communication is based on a half-duplex protocol. Telegrams are exchanged in a ping-pong fashion, with the only exceptions being the very first telegram issued by the host application during startup, and possible re-transmissions due to transmission errors (only applicable when using the serial host interface).

The host application and the Anybus CompactCom share an equal responsibility of maintaining a telegram exchange rate that provides an acceptable throughput, yet does not exceed the performance capabilities of the local CPU.

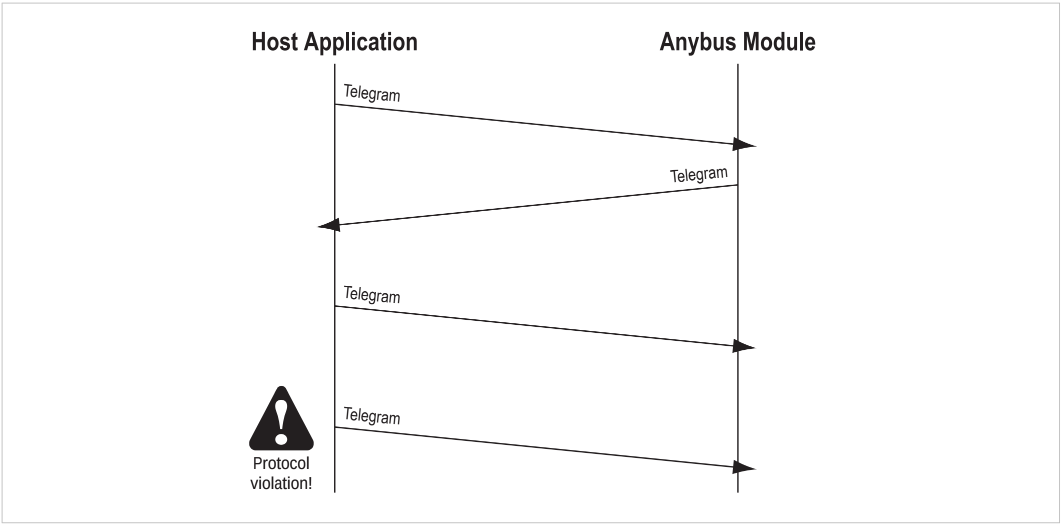


Fig. 7

The Anybus CompactCom will respond as quickly as possible based on present conditions, while the application should respond as fast as required to fulfil its own timing demands, or to fulfil the demands of the actual network, whichever is highest.

The parallel- and serial interfaces share the same basic protocol, with some slight differences to cover the needs for each interface (i.e. CRC for serial transmission etc.).

4.1.2 Telegram Contents

Each telegram consists of three subfields:

Handshake registers	This field controls the communication between the hsot and the Anybus CompactCom. See *Handshake Registers, p. 19*
Message subfield	This field is used for object messaging, an embedded sub-protocol which carries messages between the host application and the module. This field is present in all telegrams, however its contents may or may not be relevant depending on certain bits in the handshake registers. See *Object Messaging, p. 34*
Process Data Subfield	This field provides a dedicated channel for fast cyclical network I/O, known as process data. See *Process Data Subfield, p. 21*

4.2 Handshake Registers

4.2.1 Control Register (Read/Write)

This register controls the communication towards the Anybus CompactCom.

b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
CTRL_T	CTRL_M	CTRL_R	CTRL_AUX	-	-	-	-

Bit	Description
CTRL_T	The host application shall toggle this bit when sending a new telegram. CTRL_T must be set to “1” in the initial telegram sent by the application to the module.
CTRL_M	If set, the message subfield in the current telegram is valid. See *Message Fragmentation, p. 26*
CTRL_R	If set, the host application is ready to receive a new command. See *Flow Control, p. 37*
CTRL_AUX	See *Auxiliary Bit (STAT_AUX, CTRL_AUX), p. 20*
-	(reserved, set to zero)

4.2.2 Status Register (Read Only)

This register holds the current status of the Anybus CompactCom.

b7 (MSB)	b6	b5	b4	b3	b2	b1	b0 (LSB)
STAT_T	STAT_M	STAT_R	STAT_AUX	SUP	S2	S1	S0

Bit	Description			
STAT_T	When the module issues new telegrams, this bit will be set to the same value as CTRL_T in the last telegram received from the host application.			
STAT_M	If set, the message subfield in the current telegram is valid. See *Message Fragmentation, p. 26*			
STAT_R	If set, the Anybus CompactCom is ready to process incoming commands. See *Flow Control, p. 37*			
STAT_AUX	See *Auxiliary Bit (STAT_AUX, CTRL_AUX), p. 20*.			
SUP	Value: Meaning: 0: Module is not supervised. 1: Module is supervised by another network device.			
S[0... 2]	These bits indicates the current state of the module (see also *The Anybus State Machine, p. 32*).			
	S2	S1	S0	Anybus State
	0	0	0	SETUP
	0	0	1	NW_INIT
	0	1	0	WAIT_PROCESS
	0	1	1	IDLE
	1	0	0	PROCESS_ACTIVE
	1	0	1	ERROR
	1	1	0	(reserved)
	1	1	1	EXCEPTION

The Status Register shall be regarded as cleared at start-up. The first telegram issued by the host application must therefore not contain a valid message subfield since STAT_R is effectively cleared (0).

4.2.3 Supervised Bit (SUP)

While the Anybus State Machine reflects the state of the cyclic data exchange, the SUP-bit indicates the overall status of the network communication, including acyclic communication. For example, on CIP, this bit indicates that the master has a connection towards the Anybus CompactCom. This connection may be an I/O connection, or an acyclic (explicit) connection. In case of the latter, the communication will be “silent” for extended periods of time, and the state machine will indicate that the network is Idle. The SUP-bit will however indicate that there still is a connection towards the module.

Exactly how this functionality should be handled, the offered level of security, and how to correctly activate it is network specific and often depends on external circumstances, e.g. configuration of the network as well as other network devices. Whether use of the SUP-bit is appropriate must therefore be decided for each specific application and network.

See also ...

- *Status Register (Read Only), p. 19*

> **!** It is important to recognize that this bit reflects the network state at the time of transmission of each telegram. In case a message from the network, such as a parameter data request or other command, for some reason is delayed (for example if the application has cleared the CTRL_R-bit for some time), the application may have to take alternative actions depending on the situation.

4.2.4 Auxiliary Bit (STAT_AUX, CTRL_AUX)

By default, this bit is not used and shall be zero. Optionally, it is possible to specify additional functionality for this bit in the Anybus Object (Instance Attribute #15, ‘Auxiliary Bit’).

At the time of writing, the following functionality has been defined:

Default	Not used; STAT_AUX and CTRL_AUX shall be zero.
Changed Data Indication	In this mode, the CTRL_AUX- and STAT_AUX bits indicate if the process data in the current telegram has changed compared to the previous one. See *Changed Data Indication, p. 22*

See also ...

- See *Process Data Subfield, p. 21*, (Changed Data Indication)
- See *Anybus Module Objects, p. 48* (Anybus Object, instance attribute #15)

4.3 Process Data Subfield

4.3.1 General Information

There are two kinds of process data:

Read Process Data	Read process data represents data *received from* the network, and is present (if used) in telegrams received from the Anybus CompactCom.
Write Process Data	Write process data represents data that shall be *sent to* the network, and is present (if used) in telegrams sent to the Anybus CompactCom.

Exactly how the Process Data is represented on the network varies, for example on PROFIBUS, it is exchanged as IO data, while on CANopen, it is exchanged through PDOs.

> **!** The process data subfield does not exist until the module is fully initialized. This is generally not an issue when using the parallel host interface, however it is important to note that it affects the telegram length when using the serial host interface.
>
> The Anybus CompactCom does not perform any byte swapping on this data; the host application is thus solely responsible for byte swapping if needed by the implementation.

4.3.2 Process Data Mapping

The actual size and structure of the process data subfield is based on the process data map, which can be specified by the host application during startup, and in the case of certain Anybus implementations also from the network.

Process data mapping is basically a structured way of defining the cyclic I/O size on a higher level, and enables the host application to define data structures etc. by their actual data types instead of specifying anonymous blocks of binary data.

See also ...

- *Network Data Exchange, p. 15*
- *Network Object (03h), p. 58* (command details for Map_ADI_Write_Area and Map_ADI_Read_Area)
- *Application Data Object (FEh), p. 66* (command details for Remap_ADI_Write_Area and Remap_ADI_Read_Area)

4.3.3 Changed Data Indication

ⓘ *This functionality is optional and is not enabled by default.*

Optionally, it is possible to have a changed data indication with each telegram through the use of the auxiliary bits (STAT_AUX and CTRL_AUX). When this functionality is enabled, these bits are independently used as changed data signals for read and write process data respectively, as described below.

- Changed Data Indication of Read Process Data

  The Anybus CompactCom indicates if the Read Process Data has changed as follows.

  – STAT_AUX is set (1) in telegrams containing Read Process Data which differs from that of the last telegram which contained Read Process Data.

  – STAT_AUX is cleared (0) in telegrams containing Read Process Data which equals that of the last telegram which contained Read Process Data.

  – The first telegram containing Read Process Data will be compared to an ‘imaginary’ previous process data buffer containing all zeroes.

  – STAT_AUX is always set in the first telegram containing a Read Process Data map that has been modified by Remap_ADI_Read_Area.

  – The host application may choose to discard the Read Process Data in telegrams where STAT_AUX is cleared (0).

- Changed Data Indication of Write Process Data

  The host application indicates if the Write Process Data has changed as follows.

  – CTRL_AUX is set (1) in telegrams containing Write Process Data which differs from that of the last telegram which contained Write Process Data.

  CTRL_AUX is cleared (0) in telegrams containing Write Process Data which equals that of the last telegram which contained Write Process Data.

  The first telegram containing Write Process Data shall be compared to an ‘imaginary’ previous process data buffer containing all zeroes.

  CTRL_AUX shall always be set in the first telegram containing a Write Process Data map that has been modified by Remap_ADI_Write_Area.

  Setting CTRL_AUX (1) in telegrams where the Write Process Data is unchanged is acceptable although it may add extra load on the Anybus CompactCom.

See also ...

- *Anybus Object (01h), p. 49* (Instance Attribute #15)
- *Application Data Object (FEh), p. 66* (command details for Remap_ADI_Write_Area)

4.4 Anybus Watchdog

The host application can establish whether or not the module is working properly by measuring the telegram response time. If this time exceeds a certain value, the module can be assumed to be malfunctioning. The host application can then enter a safe state, reset the module, or take similar actions.

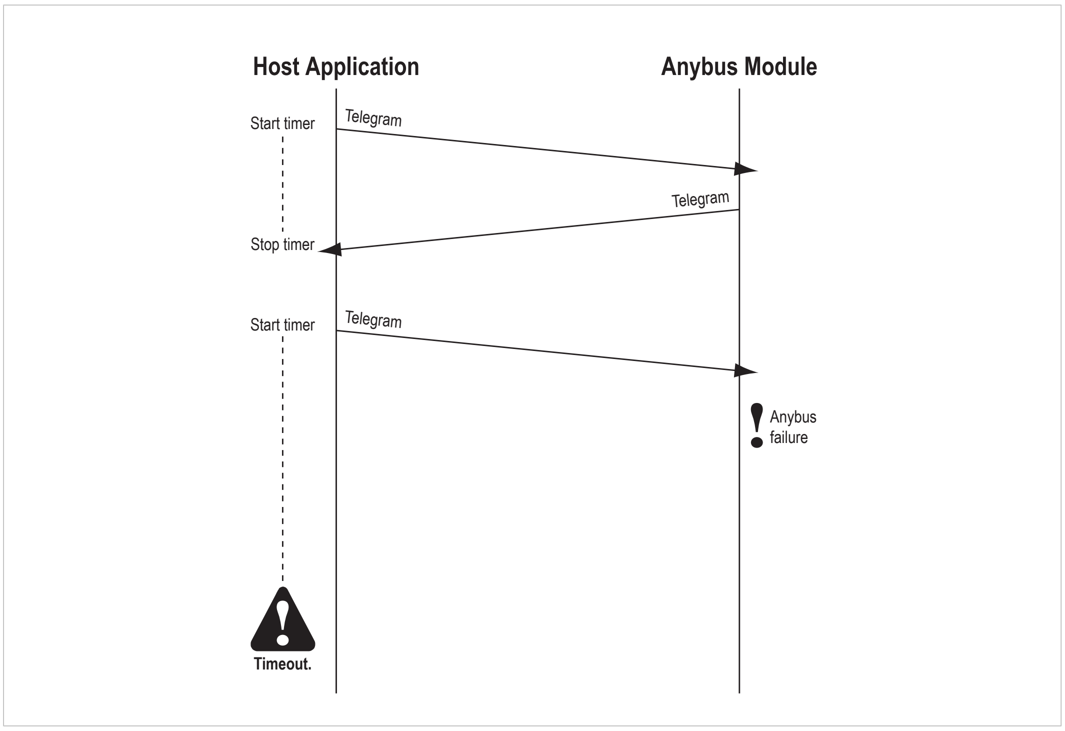


Fig. 8

Suggested minimum timeout values:

Interface	Minimum timeout value
Parallel	10 ms
Serial, 19.2 kbps	1050 ms
Serial, 57.6 kbps	360 ms
Serial, 115.2 kbps	180 ms
Serial, 625 kbps	60 ms

These are only suggested minimum values that are guaranteed to be met by the Anybus module in all cases. An appropriate value for a particular application depends highly on safety demands, quality of serial communications (if applicable) etc.

4.5 Application Watchdog

If enabled, this feature allows the Anybus CompactCom to establish whether or not the host application is working properly by measuring its telegram response time. If this time exceeds a specified value, the Anybus CompactCom considers the host application as not working and takes action accordingly.

The actions performed when a timeout occurs are network specific, however common to all Anybus CompactCom implementations is that the module shifts to the state EXCEPTION.

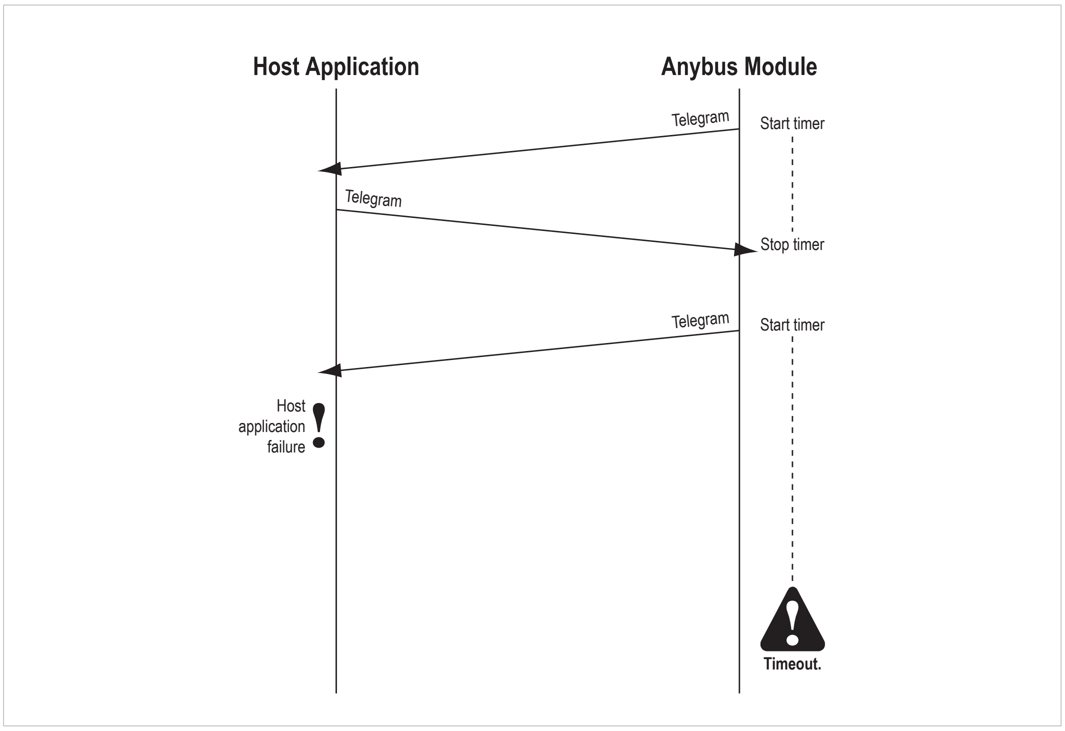


Fig. 9

When using the serial host interface, the watchdog timer is started when the module initiates a new transmission and stopped when the module receives a valid telegram that is not a re-transmission.

See also...

Anybus Object (01h), p. 49 (Instance Attribute #4, ‘Application watchdog timeout’)

4.6 Serial Host Communication

4.6.1 General Information

On the serial host interface, telegrams are transmitted through a common asynchronous serial interface. The baudrate is determined by certain signals on the host interface connector of the module; consult the Anybus CompactCom 30 Hardware Design Guide for further information.

Other communication settings are fixed to the following values:

Data bits:	8
Parity:	None
Stop bits:	1

! As is the case with most asynchronous communication devices, the actual baud rate used on the Anybus CompactCom may differ slightly from the ideal baud rate.

The baud rate deviation of the Anybus CompactCom is less than ±1.5%. To ensure proper operation, the baud rate deviation in the host application must not exceed ±1.5%.

4.6.2 Serial Telegram Frame

1 byte	16 bytes	Up to 256 bytes	2 bytes
Handshake register field	Message subfield	Process data subfield	CRC16
1st byte			(last byte)

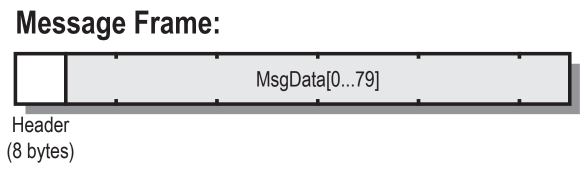
Handshake Register Field	This field contains the Control Register in telegrams sent *to* the Anybus CompactCom, and the Status Register in telegrams received *from* the Anybus CompactCom.
Message Subfield	To maintain throughput for the Process Data, the message subfield is limited to 16 bytes when using the serial interface. Longer messages are exchanged as several smaller fragments. See *Message Fragmentation, p. 26*.
Process Data Subfield	This field contains the Write Process Data in telegrams sent *to* the Anybus module, and the Read Process Data in telegrams received *from* the Anybus CompactCom. Note that this field does not exist when the Anybus CompactCom is operating in the 'SETUP'-state.
CRC16	This field holds a 16 bit Cyclic Redundancy Check (Motorola format, i.e. MSB first). The CRC covers the entire telegram except for the CRC itself. See *CRC Calculation (16–bit), p. 101*.

4.6.3 Message Fragmentation

To ensure Process Data throughput at all times, messages longer than 16 bytes are transmitted as multiple small (≤16 bytes) fragments. These fragments are then assembled by the receiving part and processed as a complete message, i.e. the receiver must wait until a complete message has been received before processing it. Note that this includes any fundamental errors that may have been detected in the first fragment (e.g. data field too large, command with error bit set (E=1) etc.).

Each message must be followed by an additional “empty” fragment (i.e. a telegram with CTRL_M or STAT_M=0) to indicate to the receiver that the message has been completed.

In the example below a message with 80 bytes of message data is sent to the module as 6 smaller fragments. An additional “empty” fragment ends the message.



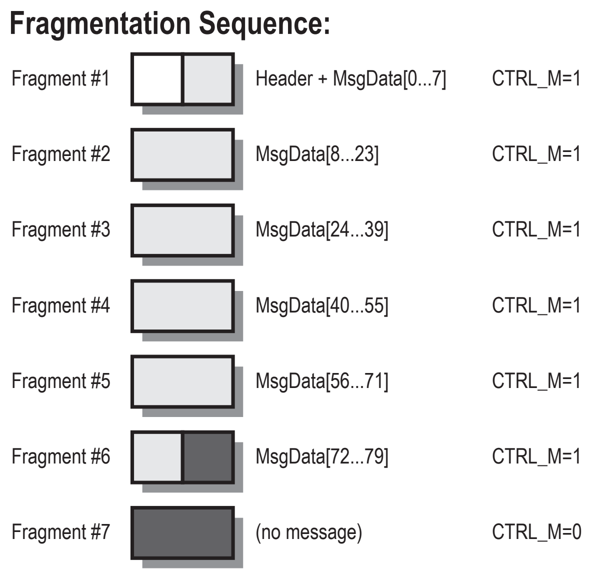


Fig. 10

A fragmented message may be aborted at any time by issuing an “empty” fragment (i.e. a telegram with CTRL_M or STAT_M=0), causing a fragmentation error at the receiving end.

The example shows a message with 72 bytes of message data, sent by the module as several smaller fragments. The message is aborted prematurely using an “empty” fragment.

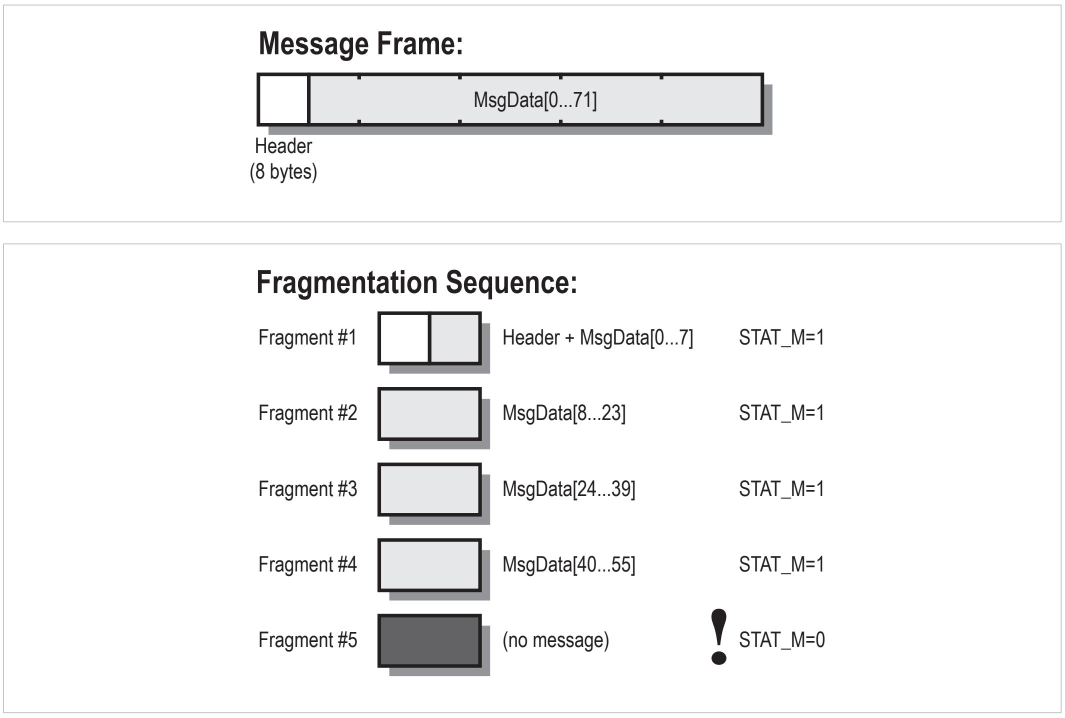


Fig. 11

Smaller messages (i.e. ≤16 bytes) are sent without fragmentation, however an additional “empty” fragment must still be sent to signal to the receiver that a full message has been completed. In the eample a small message (16 bytes) is sent to the host application. An additional “empty” fragment ends the message.

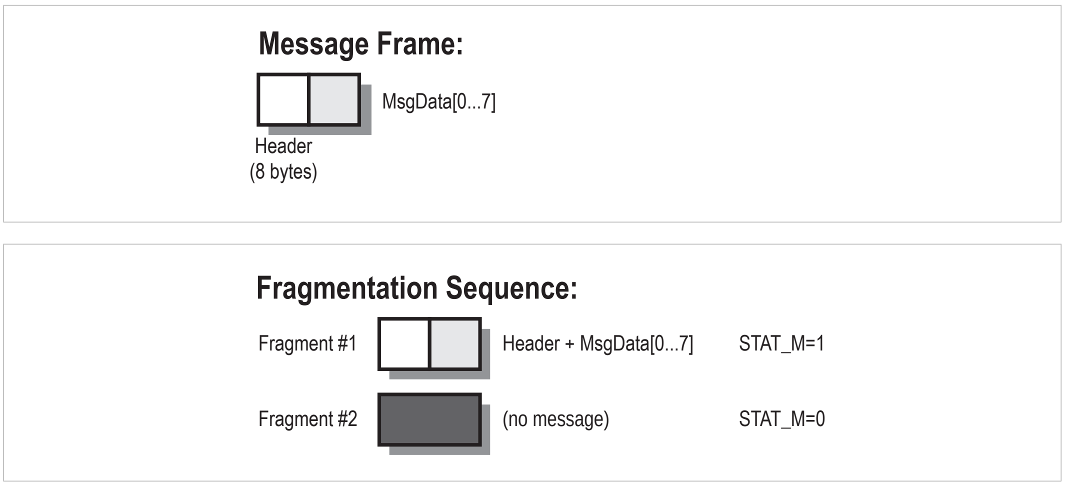


Fig. 12

4.6.4 Transmission Errors

Transmission errors are handled as follows:

- A telegram is ignored in the event of a reception error (i.e. a CRC mismatch).
- Any transmission error shall, after a timeout, result in a re-transmission of the last telegram by the host application (suggested retransmission timeout values for different baud rates are specified in the table below).
- The module detects re-transmissions by monitoring CTRL_T.

To make this concept successful, the host application must fulfil the following requirements:

- The telegram transmission time must be less than T_{SEND} (See table)
- The re-transmission timeout must not be less than $T_{TIMEOUT}$ (See table)

Baud Rate	T_{SEND}	$T_{TIMEOUT}$
19.2 kbps	175 ms	350 ms
57.6 kbps	60 ms	120 ms
115.2 kbps	30 ms	60 ms
625 kbps	6 ms	20 ms

! Note that it is the timeout itself that shall trigger the retransmission rather than the actual CRC error; it is not allowed to make a re-transmission directly after detecting a CRC error.

Scenario 1: Anybus CompactCom Detects a CRC Error

In this example, the Anybus CompactCom encounters a CRC mismatch. The erroneous telegram is simply ignored, causing a timeout in the host application, eventually forcing it to retransmit the original telegram.

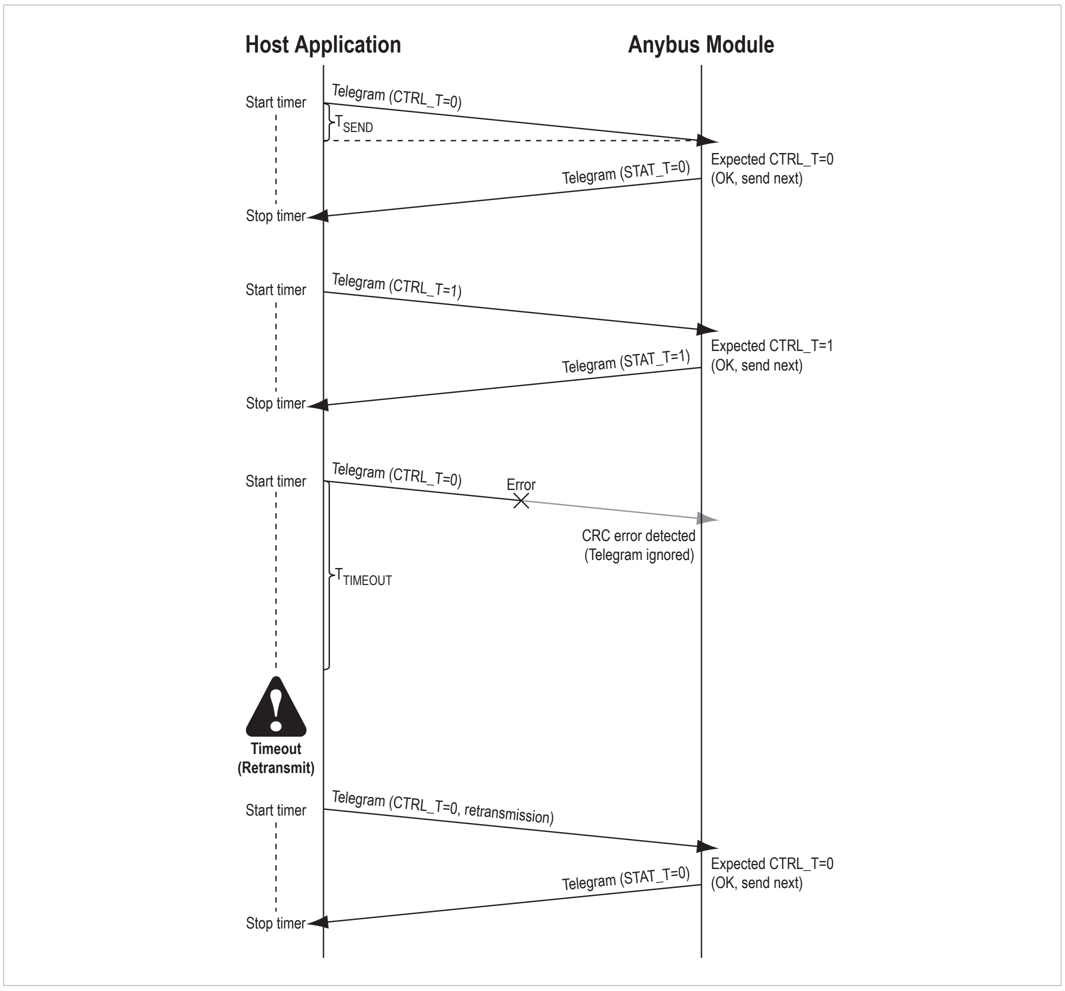


Fig. 13

Scenario 2: Error detected by Host

In this example, the host encounters a CRC mismatch. Just like the Anybus CompactCom, it must ignore the erroneous telegram, eventually causing a timeout, which shall in turn generate a re-transmission.

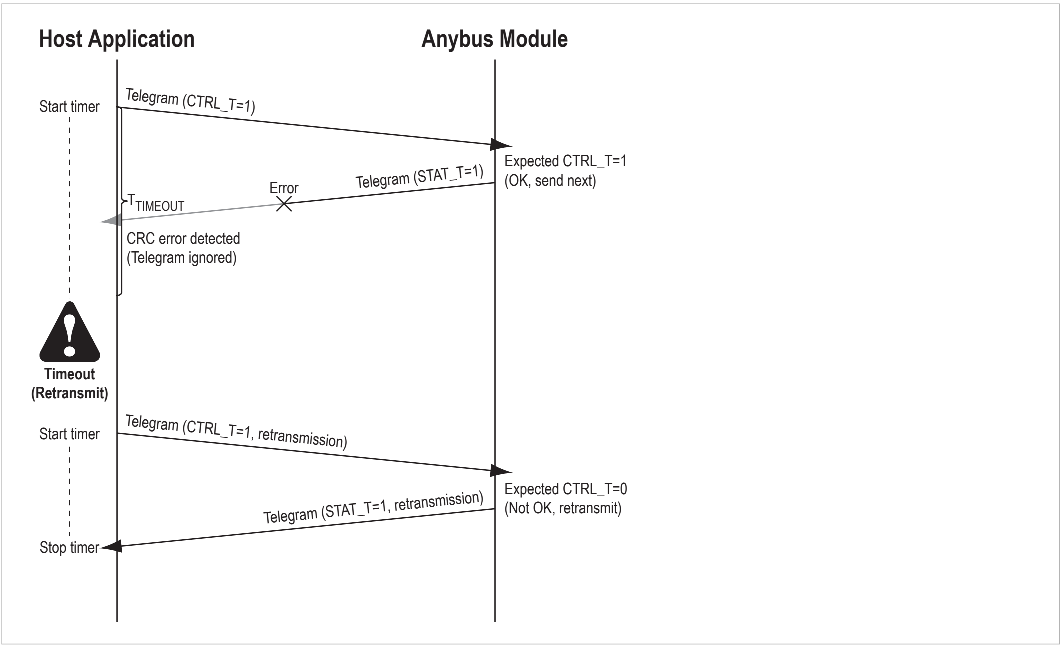


Fig. 14

!	Note that it is the timeout itself that shall trigger the re-transmission rather than the actual CRC-error; it is not allowed to make a re-transmission directly after detecting a CRC-error.

4.7 Parallel Host Communication

4.7.1 General Information

When using the parallel host interface, telegrams are exchanged via a shared memory area; the telegram subfields are simply accessed via pre-defined memory locations (see memory map below).

Telegram transmission is triggered via the control register, and the reception of a new telegram is indicated in the status register. This means that the process data and message sub-fields must be written *prior* to accessing the control register.

While waiting for reception of a telegram, any access to the parallel interface other than polling of the Status Register must be avoided.

See also ...

- *Handshake Registers, p. 19*

!	Internally, an interrupt is triggered in the module each time the control register is modified. It is therefore important not to use bit handling or other read-modify-write instructions directly on this register, since the module may interpret this as multiple accesses. All bit handling etc. must instead be performed in a temporary register, and written back.

4.7.2 Memory Map

The address offset specified below is relative to the base address of the module., i.e. from the beginning of the area in host application memory space where the parallel interface has been implemented.

Address Offset:	Area:	Access:	Notes:
0000h... 37FFh	(reserved)	-	(reserved for future use)
3800h... 38FFh	Process Data Write Area	Write Only	See *Process Data Subfield, p. 21*
3900h... 39FFh	Process Data Read Area	Read Only	See *Process Data Subfield, p. 21*
3A00h... 3AFFh	(reserved)	-	
3B00h... 3C06h	Message Write Area	Write Only	See *Object Messaging, p. 34*"
3C07h... 3CFFh	(reserved)	-	
3D00h... 3E06h	Message Read Area	Read Only	See *Object Messaging, p. 34*"
3E07h... 3FFDh	(reserved)	-	
3FFEh	Control Register	Read/Write	See *Handshake Registers, p. 19*
3FFFh	Status Register	Read Only	See *Handshake Registers, p. 19*

! C-programmers are reminded to declare the entire shared memory area as volatile.

4.7.3 Parallel Telegram Handling

On the parallel interface, transmission and reception is managed through the Handshake Registers.

See also ...

- *Handshake Registers, p. 19*

Telegram Transmission

To transmit a telegram, perform the following steps:

1. If applicable, write the Write Process Data to the Process Data Write Area (3800h...38FFh)
2. If applicable, write the message to the Message Write Area (3B00h...3C06h)
3. Update the Control Register to trigger the transmission.

Telegram Reception

Telegram reception is signalled by the STAT_T-bit in the Status Register. The host application can either poll the Status Register cyclically to detect new telegrams, or rely on interrupt operation.

Interrupt Operation (Highly Recommended)	If implemented, an interrupt is generated each time the module issues a new telegram. Generally, it is strongly recommended to use this feature as it can significantly reduce overhead compared to polling the Status Register cyclically.
Polled Operation	The host application must poll the Status Register cyclically and study the STAT_T-bit; when the value of this bit equals the value of the CTRL_T-bit, the Anybus CompactCom has issued a new telegram.

! To ensure valid results when polling the Status Register, it is required to use a read-until-two-consecutive-readings-agree procedure.

5 The Anybus State Machine

5.1 General Information

A fundamental part of the Anybus CompactCom 30 is the Anybus State Machine. At any given time, the state machine reflects the status of the module and the network, see *Status Register (Read Only), p. 19*.

The state machine shall be regarded as a Moore machine; i.e. the host application is not *required* to keep track of all state transitions, however it is *expected* to perform certain tasks in each state

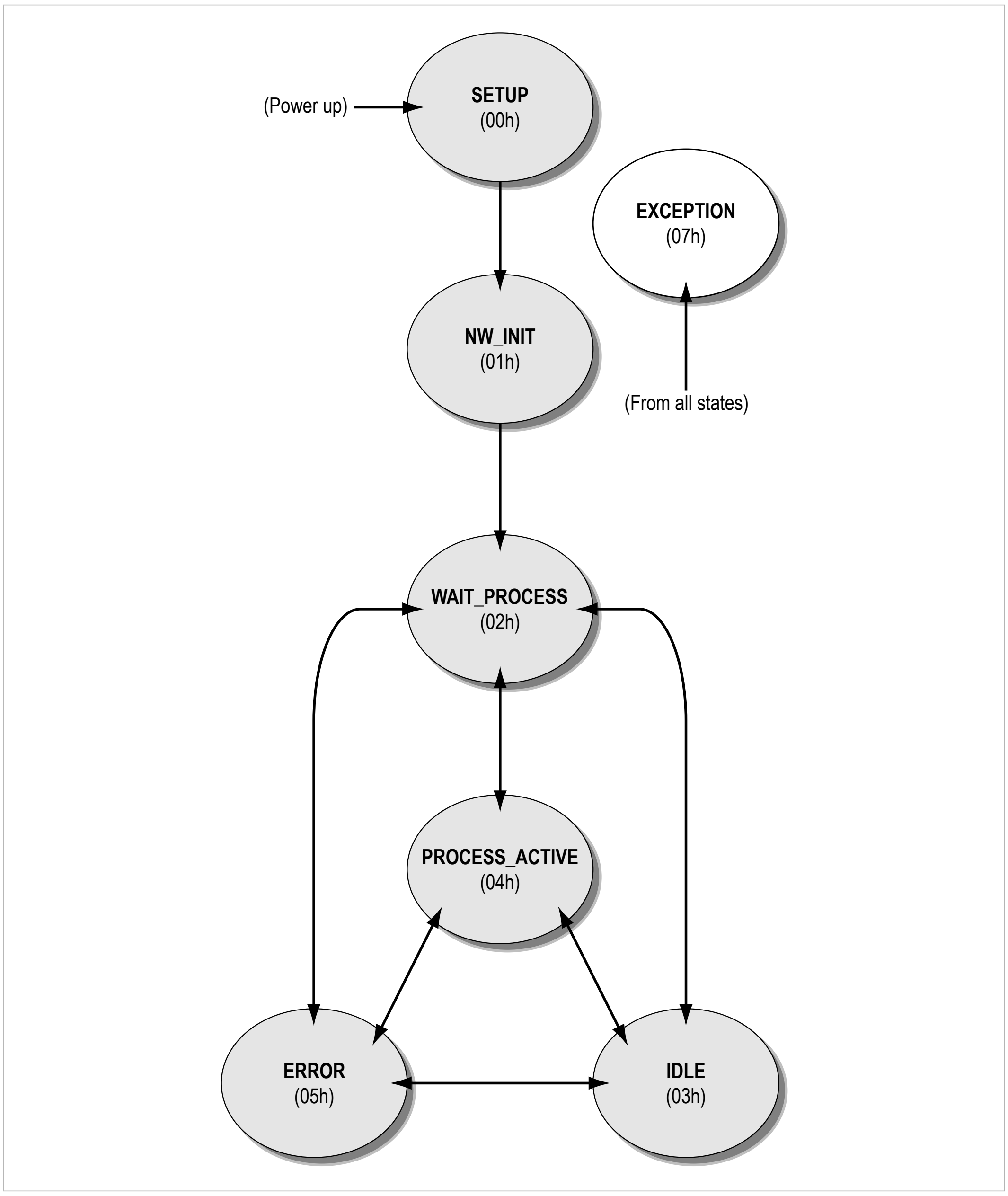


Fig. 15

5.2 State Dependent Actions

The expected actions for each state are listed below.

State	Description	Expected Actions
SETUP	Anybus CompactCom Setup in progress. The module may not send commands to the application in this state.	See *Anybus Setup (SETUP State), p. 46*
NW_INIT	The Anybus CompactCom module is currently performing network-related initialization tasks. Telegrams now contain Process Data (if such data is mapped), however the network Process Data channel is not yet active.	See .*Network Initialization (NW_INIT State), p. 47*
WAIT_PROCESS	The network Process Data channel is temporarily inactive.	The host application shall regard the Read Process Data as not valid.
IDLE	The network interface is idle. The exact interpretation of this state is network specific. Depending on the network type, the Read Process Data may be either updated or static (unchanged).	The host application may act upon the Read Process Data, or go to an idle state.
PROCESS_ACTIVE	The network Process Data channel is active and error free.	Perform normal data handling.
ERROR	There is at least one serious network error.	The Read Process Data shall be regarded as not valid. Optionally, the host application may perform network specific actions. Write Process Data could still be forwarded to the master, so the application must keep this data updated.
EXCEPTION	The module has ceased all network participation due to a host application related error. This state is unrecoverable, i.e. the module must be restarted in order to be able to exchange network data.	Correct the error if possible (details about the error can be read from the Anybus Object, see). When done, reset the Anybus CompactCom 30.

! The host application must keep the Write Process Data updated in NW_INIT (initial data), WAIT_PROCESS, IDLE, ERROR and PROCESS_ACTIVE since this data is buffered by the Anybus CompactCom, and may be sent to the network after a state shift.

See also ...

- *Network Configuration Object (04h), p. 62*

6 Object Messaging

6.1 General Information

6.1.1 Basic Principles

Object messaging involves two types of messages; commands and responses. On the message level, there is no master-slave relationship between the host application and the Anybus CompactCom module; both parts may issue commands, and are required to respond. Commands and responses are always associated with an instance within the Anybus object model. This can either be the object itself (addressed through instance #0), or an instance within it.

Commands can be issued at any time (provided that the receiving end is ready to accept new commands), while responses must only be sent as a reaction to a previously received command. Unexpected or malformed responses must always be discarded.

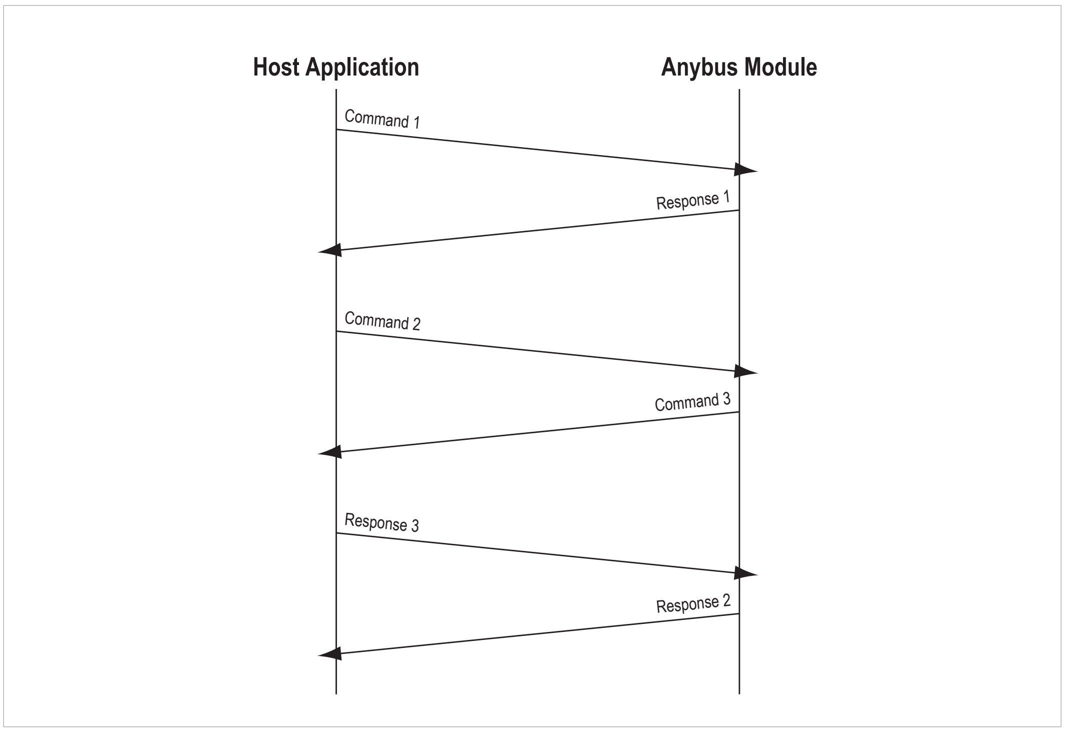


Fig. 16

Commands and responses are treated asynchronously, i.e. new commands may be issued before a response has been returned on the previous one. This also means that commands are not guaranteed to be executed in order of arrival, and that responses may return in arbitrary order (see figure). When necessary, the host application must wait for the response of any command to which the action or result may affect successive commands.

6.1.2 Source ID

To keep track of which response that belongs to which command, each message is tagged with a Source ID. When issuing commands, the host application may choose Source IDs arbitrarily, however when responding to commands issued by the Anybus module, the Source ID in the response must be copied from the original command.

See also ...

- *Message Layout, p. 36*

6.1.3 Error Handling

When a command for some reason cannot be processed, the receiver is still obliged to provide a response. In such case, an error shall be flagged in the header of the response message, together with an appropriate error code in the message data field.

The command initiator must then examine the response to see whether it is a successful response to the command or an error message. Error counters are available in the Anybus Object (01h).

If the network communication because of the error cannot continue, the module should enter the EXCEPTION state.

See also...

- *Message Layout, p. 36*
- *Error Codes, p. 40*
- *State Dependent Actions, p. 33*
- *Error Counters, p. 35*
- *Anybus Object (01h), p. 49*

6.1.4 Error Counters

The following error counters are available:

Error Counter	Abbr.	Description
Discarded commands	DC	Incremented if a command has been received with the R bit cleared
Discarded responses	DR	Incremented if the E bit in the response is cleared, but • the size or other data in the response (or similar) differs form what is expected, making the response unusable • the Source ID is not from the client sending the request • contains a command that was not sent by the client.
Serial reception errors	SE	Incremented for • a CRC error • a serial error, e.g. framing, overrun etc.
Fragmentation errors	FE	Incremented if a fragmentation error has occurred on the serial channel.

All error counters stop counting at FFFFh.

6.2 Message Layout

An object message consists of an 8 byte header followed by message related data.

Offset	Contents								Description
	b-7	b-6	b-5	b-4	b-3	b-2	b-1	b-0	
0	Source ID								See *Source ID, p. 34*
1	Object								Specifies a source/destination within the Anybus object model
2	Instance (lsb)								
3	Instance (msb)								
4	E								0: Message is either a Command, or a successful Response 1: Message is an Error Response
		C							0: Message is a Response 1: Message is a Command
			Command Code						See *Command Specification, p. 39*
5	Message Data Size								Size of the MsgData[] field in bytes (up to 255 bytes).
6	CmdExt[0]								Command-specific extension. See *Command Specification, p. 39*.
7	CmdExt[1]								
8...n	MsgData[0...n]								Message data field.

6.3 Data Format

6.3.1 Available Data Types

The Anybus CompactCom 30 uses the following data types as standard. Additional network specific data types are described in each separate network interface appendix (when applicable)

#	Type	Bits	Description	Range	Available on All Networks	Valid for Proc-ess Data
0	BOOL	8	Boolean	0 = False, !0 = True	Yes	Yes
1	SINT8	8	Signed 8 bit integer	-128... +127	Yes	Yes
2	SINT16	16	Signed 16 bit integer	-32768...+32767	Yes	Yes
3	SINT32	32	Signed 32 bit integer	-2^{31}... $+(2^{31}-1)$	Yes	Yes
4	UINT8	8	Unsigned 8 bit integer	0... +255	Yes	Yes
5	UINT16	16	Unsigned 16 bit integer	0... +65535	Yes	Yes
6	UINT32	32	Unsigned 32 bit integer	0... $+(2^{32}-1)$	Yes	Yes
7	CHAR	8	Character (ISO 8859-1)	0... +255	Yes	No
8	ENUM	8	Enumeration	0... +255	Yes	Yes
16	SINT64	64	Signed 64 bit integer	-2^{63}... $+(2^{63}-1)$	No	Yes
17	UINT64	64	Unsigned 64 bit integer	0... $+(2^{64}-1)$	No	Yes
18	FLOAT	32	Floating point (IEC 60559)	±1.17549435E-38... ±3.40282347E+38	No	Yes

- Arrays of type CHAR will be translated to the native string type of the network.
- The commands “Set_Indexed_Attribute” and “Get_Indexed_Attribute” cannot be used for the data type CHAR .
- Data of type ENUM are enumerations, limited to a consecutive range of values starting at zero.

6.3.2 Handling of Array of Char (Strings)

This section is mainly applicable when using arrays of CHAR in ADIs.

Arrays of type CHAR will be translated to the native string type (when applicable). The maximum string length, and the buffer space required to store it, is defined by the data type and the number of elements.

All elements of an array of CHAR are significant; the Anybus module does not expect any termination characters when reading, nor does it generate any when writing. The actual length of the string is defined in the payload size given in the commands Get_Attribute and Set_Attribute.

Generally, it is recommended to keep the “number of elements”, “data type”, and the message payload length, as consistent as possible. There is no guarantee that the Anybus module will check consistency between the payload length and the actual buffer space.

See also ...

- *Application Data Object (FEh), p. 66*

6.4 Flow Control

In some cases, the host application or the Anybus CompactCom may be busy or for some other reason temporarily unable to process new commands. To deal with such situations, CTRL_R and STAT_R are used to give the receiving part the possibility to signal whether it is ready to handle new commands or not.

These following rules apply:

- Responses are not blocked by these bits, i.e. when issuing a command, the host application must always have enough free resources to process a response to that command.
- CTRL_R and STAT_R are only checked *between* messages; i.e. when using the serial host interface, it is not possible to pause or cancel an ongoing message fragmentation using these bits.
- If the host application issues a command while STAT_R is cleared (0), that command will be discarded by the module, in turn causing the DC error counter to be incremented.

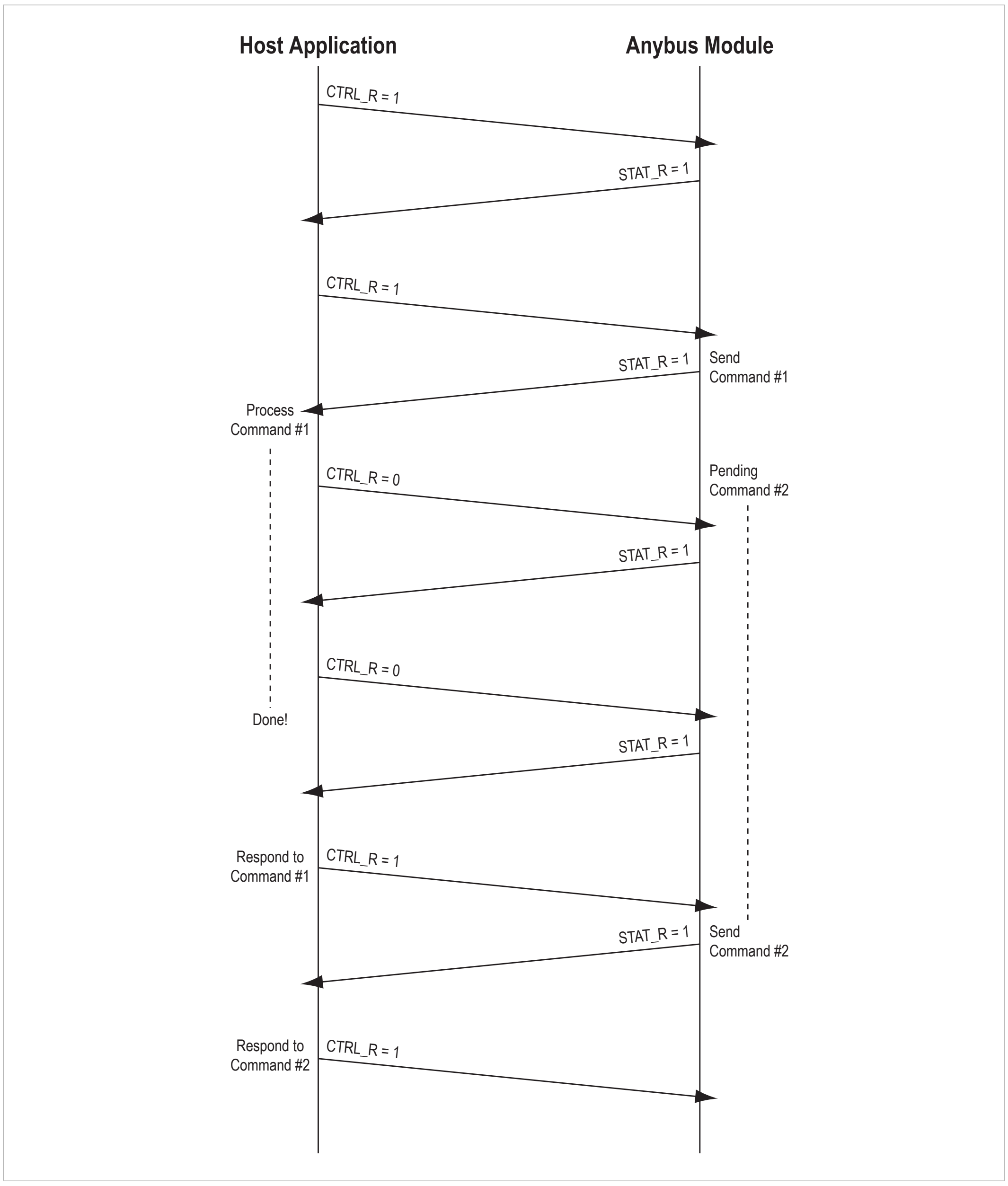


Fig. 17

See also ...

- *Handshake Registers, p. 19*
- Instance attribute #8 in *Anybus Object (01h), p. 49*

6.5 Command Specification

6.5.1 General Information

This chapter covers global commands, i.e. commands which have the same command code regardless of which object that is being accessed.

Some objects have special requirements, which are handled through object-specific commands. In such cases, unlike global commands, the same command code may have different meaning depending on context (i.e. which object that is being accessed). Object specific commands are described separately for each object (when applicable).

See also...

- *Anybus Module Objects, p. 48*
- *Host Application Objects, p. 64*

Regarding generic command descriptions it should be noted that while a command has a defined generic description and structure, the actual effect of it may differ greatly depending on the context.

For example:

- Application issues Reset →Network Configuration Object = resets network settings
- Network Reset →Anybus issues Reset →Application Object = Anybus shifts to EXCEPTION and awaits a hardware reset

! Fields marked as reserved must be treated with caution. Reserved fields in messages sent to the Anybus CompactCom must be set to 0 (zero), since they may have a defined use in future Anybus revisions. In messages received from the Anybus CompactCom, reserved fields shall simply be ignored.

6.5.2 Command Codes

The following commands are global, i.e. the same command code is used regardless of which object that is being accessed. The commands are described in the subsections below.

Command Code	Command Name
00h	(reserved)
01h	Get_Attribute
02h	Set_Attribute
03h	Create
04h	Delete
05h	Reset
06h	Get_Enum_String
07h	Get_Indexed_Attribute
08h	Set_Indexed_Attribute
09h... 0Fh	(reserved)
10h... 30h	(reserved for object specific commands)
31h... 3Eh	(reserved)
3Fh	(reserved for object specific commands)

6.5.3 Error Codes

If a command for some reason cannot be executed, the first byte in message data field (MsgData [ ]) of the response is used to supply details about problem to the command initiator.

Additional object specific error information may also be added in the message data section.

Value	Error	Meaning
00h	(reserved)	-
01h		
02h	Invalid message format	Command and error bit set
03h	Unsupported object	Object not registered
04h	Unsupported instance	The target instance does not exist
05h	Unsupported command	The target object does not support the specified command
06h	Invalid CmdExt[0]	Invalid value of CmdExt[0] or invalid combination of CmdExt [0] and CmdExt[1}
07h	Invalid CmdExt[1]	Invalid setting in CmdExt[1]
08h	Attribute not Set-able	The requested attribute is not Set-able
09h	Attribute not Get-able	The requested attribute is not Get-able
0Ah	Too Much Data	Too much data in message data field
0Bh	Not Enough Data	Not enough data in message data field
0Ch	Out of range	A specified value is out of range
0Dh	Invalid state	The command is not supported in the current state
0Eh	Out of resources	The target object cannot execute the command due to limited resources
0Fh	Segmentation failure	Invalid handling of the segmentation protocol
10h	Segmentation buffer overflow	Too much data received
11h... FEh	(reserved)	-
FFh	Object Specific Error	The object returned extended error information. Additional details may or may not be included in the message data field (MsgData[0...n])

6.5.4 Get_Attribute

Details

Command Code:	01h
Valid For:	(depends on context)

Description

This command retrieves the value of an attribute. The attribute number must be left intact in an error response.

- Command details:

Field	Contents
CMDExt[0]	Attribute number
CMDExt[1]	(reserved)

- Response details:

Field	Contents
MsgData[0..n]	Attribute Value

6.5.5 Set_Attribute

Details

Command Code:	02h
Valid For:	(depends on context)

Description

This command assigns a value to an attribute. The attribute number must be left intact in an error response

- Command details:

Field	Contents
CMDExt[0]	Attribute number
CMDExt[1]	(reserved)
MsgData[0..n]	Attribute Value

- Response details:

  (No data)

6.5.6 Create

Details

Command Code:	03h
Valid For:	Object Instance (Instance #0)

Description

This command creates a new instance within the object. If successful, the data portion of the response contains the number of the newly created instance.

- Command details:

  Object Specific

  Not all objects have any specific details for this command. If there are any object specific details, they are found in the description of the object in question.

- Response details:

Field	Contents
MsgData[0]	The number of the created Instance (low byte)
MsgData[1]	The number of the created Instance (high byte)

6.5.7 Delete

Details

Command Code:	04h
Valid For:	Object Instance (Instance #0)

Description

This command deletes a previously created instance (see above). If successful, all resources occupied by the specified instance will be released.

- Command details:

Field	Contents
CMDExt[0]	Instance number to delete (low byte)
CMDExt[1]	Instance number to delete (high byte)

- Response details (Success):

  (No data)

- Response details (Error):

Field	Contents
Invalid CMDExt[0]	The specified instance does not exist.

6.5.8 Reset

Details

Command Code:	05h
Valid For:	(depends on context)

Description

This command performs a reset command on an object.

- Command details:

Field	Contents
CMDExt[0]	(reserved)
CMDExt[1]	00h = Power-on reset (actual power-on or simulated) 01h = Factory default reset 02h = Power-on + Factory default reset

- Response details:

  (No data)

6.5.9 Get_Enum_String

Details

Command Code:	06h
Valid For:	(depends on context)

Description

This command retrieves attributes which are of enumeration type (ENUM). The returned value is the literal string associated with the specified enumeration value.

- Command details:

Field	Contents
CMDExt[0]	The number of the attribute
CMDExt[1]	The enumeration value

- Response details (Success):

Field	Contents
MsgData[0..n]	The enumeration string.

- Response details (Error):

Field	Contents
Invalid CMDExt[0..n]	The enumeration value is out of range.

6.5.10 Get_Indexed_Attribute

Details

Command Code:	07h
Valid For:	(depends on context)

Description

This command retrieves the value of a single element of an attribute which consists of multiple elements (i.e. an array). Note that this command cannot be used to access attributes of type CHAR.

- Command details:

Field	Contents
CMDExt[0]	The number of the attribute
CMDExt[1]	Index (first element has index 0)

- Response details (Success):

Field	Contents
MsgData[0..n]	Value

- Response details (Error):

Field	Contents
Invalid CMDExt[0..n]	Index is out of range

6.5.11 Set_Indexed_Attribute

Details

Command Code:	08h
Valid For:	(depends on context)

Description

This command assigns a value to a single element of an attribute which consists of multiple elements (i.e. an array). Note that this command cannot be used to access attributes of type CHAR.

- Command details:

Field	Contents
CMDExt[0]	The number of the attribute
CMDExt[1]	Index (first element has index 0)
MsgData[0...n]	Value to set

- Response details (Success):

  (No data)

- Response details (Error):

Field	Contents
Invalid CMDExt[1]	Index is out of range

7 Initialization and Startup

7.1 General Information

Before network participation, the following steps must be completed:

1. Initial Handshake

   The purpose of the startup procedure is to make sure that both parts (the host application and the Anybus CompactCom module) are ready to communicate. Normally an Anybus CompactCom module is ready to communicate in less than 1.5 s. The module will then enter the state SETUP. For more information, see *Initial Handshake, p. 45*.

2. Anybus CompactCom Setup

   This step determines how the module shall operate. When done, the module will enter the state NW_INIT.

   For more information, see *Anybus Setup (SETUP State), p. 46*.

3. Network Initialization

   At this stage, the module will attempt to read and evaluate information from the host application. When finished, the module will enter the state WAIT_PROCESS.

   For more information, see *Network Initialization (NW_INIT State), p. 47*.

! When the module is restarted after a firmware download, the application must wait for the upgrade to finish, before anything else is done, see below.

7.2 Initial Handshake

The startup procedure is slightly different depending on which type of host interface that is used.

7.2.1 Parallel Host Interface

The parallel host interface has two alternatives to handle the initial handshake:

- The host application shall wait 1.5 s after reset, followed by reading the Status Register.
- (Fast startup) When an interrupt arrives the host application immediately reads the Status Register and then enters the state SETUP.

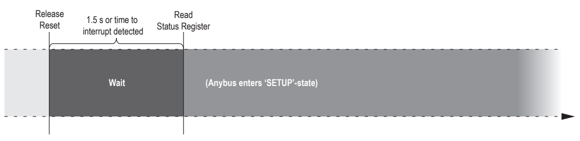

Release Reset
1.5 s or time to interrupt detected
Read Status Register
Wait
(Anybus enters 'SETUP'-state)

Fig. 18

7.2.2 Serial Host Interface

The host application must wait at least 1.5s after reset before it sends its first telegram.

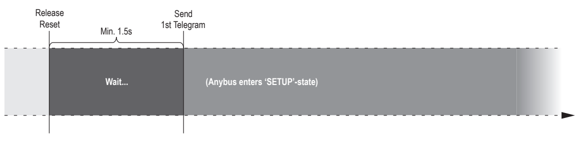


Fig. 19

The Status Register shall be regarded as cleared at startup; this means that the first telegram must not contain any message data since the STAT_T-bit is considered to be set to 0 (zero).

7.3 Anybus Setup (SETUP State)

This stage involves four distinctive steps:

1. Gather information about the Anybus CompactCom 30 (Optional)

   The host application may retrieve the network type, as well as other properties that may be relevant when configuring the module, from the Anybus Object (01h). This information may also be used to select different implementations based on e.g. the module type value.

2. Network Configuration (Optional)

   At this stage, the host application should update all instances in the Network Configuration Object of which the value originates from physical switches (i.e. node address, baud rate etc.). Settings which originate from “soft” input devices such as a keypad and display should not be updated at this point.

3. Process Data Mapping (Optional)

   The host application may optionally map ADIs to Process Data.

   This step is optional, but may be required by some networking systems and/or Anybus CompactCom implementations.

   Certain Anybus CompactCom implementations may attempt to alter the Process Data map during runtime. For more information, see application data object.

4. Finalize the Setup

   The setup procedure is finalized by setting the attribute Setup Complete in the Anybus Object (01h) to TRUE.

   If successful, the module now shifts to the state NW_INIT (below), or in case of failure, to the state EXCEPTION. In case of the latter, further information can then be read from the attribute Exception in the Anybus Object (01h).

See also..

- *Network Data Exchange, p. 15*
- *The Anybus State Machine, p. 32*
- *Anybus Object (01h), p. 49*
- *Network Configuration Object (04h), p. 62*

7.4 Network Initialization (NW_INIT State)

At this stage, the Anybus CompactCom 30 will attempt to read and evaluate information from the host application. The host application is required to respond to incoming requests to Host Application Objects. If the requested object or attribute is not implemented in the host application, simply respond with an error message. The module will in those cases use its own default values for the requested attributes, or configured virtual attributes.

The host application is free to update any instances in the Network Configuration Object, including those that do not originate from physical switches.

If a serious error is encountered (i.e. any error which prevents the module from proceeding) in this state, the module will shift to the state EXCEPTION. Further information can then be read from the attribute Exception in the Anybus Object (01h).

When done, the module enters the state WAIT_PROCESS.

> **!** The transition to this state is critical, especially if using the serial host interface, since telegrams from this point may (depending on the setup) contain Process Data. It is important to keep Write Process Data updated in this state since this data is buffered by the module and may be sent to the network on the next state transition.

See also..

- *The Anybus State Machine, p. 32*
- *Network Configuration Object (04h), p. 62*

8 Anybus Module Objects

8.1 General Information

The objects in this chapter are implemented as standard in all Anybus CompactCom implementations. Their functionality is categorized to indicate when and how to use the objects.

See also...

- *Data Format, p. 36*
- *Error Codes, p. 40*
- *Categorization of Functionality, p. 81*

For detailed information about each object, see...

- *Anybus Object (01h), p. 49*
- *Diagnostic Object (02h), p. 54*
- *Network Object (03h), p. 58*
- *Network Configuration Object (04h), p. 62*

8.2 Object Revisions

The purpose of the Object Revision attribute is to make it possible for the host application to determine if the revision of the object in the Anybus module is compatible with the software implementation in the host application, and/or to make it possible to choose different implementations based on the object revision.

As a general rule, the object revision is updated when the object is changed in such a way that the change may cause compatibility issues in the host application software implementation. Minor changes, such as when an attribute or command has been added, are normally not cause for a revision change.

! The definition of the Object Revision attribute has changed during early development. This means that Object Revisions in early Anybus CompactCom implementations has been incremented also even for minor changes. When applicable, a note will be added to each network interface appendix stating from which firmware revision the present definition is used.

In case of questions, please contact www.anybus.com/support.

8.3 Anybus Object (01h)

Category

Basic

Object Description

This object assembles data about the Anybus CompactCom 30 itself. The data in question does not as such represent the industrial network the Anybus CompactCom is adapted to, but describes data inherent to the module. This data is available for use in the application. The values may differ, depending on industrial network, and are in that case described in the respective appendices.

Most attributes in this object have access type "get" where data can be fetched using the command Get_ Attribute. The only attribute that is mandatory to set is "Setup complete" (instance #1, attribute #5), which is used by the application to notify the module that it has finished the setup. If the configuration is not accepted, the module will shift to the state EXCEPTION, and information can be read from instance #1, attribute #6 (Exception Code).

Supported Commands

Object:	Get_Attribute (01h)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Get_Enum_String (06h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Anybus"
2	Revision	Get	UINT8	04h
3	Number of instances	Get	UINT16	0001h
4	Highest instance no.	Get	UINT16	0001h

Instance Attributes (Instance #1)

#	Name	Access	Data Type	Description
1	Module Type	Get	UINT16	Value: Meaning: 0401h: Standard Anybus CompactCom 30 0402h: Anybus CompactCom Drive Profile 30 (Other) (reserved for future products)
2	Firmware version	Get	struct of: UINT8 Major UINT8 Minor UINT8 Build	Firmware version. Note that this value shall generally not be used to determine if a particular functionality is available or not. Please use the attribute Revision of each individual object for this purpose
3	Serial number	Get	UINT32	Unique serial number

#	Name	Access	Data Type	Description
4	Application watchdog timeout	Get/Set	UINT16	Application watchdog configuration Value: Meaning: 0: Disabled (default) (other): Timeout value (ms) If enabled, the watchdog timeout time is active immediately, regardless of the state of the application. The internal timer is reloaded every time it is restarted, so the value of this attribute can be changed during runtime.
5	Setup complete	Get/Set	BOOL	This attribute shall be set to TRUE when the Anybus Setup stage has been completed. If the configuration is accepted, the Anybus CompactCom shifts to the state NW_INIT. If not, i.e. if a serious error is detected in the configuration, the module will shift to the state EXCEPTION. In such case further information can be read from the attribute Exception Code (below) See also... *Anybus Setup (SETUP State), p. 46*
6	Exception code	Get	ENUM	See Exception Codes below.
7	FATAL event	Get/Set	struct of: (HMS Specific)	The latest FATAL event (if any) is logged to this instance. Used for evaluation by HMS support. (The contents of this attribute is only used as input to HMS support during application development)
8	Error Counters	Get	struct of:	Error counters (stops counting at FFFFh). (The contents of this attribute is only used during application development.)
			UINT16 DC	DC: Discarded commands (received with R = 0)
			UINT16 DR	DR: Discarded (unexpected) responses
			UINT16 SE	SE: Serial reception errors
			UINT16 FE	FE: Fragmentation errors
9	Language	Get/Set	ENUM	Current language: Value: Enumeration String: 00h: “English” (default) 01h: “Deutsch” 02h: “Español” 03h: “Italiano” 04h: “Français” See also... Application object , including details for command Change_ Language_request.
10	Provider ID	Get	UINT16	Preprogrammed and stored permanently in FLASH by HMS during production (contact HMS for further information). Value: Meaning: 0001h: HMS Industrial Networks FFFFh: (reserved) Other: Provider specific
11	Provider specific info	Get/Set	UINT16	The information stored in this attribute is provider-specific, i.e. it has no predefined meaning and is not evaluated nor used by the Anybus module. Any value written to this attribute will be stored in nonvolatile memory. Default value is 0000h.

#	Name	Access	Data Type	Description
12	LED colors	Get	struct of: UINT8 LED1A UINT8 LED1B UINT8 LED2A UINT8 LED2B	This attributes specifies the colors of the network status LEDs. See Anybus CompactCom M30 Hardware Design Guide for more information. Value: Meaning: 00h: None (not used) 01h: Green 02h: Red 03h: Yellow 04h: Orange 05h: Blue 06h: White
13	LED status	Get	UINT8	Bit field holding the current state of the network status LEDs as follows: Bit: Contents: b0: LED1A status (0=OFF, 1=ON) b1: LED1B status (0=OFF, 1=ON) b2: LED2A status (0=OFF, 1=ON) b3: LED2B status (0=OFF, 1=ON) b4... 7: (reserved)
14	(reserved)	-	-	-
15	Auxiliary Bit	Get/Set	UINT8	See also... • *Auxiliary Bit (STAT_AUX, CTRL_AUX), p. 20* • table below
16	GPIO configuration	Get/Set	UINT16	This attribute makes it possible to use the host interface GPIO pins for special purposes. See below for more information Default: 0000h

Exception Codes

When in the state EXCEPTION, this attribute provides additional information.

#	Enumeration String	Description
00h	No exception	-
01h	Application timeout	The host application has not responded within the specified watchdog timeout period.
02h	Invalid device address	The selected device address is not valid for the actual network.
03h	Invalid communication settings	The selected communication settings are not valid for the actual network.
04h	Major unrecoverable app event	The host application has reported a major unrecoverable event to the Diagnostic object.
05h	Wait for reset	The module is waiting for the host application to execute a reset.
06h	Invalid process data config	The Process Data configuration is invalid.
07h	Invalid application response	The host application has provided an invalid response to a command.
08h	Nonvolatile memory checksum error	At least one of the parameters stored in nonvolatile memory has been restored to its default value due to a checksum error.
09h	Safety module error	Something is wrong with the safety module. More information can be found in the Exception Information attribute in the Functional Safety Object. The only Anybus CompactCom 30 series module to implement functional safety is Anybus CompactCom 30 PROFINET 2-Port, the Functional Safety Objects are described in the network interface appendix for that module.
0Ah	Insufficient application implementation	The application does not implement the functionality required for the Anybus module to continue its operation.
(other)	(reserved)	-

See also...

- *The Anybus State Machine, p. 32*

Auxiliary Bit

This attribute defines the function for the CTRL_AUX and STAT_AUX bits.

#	Function	CTRL_AUX	STAT_AUX
00h	None (default)	No function, set to 0 (zero).	No function, always 0 (zero).
01h	Changed Data indication	0: Write Process Data unchanged 1: Write Process Data updated	0: Read Process Data unchanged 1: Read Process Data updated
(other)	(reserved)	-	-

See also...

- *Handshake Registers, p. 19*
- *Auxiliary Bit (STAT_AUX, CTRL_AUX), p. 20*

Object Specific Error Codes

The following object-specific error codes may be returned by the module as a response to setting the attribute Setup complete.

#	Error	Description
01h	Invalid process data configuration	The Process Data configuration is invalid
02h	Invalid device address	The selected device address is not valid for the actual network
03h	Invalid communication settings	The selected communication settings are not valid for the actual network

GPIO Configuration

This attribute makes it possible to use the GPIO (General Purpose IO) pins for other purposes than general IO. The attribute will have to be set during setup for any changes to take effect. The functionality is not introduced in all members of the Anybus CompactCom 30 product family, please consult the network appendices for more information

Please note that the general output pins are defined as active low.

#	Function	Description
0000h	Standard	GIP[0..1] are used as general input pins. GOP[0..1] are used as general output pins.
0001h	Extended LED functionality	GIP[0..1] are used as network specific, active low LED outputs associated with the left, or single port. GOP[0..1] are used as network specific, active low LED outputs associated with the right port on dual port modules.
(other)	(reserved)	

8.4 Diagnostic Object (02h)

Category

Specific to each industrial network, see network guides.

Object Description

This object provides a standardized way of reporting diagnostic events to the network. Exactly how this is represented on the network differs, however common to all implementations is that the module enters the state EXCEPTION in case of a major unrecoverable event.

When the module has been started and initialized, no instances exist in the module. When a diagnostic event, e.g. a blown fuse, occurs in the application, the application creates an instance with information on severity and kind of event. The information in this instance remains available for the application, until the application deletes the instance. The event code in the instance is processed by the module, to transfer correct network-specific information about the event to the network used.

Supported Commands

Object:	Get Attribute (01h)
	Create (03h)
	Delete (04h)
Instance:	Get Attribute (01h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“Diagnostic”
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	(depends on number of created diagnostic events)
4	Highest instance no.	Get	UINT16	(network specific)
11	Max no. of instances	Get	UINT16	Max. no. of instances that can be created (network specific) Of the maximum number of instances there should always be one instance reserved for an event of severity level “Major, unrecoverable”, to force the module into the state EXCEPTION.

Instance Attributes (Instance #1... N)

#	Name	Access	Type	Description
1	Severity	Get	UINT8	See below
2	Event Code	Get	UINT8	See below.
3	NW specific extension	Get	Array of UINT8	Network specific event information (optional)

Severity Levels

Bit Combination	Severity	Comment
00h	Minor, recoverable	-
10h	Minor, unrecoverable	Unrecoverable events cannot be deleted
20h	Major, recoverable	-
30h	Major, unrecoverable	Causes a state-shift to EXCEPTION
(other)	-	(reserved for future use)

Recoverable events shall be deleted by the application when the cause of the error is gone.

Unrecoverable events cannot be deleted. They remain active until the Anybus CompactCom is reset or power is turned off.

Event Codes

#	Meaning	Comment
10h	Generic Error	-
20h	Current	-
21h	Current, device input side	-
22h	Current, inside the device	-
23h	Current, device output side	-
30h	Voltage	-
31h	Mains Voltage	-
32h	Voltage inside the device	-
33h	Output Voltage	-
40h	Temperature	-
41h	Ambient Temperature	-
42h	Device Temperature	-
50h	Device Hardware	-
60h	Device Software	-
61h	Internal Software	-
62h	User Software	-
63h	Data Set	-
70h	Additional Modules	-
80h	Monitoring	-
81h	Communication	-
82h	Protocol Error	-
90h	External Error	-
F0h	Additional Functions	-
FFh	NW specific	Definition is network-specific; consult separate network guide for further information.

Command Details: Create

Details

Command Code:	03h
Valid For:	Object

Description

Creates a new instance, in this case representing a new diagnostic event in the host application.

- Command details:

Field	Contents
CMDExt[0]	Severity
CMDExt[1]	Event Code, see previous page
MsgData[0...n]	Network specific extension (optional, definition is network specific)

- Response details (Success):

Field	Contents
MsgData[0...1]	The number of the instance that was created as a result from the command

Command Details: Delete

Details

Command Code:	04h
Valid For:	Object

Description

Deletes an existing instance, i.e. a previously created diagnostic event.

Instances representing unrecoverable events cannot be deleted.

- Command details:

Field	Contents
CMDExt[0]	The number of the instance to delete (low byte)
CMDExt[1]	The number of the instance to delete (high byte)

- Response details (Error):

Error	Contents	Comment
Object Specific Error	MsgData[0] = FFh	-
	MsgData[1] = 01h	Error code (Not removed). The event could not be removed, either because the event itself is unrecoverable or due to a network specific reason.

See also:

– *Error Codes, p. 40*

8.5 Network Object (03h)

Category

Basic

Object Description

This object holds general information about the network (i.e. network type, data format etc.). It is also used when mapping ADIs as Process Data from the host application side.

See also...

- *Application Object (FFh), p. 75*
- *Application Data Object (FEh), p. 66*

Supported Commands

Object:	Get_Attribute (01h)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Get_Enum_String (06h)
	Map_ADI_Write_Area (10h)
	Map_ADI_Read_Area (11h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“Network”
2	Revision	Get	UINT8	02h
3	Number of instances	Get	UINT16	(Module type dependent)
4	Highest instance no.	Get	UINT16	(Module type dependent)

Instance Attributes (Instance #1)

#	Name	Access	Category	Type	Description
1	Network type	Get	Extended	UINT16	(See separate Network Guide and/or table below)
2	Network type string	Get	-	Array of CHAR	
3	Data format	Get	Basic	ENUM	Network data format: Value: Enumeration String: 00h: "LSB First" 01h: "MSB First"
4	Parameter data support	Get	Extended	BOOL	This attribute indicates if the network supports acyclic data services. It can also be used for deciding what ADIs to map to Process Data. Value: Meaning: True: Network supports acyclic data access False: No support for acyclic data
5	Write Process Data size	Get	-	UINT16	The current write Process Data size (in bytes). Updated on every successful Map_ADI_Write_Area or Remap_ADI_Write_Area.
6	Read Process Data size	Get	-	UINT16	The current read Process Data size (in bytes). Updated on every successful Map_ADI_Read_Area or Remap_ADI_Read_Area.
7	Exception Information	Get	-	UINT8	Additional network specific information may be presented here if the module has entered the EXCEPTION state (see separate network guide).

Network type	Network Type String
0099h	"BACnet MS/TP"
009Ah	"BACnet/IP"
0020h	"CANopen"
0090h	"CC-Link"
0065h	"ControlNet"
0025h	"DeviceNet"
0087h	"EtherCAT"
009Ch	"EtherNet/IP (2–port)"
009Bh	"EtherNet/IP (2-Port) BB DLR"
0045h	"Modbus RTU"
0093h	"Ethernet Modbus-TCP 2-Port"
0005h	"PROFIBUS DP-V1"
0096h	"PROFINET IO 2–Port"
0098h	"Sercos III"

Command Details: Map_ADI_Write_Area

Details

Command Code:	10h
Valid For:	Instance

Description

This command maps an ADI as Write Process Data. If successful, the response data contains the offset of the mapped ADI from the start of the Write Process Data image.

- It is strongly recommended *not* to map an ADI more than once (i.e. map it multiple times to the Read- or Write Process Data, or map it to both the Read- and Write Process Data) since this is not accepted by some networks.
- It is not possible to map only part of an ADI, i.e. all elements of an ADI must always be mapped.
- Certain Anybus CompactCom implementations allow the network to remap the Process Data during runtime. For more information, see Application Data Object (FEh).

See also...

- *Application Data Object (FEh), p. 66*
- *Application Object (FFh), p. 75*

! Error control is only performed on the command parameters. The Anybus module does not verify the correctness of these parameters by a read of the actual ADI attributes.

- Command details:

Field	Contents
CmdExt[0]	Instance number of the ADI (low byte)
CmdExt[1]	Instance number of the ADI (high byte)
MsgData[0]	Data Type of the ADI, see *Data Format, p. 36*
MsgData[1]	Number of elements in the ADI
MsgData[2]	Order Number of the ADI (low byte)
MsgData[3]	Order Number of the ADI (high byte)

The Order Number in the mapping command equals that of the command Get_Instance_Number_By_Order in the Application Data Object.

- Response details (Success):

Field	Contents
MsgData[0]	Offset of the mapped ADI from the start of the Write Process Data

- Response details (Error):

Error	Contents	
Invalid CmdExt[0]	The ADI number is not valid.	
Invalid State	Mapping of ADIs is only allowed in the SETUP state	
Object Specific Error	Object specific error, see MsgData[1] for details:	
	01h: Invalid data type	The data type is not valid for Process Data
	02h: Invalid number of elements	The number of elements is not valid (zero)
	03h: Invalid total size	The requested mapping is denied because the resulting total data size would exceed the maximum permissible (depending on network type)
	04h: Multiple mapping	The requested mapping was denied because the specific network does not accept multiple mapping of ADIs
	05h: Invalid Order Number	The order number is not valid (zero)
	06h: Invalid map command sequence	The order in which the commands were received is invalid

Error control is only performed on the command parameters. The Anybus module does not verify the correctness of these parameters by a read of the actual ADI attributes.

Command Details: Map_ADI_Read_Area

Details

Command Code:	11h
Valid For:	Instance

Description

This command is identical to Map_ADI_Write_Area, described above, except that it maps ADIs to Read Process Data.

8.6 Network Configuration Object (04h)

Category

Network specific

Object Description

This object contains network specific configuration parameters that may be set by the end user, typically settings such as baud rate, node address etc. Although the actual definition of the instances in this object are network specific, instance 1 and 2 are fixed in that they are always of an 8-bit data type.

When possible, the following convention is used for these instances:

Instance no.	Data type	Parameter
1	Any 8 bit data type	Currently selected network device address (or similar).
2	Any 8 bit data type	Currently selected network communication bit rate (or similar).

The instance values in this object must be updated whenever their originating value changes. Mechanical switches or similar need therefore be continuously monitored by the host application.

ⓘ *Instances tagged with 'shared' access (indicated by the descriptor) must be regarded as **volatile**; a set' access towards such an instance may or may not alter its value. The Anybus CompactCom will not respond with an error in case the value remains unaffected.*

Differentiation of Input Devices

The Anybus CompactCom makes a distinction between parameters originating from "hardwired" input devices (i.e. physical mechanical switches) and parameters specified using a "soft" input device such a keypad and display. This permits the Anybus module to fulfill network specific needs related to the actual origin of a parameter (e.g. some networks require that a change of value on physical switches is visually acknowledged on the on-board LEDs).

This distinction is based on the following actions from the host application (see table).

State	Actions (Host Application)	Anybus Behavior
SETUP	Poll and update parameters originating from physical switches (make sure to issue at least one Set command for each one of the affected parameters). Do not update parameters originating from "soft" input devices (do not issue any Set commands for these parameters yet).	The Anybus CompactCom identifies the affected parameters as originating from physical switches. The remainder are assumed to originate from "soft" input devices.
(other states)	Poll and update all parameters (i.e. physical switches and "soft" input methods) as necessary.	The Anybus CompactCom keeps track of the parameters which were updated during the SETUP state, and is thus able to treat them differently if required by the network.

Supported Commands

Object:	Get_Attribute (01h)
	Reset (05h) (The actual behavior is network specific)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Get_Enum_String (06h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“Network Configuration”
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	(Network dependent)
4	Highest instance no.	Get	UINT16	(Network dependent)

Instance Attributes (Instance #1... N)

Each instance represents a network configuration parameter. The attributes within it provides a comprehensive description of the parameter (name, data type etc.). Instance names and enumeration strings are multilingual .The actual strings are of course network specific, but the maximum number of characters is limited to thirteen (13).

#	Name	Access	Category	Type	Description
1	Name	Get	Application specific	Array of CHAR	Parameter name (e.g. “Node address”)
2	Data type	Get	Application specific	UINT8	Data type, see *Data Format, p. 36*
3	Number of elements	Get	Application specific	UINT8	Number of elements of the specified data type
4	Descriptor	Get	Application specific	UINT8	Bit field specifying the access rights for the parameter Bit: Access: b0: 1: Get Access b1: 1: Set Access b2: 1: Shared Access Instances tagged with shared access must be regarded as **volatile**; a Set-access towards such an instance may or may not alter its value. The Anybus CompactCom will not respond with an error in case the value remains unaffected.
5	Value	Determined by attribute #4	Application specific	Determined by attribute #2	Actual parameter value. Stored in nonvolatile memory Get access: the actually used value will be returned Set access: the configured (and possibly the actual) value will be written

Instance #1 and instance #2 are categorized as Basic, if they exist in an application. All other instances of this object are categorized in the respective network guides.

9 Host Application Objects

9.1 General Information

The objects in this group are meant to be implemented within the host application software. The Anybus module will issue commands towards these objects to access the settings and data within them. Their functionality is categorized to indicate when and how to use the objects.

See also ...

- *Data Format, p. 36*
- *Anybus Module Objects, p. 48*
- *Categorization of Functionality, p. 81*

For detailed information about each object, see...

- *Application Object (FFh), p. 75*
- *Application Data Object (FEh), p. 66*

9.2 Implementation Guidelines

Implementation of an object is generally a matter of parsing incoming commands and forming suitable responses. While the exact details as of how this is done is beyond the scope of this document, it is important to follow the following basic rules:

- An implemented object must feature all object attributes (instance #0) as specified in this document and/or the network interface appendix.
- In case a command for some reason cannot be executed (i.e. if a particular object, attribute or command hasn't been implemented), respond with a suitable error code to indicate the source of the problem.
- Support for the Application Data Object is mandatory.
- Support for the Application Object is optional, albeit highly recommended.
- Support for Network Specific Objects is optional, but recommended. It shall however be noted that the standard functionality provided by the Anybus CompactCom limits network functionality to the use of certain predefined device information and services. These limitations may be more or less significant and are described in each separate network interface appendix. In case this standard functionality is inadequate, i.e. vendor specific information or enhanced network functionality is required, Network Specific Objects may be implemented in the host application.
- During startup the module will attempt to retrieve values of attributes in the Network Specific Objects. If the module tries to access an object that is not implemented, respond with an error message (03h, Unsupported Object). If an attribute is not implemented in the host application, respond with an error message (06h, "Invalid CmdExt[0]"). The module will then use its default value. Also, if the module tries to retrieve a value of an attribute that is not listed in the network appendix, respond with an error message (06h, "Invalid CmdExt [0]".
- Support for Process Data remapping (by means of commands 'Remap_ADI_Write_Area' and 'Remap_ADI_Read_Area') is optional for the Anybus CompactCom 30 range and may provide better network integration for certain networks.

See also ...

- *Error Codes, p. 40*

> **!** The purpose of the Object Revision attribute is to make it possible for the Anybus CompactCom to establish whether or not the object implementation in the host application is compatible with that of the Anybus CompactCom, and to use different implementations if necessary. It is therefore imperative that the Object Revision attribute reflects the actual implementation, and that it is incremented based on changes in this document and/or the network guide only.

In case of questions, contact the HMS Industrial Networks technical support services at www.anybus.com/support.

9.3 Application Data Object (FEh)

Category

Basic. Please note that this object is mandatory.

Object Description

Each instance within this object (a.k.a. Application Data Instance or ADI) correlates to a block of data to be represented on the network. Each time such data is accessed from the network, the module translates such requests into object requests towards this object (or instances within it). The module may also access this object spontaneously if necessary. The exact representation on the network is highly network specific; e.g. on DeviceNet, ADIs are represented as dedicated CIP objects, while on PROFIBUS, ADIs are accessed by means of acyclic DP-V1 read and write services.

To allow the network and the Anybus CompactCom to efficiently scan the host application for ADIs, regardless of their instance number, this object implements the additional command Get_Instance_Number_By_Order. This command retrieves the ADI instance number as if the ADIs were sorted in a numbered list, allowing the Anybus CompactCom to query only for the instances that are actually implemented in the host application. The order number is also used when mapping ADIs to Process Data, see descriptions of the commands Map_ADI_ Write_Area in the *Network Object (03h), p. 58*.

In the example below, the host application has four ADIs with instance numbers 1,3, and 100.

Instance #	Implemented	Order Number
1	Yes	1
2	No	-
3	Yes	2
4... 99	No	-
100	Yes	3

In this particular case, the host application shall respond with instance number 100 to a Get_Instance_ Number_By_Order request for Order Number 3.

Please take the following into consideration when designing an application:

- The Anybus module does not take over the host application responsibility for error control of parameter requests, even if a request is clearly erroneous (e.g. a write request to an ADI with zero byte data, or an attempt to access an attribute that doesn’t exist, will not be filtered out by the module).
- The response time in the host application (i.e. the time spent processing an incoming request towards this object prior to responding to it) must be taken into consideration, since some networks may impose certain timing demands. Where applicable, special timing requirements etc. are specified in each separate network guide.
- Implementation of the following commands is mandatory when using products with profile support:
  - Get_Profile_Instance_Numbers (11h)
  - Remap_ADI_Write_Area (13h)
  - Remap_ADI_Read_Area (14h)
- Implementation the following commands is mandatory if remapping of Process Data is to be supported (object rev. 2 and higher).
  - Remap_ADI_Write_Area (13h)
  - Remap_ADI_Read_Area (14h)
  - Get_ADI_Info (12h)
- Implementation of the command Get_ADI_Info is mandatory since object rev. 2.

Supported Commands

Object:	Get_Attribute (01h)
	Get_Instance_Number_By_Order (10h)
	Get_Profile_Instance_Numbers (11h)
	Remap_ADI_Write_Area (13h)
	Remap_ADI_Read_Area (14h)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Get_Enum_String (06h)
	Get_Indexed_Attribute (07h)
	Set_Indexed_Attribute (08h)
	Get_ADI_Info (12h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“Application Data”
2	Revision	Get	UINT8	02h
3	Number of instances	Get	UINT16	(depends on application)
4	Highest instance no.	Get	UINT16	

Instance Attributes (Instance #1... n)

#	Name	Access	Type	Description
1	Name	Get	Array of CHAR	ADI name (can be multilingual)
2	Data type	Get	Array of UINT8	Data type of ADI value (see *Data Format, p. 36*)
3	Number of elements	Get	UINT8	Number of elements of the specified data type. It is strongly recommended not to use ADIs with Number of elements set to zero since this is not accepted by some networks.
4	Descriptor	Get	UINT8	Bit field specifying the access rights for the ADI value etc. b3 and b4 are mandatory if remapping of Process Data is supported. Bit: Access: b3 and b4 are mandatory if remapping of Process Data is supported. b0: 1: Get Access b1: 1: Set Access b2: -: (reserved, set to zero) b3: 1: Can be mapped as Write Process Data b4: 1: Can be mapped as Read Process Data
5	Value(s)	Determined by attribute #4	Determined by attribute #2	ADI value(s) Indexed elements can be of different types and sizes as specified in attribute #2. This attribute consists of all elements packed together with bit alignment. No implicit padding should be used. See table below for specific alignment restrictions and explicit padding.
6	Max. value	Get	Determined by attribute #2	The maximum permitted ADI value. Implementation of this attribute is optional. If not implemented, the module will use the maximum value of the specified data type for this attribute.
7	Min. value	Get	Determined by attribute #2	The minimum permitted ADI value. Implementation of this attribute is optional. If not implemented, the module will use the minimum value of the specified data type for this attribute.
8	Default value	Get	Determined by attribute #2	The default ADI value. Implementation of this attribute is optional. A zero value (float: +Min. value) will be used if not implemented.

- The byte order of attributes #5–8 is network dependent; the Anybus does not perform any byte swapping.
- The Max/Min/Default attributes is common for all elements in the ADI. That is, there is no separate Max/Min/Default value for each element in the array.
- Once defined, the number of elements is fixed and must, together with the data type, represent the buffer space needed to handle the array. It is not permitted change to the number of elements during runtime.
- The instance value(s) must fit entirely into the message data field. The number of elements, multiplied by the size of each element in bytes, must therefore never exceed 255 bytes.
- The only attributes that may be changed during runtime are attribute #1 and #5. Once defined, all other attributes must be considered fixed; changing them during runtime is not permitted.

Command Details: Get_Instance_Number_By_Order

Details

Command Code:	10h
Valid for	Object

Description

This command requests the actual instance number of an ADI as if sorted in an ordered list.

- Command details:

Field	Contents
CmdExt[0]	Requested Order Number (low byte)
CmdExt[1]	Requested Order Number (high byte)

- Response details (Success):

Field	Contents
MsgData[0...1]	The instance number of profile ADI

- Response details (Error):

Error	Contents
Invalid CmdExt[0]	The requested Order Number is not associated with an ADI.

Command Details: Get_Profile_Instance_Numbers

Details

Command Code:	11h
Valid for	Object

Description

This command is only applicable for products with built-in support for network profiles, such as the Anybus CompactCom 30 Drive Profile range of products; it is not used by the standard version of the Anybus CompactCom 30.

This command retrieves a list of the ADI instance numbers associated with a certain profile. The number of ADIs used for this, their purpose, data type etc., are dictated by the actual profile; the application may however assign instance numbers as needed to suit the implementation.

! When using the Anybus CompactCom 30 Drive Profile range of products, implementation of this command is mandatory. For further information, consult the Anybus CompactCom 30 Drive Profile Design Appendix. When using the standard version of the Anybus CompactCom 30, implementation of this command is not necessary.

- Command details:

Field	Contents	
CmdExt[0]	(reserved, ignore)	
CmdExt[1]	Value:	Profile:
	00h	(reserved)
	01h	Drive Profile
	(other)	(reserved for future use)

- Response details:

Field	Type	Meaning
MsgData[0...1]	UINT16	The instance number of profile ADI 1
MsgData[2...3]	UINT16	The instance number of profile ADI 2
...	...	...
MsgData[(2n-2)... (2n-1)]	UINT16	The instance number of profile ADI n

The contents of this list is entirely profile dependent and is specified in the separate profile design guide (i. e. the Anybus CompactCom Drive Profile Design Appendix).

Optional or conditional profile ADIs that are not supported by the host application shall be indicated with an instance number of zero.

The Anybus module enters the EXCEPTION state if any of the following occurs:

– the size of the response did not match the requested profile

– a required parameter was marked as not supported (see note 2)

– the application indicated an error in the response

Command Details: Get_ADI_Info

Details

Command Code:	12h
Valid for	Instance

Description

This command is used to gather the object attributes Data type, Number of elements and Descriptor of an ADI in a single response message.

ℹ *Implementation is mandatory since object rev. 2.*

- Command details:

Field	Contents
CmdExt[0]	(reserved, ignore)
CmdExt[1]	

- Response details (Success):

Field	Contents
MsgData[0]	Data type
MsgData[1]	Number of elements
MsgData[2]	Descriptor

- Response details (Error):

Error	Meaning
04h	Unsupported instance

Command Details: Remap_ADI_Write_Area

Details

Command Code:	13h
Valid for	Object

Description

The Anybus module issues this command when the network requests changes in the Process Data map. The ADIs are mapped at the insertion point in the same order as stated by the command. The command can remove and/or insert multiple mapping items, starting at the point indicated by the mapping item number in CmdExt[0], where a mapping item is an ADI previously mapped by a Map_ADI_Write_Area command, or an ADI (or elements of a multi-element ADI) previously mapped by a Remap_ADI_Write_Area command.

The following set of data is included in the command data for each inserted mapping item:

- The ADI number
- The index to the first element to map
- The number of consecutive elements to map

The command may be issued in the following Anybus CompactCom states: NW_INIT, WAIT_PROCESS, IDLE and ERROR.

All actions specified in the command shall either be carried out or rejected, i.e. the Process Data map must remain unchanged if the command was not accepted.

The Anybus module is limited to one outstanding remap command at a time.

See also...

- *Network Object (03h), p. 58*
- *Runtime Remapping of Process Data, p. 96*

! To support this procedure, the host application must be capable of remapping the Process Data during runtime. This is a mandatory requirement for object rev. 2, and optional for object rev. 3. Support for this command is highly recommended and is required when using the Anybus CompactCom Drive Profile range of products. It may also provide better network integration for certain networks in the standard product range.

- Command details:

Field	Contents
CmdExt[0]	Start of remap (low byte) (mapping item number, 0 = first)
CmdExt[1]	Start of remap (high byte) (mapping item number, 0 = first)
Data[0-1]	The number of current mapping items to remove
Data[2-3]	The number of mapping items to insert (0... 62)
Data[4-5]	New mapping item 1: ADI number
Data[6]	New mapping item 1: Index to the first element to map
Data[7]	New mapping item 1: Number of consecutive elements to map
Data[8-9]	New mapping item 2: ADI number
Data[10]	New mapping item 2: Index to the first element to map
Data[11]	New mapping item 2: Number of consecutive elements to map
...	(etc.)

- Response details (Success):

Field	Contents
MsgData[0]	The resulting total size of the write process data area in bytes (low byte)
MsgData[1]	The resulting total size of the write process data area in bytes (high byte)

- Response details (Error):

Error Code	Error	Meaning
01h	Mapping item error	The requested mapping is denied because of a NAK to at least one mapping item
02h	Invalid total size	The requested mapping is denied because the resulting total data size would exceed the maximum permissible for the application

Command Details: Remap_ADI_Read_Area

Details

Command Code:	14h
Valid for	Object

Description

This command is used to (re-)map ADIs to the read process data area. It is otherwise equivalent to Remap_ ADI_Write_Area.

A successful transfer of an ACK to a remap command indicates the point where the process data map will be changed. For serial applications, this means that a changed process data map shall be expected or used in telegrams following the empty telegram (or telegrams in case of retransmissions) after the ACK (see runtimeremapping).

Handling of changed data indication (Auxiliary Bit) during a remap is described in *Changed Data Indication, p. 22*.

- *Network Object (03h), p. 58*
- *Runtime Remapping of Process Data, p. 96*

! To support this procedure, the host application must be capable of remapping the Process Data during runtime. This is a mandatory requirement in the case of the Anybus CompactCom Drive Profile range of products, but may also provide better network integration for certain networks in the standard product range. For these products. support for this command is optional, but highly recommended.

9.4 Application Object (FFh)

Category

Extended

Object Description

This object groups general settings for the host application. It is not mandatory, but it is highly recommended to implement this object and its commands to be able to support multiple languages and network reset requests.

Supported Commands

Object:	Get_Attribute (01h)
	Reset (05h)
	Reset_Request (10h)
	Change_Language_Request (11h)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Get_Enum_String (06h)

Object Attributes (Instance #0)

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

Instance Attributes (Instance #1)

#	Name	Access	Type	Description
1	Configured	Get	BOOL	Indicates if the application parameters have been changed from their out-of-box value. Value: Meaning: False: Out-of-box state. True: Configured, settings have been altered. See details for commands “Reset” and “Reset_Request” below
2	Supported languages	Get	Array of ENUM	List specifying which languages that are supported by the host application. Value: Meaning: 00h: “English”. 01h: “Deutsch”. 02h: “Español”. 03h: “Italiano”. 04h: “Français”. See also ... • *Anybus Object (01h), p. 49*, instance #1, attribute #9 • Details for command Change_Language_Request below.

Command Details: Reset

Details

Command Code:	05h
Valid for:	Object

Description

This command is issued by the module when a reset is required. Depending on the network type, it may, or may not, be preceded by a “Reset_Request” command.

- Command details:

Field	Contents	Comment
CmdExt[0]	(reserved, ignore)	-
CmdExt[1]	00h: Power-on reset	This shall be regarded as a device reset, i.e. the host application shall reset the module via the /RESET signal. The Anybus module enters the state EXCEPTION prior to issuing this type of request.
	01h: Factory default reset	This shall cause the host application to return to an application specific out-of-box state. Any network-specific procedures necessary to set the module to this state are performed automatically. The state of the Anybus module, prior to this request, is network specific.
	02h: Power-on + Factory default	A combination of the two above. The Anybus module enters the state EXCEPTION prior to issuing this type of request.

- Response details:

  (No data)

Command Details: Reset_Request

Details

Command Code:	10h
Valid for:	Object

Description

On certain networks, this command may be issued prior to the Reset command (see below). This is, as the name implies, a request and *not* an actual reset command.

The requested reset can be either a Power-on reset, a Factory Default reset, or both. A Power-on reset shall be regarded as a device reset.

If the request is granted, the host application must also be prepared to receive a corresponding Reset command (see figure).

The host application is also free to respond with an error in case a reset for some reason cannot be executed. In such case, no Reset command will be issued by the module.

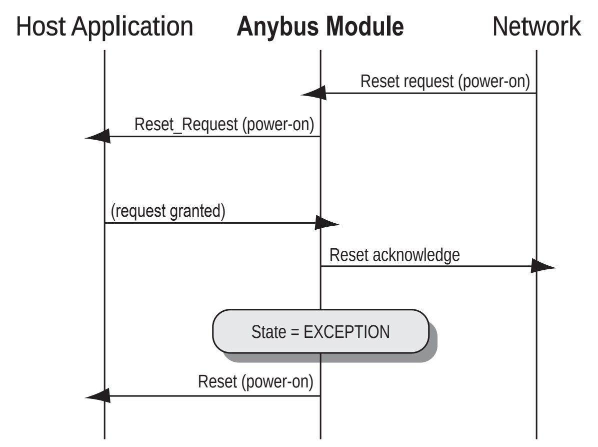


Fig. 20

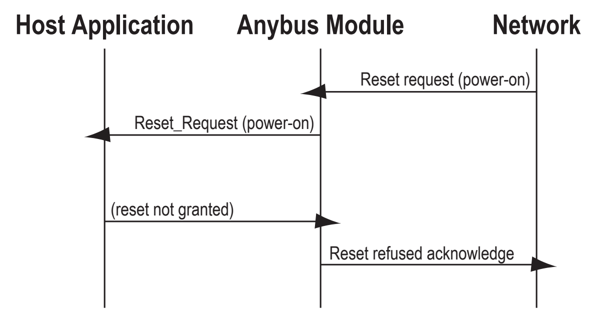


Fig. 21

This command is issued by the module when a reset is required. Depending on the network type, it may, or may not, be preceded by a “Reset_Request” command.

- Command details:

Field	Contents
CmdExt[0]	(reserved, ignore)
CmdExt[1]	00h: Power-on reset
	01h: Factory default reset
	02h: Power-on + Factory default

- Response details:

  (No data)

Command Details: Change_Language_Request

Details

Command Code:	11h
Valid for:	Object

Description

This command will be issued by the module when a change of the current language is requested from the network.

If accepted, it will result in a corresponding change of the Language Attribute (#9) in the Anybus Object (01h). The host application must also adjust its internal language settings accordingly.

- Command details:

Field	Contents	
CmdExt[0]	Reserved. Value = 00h	
CmdExt[1]	The requested language	
	Value:	Language:
	00h:	English.
	01h:	German
	02h:	Spanish.
	03h:	Italian.
	04h:	French.

- Response details:

  (No data)

9.5 Host Application Specific Object (80h)

Category

Extended

Object Description

The functionality of this object is not specified. The application is free to specify the functionality. E.g. the object can be used to access data in the application using the SSI interface on Ethernet capable modules.

A Categorization of Functionality

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

Some of the functionality offered may be specialized and/or seldom used. As most of the available network functionality is enabled and accessible, access to the specification of the industrial network may be required.

B Network Comparison

The Anybus CompactCom 30 software interface is designed to be as generic as possible without sacrificing network functionality or integration with the host system.

When designing the host application, it is important to be aware of the limitations and possibilities of each networking system. In most cases, no additional software support is needed to support a particular network. However, in order to fully exploit certain aspects of the network functionality, a degree of dedicated software support may be necessary.

A summary of the features offered by the different network implementations is presented in the table on the next page.

How to interpret the table is described below:

- The figures specify the values that are to be expected in a typical generic implementation.
- The figures in parenthesis specify the values that are possible with dedicated software support.
- Of the maximum number of diagnostic instances there is always one instance reserved for one of severity level “Major, unrecoverable” to force the module into the state EXCEPTION.
- If a data type is not supported, this means that the network has no direct counterpart for that particular type. The data may however still be represented on the network, albeit in some other format (e.g. a UINT64 may be represented as four UINT16s etc.)
- Network specific comments to the table are listed after the table.

! The information in this chapter gives a rough idea of the possibilities on the different network implementations. For in-depth information about a particular network, consult the corresponding network guide.

Item	Ethernet/IP	CC-Link	PROFIBUS DP-V1	PROFIBUS DP-V0	PROFINET	DeviceNet	CANopen	Modbus RTU	Modbus/ TCP	ControlNet	EtherCAT	SERCOS III	BACnet IP	BACnet MSTP
Network Data Format	LSB first	LSB first	MSB first	MSB first	MSB first	LSB first	LSB first	LSB first	LSB first	LSB first	LSB first	LSB first	MSB first	MSB first
Acyclic Data Support	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Max. no. of Elements Per ADI	255	255	64 (240)	N/A	255	255	254	32 (255)	32 (255)	255	255	255	1	1
Max. ADI Size (in bytes)	255	255	64 (240)	N/A	255	255	254	32 (255)	32 (255)	255	255	255	4	4
Lowest Addressable ADI no.	1	N/A	1	N/A	1	1	1	1	1	1	1	1	1	1
Highest Addressable ADI no.	65535	N/A	65025	N/A	32767	65535	16383	4062 (65023)	4062 (65023)	65535	16383	32767	65535	65535
Max. Write Process Data (in bytes)	256	14-44 (256)	152 (244)	80 (244)	256	256	256	256	256	256	256	256	256	256
Min. Write Process Data (in bytes)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max. Read Process Data (in bytes)	256	14-46 (256)	152 (244)	80 (244)	256	256	256	256	256	256	256	256	0	0
Min. Read Process Data (in bytes)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Max. Process Data (Read + Write, in bytes)	512	28-90 (512)	152 (368)	80 (380)	512	512	512	512	512	512	512	512	256	256
Min. Process Data (Read + Write, in bytes)	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Requires “Get/ Set_Indexed_ Attribute”	No	No	No	No	No	No	Yes	No	No (Yes)	No	Yes	No	No	No
Requires “Get_ Instance_ Number_By_ Order”	Yes	No	No	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Max. no. of Diagnostic Instances	6	6	6 (1)	10 (10)	6 (1)	6	6	6	6	6	6	6	1	1
Supports Network Reset Type 0: “Power-on-reset”	Yes	No	No	No	No	Yes	Yes	No	No	Yes	No	No	Yes	Yes
Supports Network Reset Type 1: “Factory default reset”	Yes	No	No	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No
Supports SINT64	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Supports UINT64	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Supports FLOAT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Extended LED functionality	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No	No

- EtherNet/IP (EIP):
  - The command “Get_Instance_Number_By_Order” is used weh accessing attributes in the Parameter Ojbect from a CIP network.
  - For modules, that support internal web pages, the command “Get_Instance_Number_By_Order” is used when the parameter web page is opened
- CC-Link (CCL):
  - The amount of process data depends on the data types of the mapped ADIs. For more information see CC-Link network interface appendix.
- PROFIBUS (DPV1):
  - Due to technical reasons, it is generally not recommended to use ADI numbers 1 ... 256, since this may cause problems when using certain PROFIBUS configuration tools.
  - Considering size of process data, check PROFIBUS DPV1 network interface appendix for addressing limitations.
- PROFIBUS (DPV0):
  - Considering size of process data, check PROFIBUS DPV0 network interface appendix for addressing limitations.
- PROFINET:
  - If API 0 is not used, or if transparent mode is used, the ADI data size will at the most be 244 bytes as the header (11 bytes) must be inserted in the Get/Set_Record message. For more information see PROFINET network appendix.
  - Considering size of process data, check PROFINET network interface appendix for addressing limitations.
  - For modules, that support internal web pages, the command “Get_Instance_Number_By_Order” is used when the parameter web page is opened
- DeviceNet (DEV):
  - The command “Get_Instance_Number_By_Order” is used weh accessing attributes in the Parameter Ojbect from a CIP network.
- Modbus-TCP (EIT):
  - The command “Get_Instance_Number_By_Order” is used weh accessing attributes in the Parameter Ojbect from a CIP network.
- ControlNet:
  - For modules, that support internal web pages, the command “Get_Instance_Number_By_Order” is used when the parameter web page is opened
- EtherCAT (ECT):
  - For EtherCAT products, the command “Get_Instance_Number_By_Order” is used during initialization to find number of ADIs
- SERCOS III:
  - Of addressable ADIs, around 16000 ADIs can be reached from SERCOS III
  - For modules, that support internal web pages, the command “Get_Instance_Number_By_Order” is used when the parameter web page is opened

- BACnet/IP:
  - For modules, that support internal web pages, the command “Get_Instance_Number_ By_Order” is used when the parameter web page is opened
  - When a BACnet product isn’t in advanced mode, the command “Get_Instance_ Number_By_Order” is used to perform initial mapping.
- BACnet MSTP:
  - When a BACnet product isn’t in advanced mode, the command “Get_Instance_ Number_By_Order” is used to perform initial mapping.

C Object Overview

Each device in the Anybus CompactCom 30 series supports a subset of the objects, described in this design guide and in the respective network guides. The following tables give an overview.

If the firmware of a module has been upgraded recently, these tables may be subject to update in the next revision of this document.

C.1 Anybus Module Objects

These objects are implemented in the product.

		Ethernet/IP	CC-Link	PROFIBUS DP-V1	PROFIBUS DP-V0	PROFINET	DeviceNet	CANopen	Modbus RTU	Modbus/ TCP	ControlNet	EtherCAT	SERCOS III	BACnet IP	BACnet MSTP
01h	Anybus Object	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
02h	Diagnostic Object	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
03h	Network Object	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
04h	Network Configuration Object	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
05h	Additional Diagnostic Object	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
07h	Socket Interface Object	Yes	No	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	No
08h	Network CC-Link Object	No	Yes	No	No	No	No	No	No	No	No	No	No	No	No
09h	SMTP Client Object	Yes	No	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	No
0Ah	Anybus File System Interface Object	Yes	No	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	No
0Bh	Network PROFIBUS DP-V1 Object	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
0Ch	Network Ethernet Object	Yes	No	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	No
0Dh	CIP Port Configuration Object	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No
0Eh	Network PROFINET IO Object	No	No	Yes	No	Yes	No	No	No	No	No	No	No	No	No
0Fh	PROFINET Additional Diagnostic Object	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
10h	PROFIBUS DP-V0 Diagnostic Object	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
11h	Functional Safety Module Object	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No

C.2 Host Application Objects

These objects are possible to implement in the host application. Depending on the application, not all objects available for a network, may be necessary.

		Ethernet/IP	CC-Link	PROFIBUS DP-V1	PROFIBUS DP-V0	PROFINET	DeviceNet	CANopen	Modbus RTU	Modbus/ TCP	ControlNet	EtherCAT	SERCOS III	BACnet IP	BACnet MSTP
E7h	Energy Reporting Object	No	No	No	No	No	No	No	No	No	No	No	No	No	No
E8h	Functional Safety Object	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
EAh	Application File System Interface Object	No	No	No	No	No	No	No	No	No	No	No	No	No	No
EDh	CIP Identity Host Object (FDh)	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No
EFh	BACnet Host Object	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes
F0h	Energy Control Object	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
F1h	SERCOS III Object	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No
F3h	ControlNet Host Object	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No
F5h	EtherCAT Object	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No
F6h	PROFINET IO Object	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
F7h	CC-Link Host Object	No	Yes	No	No	No	No	No	No	No	No	No	No	No	No
F8h	EtherNet/IP Host Object	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No
F9h	Ethernet Host Object	Yes	No	Yes	No	Yes	No	No	No	Yes	No	No	Yes	Yes	No
FAh	Modbus Host Object	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	No
FBh	CANopen Object	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No
FCh	DeviceNet Host Object	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No
FDh	PROFIBUS DP-V1 Object	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
FEh	Application Data Object	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
FFh	Application Object	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

D Timing & Performance

D.1 General Information

This chapter specifies timing and performance parameters that are verified and documented for each member of the Anybus CompactCom 30 family.

The following timing aspects, further described below, are measured:

Category	**Parameters**
Startup Delay	T1, T2
NW_INIT Delay	T3
Telgram 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
Network System Write Process Data Delay (Network System Delay)	T15, T16, T17

! At the time of writing, network specific timing specifications for all networks may not yet been publicly released. This information will be added continuously to all network guides when available. In case of questions, contact HMS Industrial Networks.

D.2 Internal Timing

D.2.1 Startup Delay

The following parameters are defined as the time measured from the point where /RESET is released to the point where the specified event occurs.

Parameter	Description	Max.	Unit.
T1	Anybus generates the first application interrupt (parallel mode)	1.0	s
T2	The Anybus is able to receive and handle the first application telegram (serial mode)	1.0	s

D.2.2 NW_INIT Delay

The time required by the Anybus module to perform the necessary actions in the NW_INIT-state is highly network specific. Furthermore, the number of commands issued towards the host application in this state may vary, not only between different networks, but also between different implementations (e.g. depending on the actual Process Data implementation etc.). This, in turn, means that the response time of the host application has a major impact on this parameter as well. It is therefore only possible to specify a maximum value that any Anybus version, together with a typical host application implementation, can fulfill.

Specifying this parameter does not, in any way, imply that the host application is required, or even expected, to supervise that it is met - the fact that the protocol is running and the correct state is indicated should be a sufficient indication of the healthiness of the Anybus module. If, however, the Anybus concept is not trusted in this respect, the host application may wait for a timeout before a no-go situation is indicated to the end user. It should then be satisfactory to use a rather long timeout value since this is, after all, during the start-up phase.

Parameter	Conditions
No. of network specific commands	Max.
No. of ADIs (single UINT8) mapped to Process Data in each direction	32 or maximum amount in case the network specific maximum is less.
Application response time	< 10 ms
No. of simultaneously outstanding Anybus commands that the application can handle	1

Parameter	Description	Communication	Max.	Unit.
T3	NW_INIT delay	Serial 19.2 kbps	30	s
		(all other modes)	10	s

D.3 Anybus Response Time

D.3.1 Overview

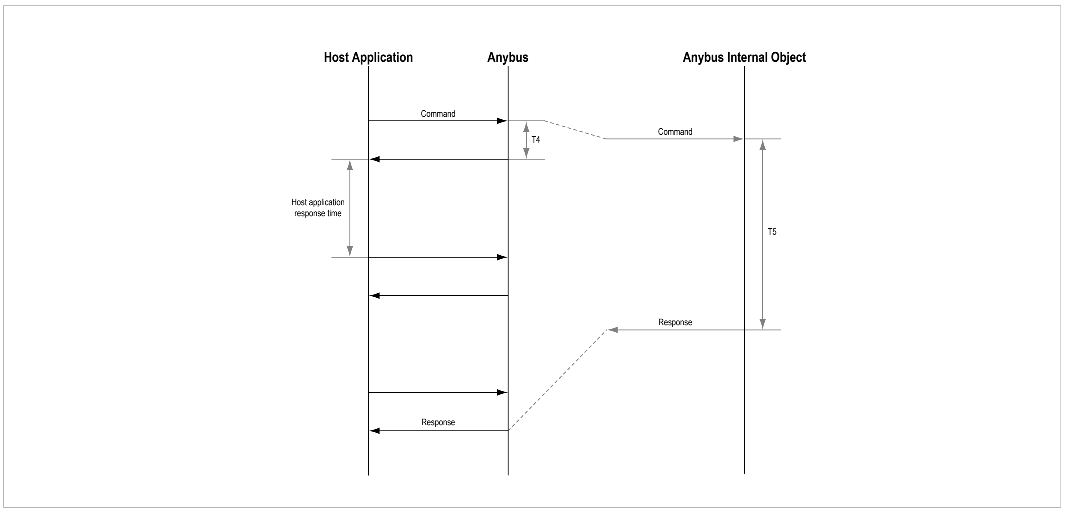


Fig. 22

D.3.2 Telegram delay

The Telegram Delay is defined as the time required by the Anybus CompactCom 30 to respond to a telegram.

It is assumed that commands are issued one by one, i.e. that no new commands are issued towards the Anybus CompactCom 30 prior to receiving a response to the previous one.

Parameter	Conditions
Communication	All modes
Host application response time	≥ 0.2 ms
Anybus state	All states
No. of ADIs (single UINT8) mapped to Process Data in each direction	32 or maximum amount in case the network specific maximum is less
Bus load, no. of nodes, baud rate etc.	Normal
No. of simultaneously outstanding Anybus commands	1

Parameter	Description	Avg.	Max.	Unit.
T4	Anybus telegram delay	< 0.4	1.5	ms

D.3.3 Command Delay

The Command Delay is defined as the time required by the Anybus CompactCom 30 to respond to commands which are handled internally, i.e. commands where no network information is exchanged. The measurement is ended when the Anybus CompactCom 30 has finished processing and is ready to send a response.

It is assumed that commands are issued one by one, i.e. that no new commands are issued towards the Anybus CompactCom 30 prior to receiving a response to the previous one.

Parameter	**Conditions**
Communication	All modes
Host application response time	≥ 0.2 ms
Anybus state	All states
No. of ADIs (single UINT8) mapped to Process Data in each direction	32 or maximum amount in case the network specific maximum is less
Bus load, no. of nodes, baud rate etc.	Normal
No. of simultaneously outstanding Anybus commands	1

Certain commands may require a considerable amount of time to execute due to various technical reasons (e.g. storage of parameters to non-volatile memory or formatting of a file system etc.). The commands are therefore categorized by their expected command delay.

Parameter	**Description**	**Category**	**Avg.**	**Max.**	**Unit.**
T5	Anybus Command delay	A	<1	1.5	ms
		B	-	5000	ms
		C	-	∞	ms

A command is of category A unless otherwise stated.

Commands of category C are used for blocking services; a response is not returned until an external event occurs of which the Anybus CompactCom 30 has no control. It must, however, be taken into consideration that such services will lock message resources for an unpredictable amount of time in both the Anybus CompactCom 30 and the host application.

D.4 Process Data

D.4.1 Overview

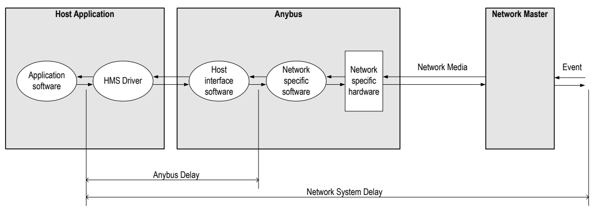


Fig. 23

D.4.2 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).

The transmission delay for the serial communication is not considered in these measurements.

Parameter	Conditions
Application CPU	-
Time system call interval	1 ms
Driver call interval	0.2... 0.3 ms All states
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

Parameter	Description	Avg.	Max.	Unit.
T6	Anybus Read Process Data delay, 8 ADIs (single UINT8)	< 0.5	1	ms
T7	Anybus Read Process Data delay, 16 ADIs (single UINT8)	< 0.7	1.2	ms
T8	Anybus Read Process Data delay, 32 ADIs (single UINT8)	< 1	1.5	ms

D.4.3 Anybus Write Process Data Delay (Anybus Delay)

The Write Process Data Delay (labelled “Anybus delay” in the figure above) 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.

The transmission delay for the serial communication is not considered in these measurements.

Parameter	**Conditions**
Application CPU	-
Time system call interval	1 ms
Driver call interval	0.2... 0.3 ms All states
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

Parameter	**Description**	**Avg.**	**Max.**	**Unit.**
T12	Anybus Write Process Data delay, 8 ADIs (single UINT8)	< 0.5	1	ms
T13	Anybus Write Process Data delay, 16 ADIs (single UINT8)	< 0.7	1.2	ms
T14	Anybus Write Process Data delay, 32 ADIs (single UINT8)	< 1	1.5	ms

D.4.4 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	Conditions
Application CPU	-
Time system call interval	1 ms
Driver call interval	0.2... 0.3 ms All states
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

Parameter	Description	Avg.	Max.	Unit.
T15	Network System Read Process Data delay, 8 ADIs (single UINT8)	(network type dependent, see separate network interface appendix)		
T16	Network System Read Process Data delay16 ADIs (single UINT8)			
T17	Network System Read Process Data delay32 ADIs (single UINT8)			

! At the time of writing, network specific timing specifications for all networks may not yet been publicly released. This information will be added continuously to all network guides when available. In case of questions, contact HMS Industrial Networks.

D.4.5 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	Conditions
Application CPU	-
Time system call interval	1 ms
Driver call interval	0.2... 0.3 ms All states
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

Parameter	Description	Avg.	Max.	Unit.
T15	Network System Write Process Data delay, 8 ADIs (single UINT8)	(network type dependent, see separate network interface appendix)		
T16	Network System Write Process Data delay16 ADIs (single UINT8)			
T17	Network System Read Process Write delay32 ADIs (single UINT8)			

At the time of writing, network specific timing specifications for all networks may not yet been publicly released. This information will be added continuously to all network guides when available. In case of questions, contact HMS Industrial Networks.

E Runtime Remapping of Process Data

This appendix describes how to handle a request from the application to remap read or write process data in parallel mode and in serial mode. Please note that the telegrams are exchanged in a ping-pong fashion.

E.1 Parallel mode

Runtime remapping of process data in parallel mode is rather straightforward, see figures below.

E.1.1 Read Process Data

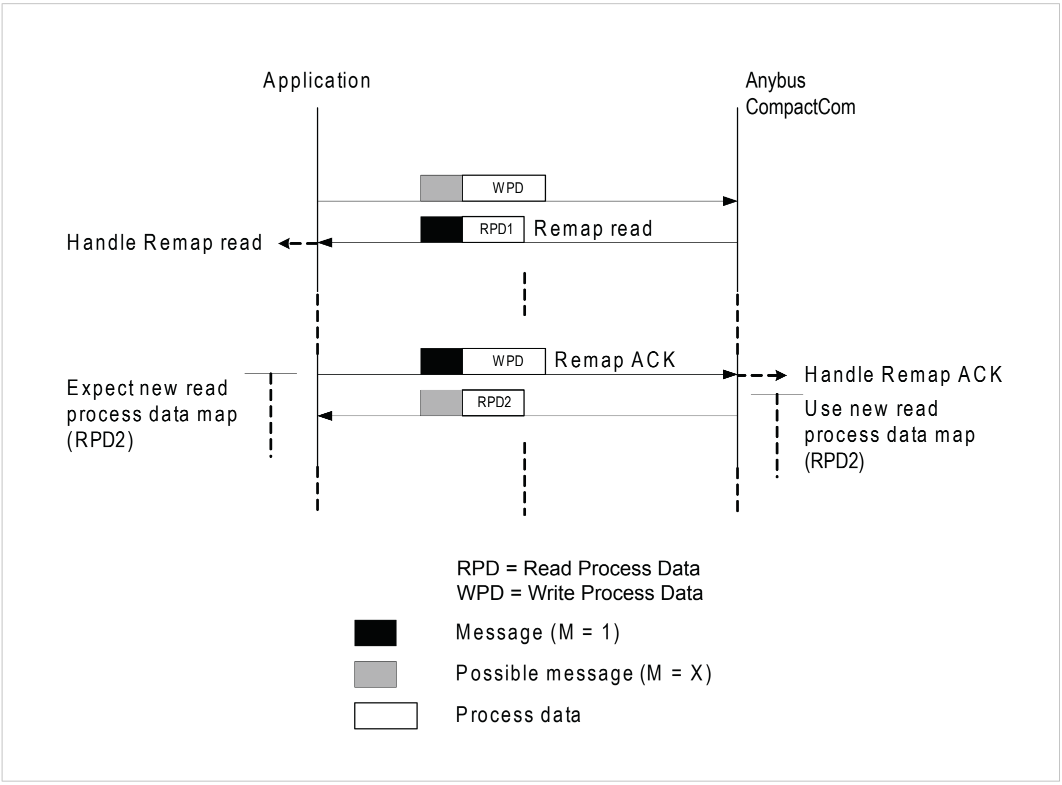


Fig. 24

E.1.2 Write Process Data

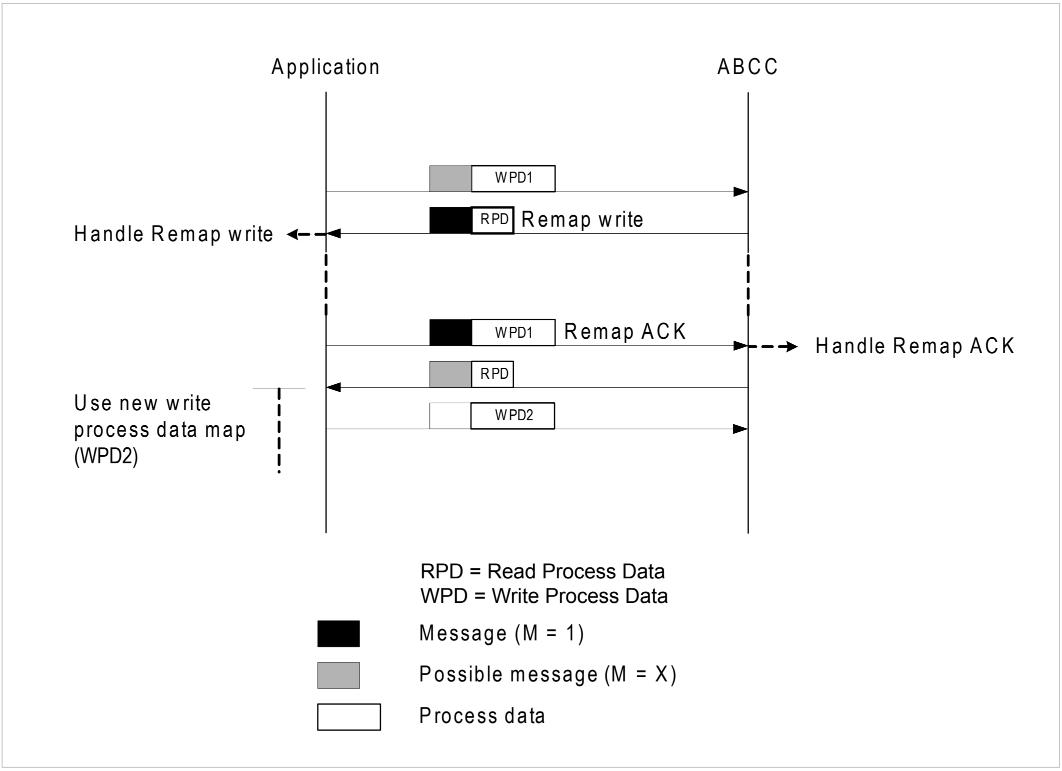


Fig. 25

E.2 Serial Mode

Please note that the telegrams are exchanged in a ping-pong fashion, and that a telegram without a message ends each command. A number of telegrams will thus have to be exchanged before the re-mapping takes effect

E.2.1 Read Process Data

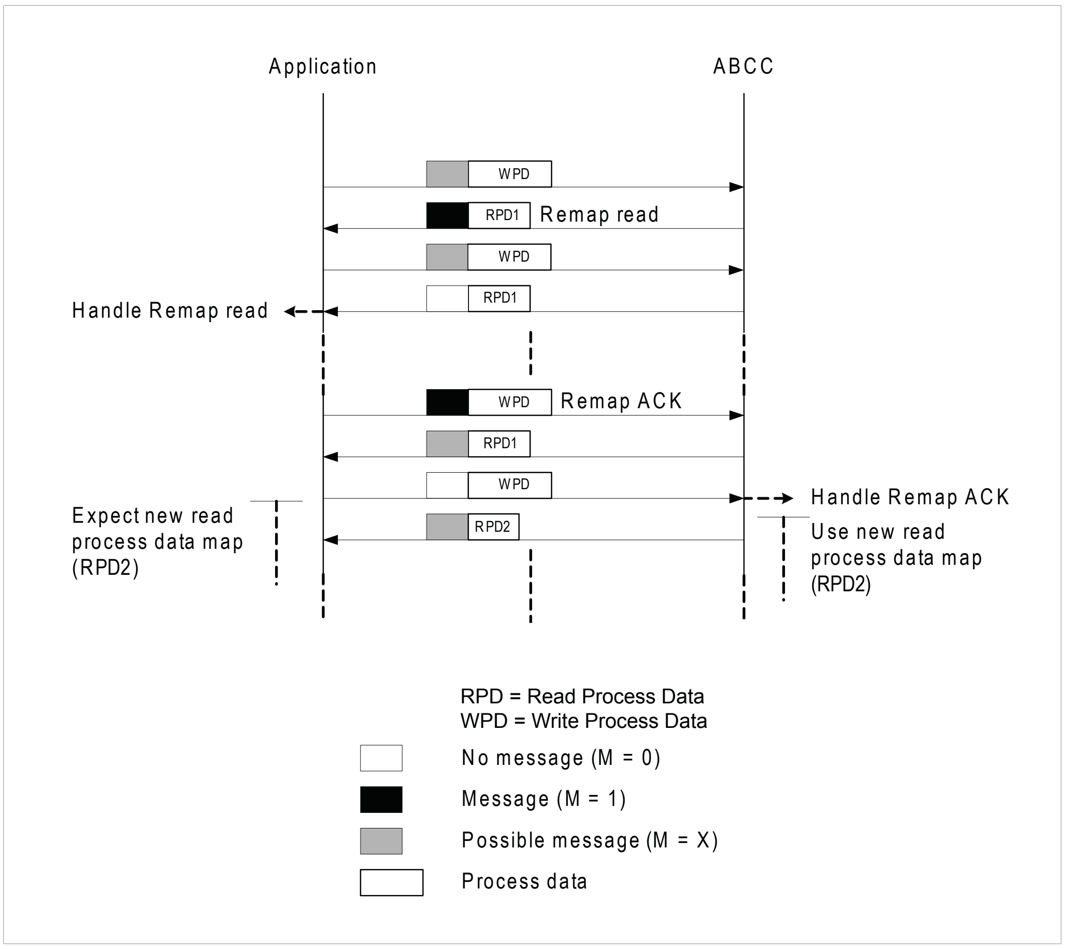


Fig. 26

E.2.2 Write Process Data

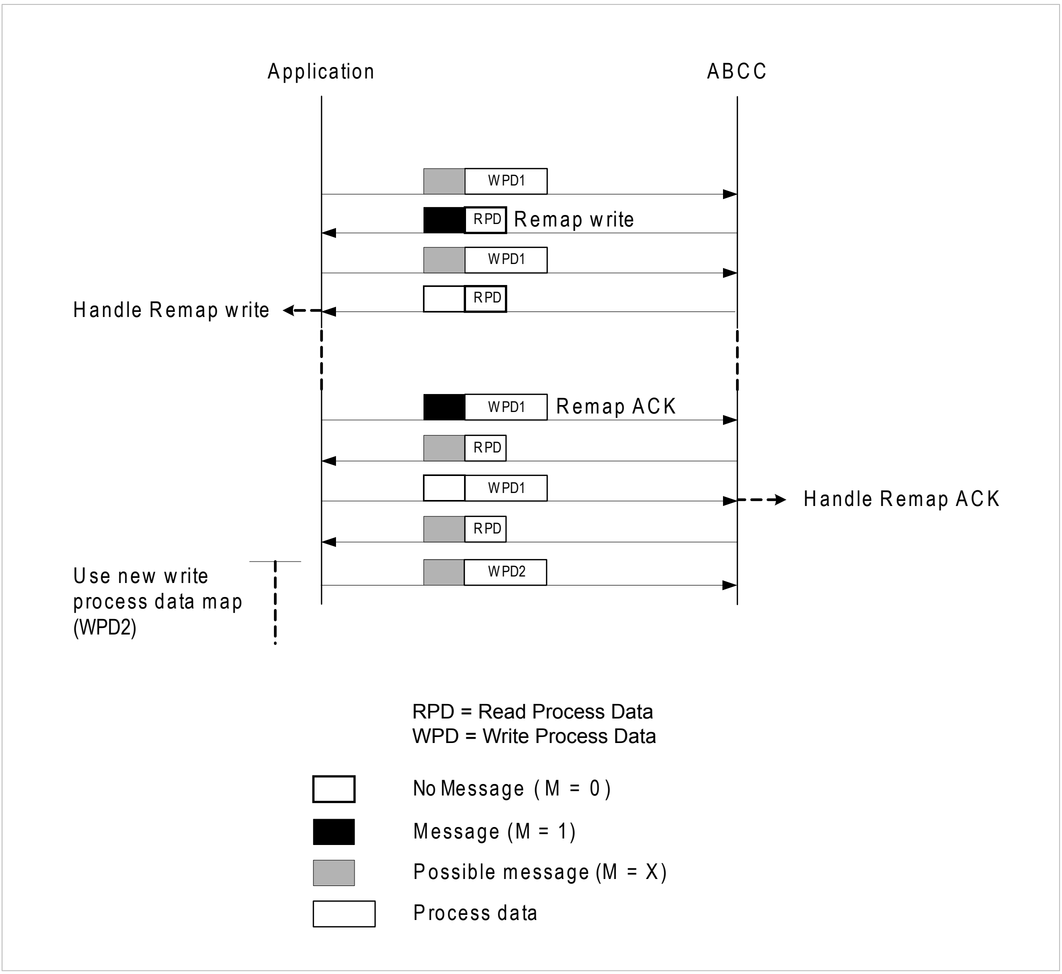
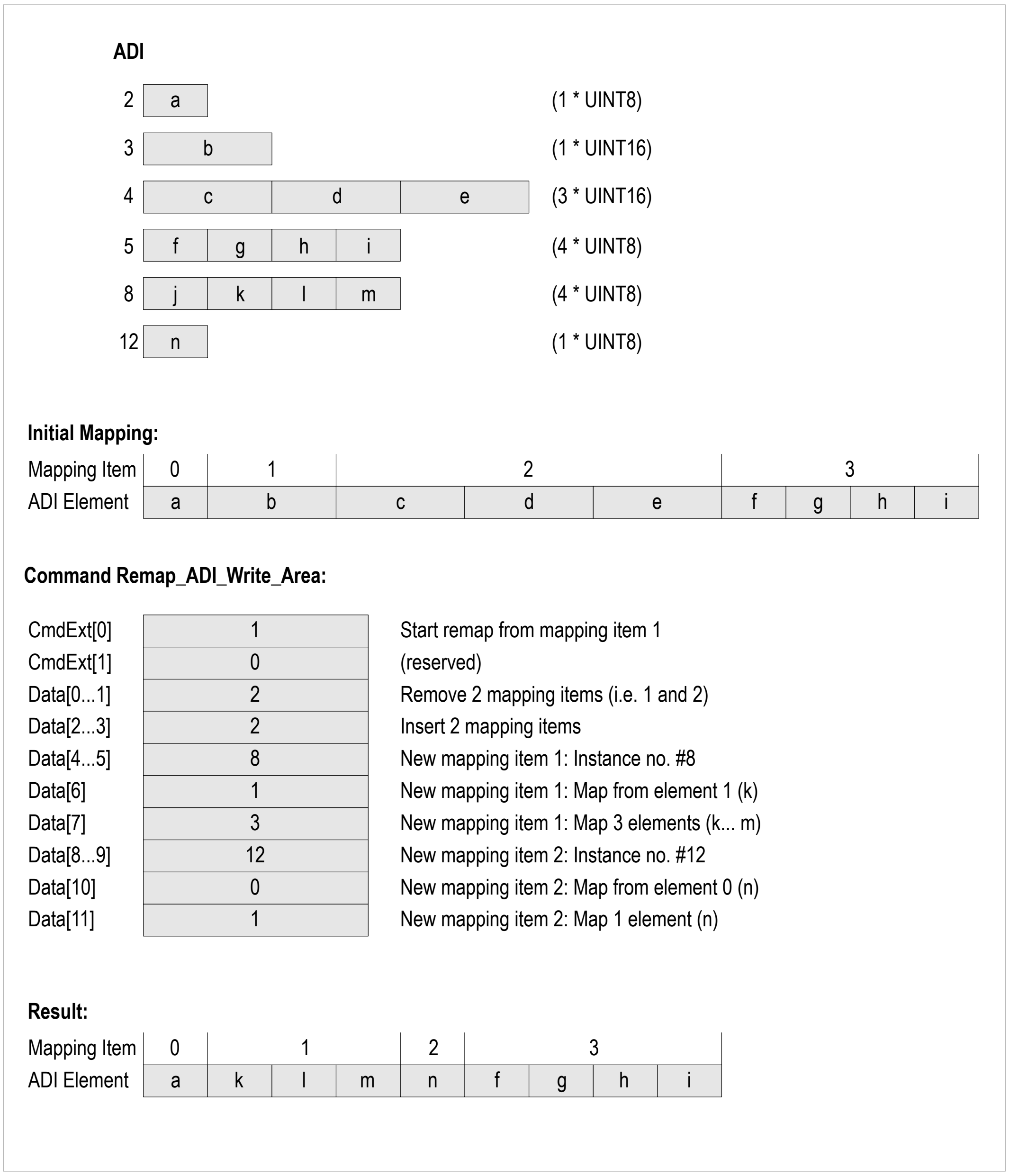


Fig. 27

E.3 Example: Remap_ADI_Write_Area



Initial Mapping:

Mapping Item	0	1	2			3			
ADI Element	a	b	c	d	e	f	g	h	i

Command Remap_ADI_Write_Area:

CmdExt[0]	1	Start remap from mapping item 1
CmdExt[1]	0	(reserved)
Data[0...1]	2	Remove 2 mapping items (i.e. 1 and 2)
Data[2...3]	2	Insert 2 mapping items
Data[4...5]	8	New mapping item 1: Instance no. #8
Data[6]	1	New mapping item 1: Map from element 1 (k)
Data[7]	3	New mapping item 1: Map 3 elements (k... m)
Data[8...9]	12	New mapping item 2: Instance no. #12
Data[10]	0	New mapping item 2: Map from element 0 (n)
Data[11]	1	New mapping item 2: Map 1 element (n)

Result:

Mapping Item	0	1			2	3			
ADI Element	a	k	l	m	n	f	g	h	i

Fig. 28

F CRC Calculation (16–bit)

F.1 General

The following information applies only when using the serial interface.

To allow the receiving part to detect transmission errors, each serial telegram frame contains a 16-bit Cyclic Redundancy Check.

The CRC is calculated as follows:

1. Load a 16-bit register with FFFFh. (Let's call it the CRC-register for simplicity)
2. XOR the first byte of the message with the low order byte of the CRC-register, putting the result in the CRC-register.
3. Shift the CRC-register one bit to the right (towards the LSB), zero-filling the MSB.
4. Examine the LSB that was just shifted out from the register. If set, Exclusive-OR the CRC-register with the polynomial value A001h (1010 0000 0000 0001).
5. Repeat steps 3 and 4 until 8 shifts have been performed.
6. XOR the next byte from the message with the low order byte of the CRC-register, putting the result in the CRC-register
7. Repeat steps 3...6 until the complete message has been processed.
8. The CRC-register now contains the final CRC16-value.

F.2 Example

When implementing the CRC calculation algorithm, use these example strings (below) to ensure that the algorithm yields the same results as the Anybus CompactCom module.

The array `{ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 }` should yield the following CRC16: `{ 0xb0, 0xcf }`.

The array `{ 0x00, 0x55, 0xAA, 0xFF, 0x0F, 0x5A, 0xA5, 0xF0 }` should yield the following CRC16: `{ 0x11 , 0x03 }`.

The array `{ 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 }` should yield the following CRC16: `{ 0x77 , 0x28 }`.

F.3 Code Example

This example uses a fast approach to calculate the CRC; all possible CRC-values are preloaded into two arrays, which are simply indexed as the function increments through the message buffer.

```
const UINT8 abCrc16Hi[] =
{
   0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80,
   0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
   0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
   0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
   0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
   0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
   0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
   0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
   0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
   0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
   0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00,
   0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
   0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
   0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
   0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
   0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
   0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
   0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
   0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,
   0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
   0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
   0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
   0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,
   0xC1, 0x81, 0x40
};
const UINT8 abCrc16Lo[] =
{
   0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07,
   0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF,
   0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8, 0xD8,
   0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F,
   0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16,
   0xD6, 0xD2, 0x12, 0x13, 0xD3, 0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30,
   0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5,
   0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE,
   0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9,
   0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
   0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26, 0x22,
   0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,
   0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64,
   0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A,
   0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
   0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C,
   0xB4, 0x74, 0x75, 0xB5, 0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3,
   0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53,
   0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C,
   0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
   0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A,
   0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C, 0x44, 0x84,
   0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41,
   0x81, 0x80, 0x40
};
```

```
UINT16 CRC_Crc16( UINT8* pbBufferStart, UINT16 iLength )
{
   UINT8   bIndex;
   UINT8   bCrcLo;
   UINT8   bCrcHi;
   bCrcLo = 0xFF;
   bCrcHi = 0xFF;
   while( iLength > 0 )
   {
      bIndex = bCrcLo ^ *pbBufferStart++;
      bCrcLo = bCrcHi ^ abCrc16Hi[ bIndex ];
      bCrcHi = abCrc16Lo[ bIndex ];
      iLength--;
   }
   return( bCrcHi << 8 | bCrcLo );
}
```

HMSI-168-97 3.1 en-US / 2019-03-01 / 12072