Anybus
BY HMS NETWORKS

Anybus® CompactCom™ 40

HMS Connecting Devices™

Important User Information

Disclaimer

The information in this document is for informational purposes only. Please inform HMS Networks of any inaccuracies or omissions found in this document. HMS Networks disclaims any responsibility or liability for any errors that may appear in this document.

HMS Networks reserves the right to modify its products in line with its policy of continuous product development. The information in this document shall therefore not be construed as a commitment on the part of HMS Networks and is subject to change without notice. HMS Networks makes no commitment to update or keep current the information in this document.

The data, examples and illustrations found in this document are included for illustrative purposes and are only intended to help improve understanding of the functionality and handling of the product. In view of the wide range of possible applications of the product, and because of the many variables and requirements associated with any particular implementation, HMS Networks cannot assume responsibility or liability for actual use based on the data, examples or illustrations included in this document nor for any damages incurred during installation of the product. Those responsible for the use of the product must acquire sufficient knowledge in order to ensure that the product is used correctly in their specific application and that the application meets all performance and safety requirements including any applicable laws, regulations, codes and standards. Further, HMS Networks will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features or functional side effects found outside the documented scope of the product. The effects caused by any direct or indirect use of such aspects of the product are undefined and may include e.g. compatibility issues and stability issues.

Table of Contents

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 40 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 M40 Hardware Design Guide	HMS	HMSI-216-126
Anybus CompactCom B40 Design Guide	HMS	HMSI-27-230
Anybus CompactCom Host Application Implementation Guide	HMS	HMSI-27-334
Anybus CompactCom 40 Network Guides (separate document for each supported fieldbus or network system)	HMS	

1.3 Document History

Version	Date	Description
0.50	2013-07-02	New document
0.60	2013-12-20	General update
1.00	2014-03-28	Major update
1.10	2014-05-26	Major update
1.20	2014-08-15	Major update
1.21	2014-08-26	Major update
1.20	2014-11-10	Major update
1.31	2015-02-06	Minor update
2.00	2015-09-24	Major update
3.0	2016-08-31	Moved from FM to DOX Major update
3.1	2017-04-03	CC-Link IE Field added Misc. updates and corrections
3.2	2017-06-15	BACnet/IP added Misc. updates and corrections
3.3	2017-07-10	Added attr #7 to Application Object (FFh) Updated information on LED status register Added appendix on backward compatibility
3.4	2017-11-28	Added attr #8 - #11 to Application Object (FFh) Minor updates
3.5	2018-03-09	Misc. updates Comparison tables updated for EPL and ECT
3.6	2018-05-28	Misc. updates
3.7	2018-09-19	Updates for MQTT support
3.8	2018-10-23	Minor corrections
3.9	2019-03-01	Corrections to Module Device object description Added info on CANopen module Minor updates Rebranded

Version	Date	Description
4.0	2019-11-11	Updated Anybus Ojbect (01h) Minor updates Changed disclaimer
4.1	2021–04–16	Added information about the Name attribute in the Assembly Mapping Object (EBh) Added Time Object (13h)

1.4   Document Conventions

Numbered lists indicate tasks that should be carried out in sequence:

1.   First do this

2.   Then do this

Bulleted lists are used for:

•   Tasks that can be carried out in any order

•   Itemized information

►   An action

      →   and a result

User interaction elements (buttons etc.) are indicated with bold text.

```
Program code and script examples
```

Cross-reference within this document: *Document Conventions, p. 7*

External link (URL): www.hms-networks.com

> ⚠ **WARNING**
> Instruction that must be followed to avoid a risk of death or serious injury.

> ⚠ **Caution**
> Instruction that must be followed to avoid a risk of personal injury.

> ❗ Instruction that must be followed to avoid a risk of reduced functionality and/or damage to the equipment, or to avoid a network security risk.

> ⓘ   *Additional information which may facilitate installation and/or operation.*

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 Trademarks

- 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 40

2.1   General Information

The Anybus CompactCom 40 network communication module is a powerful communication solution for demanding industrial applications. It is ideal for high performance, synchronized applications such as servo drive systems. Typical applications are PLCs, HMIs, robotics, AC/DC and servo drives.

The Anybus CompactCom 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 module. This allows for a very high level of integration, since the host communication protocol enables the Anybus module to retrieve information directly from the host application using explicit object requests rather than memory mapped data.

> **!** The Anybus CompactCom 40 series is backward compatible with the Anybus CompactCom 30 series though the 40 series has significantly better performance and include more functionality than the 30 series. The 40 series is backward compatible with the 30 series in the sense that an application developed for the 30 series should be possible to use with the 40 series products, even though minor application code changes may be necessary.
>
> The 40 series products can thus not replace 30 series products as is.

2.2   Features

- Hardware support for triple buffered process data

- Black channel interface, offering a transparent channel for safety communication

- Host interface is network protocol independent

- Multilingual support

- High level of integration

- Synchronization support

- 8-bit and 16-bit parallel modes

- SPI mode

- Stand-alone shift register mode

- Serial interface mode (UART)

- Optional support for advanced network specific features

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.
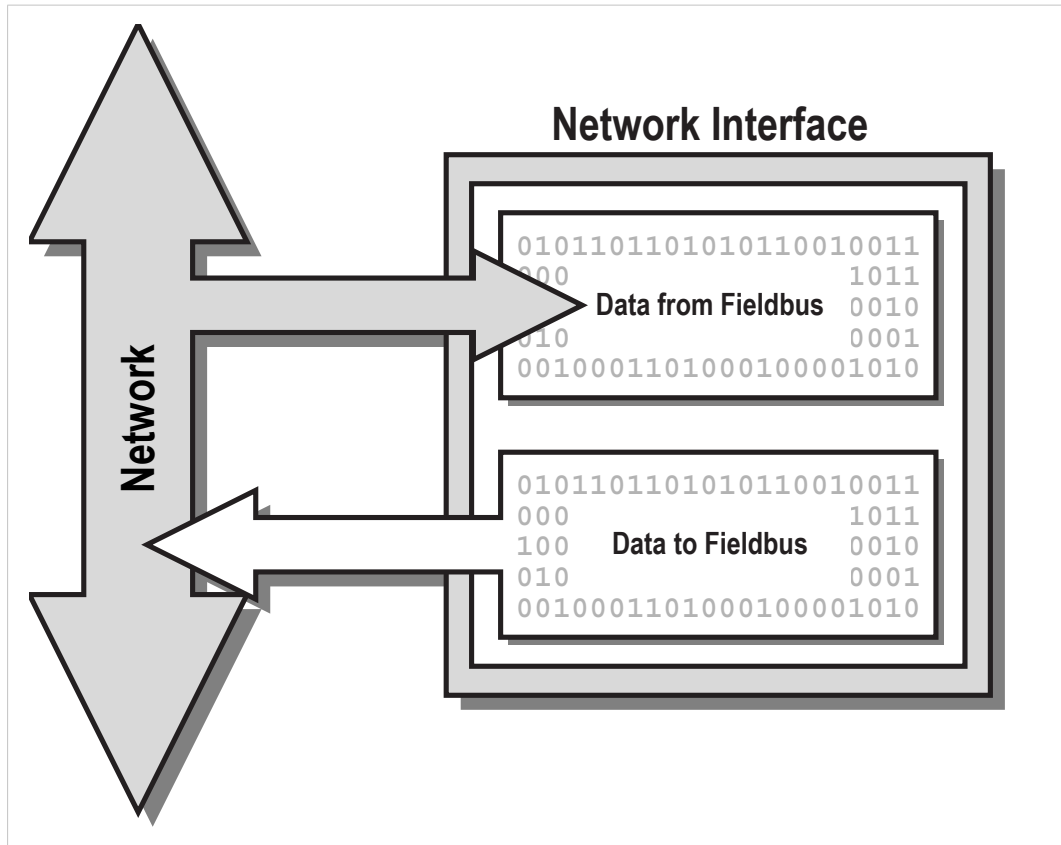
Fig. 1

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.

Network Interface


```
0101101101010110010011
000                1011
010                0001
001000110100010001010
```
Data from Fieldbus

Cyclic Data

```
0101101101010110010011
000                1011
100                0010
010                0001
001000110100010001010
```
Data to Fieldbus

Cyclic Data

Acyclic Request

Acyclic Handling

Acyclic Response

Alarm

Diagnostic Handling

Diagnostics

Network
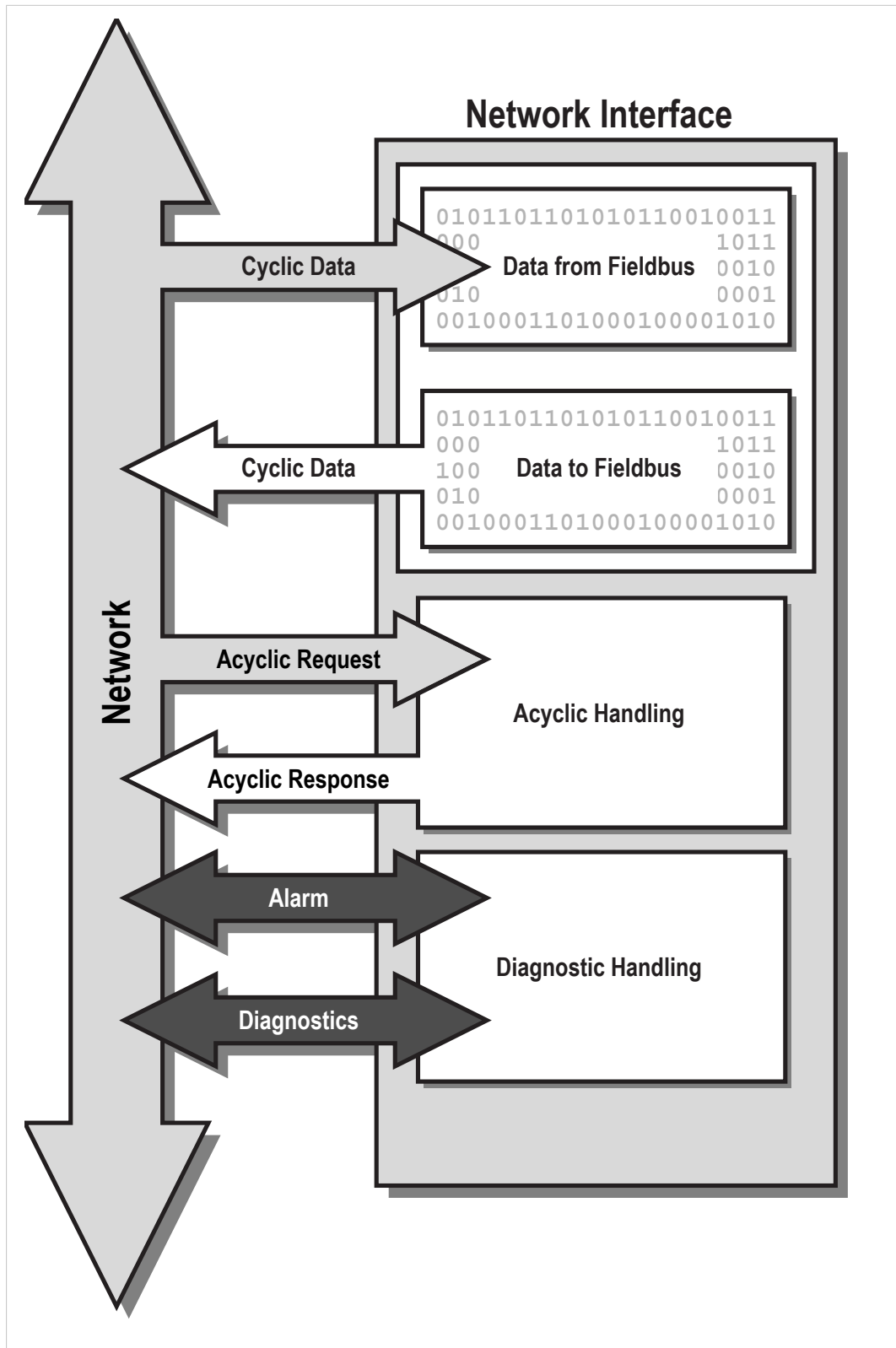
Fig. 2

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.
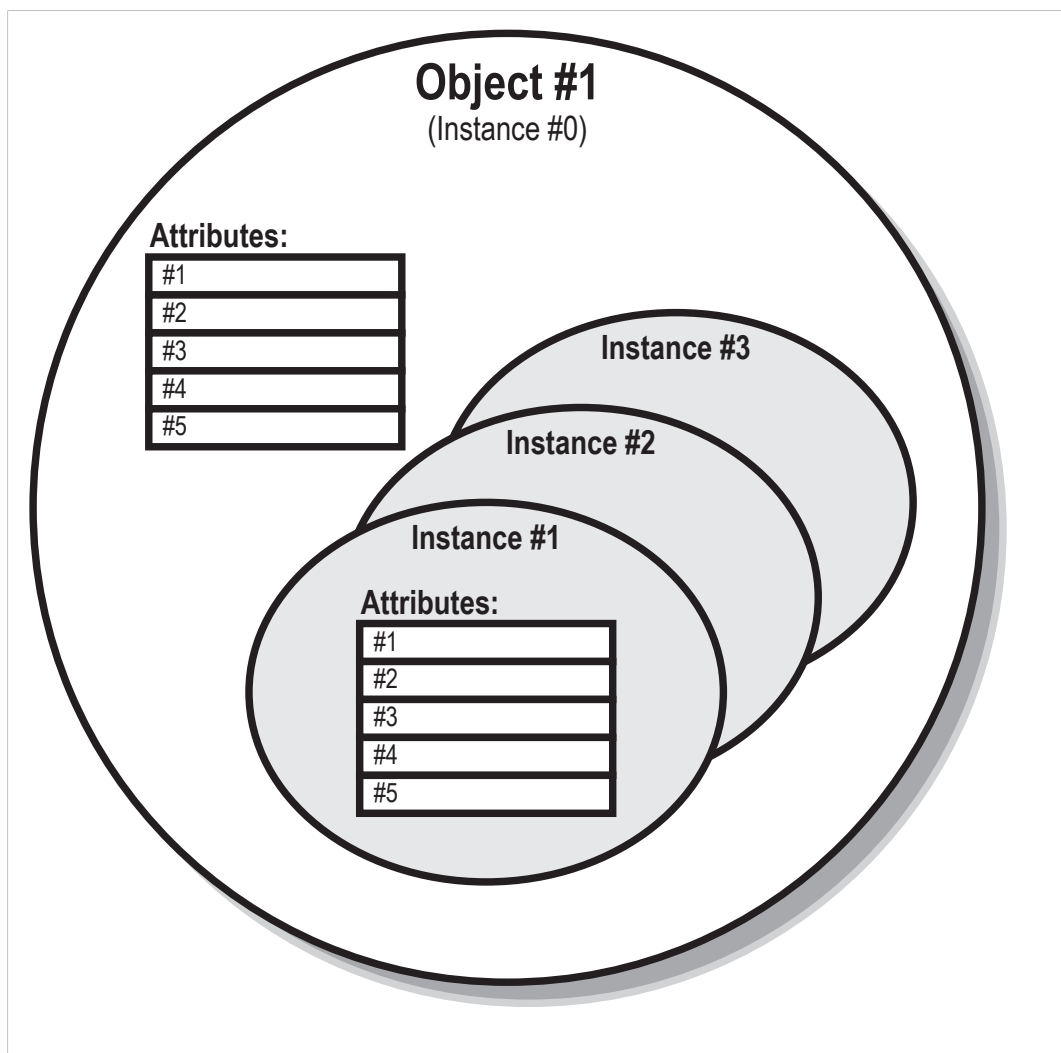
Fig. 3

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.

Anybus Module Objects

The following objects are implemented in the standard Anybus CompactCom 40 firmware:

• Anybus Object

• Diagnostic Object

• Network Object

• Network Configuration Object

• File System Interface Object

• Functional Safety Module Object

• Time 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. 61

Host Application Objects

The following objects can be implemented in the host application.

• Application Data Object (Mandatory)

• Application Object (Mandatory)

• Sync Object (Optional)

• Modular Device Object (Optional)

• Assembly Mapping Object (Optional)

• File System Interface Object (Optional)

• Energy Control Object (Optional)

• Functional Safety 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.

See also...

Host Application Objects, p. 107

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.
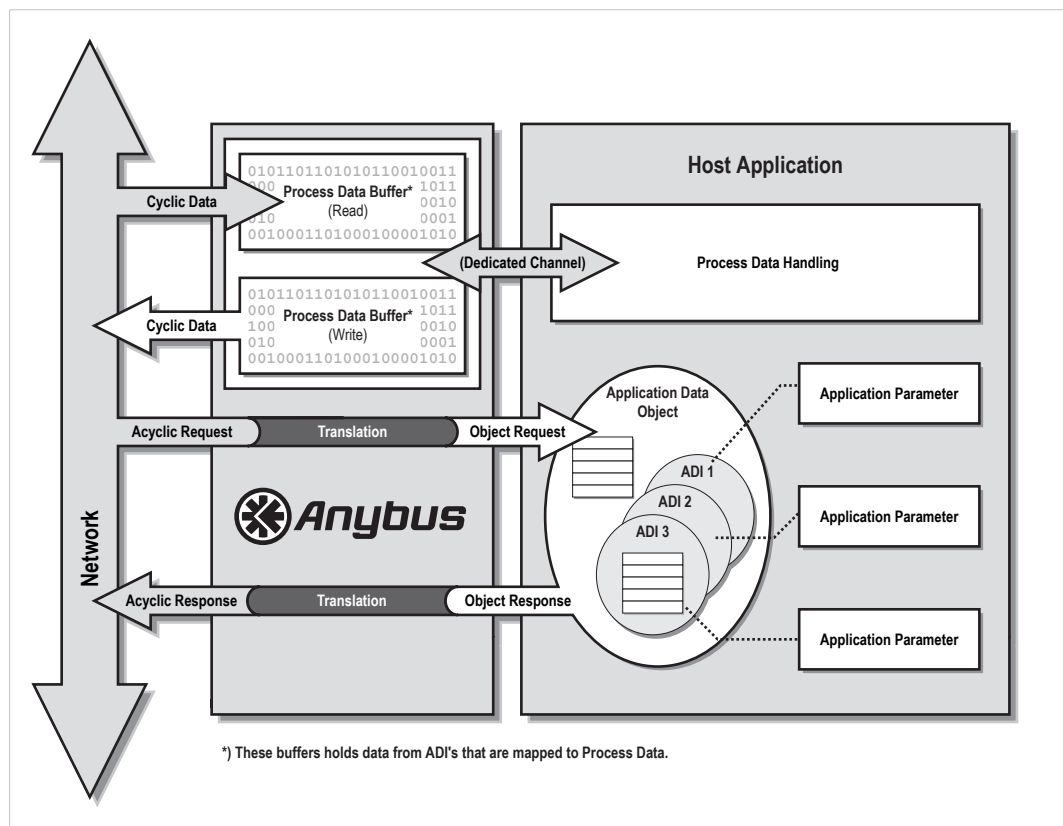
Fig. 4

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 40, 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...

The Anybus State Machine, p. 43

Network Object (03h), p. 74

Functional Safety Object (E8h), p. 109

3.4   Diagnostics

The Anybus CompactCom 40 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.
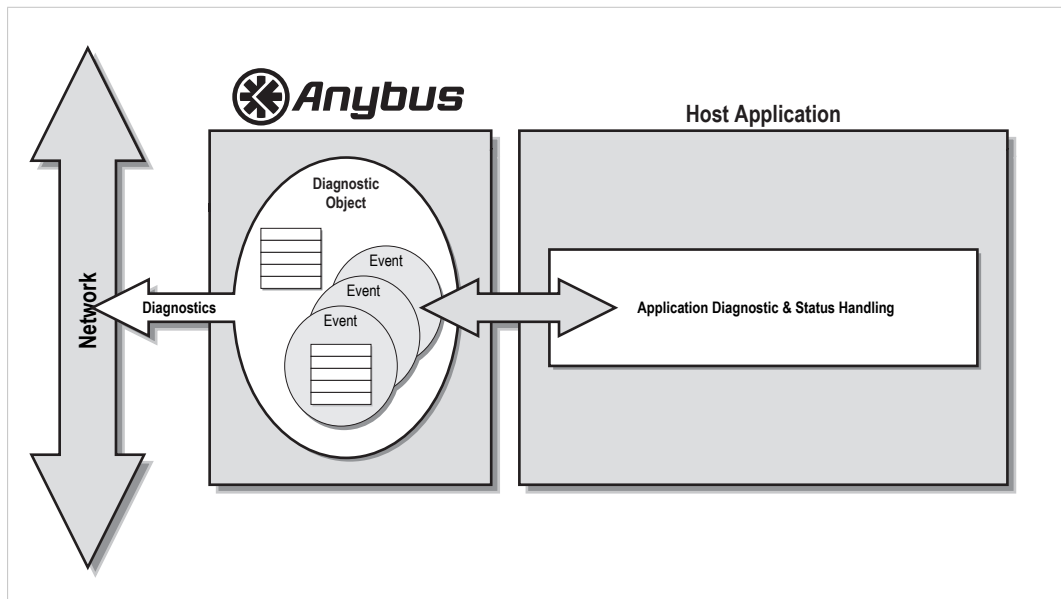
Fig. 5

See also...

Diagnostic Object (02h), p. 69

3.5   File System

The modules in the Anybus CompactCom 40 series have a built-in file system.

For modules not supporting FTP, this makes it possible to store firmware files in the firmware directory using the File System Interface Object (0Ah). No other access to or use of the file system is possible for these modules.

For modules supporting FTP, the in-built file system can be accessed from the application and from the network. The file system can not be deleted.

Three directories are predefined:

VFS	The virtual file system that e.g. holds the web pages of the module.
Application	This directory provides access to the application file system through the Application File System Interface Object (EAh) (optional). The directory can not be accessed from the application, only from the network.
Firmware	Firmware updates are stored in this directory.

> **!** In the firmware folder, it is not possible to use append mode when writing a file. Use write mode only.
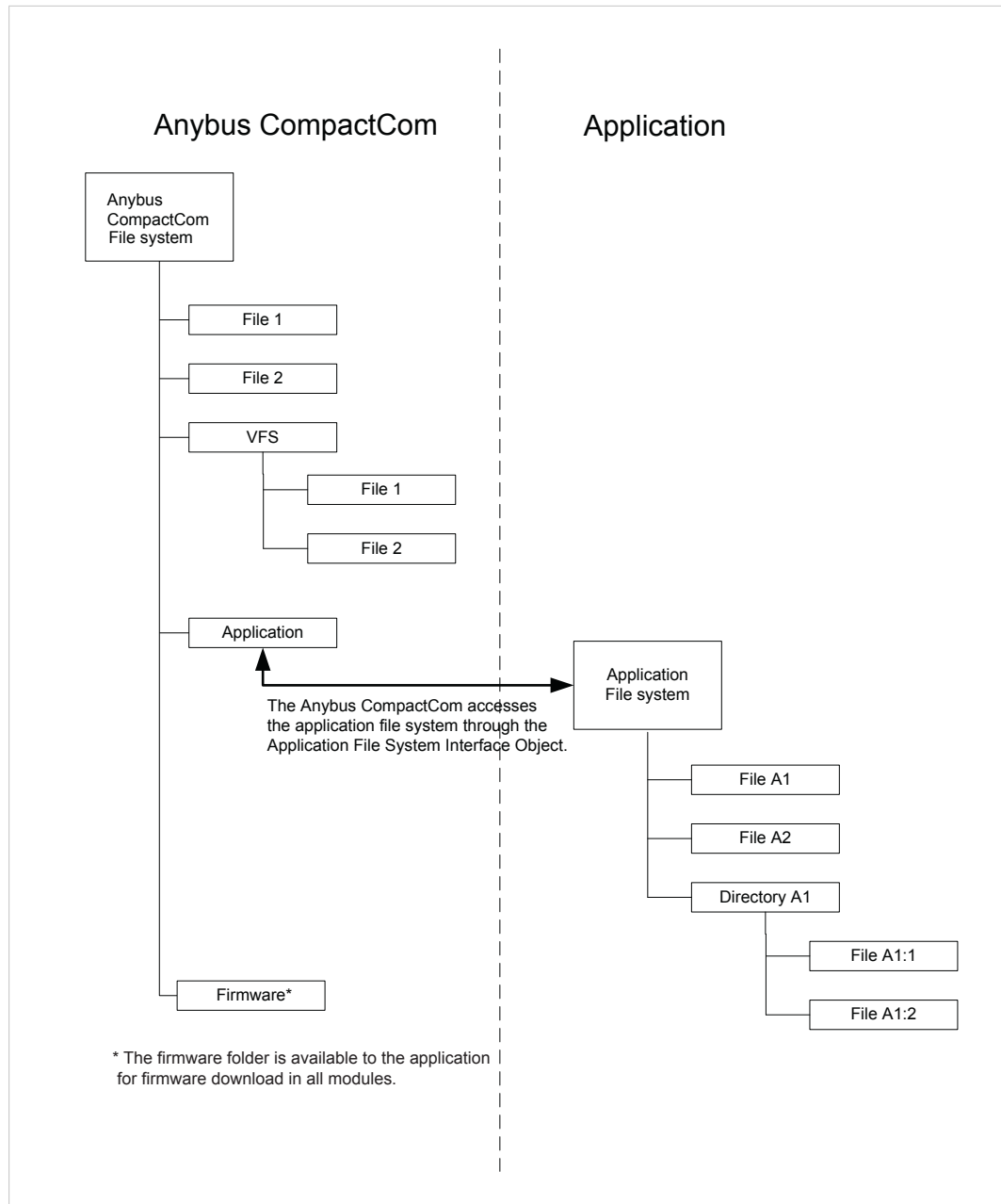
Fig. 6

See also...

Anybus CompactCom 40 Network Guides, available at www.anybus.com

3.6   Modular Device

The modular device functionality makes it possible to model a structure of the process data on to a number of modules of different types within an application, e.g. for handling digital input or output, analog input or output, or drives. The ADIs are distributed among the modules and the number of ADIs per module is configurable. The modules are physically connected to a backplane, with a number of slots. The first slot is occupied by the coupler, which contains the Anybus CompactCom module. All other slots may be empty or occupied by modules. When mapping ADIs to process data, the application shall map the process data of each module in slot order.

See also:

- *Modular Device Object (ECh), p. 138*

- Anybus CompactCom 40 Network Guides

3.7   SYNC

3.7.1   General Information

Automation systems involving many devices often require a way to synchronize events. To achieve this, the devices in the system can share a common timing signal. The Anybus CompactCom 40 supports a SYNC mechanism via the SYNC object, that is optional to implement in the application.

The following Anybus CompactCom 40 modules support the SYNC functionality:

- Ethernet POWERLINK

- PROFINET-IRT

- EtherCAT

See also:

- *Sync Object (EEh), p. 140*.

- *Application Status Register, p. 30*

- *The Anybus State Machine, p. 43* for information of the different states of the Anybus CompactCom module.

3.7.2   Functionality

For a successful SYNC implementation, there are a number of things to implement and consider.

The network master will configure attributes #1-3 and #7 of the SYNC object through the Anybus CompactCom module before entering state IDLE or PROCESS_ACTIVE. If the module attempts to set attributes #1-3 in state IDLE or PROCESS_ACTIVE, the application must respond with error code 0Dh (Invalid state). For unsupported values for the attributes, the application must respond with a suitable error code (11h (Value too high), 12h (Value too low) or 0Ch (Value out of range)).

If there is a problem with the configuration as a whole, the application must indicate this in the application status register. See *Application Status Register, p. 30*.

The application must indicate its minimum supported cycle time and the required input/output processing times in attributes #4-6 of the SYNC object at all times. The value of these attributes can be constant or vary, reflecting the timing required for the current process data mapping.

3.7.3 Synchronization Lock

If the application needs time to lock on the SYNC signal, it must write 0001h ("Application not yet synchronized") to the application status register. When synchronization lock has been achieved, and there are no configuration errors, the application must write 0000h to the application status register and then accept a transition to PROCESS_ACTIVE.

Whenever the application is not locked on the SYNC signal, and attribute #7 "Sync mode" in the SYNC object is set to "1", the application must write the most accurate nonzero status code to the application status register.

See also

Application Status Register, p. 30

3.7.4 SYNC Pulse

The SYNC signal, available from the module's application connector, indicates the synchronization event to the application by a positive pulse once every cycle. The positive edge of the SYNC pulse indicates the synchronization event.

The width of the SYNC pulse is at least 5 µs, with a maximum width of 50% of the cycle time.

The SYNC event is also available to the application as a maskable interrupt. See *Interrupt Status Register, p. 32*.

3.7.5 Network Translation

Ethernet POWERLINK does not in itself support synchronization functionality. The SYNC signal from the module is sent once for each cycle, and can as such be used by the application.

In Anybus CompactCom 40 EtherCAT, parameters and settings are stored in CoE objects 1C32h and 1C33h.

The Anybus CompactCom 40 PROFINET IRT supports both isochronous and non-isochronous modes.

For more information, please consult the respective network guides.

3.7.6   Anybus CompactCom 40 SYNC Implementation

The Goal of SYNC

To set output data to different devices simultaneously.	The PLC will tell all devices in the network to set the next set of available output data at the application, when the SYNC signal is sent.
	The time when the output data is set at the application is called the Output Valid Point.
To capture input data from different devices simultaneously.	The time when the input data is available is called the Input Capture Point.
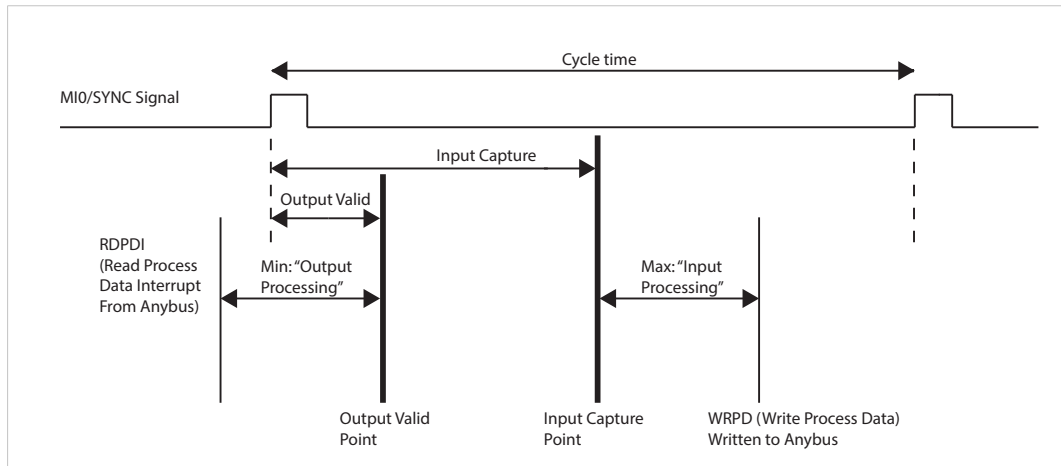
Fig. 7

Handling of Output Data

Each device needs time to handle the new output data before it can be set in the application. This time is not constant.

The device has to follow these steps:

1. Wait for indication of new output data (indicated by the Anybus CompactCom 40 through the RDPDI (Read Process Data Interrupt).

2. Read the output data from the Anybus CompactCom 40 when receiving the RDPDI.

3. Process the new output data so that it can be used by the host application

    –   copy data

    –   process output variables

    –   do calculations

    –   etc

4. Wait for the SYNC signal.

5. When the SYNC signal arrives:

    –   If "Output Valid" = 0, activate the outputs to the host application.

    –   If "Output Valid" > 0, start a hardware or a software timer on the positive edge of the SYNC signal. Activate the outputs to the host application when the timer has reached "Output Valid".

See *Buffer Control Register, p. 31* and *Interrupt Status Register, p. 32* for more information on RDPD (Read Process Data) and RPDPI (Read Process Data Interrupt)

Handling of Input Data

Each device needs time to capture, prepare and send new input data. This time is not constant.

The device has to follow these steps:

1. Wait for the SYNC signal.

2. When the SYNC signal arrives ("Input Capture" point):

    –   If "Input Valid" = 0, capture the current input process variables of the host application.

    –   If "Input Valid" > 0, start a hardware or a software timer on the positive edge of the SYNC signal. Capture the current input process data when the timer has reached "Input Valid".

3. Prepare the new input process data so that it can be written to theAnybus CompactCom 40.

4. Write the new process input data to theAnybus CompactCom 40.

Host Application Programming Guidelines

See *Sync Object (EEh), p. 140* for more information.

1. Implement the SYNC Object (part 1)

7	Sync mode	Get/Set	UINT16	This attribute is used to select synchronization mode. It enumerates the bits in attribute 8 0: Non synchronous operation. (Default value if non synchronous operation is supported) 1: Synchronous operation 2 - 65535: Reserved. Any attempt to set sync mode to an unsupported value shall generate an error response
8	Supported sync modes	Get	UINT16	A list of the synchronization modes the application supports. Each bit corresponds to a mode in attribute 7 Bit 0: 1 = Non synchronous mode supported Bit 1: 1 = Synchronous mode supported Bit 2 - 15: Reserved (0)

2. Implement the SYNC Object (part 2)

4	Output processing	Get	UINT32	Minimum required time, in nanoseconds, between RDPDI interrupt and "Output valid"
5	Input processing	Get	UINT32	Maximum required time, in nanoseconds, from "Input capture" until write process data has been completely written to the Anybus CompactCom 40
6	Min cycle time	Get	UINT32	Minimum cycle time supported by the application (in nanoseconds)

The time elapsed between receiving an RDPDI interrupt and when the process output variables have been taken over by the application has to be measured. The maximum time must be provided by the application when the Anybus CompactCom 40 asks for attribute #4 "Output processing". The network master has to make sure that the minimum output processing time is longer than the maximum time measured.

The time elapsed between capturing the input process variables and when the input process data is written to the CompactCom 40 must be measured. The maximum time must be provided by the application when the Anybus CompactCom 40 asks for attribute #5 "Input processing".

The host application must measure the maximum time needed to handle all process data (the time from receiving the RDPDI interrupt until the write process data has been written to the Anybus CompactCom 40). This value must be provided by the application when the Anybus CompactCom 40 asks for attribute #6 "Min cycle time".

3.   Implement the SYNC Object (part 3)

1	Cycle time	Get/Set	UINT32	Application cycle time in nanoseconds
2	Output valid	Get/Set	UINT32	Output valid point relative to SYNC events, in nanoseconds Default value: 0
3	Input capture	Get/Set	UINT32	Input capture point relative to SYNC events, in nanoseconds Default value: 0

These three attributes can all be set by the Anybus CompactCom 40.

Using attribute #1, "Cycle time", theAnybus CompactCom 40 informs the host application about the real used cycle time (by the Set_Attribute command). This value must be evaluated by the host application and refused if not acceptable (not suitable, e.g. in conflict with other cyclic tasks of the host application or not within the defined range). If refused, the Anybus CompactCom 40 will report this to the PLC.

Attributes #2 and #3 reflect functionality present on some networks (e.g. EtherCAT, SERCOS and PROFINET), where the input capture and output valid points can be fine-tuned. This can be used to offset one device relative to another by a small amount of time. To support values other than zero (0), timers will have to be implemented in the application.

4.   Implement the Application Status Register

See for more information.

5.   Act upon receiving an RDPDI and a SYNC signal

When receiving an RDPDI interrupt, read the output process data from the Anybus CompactCom 40, prepare it (handle and assign it to process output variables) so that it can be activated when receiving a SYNC signal.

When receiving a SYNC signal, do the following:

a.   Transfer the output process variables to the application immediately.

b.   Capture all input process variables immediately.

c.   Prepare and assign the input process variables to the input process data.

d.   Write the input process data to the CompactCom.

Steps b, c and d must be done within the time specified by attribute #5 "Input processing".

Steps a and b assumes Output Valid and Input Capture to be zero (0).

The Easiest Realization of a Synchronous Application
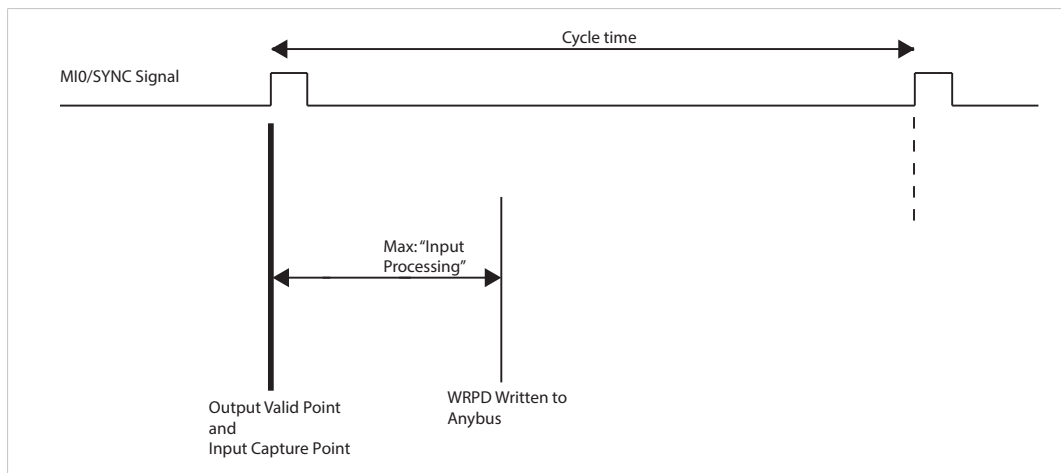
Fig. 8

The following steps show how to create a very simple synchronous application.

1. Set up an interrupt routine triggered by the positive SYNC edge. Disable the RDPDI interrupt.

2. In the SYNC interrupt routine:

    – Sample the input data and write it to the Anybus CompactCom 40.

    – Read the output data from the Anybus CompactCom 40 and start using it immediately.

3. For this application, attribute #4 "Output processing" in the SYNC object should be set to zero (0). No measurement needed.

4. Attribute #5 "Input processing" must be determined. It can probably be hard coded to a fixed value, but this is application specific and depend on the complexity of the SYNC interrupt routine and the application processor performance.

5. For attributes #2 "Output valid" and #3 "Input capture", only the value zero (0) will be accepted by the application.

This simple step-by-step method will work in all applications where the process data handling can be made fast and simple.

3.8   Multilingual Support

Where applicable, the Anybus CompactCom 40 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 40 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. 120

3.9   Firmware Download

Download and upgrade of network communication firmware for a specific fieldbus or industrial network can be performed in different ways, depending on which Anybus CompactCom 40 that is to be upgraded.

3.9.1   Important

When the Anybus CompactCom 40 is restarted after a firmware download, the application must wait for the installation to finish before initialization is started. The Anybus CompactCom 40 is protected against power failure during download and/or installation and will recover upon restart.

- If download through e.g. Firmware Manager or FTP, is interrupted, please restart the firmware download process from the beginning.

- To install the new firmware after download is completed, reset the Anybus CompactCom 40. If the installation of the new firmware is interrupted, e.g. due to power loss, please restart the Anybus CompactCom 40. The installation process will automatically start from the beginning and the new firmware will be installed without any further action.

For more information see, *Startup Procedure, p. 57*

3.9.2 Using Firmware Manager II

This tool is available without cost from www.anybus.com. It can be used to download new firmware for any Anybus CompactCom 40.
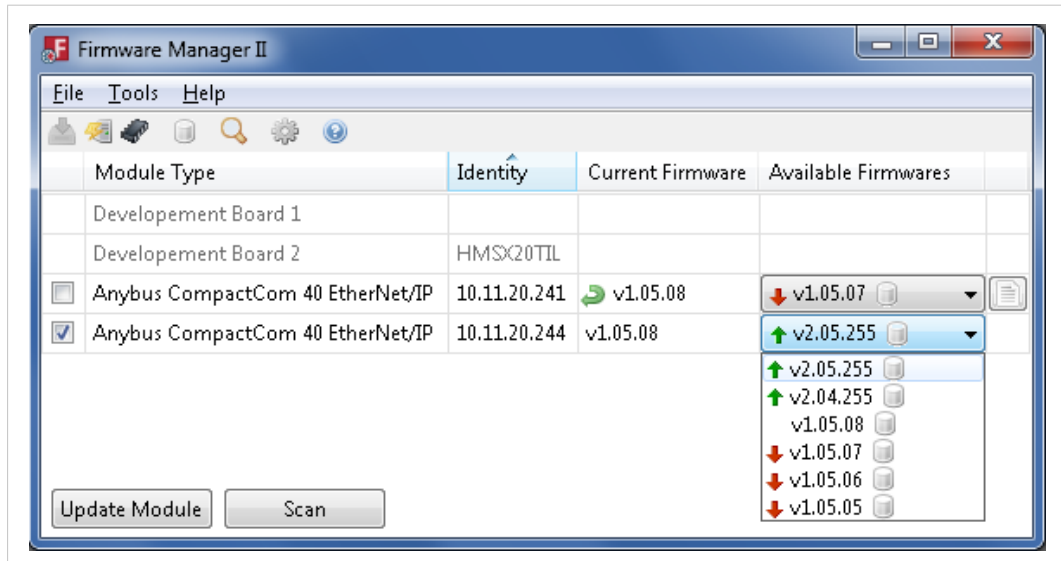
Fig. 9

Using the tool, perform the following steps to download new firmware to the module.

1. Connect a computer with the Firmware Manager II software installed to the network containing the module.

2. Start the Firmware Manager II tool.

3. Scan the network and find the module.

4. Click the Firmware Repository icon in the menu, to open the Firmware Repository window. Drag the firmware folder into the window to add the new firmware to the repository. Close the Firmware Repository window.

5. In the scan window, under the "Available Networks" tab, select the appropriate firmware for the module. Click the **Change Network** button. A confirmation window will appear. Click **Yes** to start the download of the new firmware. Please make sure that download is completely finished before continuing.

6. After download, a restart of the module is needed to install the new firmware. If the application allows it, it is possible to restart the module via the **Restart Module** button in the Firmware Manager II tool. If the application does not allow restart from the network, a manual restart of the module is needed.

For more information, see the help file in the Firmware Manager II software.

3.9.3 Using the Internal File System

The internal file system can be accessed via the File System Interface Object. Interfacing this object from the host application, makes it possible to store the new firmware in the /firmware directory in the internal file system. The next time the module is started the firmware will be upgraded. After the firmware is installed, the firmware file is deleted from the /firmware directory.

> **!** In the firmware folder, it is not possible to use append mode when writing a file. Be sure to use write mode only.

See also ...

- *Application File System Interface Object (EAh), p. 132*

3.9.4 Using FTP

If the module supports FTP, this can be used to access the file system and upload the new firmware directly to the /firmware directory. The next time the module is started the firmware will be upgraded. After the firmware is installed, the firmware file is deleted from the /firmware directory.

See also ...

- *Application File System Interface Object (EAh), p. 132*

4 Host Communication Layer

4.1 General Information

The main communication layer is used by the 8-bit/16-bit parallel modes and the SPI mode. It is divided into process data read/write areas, message data read/write areas and a number of control registers.

Below is a detailed description of the memory map and the different control registers.

4.1.1 Communication Basics

The communication between the host and the Anybus CompactCom 40 is simple, fast and flexible.

The host can read or write process data at any time. It can check for incoming data via the buffer control register or by enabling appropriate interrupts via the interrupt mask register.

> **!** Attempts to access reserved registers will produce unpredictable results. Attempts to write to a read-only register will produce unpredictable results. Reserved bits shall be written with zeros (0). Reading reserved bits returns undefined values.

4.2 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 interface has been implemented.

> **i** *The memory area is not available during reset.*

The shaded areas and registers are used for backward compatibility with the Anybus CompactCom 30 series.

Byte Address	Word Address	Area	Bytes	Access, seen from host	Notes
0000h - 0FFFh	0000h - 07FFh	Process data, write area	4096	R/W	This is the total size of the area. The actual size depends on the network. For network specific process area sizes, see *Network Comparison, p. 144*. Applicable to all protocols
1000h - 1FFFh	0800h - 0FFFh	Process data, read area	4096	R	
2000h - 25FFh	1000h - 12FFh	Message data, write area	1536	R/W	Applicable to all protocols
2600h - 2FFFh	1300h - 17FFh	Reserved	2560	-	
3000h - 35FFh	1800h - 1AFFh	Message data, read area	1536	R	
3600h - 37FFh	1B00h - 1BFFh	Reserved	512	-	
3800h - 38FFh	1C00h - 1C7Fh	Process data, write area	256	R/W	
3900h - 39FFh	1C80h - 1CFFh	Process data, read area	256	R	
3A00h - 3AFFh	1D00h - 1D7Fh	Reserved	256	-	
3B00h - 3C06h	1D80h - 1E03h	Message data, write area	263	R/W	
3C07h - 3CFFh	1E04h - 1E7Fh	Reserved	249	-	
3D00h - 3E06h	1E80h - 1F03h	Message data, read area	263	R	
3E07h - 3FE7h	1F04h - 1FF3h	Reserved	481	-	
3FE8h - 3FEFh	1FF4h - 1FF7h	Current network time	8	R	
3FF0h - 3FF1h	1FF8h	Module capability register	2	R	
3FF2h - 3FF3h	1FF9h	LED status register	2	R	
3FF4h - 3FF5h	1FFAh	Application status register	2	R/W	Applicable to the event driven protocol.
3FF6h - 3FF7h	1FFBh	Anybus CompactCom module status register	2	R	
3FF8h - 3FF9h	1FFCh	Buffer control register	2	R/W	
3FFAh - 3FFBh	1FFDh	Interrupt mask register	2	R/W	
3FFCh - 3FFDh	1FFEh	Interrupt status register	2	R/W	
3FFEh	1FFFh	Control register	1	R/W	Applicable to the half duplex protocol
3FFFh		Status register	1	R	

If an application is to use "current network time", the network time must first be sampled. This is performed by writing to this register. The register will be updated with the actual value from the network, which then can be read by the application.

> **i** *C-programmers are reminded to declare the entire shared memory area as volatile.*

4.3   Communications Registers

4.3.1   Module Capability Register

The module capability register contains one of the values below, indicating module type. The application should determine the module type by examining the low byte only. The high byte is reserved for future use.

Value	Module Type
0000h	Active Anybus CompactCom 30 series module, supporting the half duplex protocol only 8 bit parallel and serial modes
000Bh	Active Anybus CompactCom 40 series module, supporting the event driven as well as the half duplex protocol 8 bit/16 bit parallel, shift register, SPI and serial modes
000Ch	Active Anybus IP with parallel AXI bus
0101h	Passive RS232
0202h	Passive RS422
0303h	Passive USB
0404h	(reserved)
0505h	Passive Bluetooth
0606h - 0909h	(reserved)
0A0Ah	Passive RS485
0C0Ch - 0F0Fh	(reserved)

> **ℹ** *All passive modules belong to the Anybus CompactCom 30-series. There are no passive modules in the Anybus CompactCom 40-series.*

4.3.2   LED Status Register

The first 8 bits of this register reflect the LED status, as represented by the value of the instance attribute LED status (#13) in the Anybus Object. See *Anybus Object (01h), p. 62* for more information. The following for bits reflect status of the signals LED5A, LED5B, LED6A and LED6B.

If a bit is set to 1, the status of the corresponding LED is ON.

Bit	LED Signal
0	LED1A
1	LED1B
2	LED2A
3	LED2B
4	LED3A
5	LED3B
6	LED4A
7	LED4B
8	LED5A
9	LED5B
10	LED6A
11	LED6B
12–15	(none)

4.3.3   Application Status Register

The application status register is primarily used in SYNC applications. It is used in applications where the network in question supports the ability to indicate critical process data errors to the master. If this is supported, the Anybus CompactCom 40 module will accept and handle the below listed status codes written by the application.

This register is optional to use. For networks which do not support indications of critical process data errors, the module will ignore this register.

The register contains an 8 bit value. When the register is accessed from the parallel interface, the value is provided as a word aligned value with the upper 8 bits always set to 0. See *Memory Map, p. 29*.

Value	Error	Description
00h	No error	Ready for transition to state PROCESS_ACTIVE (Default)
01h	Not yet synchronized	Not ready for transition to state PROCESS_ACTIVE
02h	Sync configuration error	A problem with the current attribute values in the Sync object prevents the transition to state PROCESS_ACTIVE
03h	Read process data configuration error	A problem with the current read process data mapping prevents the transition to state PROCESS_ACTIVE
04h	Write process data configuration error	A problem with the current write process data mapping prevents the transition to state PROCESS_ACTIVE
05h	Synchronization loss	The application has lost synchronization lock If the Anybus CompactCom is in the state PROCESS_ACTIVE, it will change to a lower state
06h	Excessive data loss	The application has detected a significant loss of process data from the network If the Anybus CompactCom is in the state PROCESS_ACTIVE, it will change to a lower state
07h	Output error	Application malfunction If the Anybus CompactCom is in the state PROCESS_ACTIVE, it will change to a lower state

The Anybus state machine is described in *The Anybus State Machine, p. 43*

4.3.4 Anybus CompactCom Module Status Register

This register contains the current Anybus CompactCom module state, and a supervised bit indicated by the Anybus CompactCom module. The Anybus state machine is described in *The Anybus State Machine, p. 43*

Bit	Name	Description
0 - 2	S[0..2]	The current Anybus CompactCom module state
3	Supervised bit	1 = Supervised by another network device 0 = Not supervised by another network device See *Supervised Bit (SUP), p. 34*.
4 - 15	-	Reserved (0)

4.3.5 Buffer Control Register

This register is used by the application to control the event driven communication with the Anybus CompactCom module.

By writing to this register, it is possible to trigger appropriate events. Write "1" to trigger bits, and "0" to leave bits unaffected.

Reading this register gives the current status of the different memory areas.

For more information about how to implement and use bits 0–3, see*Communication Basics, p. 35*

Bit	Name	Description
0	WRPD (Write Process Data)	The application shall write "1" to this bit when new data has been written.
1	RDPD (Read Process Data)	When the module has updated the read process data, this bit will be read as "1". The application shall write "1" to this bit to request the latest read process data. By doing this the bit is cleared. **Note:** "updated" data does not necessarily mean "changed" data.
2	WRMSG (Write Message Data)	This bit is read as "1" when the write message area is occupied. This bit is cleared by the module when the write message area is available for a new message. When the application sends a write message to the module, it shall write "1" to this bit. The application is only allowed to write to the write message area while this bit is "0". **Note:** it is only allowed to write command messages when the ANBR bit is also set.
3	RDMSG (Read Message Data)	This bit will be read as "1" when the module has posted a new read message. The application writes "1" to this bit to acknowledge the message. By doing this the bit is cleared. The application is only allowed to read the read message area while this bit is "1".
4	ANBR (Anybus Ready)	This bit is set to "1" when the module is ready to receive a new command message. The application is only allowed to send command messages while this bit is "1". This bit can only change from "1" to "0" while WRMSG is "1". It can change from "0" to "1" at any time.
5	APPR (Application Ready)	The application writes '1' to this bit when it is ready to receive a new command message. The module will only send command messages while this bit is "1". Use APPRCLR to set this bit to "0".
6	APPRCLR (Application Ready Clear)	The application can write "1" to this bit to clear the APPR bit. This is only allowed when RDMSG is "1".
7 - 15	-	Reserved

4.3.6 Interrupt Mask Register

This register makes it possible for the application to enable or disable individual interrupts, according to the table below.

Bit	Name	Description
0	RDPDIEN	Set this bit to "1" to enable interrupt for when the RDPD bit in the buffer control register transitions from "0" to "1".
1	RDMSGIEN	Set this bit to "1" to enable interrupt for when the RDMSG bit in the buffer control register transitions from "0" to "1".
2	WRMSGIEN	Set this bit to "1" to enable interrupt for when the WRMSG bit in the buffer control register transitions from "1" to "0".
3	ANBRIEN	Set this bit to "1" to enable interrupt for when the ANBR bit in the buffer control register transitions from "0" to "1".
4	STATUSIEN	Set this bit to "1" to enable interrupt for an Anybus CompactCom module status register change.
5	-	Reserved
6	SYNCIEN	Set this bit to "1" to enable interrupt for a SYNC event.
7 - 15	-	Reserved

4.3.7 Interrupt Status Register

The module indicates the pending interrupts in this register, according to the table below.

Bit	Name	Description
0	RDPDI	This bit is set to "1" when RDPD in the buffer control register transitions from "0" to "1". The application shall write "1" to this bit to set it to "0".
1	RDMSGI	This bit is set to "1" when RDMSG in the buffer control register transitions from "0" to "1". The application shall write "1" to this bit to set it to "0".
2	WRMSGI	This bit is set to "1" when WRMSG in the buffer control register transitions from "1" to "0". The application shall write "1" to this bit to set it to "0".
3	ANBRI	This bit is set to "1" when ANBR in the buffer control register transitions from "0" to "1". The application shall write "1" to this bit to set it to "0".
4	STATUSI	This bit is set to "1" on an Anybus CompactCom module status register change. The application shall write "1" to this bit to set it to "0".
5	PWRI	This bit is set to "1" when the module is ready to start communication after a power-up or a hardware reset. The application shall write "1" to this bit to set it to "0".
6	SYNCI	This bit is set to "1" on each SYNC event. The application shall write "1" to this bit to set it to "0".
7 - 15	-	Reserved

4.3.8 Control Register (Read/Write)

ℹ️ *Only used for the half duplex (ping/pong) protocol.*

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.
CTRL_R	If set, the host application is ready to receive a new command.
CTRL_AUX	(ignored)
-	(reserved, set to zero)

4.3.9 Status Register (Read Only)

ℹ️ *Only used for the half duplex (ping/pong) protocol.*

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.

Bit	Description
STAT_R	If set, the Anybus module is ready to process incoming commands.
STAT_AUX	See *Auxiliary Bit (STAT_AUX, CTRL_AUX), p. 34*.
SUP	Value: Meaning: 0: Module is not supervised. 1: Module is supervised by another network device. See *Supervised Bit (SUP), p. 34*.
S[0... 2]	These bits indicates the current state of the module (see also *The Anybus State Machine, p. 43*).

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).

4.3.10 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 module. 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.

4.3.11 Auxiliary Bit (STAT_AUX, CTRL_AUX)

The Anybus CompactCom 40 module ignores the CTRL_AUX bit in the Control Register..

The module will set the STAT_AUX bit in the Status Register if new process data has been received from the network since the last telegram.

5   Parallel Host Communication

5.1   Flow Control

The following only applies to the event driven modes (full duplex modes). For information about the half duplex mode, see *Serial Host Communication (UART), p. 42*.

Data can be read or written from either the host or the Anybus CompactCom module, at any point and in any order. Communication can be fully controlled by writing to and reading from the buffer control register, or it can be achieved by enabling interrupts for appropriate events using the interrupt mask register. If enabled, an interrupt is generated each time the module has made new data available.

See *Buffer Control Register, p. 31* and *Interrupt Mask Register, p. 32*.

5.1.1   Communication Basics

When using the parallel host interface, data is exchanged via the shared memory area. For more information, see *Memory Map, p. 29*.

Data Transmission

To write process data :

1.   Write data to the write process data memory area. The area currently mapped by ADIs for process data must be refilled with new data.

2.   Finalize the write process by writing "1" to bit 0 (WRPD) in the buffer control register.

To write message data:

1.   Read bit 2 (WRMSG) in the buffer control register.

     –   If it is "0", the area is available for new message data.

     –   If it is "1", the area is occupied and is not yet available to receive new message data.

2.   Write data to the write message data memory area.

3.   Finalize the write process by writing "1" to bit 2 (WRMSG) in the buffer control register.

Data Reception

For the latest read process data:

1.   Write "1" to bit 1 (RDPD) in the buffer control register, to gain access to the process data.

2.   Read the latest read process data from the read process data area.

For the latest message data:

1.   Read bit 3 (RDMSG) in the buffer control register.

     –   If it is "0", no new message data has been posted.

     –   If it is "1", there is a new message in the read message data area.

2.   Read the latest message data from the read message data area.

3.   Write "1" to bit 3 (RDMSG) in the buffer control register.

5.2   Anybus Event Driven Watchdog

It is possible for the host to establish whether or not the Anybus CompactCom module is working properly by periodically measuring the message response time. If this time exceeds a specified

value, the module can be assumed to be malfunctioning. The host can then enter an application specific safe state, reset the module, or take similar actions.

It is strongly recommended to have at least a rudimentary watchdog mechanism, to be able to restart the module if needed.

5.3 Application Event Driven Watchdog

If desired by the application, an application watchdog timeout can be enabled within the Anybus CompactCom module. When this is enabled, the module will assume that the application is not working properly if the time between two write process data buffer updates exceeds the watchdog timeout selected by the application.

The application watchdog timeout is specified in the Anybus Object, instance attribute #4 (Application watchdog timeout). See *Anybus Object (01h), p. 62*.

6 SPI Host Communication

6.1 General Information

The SPI (Serial Peripheral Interface) is a serial, full duplex protocol. It is a master/slave mode, where the host acts as the master and the Anybus CompactCom module as the slave.

Each byte in the SPI frame is transmitted with the most significant bit first, but the byte order is little endian. The least significant byte is transmitted first. Errors are detected by a 32-bit CRC.

6.2 SPI Frame Format
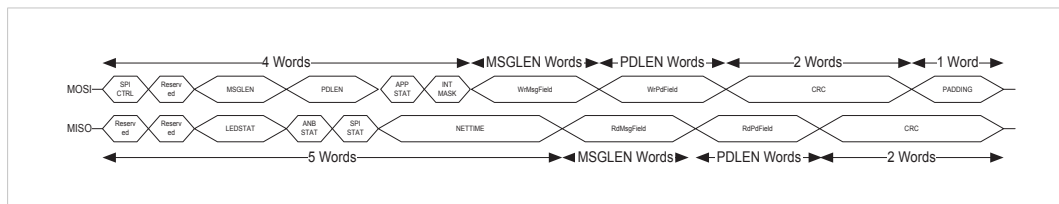
Fig. 10

6.2.1 Data Definitions for the MOSI (Master Output, Slave Input) Frame

SPI MOSI Frame Format

Byte	Name	Description
0	SPI CTRL	SPI control byte, see SPI Control Byte table below.
1	(reserved)	
2 - 3	MSGLEN	The size of the WrMsgField and RdMsgField fields, in words.
4 - 5	PDLEN	The size of the WrPdField and RdPdField fields, in words.
6	APP STAT	Application status, see *Application Status Register, p. 30*.
7	INT MASK	Interrupt mask, see *Interrupt Mask Register, p. 32*.
MSGLEN words	WrMsgField	Message field.
PDLEN words	WrPdField	Write process data field.
2 words	CRC	-
1 word	PADDING	Dummy data.

SPI Control Byte

Bit	Name	Description
0	WRPD VALID	If this bit is set, the Anybus CompactCom 40 will act on the content of the write process data field. If this bit is not set, the module will ignore the content of the write process data field.
1 - 2	CMDCNT	These two bits indicate the number of commands the application is prepared to receive. 00 = The application is not prepared to receive any commands. 01 = The application is prepared to receive at least one command. 10 = The application is prepared to receive at least two commands. 11 = The application is prepared to receive at least three commands.
3	M	If set, the message field contains a message.
4	LAST FRAG	If set, the message field contains the last fragment of a message.
5 - 6	-	Reserved, set to 0
7	TOGGLE	For the initial transmission, this bit shall be set to "1". This bit shall toggle for every SPI transfer. Note: When a CRC error has been detected, this bit shall NOT be toggled to indicate a retransmission.

6.2.2 Data Definitions for the MISO (Master Input, Slave Output) Frame

SPI MOSI Frame Format

Byte	Name	Description
0–1	(reserved)	
2 - 3	LEDSTAT	LED state, see *LED Status Register, p. 30*.
4	ANB STAT	Anybus CompactCom module state, see *Anybus CompactCom Module Status Register, p. 31*
5	SPI STAT	SPI status, see SPI status byte table below.
6–9	NETTIME	These 4 bytes hold the lower 32 bits of the network time.
MSGLEN words	RdMsgField	Message field.
PDLEN words	RdPdField	Read process data field.
2 words	CRC	-

SPI Status Byte

Bit	Name	Description
0	WRMSG FULL	If set, the write message buffer is full. If there was a message in the MOSI frame, it has been ignored by the Anybus CompactCom 40 and must be sent again. Important: The toggle bit must still be toggled, in this case.
1 - 2	CMDCNT	These two bits indicate the number of commands the module is prepared to receive. 00 = The module is not prepared to receive any commands. 01 = The module is prepared to receive at least one command. 10 = The module is prepared to receive at least two commands. 11 = The module is prepared to receive at least three commands. When WRMSG FULL is set, the module has not yet parsed the latest "write message". As a consequence, the CMDCNT may not reflect the last command. Applications that wish to send several consecutive commands without waiting in between must take this into consideration when evaluating the CMDCNT.
3	M	If set, the message field contains a message.
4	LAST FRAG	If set, the message field contains the last fragment of a message.
5	NEW PD	If set, the RDPDFIELD contains data that has been updated from the network since the last SPI transfer. Note that "updated" data does not necessarily mean "changed data". If not set, the RDPDFIELD contains the same data as in the last SPI transfer. Note that even SPI transfers with corrupt CRCs may clear this bit.
6	Reserved	-
7	Reserved	-

6.3 Interrupts

The interrupt mask is sent continuously in every MOSI frame. All interrupts are automatically cleared when a valid MOSI frame has been received by the Anybus CompactCom.

6.4 Message Fragmentation

The SPI protocol supports message fragmentation.

To disable fragmentation, just set the MSGLEN field to a value large enough to fit the maximum size of the messages that the host will send. The M and the LAST FRAG bits shall be set for every message.

To enable fragmentation, set the MSGLEN field to a value smaller than the maximum message size. The M bit shall be set for all SPI frames containing a message or message fragment. The LAST FRAG bit indicates that the current fragment is the last fragment of a message.

6.5   SPI Error Handling

Errors are detected using a 32-bit CRC. The position of the CRC in the MISO and the MOSI frames is shifted. If the Anybus CompactCom 40 detects an error in the MOSI frame, it will send an invalid CRC to the host.

When the host detects a CRC error in the MISO frame, it shall ignore the contents and retransmit the original frame. The retransmitted frame must keep the TOGGLE bit, the M bit, the LAST FRAG bit, as well as the MSGLEN and the MSGFIELD, set to the same values as the original frame. All other fields may contain new values.

The image below depicts a normal scenario. The host sends the SPI frame cyclically, toggling the toggle bit in the SPI control byte each time.
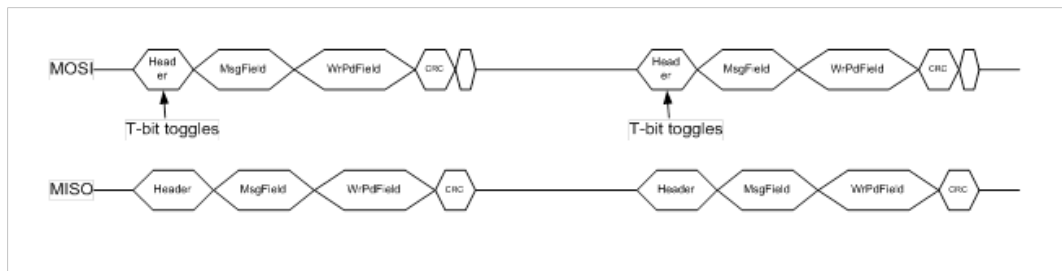
Fig. 11

In case of a reception error on the MISO line, the host will detect this using the MISO CRC and perform a retransmission. Retransmissions are indicated by NOT toggling the toggle bit in the SPI control byte of the MOSI header.

This scenario is depicted in the figure below.
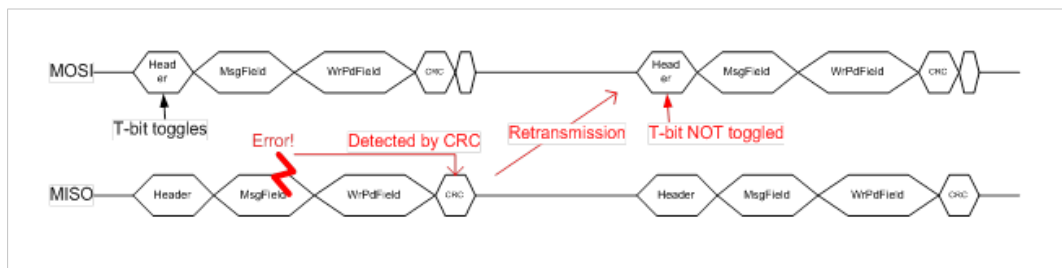
Fig. 12

In case of a reception error on the MOSI line, the Anybus CompactCom will detect this using the MOSI CRC. The Anybus will respond with destroying the MISO CRC, which will result in a retransmission of the SPI frame from the host.
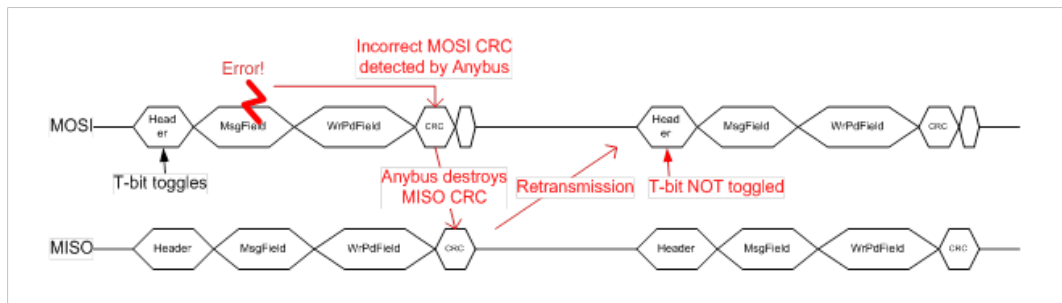
Fig. 13

6.6   Application Event Driven Watchdog

If desired by the application, an application watchdog timeout can be enabled within the Anybus CompactCom 40. When this is enabled, the module will assume that the application is not working properly if the time between two write process data buffer updates exceeds the watchdog timeout selected by the application.

The application watchdog timeout is specified in the Anybus Object, instance attribute #4 (Application watchdog timeout). See *Anybus Object (01h), p. 62*.

7 Shift Register Host Communication

7.1 General Information

The Anybus CompactCom 40 can be used stand-alone, with no host processor. Process data is communicated to shift registers on the host. The Anybus CompactCom 40 supports up to 32 registers in each direction, for a total of 256 bits of data.

> ℹ️ *The PROFIBUS version of the Anybus CompactCom 40 supports up to 24 registers in each direction, for a total of 192 bits of data.*
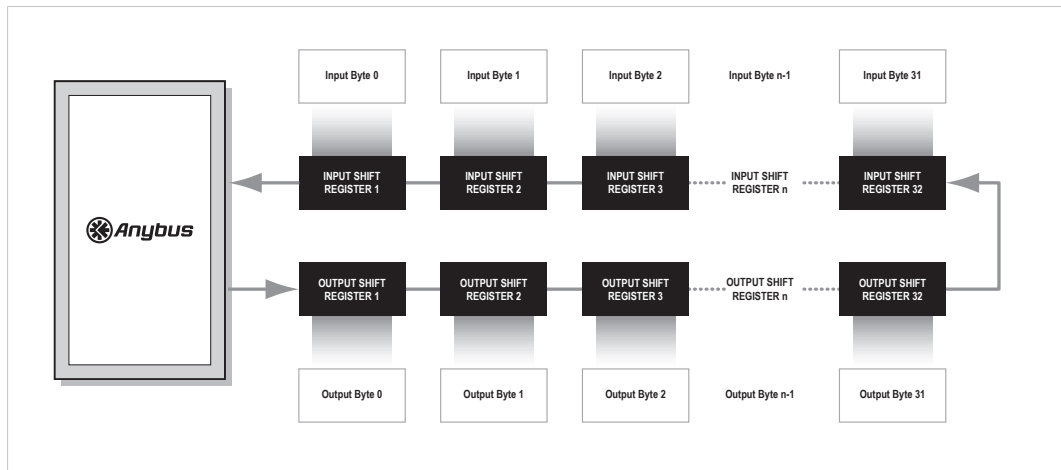
Fig. 14

The Anybus CompactCom 40 will automatically detect the number of connected input and output shift registers. Every shift register will be represented by one UINT8 ADI. The input ADIs will be named "Input 0", "Input 1", etc. The output ADIs will be named "Output 0", "Output 1", etc.

> ℹ️ *The ADI access descriptor values cannot be changed:*
>
> *Input ADIs: 09h (Get access + Write process data mapping possible).*
>
> *Output ADIs: 11h (Get access + Read process data mapping possible)*

The Anybus CompactCom 40 will always try to retrieve network specific attributes from a host application. As this is not possible in stand-alone mode, a virtual attribute list, stored in nonvolatile memory, will be used instead, see *Anybus Object (01h), p. 62*, section "Virtual Attributes". Some attributes are mandatory to implement in order to pass conformance tests, see *Conformance Test Information, p. 151*

7.2 Reset

In stand-alone mode there is no application available to handle a reset request from the network. The reset will be handled by the Anybus CompactCom 40 and the module will reset automatically on a reset request from the network.

8 Serial Host Communication (UART)

8.1 General Information

This mode is supported for backward compatibility with the Anybus CompactCom 30, and should not be used for pure Anybus CompactCom 40 applications.

On the serial host interface, telegrams are transmitted through a common asynchronous serial interface. The baud rate is determined by certain signals on the host interface connector of the module; consult the *Anybus CompactCom 40 Hardware Design Guide* for further information.

For more information on the serial host communication mode, please consult the *Anybus CompactCom 30 Software Design Guide*.

9   The Anybus State Machine

9.1   General Information

A fundamental part of the Anybus CompactCom 40 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. 33*.

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

9.2 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. 59*.
NW_INIT	The Anybus CompactCom module is currently performing network-related initialization tasks. Telegrams now contains Process Data (if such data is mapped), however the network Process Data channel is not yet active.	See *Network Initialization (NW_INIT State), p. 60*.
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 *Anybus Object (01h), p. 62*). When done, reset the Anybus module.

> **!** 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. 81*

10 Object Messaging

10.1 General Information

10.1.1 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.
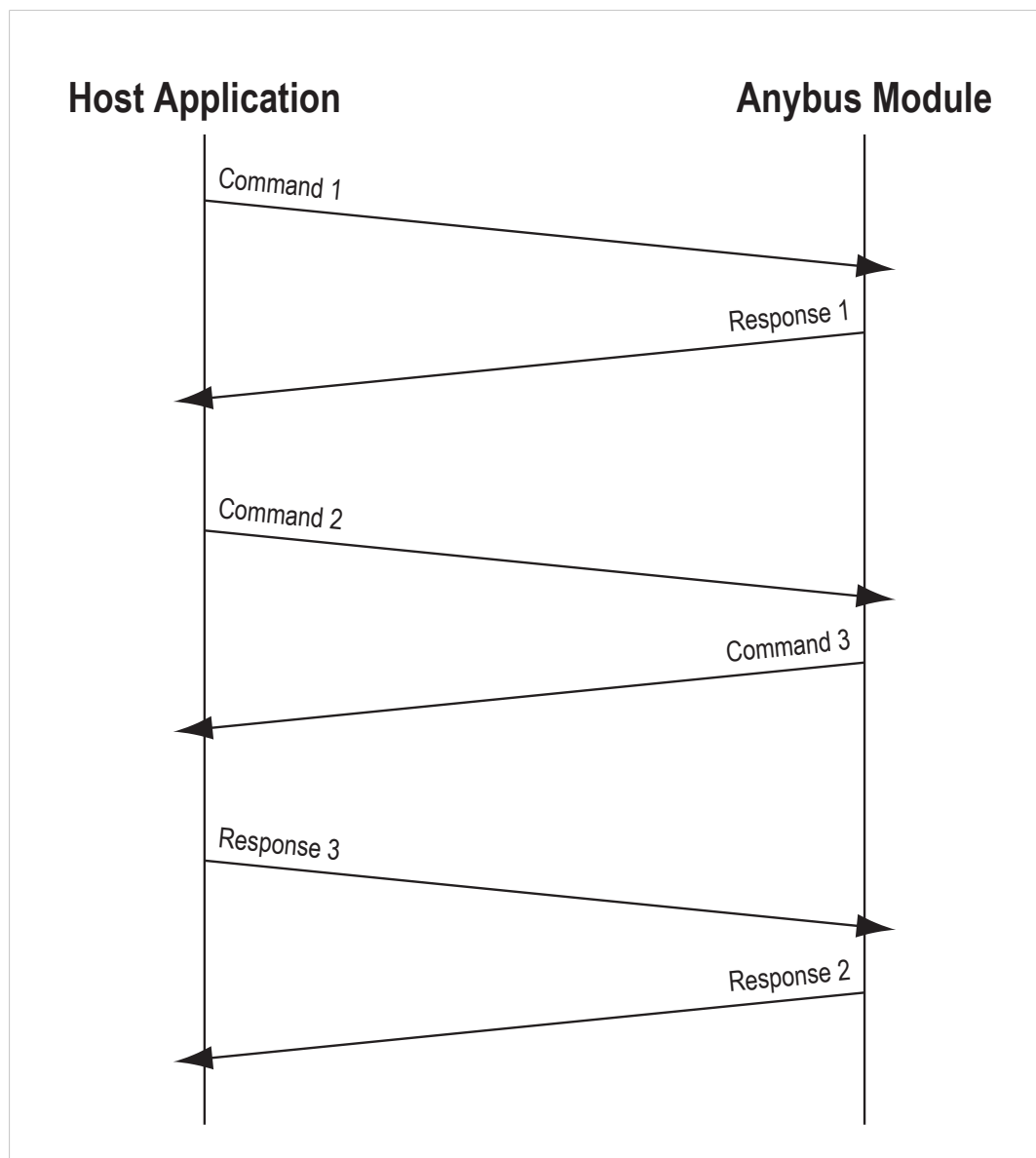
Fig. 16

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.

10.1.2   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.

10.1.3   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.
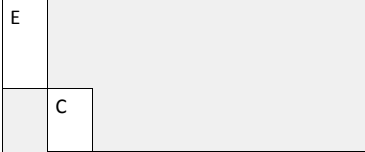
The command initiator must then examine the response to see whether it is a successful response to the command or an error message.

See also...

- *Error Codes, p. 52*

10.2   Message Layout

An object message consists of an 12 byte header followed by message related data.

Offset	Contents								Description
	b7	b6	b5	b4	b3	b2	b1	b0	
0 - 1	Message Data Size								Size of the MsgData[] field in bytes (up to 1524 bytes)
2 - 3	(reserved)								-
4	Source ID								See *Source ID, p. 46*
5	Object								Specifies a source/destination within the Anybus Object model
6	Instance (lsb)								
7	Instance (msb)								
8	E								Value: Meaning: 0: Message is either a Command, or a successful Response 1: Message is an Error Response
		C							Value: Meaning: 0: Message is a Response 1: Message is a Command
			Command Code						See *Command Codes, p. 51*
9	(reserved)								-
10	CmdExt[0]								Command-specific extension. See *Command Specification, p. 51*
11	CmdExt[1]								These fields must be left intact in an error response.
12...n	MsgData[0-n]								Message data field

If the Anybus CompactCom 40 is used in a Anybus CompactCom 30 application, an 8 byte header will have to be used. Please consult the *Anybus CompactCom 30 Software Design Guide* for information.

10.3   Message Segmentation

The maximum message size supported by the Anybus CompactCom 40 is normally 1524 bytes. In some applications a maximum message size of 255 bytes is supported, e.g. if an Anybus CompactCom 40 is to replace an Anybus CompactCom 30 without any changes to the application.

Some objects services must support messages larger than 255 bytes. In order to support this, the Anybus CompactCom 40 supports a fragmentation protocol. To avoid confusion with the fragmentation protocol used for serial telegrams, this protocol is called a segmentation protocol.

10.3.1   Command Segmentation Procedure

When a message is segmented, the initiator of the message sends the same command header multiple times. For each message, the data field is exchanged for the next segment of data.

Command Details

	Command Segment Bits	Description
CmdExt[1]	Bit 0: Bit 1: Bit 2: Bit 3-7:	FS (first segment) LS (last segment) AB (abort) Reserved (0)

Response Details

	Response Segment Bits	Description
CmdExt[1]	Bit 0-7:	Reserved (0)

Procedure

When sending segmented commands, follow the procedure below:

- For the first element, the FS bit shall be set.

- For the subsequent elements, the FS and LS bits shall be cleared (0).

- For the last element, the LS bit shall be set. (For single frame commands (<= 255 or 1524 bytes, depending on message channel) both the FS and the LS bits shall be set).

The command receiver shall send a response (ACK/NACK) for each segment, indicating if the segment was accepted or not. In case of a NACK, the segment will be discarded. The segmentation will not be terminated, however, so earlier accepted segments remain in the segmentation buffer.

The response (ACK/NACK) to the last segment contains the actual result of the operation.

The command initiator may at any time abort the operation by sending a message with the AB bit set. This shall result in that the segmentation buffer is flushed.

To determine if a command is the same as a previous one, the following shall be checked:

- Destination object

- Instance number

- Command number

- Command extension 0 (CmdExt[0])

10.3.2   Response Segmentation Procedure

When a response message is segmented, the initiator of the message requests the next segment by sending the same command multiple times. For each message, the data field is exchanged for the next segment of data.

Command Details

	Command Segment Bits	Description
CmdExt[1]	Bit 0: Bit 1: Bit 2: Bit 3-7:	Reserved (0) Reserved (0) AB (abort) Reserved (0)

Response Details

	Response Segment Bits	Description
CmdExt[1]	Bit 0: Bit 1: Bit 2-7:	FS (first segment) LS (last segment) Reserved (0)

Procedure

When sending segmented responses, follow the procedure below:

•   For the first element, the FS bit shall be set.

•   For the subsequent elements, the FS and LS bits shall be cleared (0).

•   For the last element, the LS bit shall be set. (For single frame commands (<= 255 or 1524 bytes, depending on message channel) both the FS and the LS bits shall be set).

If the LS bit is not set in a response, the command initiator requests the next segment by sending the same command again.

The command initiator may at any time abort the operation by sending a request/response with the AB bit set. This shall result in that the segmentation buffer is flushed.

To determine if a command is the same as a previous one, the following shall be checked:

•   Destination object

•   Instance number

•   Command number

•   Command extension 0 (CmdExt[0])

10.4   Data Format

10.4.1   Available Data Types

The Anybus CompactCom 40 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		0... +255	Yes	No
8	ENUM	8		0... +255	Yes	Yes
9	BITS8	8	8 bit bit field	00000000... 11111111	Yes	Yes
10	BITS16	16	16 bit bit field	0000000000000000... 1111111111111111	Yes	Yes
11	BITS32	32	32 bit bit field	00000000 00000000 00000000 00000000... 11111111 11111111 11111111 11111111	Yes	Yes
12	OCTET	8	Undefined 8 bit data	0... +255	Yes	No
13–15		(reserved)				
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
19	DOUBLE	64	Floating point (IEC 60599)	±2.2250738585072014 E-308... ±1.7976931348623157 E+308	No	Yes
32–48	PADx	0-16	Bit fields of size 0 - 16 used for padding	N/A	Yes	Yes
64	BOOL1	1	1 bit boolean	[0...1]	Yes	Yes
65–71	BITx	1-7	Bit fields of size 1-7	[0...1]... [0000000...1111111]	Yes	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.

- The data types BITS8, BITS16, BITS32, OCTET, PADx, BOOL1 and BITx are only supported by Anybus CompactCom 40.

10.4.2   Bit Fields

The bit field types should be used for parameters where each bit, or group of bits, contains individual meaning. Typical examples include control/status words or digital I/O.

Bit field parameters will be translated to network specific data types suitable for digital I/O or control/status words.

BITSx

The BITSx data types (BITS8, BITS16, and BITS32) consist of even data byte sizes and must be byte aligned. They are handled in the same way as other multibyte data types with regard to byte order.

BITx

The BITx data types (BIT1-BIT7) are packed with bits aligned, and may be placed over byte boundaries. The type code (65-71) of BITS1-BITS7 may be divided into the 5 most significant bits as specifier (always 01000) and the 3 least significant bits as bit counter, with valid values from 1 to 7 for the bit counter field.

10.4.3 Handling of Array of Char (Strings)

Readable strings can be represented in ADIs in two different ways. Either as an array of CHAR or as a string variable. The recommended way is to represent readable strings in ADIs as a variable using the attribute "Number of subelements" in the Application Data Object (FEh), i.e. a string variable that consists of one element with several subelements. This section is mainly applicable when using arrays of CHAR in ADIs. Both these types of strings are hereafter named string.

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 'Get_Attribute'- and 'Set_Attribute' commands.

Strings in ADI structures must be padded with NUL bytes up to the maximum length when accessed through Set_Attribute and Get_Attribute. A get or set string with a data field size of zero is valid and indicates an empty string.

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 CompactCom 40 will check consistency between the payload length and the actual buffer space.

See also...

10.4.4 OCTET

The OCTET type is used for undefined data of byte size.

10.4.5 PADx

The PADx types consist of 17 types, from PAD0 to PAD16. PADx variables are packed with bit alignment and might pass any byte boundaries. The value of a PADx variable is irrelevant and might be skipped completely in a network specific way.

The type code (32-48) might be divided into the 3 most significant bits as specifier (always 001) and the 5 least significant bits as bit counter, with valid values from 0 to 16 for the bit counter field.

10.5   Command Specification

10.5.1   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. 61*

- *Host Application Objects, p. 107*

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.

10.5.2   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)

10.5.3   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.

ⓘ *Normally use the first applicable error code from the table below, if more than one error code is relevant.*

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

Code	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 settable	The requested attribute is not settable
09h	Attribute not gettable	The requested attribute is not gettable
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 Use this error code only when 11h or 12h cannot be used
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	Value too high	The written data is too high
12h	Value too low	The written data is too low
13h	Attribute controlled from another channel	Used to NAK writes to "read process data" mapped attributes.
14h	Message channel to small	The message read/write area in use by the host application, does not support a message channel with large enough data field to fit the response data.
15h	General error	An error not matching any of the other existing error codes has occurred.
16h	Protected access	Read or write operation could not be performed because access is currently protected by the host application.
17h	No data available	The data source does not currently have a value available
18h... FEh	(reserved)	-
FFh	Object specific error	The object returned an object specific error code. Additional details may or may not be included in the message data field (MsgData[0... n])

ⓘ *Error codes 11h - 17h are only available for Anybus CompactCom 40 devices.*

10.5.4   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

10.5.5   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)

10.5.6   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)

10.5.7   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.

10.5.8   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)

10.5.9   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.

10.5.10   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

10.5.11   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

11 Initialization and Startup

11.1 General Information

Before network participation, the following steps must be completed:

1. Startup Procedure

   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 *Startup Procedure, p. 57*.

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. 59*.

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. 60*.

> **!** 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.

11.2 Startup Procedure

The startup procedure is slightly different depending on which type of host interface that is used, but will normally be finished within 1.5 s.

1. Enable power.

2. Release reset to the module.

3. Wait for the Anybus CompactCom 40 to respond. Depending on interface, the expected response is different:

Interface	Expected Response
Parallel Host (8/16 bit)	The host application shall wait for the Anybus CompactCom interrupt signal to go active, before starting to communicate.
SPI	After releasing the reset signal to the Anybus CompactCom module, the host application may optionally wait for the Anybus CompactCom interrupt signal to go active, thus indicating that the module is ready, before starting SPI communication. The other option for the host application is to start SPI communication immediately after releasing the Anybus CompactCom reset signal. The host application may see SPI telegrams with CRC errors at first. These telegrams shall be retransmitted according to normal error handling rules for the SPI protocol, see *SPI Error Handling, p. 39*.
Shift Register	The Anybus CompactCom 40 module initializes autonomously after reset is released.
Serial Host	This interface is backwards compatible to the Anybus CompactCom 30-series serial host interface. Please refer to the Anybus CompactCom 30 Software Design Guide for information.

11.2.1 Suggested Startup Procedure when Upgrading from Network

To allow firmware upgrade from network, implement attribute #5 of the Application Object (FFh), instance #1. The module will inform the host application when a new firmware candidate

is available in the candidate area. The host application must store the information in non-volatile memory. See *Application Object (FFh), p. 120*.

If firmware upgrade from network is allowed, the following startup procedure is suggested:

> ❗ Do not reset the Anybus CompactCom module during the startup procedure.

1. Enable power.

2. Release reset to the module.

3. Wait for the Anybus CompactCom 40 to respond. Depending on interface the expected response is different:

Interface	Expected Response
Parallel Host (8/16 bit)	Interrupt
SPI	First SPI telegram without CRC error, or an active interrupt signal
Shift Register	N/A
Serial Host	First serial telegram without CRC error

4. If a new firmware candidate is available, the module will start to reprogram the firmware. This can need up to 1 min. If no candidate firmware is available the boot time will always be less than 1.5 seconds. In case of a firmware update, do not reset the module. If possible, display a message to the end user: "Waiting for Anybus module"....

5. When a response is detected: Start the initialization of theAnybus CompactCom 40 module. Remove any previously displayed message.

If the module does not respond as described, it has not started up correctly. Please contact HMS Industrial Networks at www.anybus.com/support.

> ❗ When the Anybus CompactCom 40 is reset after a firmware download, the application must wait for the installation to finish, before initialization is started. The Anybus CompactCom is protected against problems occurring during download and/or installation and will recover upon restart.
>
> To install the new firmware after download, reset the Anybus CompactCom 40. If the installation of the new firmware is interrupted, e.g. due to power loss, please restart the Anybus CompactCom 40. The installation process will automatically start from the beginning and the new firmware will be installed without any further action.

11.3   Anybus Setup (SETUP State)

This stage involves four distinctive steps:

1.   Gather information about the Anybus Module (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 implementations.

     Certain Anybus implementations may attempt to alter the Process Data map during runtime. For more information, see *Application Data Object (FEh), p. 111*.

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. 43*
- *Anybus Object (01h), p. 62*
- *Network Configuration Object (04h), p. 81*

11.4   Network Initialization (NW_INIT State)

At this stage, the Anybus module 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..

12   Anybus Module Objects

12.1   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. Network specific Anybus module objects are described in the respective network guides. An overview is shown in *Object Overview, p. 149*.

See also..

- *Message Segmentation, p. 46*
- *Error Codes, p. 52*
- *Categorization of Functionality, p. 143*

For detailed information about each object, see...

- *Anybus Object (01h), p. 62*
- *Diagnostic Object (02h), p. 69*
- *Network Object (03h), p. 74*
- *Network Configuration Object (04h), p. 81*
- *Anybus File System Interface Object (0Ah), p. 83*
- *Functional Safety Module Object (11h), p. 98*
- *Time Object (13h), p. 105*

12.2   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.

12.3   Anybus Object (01h)

Category

Basic

Object Description

This object assembles data about the Anybus CompactCom module itself. The data in question does not as such represent the industrial network the module 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)
	Reset (05h)
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 0403h: Standard Anybus CompactCom 40 0404h: Anybus IP (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 <u>not</u> 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. If the timeout value is exceeded, the Anybus CompactCom 40 will enter the state EXCEPTION with exception code 01h.
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 module 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. 59*
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: UINT16 DC UINT16 DR UINT16 SE UINT16 FE	Error counters (stops counting at FFFFh). (The contents of this attribute is only used during application development.) DC: Discarded commands (received with R = 0) DR: Discarded (unexpected) responses SE: Serial reception errors 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 (FFh), p. 120* , 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 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 M40 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: LED3A status (0=OFF, 1=ON) b5: LED3B status (0=OFF, 1=ON) b6: LED4A status (0=OFF, 1=ON) b7: LED4B status (0=OFF, 1=ON)
14	Switch status	Get	struct of UINT8 SW1 UINT8 SW2	Values of DIP switches representing node address and baud rate, connected to DIP pins in the host connector, see Anybus CompactCom M40 Hardware Design Guide or Anybus CompactCom B40 Design Guide. Supported in serial, SPI and shift register mode. in other modes the attribute contains random data. This attribute is only supported on Anybus CompactCom 40.
15	(not used for Anybus CompactCom 40)	-	-	-

16	GPIO configuration	Get/Set	UINT16	Configuration of the host interface GPIO pins.
				Set access is only valid during SETUP state.
				
				Code: Description
				
				0000h, GIP[0..1] are used as general input pins.
				    $\overline{GOP[0..1]}$ are used as general output pins.
				    For the Anybus CompactCom 40 this mode is identical to the Extended LED functionality (0001h) mode
				
				0001h: Extended LED functionality (default):
				    GIP[0..1] are used as network specific, active low LED outputs associated with the left, or single, port.
				    $\overline{GOP[0..1]}$ are used as network specific, active low LED outputs associated with the right port.
				
				0002h: RMII: GIP[0..1], /GOP[0..1], LED1A, LED1B, LED2A, and LED2B are used to create RMII interface against the application. Only valid for modules supporting RMII, Please note that this code is not valid when running in 16 bit parallel mode.
				
				0003h Three-state:
				    GIP[0..1] and $\overline{GOP[0..1]}$ are set to three-state.

#	Name	Access	Data Type	Description
17	Virtual attributes	Get/Set	Array of UINT8	This attribute is used to implement virtual host application attributes in the module, e.g. when using the module stand-alone. This attribute is only supported on Anybus CompactCom 40. Set access is only valid during SETUP state. Maximum size: 1524 bytes. Stored in nonvolatile memory. See "Virtual Attributes" below.
18	Black list/White list	Get/Set	struct of: UINT8 InfoBits UINT8 ListLen UINT16 Prot#1 UINT16 Prot#2 ... UINT16 Prot#n	This attribute is used to implement the black list/white list. See" Black List / White List" below.. This attribute is only supported on Anybus CompactCom 40. Format: InfoBits: bit 0: 0 = black list / 1 = white list bit 1 - 7: reserved ListLen: Length of the list, equal to #n entries (Prot#n) / 0 = List disabled Prot#: The network type identifier
19	Network time	Get	UINT64	The current network time. The format of the network time is specific to each network. 0 = the network does not support network time. **Note**: This attribute is not supported by all networks. Consult the network guides for more information. This attribute is only supported on Anybus CompactCom 40.
20	Firmware custom version	Get	UINT8	This attribute holds a firmware version prefix, indicating a special branch of the firmware.
21	Anybus IP License	Get	UINT8	Information about what license chip detected by Anybus IP. See below for values. Only supported on the Anybus IP platform.

Virtual Attributes

The virtual attributes list is a 1524 bytes array, stored in nonvolatile memory. The attributes are created using the format below:

> Object (8 bit)
>
> Instance (16 bit)
>
> Attribute (8 bit)
>
> Length (16 bit)
>
> Data (Length * 8 bit)

The virtual attributes are accessed via attribute #17 in the Anybus object.

When the Anybus CompactCom 40 tries to retrieve network specific attributes from the host application and the application cannot supply these attributes, an error code is returned. The module will then check for the missing attributes in the virtual attributes list. Please note that the attribute number has to be left intact in the error response, or the requested attribute can not be found in the list.

Using the virtual attributes list, it is possible to provide network specific objects and/or attributes to the module without implementing them in the host application. This may e.g. be useful when an application is to be adapted to new networks, and need to support network specific attributes, that are not available in the original application. Some attributes are mandatory in order to pass conformance tests, see *Conformance Test Information, p. 151*.

If the array data in the virtual attributes list does not fit into a single message, a Get_Attribute request will return the error code "Messaging channel too small" (14h).

If the Anybus CompactCom 40 is used in stand-alone mode, no host application is available and the Anybus CompactCom 40 will check for attributes in the virtual attributes list. The virtual attributes can only be set before the "Setup complete" attribute is set.

> ❗ If you change the module's identity when implementing a stand-alone shift register solution, it is necessary to implement the virtual attributes.

Black List / White List

Using the black list/white list, it is possible for the host to block or accept certain network types.

Bit 0 in the header of the list decides if it is a black list or a white list. If configured as a white list, only the network types in the list will be accepted. If configured as a black list, all network types in the list will be rejected.

A white list makes it possible to accept only a predefined choice of network types.

A black list makes it possible to block already defined network types.

The black list/white list is accessed via attribute #18 in the Anybus object.

Network type	Network
0005h	PROFIBUS DP-V1
0025h	DeviceNet
0087h	EtherCAT
0089h	PROFINET IRT
0090h	CC-Link
0093h	Modbus TCP
009Bh	EtherNet/IP
009Dh	PROFINET IRT Fiber Optic
009Eh	CC-Link IE Field
009Fh	Ethernet POWERLINK
00A3h	Common Ethernet

Anybus IP license

Code	License	Description
0x00	None	The security chip has not been probed yet.
0x01	Time bomb	The security chip is not mounted or something went wrong when probing or reading from the security chip. Full functionality of Anybus IP is enabled but the module only functional for a limited amount of time.
0x02	Standard	Anybus IP is running with limited functionality.
0x03	Extended	Full functionality enabled.

See also...

Command Details: Reset

Details

Command Code	05h
Valid for:	Object Instance

Description

This command is sent from the host application to the Anybus object (instance 0). A power-on reset is a request to make the module ready for a power-on reset from the host application. A power-on + factory default reset is a request a to return the module to the application specific out-of-box state and then make the module ready for a power-on reset from the host application.

A power-on reset shuts down the network and then sets the module in the EXCEPTION state waiting for the host to perform the power-on reset. Note that this command does not clear or reset any functionality stored in non-volatile memory. No command data shall be supplied together with this reset type of reset command.

A power-on + factory default request shuts down the network, resets the functionality specified by the bit field in the command data, reports the result in the response data and then sets the module in EXCEPTION state waiting for the host application to perform the power-on reset.

If a power-on + factory default reset is successful the response bit field equals the command bit field, meaning that all the targeted functionality was supported and reset successfully.

- Command details:

Field	Contents	
CMDExt[0]	(reserved)	
CMDExt[1]	00h:	Power-on reset (actual power-on or simulated)
	01h:	(reserved)
	02h:	Power-on + Factory default reset
Data[0-3]	Bitmask specifying what to reset to factory default state (UINT32)	
	Bit(s):	Description:
	0:	Network configuration parameters
	1:	Anybus file system
	2:	User created accounts and certificates
	3-23	(reserved)
	24-31	Reserved or network specific

- Response details:

Field	Contents
Data[0-3]	Bitmask specifying what the Anybus CompactCom was supported to reset. See command data for bit specification.

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.

#	Enumeration String	Description
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	ASM communication error	Communication is lost between the Anybus CompactCom module and the attached Anybus safety module.
0Ah	Insufficient application implementation	The application does not implement the functionality required for the Anybus module to continue its operation.
0Bh	Missing serial number	The module is missing a serial number. This might happen in product configurations which does not have an embedded serial number (e.g. Anybus IP), and the application fails to supply one.
0Ch	Corrupt file system	The file system is corrupt and must be formatted by user.
(other)	(reserved)	-

See also...

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

12.4   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 (04)
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.
12	Supported functionality	Get	BITS32	Bit 0: "1" if latching events are supported "0" if latching events are not supported Bit 1 - 31: reserved (shall be "0")

Instance Attributes (Instance #1... N)

#	Name	Access	Type	Description
1	Severity	Get	UINT8	This attribute should be viewed as a bit field Bit 0 (the least significant bit) indicates whether extended diagnostics are used by the instance • Bit 0 = 1: extended diagnostics are used • Bit 0 = 0: extended diagnostics are not used Bits 4 - 6 are used for severity level information. See below.
2	Event Code	Get	UINT8	See below.
3	NW specific extension	Get	Array of UINT8	Network specific event information (optional)
4	Slot	Get	UINT16	Indicates which slot in a modular device the diagnostic event is associated with Default value: 0 For more information, see "Modular Device Object (ECh)" on page 125
5	ADI	Get	UINT16	Indicates which ADI the diagnostic event is associated with Default value: 0 (if the diagnostic instance is not associated with any particular ADI)
6	Element	Get	UINT8	Indicates which element in the ADI the diagnostic event is associated with The value 255 is used if the diagnostic event is associated with the entire ADI Default value: 255
7	Bit	Get	UINT8	Indicates the bit in the element that the diagnostic event is associated with The value 255 is used to indicate that the diagnostic event is associated with the entire element Default value: 255

Severity

This parameter indicates the severity level of the event. Only bits 4 - 6 are used for severity level information.

Severity Levels

Bit Combination	Severity	Comment
000	Minor, recoverable	-
001	Minor, unrecoverable	Unrecoverable events cannot be deleted
010	Major, recoverable	-
011	Major, unrecoverable	Causes a state-shift to EXCEPTION
101	Minor, latching	
110	Major, latching	
(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.

Latching events remain active until explicitly acknowledged by the network master. If the network does not support acknowledgment of latching diagnostic events, the module shall refuse the creation of latching diagnostic events.

When the network master acknowledges one or more latching events, the module shall send a "Reset Diagnostic" request to the application object. The request contains a list of diagnostic instances which the master wishes to acknowledge. The application object shall respond with a list of diagnostic instances which it allows the module to delete. The module will then delete the allowed instances, and report the appropriate information to the network master.

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	Note
CMDExt[0]	Bit 0: Extended Diagnostic Bit 4–6: Severity Other bits Reserved. Set to zero.	
CMDExt[1]	Event Code, see previous page	
MsgData[0...1]	Slot number associated with the event Set to "0" if unknown or unsupported	These fields only exist if bit 0 (Extended Diagnostic) is set
MsgData[2...3]	ADI associated with the event Set to "0" if unknown or unsupported	
MsgData[4]	Element associated with the event Set to "255" if unknown or unsupported	
MsgData[5]	Bit in element associated with the event Set to "255" if unknown or unsupported	
MsgData[6...7]	Reserved. Set to zero	
MsgData[0/8...n]	Network specific extension (optional, definition is network specific)	MsgData[8...n] if bit 0 in CmdExt[0] is set MsgData[0...n] if bit 0 in CmdExt[0] is not set

- Response details (Success):

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

- Response details (Error):

Error	Contents	Comment
Object Specific Error	MsgData[1] = 02h	Error code (Latching event not supported) The event could not be created since the module does not support latching events
	MsgData[1] = FFh	Error code (Network specific error) The event could not be created due to a network specific reason. Information about the event is found in response MsgData[2-n]

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 and latching 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[1] = 01h	Error code (Not removed). The event could not be removed, either because the event itself is unrecoverable, latching, or due to a network specific reason.
	MsgData[1] = FFh	Error code (Network specific error) The event could not be deleted due to a network specific reason Information about the event is found in response MsgData[2-n]

See also:

12.5 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...

- *Functional Safety Object (E8h), p. 109*
- *Application Object (FFh), p. 120*

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)
	Map_ADI_Write_Ext_Area (12h)
	Map_ADI_Read_Ext_Area (13h)

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, Map_ADI_Write_Ext_Area, Remap_ADI_Write_Area or any network specific mapping command.
6	Read Process Data size	Get	-	UINT16	The current read Process Data size (in bytes). Updated on every successful Map_ADI_Read_Area, Map_ADI_Read_Ext_Area, Remap_ADI_Read_Area or any network specific mapping command.
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
0005h	"PROFIBUS DP-V1"
0025h	"DeviceNet"
0087h	"EtherCAT"
0089h	"PROFINET IRT"
0090h	"CC-Link"
0093h	"Modbus TCP"
009Bh	"EtherNet/IP"
009Dh	"PROFINET IRT Fiber Optic"
009Eh	"CC-Link IE Field"
009Fh	"Ethernet POWERLINK"
00A3h	"Common Ethernet"

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.

- It is not allowed to mix mapping commands Map_ADI_Read/Write_Area and Map_ADI_Read/Write_Ext_ Area within one area.

- It is not allowed to map BITSx types of fractional byte size (BIT1 - BIT7) or PADx types using this command.

- It is only allowed to map variables and arrays with this command. It is not allowed to map structures.

- Certain Anybus implementations allow the network to remap the Process Data during runtime. For more information, see *Application Data Object (FEh), p. 111*.

- To map more than 256 bytes, use the commands Map_ADI_Read/Write_Ext_Area.

- The commands Map_ADI_Read/Write_Area should only be used in applications that include one or more Anybus CompactCom 30 devices. For new applications, including only Anybus CompactCom 40 devices, use the commands Map_ADI_Read/Write_Ext_Area.

See also...

- *Application Object (FFh), p. 120*

> **!** Error control is only performed on the command parameters. The Anybus module does <u>not</u> 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. 49*
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 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.

Command Details: Map_ADI_Write_Ext_Area

Details

Command Code:	12h
Valid For:	Instance

Description

This command is only supported by Anybus CompactCom 40 devices.

This command is equivalent to Map_ADI_Write_Area, but can map more than 256 bytes of data. It supports mapping fractional byte size types, and it can be used to map only specific parts of an ADI.

It maps an ADI as Write Process Data. If successful, the response data contains the offset, in bits, for the mapped ADI from the start of the Write Process Data area.

- Mapping 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) is not accepted by all networks.

- It is not allowed to mix mapping commands Map_ADI_Read/Write_Area and Map_ADI_Read/Write_Ext_ Area within one area (Read/Write).

- It is recommended to only map one item for each mapping command during initial development, since data area offset is only given for the first mapping item, and all mapping items may be rejected using one single error code.

- All mapped elements, except those of types BIT1-BIT7 and PADx, must be byte aligned.

- The only implicit padding done is from the very last mapped item up to byte alignment, since the process data needs to be of byte size when the setup is complete.

- Explicit padding is done either through available ADI elements of PADx type, or through the imaginary ADI 0, which is assumed to be an array with 255 elements of type PAD1. Explicit padding of process data is the only correct use of ADI 0. Padding bits might not be visible on the network.

- This command may permanently alter the state of the Anybus CompactCom 40 even though the command is returned with an error. Network specific restrictions may lead to n mapping items to be accepted, but with an error on mapping item n+1. If so, the mappings up to and including n will be accepted, but all other mapping items, starting with n+1, are rejected. The number of accepted mappings is declared in CmdExt[0] of the answer.

- Certain Anybus implementations allow the network to remap the Process Data during runtime. For more information, see *Application Data Object (FEh), p. 111*.

See also...

Application Object (FFh), p. 120

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

- Command details:

Field	Contents
CmdExt[0]	The number of mapping items to add (0-217)
CmdExt[1]	Reserved. Set to 0
MsgData[0-1]	New mapping item 1: ADI number
MsgData[2]	New mapping item 1: Number of elements in the ADI
MsgData[3]	New mapping item 1: Index to the first element to map (0-254)
MsgData[4]	New mapping item 1: Number of consecutive elements to map (1-255)
MsgData[5]	New mapping item 1: Number of type descriptors (1-255)
MsgData[6..n]	New mapping item 1: Array of type specifiers for each mapped element
...	Repeat MsgData[0-n] (as above) for mapping item 2 and onwards.

- Response details (Success):

Field	Contents
CmdExt[0]	The number of accepted mapping items (0-217)
MsgData[0]	Bit offset of the mapped ADI from the start of the Write Process Data (Least significant byte)
MsgData[1]	Bit offset of the mapped ADI from the start of the Write Process Data
MsgData[2]	Bit offset of the mapped ADI from the start of the Write Process Data
MsgData[3]	Bit offset of the mapped ADI from the start of the Write Process Data (Most significant byte)

- Response details (Error):

Error	Contents	
Invalid CmdExt[0]	The number of accepted mapping items, before an error occurred	
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, or too many elements)
	03h: Invalid total size	The requested mapping is denied because the resulting total data size would exceed the maximum permissible (depending on network type)
	06h: Invalid map command sequence	The order in which the commands were received is invalid
	07h: Invalid mapping command	Inconsistencies in the command makes it impossible to parse
	08h: Bad alignment	The alignment rules for process data are not followed
	09h: Invalid use of ADI 0	ADI 0 is an array (255 elements) of type PAD1
	FFh: Network specific restriction	The mapping is denied because of a network specific reason, stated in response Data[2-n]. Consult the relevant network guide

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

Command Details: Map_ADI_Read_Ext_Area

Details

Command Code:	13h
Valid For:	Instance

Description

This command is only supported by Anybus CompactCom 40 devices.

This command is equivalent to Map_ADI_Read_Area, but can map more than 256 bytes of data.

It is identical to Map_ADI_Write_Ext_Area, described above, except that it maps ADIs to Read Process Data.

12.6   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 (when possible).

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

Instance no.	Data type	Parameter
1	Any 8 bit or 16 bit data type	Currently selected network device address (or similar).
2	Any 8 bit or 16 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 module will not respond with an error in case the value remains unaffected.

- When a set request with 8 bits of data is directed to a 16 bit instance, the set request is accepted and the upper 8 bits are set to zero.

- When a set request with 16 bits of data is directed to a 8 bit instance, the set request is accepted and the upper 8 bits are discarded.

Differentiation of Input Devices

The Anybus module 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 module 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 module 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. 49*
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 module 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
6	Configured value	Get	Application specific	Determined by attribute #2	The configured parameter value Returns the configured value of an attribute. It is useful when 'Value' is not being used directly when set, e.g. when a power cycle is needed

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.

12.7   Anybus File System Interface Object (0Ah)

Category

Extended

Object Description

This object provides an interface to the built-in file system. Each instance represents a handle to a file stream and contains services for file system operations. This provides the host application with access to the built-in file system of the module. Instances are created and deleted dynamically during runtime.

The object is structurally almost identical to the Application File System Interface Object (EAh), see *Application File System Interface Object (EAh), p. 132*.

> ⓘ *Ethernet modules have a file system that is accessible to the application for different purposes, e.g. for firmware download/upgrade and internal web pages. See the respective network guides for more information. For all other modules, only one folder is present. This folder is only used for downloading and upgrading firmware.*

Supported Commands

Object:	Get_Attribute (01h)
	Set_Attribute (02h)
	Create (03h)
	Delete (04h)
	FormatDisc (30h)
Instance:	Get_Attribute (01h)
	File Open (10h)
	File Close (11h)
	File Delete (12h)
	File Copy (13h)
	File Rename (14h)
	File Read (15h)
	File Write (16h)
	Directory Open (20h)
	Directory Close (21h)
	Directory Delete (22h)
	Directory Read (23h)
	Directory Create (24h)
	Directory Change (25h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value/Description
1	Name	Get	Array of CHAR	"Anybus File System Interface"
2	Revision	Get	UINT8	02h
3	Number of instances	Get	UINT16	-
4	Highest instance no.	Get	UINT16	-
11	Max no. of instances	Get	UINT16	4: valid for Anybus CompactCom 40 modules supporting IT functionality. 1: valid for Anybus CompactCom 40 modules, not supporting multiple open file/directory streams.
12	Disable virtual file system	Set	BOOL	If the virtual file system is disabled it will not be possible to access the internal web pages. 0 = the virtual file system is enabled (default) 1 = the virtual file system is disabled
13	Total disc size	Get	UINT32	Disc size in bytes.
14	Free disc size	Get	UINT32	Free disc size in bytes.

Instance Attributes (Instance #1... 4)

#	Name	Access	Category	Type	Description
1	Instance Type	Get	Extended	UINT8	Value: Meaning: 0: Reserved 1: File instance 2: Directory instance
2	File size	Get	Extended	UINT32	File size (0 for a directory)
3	Path	Get	Extended	Array of CHAR	The file path to where the instance operates

File System Errors

In case of errors for services calling the file system interface object, the module will return FFh (object specific error). A descriptive file system error will be returned in the error response data field.

#	Name	Description
1	FILE_OPEN_FAILED	Could not open file
2	FILE_CLOSE_FAILED	Could not close file
3	FILE_DELETE_FAILED	Could not delete file
4	DIRECTORY_OPEN_FAILED	Could not open directory
5	DIRECTORY_CLOSE_FAILED	Could not close directory
6	DIRECTORY_CREATE_FAILED	Could not create directory
7	DIRECTORY_DELETE_FAILED	Could not delete directory
8	DIRECTORY_CHANGE_FAILED	Could not change directory
9	FILE_COPY_OPEN_READ_FAILED	Could not open file for copy
10	FILE_COPY_OPEN_WRITE_FAILED	Could not open file for destination
11	FILE_COPY_WRITE_FAILED	Could not write file when copying
12	FILE_RENAME_FAILED	Could not rename file

Command Details: File Open

Details

Command Code:	10h
Valid for:	Instance

Description

Opens a file for reading, writing or appending.

- Command details:

Field	Contents	Comment
CmdExt[0]	00h - Read mode	Opens a file for read only access.
	01h - Write mode	Opens a file for write only access. If the specified file does not exist, it will be created. If the specified file already exists, it will be overwritten.
	02h - Append mode	Opens a file for writing at end-of-file. If the specified file does not exist, it will be created. If the specified file exists, any data written to the file will be appended at end-of-file.
CmdExt[1]	(reserved, 0)	-
MsgData[0...n]	Path + filename of the file to open relative to current path	-

- Response details:

  (No data)

Command Details: File Close

Details

Command Code:	11h
Valid for:	Instance

Description

Closes an open file.

- Command details:

  (No data)

- Response details:

Field	Contents	Comment
CmdExt[0]	Reserved (0)	-
CmdExt[1]	Reserved (0)	-
MsgData[0]	File size (low byte)	The size of the closed file
MsgData[1]	File size	
MsgData[2]	File size	
MsgData[3]	File size (high byte)	

Command Details: File Delete

Details

Command Code:	12h
Valid for:	Instance

Description

Deletes the specified file.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0...n]	Path + filename of the file to delete relative to current path

- Response details:

  (No data)

Command Details: File Copy

Details

Command Code:	13h
Valid for:	Instance

Description

Makes a copy of a file.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Path + filename of the source file, relative to the current path
MsgData[x]	NULL (00h)
MsgData[y]...	Path + filename of the destination file, relative to the current path

- Response details:

  (No data)

Command Details: File Rename

Details

| Command Code: | 14h |
| Valid for: | Instance |

Description

Renames or moves a file.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Old path + filename, relative to the current path
MsgData[x]	NULL (00h)
MsgData[y]...	New path + filename, relative to the current path

- Response details:

  (No data)

Command Details: File Read

Details

| Command Code: | 15h |
| Valid for: | Instance |

Description

Reads data from a file open for reading.

- Command details:

Field	Contents
CmdExt[0]	The number of bytes to read (low byte)
CmdExt[1]	The number of bytes to read (high byte)

- Response details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Data read from the file

Command Details: File Write

Details

| Command Code: | 16h |
| Valid for: | Instance |

Description

Writes data to a file open for writing or appending.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Data to write from the file

- Response details:

Field	Contents
CmdExt[0]	Bytes written (low byte)
CmdExt[1]	Bytes written (high byte)

Command Details: Directory Open

Details

| Command Code: | 20h |
| Valid for: | Instance |

Description

Opens a directory.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Path + name to the directory to open relative to the current path

- Response details:

(No data)

Command Details: Directory Close

Details

Command Code:	21h
Valid for:	Instance

Description

Opens a directory.

- Command details:

   (No data)

- Response details:

   (No data)

Command Details: Directory Delete

Details

Command Code:	22h
Valid for:	Instance

Description

Deletes a directory in the file system. The directory must be empty to be deleted. An attempt to delete a directory that is not empty will result in an error.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Path + name to the directory to delete, relative to the current path

- Response details:

   (No data)

Command Details: Directory Read

Details

Command Code:	23h
Valid for:	Instance

Description

This command reads data from a directory previously opened for reading by the Directory Open command.

For each command sent the next directory entry (file or directory) is returned. When all entries in the directory have been read, the response data size will be set to zero (0) and no message data will be returned, to indicate that no more entries exist in the directory.

- Command details:

    (No data)

- Response details:

Field	Contents
CmdExt[0]	Reserved (0)
CmdExt[1]	Reserved (0)
MsgData[0]	Size of object (low byte)
MsgData[1]	Size of object
MsgData[2]	Size of object
MsgData[3]	Size of object (high byte)
MsgData[4]	Object flags
MsgData[5]...	Object name (file or directory)

- Object Flags

Field	Contents
01h	The object is a directory
02h	The object is read only
04h	The object is hidden
08h	The object is a system object

Command Details: Directory Create

Details

Command Code:	24h
Valid for:	Instance

Description

Creates a directory in the file system.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Path + name to the directory to create, relative to the current path

- Response details:

  (No data)

Command Details: Directory Change

Details

Command Code:	25h
Valid for:	Instance

Description

Change directory/path of the instance.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0]...	Path + name to the directory to change to, relative to the current path

- Response details:

  (No data)

Command Details: Format Disc

Details

Command Code:	30h
Valid for:	Object

Description

Formats a disc in the file system (will erase all data on the disc).

- Command details:

Field	Contents
CmdExt[0]	Disc to format. Set to zero (0)
CmdExt[1]	(reserved, 0)

- Response details:

  (No data)

Examples

In this section are presented examples for a couple of common cases where the end user would use the File System Interface Object.

An imaginary folder structure will be used in the example, with the following files in the root folder:
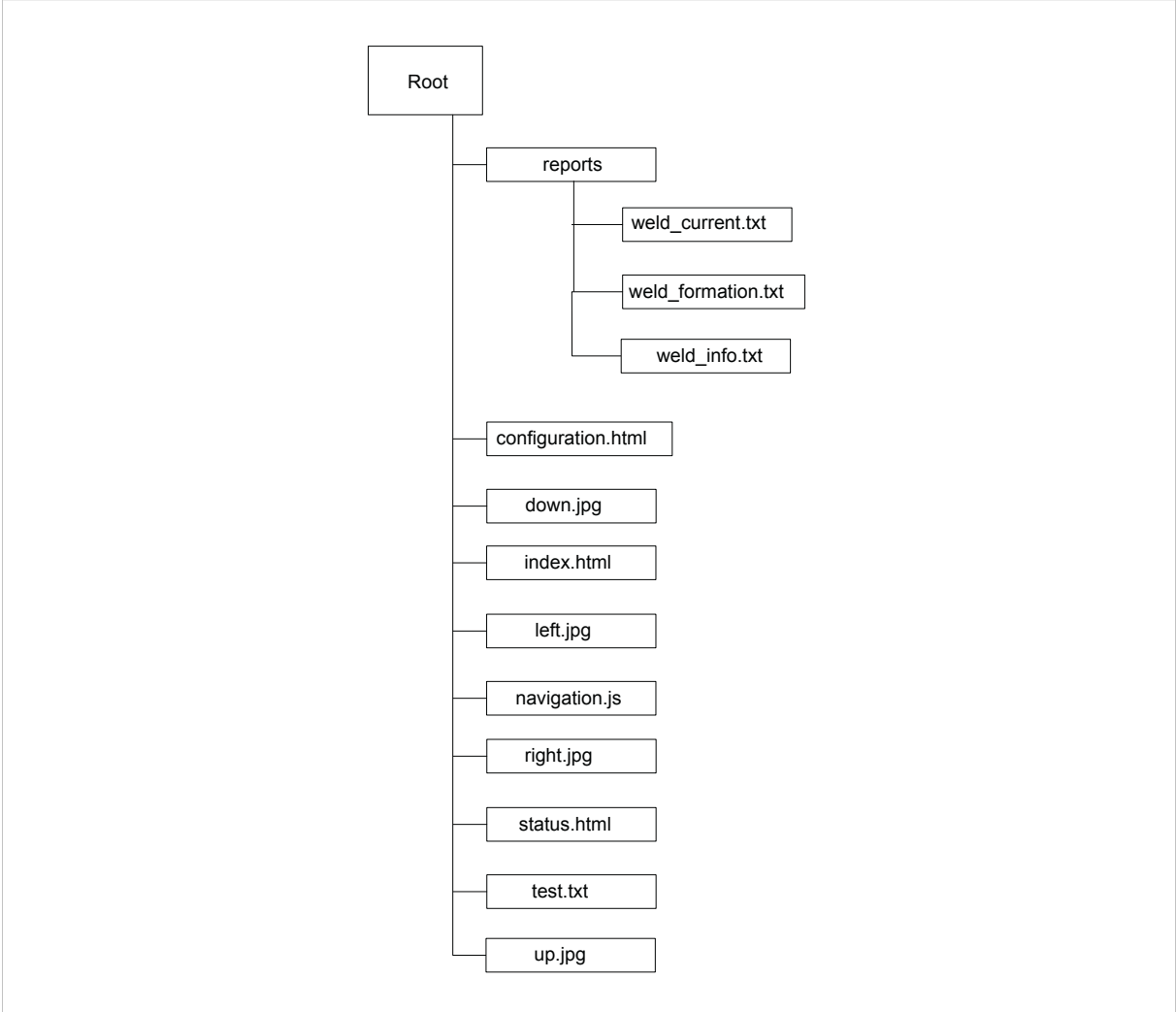
Fig. 17

Read a File

The following example opens weld_info.txt in the reports folder an read data from the file.
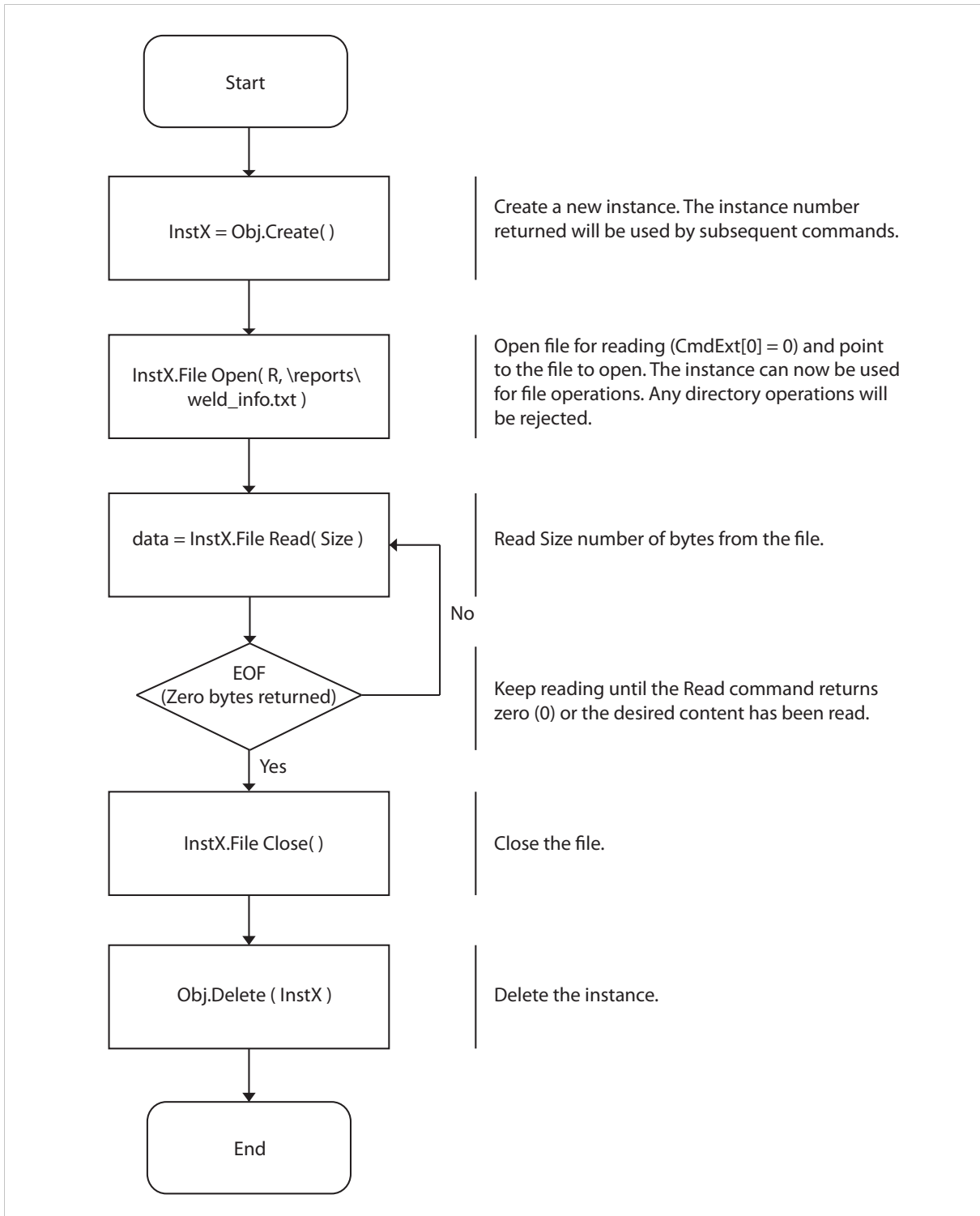
Fig. 18

Write a File

The following example opens up the test.txt file for writing.


```
            Start

InstX = Obj.Create( )          Create a new instance. The instance number
                               returned will be used by subsequent commands.

InstX.File Open( W, \test.txt )  Open file for reading (CmdExt[0] = 1) and point
                                 to the file to open. The instance can now be used
                                 for file operations. Any directory operations will
                                 be rejected.

InstX.File Write( data )         Write the desired data to the file.
                         No

         Done                    Keep writing until the desired content has been
                                 written.
         Yes

InstX.File Close( )              Close the file.

Obj.Delete ( InstX )            Delete the instance.

            End
```

Fig. 19

List Directory Contents

The following example lists the contents of the reports directory.

Start

InstX = Obj.Create( ) — Create a new instance. The instance number returned will be used by subsequent commands.

InstX.Directory Open(\reports\ ) — Open the report directory. The instance can now be used for directory operations. Any file operations will be rejected.

data = InstX.Directory Read( ) — Read the directory entry by entry.

Done — No — Keep reading until all entries have been read. When there are no more entries, this is indicated by a zero data size in the response.

Yes

InstX.Directory Close( ) — Close the file.

Obj.Delete (InstX) — Delete the instance.

End

Fig. 20

12.8   Functional Safety Module Object (11h)

Category

Extended

Object Description

This object contains information provided by the Safety Module connected to the Anybus CompactCom module. Please consult the manual for the Safety Module used, for values of the attributes below.

Supported Commands

Object:	Get_Attribute
	Error_Confirmation
	Set_IO_Config_String
	Get_Safety_Output_PDU
	Get_Safety_Input_PDU
Instance:	Get_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Functional Safety Module"
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	Data Type	Description
1	State	Get	UINT8	Current state of the Safety Module Please consult the manual for the Safety Module used.
2	Vendor ID	Get	UINT16	Identifies vendor of the Safety Module. E.g. 0001h (HMS Industrial Networks) Please consult the manual for the Safety Module used.
3	IO Channel ID	Get	UINT16	Describes the IO Channels that the Safety Module is equipped with. Please consult the manual for the Safety Module used.
4	Firmware version	Get	Struct of UINT8 (Major) UINT8 (Minor) UINT8 (Build)	Safety Module firmware version. Format: version "2.18.3" would be represented as: first byte = 0x02, second byte = 0x12, third byte = 0x03.
5	Serial number	Get	UINT32	32 bit number, assigned to the Safety Module at production. Please consult the manual for the Safety Module used.
6	Output data	Get	Array of UINT8	Current value of the Safety Module output data, i.e. data FROM the network Note: This data is unsafe, since it is provided by the Anybus CompactCom module.
7	Input data	Get	Array of UINT8	Current value of the Safety Module input data, i.e. data sent TO the network. **Note**: This data is unsafe, since it is provided by the Anybus CompactCom module.
8	Error counters	Get	Struct of UINT16 (ABCC DR) UINT16 (ABCC SE) UINT16 (SM DR) UINT16 (SM SE)	Error counters (each counter stops counting at FFFFh) ABCC DR: Responses (unexpected) from the Safety Module, discarded by the Anybus CompactCom module. ABCC SE: Serial reception errors detected by the Anybus CompactCom module. SM DR: Responses (unexpected) from the Anybus CompactCom module, discarded by the Safety Module. SM SE: Serial reception errors detected by the Safety Module.
9	Event log	Get	Array of UINT8	Latest Safety Module event information (if any) is logged to this attribute. Any older event information is erased when a new event is logged. For evaluation by HMS support.
10	Exception information	Get	UINT8	If the Exception Code in the Anybus object is set to "Safety communication error" (09h), additional exception information is presented here, see table below.
11	Bootloader version	Get	Struct of UINT8 Major UINT8 Minor	Safety Module bootloader version. Format: version "2.12" would be represented as: first byte = 0x02, second byte = 0x0C
12	Vendor block safe uc1	Get	Array of UINT8	The Safety Module may supply additional vendor-specific data to the Anybus CompactCom. If such data is available it is presented in this attribute.
13	Vendor block safe uc2	Get	Array of UINT8	The Safety Module may supply additional vendor-specific data to the Anybus CompactCom. If such data is available it is presented in this attribute.

Exception Information

If Exception Code 09h is set in the Anybus object, there is an error regarding the functional safety module in the application. Exception information is presented in instance attribute #10 according to this table:

Value	Exception Information
00h	No information
01h	Baud rate not supported
02h	No start message
03h	Unexpected message length
04h	Unexpected command in response
05h	Unexpected error code
06h	Safety application not found
07h	Invalid safety application CRC
08h	No flash access
09h	Answer from wrong safety processor during boot loader communication
0Ah	Boot loader timeout
0Bh	Network specific parameter error
0Ch	Invalid IO configuration string
0Dh	Response differed between the safety microprocessors (e.g. different module types)
0Eh	Incompatible module (e.g. supported network)
0Fh	Max number of retransmissions performed (e.g. due to CRC errors)
10h	Firmware file error
11h	The cycle time value in attribute #4 in the Functional Safety Host Object can not be used with the current baud rate
12h	Invalid SPDU input size in start-up telegram
13h	Invalid SPDU output size in start-up telegram
14h	Badly formatted input SPDU
15h	Anybus to safety module initialization failure

Command Details: Error_Confirmation

Category

Extended

Details

Command Code	10h
Valid for:	Object

Description

When the Safety Module has entered the Safe State, for any reason, it must receive an error confirmation before it can leave the Safe State. With this command it is possible to reset all safety channels of the safety which, for any reason, are in the Safe State at the same time. The application issues this command to the Anybus CompactCom module, when an error has been cleared by for example an operator. The Anybus CompactCom forwards the command to the Safety Module.

The channel Safe State can also be confirmed by the safety PLC or by the safety module.

With this command

- Command Details

  (no data)

- Response Details

  (no data)

Command Details: Set_IO_Config_String

Category

Extended

Details

Command Code	11h
Valid for:	Object

Description

This command is sent from the host application when there is a need to change the default configuration of the safety inputs and outputs. This string is used by networks where there are no other means (e.g. PLC or some other tool) to provide the configuration to the safety module. Consult the specification of the safety module for more information. The byte string passed is generated by HMS and need to be passed unmodified using this command.

Information about this string is located in the specification of the safety module to which the string shall be sent.

- Command Details

Field	Contents
CmdExt[0]	(not used)
CmdExt[1]	
Data[0... n]	Data (byte string) The data consists of an IO configuration string, where the data format depends on the safety network.

- Response Details

  (no data)

Command Details: Get_Safety_Output_PDU

Category

Extended

Details

Command Code	12h
Valid for:	Object

Description

This command can be issued by the application to get the complete safety output PDU sent by the PLC. The Anybus CompactCom 40 will respond with the complete safety PDU, that the application then has to interpret.

- Command Details

  (no data)

- Response Details

Field	Contents
CmdExt[0]	(not used)
CmdExt[1]	
Data[0... n]	Safety PDU from PLC

Command Details: Get_Safety_Input_PDU

Category

Extended

Details

Command Code	13h
Valid for:	Object

Description

This command can be issued by the application to get the complete safety input PDU sent by the safety module. The Anybus CompactCom 40 will respond with the complete safety PDU, that the application then has to interpret.

- Command Details

  (no data)

- Response Details

Field	Contents
CmdExt[0]	(not used)
CmdExt[1]	
Data[0... n]	Safety PDU from safety module

Object Specific Error Codes

Error Code	Description	Comments
01h	The safety module rejected a message.	Error code sent by safety module is found in MsgData[2] and MsgData[3].
02h	Message response from the safety module has incorrect format (for example, wrong length).	-

12.9   Time Object (13h)

Category

Extended

Object Description

In some networks there are multiple possible time sources. This object is used to present all known time sources using a common format. The quality of the different time sources may vary, which the host application has to consider when using the time value.

Check Time Object support in the applicable Anybus CompactCom 40 Network Guide.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value	Description
1	Name	Get	STRING	"Time Object"	Object name
2	Revision	Get	UINT8	1	Revision of object
3	Number of instances	Get	UINT16	N/A	Supported number of instances
4	Highest instance number	Get	UINT16	N/A	Highest implemented instance
11	Protocols	Get	Array of: Struct of: UINT16 Instance ENUM Protocol UINT8 Reserved	N/A	Array of available time protocols. Instance: Corresponding instance number. Protocol: Enumeration of time protocols. See *Time Protocols, p. 105*. Reserved: Should not be used.

Instance Attributes (Instance #n)

#	Name	Access	Data Type	Description
1	Protocol	Get	ENUM	Enumeration identifier of the time protocol. See *Time Protocols, p. 105* for supported protocols.
2	Current time	Get	UINT64	Current time in protocol specific format. If the time is not valid the value will be set to 0. See *Time Protocols, p. 105* for protocol formats.

Time Protocols

Enum value	Priority	Protocol	Format	Epoch
0	0	CIP Sync (IEEE 1588 PTP)	64 bit nanoseconds	23:59:51. 51.999918, December 31, 1969
1	0	SERCOS III (IEEE 1588 PTP)	32 bit seconds and 32 bit nanoseconds	January 1, 1970
2	0	POWERLINK (IEEE 1588 PTP)	32 bit seconds and 32 bit nanoseconds Bit 31 of nanoseconds represents a sign bit of the total value.	January 1, 1970
3	0	EtherCAT	64 bit nanoseconds	12:00 am, January 1, 2000

Enum value	Priority	Protocol	Format	Epoch
4	0	PROFINET	64 bit tick value (bus cycle counter) Resolution: 31250 nanoseconds	N/A
5	0	CANopen	48 bit value expressed as a struct, 28 bit milliseconds and 16 bit days (plus 4 reserved bits)	The millisecond component is the time in milliseconds after midnight, and the days component is the number of days since January 1, 1984.
6	2	BACnet	Date and time structures. Resolve to 64 bit milliseconds value based on epoch.	January 1, 1970 (epoch defined by HMS)
7	1	NTP	32 bit seconds and 32 bit fractional second	January 1, 1900
8	2	OPC UA Discovery server timestamp	64 bit signed number of 100 nanosecond intervals	January 1, 1601

13 Host Application Objects

13.1 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. Network specific host application objects are described in the respective network guides. An overview is shown in *Object Overview, p. 149*.

See also ...

- *Message Segmentation, p. 46*

- *Anybus Module Objects, p. 61*

- *Categorization of Functionality, p. 143*

For detailed information about each object, see...

- *Functional Safety Object (E8h), p. 109*

- *Application File System Interface Object (EAh), p. 132*

- *Assembly Mapping Object (EBh), p. 135*

- *Modular Device Object (ECh), p. 138*

- *Sync Object (EEh), p. 140*

- *Application Data Object (FEh), p. 111*

- *Application Object (FFh), p. 120*

13.2 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 Object and the Application Data Object are mandatory.

- Support for Network Specific Objects is optional, but recommended. It shall however be noted that the standard functionality provided by the Anybus module 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 40 range and may provide better network integration for certain networks.

See also ...

- *Error Codes, p. 52*

> ! The purpose of the Object Revision attribute is to make it possible for the Anybus module to establish whether or not the object implementation in the host application is compatible with that of the Anybus module, 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.

13.3 Functional Safety Object (E8h)

Category

Extended

Object Description

> ❗ Do not implement this object if a safety module is not used.

This object specifies the safety settings of the application. It is mandatory if Functional Safety is to be supported and a Safety Module is connected to the Anybus CompactCom module.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Functional Safety"
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	Data Type	Default Value	Comment
1	Safety enabled	Get	BOOL	-	When TRUE, enables communication with the Safety Module. **Note**: If functional safety is not supported, this attribute must be set to FALSE.
2	Baud Rate	Get	UINT32	1020 kbit/s	This attribute sets the baud rate of the communication in bits/s between the Anybus CompactCom and the Safety Module. Valid values: • 625 kbit/s • 1000 kbit/s • 1020 kbit/s (default) Any other value set to this attribute, will cause the module to enter the EXCEPTION state. The attribute is optional. If not implemented, the default value will be used. **Note**: The host application shall never implement this attribute when using the IXXAT Safe T100.
3	(reserved)				

#	Name	Access	Data Type	Default Value	Comment
4	Cycle Time	Get	UINT8	-	Communication cycle time between the Anybus and the Safety module in milliseconds. **Note**: The host application shall never implement this attribute when using the IXXAT Safe T100. Valid values: • 2 ms • 4 ms • 8 ms • 16 ms If another value is set in this attribute the Anybus will enter Exception state. Optional attribute; If not implemented the minimum cycle time for the chosen baud rate will be used: • 2 ms for 1020 kbit/s • 2 ms for 1000 kbit/s • 4 ms for 625 kbit/s The Anybus CompactCom validates the cycle time according to the minimum values above. If e.g. baud rate is 625 kbit/s and the cycle time is set to 2 ms the Anybus CompactCom will enter the EXCEPTION state.
5	FW upgrade in progress	Set	BOOL	False	Indicates if the Anybus CompactCom is upgrading the connected Safety module firmware. This means that the Anybus CompactCom will stay in the NW_INIT state longer than normal.

13.4   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.

An Application Data Object instance may be used to model different classes of data: variables, arrays or structures. Every class can be distinguished by the instance attributes, as described in the table below.

Class	Distinguished by	Remarks
Variable	Number of elements is 1.	Starting with revision 3 of the Application Data object, the attribute Number of subelements used in conjunction with a variable of CHAR type is the recommended way to create a string variable. I.e. a string variable here consists of one element with several subelements.
Array	Number of elements is > 1, and number of data types in Data type is 1.	The attribute Number of subelements is not valid for arrays. Arrays of CHAR will be translated to string variables on networks supporting strings. See the remark for Variable above, for the recommended way to represent string variables.
Structure	Number of elements is > 1, and equals the number of Data types.	A structure consists of elements that may have different data types. Possible from object revision 3.

To allow the network and the Anybus module 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 module 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 and Map_ADI_Write_Ext_Area in the *Network Object (03h), p. 74*.

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 appendix.

- If remapping of Process Data is to be supported, implementation is mandatory of object commands Remap_ADI_Write_Area, Remap_ADI_Read_Area and Get_Instance_numbers.

- If remapping of Process Data is supported, object attributes #11 and #12 are mandatory.

- It is recommended to implement the commands Get_Indexed_Attribute and Set_Indexed_Attribute for all attributes that are of data class Array or Structure within the Application Data Object.

Supported Commands

Object:	Get_Attribute (01h)
	Get_Instance_Number_By_Order (10h)
	Remap_ADI_Write_Area (13h)
	Remap_ADI_Read_Area (14h)
	Get_Instance_Numbers (15h)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Get_Enum_String (06h)
	Get_Indexed_Attribute (07h)
	Set_Indexed_Attribute (08h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Application Data"
2	Revision	Get	UINT8	04h
3	Number of instances	Get	UINT16	(depends on application)
4	Highest instance no.	Get	UINT16	
11	No. of read process data mappable instances	Get	UINT16	
12	No. of write process data mappable instances	Get	UINT16	
13	No. of non-volatile instances	Get	UINT16	

Attributes #3, #11, #12 and #13 corresponds to list types 1, 2, 3, and 4 in the command Get_Instance_Numbers.

Attributes #11 and #12 are mandatory for applications supporting remapping of process data.

Attribute #13 is mandatory for applications that supports listing of non-volatile application data instances.

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	Each UINT8 defines the data type of the corresponding element of the instance value for structures and variables. For arrays, one UINT8 defines the data type for all subelements of the corresponding array element.
3	Number of elements	Get	UINT8	Number of elements in attribute #5,. 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	Array of UINT8	Each UINT8 is a bit field specifying the access rights etc. for the corresponding element of the instance value for structures and variables. For arrays, one UINT8 defines the descriptor for all subelements of the corresponding array element. 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. 0: 1: Get Access 1: 1: Set Access 2: -: (reserved, set to zero) 3: 1: Can be mapped as Write Process Data 4: 1: Can be mapped as Read Process Data 5 1: The ADI instance is a non-volatile storage parameter in the application 6 1: Data notification is enabled
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.
9	Number of subelements	Get	Array of UINT16	Each UINT16 defines the number of subelements of the corresponding element of the instance value for structures and variables. Implementation of this attribute is optional , and it must not be implemented for arrays. Data types with bit alignment cannot be used with more than 1 subelement. If this attribute is not implemented, one (1) subelement for each element is assumed.
10	Element name	Get	Struct of Strings (Array of CHAR, separated by NULL byte)	This attribute is used to enable reading the name of each element in an ADI of class Structure. Each string is separated by a NULL byte. There is no NULL byte at the end of the last string. The attribute reflects the element names used on the network. The number of elements in the structure has to be equal to the value of attribute #3 (Number of elements). Commands possible for this attribute are Get_Attribute (response includes strings and separating NULL bytes) and Get_Indexed_ Attribute (the string is returned without any NULL byte). The entire response must fit into the message data field. The largest response accepted is 255 or 1524 bytes, depending on used channel..

- The byte order of attributes #5–8 is network dependent; the Anybus does not perform any byte swapping.

- Unless the data class is a structure 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. For structured ADIs, the format is the same as for attribute #5.

- The instance value(s) must fit entirely into the message data field. The total byte size of all elements must therefore never exceed 255 or 1524 bytes, depending on used channel.

- 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.

Notes on Parameter Access

The following list gives rules and notes on accessing parameters in the Anybus CompactCom 40.

- Structures may not contain elements of type ENUM

- If there is a "hole" in a structured ADI, its data type is defined as PAD0. The Descriptor (attribute #49 should define it as neither settable nor gettable, and "Invalid CmdExt[1]" (07h) shall be returned for the commands Get_Indexed_Attribute/Set_Indexed_Attribute.

- Names of elements are generated by the Anybus CompactCom, unless defined in attribute #10, e.g. "ADIName.0".

- All elements, except those of data type BIT1 - BIT7, BOOL1 and PAD0 - PAD16, must be byte aligned.

- The only implicit padding done for parameter access is from the very last accessed element up to byte alignment, since messages are always complete bytes.

- Explicit padding is done using elements of PADx data type.

- Elements, which are not byte aligned, shall be shifted down to be byte aligned when accessed through Get_Indexed_attribute, and vice versa for Set_Indexed_Attribute.

- Descriptors may differ between elements of the same ADI

  – For a Get_Attribute of an ADI of class Structure with inconsistent settings of the Descriptor bit "Get access" for different elements, the application should fill the unreadable elements with zero in the response. If the "Get access" descriptor bits are consistently set to 0, the Get_Attribute should be returned with error code "Attribute not gettable (09h)".

  – For a Set_Attribute of an ADI of class Structure with inconsistent settings of the Descriptor bit "Set access" for different elements, the application should ignore the non-settable elements and apply the values of the settable ones. If the "Set access" descriptor bits are consistently set to 0, the Set_Attribute should be returned with error code "Attribute not settable" (08h).

- For a Set_Attribute of an ADI of class Structure where at least one element is out of range, compared to the optional Min/Max attributes, the application should ignore all elements and respond with "Out of range". "Out of range" is used so that there will be no conflicts if some elements are too low and others too high, with the added benefit of being able to stop checking after finding one element that is out of range.

- The attribute "Number of elements" must be consistent with the number of elements of the "Data type","Descriptor", and (if it exists) "Number of subelements" attributes.

- ADI elements with a bit aligned data type (PADx, BOOL1, BIT1-BIT7) cannot be used together with attribute "Number of subelements" > 1.

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 the ADI corresponding to the submitted Order Number.

- Response details (Error):

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

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[0–1], 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. 74*

• *Runtime Remapping of Process Data, p. 156*

> **!** 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.

• Command details:

Field	Contents
CmdExt[0–1]	Start of remap (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... 380)
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 *Runtime Remapping of Process Data, p. 156*).

- *Network Object (03h), p. 74*

- *Runtime Remapping of Process Data, p. 156*

> **!** To support this procedure, the host application must be capable of remapping the Process Data during runtime. Support for this command is optional, but highly recommended.

Command Details: Get_Instance_Numbers

Details

Command Code:	15h
Valid for	Object

Description

This command is used to produce lists of ADIs with certain properties. List types 01h - 03h are mandatory for all applications supporting dynamic process data mapping. For a complete list of list types, see "Table of List Types" below.

The application shall respond with a number of instances equal to or less than the requested number (less if a fewer number than requested exists in the application). If the requested starting order number is higher than the highest instance, an empty response shall be returned. If an unsupported list type is requested, an error response with "Invalid CmdExt[1]" shall be generated.

The Anybus CompactCom module may issue several commands with increasing order number to retrieve a complete list.

- Command details:

Field	Contents
CmdExt[0]	Reserved = 00h
CmdExt[1]	List type (See Table of List Types below)
Data[0-1]	Starting order number
Data[2-3]	Requested number of instances

- Response details:

Field	Type	Contents
MsgData[0-1]	UINT16	The instance number of the ADI (with the instance number corresponding to the order number), matching the selected list type
MsgData[2-3]	UINT16	The instance number of the ADI (with the instance number corresponding to the order number + 1), matching the selected list type
MsgData[4-5]	UINT16	The instance number of the ADI (with the instance number corresponding to the order number + 2), matching the selected list type
...	...	...

Table of List Types

List Number	List Type
00h	Reserved
01h	All ADIs
02h	All read process data mappable ADIs (All ADIs where bit 4 in the descriptor attribute is set to "1". See "Instance Attributes (Instance #1... n)" on page 93 for more information.)
03h	All write process data mappable ADIs (All ADIs where bit 3 in the descriptor attribute is set to "1". See "Instance Attributes (Instance #1... n)" on page 93 for more information.)

13.5   Application Object (FFh)

Category

Basic, Extended

Object Description

This object is mandatory, and groups general settings for the host application. The object and its commands makes it possible to support multiple languages, network reset requests and latching diagnostic events.

A control sum, available from the application, that specifies the current parameter settings, can be used to enhance startup time.

Information on if there is a candidate firmware available, and if it is possible to configure the address of the module via hardware switches, can also be read from this object.

Supported Commands

Object:	Get_Attribute (01h)
	Reset (05h)
	Reset_Request (10h)
	Change_Language_Request (11h)
	Reset Diagnostic (12h)
	Get_Data_Notification (13h)
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	02h
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: Enumeration String: 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. 62*, instance #1, attribute #9 • Details for command Change_Language_Request below.
3	Serial number	Get	UINT32	The vendor's serial number for the device If a serial number in the corresponding network specific host object is not available, the module will use this number instead, converted according to network requirements. If neither a network specific attribute nor this attribute is implemented, the module will use its own serial number, see *Anybus Object (01h), p. 62*, attribute #3. Only valid for Anybus CompactCom 40.
4	Parameter control sum	Get	Array of UINT8 (128 bits)	This attribute will hold a control sum from the application that specifies the current parameter settings in the application. How the application calculates the control sum is not specified, the only requirement is that as soon as a parameter in the application changes the control sum also changes. The control sum is used to improve the startup time for the Anybus CompactCom for POWERLINK and PROFINET. Common to these networks are that the master sets a parameter (configuration date and configuration time for POWERLINK and UUID for PROFINET) that specifies the parameter setting the master will transfer to the slave. In order for the slave to determine whether the application already has these parameter settings it must compare the parameter received from the master with the parameter received from the application. If the saved (in non-volatile memory) master parameter and application parameter match the ones newly received from the application and master there is no need for re-parameterization of the application. It's not required by the application to implement this attribute, but it is recommended if quickconnect/fast startup for the above mentioned networks are used. Only valid for Anybus CompactCom 40.

#	Name	Access	Type	Description
5	Candidate firmware available	Get/Set	BOOL	Indicates if there is an firmware file available in the candidate area, for firmware upgrade at the next restart. The application can use this to determine if the next restart will be extended due to a firmware upgrade. The attribute is cleared at startup. Value: Meaning: False: Firmware file not available in the candidate area. True: Firmware file available in the candidate area. See also ... • *Firmware Download, p. 25* • *Startup Procedure, p. 57* Only valid for Anybus CompactCom 40.
6	Hardware configurable address	Get	BOOL	Indicates if the address of the module can be configured via hardware switches. An address may be hardware configurable, but not necessarily hardware configured. Some networks, e.g. EtherNet/IP need to be able to make this distinction. Value: Meaning: False: The address is not hardware configurable. True: The address is hardware configurable. Only valid for Anybus CompactCom 40.
7	Mode	Get	BITS32	Changes the LED indications of the module. Value: Meaning: 00h: Normal LED indication mode (default) 01h: AIDA LED indication mode. Only used for Anybus CompactCom 40 PROFINET, see the network guide for more information. Other: (reserved) Only valid for Anybus CompactCom 40.
8	Vendor name	Get	Array of CHAR	The vendor name. If a vendor name in the corresponding network specific host object is not available, then the device will use this name instead, converted according to the requirements of the network. If neither a network specific attribute nor this attribute is implemented the device will use "HMS Industrial Networks". Maximum number of characters is limited to 64. If a vendor name is to be used for multiple networks, it is recommended to keep it as short as possilbe, as the maximum length may be shorter for some networks, see table below. Implementation of this attribute is optional. Only valid for Anybus CompactCom 40.
9	Product name	Get	Array of CHAR	The vendor's product name for the device. If a product name in the corresponding network specific host object is not available, then the device will use this name instead, converted according to the requirements of the network. If neither a network specific attribute nor this attribute is implemented the device will use the Anybus product name. Maximum number of characters is limited to 64. If a product name is to be used for multiple networks, it is recommended to keep it as short as possilbe, as the maximum length may be shorter for some networks, see table below. Implementation of this attribute is optional. Only valid for Anybus CompactCom 40.

#	Name	Access	Type	Description
10	Firmware version	Get	Struct of UINT8 Major UINT8 Minor UINT8 Build	The vendor's firmware version for the device. Format: version "2.18.3" would be represented as: first byte = 0x02, second byte = 0x12, third byte = 0x03. If a firmware version in the corresponding network specific host object is not available, then the device will use this version instead, converted according to the requirements of the network. If neither a network specific attribute nor this attribute is implemented the device will use the Anybus firmware version, see 5.6.1.3 Anybus object instance attribute 2. Implementation of this attribute is optional. Only valid for Anybus CompactCom 40.
11	Hardware version	Get	UINT16	The vendor's hardware version for the device. If a hardware version in the corresponding network specific host object is not available, then the device will use this version instead, converted according to the requirements of the network. If neither a network specific attribute nor this attribute is implemented the device will use the Anybus Hardware Functionality ID. Implementation of this attribute is optional. Only valid for Anybus CompactCom 40.

Vendor/Product Name Length Network Limitations

Some networks support a lower maximum string length for vendor name and/or product name than specified by this object. If the network limit is lower than the limit specified by this object, the network output will be truncated to the network's limit on all the module's interfaces. The table below lists the limit for every network supported by ABCC40 series.

Network	Vendor Name Max Length	Product Name Max Length
DeviceNet	N/A	32
EtherNet/IP	N/A	32
PROFINET	N/A	25
PROFIBUS	N/A	20
EtherCAT	N/A	64
Ethernet POWERLINK	64	64
BACnet IP	64	64
Modbus-TCP	244	244
CC-Link	N/A	N/A
CC-Link IE Field	31	19

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.

In stand-alone shift register mode, a reset is automatically handled by the module upon a reset request from the network, as there is no application available.

• 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.

In stand-alone shift register mode, a reset request from the network is automatically handled by the module, which will reset automatically.
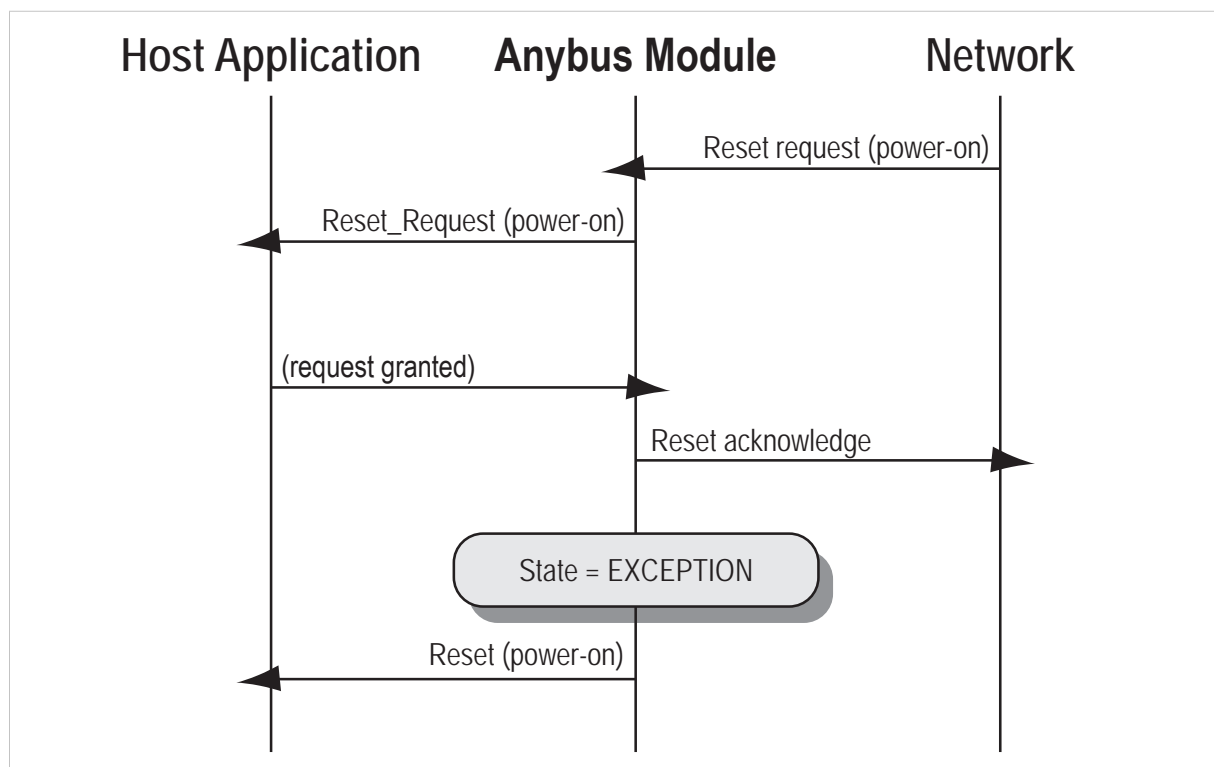
Fig. 21
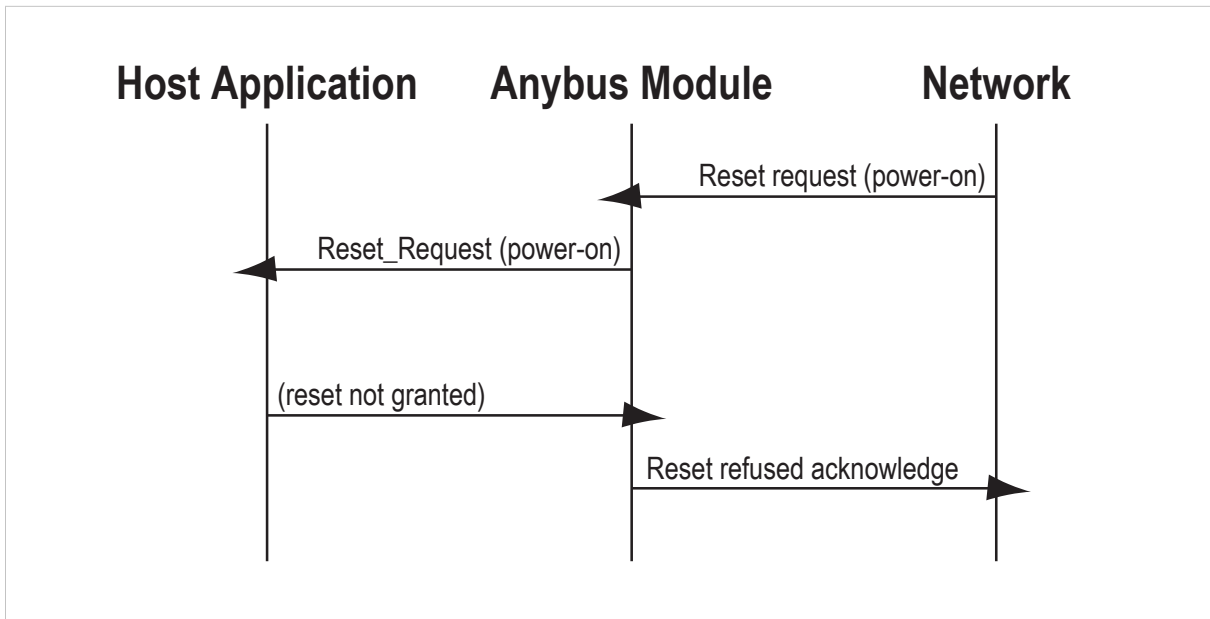
Fig. 22

- 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)

Command Details: Reset_Diagnostic

Details

Command Code:	12h
Valid for:	Object

Description

The Reset_Diagnostic request will be sent to the application object when the network master wishes to acknowledge/reset one or several latching diagnostic events.

This service is only mandatory if the application supports latching diagnostic events.

It is for the application to decide if diagnostic events can be deleted or not. In the Reset_Diagnostic response, the application is expected to provide a list of diagnostic instances that can be deleted (where the error is no longer present). This list may be identical to the list in the Reset_Diagnostic request, or it may be a subset of that list. The application may also respond with a zero sized list, if no instances can be deleted, or with an error in the case that the Reset_Diagnostic request is refused.

See *Diagnostic Object (02h), p. 69* for more information.

- Command details:

Field	Contents
CmdExt[0]	Reserved. Value = 00h
CmdExt[1]	Reserved. Value = 00h
MsgData[0–n]	UINT16 list of diagnostic instances which the Anybus CompactCom module requests permission to delete

- Response details:

Field	Contents
CmdExt[0]	Reserved. Value = 00h
CmdExt[1]	Reserved. Value = 00h
MsgData[0–n]	UINT16 list of diagnostic instances which the Anybus CompactCom module is permitted to delete

Command Details: Get_Data_Notification

Details

Command Code:	13h
Valid for:	Object

Description
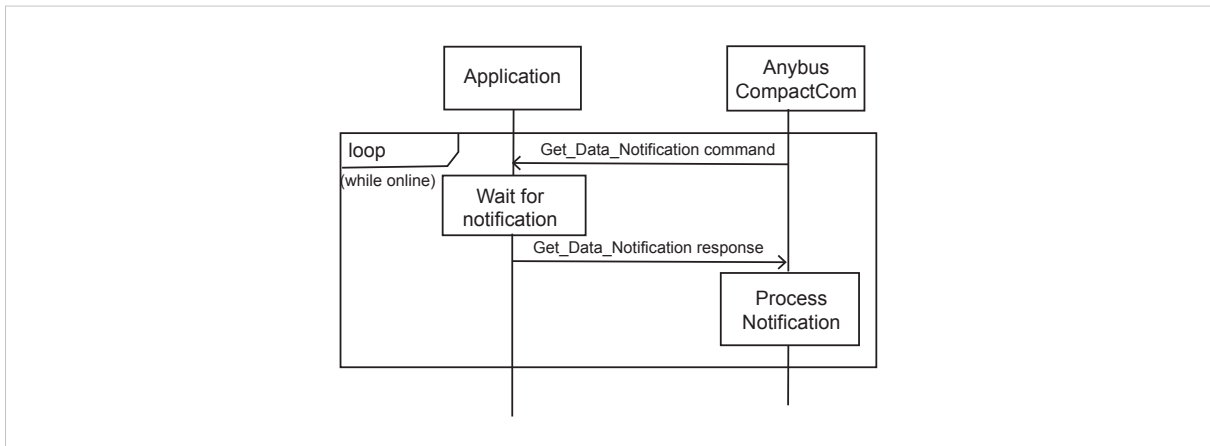
Fig. 23

The Get Data_Notification command is used to send data notifications to the network. It is sent to the application from the Anybus CompactCom, if the Anybus CompactCom implements a network that supports a data notification mechanism. This gives the application a data notification resource, that can be used to send data notifications to the network.

The application can decide to hold the request until an internal notification trigger occurs, e.g. a data change or a time trigger) and then respond to the request. If there is no data to notify about when the request arrives, and the application does not support to hold the request, the application can respond immediately with the error code 17h (No data available). If the application is unable to handle the request due to being out of resources, it may respond with error code 0Eh (Out of resources). In these cases, the Anybus CompactCom will send the request again after a suitable time, depending on the network.

The command enables segmented responses to allow large amounts of data to be transferred. The amount of data that the Anybus CompactCom can handle, depends on the network.

The payload of the Get_Data_Notification response shall contain a dataset, to notify about. A dataset is a predefined set of data that can contain a single value, a group of values or some user defined data. It is network dependent what datasets the Anybus CompactCom supports. A dataset is uniquely identified by combining the dataset type with the dataset identifier.

- Command details:

Field	Contents	Comments
CmdExt[0]	Reserved	Reserved
CmdExt[1]	Response segmentation	See *Message Segmentation, p. 46*
MsgData[0–1]	Capabilities bit field (BITS16)	This bit field equals the header of the NotificationEntry structure (see below). The Anybus CompactCom sets the bits that it supports to receive in the response from the host application.

- Response details:

Field	Contents		Description
CmdExt[0]	Bit 0-3 (Value):	Dataset type:	Bit header that defines and configures the dataset type. The decimal value of bits 0-3 decides which dataset type is used.
	0	Single ADI	
	1	Assembly Mapping instance	
	2	Transparent network payload	
	3–15	Reserved for future dataset types	
	Bit 4-7:	Reserved	
CmdExt[1]	Response segmentation		See *Message Segmentation, p. 46*
MsgData[0–n]	Dataset of the given type		The format of the dataset is decided by the dataset type bit that is set in the CmdExt[0] field.
		Dataset identifier	These fields holds the identifier, the network channels and the NotificationEntry, that are included in a dataset. See below for more information
		Network channels	
		NotificationEntry	
			Header	The NotificationEntry structure is described in more detail below.
			Sub identifier	
			Value	
			Timestamp	

MsgData

Dataset identifier	UINT16	Dataset type	Contents
		Single ADI	An instance number of the Application Data object
		Assembly Mapping instance	An instance number of the Assembly Mapping instance
		Transparent network payload	Identifier of the user defined dataset. The identifier of a transparent network payload is not related to any Anybus object. The host application must ensure that the identifier is unique within the set of defined transparent network payloads.
Network channels	BITS16	The dataset will be forwarded to the channels defined by this bit field	
		Bit 0:	MQTT
		Bit 1-15:	Reserved, shall be set to 0
NotificationEntry		The contents of the NotificationEntry depends on the dataset type defined in CmdExt[0] in the response, see below for details.	

NotificationEntry

A notification of a dataset is represented by a NotificationEntry. A NotificationEntry can represent a value or an event with an identifier, and also include e.g. a timestamp of the value.

Field	Data Type	Contents
Header	BITS16	Bit header that specifies what optional content that will follow this header, and bit notifications. Bits that add an optional field, starting from bit index 0 in ascending order. Optional fields appears in the bit order of this header. Bit: Descriptions: 0: Sub identifier field exists 1: Value field exists 2: Timestamp field exists 3-15: Reserved, set to 0
Sub identifier	UINT16	Second identifier of value, actual content depends on the dataset type in CmdExt[0]
Value	UINT32	Length of the payload in the Value field
	Defined by the value attribute identified by the identifier fields	The value pointed out by the Identifier and Sub identifier fields
Timestamp	UINT64	Timestamp of the value Formatted in nanoseconds since 00:00:00 (UTC), 1January 1970

Depending on dataset type, the available fields and bit notifications in the NotificationEntry structure are either required, optional, or not needed. The mapping of the NotificationEntry fields and bit notifications to each dataset type, is described in the table below.

NotificationEntry element	Single ADI	Assembly Mapping instance	Transparent network payload
Sub identifier	Enabled if the value included is a subset of the ADI elements. THhe sub identifier contains the start element index and the number of consecutive elements included in the NotificationEntry. LSB: Index to the first element included. MSB: Number of consecutive elements included.	Not used	Not used
Value	ADI attribute 5, Value	Assembly Data	User defined payload as an array of OCTET
Timestamp	Optional	Optional	Not used

13.6   Application File System Interface Object (EAh)

Category

Extended

Object Description

This object is used to enlarge the available file system in the Anybus CompactCom.

The file system in the Anybus CompactCom includes a directory named "Application". Commands that are sent via FTP from the PC, addressing this directory, will be routed by the Anybus CompactCom to the application, see the figure below. The application has to handle all events, arriving in this manner, in a correct way.

The object can then be used to create and delete file system interface instances dynamically during runtime. Each instance is a handle to a file stream and contains services for file system operations. The object is mostly similar in structure to the Anybus File System Interface Object (0Ah).
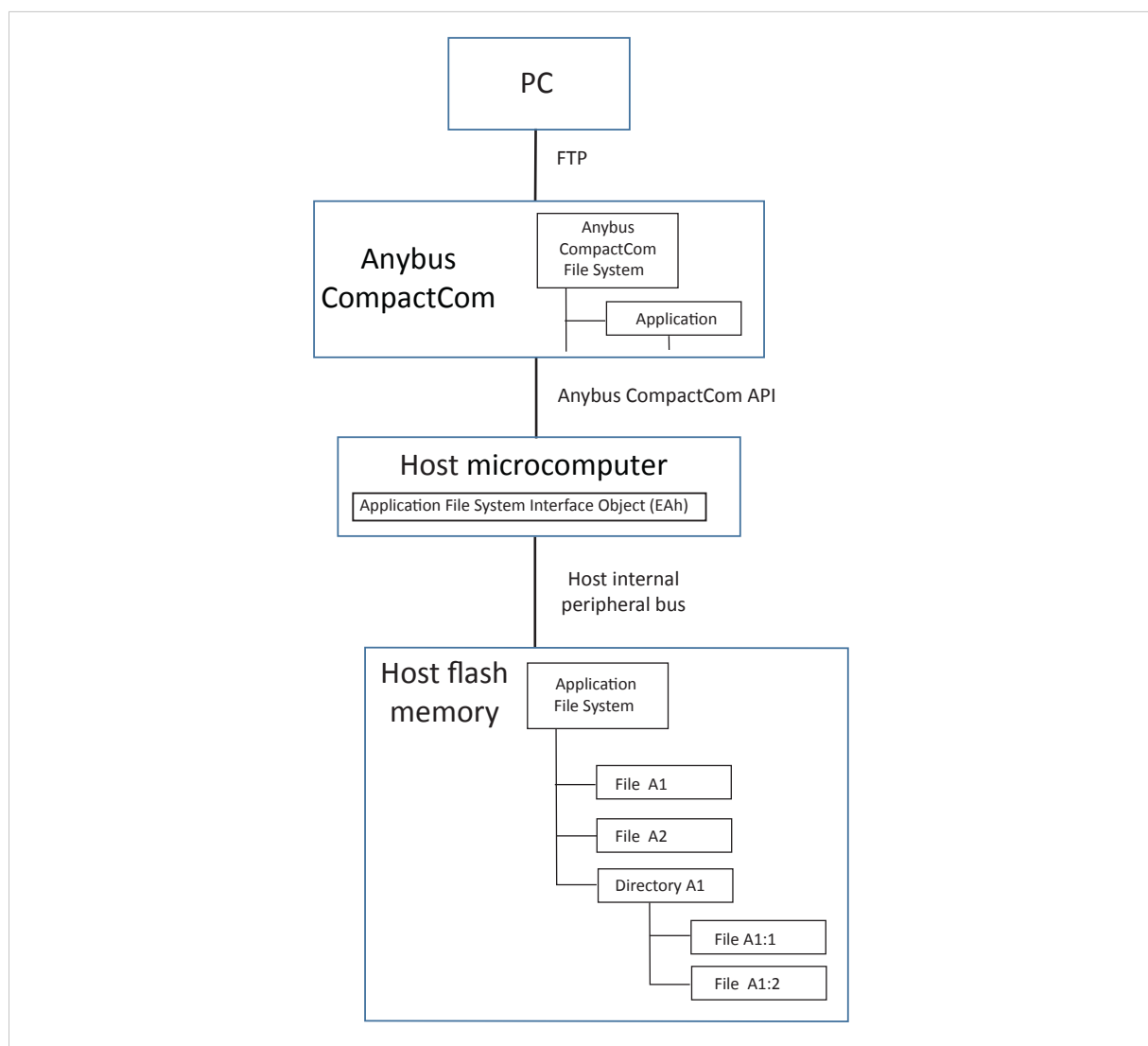
Fig. 24

Supported Commands

For object specific command details, see *Anybus File System Interface Object (0Ah), p. 83*

Object:	Get_Attribute (01h)
	Set_Attribute (02h)
	Create (03h)
	Delete (04h)
Instance:	Get_Attribute (01h)
	File Open (10h)
	File Close (11h)
	File Delete (12h)
	File Copy (13h)
	File Rename (14h)
	File Read (15h)
	File Write (16h)
	Directory Open (20h)
	Directory Close (21h)
	Directory Delete (22h)
	Directory Read (23h)
	Directory Create (24h)
	Directory Change (25h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value/Description
1	Name	Get	Array of CHAR	"Application File System Interface"
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	-
4	Highest instance no.	Get	UINT16	-
11	Max no. of instances	Get	UINT16	20 (recommended)
13	Total disc size	Get	UINT32	Disc size in bytes.
14	Free disc size	Get	UINT32	Free disc sizes in bytes.

Instance Attributes (Instance #1... 20

#	Name	Access	Type	Description	
1	Instance Type	Get	UINT8	Value: 0: 1: 2:	Meaning: Reserved File instance Directory instance
2	File size	Get	UINT32	File size (0 for a directory)	
3	Path	Get	Array of CHAR	The file path to where the instance operates	

File System Errors

In case of errors for services calling the file system interface object, the module will return FFh (object specific error). A descriptive file system error will be returned in the error response data field.

#	Name	Description
1	FILE_OPEN_FAILED	Could not open file
2	FILE_CLOSE_FAILED	Could not close file
3	FILE_DELETE_FAILED	Could not delete file
4	DIRECTORY_OPEN_FAILED	Could not open directory
5	DIRECTORY_CLOSE_FAILED	Could not close directory
6	DIRECTORY_CREATE_FAILED	Could not create directory
7	DIRECTORY_DELETE_FAILED	Could not delete directory
8	DIRECTORY_CHANGE_FAILED	Could not change directory
9	FILE_COPY_OPEN_READ_FAILED	Could not open file for copy
10	FILE_COPY_OPEN_WRITE_FAILED	Could not open file for destination
11	FILE_COPY_WRITE_FAILED	Could not write file when copying
12	FILE_RENAME_FAILED	Could not rename file

13.7   Assembly Mapping Object (EBh)

Category

Extended

Object Description

This object provides support for the possibility to establish I/O connections to different sets of data (assemblies). Assemblies represent, for example, PDOs on EtherCAT or assembly instances on EtherNet/IP. Each assembly is represented by an instance of this object, implemented in the host application.

The sum of the sizes of all write assemblies must not exceed the maximum supported write process data size.

If the application supports the modular device object, all ADIs within one assembly mapping must be in slot order.

If this object is not implemented, the module will provide only one read and one write assembly on the network.

Supported Commands

Object:	Get_Attribute (01h)
Instance:	Get_Attribute (01h)
	Set_Attribute (02h)
	Write_Assembly_Data (10h)
	Read_Assembly_Data (11h)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Assembly mapping"
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	Number of assembly mappings
4	Highest instance no.	Get	UINT16	Highest assembly mapping number
11	Write PD instance list	Get	Array of UINT16	List of currently present instances that can be mapped to write process data
12	Read PD instance list	Get	Array of UINT16	List of currently present instances that can be mapped to read process data

Instance Attributes (Instance #1 ... n)

#	Name	Access	Type (if using 263 bytes message size)	Type (if using 1536 bytes message size)	Description

1	Assembly descriptor	Get	UINT32	UINT32	Bit 0: 0 = Write assembly
					             1 = Read assembly
					
					Bit 1: 0 = ADI Assembly Map is static
					             1 = ADI Assembly Map is dynamic
					
					Bit 2–31 0 = Reserved

2	ADI Assembly Map 0	Get	Array of BITS32 [0-61]	Array of BITS32 [0-379]	Array of ADI items populating this assembly. See "ADI Assembly Map" below
3	ADI Assembly Map 1	Get	Array of BITS32 [62-123]	Array of BITS32 [380-759]	
...	...	...	...	...	
12	ADI Assembly Map 10	Get	Array of BITS32 [620-681]	Array of BITS [3800-4095]	
13	Name	Get	Array of CHAR	Array of CHAR	Assembly instance name string (optional) For EtherCAT, this attribute is used as the object name for the corresponding PDO configuration object.

Set access is supported for attributes #2–12 if dynamic remapping is allowed from the network, i. e. if bit 1 in the assembly descriptor is set to "1".

ADI Assembly Map

The ADIs constituting an assembly are defined in ADI assembly maps. A total of 4096/682 ADIs are allowed for each assembly, for message sizes of 1536/263 respectively. Large ADI assembly maps have to be split up in segments of 380/62 ADI items. Each segment has to be entered as a list in instance attributes #2–#12. Each ADI assembly map attribute must be fully populated with ADI items before using the next attribute.

The ADI item format:

Bits	Description
0-15	ADI number
16-23	Index of first element to map
24-31	Number of consecutive elements to map

Upon a network connection to a read assembly, the Anybus CompactCom module will read all ADI assembly map attributes and generate matching Remap_ADI_Read_Area commands.

Command Details: Write_Assembly_Data

Details

Command Code:	10h
Valid for:	Instance

Description

This command is used to write data to all ADIs within a write assembly mapping.

- Command details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0-n]	Assembly data

- Response details:

    The application can accept the write request, or send an error response.

    – If the written assembly contains an ADI currently mapped to the process data channel, and the Anybus CompactCom module state is PROCESS_ACTIVE, then it is recommended to NAK the request and send error response: "Attribute controlled from another channel".

    – Requests where the assembly data size is incorrect shall generate an error response with error "Not enough data", "Too much data" or "Segmentation data overflow".

Command Details: Read_Assembly_Data

Details

Command Code:	11h
Valid for:	Instance

Description

This command is used to read data from all ADIs within a read assembly mapping.

- Command details:

    -

- Response details:

Field	Contents
CmdExt[0]	(reserved, 0)
CmdExt[1]	(reserved, 0)
MsgData[0-n]	Assembly data

13.8   Modular Device Object (ECh)

Category

Extended

Object Description

This object is used to describe a modular device. Modular devices consist of a backplane with a number of slots. The first slot is occupied by the coupler which contains the Anybus CompactCom module. All other slots may be empty or occupied by modules.

Each instance of this object represents a slot in the modular device. Instance #1 corresponds to the coupler. Instances #2 and onwards correspond to slots in the backplane, occupied as well as empty. There are no instance attributes, but the command Get_List returns a list of the module IDs that are found in the backplane.

When mapping ADIs to process data, the application shall map the process data of each module in slot order. This means that process data mapping commands for ADIs that correspond to the coupler must occur before mapping commands for ADIs that correspond to the module next to the coupler. If the application maps the process data in any other order, the Anybus CompactCom module will enter EXCEPTION state.

> **!** The implementation of modular device functionality differs between networks. Please consult the respective Network Guides for more information.

Supported Commands

Object:	Get_Attribute (01h)
	Get_List (15h)
Instance:	-

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Modular device"
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	Number of physically connected modules in the backplane, including the coupler.
4	Highest instance no.	Get	UINT16	Instance number of the last occupied slot, i.e. the highest instance number currently used.
11	Number of slot	Get	UINT16	Number of available slots in the backplane, including the coupler On most networks this attribute must not be set to a value higher than 256
12	Number of ADIs per slot	Get	UINT16	Used to determine which ADI belongs to which slot, according to the following formula: ADI = slot * x + index + 1 slot = (ADI - 1) / x index = (ADI - 1) MOD x (x equals the value of THIS attribute) For compatibility with EtherCAT, the value of this attribute multiplied with the number of slots (attribute #11 above) must not exceed 4096

Command Details: Get_List

Details

Command Code:	15h
Valid for:	Object

Description

This command shall return a list of module type numbers representing the modules and empty slots in the backplane. For supported list types, see below. List type 01h is mandatory to implement.

The application shall respond with a number of module type IDs including the coupler and empty slots. The size of the array must be equal to or less than the requested number (less if a fewer number of instances than requested exists in the application). The module type ID is selected by the implementor. It is a unique number for each type of module in the backplane. The only value that is specified in advance is "empty slot" , which is 00000000h. If the requested starting order number is higher than the highest instance, an empty response shall be returned. If an unsupported list type is requested, an error response with "Invalid CmdExt[1]" shall be generated.

The Anybus CompactCom may issue several commands with increasing starting instance number to retrieve a complete list.

- Command details:

Field	Contents
CmdExt[0]	Reserved (0)
CmdExt[1]	List type, see below
MsgData[0–1]	Starting instance number
MsgData[2–3]	Requested number of instances

- Response details:

Field	Type	Contents
MsgData[0-3]	UINT32	Module type ID of starting instance.
MsgData[4-7]	UINT32	Module type ID of starting instance + 1
…	…	…

List Types

List Number	List Type
00h	Reserved
01h	List of all module type IDs

13.9   Sync Object (EEh)

Category

Extended

Object Description

This object contains the host application SYNC settings. For more information about how to use SYNC in applications, see

• *Application Status Register, p. 30*

• *SYNC, p. 19*

Supported Commands

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

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	"Sync"
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	Data Type	Value
1	Cycle time	Get/Set	UINT32	Application cycle time in nanoseconds
2	Output valid	Get/Set	UINT32	Output valid point relative to SYNC events, in nanoseconds Default value: 0
3	Input capture	Get/Set	UINT32	Input capture point relative to SYNC events, in nanoseconds Default value: 0
4	Output processing	Get	UINT32	The network master interprets this as the minimum required time, in nanoseconds, between RDPDI interrupt and "Output valid". The host application interprets this as the maximum time available for processing the output data.
5	Input processing	Get	UINT32	Maximum required time, in nanoseconds, from "Input capture" until write process data has been completely written to the Anybus CompactCom module
6	Min cycle time	Get	UINT32	Minimum cycle time supported by the application in nanoseconds
7	Sync mode	Get/Set	UINT16	This attribute is used to select synchronization mode. It enumerates the bits in attribute #8 0: Non synchronous operation. (Default value if non synchronous operation is supported) 1: Synchronous operation 2 - 65535: Reserved. Any attempt to set sync mode to an unsupported value shall generate an error response
8	Supported sync modes	Get	UINT16	A list of the synchronization modes the application supports. Each bit corresponds to a mode in attribute 7 Bit 0: 1 = Non synchronous mode supported Bit 1: 1 = Synchronous mode supported Bit 2 - 15: Reserved (0)
9	Control task cycle factor	Set	UINT16	If the synchronous task operates at a cycle that is longer than the data cycle (see attribute#1, Cycle time), this attribute provides a scaling factor for the cycle time such that: *Contral task duration = Control task cycle factor x Cycle time* The information may be used to e.g. interpolate output values, if required by the process. For PROFINET this is equivalent to PRFOINET Controller Cycle Factor (CACF) Note that synchronization to the application cycle is left to the application to perform. Default: 1

13.10   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 40 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	CC-Link IE Field	PROFIBUS DP-V1	PROFINET IRT	DeviceNet	Modbus-TCP	EtherCAT	Ethernet POWERLINK	BACnet/IP	CANopen
Network Data Format	LSB first	LSB first	LSB first	MSB first	MSB first	LSB first	LSB first	LSB first	LSB first	MSB first	LSB first
Acyclic Data Support	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Max. no. of Elements Per ADI (Acyclic access)	255	N/A	255	240	255	255	32 (255)	255	254	1	254 (high)
Max. ADI Size (in bytes) (Acyclic access)	1524	N/A	1524	240	1308	512	32 (1524)	1524	1524	4	1524
Lowest Addressable ADI no.	1	1	1	1	1	1	1	1	1	1	1
Highest Addressable ADI no.	65535	65535	65535	65025	32767	65535	3839 (61424)	57343 (16383)	57343	65535	76343
Max. Write Process Data (in bytes)	1448	368	1536	244	1308	512	1536	1486	1490	N/A	512
Min. Write Process Data (in bytes)	0	0	0	0	0	0	0	0	0	N/A	0
Max. Read Process Data (in bytes)	1448	368	1536	244	1308	512	1536	1486	1490	N/A	512
Min. Read Process Data (in bytes)	0	0	0	0	0	0	0	0	0	N/A	0
Max. Process Data (Read + Write, in bytes)	2896	736	3072	488	2616	1024	3072	2972	2980	N/A	1024
Min. Process Data (Read + Write, in bytes)	0	0	0	1	0	0	0	0	0	N/A	0
Requires "Get/Set_ Indexed_Attribute"	No	No	No	No	No	No	No	Yes	Yes	No	Yes
Requires "Get_ Instance_Number_ By_Order"	Yes	No	No	No	Yes	Yes	No	Yes	No	Yes	No
Runtime Remapping	Yes	No	No	Yes	Yes	No	No	Yes	Yes	No	Yes
Max. no. of Diagnostic Instances	6	6	2	6	6	6	6	6	1	1	6
Supports Network Reset Type 0: "Power-on-reset"	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes
Supports Network Reset Type 1: "Factory default reset"	Yes	No	No	No	Yes	Yes	No	Yes	No	No	Yes
Supports SINT64	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Supports UINT64	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Supports FLOAT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Cycle time	1 ms	~1 ms	200 µs - 200 ms	-	250 µs	10 ms	-	100 µs	200 µs - 2147483 µs	N/A	N/A

B.1   **Network Specific Comments**

- EtherNet/IP (EIP):

    - The command Get_Instance_Number_By_Order is needed when accessing attributes in the Parameter Object from a CIP network.

    - The command Get_Instance_Number_By_Order is also used for modules, that support internal web pages, when the parameter web page is opened.

- CC-Link (CCL):

    - The max. no of elements per ADI and the max. ADI size depend on ADI mapping, please consult the Anybus CompactCom 40 CC-Link Network Guide

    - The cycle time is given for transmission speed 10 Mbps and only one Remote device occupying one station on the network. Depends on network configuration.

- 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. Lowest addressable ADI no. would in that case be 257.

- DeviceNet (DEV):

    - The command Get_Instance_Number_By_Order is needed when accessing attributes in the Parameter Object from a CIP network.

- Modbus-TCP (EIT):

    - The Highest addressable ADI no. default value is 3839.

    - If changing Number of ADI indexing bits and limiting ADI size, the highest addressable ADI is as high as 61424.

- EtherCAT (ECT):

    - The Highest addressable ADI when the module is in generic mode is 57343.

    - The Highest addressable ADI when the modular device profile is enabled is 16383.

    - The command Get_Instance_Number_By_Order (or alternatively, Get_Instance_ Numbers) is used during initialization to find number of ADIs

    - Network Reset Type 0 is supported for firmware upgrade purposes.

- Ethernet POWERLINK (EPL):

    s

    - The network puts no limit to max. ADI Size.. The present implementation allows 30 kB / ADI.

    - The command Get_Instance_Number_by_Order is used for XDD file generation.

- BACnet/IP:

    - Read Process Data not supported

    - When a BACnet product isn't in advanced mode, the command Get_Instance_Number_ By_Order is used to perform initial mapping.

- CANopen:

    - It is not guaranteed that the entire read process data buffer has been updated when sent to the host application, as PDOs are sent from the master or other producing nodes at different times.

C   Industrial Ethernet Network Comparison

The Anybus CompactCom 40 series product family support a number of Industrial Ethernet networks. In the product family there is also a Common Ethernet module, offering an Ethernet platform which can be used as is or to which you can download the Ethernet firmware of your choice.

The table below shows Ethernet features that are available for the different networks.

> **!** For features, specific to a particular network, and for in-depth information about a particular network, consult the corresponding network guide.
>
> It is recommended to contact technical support at HMS Industrial Networks prior to submitting a device for conformance testing.

Item	Ethernet/IP	PROFINET IRT	Modbus-TCP	EtherCAT	Ethernet POWERLINK	Common Ethernet	CC-Link IE Field	BACnet/IP
Application interfaces	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)	8/16-bit DPRAM (30 ns) SPI (max. 20 Mbit/s) Shift register (12.5 MHz)
Network cycle time	≥ 1 ms	≥ 250 µs	N/A	≥ 100 µs	≥ 200 µs	N/A	≥ 230 µs	N/A
Latency	160 µs with 32 bytes of process data	tbd	N/A	< 250 ns	< 15 µs	N/A	< 2µs	N/A
Jitter	40 µs with 32 bytes of process data	tbd	N/A	< 200 ns	200 ns	N/A	N/A	N/A
Max. process data (in/out, byte)	1448 / 1448	1440 / 1440 including status byte	1536 / 1536	1486 / 1486	1490 / 1490	-	1536 / 1536	N/A
Max. parameter data (in/out, byte)	1448	1512 / 1294	248 / 248	1524 /1524	1490 / 1490	-	960 / 960	4 / 4
IT protocol support	TCP, UDP, FTP, HTTP, SMTP	TCP, UDP, FTP, HTTP, SMTP	TCP, UDP, FTP, HTTP, SMTP	TCP, UDP, FTP, HTTP, SMTP	TCP, UDP, FTP, HTTP, SMTP	TCP, UDP, FTP, HTTP, SMTP	No	TCP, UDP, FTP, HTTP, SMTP
Web server included	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Transparent Ethernet (RMII, separate product version)	Yes	Yes	Yes	Yes	No	No	No	No
Network accessible FLASH disk	Yes, 28 Mbyte	Yes, 28 Mbyte	Yes, 28 Mbyte	Yes, 28 Mbyte	Yes, 28 Mbyte	Yes, 28 Mbyte	No	Yes, 28 Mbyte
Socket communication support	Max 20 connections at the same time	Max 20 connections at the same time	Max 20 connections at the same time	Max 20 connections at the same time	Max 20 connections at the same time	Max 20 connections at the same time	No	Max 8 connections at the same time

Item	Ethernet/IP	PROFINET IRT	Modbus-TCP	EtherCAT	Ethernet POWERLINK	Common Ethernet	CC-Link IE Field	BACnet/IP
Socket data throughput	max. 70 Mbit/s	20 Mbit/s	-	-	20 Mbit/s	-	-	-
Port disabling supported	Yes	No	Yes	No	N/A	No	No	Yes
Diagnostic capabilities	LED outputs, ABCC Diagnostic Object, EtherNet/IP diagnostic counter support, web site	LED outputs, ABCC Diagnostic Object, web site, SNMP MIB2	LED, ABCC Diagnostic Object, web site	LED outputs, ABCC Diagnostic Object, web site	LED outputs, ABCC Diagnostic Object, diagnostics in ESC	LED outputs, ABCC Diagnostic Object, web site	-	LED outputs, ABCC Diagnostic Object, acyclic "Get Statistics" service	LED, ABCC Diagnostic Object, web site
Node address setting	Configuration object, DIP switch, via network, IPconfig, HTTP	Configuration object, via network, IPconfig, HTTP	Configuration object, DIP switch, IPconfig, HTTP	Configuration object, DIP switch, IPconfig, HTTP	Configuration object, DIP switch, IPconfig, HTTP	Configuration object, DIP switch, IPconfig, HTTP	Configuration object, DIP switch	Configuration object, DIP switch, IPconfig, HTTP
Approvals	UL, cUL	UL, cUL	UL, cUL	UL, cUL	UL, cUL	UL, cUL	UL, cUL	UL, cUL
IT security	HTTP authentication (basic + digest), FTP password	HTTP authentication (basic + digest), FTP password	HTTP authentication (basic + digest), FTP password	HTTP authentication (basic + digest), FTP password	HTTP authentication (basic + digest), FTP password	HTTP authentication (basic + digest), FTP password	N/A (no IT support)	HTTP authentication (basic + digest), FTP password
Safety support	CIP Safety, T100 IO & black channel. planned	PROFIsafe, T100 IO & black channel	No	FSoE, black channel	TBD	No	No	No
Secure Host IP Configuration Protocol (HICP)	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
HMS Firmware Manager supported	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Note: EtherCAT network performance FSoE: Sample application FSoE black channel to safety module.

D Object Overview

Each device in the Anybus CompactCom 40 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.*

D.1 Anybus Module Objects

These objects are implemented in the product.

		EtherNet/IP	CC-Link	CC-Link IE Field	PROFIBUS DP-V1	PROFINET	DeviceNet	Modbus-TCP	EtherCAT	Ethernet POWERLINK	BACnet/IP	Common Ethernet	CANopen
01h	Anybus Object	X	X	X	X	X	X	X	X	X	X	X	X
02h	Diagnostic Object	X	X	X	X	X	X	X	X	X	X	X	X
03h	Network Object	X	X	X	X	X	X	X	X	X	X	X	X
04h	Network Configuration Object	X	X	X	X	X	X	X	X	X	X	X	X
07h	Socket Interface Object	X				X		X	X	X	X	X	
08h	Network CC-Link Object		X										
09h	SMTP Client Object	X				X		X	X	X	X	X	
0Ah	Anybus File System Interface Object	X	X	X	X	X	X	X	X	X	X	X	X
0Ch	Network Ethernet Object	X				X		X	X	X	X	X	
0Dh	CIP Port Configuration Object	X											
0Eh	Network PROFINET IO Object					X							
10h	PROFIBUS DP-V0 Diagnostic Object				X								
11h	Functional Safety Module Object	X				X							
12h	Network CC-Link IE Field Object			X									

ⓘ *All fieldbuses and networks can use the Anybus File System Interface Object (0Ah) for firmware update from the host application.*

For the following fieldbuses and networks, the Anybus File System Interface Object (0Ah) is available only for firmware update from the host application: CC-Link, CC-Link IE Field, PROFIBUS DP-V1, and DeviceNet.

D.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	CC-Link IE Field	PROFIBUS DP-V1	PROFINET	DeviceNet	Modbus-TCP	EtherCAT	Ethernet POWERLINK	BACnet/IP	Common Ethernet	CANopen
E2h	MQTT Host Object	X				X							
E3h	OPC UA Object	X				X							
E4h	Energy Measurement Object					X							
E5h	Asset Management Object					X							
E6h	CC-Link IE Field Network Host Object			X									
E7h	Energy Reporting Object	X					X						
E8h	Functional Safety Object	X				X							
E9h	POWERLINK Object									X			
EAh	Application File System Interface Object	X				X		X	X	X	X	X	
EBh	Assembly Mapping Object	X							X				X
ECh	Modular Device Object	X			X	X	X		X				
EDh	CIP Identity Host Object	X					X						
EEh	Sync Object	X				X			X	X			
EFh	BACnet Host Object										X		
F0h	Energy Control Object	X				X	X						
F5h	EtherCAT Object								X				
F6h	PROFINET IO Object					X							
F7h	CC-Link Host Object		X										
F8h	EtherNet/IP Host Object	X											
F9h	Ethernet Host Object	X		X		X		X	X	X	X	X	
FAh	Modbus Host Object							X					
FBh	CANopen Object												X
FCh	DeviceNet Host Object						X						
FDh	PROFIBUS DP-V1 Object				X								
FEh	Application Data Object	X	X	X	X	X	X	X	X	X	X	X	X
FFh	Application Object	X	X	X	X	X	X	X	X	X	X	X	X

E   Conformance Test Information

Stand-Alone Mode

In order to pass conformance tests in stand-alone shift register mode, the host application has to implement some virtual attributes.

E.1   EtherCAT

Virtual attributes, needed to pass EtherCAT certification test in shift register mode:

E.1.1   Mandatory Implementations

EtherCAT Object (F5h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
1	Vendor ID	UINT32	0xE000001B HMS	HMS secondary vendor id. A secondary vendor id will never pass the CT test.

E.1.2   Optionally – Improved Functionality, Customization and Identification of the Product

EtherCAT Object (F5h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
2	Product code	UINT32	00000036h ABCC 40 ECT	
6	Manufacture Device Name	Array of CHAR (Max 64 bytes)	"Anybus CompactCom 40 EtherCAT"	

E.2   CC-Link

Virtual attributes needed to pass CC-Link certification test in shift register mode:

E.2.1   Mandatory Implementations

Host CC-Link Object (F7h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description	Validation in Test ref.
1	Vendor code	NA	NA	Extracted from the CLPA assigned member number.	3.(1) Confirmation of station information according BAP-C0401-012-F.

E.2.2   Optionally – Improved Functionality, Customization and Identification of the Product

Host CC-Link Object (F7h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description	Validation in Test ref.
2	SW Version	1-63	Depends on ABCC version.	Incremented when the CC-Link network behavior is affected.	3.(1) Confirmation of station information according BAP-C0401-012-F.
3	Model code	1-127	127	Corresponds to the module profile.	3.(1) Confirmation of station information. 8(1) Profile confirmation according BAP-C0401-012-F.

E.3   Ethernet POWERLINK

Virtual attributes needed to pass Ethernet POWERLINK certification test in shift register mode:

E.3.1   Mandatory Implementations

POWERLINK Object (E9h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description	Note.
1	Vendor ID	UINT32	NA	This value replace the default value for CANopen identity object (0x1018 sub-index 0x01).	A unique Vendor-ID **SHALL** be assigned to the vendor by EPSG.

E.3.2   Optionally – Improved Functionality, Customization and Identification of the Product

POWERLINK Object (E9h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
2	Product code	UINT32	0x00000028	This value replace the default value for CANopen identity object (0x1018 sub-index 0x02).
3	Revision High word	UINT16	0x0000	This value replace the default value for CANopen identity object (0x1018 sub-index 0x03, high word).
4	Revision low word	UINT16	ABCC (hardware revision)	This value replace the default value for CANopen identity object (0x1018 sub-index 0x03, low word).
6	Manufacture Device Name	Array of CHAR (max 64 bytes)	"Anybus CompactCom 40 Ethernet POWERLINK"	Corresponds to Manufacturer Device Name object (0x1008).
14	Manufacture Name	Array of CHAR (max 64 bytes)	"HMS"	Will be used as part of Interface Description string in the Interface Group object (0x1030)

To ensure the functioning of the SYNC signal, define the following attributes in the SYNC Object (EEh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
1	Cycle time	UINT32	-	NMT Cycle Length Object 0x1006 (converted from microseconds to nanoseconds).
7	Sync mode	UINT16	-	If attribute 8 indicates support for synchronous operation, the ABCC will set this attribute to 1 at the start of synchronous operation and to 0 at the start of non-synchronous operation. If synchronous operation is not supported the ABCC will never change the value.
8	Supported sync modes	UINT16	-	Bit 0: Nonsynchronous operation. (Default value if nonsynchronous operation is supported.) Bit 1: 1=Synchronous operation supported. Bit 2-15: Reserved. Set to zero.

E.4   EtherNet/IP

Virtual attributes, needed to pass EtherNet/IP certification/plugfest test in shift register mode:

E.4.1   Mandatory Implementations

EtherNet/IP Object (F8h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
1	Vendor ID	UINT16	0x005A HMS	-

E.4.2   Optionally – Improved Functionality, Customization and Identification of the Product

EtherNet/IP Object (F8h), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
3	Product code	UINT16	0x0037 ABCC 40 EIP	Product code is assigned by the vendor
4	Revision	Struct of UINT8 UINT8	Anybus CompactCom Firmware revision	Revision of end product
6	Product Name	Array of CHAR (Max 32 bytes)	"CompactCom 40 EtherNet/IP(TM)"	Product name

E.5   DeviceNet

Virtual attributes, needed to pass DeviceNet certification test in shift register mode:

E.5.1   Mandatory Implementations

DeviceNet Object (FCh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
1	Vendor ID	UINT16	0x005A HMS	-

E.5.2   Optionally – Improved Functionality, Customization and Identification of the Product

DeviceNet Object (FCh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
3	Product code	UINT16	0x003F ABCC 40 DEV	Product code is assigned by the vendor
4	Revision	Struct of UINT8 UINT8	Anybus CompactCom Firmware revision	Revision of end product
6	Product Name	Array of CHAR (Max 32 bytes)	"CompactCom 40 DeviceNet(TM)"	Product name

E.6   Modbus-TCP

Virtual attributes, needed to pass Modbus-TCP certification test in shift register mode:

E.6.1 Mandatory Implementations

Modbus Host Object (FAh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
1	Vendor Name	Array of CHAR	HMS	-

E.6.2 Optionally – Improved Functionality, Customization and Identification of the Product

Modbus Host Object (FAh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
2	Product Code	Array of CHAR (Max 244 chars)	"Anybus CompactCom 40 Modbus-TCP"	Product code is assigned by the vendor.
3	Major Minor Revision	Array of CHAR (Max 244 chars)	""	Revision of end product
4	Vendor URL	Array of CHAR (Max 244 chars)	""	Vendor supplied URL
5	Product Name	Array of CHAR (Max 244 chars)	""	Vendor defined Product name
6	Model Name	Array of CHAR (Max 244 chars)	""	Vendor defined Model name
7	User Application Name	Array of CHAR (Max 244 chars)	""	Name of the user application.

E.6.3 CANopen

Virtual attributes, needed to pass CANopen certification test in shift register mode:

Mandatory Implementations

CANopen Object (FBh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
1	Vendor ID	UINT32	0xE000001B	The HMS Industrial Networks vendor ID cannot be used for certification of non-HMS products.

Optionally – Improved Functionality, Customization and Identification of the Product

CANopen Object (FBh), instance #1:

Attribute #	Attribute Name	Value range	Default value	Description
2	Product code	UINT32	0000000Dh	-
6	Manufacture Device Name	Array of CHAR (Max 64 bytes)	"Anybus CompactCom 40 CANopen"	-

F   Runtime Remapping of Process Data

This appendix describes how to handle a request from the network to remap read or write process data.

The functionality is available on EtherNet/IP, EtherCAT, PROFINET, PROFIBUS, and Ethernet POWERLINK.

F.1   SPI Mode

In SPI mode, telegrams are sent in full duplex. In the pictures this is illustrated by showing MISO and MOSI telegrams adjacent to each other. For more information on the SPI mode see *SPI Host Communication, p. 37*

F.1.1   Read Process Data

When the application has received a Remap_ADI_Read_Area request from the Anybus CompactCom 40 and has acknowledged the request, the Anybus CompactCom 40 will start sending read process data to the application according to the new mapping, the next time it receives new process data from the network. Not updated read process data will be sent according to the old mapping.

The Anybus CompactCom 40 sends Remap_ADI_Read_Area requests to the application in states where the read process data is inactive/invalid. Valid process data according to the new mapping will typically not be detected until the next time the Anybus CompactCom 40 enters the PROCESS_ACTIVE state.
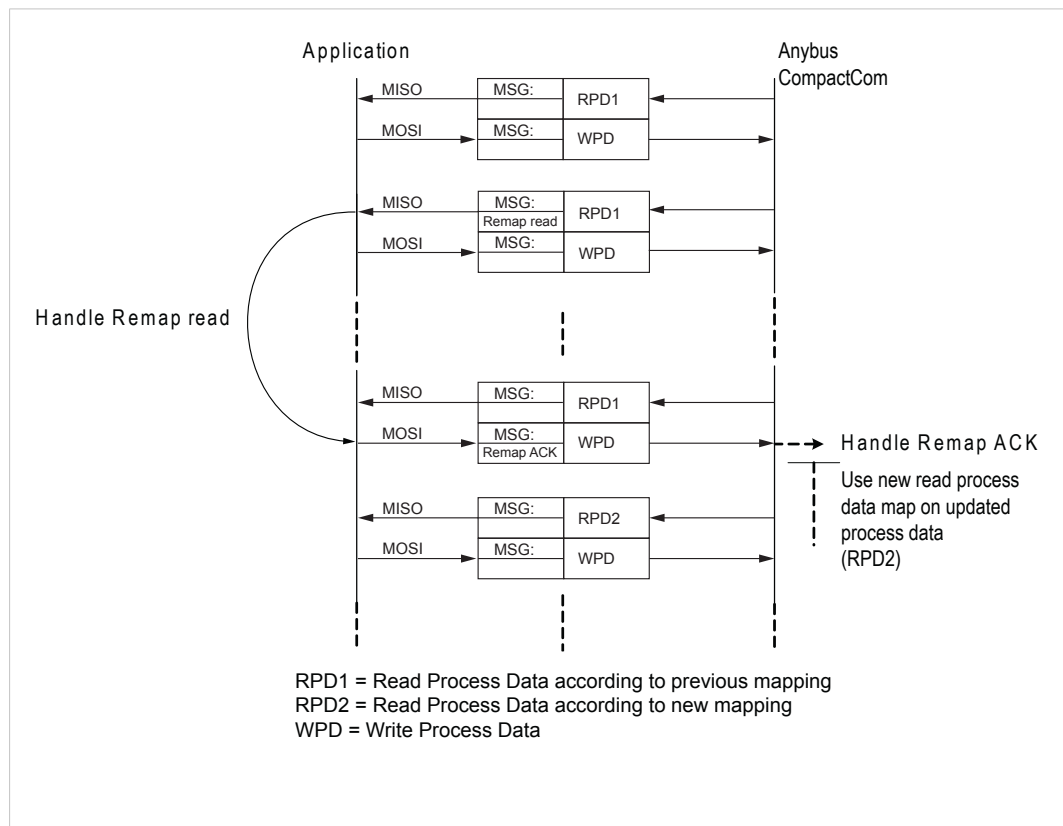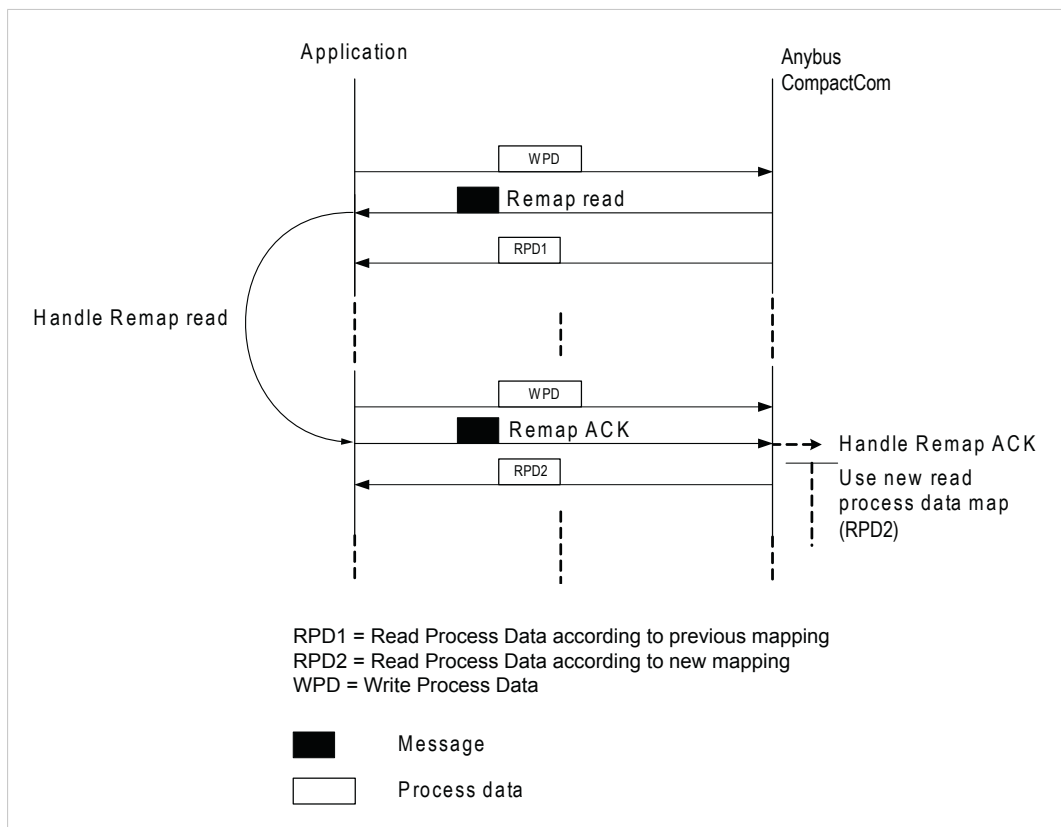

Fig. 25

F.1.2 Write Process Data

When the application has received a Remap_ADI_Write_Area request, it sends process data according to the new mapping starting with the SPI telegram that acknowledges the Remap_ADI_Write_Area request.
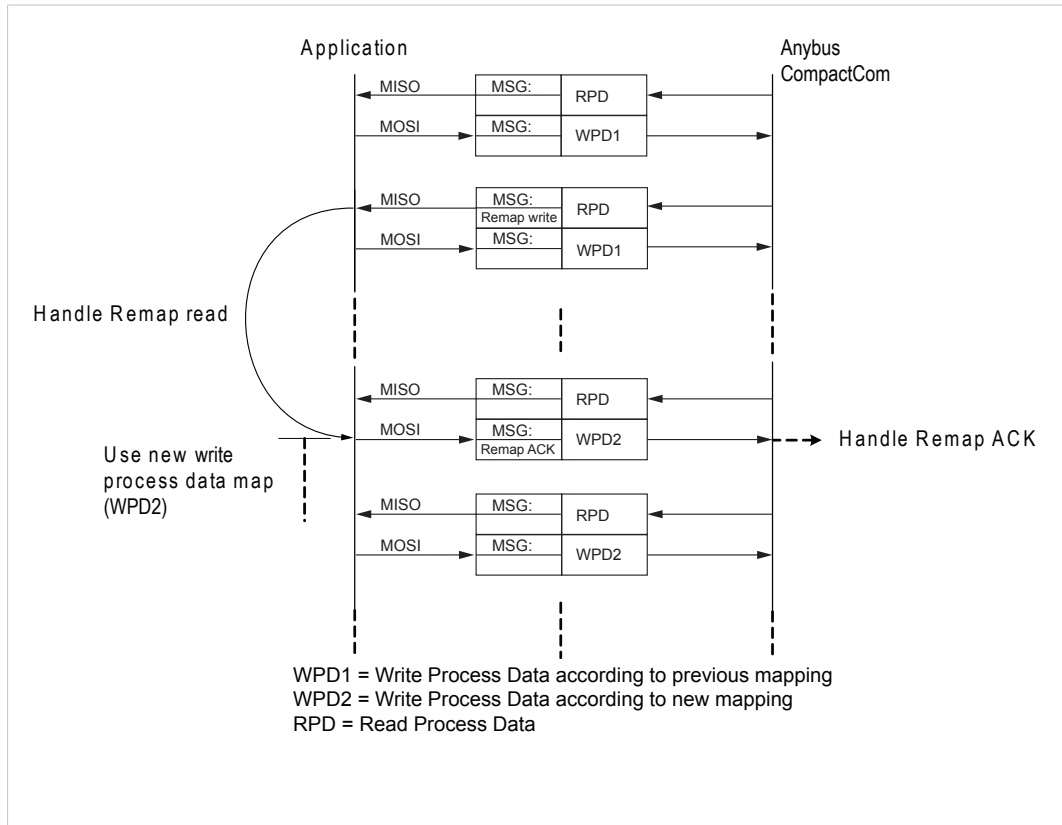
Fig. 26

F.2   Parallel Mode, 8/16 Bits, Event Driven

F.2.1   Read Process Data

When the application has received a Remap_ADI_Read_Area request from the Anybus CompactCom 40 and has acknowledged the request, the Anybus CompactCom 40 will start sending read process data to the application according to the new mapping the next time it receives such data from the network.

The Anybus CompactCom 40 sends Remap_ADI_Read_Area requests to the application in states where the read process data is inactive/invalid. Valid process data according to the new mapping will typically not be detected until the next time the Anybus CompactCom 40 enters the PROCESS_ACTIVE state.

Fig. 27

F.2.2   Write Process Data

When the application has received a Remap_ADI_Write_Area requests, it starts sending process data according to the new mapping to the Anybus CompactCom 40 before acknowledging the Remap_ADI_Write_Area request.

The Anybus CompactCom 40 regards the write process data as invalid between the time it sends a Remap_ADI_Write_Area request to the application until the remap request is acknowledged or not acknowledged.
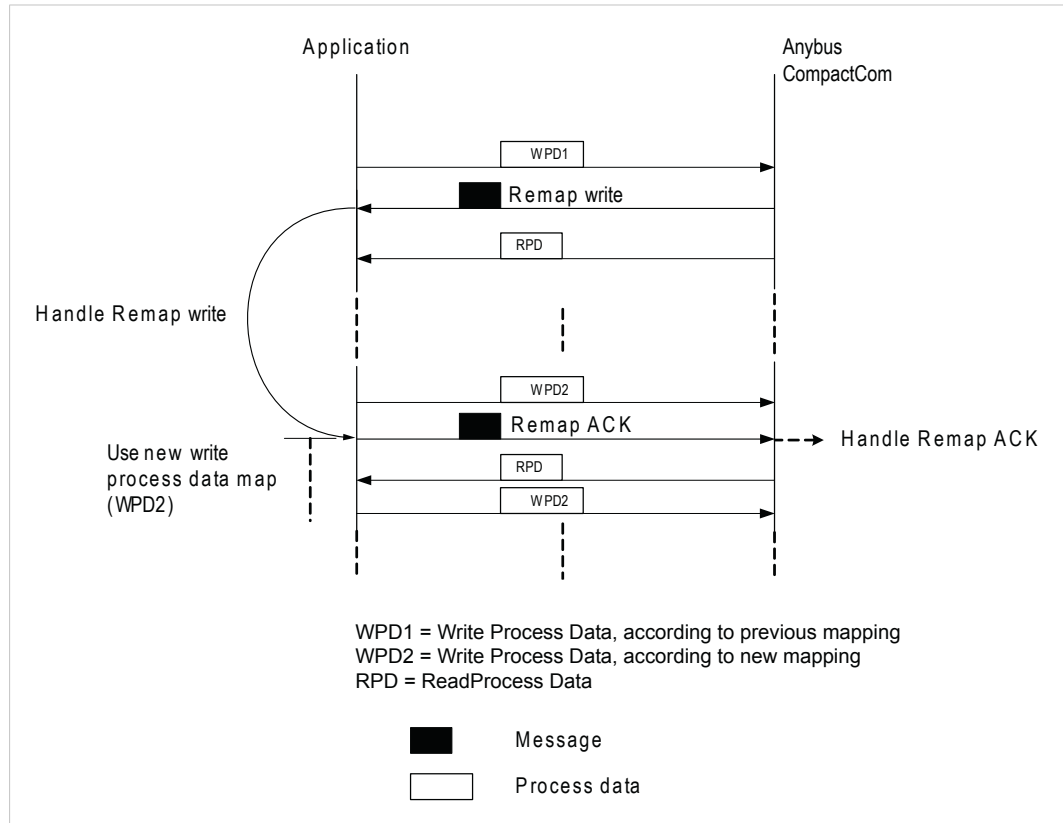
Fig. 28

F.3   Backwards Compatible Modes

In this section is described runtime remapping of process data in parallel and serial modes, backwards compatible to the Anybus CompactCom 30 series.

Please note that the telegrams are exchanged in a ping-pong fashion.

F.3.1   Parallel mode

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

Read Process Data
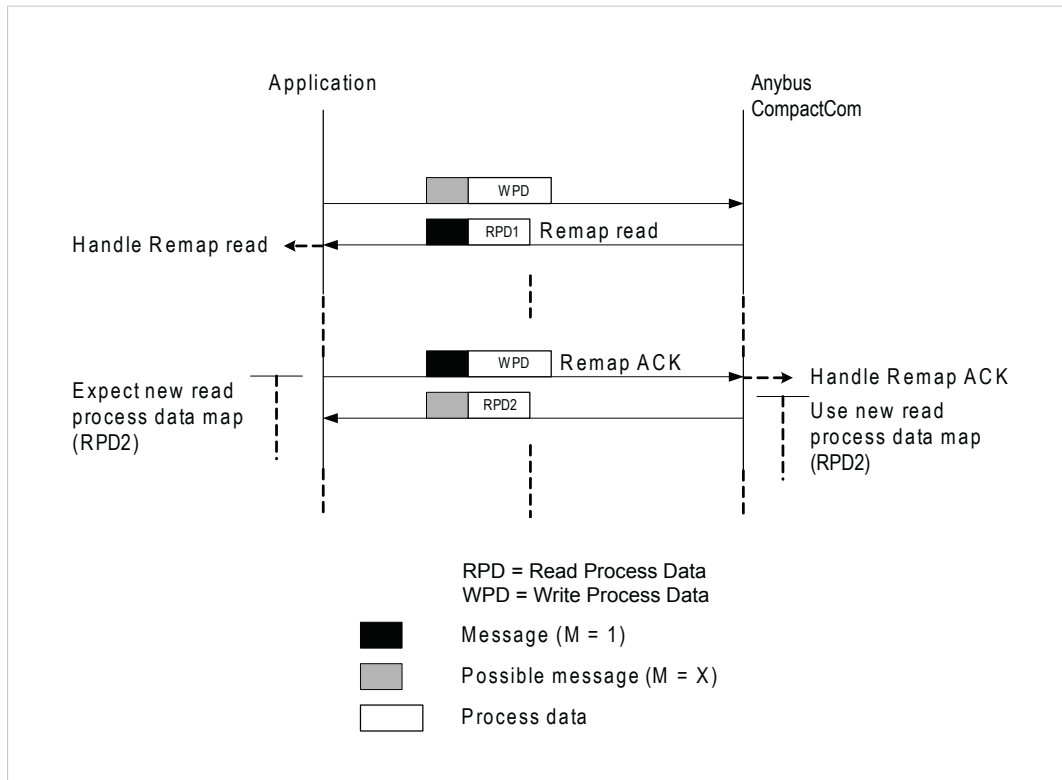
Fig. 29

Write Process Data
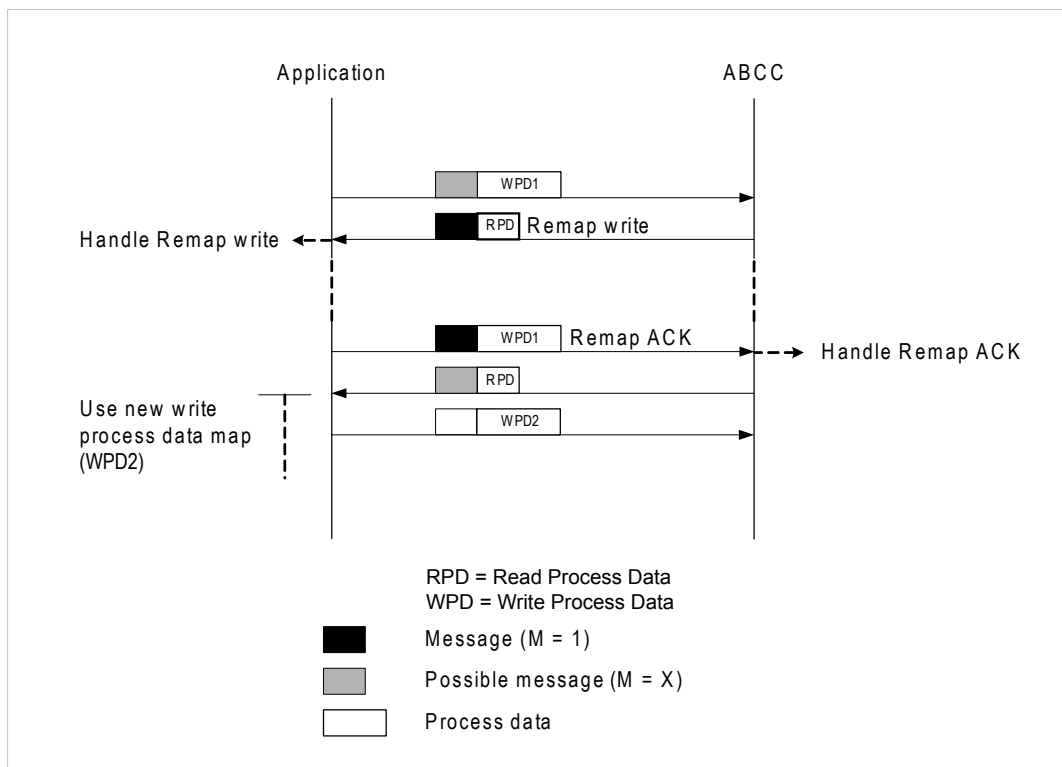
Fig. 30

F.3.2 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

This mode is backwards compatible to Anybus CompactCom 30.
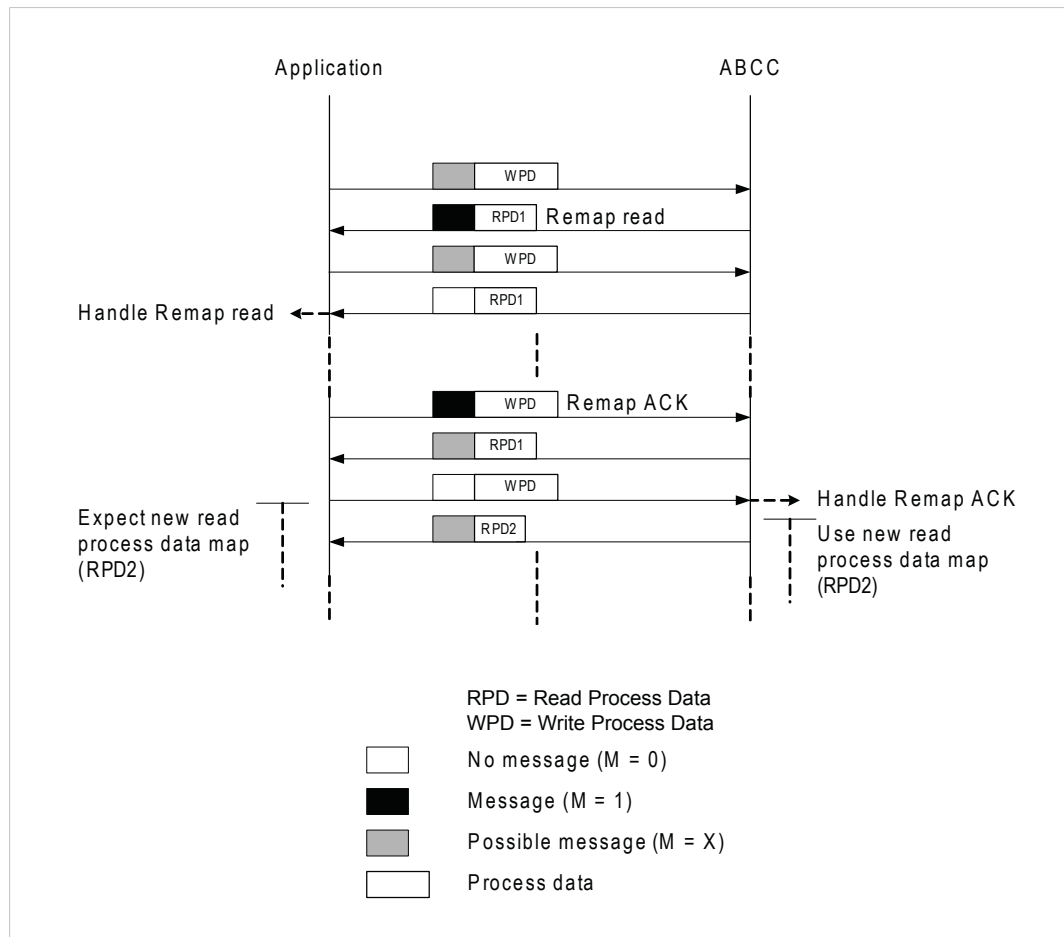
Read Process Data
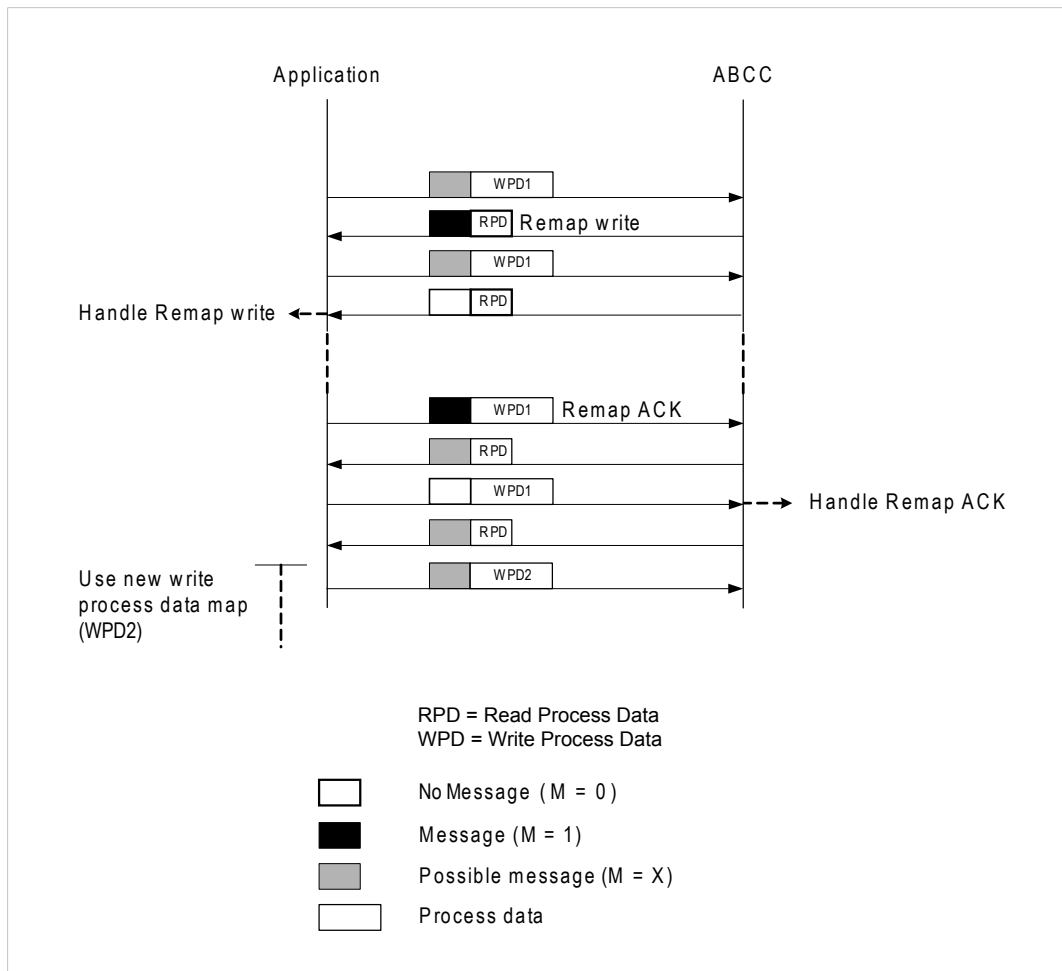

Fig. 31
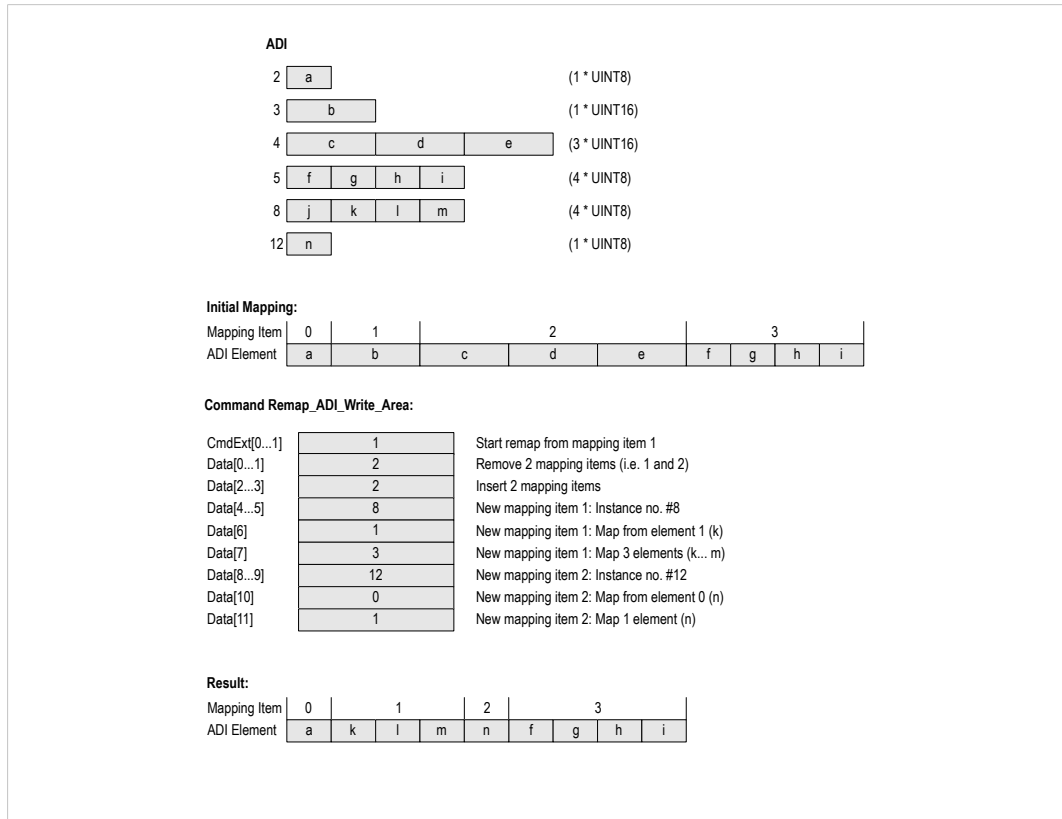
Write Process Data

Fig. 32

F.4 Example: Remap_ADI_Write_Area

Fig. 33

G   CRC Calculation (16–bit)

G.1   General

> **ℹ**   *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.

G.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 }`.

G.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 );
}
```

H   CRC Calculation (32–bit)

The example below shows how the CRC algorithm can be implemented in SPI mode. Please note that the order of bits and bytes have to be changed as SPI and the CRC algorithm use different bytes and bits orders.

H.1   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 CRC32: { 0xeb 0xf4 0x72 0x27 }.

The array { 0x00, 0x55, 0xAA, 0xFF, 0x0F, 0x5A, 0xA5, 0xF0 } should yield the following CRC32: { 0xbe 0xa7 0x3a 0x2d }.

The array { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 } should yield the following CRC32: { 0x9a 0xf6 0x4b 0x49 }.

H.2   Code Example

This example uses a fast approach to calculate the CRC.

```
const UINT8 abBitReverseTable16[] = { 0x0, 0x8, 0x4, 0xC, 0x2, 0xA,
      0x6, 0xE, 0x1, 0x9, 0x5, 0xD, 0x3, 0xB, 0x7, 0xF };
const UINT32 crc_table32[] = {
     0x4DBDF21CUL, 0x500AE278UL, 0x76D3D2D4UL, 0x6B64C2B0UL,
     0x3B61B38CUL, 0x26D6A3E8UL, 0x000F9344UL, 0x1DB88320UL,
     0xA005713CUL, 0xBDB26158UL, 0x9B6B51F4UL, 0x86DC4190UL,
     0xD6D930ACUL, 0xCB6E20C8UL, 0xEDB71064UL, 0xF0000000UL
};
UINT32 CRC_Crc32( UINT8* pbBufferStart, UINT16 iLength )
{
   UINT8 bCrcReverseByte;
   UINT16 i;
   UINT32 lCrc = 0x0;
   for(i = 0; i < iLength; i++)
   {
     bCrcReverseByte =
        lCrc ^ abBitReverseTable16[ (*pbBufferStart >> 4 ) & 0xf ];
     lCrc = (lCrc >> 4) ^ crc_table32[ bCrcReverseByte & 0xf ];
     bCrcReverseByte =
        lCrc ^ abBitReverseTable16[ *pbBufferStart  & 0xf ];
     lCrc = (lCrc >> 4) ^ crc_table32[ bCrcReverseByte & 0xf ];
     pbBufferStart++;
   }

   lCrc = ((UINT32)abBitReverseTable16 [ (lCrc & 0x000000F0UL) >> 4 ] )|
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x0000000FUL) ] ) << 4 |
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x0000000FUL) ] ) << 4 |
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x0000F000UL ) >> 12 ] << 8)|
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x00000F00UL ) >> 8 ] << 12)|
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x00F00000UL ) >> 20 ]<< 16)|
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x000F0000UL ) >> 16 ]<< 20)|
      ((UINT32)abBitReverseTable16 [ (lCrc & 0xF0000000UL ) >> 28 ]<< 24) |
      ((UINT32)abBitReverseTable16 [ (lCrc & 0x0F000000UL ) >> 24 ]<< 28);
   return lCrc;
}
```

I   Timing & Performance

I.1   General Information

This chapter specifies timing and performance parameters that are verified and documented for each member of the Anybus CompactCom 40 family.

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

Category	Parameters
Startup Delay	T1, T2
NW_INIT Handling	T100
Event Based WrMsg Busy Time	T103
Event Based Process Delay	T101, T102

> **!** At the time of writing, network specific timing specifications for all networks has not yet been publicly released. This information will be added continuously to all network guides when available.

I.2   Internal Timing

I.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)	1.5	s
T2	The Anybus is able to receive and handle the first application telegram (serial mode)	1.5	s

I.2.2   NW_INIT Handling

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.
Event based application message response time	> 1 ms

Parameter	Conditions
Ping-pong application response time	> 10 ms
No. of simultaneously outstanding Anybus commands that the application can handle	1

Parameter	Description	Communication	Max.	Unit.
T100	NW_INIT handling	All event based modes Required for modules supporting quick connect Recommended for all other modules	100	ms
		Serial 19.2 kbps	30	s
		(all other modes)	10	s

I.2.3 Event Based WrMsg Busy Time

The Event based WrMsg busy time is defined as the time it takes for the module to return the H_WRMSG area to the application after the application has posted a message.

Parameter	Description	Max.	Unit.
T103	H_WRMSG area busy time	500	µs

I.2.4 Event Based Process Data Delay

"Read process data delay" is defined as the time from when the last bit of the network frame enters the module, to when the RDPDI interrupt is asserted to the application.

"Write process data delay" is defined in two different ways, depending on network type.

• For software stack based cyclic/polled networks, it is defined as the time from when the module exchanges write process data buffers, to when the first bit of the new process data frame is sent out on the network.

• For COS (Change Of State) networks, it is defined as the time from when the application exchanges write process data buffers, to when the first bit of the new process data frame is sent out on the network.

A maximum delay of 500 µs for 32 byte process data is defined for compatibility with old ping-pong performance requirements, but high performance synchronous event based modules will never have a delay of more than 15 µs for 32 byte process data.

Parameter	Description	Recommended max for 32 byte process data	Absolute max for 32 byte process data	Unit
T101	Read process data delay	15	500	µs
T102	Write process data delay	15	500	µs

> **!** At the time of writing, network specific timing specifications for all networks has not yet been publicly released. This information will be added continuously to all network guides when available.

J   Backward Compatibility

The Anybus CompactCom M40 series of industrial network modules have significantly better performance and include more functionality than the modules in the Anybus CompactCom 30 series. The 40 series is backward compatible with the 30 series in that an application developed for the 30 series should be possible to use with the 40 series, without any major changes. Also it is possible to mix 30 and 40 series modules in the same application.

This appendix presents the backwards compatibility issues that have to be considered for Anybus CompactCom 40, when designing with both series in one application, or when adapting a 30 series application for the 40 series. For network specific information on backward compatibility, please consult the Anybus CompactCom 40 network guides.

J.1   Initial Considerations

There are two options to consider when starting the work to modify a host application developed for Anybus CompactCom 30-series modules to also be compatible with the 40-series modules:

- Add support with as little work as possible i.e. reuse as much as possible of the current design.
  - This is the fastest and easiest solution but with the drawback that many of the new features available in the 40-series will not be enabled (e.g. enhanced and faster communication interfaces, larger memory areas, and faster communication protocols).
  - You have to check the hardware and software differences below to make sure the host application is compatible with the 40-series modules. Small modifications to your current design may be needed.
- Make a redesign and take advantage of all new features presented in the 40-series.
  - A new driver and host application example code are available at www.anybus.com/starterkit40 to support the new communication protocol. This driver supports both 30-series and 40-series modules.
  - You have to check the hardware differences below and make sure the host application is compatible with the 40-series modules.

(**i**)   *This information only deals with differences between the 30-series and the 40-series.*

Link to support page: www.anybus.com/support.

J.2   Hardware Compatibility

Anybus CompactCom is available in three hardware formats; Module, Chip, and Brick.

J.2.1   Module

The modules in the 30-series and the 40-series share physical characteristics, like dimensions, outline, connectors, LED indicators, mounting parts etc. They are also available as modules without housing.

Fig. 34   **Anybus CompactCom M30/M40**

J.2.2   Chip

The chip (C30/C40) versions of the Anybus CompactCom differ completely when it comes to physical dimensions.

> **!**   There is no way to migrate a chip solution from the 30-series to the 40-series without a major hardware update.

J.2.3   Brick

The Anybus CompactCom B40-1 does not share dimensions with the Anybus CompactCom B30. The B40-1 is thus not suitable for migration. However HMS Industrial Networks has developed a separate brick version in the 40-series, that can be used for migration. This product, B40-2, shares dimensions etc. with the B30. Please contact HMS Industrial Networks for more information on the Anybus CompactCom B40-2.
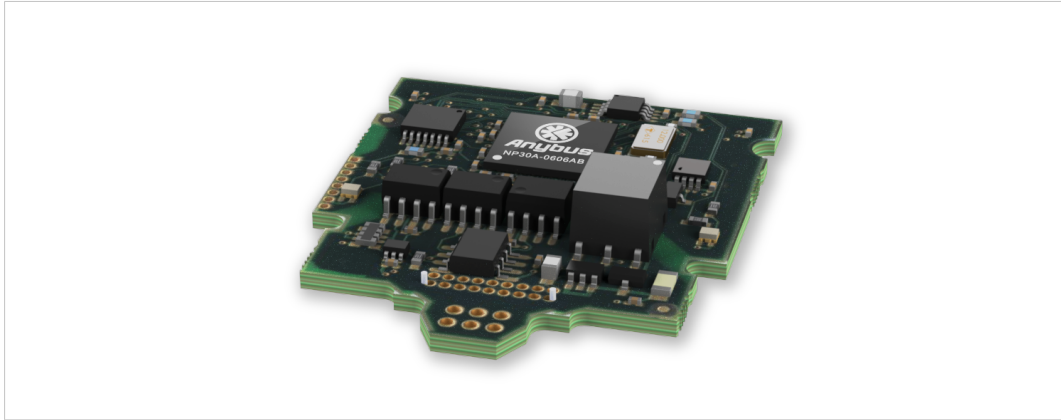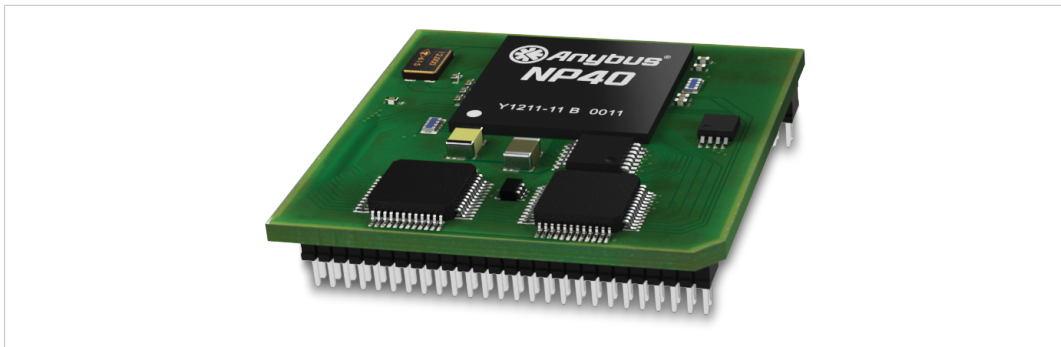

Fig. 35   Anybus CompactCom B30

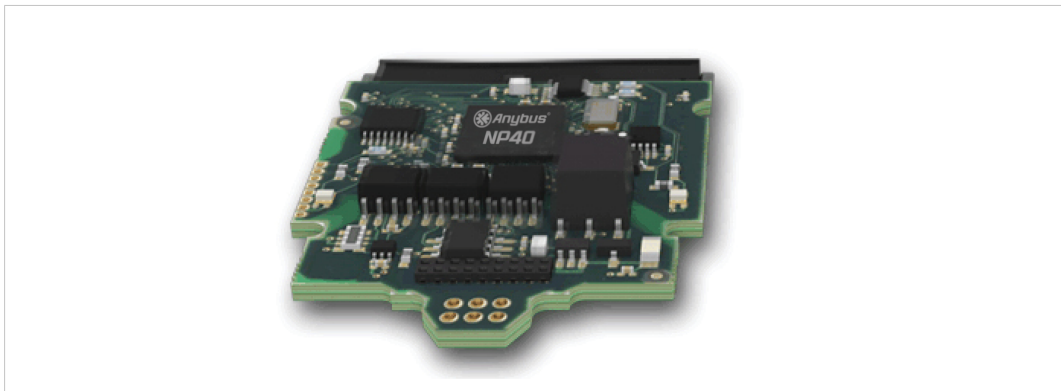

Fig. 36   Anybus CompactCom B40–1 (not for migration)

Fig. 37   Anybus CompactCom B40–2

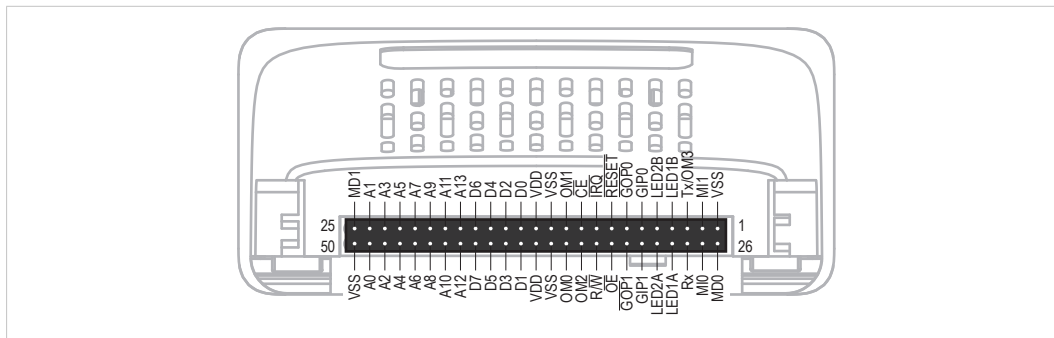## J.2.4   Host Application Interface

Fig. 38

Some signals in the host application interface have modified functionality and/or functions which must be checked for compatibility. See the following sections.

Tx/OM3

In the 30-series, this pin is only used for Tx. It is tri-stated during power up, and driven by the Anybus CompactCom UART after initialization. In the 40-series this pin is used as a fourth operating mode setting pin (OM3). During startup after releasing the reset, this pin is read to determine the operating mode to use. The pin is then changed to a Tx output.

In the 40-series, this pin has a built-in weak pull-up. If this pin, on a 30-series module or brick is unconnected, pulled high, or connected to a high-Z digital input on the host processor, it will be compatible with the 40-series. An external pull-up is recommended, but not required.

> **!** If this pin is pulled low by the host during startup in a 30-series application, any 40-series module or brick, substituted in the application, will not enter the expected operating mode.

Related Information: Anybus CompactCom M40 Hardware Design Guide (HMSI-216-126), Section "Application Connector Pin Overview"

Module Identification (MI[0..1])

These pins are used by the host application (i.e. your product) to identify what type of Anybus CompactCom that is mounted. The identification differs between the 30-series and the 40-series.

> **ⓘ** *If your software use this identification you need to handle the new identification value.*

MI1	MI0	Module Type
LOW	LOW	Active Anybus CompactCom 30
HIGH	LOW	Active Anybus CompactCom 40

MI[0..1] shall only be sampled by the application during the time period from power up to the end of SETUP state. The pins are low at power up and before reset release.

Related Information: *Anybus CompactCom M40 Hardware Design Guide (HMSI-216-126)*, Section "Settings/Sync".

GIP[0..1]/LED3[A..B]

These pins are tri-stated inputs by default in the 30-series. In the 40-series, these pins are tri-stated until the state NW_INIT. After that they become open-drain, active low LED outputs (LED3A/LED3B).

No modification of the hardware is needed, if your current design has

• tied these pins to GND

• pulled up the pins

• pulled down the pins

• left the pins unconnected

However, if the application drive the pins high, a short circuit will occur.

If you connect the pins to LEDs, a pull-up is required.

In the 40-series, there is a possibility to set the GIP[0..1] and GOP[0..1] in high impedance state (tri-state) by using attribute #16 (GPIO configuration) in the Anybus object (01h). I.e. if it is not possible to change the host application hardware, this attribute can be configured for high impedance state of GIP and GOP before leaving NW_INIT state.

Related Information: *Anybus CompactCom M40 Hardware Design Guide (HMSI-216-126)*, Section "LED Interface/D8-D15 (Data Bus)".

GOP[0..1]/LED4[A..B]

These pins are outputs (high state) by default in the 30-series. In the 40-series, these pins are tri-stated until the state NW_INIT, and after that they become push-pull, active low LED outputs (LED4A/LED4B).

This change should not affect your product.

Related Information: *Anybus CompactCom M40 Hardware Design Guide (HMSI-216-126)*, Section 3.2.3, "LED Interface/D8-D15 (Data Bus)".

Address Pins A[11..13]

The address pins 11, 12, and 13 are ignored by the 30-series. These pins must be high when accessing the 40-series module in backwards compatible 8-bit parallel mode. If you have left these pins unconnected or connected to GND, you need to make a hardware modification to tie them high.

Max Input Signal Level (V_{IH})

The max input signal level for the 30-series is specified as $V_{IH}=V_{DD}+0,2$ V, and for the 40-series as $V_{IH}=3.45$ V. Make sure that you do not exceed 3.45 V for a logic high level.

RMII Compatibility

If the RMII mode is being used on an Anybus CompactCom 40 module and it is desired to remain compatible with the 30 series, it is important to disable this connection when switching to an Anybus CompactCom 30 module due to pin conflicts. The RMII port of the host processor should be set to tristate by default, and only be enabled if an RMII capable Anybus CompactCom 40 is detected. In case the RMII connection cannot be disabled through an internal hardware control on the host processor, it will be necessary to design in external hardware (i.e. a FET bus switch) to prevent short circuits

Related Information: *Anybus CompactCom M40 Hardware Design Guide (HMSI-216-126)*, Section 3.2.5, "RMII — Reduced Media-Independent Interface".

J.3   General Software

J.3.1   Extended Memory Areas

The memory areas have been extended in the 40-series, and it is now possible to access larger sizes of process data (up to 4096 bytes instead of former maximum 256 bytes) and message data (up to 1524 bytes instead of former maximum 255 bytes). The 30-series has reserved memory ranges that the application should not use. The 40-series implements new functionality in some of these memory areas.

> (i)   *To use the extended memory areas you need to implement a new communication protocol which is not part of this document.*
>
> *Memory areas not supported by the specific network cannot be used. Make sure you do not access these areas, e.g. for doing read/write memory tests.*

Related Information: *Anybus CompactCom 40 Software Design Guide (HMSI-216-125)*, Section "Memory Map"

J.3.2   Faster Ping-Pong Protocol

The ping-pong protocol (the protocol used in the 30-series) is faster in the 40-series. A 30-series module typically responds to a so called ping within 10-100 µs. The 40-series typically responds to a ping within 2 µs.

Interrupt-driven applications (parallel operating mode) may see increased CPU load due to the increased speed.

J.3.3   Requests from Anybus CompactCom to Host Application During Startup

All requests to software objects in the host application must be handled and responded to (even if the object does not exist). This applies for both the 30-series and the 40-series. The 40-series introduces additional objects for new functionality.

There may also be additional commands in existing objects added to the 40-series that must be responded to (even if it is not supported).

If your implementation already responds to all commands it cannot process, which is the expected behavior, you do not need to change anything.

J.3.4   Anybus Object (01h)

Attribute	30-series	40-series	Change/Action/Comment
#1, Module Type	0401h	0403h	Make sure the host application accepts the new module type value for the 40-series.
#15, Auxiliary Bit	Available	Removed	It is not possible to turn off the "Changed Data Indication" in the 40-series. Also see "Control Register CTRL_AUX-bit" and "Status Register STAT_AUX-bit" below.
#16, GPIO Configuration	Default: General input and output pins	Default: LED3 and LED4 outputs	See also .. • *GIP[0..1]/LED3[A..B], p. 174* • *GOP[0..1]/LED4[A..B], p. 174*

J.3.5   Control Register CTRL_AUX-bit

30-series   The CTRL_AUX bit in the control register indicates to the Anybus CompactCom if the process data in the current telegram has changed compared to the previous one.

40-series   The value of the CTRL_AUX bit is always ignored. Process data is always accepted.

All released Anybus CompactCom 30 example drivers from Anybus CompactCom comply with this difference.

Related Information: *Anybus CompactCom 40 Software Design Guide (HMSI-216-125)*, section "Control Register".

J.3.6   Status Register STAT_AUX-bit

30-series   The STAT_AUX bit in the status register indicates if the output process data in the current telegram has changed compared to the previous one. This functionality must be enabled in the Anybus object (01h), Attribute #15. By default, the STAT_AUX bit functionality is disabled.

40-series   The STAT_AUX bit indicates updated output process data (not necessarily changed data) from the network compared to the previous telegram. The functionality is always enabled.

All released Anybus CompactCom 30 example drivers from HMS Industrial Networks comply with this difference.

Related Information: *Anybus CompactCom 40 Software Design Guide (HMSI-216-125)*, section "Status Register".

J.3.7   Control Register CTRL_R-bit

30-series   The application may change this bit at any time.

40-series   For the 8-bit parallel operating mode, the bit is only allowed to transition from 1 to 0 when the STAT_M-bit is set in the status register. When using the serial operating modes, it is also allowed to transition from 1 to 0 in the telegram immediately after the finalizing empty fragment.

All released Anybus CompactCom 30 example drivers from HMS Industrial Networks comply with this difference.

Related Information: *Anybus CompactCom 40 Software Design Guide (HMSI-216-125)*, section "Control Register".

J.3.8 Modifications of Status Register, Process Data Read Area, and Message Data Read Area

In the 40-series, the Status Register, the Process Data Read Area, and the Message Data Read Area are write protected in hardware (parallel interface). If the software for some reason writes to any of those areas, a change is needed.

All releasedAnybus CompactCom 30 example drivers from HMS Industrial Networks comply with this difference.