Anybus CompactCom 40

EtherNet/IP

NETWORK GUIDE

SCM-1202-031 1.5 ENGLISH

Important User Information

Liability

Intellectual Property Rights

Table of Contents

Page

1 Preface

1.1 About this document

This document is intended to provide a good understanding of the functionality offered by the Anybus CompactCom 40 EtherNet/IP. The document describes the features that are specific to Anybus CompactCom 40 EtherNet/IP. For general information regarding Anybus CompactCom, consult the Anybus CompactCom design guides.

The reader of this document is expected to be familiar with high level software design and communication systems in general. The information in this network guide should normally be sufficient to implement a design. However if advanced EtherNet/IP specific functionality is to be used, in-depth knowledge of EtherNet/IP networking internals and/or information from the official EtherNet/IP specifications may be required. In such cases, the persons responsible for the implementation of this product should either obtain the EtherNet/IP specification to gain sufficient knowledge or limit their implementation in such a way that this is not necessary.

For additional related documentation and file downloads, please visit the support website at www.anybus.com/support.

1.2 Related Documents

Document	Author	Document ID
Anybus CompactCom 40 Software Design Guide	HMS	HMSI-216-125
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
CIP specification, Volumes 1 (CIP Common) and 2 (EtherNet/IP)	ODVA	

1.3 Document History

1.1	2017-01-18	FM to DOX, change of document number from HMSI-27-212 to SCM-1202-031. Version numbering restarted. M12 connectors added Minor corrections and updates
1.2	2017-05-23	Ethernet Host Object updated (disabling of DHCP) Port Object updated
1.3	2017-07-11	Added appendix for backward compatibility Updated TCP/IP Interface object (CIP)
1.4	2017-11-28	Added Assembly Mapping Object guide
1.5	2017-12-15	Updated Copyright Appendix

1.4 Document Conventions

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

1. First do this
2. Then do this

Unordered (bulleted) lists are used for:

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

...and for action-result type instructions:

► This action...

 ➡ leads to this result

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

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

This is a cross-reference within this document: *Document Conventions, p. 5*

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

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

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

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

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

1.5 Document Specific Conventions

- The terms “Anybus” or “module” refers to the Anybus CompactCom module.
- The terms “host” or “host application” refer to the device that hosts the Anybus.
- Hexadecimal values are written in the format NNNNh or 0xNNNN, where NNNN is the hexadecimal value.
- A byte always consists of 8 bits.
- The terms “basic” and “extended” are used to classify objects, instances and attributes.

1.6 Abbreviations

Abbreviation	Meaning
API	assigned packet interval
RPI	requested packet interval
T	target (in this case the module)
O	origin (in this case the master)

1.7 Trademark Information

Anybus® is a registered trademark of HMS Industrial Networks AB.

EtherNet/IP is a trademark of ODVA, Inc.

2 About the Anybus CompactCom 40 EtherNet/IP

2.1 General

The Anybus CompactCom 40 EtherNet/IP communication module provides instant EtherNet/IP conformance tested connectivity via the patented Anybus CompactCom host interface. Any device that supports this standard can take advantage of the features provided by the module, allowing seamless network integration regardless of network type. The module supports both linear and ring network topology (DLR, Device Level Ring).

The modular approach of the Anybus CompactCom 40 platform allows the CIP-object implementation to be extended to fit specific application requirements. Furthermore, the Identity Object can be customized, allowing the end product to appear as a vendor-specific implementation rather than a generic Anybus module.

This product conforms to all aspects of the host interface for Anybus CompactCom 40 modules defined in the Anybus CompactCom 40 Hardware and Software Design Guides, making it fully interchangeable with any other device following that specification. Generally, no additional network related software support is needed, however in order to be able to take full advantage of advanced network specific functionality, a certain degree of dedicated software support may be necessary.

2.2 Features

- Two EtherNet/IP ports
- Ethernet connectors or M12 connectors
- Max. read process data: 1448 bytes
- Max. write process data: 1448 bytes
- Max. process data (read + write, in bytes): 2896 bytes
- Beacon Based DLR (Device Level Ring) and linear network topology supported
- Black channel interface, offering a transparent channel supporting CIP Safety.
- 10/100 Mbit, full/half duplex operation
- Web server w. customizable content
- FTP server
- Email client
- Server Side Include (SSI) functionality
- JSON functionality
- Customizable Identity Information
- Up to 65535 ADIs
- CIP Parameter Object support
- Expandable CIP-object implementation
- Supports unconnected CIP routing
- Transparent Socket Interface
- Modular Device functionality
- QuickConnect supported
- Multiple IO assembly instances can be created

3 Basic Operation

3.1 General Information

3.1.1 Software Requirements

No additional network support code needs to be written in order to support the Anybus CompactCom 40 EtherNet/IP, however due to the nature of the EtherNet/IP networking system, certain restrictions must be taken into account:

- Certain functionality in the module requires that the command Get_Instance_Number_By_Order (Application Data Object, FEh) is implemented in the host application.
- Up to 5 diagnostic instances (See *Diagnostic Object (02h), p. 99*) can be created by the host application during normal conditions. An additional 6th instance may be created in event of a major fault. This limit is set by the module, not by the network.
- EtherNet/IP in itself does not impose any specific timing demands when it comes to acyclic requests (i.e. requests towards instances in the Application Data Object), however it is generally recommended to process and respond to such requests within a reasonable time period. The application that sends the request, also decides the timeout, e.g. EIPScan employs a timeout of 10 seconds.
- The use of advanced CIP-specific functionality may require in-depth knowledge in CIP networking internals and/or information from the official CIP and EtherNet/IP specifications. In such cases, the people responsible for the implementation of this product is expected either to obtain these specifications to gain sufficient knowledge or limit their implementation is such a way that this is not necessary.

See also...

- *Diagnostic Object (02h), p. 99* (Anybus Module Objects)
- Anybus CompactCom 40 Software Design Guide, “Application Data Object (FEh)”

For in depth information regarding the Anybus CompactCom software interface, consult the Anybus CompactCom 40 Software Design Guide.

3.1.2 Electronic Data Sheet (EDS)

On EtherNet/IP, the characteristics of a device is stored in an ASCII data file with the suffix EDS. This file is used by configuration tools etc. when setting up the network configuration. HMS Industrial Networks AB supplies a standard (generic) EDS file, which corresponds to the default settings in the module. However, due to the flexible nature of the Anybus CompactCom concept, it is possible to alter the behavior of the product in ways which invalidate the generic EDS file. In such case, a custom EDS file needs to be created, which in turn invalidates the default identity information and require re-certification of the product.

Since the module implements the Parameter Object, it is possible for configuration tools such as RSNetWorx to automatically generate a suitable EDS-file. Note that this functionality requires that the command Get_Instance_Number_By_Order (Application Data Object, FEh) has been implemented in the host application.

See also..

- *Parameter Object (0Fh), p. 75* (CIP object)
- Anybus CompactCom 40 Software Design Guide, “Application Data Object (FEh)”

! HMS Industrial Networks AB approves use of the standard EDS-file only under the condition that it matches the actual implementation and that the identity information remains unchanged.

3.2 Network Identity

By default, the module uses the following identity settings:

Vendor ID:	005Ah (HMS Industrial Networks)
Device Type:	002Bh (Generic Device)
Product Code:	0037h (Anybus CompactCom 40 EtherNet/IP)
Product Name:	“Anybus CompactCom 40 EtherNet/IP(TM)”

Optionally, it is possible to customize the identity of the module by implementing the corresponding instance attributes in the EtherNet/IP Host Object.

See also...

- *Identity Object (01h), p. 64* (CIP object)
- *EtherNet/IP Host Object (F8h), p. 150* (Host Application Object)

! According to the CIP specification, the combination of Vendor ID and serial number must be unique. It is not permitted to use a custom serial number in combination with the HMS Vendor ID (005Ah), nor is it permitted to choose Vendor ID arbitrarily. Failure to comply to this requirement will induce interoperability problems and/or other unwanted side effects. HMS approves use of the HMS Vendor ID (005Ah), in combination with the default serial number, under the condition that the implementation requires no deviations from the standard EDS-file.

To obtain a Vendor ID, contact the ODVA.

3.3 Communication Settings

Network related communication settings are grouped in the Network Configuration Object (04h), and includes:

IP settings	These settings must be set properly in order for the module to be able to participate on the network. The module supports DHCP, which may be used to retrieve the IP settings from a DHCP-server automatically. DHCP is enabled by default, but can be disabled if necessary.
Physical Link Settings	By default, the module uses auto negotiation to establish the physical link settings, however it is possible to force a specific setting if necessary.

The parameters in the Network Configuration Object (04h) are available from the network through the built in web server, and through the TCP/IP Interface Object (CIP).

See also...

- *Web Server, p. 25*
- *TCP/IP Interface Object (F5h), p. 88* (CIP object)
- *Ethernet Link Object (F6h), p. 92* (CIP object)
- *Network Configuration Object (04h), p. 101* (Anybus Module Object)
- *Secure HICP (Secure Host IP Configuration Protocol), p. 165*

3.3.1 Communication Settings in Stand Alone Shift Register Mode

If the Anybus CompactCom is used stand alone, there is no application from which to set the IP address. The IP address is instead set using the DIP1 switches (IP address byte 3) and the virtual attributes (Ethernet Host object (F9h), attribute #17), that are written to memory during set-up (IP address byte 0 - 2). A flowchart is shown below.

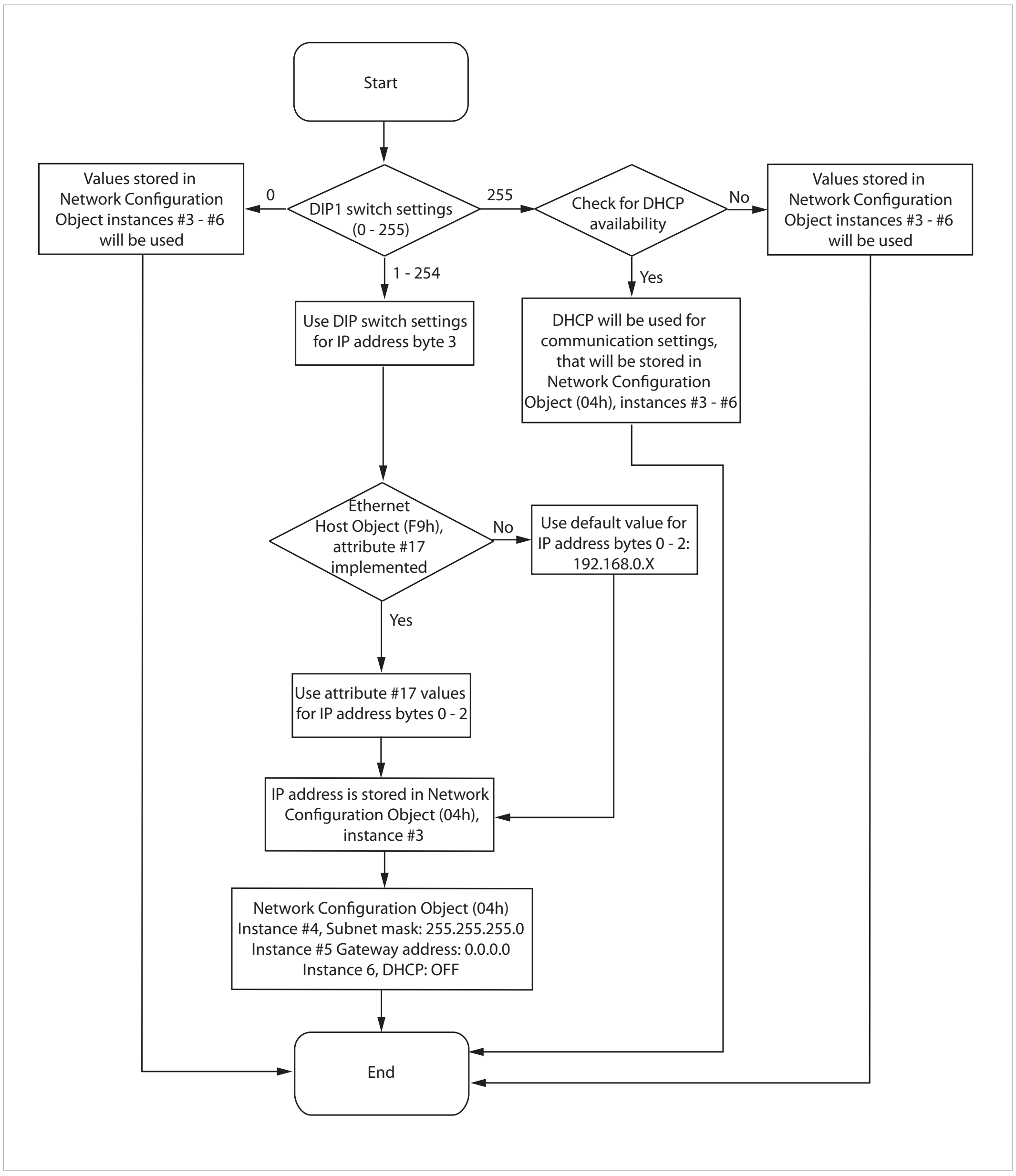

Start
DIP1 switch settings (0 - 255)
0
Values stored in Network Configuration Object instances #3 - #6 will be used
255
Check for DHCP availability
No
Values stored in Network Configuration Object instances #3 - #6 will be used
Yes
DHCP will be used for communication settings, that will be stored in Network Configuration Object (04h), instances #3 - #6
1 - 254
Use DIP switch settings for IP address byte 3
Ethernet Host Object (F9h), attribute #17 implemented
No
Use default value for IP address bytes 0 - 2: 192.168.0.X
Yes
Use attribute #17 values for IP address bytes 0 - 2
IP address is stored in Network Configuration Object (04h), instance #3
Network Configuration Object (04h) Instance #4, Subnet mask: 255.255.255.0 Instance #5 Gateway address: 0.0.0.0 Instance 6, DHCP: OFF
End

Fig. 1

See also ...

- *Ethernet Host Object (F9h), p. 159*
- Anybus CompactCom M40 Hardware Design Guide
- *Network Configuration Object (04h), p. 101*

3.4 Beacon Based DLR (Device Level Ring)

Device Level Ring (DLR) is a network technology for industrial applications that uses embedded switch functionality in automation end devices, such as programmable automation controllers and I/O modules, to enable Ethernet ring network topologies at the device level. DLR technology adds network resilience to optimize machine operation. Beacon based DLR networks consist of a ring supervisor and a number of ring nodes, and use “beacons” to detect breaks in the ring. When a DLR network detects a break in the ring, it provides ways to alternatively route the data to recover the network. Diagnostics built into DLR products can identify the point of failure, thus helping to speed maintenance and reduce repair time. The Anybus CompactCom 40 EtherNet/IP implements the DLR protocol, and it is enabled by default. The device is able to process and act on beacon frames sent by ring supervisors, and supports beacon rates down to 100 µs. If needed, the DLR functionality can be disabled. This can be done by setting attribute #31 (Enable DLR) in the EtherNet/IP Host Object to False. See *EtherNet/IP Host Object (F8h), p. 150*.

3.5 Network Data Exchange

3.5.1 Application Data

Application Data Instances (ADIs) are represented through the ADI Object (CIP). Each instance within this objects corresponds directly to an instance in the Application Data Object on the host application side.

Accessible range of ADIs is 1 to 65535.

See also...

- *Parameter Object (0Fh), p. 75* (CIP object)
- *ADI Object (A2h), p. 84* (CIP object)

3.5.2 Process Data

Process Data is represented as dedicated instances in the Assembly Object (CIP).

See also...

- *Assembly Object (04h), p. 68* (CIP object)
- *Connection Manager (06h), p. 71* (CIP object)

3.5.3 Translation of Data Types

The Anybus data types are translated to CIP-standard and vice versa as follows:

Anybus Data Type	CIP Data Type	Comments
BOOL	BOOL	Each ADI element of this type occupies one byte.
ENUM	USINT	
SINT8	SINT	
UINT8	USINT	
SINT16	INT	Each ADI element of this type occupies two bytes.
UINT16	UINT	
SINT32	DINT	Each ADI element of this type occupies four bytes.
UINT32	UDINT	
FLOAT	REAL	
CHAR	SHORT_ STRING	SHORT_STRING consists of a single-byte length field (which in this case represents the number of ADI elements) followed by the actual character data (in this case the actual ADI elements). This means that a 10-character string occupies 11 bytes.

Anybus Data Type	CIP Data Type	Comments
SINT64	LINT	Each ADI element of this type occupies eight bytes.
UINT64	ULINT	
BITS8	BYTE	Each ADI element of this type occupies one byte.
BITS16	WORD	Each ADI element of this type occupies two bytes.
BITS32	DWORD	Each ADI element of this type occupies four bytes.
OCTET	USINT	
BITS1-7	BYTE	Bit fields of size 1 - 7
PAD0-8	BYTE	Bit fields of size 0 - 8 used for padding
PAD9-16	BYTE	Bit fields of size 9 - 16 used for padding

3.6 Web Interface

The web interface can be fully customized to suit a particular application. Dynamic content can be created by means of JSON and SSI scripting. Data and web pages are stored in a FLASH-based file system, which can be accessed using any standard FTP-client.

See also...

- *File System, p. 16*
- *FTP Server, p. 23*
- *Web Server, p. 25*
- *Server Side Include (SSI), p. 33*
- *JSON, p. 53*

3.7 E-mail Client

The built-in e-mail client enables the host application to send e-mail messages stored in the file system, or defined directly within the SMTP Client Object (09h). Messages are scanned for SSI content, which means it's possible to embed dynamic information from the file system.

See also...

- *File System, p. 16*

3.8 Modular Device Functionality

Modular devices consist of a backplane with a certain 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.

A list of modules in a Modular Device is available to the EtherNet/IP network master by a request to the CIP Identity object.

See also ...

- "Modular Device Object (ECh)" (see Anybus CompactCom 40 Software Design Guide)
- *Identity Object (01h), p. 64* (CIP object)

3.9 File System

3.9.1 Overview

The Anybus CompactCom 40 EtherNet/IP has an in-built file system, that can be accessed from the application and from the network. Three directories are predefined:

VFS	The virtual file system that e.g. holds the web pages of the module. The virtual file system is enabled by default in the Anybus File System Interface Object (0Ah).
Application	This directory provides access to the application file system through the Application File System Interface Object (EAh) (optional).
Firmware	Firmware updates are stored in this directory.

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

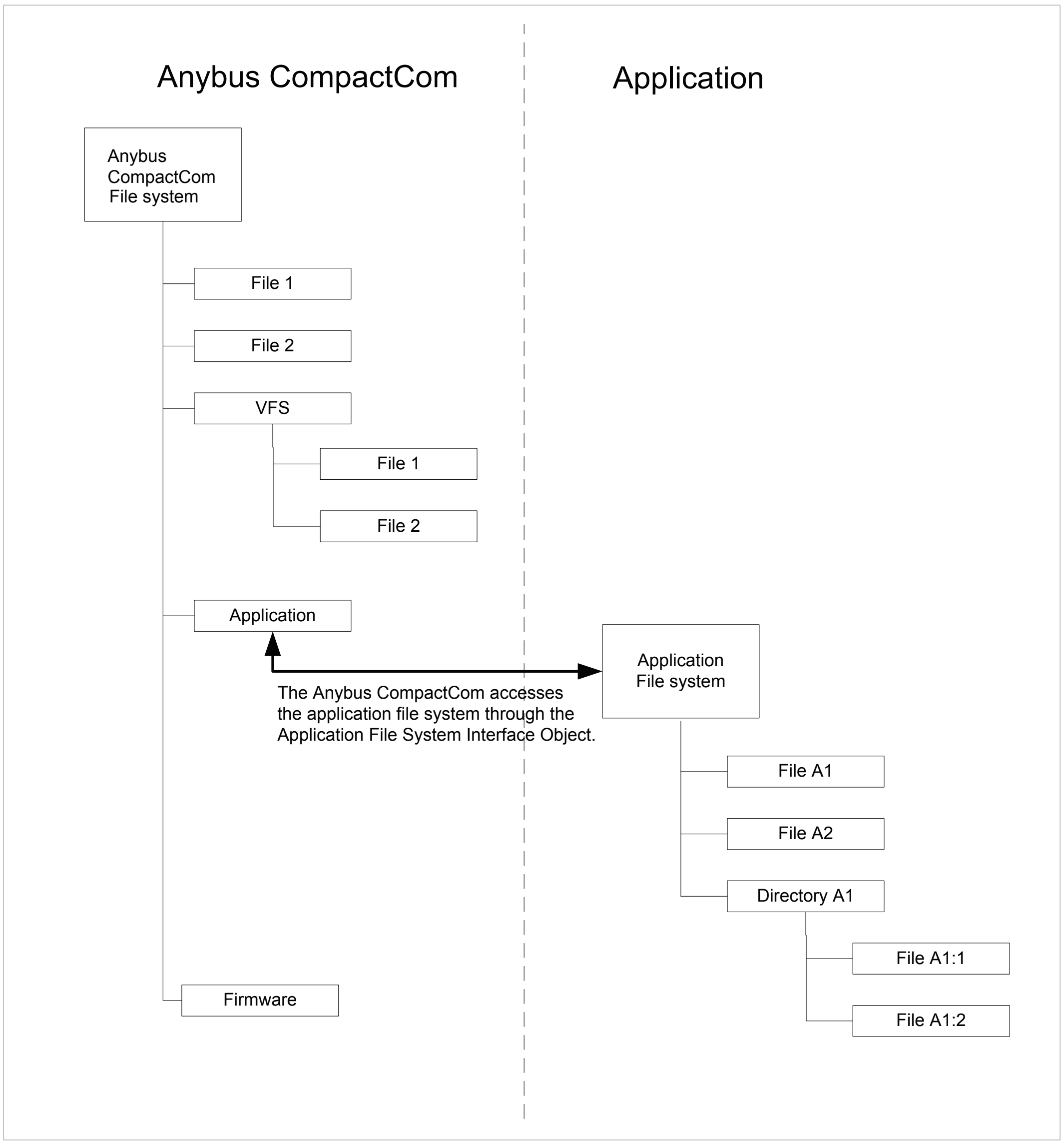

Anybus CompactCom
Application
Anybus CompactCom File system
File 1
File 2
VFS
File 1
File 2
Application
Application File system
The Anybus CompactCom accesses the application file system through the Application File System Interface Object.
File A1
File A2
Directory A1
File A1:1
File A1:2
Firmware

Fig. 2

3.9.2 General Information

The built-in file system hosts 28 MByte of non volatile storage, which can be accessed by the HTTP and FTP servers, the email client, and the host application (through the Anybus File System Interface Object (0Ah)).

The maximum number of directories and files, that can be stored in the root directory, is 511 if only short filenames are used (8 bytes name + 3 bytes extension). The number of files that can be stored in other directories, than the root directory, is unlimited.

The file system uses the following conventions:

- \ (backslash) is used as a path separator
- Names may contain spaces, but must not begin or end with one.
- Valid characters in names are ASCII character numbers less than 127, excluding the following characters: \ / : * ? “ < > |
- Names cannot be longer than 48 characters
- A path cannot be longer than 126 characters (filename included)

See also ...

- *FTP Server, p. 23*
- *Web Server, p. 25*
- *E-mail Client, p. 32*
- *Server Side Include (SSI), p. 33*
- *Anybus File System Interface Object (0Ah), p. 132*
- *Application File System Interface Object (EAh), p. 146*

! *The file system is located in flash memory. Due to technical reasons, each flash segment can be erased approximately 100000 times before failure, making it unsuitable for random access storage.*

The following operations will erase one or more flash segments:

- Deleting, moving or renaming a file or directory
- Writing or appending data to an existing file
- Formatting the file system

3.9.3 System Files

The file system contains a set of files used for system configuration. These files, known as “system files”, are regular ASCII files which can be altered using a standard text editor (such as the Notepad in Microsoft Windows™). The format of these files are, with some exceptions, based on the concept of keys, where each keys can be assigned a value, see below.

Example 1:

```
[Key1]
value of Key1

[Key2]
value of Key2
```

4 EtherNet/IP Implementation Details

4.1 General Information

This chapter covers EtherNet/IP specific details in the Anybus implementation. Note that the use of such functionality may require in-depth knowledge in EtherNet/IP networking internals and/or information from the official EtherNet/IP and CIP specifications. In such cases, the people responsible for the implementation of this product are expected either to obtain these specifications to gain sufficient knowledge or limit their implementation in such a way that this is not necessary. The EDS file must be changed to reflect all changes.

4.2 EtherNet/IP & CIP Implementation

By default, the module supports the generic CIP profile. Optionally, it is possible to re-route requests to unimplemented CIP objects to the host application, thus enabling support for other profiles etc.

To support a specific profile, perform the following steps:

1. Set up the identity settings in the EtherNet/IP Host Object according to profile requirements.
2. Implement the Assembly Mapping Object in the host application.
3. Set up the Assembly Instance Numbers according to profile requirements.
4. Enable routing of CIP messages to the host application in the EtherNet/IP Host Object.
5. Implement the required CIP objects in the host application.

See also...

- *Using the Assembly Mapping Object (EBh), p. 19*
- *EtherNet/IP Host Object (F8h), p. 150* (Host Application Object), details for the command Process_CIP_Object_Request.

4.3 Using the Assembly Mapping Object (EBh)

4.3.1 Introduction

This guide will describe how to map CIP instances to ADI data, using the assembly mapping object (EBh).

4.3.2 Adding Data - The Application Data Object

According to the Anybus object model, all data that is used in the application must be represented by application data instances (ADIs). ADIs are small portions of structured data, each representing only one of three possible different types: variable, array or structure.

See the Application Data Object (FEh) in the Software Design Guide for more information.

Below is an example with 30 ADIs. Instances 1 - 6 and 30 are implemented in the application, and 7 - 29 are not implemented.

Application Data Object (FEh) Instances

Instance #	Implemented	Order #
1	Yes	1
2	Yes	2
3	Yes	3
4	Yes	4
5	Yes	5
6	Yes	6
7...29	No	-
30	Yes	7

4.3.3 Grouping Data - The Assembly Mapping Object

The assembly mapping object makes it possible to create an arbitrary number of process data sets, called assembly mappings. Each assembly mapping instance represents a different logical set of process data, that can be chosen by the network and received over a single connection.

Every instance of the assembly mapping object, as seen below, contains an ADI map, referring to an arbitrary number of ADIs.

The instance numbers can be set freely.

Assembly Mapping Object (EBh) Instances

Instance #	Type	ADI Map
1	Read	1, 2
2	Read	2, 3
10	Write	3, 4, 30
11	Write	4, 5
30	Read	5, 6
51	Write	6, 30

There are two object instance attributes in the assembly mapping object, called Write PD Instance List and Read PD Instance List. These two attributes contain references to all read instances and all write instances, respectively. The example above will automatically generate the following content in these two attributes.

Name	Attribute	Values
Write PD Instance List	11	10, 11, 51
Read PD Instance List	12	1, 2, 30

The attributes Write PD Instance List and Read PD Instance List adopts the view of the network, e.g. an input will produce data on the network and an output will consume data on the network.

Write PD Instance List will contain all assembly mapping object instances with type “Read”. Read PD Instance List will contain all assembly mapping object instances with type “Write”.

4.3.4 Configuring CIP Assembly Numbers

The read and write instance list attributes in the assembly mapping object are bound to two corresponding attributes in the EtherNet/IP host object, according to the following table.

This routes application data to CIP assembly data, by linking CIP instance numbers to assembly mapping object instances.

The lists are matched index-wise, and must thus be of equal length.

Assembly Mapping Object Attribute	Value		Value	EtherNet/IP Host Object Instance Attribute
11 - Write PD Instance List	10	<—>	10	7 - Producing Instance Number
	11	<—>	22	
	51	<—>	100	
12 - Read PD Instance List	1	<—>	1	8 - Consuming Instance Number
	2	<—>	2	
	30	<—>	150	

For conformity with the CIP specification, both the Write_Assembly_Data and the Read_Assembly_Data services must be implemented.

4.3.5 Going Forward

During the initialization phase, in the NW_INIT state, all write assemblies (e.g. the instances of the assembly mapping object with type“write”) will be remapped to the write process data area. For this to happen, the device will issue the Remap_ADI_Write_Area command to the application data object in the host.

See the appendix about “Runtime Remapping of Process Data” in the Anybus CompactCom 40 Software Design Guide for more information.

When the network has been initialized, the device transitions from NW_INIT to the WAIT_PROCESS state. When the device receives a forward open request, the producing/consuming parameters in the request are verified and matched against the EtherNet/IP Host Object instance numbers (producing/consuming)

If the verification is successful, the read process data is remapped and the device transitions to the PROCESS_ACTIVE state. The I/O connection will then be established, and data can be exchanged over the network.

4.4 Socket Interface (Advanced Users Only)

The built in socket interface allows additional protocols to be implemented on top of TCP/IP.

See also..

- *Socket Interface Object (07h), p. 110* (Anybus Module Object)
- *Message Segmentation, p. 125*

4.5 Diagnostics

The severity value of all pending events are combined (using logical OR) and copied to the corresponding bits in the “Status” attribute of the Identity Object (CIP).

See also...

- *Identity Object (01h), p. 64* (CIP Object)
- *Diagnostic Object (02h), p. 99* (Anybus Module Object)

4.6 QuickConnect

The module supports the QuickConnect functionality. It is enabled in the EtherNet/IP Host Object. The module fulfills Class A with a startup time of less than 180 ms, with 16 bytes of I/O data mapped with parallel, SPI or shift register application interface.

See also ...

- *EtherNet/IP Host Object (F8h), p. 150* (Host Application Object)
- *TCP/IP Interface Object (F5h), p. 88* (CIP object)

4.7 CIP Safety

The Anybus CompactCom 40 EtherNet/IP supports the CIP safety profile. This profile makes it possible for a user to send data on a black channel interface, i.e. a safe channel over EtherNet/IP using an add-on safety module, e.g. the IXXAT Safe T100. For an application to support CIP safety, the Functional Safety Object (E8h, host application object) has to be implemented. The Anybus CompactCom serial channel is used for the functional safety communication. When this channel is used for the host application, a second separate serial channel is implemented for the functional safety communication. See the Anybus CompactCom Hardware Design Guide for more information.

See ...

- *Functional Safety Module Object (11h), p. 137*
- *Functional Safety Object (E8h), p. 144*

4.7.1 Safety Module Firmware Upgrade

The firmware of the connected safety module can be upgraded through the Anybus CompactCom. The safety firmware (hiff file) has to be downloaded to the firmware directory in the Anybus CompactCom. At restart, the Anybus CompactCom detects and validates the firmware. Firmware upgrade in progress is indicated to the application by attribute #5 (instance #1) in the Functional Safety Object (E8h), which is set to TRUE during the firmware upgrade. The Anybus CompactCom will need more time to initialize , please do not restart the module during this time.

4.7.2 Reset Request from Network

When a reset request arrives from the network, a delay of 1 s is introduced before the Anybus CompactCom 40 EtherNet/IP is reset, if CIP safety is enabled.

5 FTP Server

5.1 General Information

The built-in FTP-server makes it easy to manage the file system using a standard FTP client. It can be disabled using attribute #6 in the Ethernet Host Object (F9h).

By default, the following port numbers are used for FTP communication:

- TCP, port 20 (FTP data port)
- TCP, port 21 (FTP command port)

The FTP server supports up to two concurrent clients.

5.2 User Accounts

User accounts are stored in the configuration file \ftp.cfg. This file holds the usernames, passwords, and home directory for all users. Users are not able to access files outside of their home directory.

File Format:

```
User1:Password1:Homedirectory1
User2:Password2:Homedirectory2
User3:Password3:Homedirectory3
```

Optionally, the UserN:PasswordN-section can be replaced by a path to a file containing a list of users as follows:

File Format (\ftp.cfg):

```
User1:Password1:Homedirectory1
User2:Password2:Homedirectory2
.
.
UserN:PasswordN:HomedirectoryN
\path\userlistA:HomedirectoryA
\path\userlistB:HomedirectoryB
```

The files containing the user lists shall have the following format:

File Format:

```
User1:Password1
User2:Password2
User3:Password3
.
.
.UserN:PasswordN
```

Notes:

- Usernames must not exceed 16 characters in length.
- Passwords must not exceed 16 characters in length.
- Usernames and passwords must only contain alphanumeric characters.

- If \ftp.cfg is missing or cannot be interpreted, all username/password combinations will be accepted and the home directory will be the FTP root (i.e. \ftp\).
- The home directory for a user must also exist in the file system, if the user shall be able to log in. It is not enough just to add the user information to the ftp.cfg file.
- If Admin Mode has been enabled in the Ethernet Object, all username/password combinations will be accepted and the user will have unrestricted access to the file system (i.e. the home directory will be the system root). The vfs folder is read-only.
- It is strongly recommended to have at least one user with root access (\) permission. If not, Admin Mode must be enabled each time a system file needs to be altered (including \ftp.cfg).

5.3 Session Example

The Windows Explorer features a built-in FTP client which can easily be used to access the file system as follows:

1. Open the Windows Explorer.
2. In the address field, type FTP://<user>:<password>@<address>
   – - Substitute <address> with the IP address of the Anybus module
   – - Substitute <user> with the username
   – - Substitute <password> with the password
3. Press **Enter**. The Explorer will now attempt to connect to the Anybus module using the specified settings. If successful, the file system will be displayed in the Explorer window.

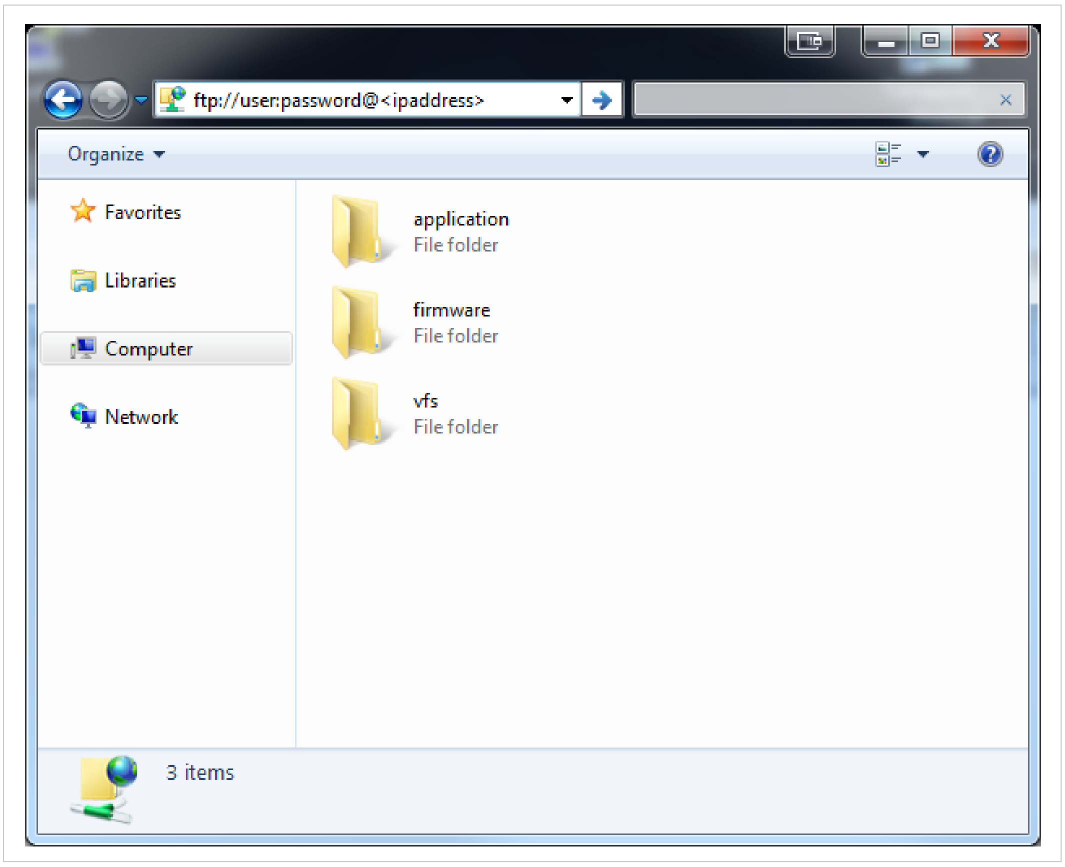


Fig. 3

6 Web Server

6.1 General Information

The built-in web server provides a flexible environment for end-user interaction and configuration purposes. JSON, SSI and client-side scripting allow access to objects and file system data, enabling the creation of advanced graphical user interfaces.

The web interfaces are stored in the file system, which can be accessed through the FTP server. If necessary, the web server can be completely disabled in the Ethernet Host Object (F9h).

See also...

- *FTP Server, p. 23*
- *Server Side Include (SSI), p. 33*
- *JSON, p. 53*
- *Ethernet Host Object (F9h), p. 159*

6.2 Default Web Pages

The default web pages provide access to:

- Network configuration parameters
- Network status information
- Access to the host application ADIs

The default web pages are built of files stored in a virtual file system accessible through the vfs folder. These files are read only and cannot be deleted or overwritten. The web server will first look for a file in the web root folder. If not found it will look for the file in the vfs folder, making it appear as the files are located in the web root folder. By loading files in the web root folder with exactly the same names as the default files in the vfs folder, it is possible to customize the web pages, replacing such as pictures, logos and style sheets.

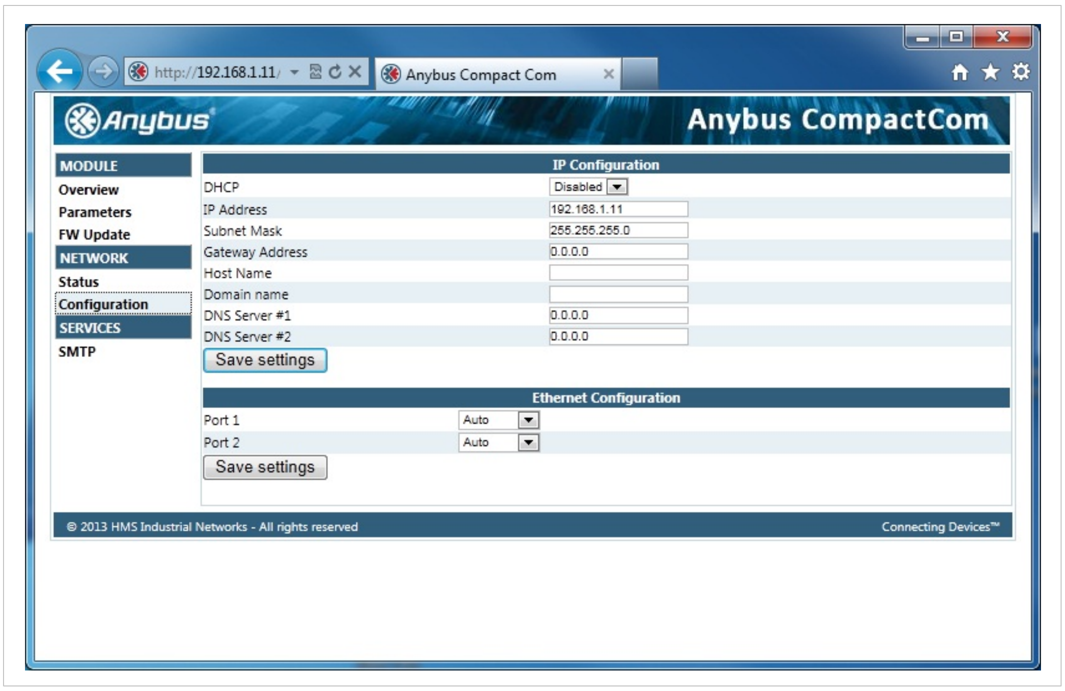
If a complete customized web system is designed and no files in the vfs folder are to be used, it is recommended to turn off the virtual file system completely, see the File System Interface Object.

See also ...

- *File System, p. 16*
- *Anybus File System Interface Object (0Ah), p. 132*

6.2.1 Network Configuration

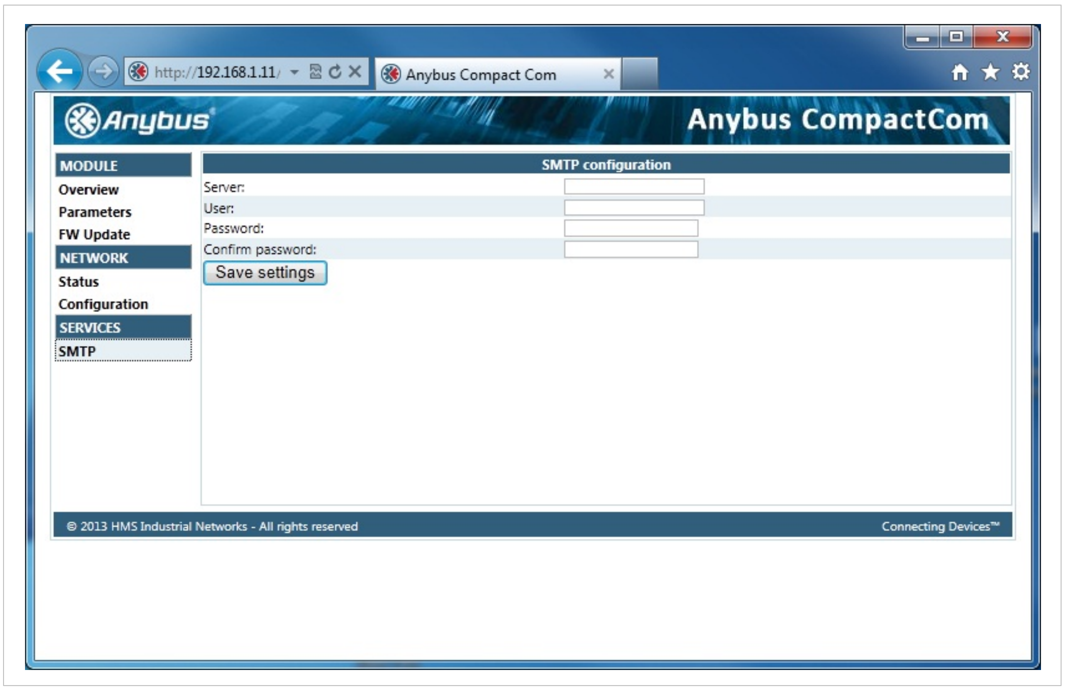
The network configuration page provides interfaces for changing TCP/IP and SMTP settings in the Network Configuration Object.



IP Configuration	
DHCP	Disabled
IP Address	192.168.1.11
Subnet Mask	255.255.255.0
Gateway Address	0.0.0.0
Host Name	
Domain name	
DNS Server #1	0.0.0.0
DNS Server #2	0.0.0.0

Ethernet Configuration	
Port 1	Auto
Port 2	Auto

Fig. 4



SMTP configuration	
Server:	
User:	
Password:	
Confirm password:	

Fig. 5

The module needs to be reset for the TCP/IP and SMTP settings to take effect. The Ethernet Configuration settings will take effect immediately.

IP Configuration

The module needs a reset for any changes to take effect.

Name	Description
DHCP	Enable or disable DHCP Default value: enabled
IP address	The TCP/IP settings of the module Default values: 0.0.0.0 Value ranges: 0.0.0.0 - 255.255.255.255
Subnet mask	
Gateway	
Host name	IP address or name Max 64 characters
Domain name	IP address or name Max 48 characters
DNS 1	Primary and secondary DNS server, used to resolve host name Default values: 0.0.0.0 Value ranges: 0.0.0.0 - 255.255.255.255
DNS 2	

Ethernet Configuration

Changes will take effect immediately.

Name	Description
Port 1	Ethernet speed/duplex settings Default value: auto
Port 2	

SMTP Settings

The module needs a reset before any changes take effect

Name	Description
Server	IP address or name Max 64 characters
User	Max 64 characters
Password	Max 64 characters
Confirm password	

6.2.2 Ethernet Statistics Page

The Ethernet statistics web page contains the following information:

Ethernet Link		Description
Port 1	Speed:	The current link speed.
	Duplex:	The current duplex configuration.
Port 2	Speed:	The current link speed.
	Duplex:	The current duplex configuration.

Ethernet/IP Statistics	Description
Established Class1 Connections	Current number of established class1 connections
Established Class3 Connections	Current number of established class3 connections
Connection Open Requests	Number of received connection open requests
Connection Open Format Rejects	Connection open requests rejected due to request format error
Connection Open Resource Rejects	Connection open requests rejected due to lack of resources
Connection Open Other Rejects	Connection open requests rejected due to other reasons
Connection Close Requests	Number of received connection open requests

Ethernet/IP Statistics	**Description**
Connection Close Format Rejects	Connection close requests rejected du to request format error
Connection Close Other Rejects	Connection close requests rejected due to other reasons
Connection Timeouts	Number of connection timeouts

Interface Counters	**Description**
In Octets:	Received bytes.
In Ucast Packets:	Received unicast packets.
In NUcast packets:	Received non unicast packets (broadcast and multicast).
In Discards:	Received packets discarded due to no available memory buffers.
In Errors:	Received packets discarded due to reception error.
In Unknown Protos:	Received packets with unsupported protocol type.
Out Octets:	Sent bytes.
Out Ucast packets:	Sent unicast packets.
Out NUcast packets:	Sent non unicast packets (broadcast and multicast).
Out Discards:	Outgoing packets discarded due to no available memory buffers.
Out Errors:	Transmission errors.

Media Counters	**Description**
Alignment Errors	Frames received that are not an integral number of octets in length.
FCS Errors	Frames received that do not pass the FCS check.
Single Collisions	Successfully transmitted frames which experienced exactly one collision.
Multiple Collisions	Successfully transmitted frames which experienced more than one collision.
SQE Test Errors	Number of times SQE test error messages are generated. (Not provided with current PHY interface.)
Deferred Transmissions	Frames for which first transmission attempt is delayed because the medium is busy.
Late Collisions	Number of times a collision is detected later than 512 bit-times into the transmission of a packet.
Excessive Collisions	Frames for which a transmission fails due to excessive collisions.
MAC Receive Errors	Frames for which reception of an interface fails due to an internal MAC sublayer receive error.
MAC Transmit Errors	Frames for which transmission fails due to an internal MAC sublayer receive error.
Carrier Sense Errors	Times that the carrier sense condition was lost or never asserted when attempted to transmit a frame.
Frame Size Too Long	Frames received that exceed the maximum permitted frame size.
Frame Size Too Short	Frames received that are shorter than lowest permitted frame size.

6.3 Server Configuration

6.3.1 General Information

Basic web server configuration settings are stored in the system file \http.cfg. This file holds the web server name, root directory for the web interface, content types, and a list of file types which shall be scanned for SSI.

```
File Format:
   [ServerName]
   WebServerName
   [WebRoot]
   \web

   [FileTypes]
   FileType1:ContentType1
   FileType2:ContentType2
   ...
   FileTypeN:ContentTypeN

   [SSIFileTypes]
   FileType1
   FileType2
   ...
   FileTypeN
```

Web Server Name **`[ServerName]`**	Configures the web server name included in the HTTP header of the responses from the module.
Web Root Directory **`[WebRoot]`**	The web server cannot access files outside this directory.
Content Types **`[FileTypes]`**	A list of file extensions and their reported content types. See also... Default content types below
SSI File Types **`[SSIFileTypes]`**	By default, only files with the extension “shtm” are scanned for SSI. Additional SSI file types can be added here as necessary.

The web root directory determines the location of all files related to the web interface. Files outside of this directory and its subdirectories *cannot* be accessed by the web server.

6.3.2 Index page

The module searches for possible index pages in the following order:

1. <WebRoot>\index.htm
2. <WebRoot>\index.html
3. <WebRoot>\index.shtm
4. <WebRoot>\index.wml

Substitute <WebRoot> with the web root directory specified in \http.cfg.

If no index page is found, the module will default to the virtual index file (if enabled).

See also ...

- Default web pages

6.3.3 Default Content Types

By default, the following content types are recognized by their file extension:

File Extension	**Reported Content Type**
htm, html, shtm	text/html
gif	image/gif
jpeg, jpg, jpe	image/jpeg
png	image/x-png
js	application/x-javascript
bat, txt, c, h, cpp, hpp	text/plain
zip	application/x-zip-compressed
exe, com	application/octet-stream
wml	text/vnd.wap.wml
wmlc	application/vnd.wap.wmlc
wbmp	image/vnd.wap.wbmp
wmls	text/vnd.wap.wmlscript
wmlsc	application/vnd.wap.wmlscriptc
xml	text/xml
pdf	application/pdf
css	text/css

Content types can be added or redefined by adding them to the server configuration file.

6.3.4 Authorization

Directories can be protected from web access by placing a file called “web_accs.cfg” in the directory to protect. This file shall contain a list of users that are allowed to access the directory and its subdirectories.

Optionally, a login message can be specified by including the key [AuthName]. This message will be displayed by the web browser upon accessing the protected directory.

```
File Format:
   Username1:Password1
   Username2:Password2
   ...
   UsernameN:PasswordN

   [AuthName]
     (message goes here)
```

The list of approved users can optionally be redirected to one or several other files.

If the list of approved users is put in another file, be aware that this file can be accessed and read from the network.

In the following example, the list of approved users will be loaded from here.cfg and too.cfg.

```
[File path]
\i\put\some\over\here.cfg
\i\actually\put\some\of\it\here\too.cfg

[AuthType]
Basic

[AuthName]
Howdy. Password, please.
```

The field “AuthType” is used to identify the authentication scheme.

Value	Description
Basic	Web authentication method using plaintext passwords.
Digest	More secure method using challenge-response authentication. Used as default if no [Authtype] field is specified.

7 E-mail Client

7.1 General Information

The built-in e-mail client allows the application to send e-mail messages through an SMTP-server. Messages can either be specified directly in the SMTP Client Object (04h), or retrieved from the file system. The latter may contain SSI, however note that for technical reasons, certain commands cannot be used (specified separately for each SSI command).

The client supports authentication using the ‘LOGIN’ method. Account settings etc. are stored in the Network Configuration Object (04h).

7.2 How to Send E-mail Messages

To be able to send e-mail messages, the SMTP-account settings must be specified.

This includes:

- A valid SMTP-server address
- A valid username
- A valid password

To send an e-mail message, perform the following steps:

1. Create a new e-mail instance using the Create command (03h)
2. Specify the sender, recipient, topic and message body in the e-mail instance
3. Issue the Send Instance Email command (10h) towards the e-mail instance
4. Optionally, delete the e-mail instance using the Delete command (04h)

Sending a message based on a file in the file system is achieved using the Send Email from File command. This command is described in the SMTP Client Object (04h).

8 Server Side Include (SSI)

8.1 General Information

Server Side Include functionality, or SSI, allows data from files and objects to be represented on web pages and in e-mail messages.

SSI are special commands embedded within the source document. When the Anybus CompactCom module encounters such a command, it will execute it, and replace it with the result (if applicable).

By default, only files with the extension ‘shtm’ are scanned for SSI.

8.2 Include File

This function includes the contents of a file. The content is scanned for SSI.

This function cannot be used in e-mail messages.

Syntax:

```
<?--#include file="filename"-->
```

filename:	Source file

Scenario	Default Output
Success	(contents of file)

8.3 Command Functions

8.3.1 General Information

Command functions executes commands and includes the result.

General Syntax

```
<?--#exec cmd_argument='command'-->
```

command:	Command function, see below

“command” is limited to a maximum of 500 characters.

Command Functions

Command	Valid for E-mail Messages
GetConfigItem()	Yes
SetConfigItem()	No
SsiOutput()	Yes
DisplayRemoteUser	No
ChangeLanguage()	No
IncludeFile()	Yes
SaveDataToFile()	No

Command	Valid for E-mail Messages
printf()	Yes
scanf()	No

8.3.2 GetConfigItem()

This command returns specific information from a file in the file system.

File Format

The source file must have the following format:

```
[key1]
value1

[key2]
value2
...
[keyN]
valueN
```

Syntax:

```
<?--exec cmd_argument='GetConfigItem("filename", "key"[,"separator"])'-->
```

filename:	Source file to read from
key:	Source [key] in file.
separator:	Optional; specifies line separation characters (e.g. “ ”). (default is CRLF).

Default Output

Scenario	Default Output
Success	*(value of specified key)*
Authentication Error	“Authentication error”
File open error	“Failed to open file ‘*filename*’”
Key not found	“Tag (*key*) not found”

Example

The following SSI...

```
<?--exec cmd_argument='GetConfigItem("\example.cnf", "B")'-->
```

... in combination with the following file ('\example.cnf')...

```
[A]
First
[B]
Second
[C]
Third
```

... returns the string 'Third'.

8.3.3 SetConfigItem()

This function stores an HTML-form as a file in the file system.

This function cannot be used in e-mail messages.

File Format

Each form object is stored as a [tag], followed by the actual value.

```
[form object name 1]
form object value 1

[form object name 2]
form object value 2

[form object name 3]
form object value 3

...
[form object name N]
form object value N
```

Form objects with names starting with underscore will not be stored.

Syntax:

```
<?--exec cmd_argument='SetConfigItem("filename"[, Overwrite])'-->
```

filename:	Destination file. If the specified file does not exist, it will be created (provided that the path is valid).
Overwrite:	Optional; forces the module to create a new file each time the command is issued. The default behavior is to modify the existing file.

Default Output

Scenario	Default Output
Success	"Configuration stored to'*filename*'"
Authentication Error	"Authentication error"
File open error	"Failed to open file '*filename*'"
File write error	"Could not store configuration to '*filename*'"

Example

The following example demonstrates how to use this function. The resulting page sends a form to itself, which is then evaluated by the SetConfigItem command.

```
<HTML>
<HEAD><TITLE>SetConfigItem Test</TITLE></HEAD>
<BODY>

<?--#exec cmd_argument='SetConfigItem("\food.txt")'-->

<FORM action="test.shtm">
   <P>
      <LABEL for="Name">Name: </LABEL><BR>
      <INPUT type="text" name="Name"><BR><BR>

      <LABEL for="_Age">Age: </LABEL><BR>
      <INPUT type="text" name="_Age"><BR><BR>

      <LABEL for="Food">Food: </LABEL><BR>
      <INPUT type="radio" name="Food" value="Cheese"> Cheese<BR>
      <INPUT type="radio" name="Food" value="Sausage"> Sausage<BR><BR>

      <LABEL for="Drink">Drink: </LABEL><BR>
      <INPUT type="radio" name="Drink" value="Wine"> Wine<BR>
      <INPUT type="radio" name="Drink" value="Beer"> Beer<BR><BR>

      <INPUT type="submit" name="_submit">
      <INPUT type="reset" name="_reset">
   </P>
</FORM>

</BODY>
</HTML>
```

The resulting file (‘\food.txt’) may look somewhat as follows:

```
[Name]
Cliff Barnes

[Food]
Cheese

[Drink]
Beer
```

In order for this example to work, the HTML file must be named “test.shtm”.

8.3.4 SsiOutput()

This command temporarily modifies the SSI output of the following command function.

Syntax:

```
<?--#exec cmd_argument='SsiOutput("success", "failure")'-->
```

success:	String to use in case of success
failure:	String to use in case of failure

Default Output

(this command produces no output on its own)

Example

The following example illustrates how to use this command.

```
<?--#exec cmd_argument='SsiOutput ("Parameter stored", "Error")'-->
<?--#exec cmd_argument='SetConfigItem("File.cfg", Overwrite)'-->
```

See also...

- *SSI Output Configuration, p. 52*

8.3.5 DisplayRemoteUser

This command stores returns the username on an authentication session.

This command cannot be used in e-mail messages.

Syntax:

```
<?--#exec cmd_argument='DisplayRemoteUser'-->
```

Default Output

Scenario	Default Output
Success	(current user)

8.3.6 ChangeLanguage()

This command changes the language setting based on an HTML form object.

This function cannot be used in e-mail messages.

Syntax:

```
<?--#exec cmd_argument='ChangeLanguage( "source" )'-->
```

source:	Name of form object which contains the new language setting.

The passed value must be a single digit as follows:

Form value	Language
“0”	English
“1”	German
“2”	Spanish
“3”	Italian
“4”	French

Default Output

Scenario	Default Output
Success	“Language changed”
Error	“Failed to change language”

Example

The following example demonstrates how to use this function. The resulting page sends a form to itself, which is then evaluated by the ChangeLanguage() command.

```
<HTML>
<HEAD><TITLE>ChangeLanguage Test</TITLE></HEAD>
<BODY>

<?--#exec cmd_argument='ChangeLanguage("lang")'-->

<FORM action="test.shtm">
   <P>
      <LABEL for="lang">Language(0-4): </LABEL><BR>
      <INPUT type="text" name="lang"><BR><BR>

      <INPUT type="submit" name="_submit">
   </P>
</FORM>

</BODY>
</HTML>
```

In order for this example to work, the HTML file must be named “test.shtm”.

8.3.7 IncludeFile()

This command includes the content of a file. Note that the content is not scanned for SSI.

Syntax:

```
<?--#exec cmd_argument='IncludeFile("filename" [, separator])'-->
```

filename:	Source file
separator:	Optional; specifies line separation characters (e.g. “ ”).

Default Output

Scenario	Default Output
Success	*(file contents)*
Authentication Error	“Authentication error”
File Open Error	“Failed to open file ‘*filename*’”

Example

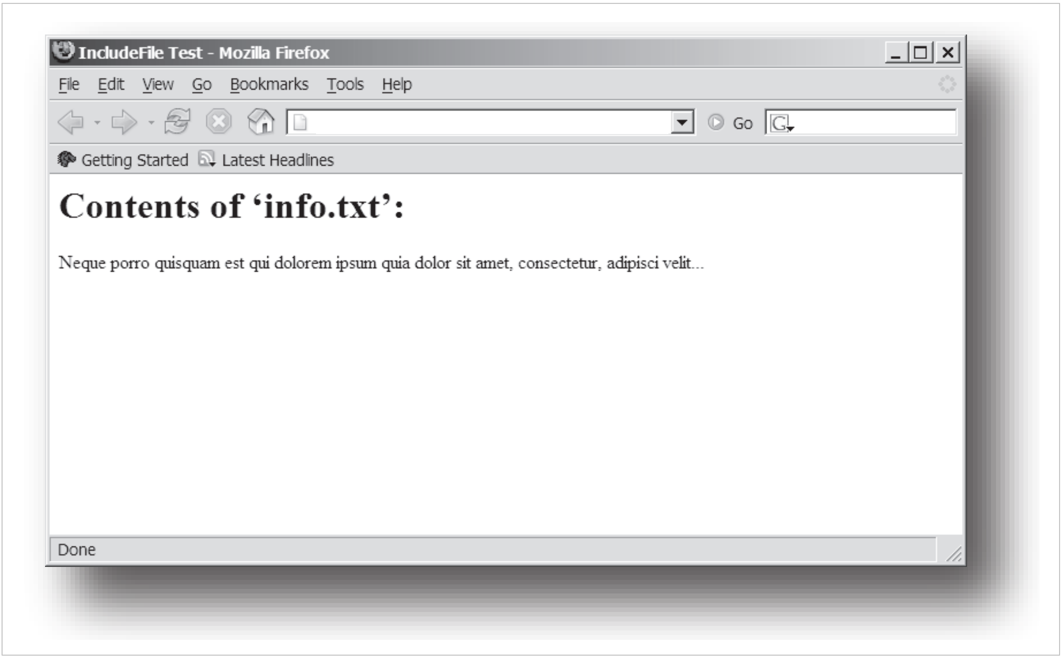
The following example demonstrates how to use this function.

```
<HTML>
<HEAD><TITLE>IncludeFile Test</TITLE></HEAD>
<BODY>
   <H1> Contents of ‘info.txt’:</H1>
   <P>
      <?--#exec cmd_argument='IncludeFile("info.txt")'-->.
   </P>
</BODY>
</HTML>
```

Contents of ‘info.txt’:

```
Neque porro quisquam est qui dolorem ipsum quia dolor sit
amet,consectetur, adipisci velit...
```

When viewed in a browser, the resulting page should look somewhat as follows:


IncludeFile Test - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Getting Started Latest Headlines
Done

Contents of ‘info.txt’:

Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...

Fig. 6

See also...

- *Include File, p. 33*

8.3.8 SaveDataToFile()

This command stores data from an HTML form as a file in the file system. Content from the different form objects are separated by a blank line (2*CRLF).

This function cannot be used in e-mail messages.

Syntax:

```
<?--#exec cmd_argument='SaveDataToFile("filename" [, "source"],
Overwrite|Append)'-->
```

filename	Destination file. If the specified file does not exist, it will be created (provided that the path is valid).
source:	Optional; by specifying a form object, only data from that particular form object will be stored. Default behavior is to store data from all form objects except the ones where the name starts with underscore.
Overwrite|Append	Specifies whether to overwrite or append data to existing files.

Default Output

Scenario	**Default Output**
Success	“Configuration stored to ‘*filename*’”
Authentication Error	“Authentication error”
File Write Error	“Could not store configuration to ‘*filename*’”

Example

The following example demonstrates how to use this function. The resulting page sends a form to itself, which is then evaluated by the SaveDataToFile command.

```
<HTML>
<HEAD><TITLE>SaveDataToFile Test</TITLE></HEAD>
<BODY>

<?--#exec cmd_argument='SaveDataToFile("\stuff.txt", “Meat”, Overwrite)'-->

<FORM action="test.shtm">
   <P>
      <LABEL for="Fruit">Fruit: </LABEL><BR>
      <INPUT type="text" name="Fruit"><BR><BR>

      <LABEL for="Meat">Meat: </LABEL><BR>
      <INPUT type="text" name="Meat"><BR><BR>

      <LABEL for="Meat">Bread: </LABEL><BR>
      <INPUT type="text" name="Bread"><BR><BR>

      <INPUT type="submit" name="_submit">
   </P>
</FORM>

</BODY>
</HTML>
```

The resulting file (\stuff.txt) will contain the value specified for the form object called “Meat”.

In order for this example to work, the HTML file must be named “test.shtm”.

8.3.9 printf()

This function returns a formatted string which may contain data from the Anybus CompactCom module and/or application. The formatting syntax used is similar to that of the standard C-function printf().

The function accepts a template string containing zero or more formatting tags, followed by a number of arguments. Each formatting tag corresponds to a single argument, and determines how that argument shall be converted to human readable form.

Syntax:

```
<?--#exec cmd_argument='printf("template" [, argument1, ..., argumentN])'-->
```

template:	Template which determines how the arguments shall be represented. May contain any number of formatting tags which are substituted by subsequent arguments and formatted as requested. The number of format tags must match the number of arguments; if not, the result is undefined. See section “Formatting Tags” below for more information.
argument:	Source arguments; optional parameters which specify the actual source of the data that shall be inserted in the template string. The number of arguments must match the number of formatting tags; if not, the result is undefined. At the time of writing, the only allowed argument is ABCCMessage(). See also... • *ABCCMessage(), p. 48*

Default Output

Scenario	Default Output
Success	(printf() result)
ABCCMessage error	ABCCMessage error string (*Errors, p. 51*)

Example

See ..

- *ABCCMessage(), p. 48*
- *Example (Get_Attribute):, p. 50*

Formatting Tags

Formatting tags are written as follows:

```
%[Flags][Width][.Precision][Modifier]type
```

- Type (Required)

  The Type-character is required and determines the basic representation as follows:

Type Character	Representation	Example
c	Single character	b
d, i	Signed decimal integer.	565
e, E	Floating-point number in exponential notation.	5.6538e2
f	Floating-point number in normal, fixed-point notation.	565.38
g, G	%e or %E is used if the exponent is less than -4 or greater than or equal to the precision; otherwise %f is used. Trailing zeroes/decimal point are not printed.	565.38
o	Unsigned octal notation	1065
s	String of characters	Text
u	Unsigned decimal integer	4242
x, X	Hexadecimal integer	4e7f
%	Literal %; no assignment is made	%

- Flags (Optional)

Flag Character	Meaning
-	Left-justify the result within the give width (default is right justification)
+	Always include a + or - to indicate whether the number is positive or negative
(space)	If the number does not start with a + or -, prefix it with a space character instead.
0 (zero)	Pad the field with zeroes instead of spaces
#	For %e, %E, and %f, forces the number to include a decimal point, even if no digits follow. For %x and %X, prefixes 0x or 0X, respectively.

- Width (Optional)

Width	Meaning
number	Specifies the minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded to make up the field width. The result is never truncated even if the result is larger.

- Precision (Optional)

  The exact meaning of this field depends on the type character:

Type Character	Meaning
d, i, o, u, x, X	Specifies the minimum no. of decimal digits to be printed. If the value to be printed is shorter than this number, the result is padded with space. Note that the result is never truncated, even if the result is larger.
e, E, f	Specifies the no. of digits to be printed after the decimal point (default is 6).
g, G	Specifies the max. no. of significant numbers to be printed.
s	Specifies the max. no. of characters to be printed
c	(no effect)

- Modifier

Modifier Character	Meaning
hh	Argument is interpreted as SINT8 or UINT8
h	Argument is interpreted as SINT16 or UINT16
L	Argument is interpreted as SINT32 or UINT32

8.3.10 scanf()

This function is very similar to the printf() function described earlier, except that it is used for input rather than output. The function reads a string passed from an HTML form object, parses the string as specified by a template string, and sends the resulting data to the specified argument. The formatting syntax used is similar to that of the standard C-function scanf().

The function accepts a source, a template string containing zero or more formatting tags, followed by a number of arguments. Each argument corresponds to a formatting tag, which determines how the data read from the HTML form shall be interpreted prior sending it to the destination argument.

This command cannot be used in e-mail messages.

Syntax:

```
<?--#exec cmd_argument='scanf("source", "template" [,
                                    argument1, ..., argumentN])'-->
```

source	Name of the HTML form object from which the string shall be extracted.
template:	Template which specifies how to parse and interpret the data. May contain any number of formatting tags which determine the conversion prior to sending the data to subsequent arguments. The number of formatting tags must match the number of arguments; if not, the result is undefined. See section “Formatting Tags” below for more information.
argument:	Destination argument(s) specifying where to send the interpreted data. The number of arguments must match the number of formatting tags; if not, the result is undefined. At the time of writing, the only allowed argument is ABCCMessage(). See also... • *ABCCMessage(), p. 48*

Default Output

Scenario	Default Output
Success	“Success”
Parsing error	“Incorrect data format”
Too much data for argument	“Too much data”
ABCCMessage error	ABCCMessage error string (*Errors, p. 51*)

Example

See also...

ABCCMessage(), p. 48

Example (Set_Attribute):, p. 50

Formatting Tags

Formatting tags are written as follows:

```
%[*][Width][Modifier]type
```

- Type (Required)

  The Type-character is required and determines the basic representation as follows:

Type	Input	Argument Data Type
c	Single character	CHAR
d	Accepts a signed decimal integer	SINT8 SINT16 SINT32
i	Accepts a signed or unsigned decimal integer. May be given as decimal, hexadecimal or octal, determined by the initial characters of the input data: Initial Characters: Format: 0x Hexadecimal 0: Octal 1... 9: Decimal	SINT8/UINT8 SINT16/UINT16 SINT32/UINT32
u	Accepts an unsigned decimal integer.	UINT8 UINT16 UINT32
o	Accepts an optionally signed octal integer.	SINT8/UINT8 SINT16/UINT16 SINT32/UINT32
x, X	Accepts an optionally signed hexadecimal integer.	SINT8/UINT8 SINT16/UINT16 SINT32/UINT32
e, E, f, g, G	Accepts an optionally signed floating point number. The input format for floating-point numbers is a string of digits, with some optional characteristics: – It can be a signed value – It can be an exponential value, containing a decimal rational number followed by an exponent field, which consists of an ‘E’ or an ‘e’ followed by an integer.	FLOAT
n	Consumes no input; the corresponding argument is an integer into which scanf writes the number of characters read from the object input.	SINT8/UINT8 SINT16/UINT16 SINT32/UINT32
s	Accepts a sequence of nonwhitespace characters	STRING
[scanset]	Accepts a sequence of nonwhitespace characters from a set of expected bytes specified by the scanlist (e.g ‘[0123456789ABCDEF]’) A literal ‘]’ character can be specified as the first character of the set. A caret character (^) immediately following the initial ‘[’ inverts the scanlist, i.e. allows all characters except the ones that are listed.	STRING
%	Accepts a single %input at this point; no assignment or conversion is done. The complete conversion specification should be %%.	-

- * (Optional)

  Data is read but ignored. It is not assigned to the corresponding argument.

- Width (Optional)

  Specifies the maximum number of characters to be read

- Modifier (Optional)

  Specifies a different data size.

Modifier	Meaning
h	SINT8, SINT16, UINT8 or UINT16
l	SINT32 or UINT32

8.4 Argument Functions

8.4.1 General Information

Argument functions are supplied as parameters to certain command functions.

General Syntax:

(Syntax depends on context)

Argument Functions:

Function	Description
ABCCMessage()	-

8.4.2 ABCCMessage()

This function issues an object request towards an object in the module or in the host application.

Syntax

```
ABCCMessage(object, instance, command, ce0, ce1,
            msgdata, c_type, r_type)
```

object	Specifies the Destination Object
instance	Specifies the Destination Instance
command	Specifies the Command Number
ce0	Specifies CmdExt[0] for the command message
ce1	Specifies CmdExt[1] for the command message
msgdata	Specifies the actual contents of the MsgData[] subfield in the command • Data can be supplied in direct form (format depends on c_type) • The keyword “ARG” is used when data is supplied by the parent command (e.g. scanf()).
c_type:	Specifies the data type in the command (msgdata), see below.
r_type:	Specifies the data type in the response (msgdata), see below.

Numeric input can be supplied in the following formats:

Decimal (e.g. 50)	(no prefix)
Octal (e.g. 043)	Prefix 0 (zero)
Hex (e.g. 0x1f)	Prefix 0x

- Command Data Types (c_type)

  For types which support arrays, the number of elements can be specified using the suffix [n], where n specifies the number of elements. Each data element must be separated by space.

Type	**Supports Arrays**	**Data format (as supplied in msgdata)**
BOOL	Yes	1
SINT8	Yes	-25
SINT16	Yes	2345
SINT32	Yes	-2569
UINT8	Yes	245
UINT16	Yes	40000
UINT32	Yes	32
CHAR	Yes	A
STRING	No	“abcde” **Note:** Quotes can be included in the string if preceded by back-slash(“\”) Example: “We usually refer to it as \‘the Egg\’”
FLOAT	Yes	5.6538e2
NONE	No	Command holds no data, hence no data type

- Response Data Types (r_type)

  For types which support arrays, the number of elements can be specified using the suffix [n], where n specifies the number of elements.

Type	**Supports Arrays**	**Data format (as supplied in msgdata)**
BOOL	Yes	Optionally, it is possible to exchange the BOOL data with a message based on the value (true or false). In such case, the actual data type returned from the function will be STRING. Syntax: BOOL<true><false> For arrays, the format will be BOOL[n]<true><false>.
SINT8	Yes	-
SINT16	Yes	-
SINT32	Yes	-
UINT8	Yes	This type can also be used when reading ENUM data types from an object. In such case, the actual ENUM value will be returned.
UINT16	Yes	-
UINT32	Yes	-
CHAR	Yes	-
STRING	No	-
ENUM	No	When using this data type, the ABCCMessage() function will first read the ENUM value. It will then issue a ‘Get Enum String’-command to retrieve the actual enumeration string. The actual data type in the response will be STRING.
FLOAT	Yes	-
NONE	No	Response holds no data, hence no data type

! *It is important to note that the message will be passed transparently to the addressed object. The SSI engine performs no checks for violations of the object addressing scheme, e.g. a malformed Get_Attribute request which (wrongfully) includes message data will be passed unmodified to the object, even though this is obviously wrong. Failure to observe this may cause loss of data or other undesired side effects.*

Example (Get_Attribute):

This example shows how to retrieve the IP address using printf() and ABCCMessage().

```
<?--#exec cmd_argument='printf( "%u.%u.%u.%u",
             ABCCMessage(4,3,1,5,0,0,NONE,UINT8[4] ) )'-->
```

Variable	Value	Comments
object	4	Network Configuration Object (04h)
instance	3	Instance #3 (IP address)
command	1	Get_attribute
ce0	5	Attribute #5
ce1	0	-
msgdata	0	-
c_type	NONE	Command message holds no data
r_type	UINT8[4]	Array of 4 unsigned 8-bit integers

Example (Set_Attribute):

This example shows how to set the IP address using scanf() and ABCCMessage(). Note the special parameter value “ARG”, which instructs the module to use the passed form data (parsed by scanf()).

```
<?--#exec cmd_argument='scanf("IP", "%u.%u.%u.%u",
             ABCCMessage(4,3,2,5,0,ARG,UINT8[4],NONE ) )'-->
```

Variable	Value	Comments
object	4	Network Configuration Object (04h)
instance	3	Instance #3 (IP address)
command	2	Set_attribute
ce0	5	Attribute #5
ce1	0	-
msgdata	ARG	Use data parsed by scanf() call
c_type	UINT8[4]	Array of 4 unsigned 8-bit integers
r_type	NONE	Response message holds no data

Errors

In case an object request results in an error, the error code in the response will be evaluated and translated to readable form as follows:

Error Code	Output
0	“Unknown error”
1	“Unknown error”
2	“Invalid message format”
3	“Unsupported object”
4	“Unsupported instance”
5	“Unsupported command”
6	“Invalid CmdExt[0]”
7	“Invalid CmdExt[1]”
8	“Attribute access is not set-able”
9	“Attribute access is not get-able”
10	“Too much data in msg data field”
11	“Not enough data in msg data field”
12	“Out of range”
13	“Invalid state”
14	“Out of resources”
15	“Segmentation failure”
16	“Segmentation buffer overflow”
17... 255	“Unknown error”

See also...

SSI Output Configuration, p. 52

8.5 SSI Output Configuration

Optionally, the SSI output can be permanently changed by adding the file \output.cfg.

File format:

`[ABCCMessage_X]`
`0:"Success string"`
`1:"Error string 1"`
`2:"Error string 2"`
`...`
`16":Error string 16"`
	Each error code corresponds to a dedicated output string, labelled from 1 to 16.
	See *Errors, p. 51*

`[GetConfigItem_X]`
`0: "Success string"`
`1:"Authentication error string"`
`2:"File open error string"`
`3:"Tag not found string"`
	Use "%s" to include the name of the file.

`[SetConfigItem_X]`
`0: "Success string"`
`1:"Authentication error string"`
`2:"File open error string"`
`3:"File write error string"`
	Use "%s" to include the name of the file.

`[IncludeFile_X]`
`0: "Success string"`
`1:"Authentication error string"`
`2:"File read error string"`
	Use "%s" to include the name of the file.

`[scanf_X]`
`0: "Success string"`
`1:"Parsing error string"`
	-

`[ChangeLanguage_X]`
`0: "Success string"`
`1:"Change error string"`
	-

All content above can be included in the file multiple times changing the value "X" in each tag for different languages. The module will then select the correct output string based on the language settings. If no information for the selected language is found, it will use the default SSI output.

Value of X	Language
0	English
1	German
2	Spanish
3	Italian
4	French

See also...

- *SsiOutput(), p. 38*

9 JSON

9.1 General Information

JSON is an acronym for JavaScript Object Notation and an open standard format for storing and exchanging data in an organized and intuitive way. It is used as an alternative to XML, to transmit data objects consisting of attribute - value pairs between a server and a web application. JavaScripts are used to create dynamic web pages to present the values.

JSON is more versatile than SSI in that you not only can change the values on a web page, but also the size and the look of the web page dynamically. A simple example of how to create a web page is added at the end of this chapter.

JSON requests shall be UTF-8 encoded. The module will interpret JSON requests as UTF-8 encoded, while all other HTTP requests will be interpreted as ISO-8859-1 encoded. All JSON responses, sent by the module, are UTF-8 encoded, while all other files sent by the web server are encoded as stored in the file system.

9.1.1 Access

The JSON resources should be password protected. Add password protection by adding a file called web_accs.cfg in the root directory.

9.2 JSON Objects

9.2.1 ADI

info.json

GET adi/info.json[?callback=<function>].

This object holds data common to all ADIs that are static during runtime. Optionally, a callback may be passed to the GET-request for JSONP output.

Name	Data Type	Note
dataformat	Number	0 = Little endian 1 = Big endian (Affects value, min and max representations)
numadis	Number	Total number of ADIs
webversion	Number	Web/JSON API version

JSON object layout:

```
{
   "dataformat": 0,
   "numadis":    123,
   "webversion": 1
}
```

data.json

GET adi/data.json?offset=<offset>&count=<count>[&callback=<function>].

This object call fetches values for up to <count> ADIs, starting from <offset> in a list sorted by ADI order number. The values may change at any time during runtime. Optionally, a callback may be passed to the GET-request for JSONP output.

JSON object layout:

```
[
   "FF",
   "A201",
   "01FAC105"
]
```

metadata.json

GET adi/metadata.json?offset=<offset>&count=<count>[&callback=<function>].

This object call fetches metadata for up to <count> ADIs, starting from <offset> in a list sorted by ADI order number. This data is static during runtime. Optionally, a callback may be passed to the GET-request for JSONP output.

Name	Data Type	Note
instance	Number	-
name	String	May be NULL if no name is present.
numelements	Number	-
datatype	Number	-
min	String	Minimum value. May be NULL if no minimum value is present.
max	String	Maximum value. May be NULL of no maximum value is present.
access	Number	Bit 0: Read accessBit 1: Write access

JSON object layout:

```
[
{
   "instance":    1,
   "name":        "Temperature threshold",
   "numelements": 1,
   "datatype":    0,
   "min":         "00",
   "max":         "FF",
   "access":      0x03
},
{
   nine more...
}
]
```

enum.json

GET adi/enum.json?inst=<instance>[&value=<element>][&callback=<function>].

This object call fetches enum strings for the instance <instance>. If an <element> is specified, only the enum string for that value is returned. If no enum strings are available, an empty list is returned. Optionally, a callback may be passed to the GET-request for JSONP output.

Name	Data Type	Note
string	String	-
value	Number	-

JSON object layout:

```
[
   {
      "string": "String for value 1",
......"value": 1
   },
   {
      "string": "String for value 1",
......"value": 1
...},
   ...
]
```

update.json

POST adi/update.json - form data:

inst=<instance>&value=<data>[&elem=<element>][&callback=<function>].

Updates the value of an ADI for the specified ADI instance <instance>. The value, <data>, shall be hex formatted (see *Hex Format Explained, p. 61* for more information). If <element> is specified, only the value of the specified element is updated. In this case, <data> shall only update that single element value. When <element> is not specified, <data> shall represent the entire array value. Optionally, a callback may be passed to the request for JSONP output

Name	Data Type	Note
result	Number	0 = success

POST adi/update.json - form data: inst=15&value=FF01

```
{
   "result" : 0
}
```

9.2.2 Module

info.json

GET module/info.json

Name	Data Type	Note
modulename	String	-
serial	String	32 bit hex ASCII
fwver	Array of Number	(major, minor, build)
uptime	Array of Number	[high, low] milliseconds (ms)
cpuload	Number	CPU load in %

JSON object layout:

```
{
   "modulename":  "ABCC M40",
   "serial":      "ABCDEF00",
   "fwver":       [ 1, 5, 0 ],
   "uptime":      [ 5, 123456 ],
   "cpuload":     55
}
```

9.2.3 Network

ethstatus.json

GET network/ethstatus.json.

Name	Data Type	Note
mac	String	6 byte hex
comm1	Object	See object definition in the table below
comm2	Object	See object definition in the table below

Comm Object Definition:

Name	Data Type	Note
link	Number	0: No link 1: Link
speed	Number	0: 10 Mbit 1: 100 Mbit
duplex	Number	0: Half 1: Full

JSON object layout:

```
{
   "mac":          "003011FF0201",
   "comm1":        {
      "link":        1,
      "speed":       1,
      "duplex":      1
   },
   "comm2":        {
      "link":        1,
      "speed":       1,
      "duplex":      1
...}
}
```

ipstatus.json & ipconf.json

These two object share the same data format. The object ipconf.json returns the configured IP settings, and ipstatus.json returns the actual values that are currently used. ipconf.json can also be used to alter the IP settings.

GET network/ipstatus.json, or GET network/ipconf.json.

Name	Data Type	Note
dhcp	Number	-
addr	String	-
subnet	String	-
gateway	String	-
dns1	String	-
dns2	String	-
hostname	String	-
domainname	String	-

```
{
   "dhcp":       0,
   "addr":       "192.168.0.55",
   "subnet":     "255.255.255.0",
   "gateway":    "192.168.0.1",
   "dns1":       "10.10.55.1",
   "dns2":       "10.10.55.2"
   "hostname":   "<hostname>",
   "domainname": "hms.se"
}
```

To change IP settings, use network/ipconf.json. It accepts any number of arguments from the list above. Values should be in the same format.

Example:

GET ipconf.json?dhcp=0&addr=10.11.32.2&hostname=abcc123&domainname=hms.se

ethconf.json

GET network/ethconf.json

Name	Data Type	Note
comm1	Number	-
comm2	Number	-

ifcounters.json

GET network/ifcounters.json?port=<port>. Valid values for the argument <port> are 0, 1, and 2.

- Valid values for the argument <port> are 0, 1, and 2.
- Port number 2 option is only valid if two Ethernet ports are activated in the module.
- Port number 0 option refers to the internal port (CPU port).

Name	Data Type	Note
inoctets	Number	IN: bytes
inucast	Number	IN: unicast packets
innucast	Number	IN: broadcast and multicast packets
indiscards	Number	IN: discarded packets
inerrors	Number	IN: errors
inunknown	Number	IN: unsupported protocol type
outoctets	Number	OUT: bytes
outucast	Number	OUT: unicast packets
outnucast	Number	OUT: broadcast and multicast packets
outdiscards	Number	OUT: discarded packets
outerrors	Number	OUT: errors

mediacounters.json

GET network/mediacounters.json?port=<port>. The argument <port> is either 1 or 2.

Name	Data Type	Note
align	Number	Frames received that are not an integral number of octets in length
fcs	Number	Frames received that do not pass the FCS check
singlecoll	Number	Successfully transmitted frames which experienced exactly one collision
multicoll	Number	Successfully transmitted frames which experienced more than one collision
latecoll	Number	Number of collisions detected later than 512 bit times into the transmission of a packet
excesscoll	Number	Frames for which transmissions fail due to excessive collisions
sqetest	Number	Number of times SQE test error is generated
deferredtrans	Number	Frames for which the first transmission attempt is delayed because the medium is busy
macrecerr	Number	Frames for which reception fails due to an internal MAC sublayer receive error
mactranserr	Number	Frames for which transmission fails due to an internal MAC sublayer transmit error
cserr	Number	Times that the carrier sense was lost or never asserted when attempting to transmit a frame
toolong	Number	Frames received that exceed the maximum permitted frame size
tooshort	Number	Frames received that are shorter than the lowest permitted frame size

nwstats.json

GET network/nwstats.json.

This object lists available statistics data. The data available depends on the product.

Example output:

```
[]
or
[ { "identifier": "eip", "title": "EtherNet/IP Statistics" } ]
or
[
   { "identifier": "bacnet",    "title": "BACnet/IP Statistics" },
   { "identifier": "bacnetae",  "title": "BACnet Alarm and Event" },
   { "identifier": "bacnetapl", "title": "BACnet APL Statistics" }
]
```

Get network specific statistics:

GET network/nwstats.json?get=<ID>. <ID> is an “identifier” value returned from the previous command (“eip”, for example)

```
[
   { "name": "Established Class1 Connections", "value": 0 },
   { "name": "Established Class3 Connections", "value": 1 }
]
```

9.2.4 Services

smtp.json

GET services/smtp.json.

Password is not returned when retrieving the settings.

Name

Data Type

Note

server

String

-

user

String

-

9.2.5 Hex Format Explained

The metadata max and min fields and the ADI values are ABP data encoded in a hex format. If the data type is an integer, the endianness used is determined by the data format field found in adi/info.json.

Examples:

The value 5 encoded as a UINT16, with data format = 0 (little endian):

```
0500
```

The character array “ABC” encoded as CHAR[3] (data format is not relevant for CHAR):

```
414243
```

9.3 Example

This example shows how to create a web page that fetches Module Name and CPU load from the module and presents it on the web page. The file, containing this code, has to be stored in the built-in file system, and the result can be seen in a common browser.

```
<html>
   <head>
      <title>Anybus CompactCom</title>

      <!-- Imported libs -->
      <script type="text/javascript" src="vfs/js/jquery-1.9.1.js"></script>
      <script type="text/javascript" src="vfs/js/tmpl.js"></script>
   </head>
   <body>
      <div id="info-content"></div>
      <script type="text/x-tmpl" id="tmpl-info">
         <b>From info.json</b><br>
         Module name:
         {%=o.modulename%}<br>

         CPU Load:
         {%=o.cpuload%}%<br>
      </script>
      <script type="text/javascript">
         $.getJSON( "/module/info.json", null, function(data){
            $("#info-content").html( tmpl("tmpl-info", data ) );
         });
      </script>
   </body>
</html>
```

10 CIP Objects

10.1 General Information

This chapter specifies the CIP-object implementation in the module. These objects can be accessed from the network, but not directly by the host application.

Mandatory objects

- *Identity Object (01h), p. 64*
- *Message Router (02h), p. 67*
- *Assembly Object (04h), p. 68*
- *Connection Manager (06h), p. 71*
- *QoS Object (48h), p. 79*
- *TCP/IP Interface Object (F5h), p. 88*
- *Ethernet Link Object (F6h), p. 92*

CIP Energy Objects:

- *Base Energy Object (4Eh), p. 80*
- *Power Management Object (53h), p. 82*

Optional Objects:

- *Port Object (F4h), p. 86* (Optional)
- *Parameter Object (0Fh), p. 75*
- *DLR Object (47h), p. 78*

Vendor Specific Objects:

- *ADI Object (A2h), p. 84*

It is possible to implement additional CIP-objects in the host application using the CIP forwarding functionality, see *EtherNet/IP Host Object (F8h), p. 150* and commend details for Process_CIP_Object_Request.

Unconnected CIP routing is supported, which means that a message can be sent to a device without first setting up a connection.

10.2 Translation of Status Codes

If an error occurs when an object is requested from the application, an error code is returned. These Anybus CompactCom error codes are translated to CIP status codes according to the table below.

Anybus CompactCom 40 Error Code		**CIP Status Code**	
Value	**Error**	**Value**	**Status**
00h	Reserved	1Eh	Embedded service error
01h	Reserved	1Eh	Embedded service error
02h	Invalid message format	1Eh	Embedded service error
03h	Unsupported object	05h	Path destination unknown
04h	Unsupported instance	05h	Path destination unknown
05h	Unsupported Command	08h	Service not supported
06h	Invalid CmdExt(0)	14h	Depending on Anybus CompactCom Service returning this reply, e.g. attribute not supported
07h	Invalid CmdExt(1)	-	Depending on Anybus CompactCom Service returning this reply
08h	Attribute not settable	0Eh	Attribute not settable
09h	Attribute not gettable	2Ch	Attribute not gettable
0Ah	Too Much Data	15h	Too much data
0Bh	Not Enough Data	13h	Not enough data
0Ch	Out of range	09h	Invalid attribute value
0Dh	Invalid state	0Ch	Object state conflict
0Eh	Out of resources	02h	Resource unavailable
0Fh	Segmentation failure	1Eh	Embedded service error
10h	Segmentation buffer overflow	23h	Buffer overflow
11h	Value too high	09h	Invalid attribute value
12h	Value too low	09h	Invalid attribute value
13h	Attribute controlled	0Fh	A permission/privilege check failed
14h	Message channel too small	11h	Reply data too large
FFh	Object Specific Error	1Fh	Vendor specific error. No additional error codes will be sent on EtherNet/IP
Other	-	1Eh	Embedded service error

For further information about the Anybus CompactCom error codes, please consult the Anybus CompactCom 40 Software Design Guide.

10.3 Identity Object (01h)

Category

Extended

Object Description

The Identity Object provides identification of and general information about the module.

The object supports multiple instances. Instance 1, which is the only mandatory instance, describes the whole product. It is used by applications to determine what nodes are on the network and to match an EDS file with a product on the network. The other (optional) instances describe different parts of the product, e.g. the software.

If modular device functionality is enabled, a list of the modules in the slots can be retrieved and made available to the network master by sending a get request to class attribute 100.

Instance attributes 1 - 7 can be customized by implementing the EtherNet/IP Host Object.

Additional identity instances can be registered by implementing the CIP Identity Host Object (host application object).

See also ....

- *EtherNet/IP Host Object (F8h), p. 150*
- *CIP Identity Host Object (EDh), p. 147*

Supported Services

Class:	Get_Attribute_Single
	Get_Attributes_All
Instance:	Get_Attribute_Single
	Set_Attribute_Single
	Get_Attributes_All
	Reset

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0001h (Object revision)
2	Max instance	Get	UINT	Maximum instance number
3	Number of instances	Get	UINT	Number of instances
100	Module ID List	Get	Array of UINT32	If modular device functionality is enabled, a request to this attribute will generate a Get_List request to the Modular Device Object in the host application.

Instance Attributes

Attributes #1–4 and #6–7 can be customized by implementing the EtherNet/IP Host Object, see *EtherNet/IP Host Object (F8h), p. 150*

#	Name	Access	Type	Value/Description
1	Vendor ID	Get	UINT	005Ah (HMS Industrial Networks AB)
2	Device Type	Get	UINT	002Bh (Generic Device)
3	Product Code	Get	UINT	0037h (Anybus CompactCom 40 EtherNet/IP)
4	Revision	Get	Struct of: USINT USINT	Major and minor firmware revision
5	Status	Get	WORD	See Device Status table below
6	Serial Number	Get	UDINT	Unique serial number (assigned by HMS)
7	Product Name	Get	SHORT_ STRING	“Anybus CompactCom 40 EtherNet/IP (TM)”
11	Active language	Set	Struct of: USINT USINT USINT	Requests sent to this instance are forwarded to the Application Object. If the request is accepted, the module will update the language accordingly.
12	Supported Language List	Get	Array of: Struct of: USINT USINT USINT	List of languages supported by the host application. The list is read from the Application Object and translated to CIP standard. By default the only supported language is English. The application has to implement the corresponding attributes in the application object to enable more languages.

Device Status

bit(s)	Name
0	Module Owned
1	(reserved)
2	Configured This bit shows if the product has other settings than "out-of-box". The value is set to true if the configured attribute in the Application Object is set and/or the module's NV storage is changed from default.
3	(reserved)
4... 7	Extended Device Status: Value: Meaning: 0000b Unknown 0010b Faulted I/O Connection 0011b No I/O connection established 0100b Non volatile configuration bad 0101b Major fault 0110b Connection in Run mode 0111b Connection in Idle mode (other) (reserved)
8	Set for minor recoverable faults. See *Diagnostic Object (02h), p. 99*
9	Set for minor unrecoverable faults. See *Diagnostic Object (02h), p. 99*
10	Set for major recoverable faults. See *Diagnostic Object (02h), p. 99*
11	Set for major unrecoverable faults. See *Diagnostic Object (02h), p. 99*
12... 15	(reserved)

Service Details: Reset

This service is not supported if safety is enabled in the Functional Safety Object (E8h).

The module forwards reset requests from the network to the host application. For more information about network reset handling, consult the general Anybus CompactCom 40 Software Design Guide.

There are two types of network reset requests on EtherNet/IP:

Type 0: Power Cycling Reset	This service emulates a power cycling of the module, and corresponds to Anybus reset type 0 (Power cycling). For further information, consult the general Anybus CompactCom 40 Software Design Guide.
Type 1: Out of box reset	This service sets a “out of box” configuration and performs a reset, and corresponds to Anybus reset type 2 (Power cycling + factory default). For further information, consult the general Anybus CompactCom 40 Software Design Guide.

10.4 Message Router (02h)

Category

Extended

Object Description

The Message Router Object provides a messaging connection point through which a client may address a service to any object class or instance residing in the physical module.

In the Anybus CompactCom module it is used internally to direct object requests.

Supported Services

Class:	-
Instance:	-

Class Attributes

-

Instance Attributes

-

10.5 Assembly Object (04h)

Category

Extended

Object Description

The Assembly object uses static assemblies and holds the Process Data sent/received by the host application. It allows data to and from each object to be sent or received over a single connection. The default assembly instance IDs used are in the vendor specific range.

It is possible for the application to create and support up to six consuming and six producing instances if the Assembly Mapping Object is implemented.

The terms “input” and “output” are defined from the network’s point of view. An input will produce data on the network and an output will consume data from the network.

See also ....

- *EtherNet/IP Host Object (F8h), p. 150*
- Assembly Mapping Object (see Anybus CompactCom 40 Software Design Guide)

Supported Services

Class:	Get_Attribute_Single
Instance:	Get_Attribute_Single
	Set_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0002h (Object revision)
2	Max instance	Get	UINT	Maximum instance number

Instance 03h Attributes (Heartbeat, Input-Only)

This instance is used as heartbeat for Input-Only connections. The data size of the Heartbeat instance in the Forward_Open-request should be 0 bytes, however other values are also permitted.

#	Name	Access	Type	Value/Description
3	Data	Set	N/A	- (The data size of this attribute is zero)
4	Size	Get	UINT	0 (Number of bytes in attribute 3)

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

Instance 04h Attributes (Heartbeat, Listen-Only)

This instance is used as heartbeat for listen-only connections. The data size of the Heartbeat instance in the Forward_Open-request should be 0 bytes, however other values are also permitted.

#	Name	Access	Type	Value/Description
3	Data	Set	N/A	- (The data size of this attribute is zero)
4	Size	Get	UINT	0 (Number of bytes in attribute 3)

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

Instance 05h Attributes (Configuration Data)

Configuration Data that is sent through the service Forward_Open will be written to this instance.

#	Name	Access	Type	Value/Description
3	Data	Set	N/A	- (Configuration data written to the application when the forward open command has the configuration data included)- (The data size of this attribute is zero)
4	Size	Get	UINT	0 (Number of bytes in attribute 3)

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

See command details for Set_Configuration_Data nad Get_Contfiguration_Data in the *EtherNet/IP Host Object (F8h), p. 150*.

Instance 06h Attributes (Heartbeat, Input-Only Extended)

This instance is used as heartbeat for input-only extended connections, and does not carry any attributes. The state of connections made to this instance does not affect the state of the Anybus CompactCom module, i.e. if the connection times out, the module does not switch to the Error state. The data size of the Heartbeat instance in the Forward_Open-request should be 0 bytes, however other values are also permitted.

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

#	Name	Access	Type	Value/Description
3	Data	Set	N/A	- (The data size of this attribute is zero)
4	Size	Get	UINT	0 (Number of bytes in attribute 3)

Instance 07h Attributes (Heartbeat, Listen-Only Extended)

This instance is used as heartbeat for listen-only extended connections, and does not carry any attributes. The state of connections made to this instance does not affect the state of the Anybus CompactCom 40 module, i.e. if the connection times out, the module does not switch to the Error state. The data size of the Heartbeat instance in the Forward_Open-request should be 0 bytes, however other values are also permitted.

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

#	Name	Access	Type	Value/Description
3	Data	Set	N/A	- (The data size of this attribute is zero)
4	Size	Get	UINT	0 (Number of bytes in attribute 3)

Instance 64h Attributes (Producing Instance)

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

#	Name	Access	Type	Value/Description
3	Produced Data	Get	Array of BYTE	This data corresponds to the Write Process Data.
4	Size	Get	UINT	Number of bytes in attribute 3

See also...

Network Data Exchange, p. 14

EtherNet/IP Host Object (F8h), p. 150(Instance attribute #7)

Instance 96h Attributes (Consuming Instance)

The instance number for this instance can be changed by implementing the corresponding attribute in the EtherNet/IP Host Object.

#	Name	Access	Type	Value/Description
3	Produced Data	Set	Array of BYTE	This data corresponds to the Read Process Data.
4	Size	Get	UINT	Number of bytes in attribute 3

See also...

Network Data Exchange, p. 14

EtherNet/IP Host Object (F8h), p. 150 (Instance attribute #7)

10.6 Connection Manager (06h)

Category

Extended

Object Description

This object is used for connection and connectionless communications, including establishing connections across multiple subnets.

Supported Services

Class:	-
Instance:	Get Attribute All
	Get Attribute Single
	Set Attribute Single
	Large_Forward_Open
	Forward_Open
	Forward_Close
	Unconnected Send (when unconnected routing is enabled)

Class Attributes

(No supported class attributes)

Instance Attributes

#	Name	Access	Type	Value/Description
1	Open Requests	Set	UINT	Number of Forward Open service requests received.
2	Open Format Rejects	Set	UINT	Number of Forward Open service requests which were rejected due to bad format.
3	Open Resource Rejects	Set	UINT	Number of Forward Open service requests which were rejected due to lack of resources.
4	Open Other Rejects	Set	UINT	Number of Forward Open service requests which were rejected for reasons other than bad format or lack of resources.
5	Close Requests	Set	UINT	Number of Forward Close service requests received.
6	Close Format Rejects	Set	UINT	Number of Forward Close service requests which were rejected due to bad format.
7	Close Other Rejects	Set	UINT	Number of Forward Close service requests which were rejected for reasons other than bad format.
8	Connection Timeouts	Set	UINT	Total number of connection timeouts that have occurred in connections controlled by this Connection Manager.

Class 0 Connection Details

General

Class 0 connections are only supported for safety connections. The Anybus CompactCom device will act as a transparent bridge for safety connections, meaning that open and close requests for safety connections and safety I/O data will be forwarded to the safety module. Class 0 connections use UDP transport.

Total number of supported class 0 connections:	2
Max input connection size:	241 bytes (Including the Mode Byte, Actual, Complement and Time stamp sections.)
Max output connection size:	239 bytes (Including the Mode Byte, Actual, Complement and Time stamp sections.)
Supported RPI (Requested Packet Interval):	1... 20000 ms

Class 1 Connection Details

General

Class 1 connections are used to transfer I/O data, and can be established to instances in the Assembly Object. Each Class 1 connection will establish two data transports; one consuming and one producing. The heartbeat instances can be used for connections that shall only access inputs. Class 1 connections use UDP transport. Null forward open is supported.

Total number of supported class 1 connections:	4
Max input connection size:	1448 bytes with Large_Forward_Open, 509 bytes with Forward_Open
Max output connection size:	1448 bytes with Large_Forward_Open, 505 bytes with Forward_Open
Supported RPI (Requested Packet Interval):	1... 3200ms
T→O Connection type:	Point-to-point, Multicast, Null
O→-T Connection type:	Point-to-point, Null
Supported trigger types:	Cyclic, CoS (Change of State)
Supported priorities:	Low, High, Scheduled, Urgent

T	Target, in this case the module
O	Origin, in this case the master

Connection Types

- **Exclusive-Owner connection**

  This type of connection controls the outputs of the Anybus module and does not depend on other connections.

Max. no. of Exclusive-Owner connections:	1
Connection point O →T:	Assembly Object, instance 96h (Default)
Connection point T →O:	Assembly Object, instance 64h (Default)

- **Input-Only connection**

  This type of connection is used to read data from the Anybus module without controlling the outputs. It does not depend on other connections.

Max. no. of Input-Only connections:	Up to 4 (Shared with Exclusive-Owner and Listen-Only connections)
Connection point O →T:	Assembly Object, instance 03h (Default)
Connection point T →O:	Assembly Object, instance 64h (Default)

  Please not that if an Exclusive-Owner connection has been opened towards the module and times out, the Input-Only connection times out as well. If the Exclusive-Owner connection is properly closed, the Input-Only connection remains unaffected.

- **Input-Only Extended connection**

  This connections functionality is the same as the standard Input-Only connection. However when this connection times out it does not affect the state of the application.

Connection point O →T:	Assembly Object, instance 06h (Default)
Connection point T →O:	Assembly Object, instance 64h (Default)

- **Listen-Only connection**

  This type of connection requires another connection in order to exist. If that connection (Exclusive-Owner or Input-Only) is closed, the Listen-Only connection will be closed as well.

Max. no. of Input-Only connections:	Up to 4 (Shared with Exclusive-Owner and Input-Only connections)
Connection point O →T:	Assembly Object, instance 04h (Default)
Connection point T →O:	Assembly Object, instance 64h (Default)

- **Listen-Only Extended connection**

  This connections functionality is the same as the standard Listen-Only connection. However when this connection times out it does not affect the state of the application.

Connection point O →T:	Assembly Object, instance 07h (Default)
Connection point T →O:	Assembly Object, instance 64h (Default)

- **Redundant-Owner connection**

  This connection type is not supported by the module.

Class 3 Connection Details

General

Class 3 connections are used to establish connections towards the message router. Thereafter, the connection is used for explicit messaging. Class 3 connections use TCP transport.

No. of simultaneous Class 3 connections:	6
Supported RPI (Requested Packet Interval):	1... 10000 ms
T→O Connection type:	Point-to-point
O→-T Connection type:	Point-to-point
Supported trigger type:	Application
Supported connection size:	1448 bytes

10.7 Parameter Object (0Fh)

Category

Extended

Object Description

The Parameter Object provides an interface to the configuration data of the module. It can provide all the information necessary to define and describe each of the module configuration parameters, as well as a full description of each parameter, including minimum and maximum values and a text string describing the parameter. Configuration tools, such as RSNetworx, can extract information about the Application Data Instances (ADIs) and present them with their actual name and range to the user.

Since this process may be somewhat time consuming, especially when using the serial host interface, it is possible to disable support for this functionality in the EtherNet/IP Host Object.

Each parameter is represented by one instance. Instance numbers start at 1, and are incremented by one, with no gaps in the list. Due to limitations imposed by the CIP standard, ADIs containing multiple elements (i. e. arrays etc.) cannot be represented through this object. In such cases, default values will be returned.

See also ....

- *ADI Object (A2h), p. 84* (CIP Object)
- *EtherNet/IP Host Object (F8h), p. 150* (Host Application Object)

Supported Services

Class:	Get_Attribute_Single
Instance:	Get_Attribute_Single
	Set_Attribute_Single
	Get_Attributes_All
	Get_Enum_String

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0001h (Object revision)
2	Max instance	Get	UINT	Maximum created instance number = class attribute 3 in the Application Data Object (see Anybus CompactCom 40 Software Design Guide)
8	Parameter Class Descriptor	Get	WORD	Default: 0000 0000 0000 1011b Bit: Contents: 0 Supports parameter instances 1 Supports full attributes 2 Must do non-volatile storage save command 3 Parameters are stored in non-volatile storage
9	Configuration Assembly Instance	Get	UINT	0000h (Application does not support configuration data) 0005h (If the application supports configuration data, unless the configuration instance number has been changed using attribute 15 in the EtherNet/IP Host Object.)

Instance Attributes

#	Name	Access	Type	Value/Description
1	Parameter Value	Get/Set	Specified in attributes 4, 5 & 6.	Actual value of parameter This attribute is read-only if bit 4 of Attribute #4 is true
2	Link Path Size	Get	USINT	0007h (Size of link path in bytes)
3	Link Path	Get	Packed EPATH	20 A2 25 nn nn 30 05h (Path to the object from where this parameter's value is retrieved, in this case the ADI Object)
4	Descriptor	Get	WORD	Bit: Contents: 0 Supports Settable Path (N/A) 1 Supports Enumerated Strings 2 Supports Scaling (N/A) 3 Supports Scaling Links (N/A) 4 Read only Parameter 5 Monitor Parameter (N/A) 6 Supports Extended Precision Scaling (N/A)
5	Data Type	Get	USINT	Data type code
6	Data Size	Get	USINT	Number of bytes in parameter value
7	Parameter Name String	Get	SHORT_ STRING	Name of the parameter, truncated to 16 chars
8	Units String	Get	SHORT_ STRING	“” (default string)
9	Help String	Get	SHORT_ STRING	
10	Minimum Value	Get	(Data type)	Minimum value of parameter The Data Type is defined in attribute 5.
11	Maximum Value	Get	(Data type)	Maximum value of parameter The Data Type is defined in attribute 5.
12	Default Value	Get	(Data type)	Default value of parameter The Data Type is defined in attribute 5.
13	Scaling Multiplier	Get	UINT	0001h
14	Scaling Divisor	Get	UINT	
15	Scaling Base	Get	UINT	
16	Scaling Offset	Get	INT	0000h
17	Multiplier Link	Get	UINT	
18	Divisor Link	Get	UINT	
19	Base Link	Get	UINT	
20	Offset Link	Get	UINT	
21	Decimal Precision	Get	USINT	00h

Default Values

#	Name	Value	Comments
1	Parameter Value	0	-
2	Link Path Size	0	Size of link path in bytes.
3	Link Path	-	NULL Path
4	Descriptor	0010h	Read only Parameter
5	Data type	C6h	USINT
6	Data size	1	-
7	Parameter Name String	“Reserved”	-
8	Units String	“”	-
9	Help String	“”	-
10	Minimum value	N/A	0
11	Maximum value	N/A	0
12	Default value	N/A	0
13	Scaling Multiplier	N/A	1
14	Scaling Divisor	N/A	1
15	Scaling Base	N/A	1
16	Scaling Offset	N/A	0
17	Multiplier Link	N/A	0
18	Divisor Link	N/A	0
19	Base Link	N/A	0
20	Offset Link	N/A	0
21	Decimal Precision	N/A	0

10.8 DLR Object (47h)

Category

Extended

Object Description

The Device Level Ring (DLR) Object provides the status information interface for the DLR protocol. This protocol enables the use of an Ethernet ring topology, and the DLR Object provides the CIP application-level interface to the protocol.

This object is not available if DLR is disabled in the EtherNet/IP Host Object, see *Ethernet Host Object (F9h), p. 159*

.

Supported Services

Class:	Get_Attribute_Single
	Get_Attributes_All
Instance:	Get_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0003h (Object revision)

Instance Attributes

Attributes #1–4 and #6–7 an be customized by implementing the EtherNet/IP Host Object, see *EtherNet/IP Host Object (F8h), p. 150*

#	Name	Access	Type	Value/Description
1	Network Topology	Get	USINT	Bit: Contents: 0 "Linear" 1 "Ring"
2	Network Status	Get	USINT	Bit: Contents: 0 "Normal" (N/A) 1 "Ring Fault" 2 "Unexpected Loop Detected" 3 "Partial Network Fault" 4 "Rapid Fault/Restore Cycle"
10	Active Supervisor Address	Get	Struct of: UDINT Array of: 6 USINTs	This attribute holds the IP address (IPv4) and/or the Ethernet Mac address of the active ring supervisor.
12	Capability Flags	Get	DWORD	82h (Beacon-based ring node, Flush_Table frame capable)

10.9 QoS Object (48h)

Category

Extended

Object Description

Quality of Service (QoS) is a general term that is applied to mechanisms used to treat traffic streams with different relative priorities or other delivery characteristics. Standard QoS mechanisms include IEEE 802.1D/Q (Ethernet frame priority) and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

The QoS Object provides a means to configure certain QoS related behaviors in EtherNet/IP devices.

The object is required for devices that support sending EtherNet/IP messages with nonzero DiffServ code points (DSCP), or sending EtherNet/IP messages in 802.1Q tagged frames.

Supported Services

Class:	Get_Attribute_Single
Instance:	Get_Attribute_Single Set_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0001h (Object revision)

Instance Attributes

Attributes #1–4 and #6–7 an be customized by implementing the EtherNet/IP Host Object, see *EtherNet/IP Host Object (F8h), p. 150*

#	Name	Access	Type	Value/Description
1	802.1Q Tag Enable	Set	USINT	Enables or disables sending 802.1Q frames. Bit: Contents: 0 Disabled (Default) 1 Enabled
4	DSCP Urgent	Set	USINT	CIP transport class 1 messages with priority Urgent Default: 55
5	DSCP Scheduled	Set	USINT	CIP transport class 1 messages with priority Scheduled Default: 47
6	DSCP High	Set	USINT	CIP transport class 1 messages with priority High Default: 43
7	DSCP Low	Set	USINT	CIP transport class 1 messages with priority Low Default: 31
8	DSCP Explicit	Set	USINT	CIP UCMM and CIP class 3 Default: 27

10.10 Base Energy Object (4Eh)

Category

Extended

Object Description

The Base Energy Object acts as an “Energy Supervisor” for CIP Energy implementations. It is responsible for providing a time base for energy values, provides energy mode services, and can provide aggregation services for aggregating energy values up through the various levels of an industrial facility. It also provides a standard format for reporting energy metering results. The object is energy type independent and allows energy type specific data and functionality to be integrated into an energy system in a standard way. The Anybus CompactCom 40 EtherNet/IP supports one instance of the Base Energy Object. For instance, an electric power monitor may count metering pulse output transitions of a separate metering device. The count of such transitions, represented by a Base Energy Object instance, would reflect the energy consumption measured by the separate metering device.

An instance of the Base Energy Object may exist as a stand-alone instance, or it may exist in conjunction with an Electrical and/or Non-Electrical Energy Object instance (These objects are not implemented in the Anybus CompactCom 40 EtherNet/IP). If an instance of any of these objects is implemented in a device, it must be associated with a Base Energy Object instance in the device.

For this object to be able to access the network, the Energy Reporting Object (E7h) must be implemented in the host application, see the Anybus CompactCom 40 Software Design Guide for more information.

Supported Services

Class:	Get_Attribute_Single
Instance:	Get_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0002h (Object revision)

Instance Attributes

Attributes #1–4 and #6–7 an be customized by implementing the EtherNet/IP Host Object, see *EtherNet/IP Host Object (F8h), p. 150*

#	Name	Access	Type	Value/Description
1	Energy/Re-source Type	Get	UINT	Type of energy managed by this instance Always 0 (Generic)
2	Base Energy Object Capabilities	Get	UINT	Always 0 (Energy measured)
3	Energy Accuracy	Get	UINT	Specifies the accuracy of power and energy metering results, either in 0.01 percent of reading (default) or 0.01 of other units specified in attrib-ute #4. If 0, unknown.
4	Energy Ac-curacy Basis	Get	UINT	Always 0 (Percent of reading)
7	Consumed Energy Odometer	Get	ODOMETER	The value of the consumed energy.
8	Generated Energy Odometer	Get	ODOMETER	The value of the generated energy.
12	Energy Type Specific Ob-ject Path	Get	Struct of: UINT (Path size) padded EPATH (Path)	NULL path

- Depending on whether the instance reports consumed or generated energy, either attribute #7 or attrib-ute #8 is required.
- The struct data type ODOMETER makes it possible to represent very large values, for more information please consult the CIP specification Volume 1 (CIP Common).

10.11 Power Management Object (53h)

Category

Extended

Object Description

The Power Management Object provides standardized attributes and services to support the control of devices into and out of paused or sleep states. The Energy Control Object (F0h) has to be implemented for this object to gain access to the network.

See also ..

- Energy Control Object (F0h) (Anybus CompactCom 40 Software Design Guide)

Supported Services

Class:	Get_Attribute_Single
Instance:	Get_Attribute_Single
	Power_Management
	Set_Pass_Code
	Clear_Pass_Code

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0002h (Object revision)

Instance Attributes

#	Name	Access	Type	Value/Description
1	Power Management Command	Get	DWORD	Collection of bit fields comprising the most recent power management request.
2	Power Management Status	Get	DWORD	Collection of bit fields providing Power Management status information.
3	Client Path	Get	Struct of:	Specifies the EPATH from this instance (server) to its current owner (client).
			UINT (Path Size)	Size of path (in words
			Padded EPATH (Path)	
4	Number of Power Management Modes	Get	UINT	Number of Power Management Mode array entries in attribute 5.
5	Power Management Nodes	Get	Array of:	Array of low power modes
			Struct of:	Modes (Array of mode structures)
			USINT	Minimum Pause Units (Specifies the unit of Minimum Pause Time)
			UINT	Minimum Pause Time
			USINT	Resume Units (Specifies the unit of Resume Time)
			UINT	Resume Time (Required time to transition from the paused stated to the owned state.
			REAL	Power Level (Power in kW for this mode)
			BOOL	Availability (Specifies whether this mode can be entered given the current device state)
6	Sleeping State Support	Get	BOOL	0 (Sleeping state not supported)

10.12 ADI Object (A2h)

Category

Extended

Object Description

This object maps instances in the Application Data Object to EtherNet/IP. All requests to this object will be translated into explicit object requests towards the Application Data Object in the host application; the response is then translated back to CIP-format and sent to the originator of the request.

The ADI Object can be disabled using attribute 30 in the EtherNet/IP Host Object (F8h). This attribute can also be used to change the ADI Object number.

See also ..

- Application Data Object (see Anybus CompactCom 40 Software Design Guide)
- *Parameter Object (0Fh), p. 75* (CIP Object)
- *EtherNet/IP Host Object (F8h), p. 150*

Supported Services

Class:	Get_Attribute_Single
Instance:	Get_Attribute_Single
	Set_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0002h (Object revision)
2	Max Instance	Get	UINT	Equals attribute #4 in the Application Data Object
3	Number of instances	Get	UINT	Equals attribute #3 in the Application Data Object

For information about the Application Data Object, please consult the Anybus CompactCom 40 Software Design Guide.

Instance Attributes

Each instance corresponds to an instance within the Application Data Object (for more information, please consult the general Anybus CompactCom 40 Software Design Guide).

#	Name	Access	Type	Value/Description
1	Name	Get	SHORT_STRING	Parameter name (Including length)
2	ABCC Data type	Get	Array of USINT	Data type of instance value
3	No. of Elements	Get	USINT	Number of elements of the specified data type
4	Descriptor	Get	Array of USINT	Bit field describing the access rights for this instance Bit: Meaning: 0 Set = Get Access 1 Set = Set Access
5	Value	Get/Set	Determined by attribute #2	Instance value
6	Max Value	Get		The maximum permitted parameter value.
7	Min Value	Get		The minimum permitted parameter value.
8	Default Value	Get		The default parameter value.
9	Number of subelements	Get	UINT	Number of subelements in the ADI. Default value is 1 unless implemented in the application.

Attributes #5–8 are converted to/from CIP standard by the module

10.13 Port Object (F4h)

Category

Extended

Object Description

The Port Object describes the CIP ports present on the device. Each routable CIP port is described in a separate instance. Non-routable ports may be described. Devices with a single CIP port are not required to support this object.

The object exists only if enabled in the EtherNet/IP Host Object (Instance Attribute #17).

See also ..

- *EtherNet/IP Host Object (F8h), p. 150* (Anybus Module Object)
- *CIP Port Configuration Object (0Dh), p. 135* (Host Application Object)

Supported Services

Class:	Get_Attributes_All
	Get_Attribute_Single
Instance:	Get_Attributes_All
	Get_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0002h (Object revision)
2	Max Instance	Get	UINT	Max. instance number
3	Number of Instances	Get	UINT	Number of ports currently created.
8	Entry Port	Get	UINT	Returns the instance of the Port Object that describes the port through which this request entered the device.
9	Port Instance Info	Get	Array of:	Array of structures containing instance attributes 1 and 2 from each instance. The array is indexed by instance number, up to the maximum number of instances. The value at index 1 (offset 0) and any non-instantiated instances will be zero.
			Struct of: UINT (Type) UINT (Number)	Enumerates the type of port (see instance attribute #1) CIP port number associated with this port (see instance attribute #2)

Instance Attributes (Instance #1)

#	Name	Access	Type	Value/Description
1	Port Type	Get	UINT	0h (default) 4h (if the application registers a port)
2	Port Number	Get	UINT	2h
3	Link Object	Get	Struct of: UINT Padded EPATH	- 2h (Path Length) 20 F5 24 01h (Link Path)
4	Port Name	Get	SHORT_ STRING	“EtherNet/IP”
5	Port Type Name	Get	SHORT_ STRING	“”
6	Port Description	Get	SHORT_ STRING	“”
7	Node Address	Get	Padded EPATH	-
10	Port Routing Capabilities	Get	UDINT	1h (Routing of incoming Unconnected Messaging supported)

See also...

CIP Port Configuration Object (0Dh), p. 135

Instance Attributes (Instances #2... #8)

#	Name	Access	Type	Value/Description
1	Port Type	Get	UINT	Enumerates the type of port
2	Port Number	Get	UINT	CIP port number associated with this port
3	Link Object	Get	Struct of: UINT Padded EPATH	- Path length (number of 16-bit words) Logical path segments which identify the object for this port. The path must consist of one logical class segment and one logical instance segment. The maximum size is 12 bytes.
4	Port Name	Get	SHORT_ STRING	Name of port, e.g. “Port A”. Max. 64 characters.
5	Port Type Name	Get	SHORT_ STRING	“”
6	Port Description	Get	SHORT_ STRING	“”
7	Node Address	Get	Padded EPATH	Node number of this device on port. The range within this data type is restricted to a Port Segment.
8	Port Node Range	Get	Struct of: UINT (Min.) UINT (Max.)	- Min. node number on port Max. node number on port
10	Port Routing Capabilities	Get	UDINT	1h (Routing of incoming Unconnected Messaging supported)

See also...

CIP Port Configuration Object (0Dh), p. 135 , “Instance Attributes.”.

10.14 TCP/IP Interface Object (F5h)

Category

Extended

Object Description

This object provides the mechanism to configure the TCP/IP network interface of the module. It groups the TCP/IP-related settings in one instance for each TCP/IP capable communications interface.

See also ..

- *Communication Settings, p. 12*
- *Network Configuration Object (04h), p. 101* (Anybus Module Object)

Supported Services

Class:	Get_Attribute_All
	Get_Attribute_Single
Instance:	Get_Attribute_All
	Get_Attribute_Single
	Set_Attribute_Single

Class Attributes

#	Name	Access	Type	Value
1	Revision	Get	UINT	0004h (Object revision)
2	Max instance	Get	UINT	1 (Maximum instance number)
3	Number of instances	Get	UINT	1 (Number of instances)
6	Maximum ID Number Class Attributes	Get	UINT	7 (The attribute number of the last implemented class attribute)
7	Maximum ID Number Instance Attributes	Get	UINT	13 (The attribute number of the last implemented instance attribute)

Instance Attributes

#	Name	Access	Type	Value	Comments
1	Status	Get	DWORD	-	Bit: Meaning: (reserved, set to 0) 0–3 When set to h, attribute #5 contains valid configuration from DHCP or non-volatile storage. When set to 2h, attribute #5 contains valid configuration from hardware settings. Remaining values are reserved for future use. 4 Multicast pending if set to 1. 5 Interface configuration pending if set to 1. A new configuration will be loaded at the next reset. 6 AcdStatus. Set to 1 if an address conflict is detected. Address conflict detection is enabled/disabled in attribute #10. 7 AcdFault 8–31 (reserved, set to 0)
2	Configuration Capability	Get	DWORD	-	Bit: Meaning: 0-1: Always 0. For more information, consult the CIP specifications. 2: If set to 1, the device is capable of acting as a DHCP client. The bit is set to 0 if attribute #24 (Enable DHCP Client) is disabled in the *Ethernet Host Object (F9h), p. 159* 3: Always 0. For more information, consult the CIP specifications. 4: The ‘Configuration Settable’-bit reflects the value of instance attribute #9 in the "EtherNet/IP Host Object (F8h)" on page 161. 5: The module is hardware configurable when this bit is set to 1. The bit will be set if any of the address attributes is set in the Network Configuration Object (04h) during setup or if attribute #6 (Hardware configurable address) in the Application Object (FFh) is set. 6: Always 0. For more information, consult the CIP specifications. 7: If set to 1, the device is capable of detecting address conflicts. The bit is set to 0 if address conflict detection is disabled in the Ethernet Host Object, see page *159* 8 - 31: (reserved, set to 0)
3	Configuration Control	Get/Set	DWORD	-	Value: Meaning 0: Configuration from non-volatile memory 2: Configuration from DHCP
4	Physical Link Object	Get	Struct of: UINT (Path size) Padded EPATH	- 0002h 20 F6 24 03h	- - Path to Ethernet Link Object, Instance #3
5	Interface Configuration	Get/Set	Struct of: UDINT (IP) UDINT (Mask) UDINT (GW) UDINT (DNS1) UDINT (DNS2) STRING (Domain)		- IP address Subnet mask Default gateway Primary DNS Secondary DNS Default domain

#	Name	Access	Type	Value	Comments
6	Host Name	Get/Set	STRING	-	Host name of Anybus module
8	TTL Value	Get/Set	USINT	1	TTL value for EtherNet/IP multicast packets

#	Name	Access	Type	Value	Comments
9	Mcast Config	Set	Struct of:		IP multicast configuration.
	Alloc Control		USINT	0	Value: Meaning: 0: Use default allocation algorithm to generate multicast addresses 1: Allocate multicast addresses according to the values in the “Num Mcast”- and “Mcast Start Addr”-fields.
	(reserved)		USINT	0	Set to zero. Do not change.
	Num Mcast		UINT	-1	Number of multicast addresses to allocate for EtherNet/IP
	Mcast Start Addr		UDINT	-	Starting multicast address from which to begin allocation
10	SelectAcd	Set	Bool	1	Value: Meaning: 0: Disable ACD 1: Enable ACD (Default). If ACD (address conflict detection) is enabled, bit 6 in attribute #1 will be set if an ACD conflict is detected. The Network Status LED will also indicate a detected conflict, see *Front View, p. 166* .
11	LastConflict-Detected	Set	Struct of:		ACD Diagnostic parameters Related to the last conflict detected.
	AcdActiviity		USINT	-	State of ACD activity when last conflict detected.
	RemoteMAC		ARRAY of 6 USINT	-	MAC address of remote node form the ARP PDU in which a conflict was detected.
	ArpPdu		ARRAY of 28 USINT	-	Copy of the raw ARP PDU in which a conflict was detected.
12	EIP Quick-Connect	Set	Bool	0	Value: Meaning: 0: Disable EIP QuickConnect (Default) 1: Enable EIP QuickConnect If EIP QuickConnect is enabled, the Quick-Connect feature will direct EtherNet/IP target devices to quickly power up and join an EtherNet/IP network.
13	Encapsulation inactivity timeout	Set	UINT	0 - 3600	Number of seconds of inactivity before a TCP connection is closed. 0: Disabled

- Support for configuring network settings (attributes #3 and #5) from the network can be disabled by implementing attribute #9 in the EtherNet/IP Host Object, see *EtherNet/IP Host Object (F8h), p. 150*
- Attributes #10 and #11 will not be available if ACD is disabled using attribute #11 in the Ethernet Host Object (F9h).
- Attribute #12:
  - If the module is configured to use EIP QuickConnect functionality, the EDS file has to be changed. As the EDS file is changed, the identity of the module has to be changed and the module will require certification.
  - This attribute exists if attribute #26 in the EtherNet/IP Host Object is implemented, see *EtherNet/IP Host Object (F8h), p. 150*.

10.15 Ethernet Link Object (F6h)

Category

Extended

Object Description

This object maintains link specific counters and status information for an IEEE 802.3 communications interface. Exactly one instance for each communications interface on the module is supported. Instances for internally accessible interfaces can also be supported.

See also ..

- *Communication Settings, p. 12*
- *Network Configuration Object (04h), p. 101* (Anybus Module Object)

Supported Services

Class:	Get_Attributes_All
	Get_Attribute_Single
Instance:	Get_Attributes_All
	Get_Attribute_Single
	Set_Attribute_Single
	Get_And_Clear

Class Attributes

By default, three instances (port 1, port 2 and the internal port) are implemented, meaning that two ports are activated.

If port 2 is inactivated in the Port 2 State attribute of the Ethernet Host Object (F9h), only one instance (port 1) should be implemented.

#	Name	Access	Type	Value
1	Revision	Get	UINT	0004h (Object revision)
2	Max Instance	Get	UINT	1 or 3 (Maximum instance number)
3	Number of Instances	Get	UINT	1 or 3 (Number of instances)
6	Maximum ID Number Class Attributes	Get	UINT	7 (The attribute number of the last implemented class attribute.)
7	Maximum ID Number Instance Attributes	Get	UINT	11 (The attribute number of the last implemented instance attribute.)

Instance Attributes

#	Name	Access	Type	Value	Comments
1	Interface Speed	Get	UDINT	10 or 100	Actual Ethernet interface speed.
2	Interface Flags	Get	DWORD	-	See table “Interface Flags” below.
3	Physical Address	Get	Array of 6 USINTs	(MAC ID)	Physical network address, i.e. assigned MAC address.
4	Interface Counters	Get	Struct of:		
	In Octets		UDINT	N/A	Octets received on the interface
	In Ucast Packets		UDINT	N/A	Unicast packets received on the interface
	In NUcast Packets		UDINT	N/A	Nonunicast packets received on the interface
	In Discards		UDINT	N/A	Inbound packets with unknown protocol
	In Errors		UDINT	N/A	Inbound packets that contain errors (does not include In discards)
	In Unknown Protos		UDINT	N/A	Inbound packets with unknown protocol
	Out Octets		UDINT	N/A	Octets sent on the interface
	Out Ucast Packets		UDINT	N/A	Unicast packets sent on the interface
	Out NUcast Packets		UDINT	N/A	Nonunicast packets sent on the interface
	Out Discards		UDINT	N/A	Outbound packets with unknown protocol
	Out Errors		UDINT	N/A	Outbound packets that contain errors (does not include Out discards)
5	Media Counters	Get	Struct of:		Media specific counters
	Alignment Errors		UDINT	N/A	Frames received that are not an integral number of octets in length
	FCS Errors		UDINT	N/A	Frames received that do not pass the FCS check
	Single Collisions		UDINT	N/A	Successfully transmitted frames that have experienced exactly one collision
	Multiple Collisions		UDINT	N/A	Successfully transmitted frames that have experienced more than one collision
	SQE Test Errors		UDINT	0	The number of times the SQE test error message is generated(Counter not provided with current PHY interface)
	Deferred Transmissions		UDINT	N/A	Frames for which the first transmission attempt is delayed because the medium is busy
	Late Collisions		UDINT	N/A	The number of times a collision is detected later than 512 bit-times into the transmission of a packet
	Excessive Collisions		UDINT	N/A	Frames for which a transmission fails due to excessive collisions
	MAC Transmit Errors		UDINT	N/A	Frames for which a transmission fails due to an internal MAC sublayer receive error
	Carrier Sense Errors		UDINT	N/A	The number of times that the carrier sense condition was lost or never asserted when attempting to transmit a frame
	Frame Too Long		UDINT	N/A	Frames received that exceed the maximum permitted frame size
	MAC Receive Errors		UDINT	N/A	Frames for which reception on an interface fails due to an internal MAC sublayer receive error

#	Name	Access	Type	Value	Comments
6	Interface Control	Get/Set	Struct of:		
	Control Bits		WORD	-	Interface control bits
	Forced Interface Speed		UINT	-	Speed at which the interface shall be forced to operate. Returns ‘Object state Conflict’ if auto-negotiation is enabled
7	Interface Type	Get	USINT	-	See table “Interface State” below.
8	Interface State	Get	USINT	-	See table “Interface Type” below.
9	Admin State	Get/Set	USINT	-	See table “Admin State” below.
10	Interface Label	Get	SHORT_ STRING	—	See table “Interface Label” below.
11	Interface Capability	Get	Struct of:	-	Indication of the capabilities of the interface
	Capability Bits		DWORD	-	Interface capabilities, other than speed/duplex See table “Interface Capability” below.
	Speed/Duplex Options		Struct of:	-	Indicates speed/duplex pairs supported in the Interface Control Attribute
			USINT	-	Speed/duplex array count
			Array of Struct of:	-	Speed/duplex array
			UINT	-	Interface speed
			USINT	-	Interface Duplex Mode 0 = half duplex 1 = full duplex 2 - 255 = Reserved

- Support for attribute #6 can be disabled by implementing attribute #9 in the EtherNet/IP Host Object (F8h). see *EtherNet/IP Host Object (F8h), p. 150*
- Support for attribute #8 can be disabled by implementing the port state attributes (#12 or #13) in the Ethernet Host object (F9h) see *Ethernet Host Object (F9h), p. 159*

Interface Flags

Bit	Name	Description
0	Link status	Indicates whether or not the Ethernet 802.3 communications interface is connected to an active network. Value: Meaning: 0 Inactive link 1 Active link
1	Half/full duplex	Indicates the duplex mode currently in use. Value: Meaning: 0 Half duplex 1 Full duplex
2 - 4	Negotiation Status	Indicates the status of link auto-negotiation. Value: Meaning: 0 Auto-negotiation in progress. 1 Auto-negotiation and speed detection failed (using default values) (Recommended default values are 10 Mbps, half duplex) 2 Auto negotiation failed but detected speed (using default duplex value) 3 Successfully negotiated speed and duplex. 4 Auto-negotiation not attempted. Forced speed and duplex.
5	Manual Setting requires Reset	Value: Meaning: 0 Interface can activate changes to link parameters during runtime 1 Reset is required in order for changes to have effect
6	Local Hardware Fault	Value: Meaning: 0 No local hardware fault detected 1 Local hardware fault detected
7-31	(reserved)	Set to 0.

Interface State

This attribute indicates the current operational state of the interface.

Value	Description
0	Unknown interface state.
1	The interface is enabled and is ready to send and receive data.
2	The interface is disabled.
3	The interface is testing.

Admin State

This attribute controls the administrative setting of the interface state.

Value	Description
0	(reserved)
1	Enable the interface.
2	Disable the interface.
3-255	(reserved)

Interface Label

This attribute is configurable via the EtherNet/IP Host Object, see page *150*

Instance	Value
1	Port 1
2	Port 2
3	Internal

Interface Type

Instance	Value	Description
1	2	Twisted-pair
2	2	Twisted-pair
3	1	Internal interface

Interface Capability

Bit	Name	Description		Implementation
0	Manual setting requires reset	Indicates whether or not the device requires a reset to apply changes made to the Interface Control attribute (#6).		Return 0
		0	Indicates that the device automatically applies changes made to the Interface Control attribute (#6) and, therefore, does not require a reset in order for changes to take effect. This bit shall have this value when the Interface Control attribute (#6) is not implemented.	
		1	1 = Indicates that the device does not automatically apply changes made to the Interface Control attribute (#6) and, therefore, will require a reset in order for changes to take effect. Note: this bit shall also be replicated in the Interface Flags attribute (#2), in order to retain backwards compatibility with previous object revisions.	
1	Auto-negotiate	0	Indicates that the interface does not support link auto-negotiation	0 for internal interface, 1 for external interfaces
		1	Indicates that the interface supports link auto-negotiation	
2	Auto-MDIX	0	Indicates that the interface does not support auto MDIX operation	0 for internal interface, 1 for external interfaces
		1	Indicates that the interface supports auto MDIX operation	
3	Manual speed/duplex	0	Indicates that the interface does not support manual setting of speed/duplex. The Interface Control attribute (#6) shall not be supported.	0 for internal interface, 1 for external interfaces
		1	Indicates that the interface supports manual setting of speed/duplex via the Interface Control attribute (#6)	
4 - 31	Reserved	Shall be set to 0		Return 0

11 Anybus Module Objects

11.1 General Information

This chapter specifies the Anybus Module Object implementation and how they correspond to the functionality in the Anybus CompactCom 40 EtherNet/IP.

Standard Objects:

- *Anybus Object (01h), p. 98*
- *Diagnostic Object (02h), p. 99*
- *Network Object (03h), p. 100*
- *Network Configuration Object (04h), p. 101*

Network Specific Objects:

- *Socket Interface Object (07h), p. 110*
- *SMTP Client Object (09h), p. 127*
- *Anybus File System Interface Object (0Ah), p. 132*
- *Network Ethernet Object (0Ch), p. 133*
- *CIP Port Configuration Object (0Dh), p. 135*
- *Functional Safety Module Object (11h), p. 137*

11.2 Anybus Object (01h)

Category

Basic

Object Description

This object assembles all common Anybus data, and is described thoroughly in the general *Anybus CompactCom 40 Software Design Guide*.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute Set_Attribute Get_Enum_String

Object Attributes (Instance #0)

(Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.)

Instance Attributes (Instance #1)

Basic

#	Name	Access	Type	Value
1	Module type	Get	UINT16	0403h (Standard Anybus CompactCom 40)
2... 11	-	-	-	Consult the general Anybus CompactCom 40 Software Design Guide for further information.
12	LED colors	Get	struct of: UINT8 (LED1A) UINT8 (LED1B) UINT8 (LED2A) UINT8 (LED2B)	Value: Color: 01h Green 02h Red 01h Green 02h Red
13... 16	-	-	-	Consult the general Anybus CompactCom 40 Software Design Guide for further information.

Extended

#	Name	Access	Type	Value
17	Virtual attributes	Get/Set	-	Consult the general Anybus CompactCom 40 Software Design Guide for further information.
18	Black list/White list	Get/Set		
19	Network time	Get	UINT64	0 (Not supported)

11.3 Diagnostic Object (02h)

Category

Basic

Object Description

This object provides a standardized way of handling host application events & diagnostics, and is thoroughly described in the general *Anybus CompactCom 40 Software Design Guide*.

Supported Commands

Object:	Get_Attribute
	Create
	Delete
Instance:	Get_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1... 4	-	-	-	Consult the general Anybus CompactCom 40 Software Design Guide for further information.
11	Max no. of instances	Get	UINT16	5+1 (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 ‘EXCEPTION’-state.)
12	Supported functionality	Get	BITS32	Bit 0: “0” (Latching events are not supported) Bit 1 - 31: reserved (shall be “0”)

Instance Attributes (Instance #1)

Extended

#	Name	Access	Data Type	Value
1	Severity	Get	UINT8	Consult the general Anybus CompactCom 40 Software Design Guide for further information.
2	Event Code	Get	UINT8	
3	-	-	-	Not implemented in product
4	Slot	Get	UINT16	Consult the general Anybus CompactCom 40 Software Design Guide for further information.
5	ADI	Get	UINT16	
6	Element	Get	UINT8	
7	Bit	Get	UINT8	

Attributes #2 and #4–7 can not be represented on the network and are ignored by the module.

In this implementation, the severity level of all instances are combined (using logical OR) and represented on the network through the CIP Identity Object.

11.4 Network Object (03h)

Category

Basic

Object Description

For more information regarding this object, consult the general *Anybus CompactCom 40 Software Design Guide*.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute
	Set_Attribute
	Get_Enum_String
	Map_ADI_Write_Area
	Map_ADI_Read_Area
	Map_ADI_Write_Ext_Area
	Map_ADI_Read_Ext_Area

Object Attributes (Instance #0)

(Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.)

Instance Attributes (Instance #1)

Basic

#	Name	Access	Type	Value
1	Network type	Get	UINT16	009Bh (EtherNet/IP Beacon Based 2–port)
2	Network type string	Get	Array of CHAR	“Ethernet/IP(TM)”
3	Data format	Get	ENUM	00h (LSB first)
4	Parameter data support	Get	BOOL	True
5	Write process data size	Get	UINT16	Current write process data size (in bytes) Updated on every successful Map_ADI_Write_Area. (Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.)
6	Read process data size	Get	UINT16	Current read process data size (in bytes) Updated on every successful Map_ADI_Read_Area. (Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.)
7	Exception Information	Get	UINT8	Additional information available if the module has entered the EXCEPTION state. Value: Meaning: 00h No information available 01h Invalid assembly instance mapping 02h Missing MAC address (Only valid for Anybus IP)

11.5 Network Configuration Object (04h)

Category

Extended

Object Description

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

If the settings in this object do not match the configuration used, the Module Status LED will flash red to indicate a minor error.

The object is described in further detail in the Anybus CompactCom 40 Software Design Guide.

See also...

- *Communication Settings, p. 12*
- *TCP/IP Interface Object (F5h), p. 88* (CIP-object)
- *Ethernet Link Object (F6h), p. 92*
- *E-mail Client, p. 32*

Supported Commands

Object:	Get_Attribute
	Reset
Instance:	Get_Attribute
	Set_Attribute
	Get_Enum_String

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value	Description
3	Number of instances	Get	UINT16	15	Supported number of instances
4	Highest instance number	Get	UINT16	19	Highest instance number

(Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.)

Instance Attributes (Instance #3, IP Address)

Value is used after module reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	"IP address" (Multilingual, see page *109*)
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	04h (four elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of UINT8	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)
6	Configured Value	Get	Array of UINT8	Holds the configured value, which will be written to attribute #5 after the module has been reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)

Instance Attributes (Instance #4, Subnet Mask)

Value is used after module reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	"Subnet mask" (Multilingual, see page *109*)
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	04h (four elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of UINT8	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)
6	Configured Value	Get	Array of UINT8	Holds the configured value, which will be written to attribute #5 after the module has been reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)

Instance Attributes (Instance #5, Gateway Address)

Value is used after module reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	"Gateway" (Multilingual, see page *109*)
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	04h (four elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of UINT8	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)
6	Configured Value	Get	Array of UINT8	Holds the configured value, which will be written to attribute #5 after the module has been reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)

Instance Attributes (Instance #6, DHCP Enable)

Value is used after module reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“DHCP” (Multilingual, see page 109)
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	ENUM	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. (Multilingual, see page 109) Value / String / Meaning 00h / “Disable” / DHCP disabled 01h / “Enable” / DHCP enabled (default)
6	Configured Value	Get	ENUM	Holds the configured value, which will be written to attribute #5 after the module has been reset. Value / String / Meaning 00h / “Disable” / DHCP disabled 01h / “Enable” / DHCP enabled

Instance Attributes (Instance #7 Ethernet Communication Settings 1)

Changes have immediate effect.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Comm 1” (Multilingual, see page 109)
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	ENUM	Value / String / Meaning (Multilingual, see page 109) 00h / “Auto” / Auto negotiation (default) 01h / “10 HDX” / 10Mbit, half duplex 02h / “10 FX” / 10Mbit, full duplex 03h / “100HDX” / 100Mbit, half duplex 04h / “100FX” / 100Mbit, full duplex
6	Configured Value	Get	ENUM	Holds the configured value, which will be written to attribute #5 after the module has been reset. Value / String / Meaning (Multilingual, see page 109) 00h / “Auto” / Auto negotiation 01h / “10 HDX” / 10Mbit, half duplex 02h / “10 FX” / 10Mbit, full duplex 03h / “100HDX” / 100Mbit, half duplex 04h / “100FX” / 100Mbit, full duplex

Instance Attributes (Instance #8 Ethernet Communication Settings 2)

Changes have immediate effect.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Comm 2” (Multilingual, see page 109)
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	ENUM	Value — String — Meaning (Multilingual, see page 109) 00h — “Auto” — Auto negotiation (default) 01h — “10 HDX” — 10Mbit, half duplex 02h — “10 FX” — 10Mbit, full duplex 03h — “100HDX” — 100Mbit, half duplex 04h — “100FX” — 100Mbit, full duplex
6	Configured Value	Get	ENUM	Holds the configured value, which will be written to attribute #5 after the module has been reset. Value — String — Meaning (Multilingual, see page 109) 00h — “Auto” — Auto negotiation 01h — “10 HDX” — 10Mbit, half duplex 02h — “10 FX” — 10Mbit, full duplex 03h — “100HDX” — 100Mbit, half duplex 04h — “100FX” — 100Mbit, full duplex

Instance Attributes (Instance #9, DNS1)

This instance holds the address to the primary DNS server. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“DNS1” (Multilingual, see page 109)
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	04h (four elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of UINT8	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)
6	Configured Value	Get	Array of UINT8	Holds the configured value, which will be written to attribute #5 after the module has been reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)

Instance Attributes (Instance #10, DNS2)

This instance holds the address to the secondary DNS server. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“DNS2” (Multilingual, see page *109*)
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	04h (four elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of UINT8	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)
6	Configured Value	Get	Array of UINT8	Holds the configured value, which will be written to attribute #5 after the module has been reset. Valid range: 0.0.0.0 - 255.255.255.255 (Default =0.0.0.0)

Instance Attributes (Instance #11, Host name)

This instance holds the host name of the module. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Host name” (Multilingual, see page *109*)
2	Data type	Get	UINT8	07h (= CHAR)
3	Number of elements	Get	UINT8	40h (64 elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of CHAR	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Host name, 64 characters
6	Configured Value	Get	Array of CHAR	Holds the configured value, which will be written to attribute #5 after the module has been reset. Host name, 64 characters

Instance Attributes (Instance #12, Domain name)

This instance holds the domain name. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Host name” (Multilingual, see page *109*)
2	Data type	Get	UINT8	07h (= CHAR)
3	Number of elements	Get	UINT8	30h (48 elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of CHAR	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Domain name, 48 characters
6	Configured Value	Get	Array of CHAR	Holds the configured value, which will be written to attribute #5 after the module has been reset. Domain name, 48 characters

Instance Attributes (Instance #13, SMTP Server)

This instance holds the SMTP server address. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“SMTP server” (Multilingual, see page *109*)
2	Data type	Get	UINT8	07h (= CHAR)
3	Number of elements	Get	UINT8	40h (64 elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of CHAR	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. SMTP server address, 64 characters.
6	Configured Value	Get	Array of CHAR	Holds the configured value, which will be written to attribute #5 after the module has been reset. SMTP server address, 64 characters.

Instance Attributes (Instance #14, SMTP User)

This instance holds the user name for the SMTP account. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“SMTP user” (Multilingual, see page *109*)
2	Data type	Get	UINT8	07h (= CHAR)
3	Number of elements	Get	UINT8	40h (64 elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of CHAR	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. SMTP account user name, 64 characters
6	Configured Value	Get	Array of CHAR	Holds the configured value, which will be written to attribute #5 after the module has been reset. SMTP account user name, 64 characters

Instance Attributes (Instance #15, SMTP Password)

This instance holds the password for the SMTP account. Changes are valid after reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“SMTP Pswd” (Multilingual, see page *109*)
2	Data type	Get	UINT8	07h (= CHAR)
3	Number of elements	Get	UINT8	40h (64 elements)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	Array of CHAR	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. SMTP account password, 64 characters
6	Configured Value	Get	Array of CHAR	Holds the configured value, which will be written to attribute #5 after the module has been reset. SMTP account password, 64 characters

Instance Attributes (Instance #16, MDI 1 Settings)

This instance holds the settings for MDI/MDIX 1. Changes have immediate effect.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“MDI 1”
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	ENUM	Value (ENUM): String: Meaning: 00h “Auto” (default) 01h “MDI” 02h “MDIX”
5	Configured Value	Get	ENUM	Holds the configured value, which will be written to attribute #5 after the module has been reset. Value (ENUM): String: Meaning: 00h “Auto” 01h “MDI” 02h “MDIX”

Instance Attributes (Instance #17, MDI 2 Settings)

This instance holds the settings for MDI/MDIX 2. Changes have immediate effect.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“MDI 2”
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	ENUM	Value (ENUM): String: Meaning: 00h “Auto” (default) 01h “MDI” 02h “MDIX”
5	Configured Value	Get	ENUM	Holds the configured value, which will be written to attribute #5 after the module has been reset. Value (ENUM): String: Meaning: 00h “Auto” 01h “MDI” 02h “MDIX”

Instance Attributes (Instances #18 and #19)

These instances are reserved for future attributes.

Instance Attributes (Instance #20, QuickConnect)

tance Attributes (Instance #20, QuickConnect)

This instance enables or disables the QuickConnect functionality from the application. Changes are valid after reset or power cycle. The value of the QuickConnect attribute (#12) in the TCP/IP Interface object (F5h), will change immediately.

QuickConnect has to be enabled in the Ethernet Host object (F9h) for this instance to be available.

See also...

- *TCP/IP Interface Object (F5h), p. 88*
- *Ethernet Host Object (F9h), p. 159*

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“QuickConnect”
2	Data type	Get	UINT8	08h (= ENUM)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	ENUM	If read, the actual value will be received. If written, the written value is reflected in attribute #6 until a reset. Value: Meaning: 00h Disable (default) 01h Enable
6	Configured Value	Get	ENUM	Holds the configured value, which will be written to attribute #5 after the module has been reset. Value: Meaning: 00h Disable 01h Enable

Multilingual Strings

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

Instance	English	German	Spanish	Italian	French
3	IP address	IP-Adresse	Dirección IP	Indirizzo IP	Adresse IP
4	Subnet mask	Subnetz-maske	Masac. subred	Sottorete	Sous-réseau
5	Gateway	Gateway	Pasarela	Gateway	Passerelle
6	DHCP	DHCP	DHCP	DHCP	DHCP
	Enable	Einschalten	Activado	Abilitato	Activé
	Disable	Ausschalten	Desactivado	Disabilitato	Désactivé
7	Comm 1	Komm 1	Comu 1	Connessione 1	Comm 1
	Auto	Auto	Auto	Auto	Auto
	10 HDX	10 HDX	10 HDX	10 HDX	10 HDX
	10 FDX	10 FDX	10 FDX	10 FDX	10 FDX
	100 HDX	100 HDX	100 HDX	100 HDX	100 HDX
	100 FDX	100FDX	100 FDX	100 FDX	100 FDX
8	Comm 2	Komm 2	Comu 2	Connessione 2	Comm 2
	Auto	Auto	Auto	Auto	Auto
	10 HDX	10 HDX	10 HDX	10 HDX	10 HDX
	10 FDX	10 FDX	10 FDX	10 FDX	10 FDX
	100 HDX	100 HDX	100 HDX	100 HDX	100 HDX
	100 FDX	100FDX	100 FDX	100 FDX	100 FDX
9	DNS1	DNS 1	DNS Primaria	DNS1	DNS1
10	DNS2	DNS 2	DNS Secundia.	DNS2	DNS2
11	Host name	Host name	Nombre Host	Nome Host	Nom hôte
12	Domain name	Domain name	Nobre Domain	Nome Dominio	Dom Domaine
13	SMTP Server	SMTP Server	Servidor SMTP	Server SMTP	SMTP serveur
14	SMTP User	SMTP User	Usuario SMTP	Utente SMTP	SMTP utilisa.
15	SMTP Pswd	SMTP PSWD	Clave SMTP	Password SMTP	SMTP mt passe

11.6 Socket Interface Object (07h)

Category

Extended

Object Description

This object provides direct access to the TCP/IP stack socket interface, enabling custom protocols to be implemented over TCP/UDP.

Note that some of the commands used when accessing this object may require segmentation. A message will be segmented if the amount of data sent or received is larger than the message channel can handle. For more information, see *Message Segmentation, p. 125*.

The use of functionality provided by this object should only be attempted by users who are already familiar with socket interface programming and who fully understands the concepts involved in TCP/IP programming.

Supported Commands

Object:	Get_Attribute
	Create (See below)
	Delete (See below)
	DNS_Lookup (See below)
Instance:	Get_Attribute
	Set_Attribute
	Bind (See below)
	Shutdown (See below)
	Listen (See below)
	Accept (See below)
	Connect (See below)
	Receive (See below)
	Receive_From (See below)
	Send (See below)
	Send_To (See below)
	P_Add_membership (See below)
	IP_Drop_membership (See below)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value	
1	Name	Get	Array of CHAR	“Socket interface”	
2	Revision	Get	UINT8	01h	
3	Number of instances	Get	UINT16	Number of opened sockets	
4	Highest instance no.	Get	UINT16	Highest created instance number	
11	Max. no. of instances	Get	UINT16	0008h (8 instances):	BACnet/IP
				0014h (20 instances):	All other industrial Ethernet networks

Instance Attributes (Sockets #1...Max. no. of instances)

Extended

#	Name	Access	Data Type	Description
1	Socket Type	Get	UINT8	Value: Socket Type 00h SOCK_STREAM, NONBLOCKING (TCP) 01h SOCK_STREAM, BLOCKING (TCP) 02h SOCK_DGRAM, NONBLOCKING (UDP) 03h SOCK_DGRAM, BLOCKING (UDP)
2	Port	Get	UINT16	Local port that the socket is bound to
3	Host IP	Get	UINT32	Host IP address, or 0 (zero) if not connected
4	Host port	Get	UINT16	Host port number, or 0 (zero) if not connected
5	TCP State	Get	UINT8	State (TCP sockets only): Value State/Description 00h CLOSED Closed 01h LISTEN Listening for connection 02h SYN_SENT Active, have sent and received SYN 03h SYN_RECEIVED Have sent and received SYN 04h ESTABLISHED Established. 05h CLOSE_WAIT Received FIN, waiting for close 06h FIN_WAIT_1 Have closed, sent FIN 07h CLOSING Closed exchanged FIN; await FIN ACK 08h LAST_ACK Have FIN and close; await FIN ACK 09h FIN_WAIT_2 Have closed, FIN is acknowledged Ah TIME_WAIT Quiet wait after close
6	TCP RX bytes	Get	UINT16	Number of bytes in RX buffers (TCP sockets only)
7	TCP TX bytes	Get	UINT16	Number of bytes in TX buffers (TCP sockets only)
8	Reuse address	Get/Set	BOOL	Socket can reuse local address Value Meaning 1 Enabled 0 Disabled (default)
9	Keep alive	Get/Set	BOOL	Protocol probes idle connection (TCP sockets only). If the Keep alive attribute is set, the connection will be probed for the first time after it has been idle for 120 minutes. If a probe attempt fails, the connection will continue to be probed at intervals of 75s. The connection is terminated after 8 failed probe attempts. Value Meaning 1 Enabled 0 Disabled (default)
10	IP Multicast TTL	Get/Set	UINT8	IP Multicast TTL value (UDP sockets only). Default = 1.
11	IP Multicast Loop	Get/Set	BOOL	IP multicast loop back (UDP sockets only) Must belong to group in order to get the loop backed message Value Meaning 1 Enabled (default) 0 Disabled
12	(reserved)			
13	TCP No Delay	Get/Set	BOOL	Don't delay send to coalesce packets (TCP). Value Meaning 1 Delay (default) 0 Don't delay (turn off Nagle's algorithm on socket)
14	TCP Connect Timeout	Get/Set	UINT16	TCP Connect timeout in seconds (default = 75s)

Command Details: Create

Category

Extended

Details

Command Code	03h
Valid for:	Object Instance

Description

This command creates a socket.

This command is only allowed in WAIT_PROCESS, IDLE and PROCESS_ACTIVE states.

- Command Details

Field	Contents	
CmdExt[0]	(reserved, set to zero)	
CmdExt[1]	Value:	Socket Type:
	00h	SOCK_STREAM, NON-BLOCKING (TCP)
	01h	SOCK_STREAM, BLOCKING (TCP)
	02h	SOCK_DGRAM, NON-BLOCKING (UDP)
	03h	SOCK_DGRAM, BLOCKING (UDP)

- Response Details

Field	Contents	Comments
Data[0]	Instance number (low)	Instance number of the created socket.
Data[1]	Instance number (high)	

Command Details: Delete

Category

Extended

Details

Command Code	04h
Valid for:	Object Instance

Description

This command deletes a previously created socket and closes the connection (if connected).

- If the socket is of TCP-type and a connection is established, the connection is terminated with the RST-flag.
- To gracefully terminate a TCP-connection, it is recommended to use the ‘Shutdown’-command (see below) before deleting the socket, causing the connection to be closed with the FIN-flag instead.
- Command Details

Field	Contents	Comments
CmdExt[0]	Instance number to delete (low)	Instance number of socket that shall be deleted.
CmdExt[1]	Instance number to delete (high)	

- Response Details

  (no data)

Command Details: Bind

Category

Extended

Details

Command Code	10h
Valid for:	Instance

Description

This command binds a socket to a local port.

- Command Details

Field	Contents	Comments
CmdExt[0]	Requested port number (low)	Set to 0 (zero) to request binding to any free port.
CmdExt[1]	Requested port number (high)	

- Response Details

Field	Contents	Comments
CmdExt[0]	Bound port number (low)	Actual port that the socket was bound to.
CmdExt[1]	Bound port number (high)	

Command Details: Shutdown

Category

Extended

Details

Command Code	11h
Valid for:	Instance

Description

This command closes a TCP-connection using the FIN-flag. Note that the response does not indicate if the connection actually shut down, which means that this command cannot be used to poll non-blocking sockets, nor will it block for blocking sockets.

- Command Details

Field	Contents	
CmdExt[0]	(reserved, set to zero)	
CmdExt[1]	Value:	Mode:
	00h	Shutdown receive channel
	01h	Shutdown send channel
	02h	Shutdown both receive- and send channel

- Response Details

  (no data)

The recommended sequence to gracefully shut down a TCP connection is described below.

Application initiates shutdown:

1. Send shutdown with CmdExt[1] set to 01h. This will send FIN-flag to host shutting down the send channel, note that the receive channel will still be operational.
2. Receive data on socket until error message Object specific error (EPIPE (13)) is received, indicating that the host closed the receive channel. If host does not close the receive channel use a timeout and progress to step 3.
3. Delete the socket instance. If step 2 timed out, RST-flag will be sent to terminate the socket.

Host initiates shutdown:

1. Receive data on socket, if zero bytes received it indicates that the host closed the receive channel of the socket.
2. Try to send any unsent data to the host.
3. Send shutdown with CmdExt[1] set to 01h. This will send FIN-flag to host shutting down the send channel.
4. Delete the socket instance.

Command Details: Listen

Category

Extended

Details

Command Code	12h
Valid for:	Instance

Description

This command puts a TCP socket in listening state.

- Command Details

Field	Contents
CmdExt[0]	(reserved, set to zero)
CmdExt[1]	(reserved)

- Response Details

  (no data)

Command Details: Accept

Category

Extended

Details

Command Code	13h
Valid for:	Instance

Description

This command accepts incoming connections on a listening TCP socket. A new socket instance is created for each accepted connection. The new socket is connected with the host and the response returns its instance number.

NONBLOCKING mode	This command must be issued repeatedly (polled) for incoming connections. If no incoming connection request exists, the module will respond with error code 0006h (EWOULDBLOCK).
BLOCKING mode	This command will block until a connection request has been detected.

This command will only be accepted if there is a free instance to use for accepted connections. For blocking connections, this command will reserve an instance.

- Command Details

  (no data)

- Response Details

Field	Contents
Data[0]	Instance number for the connected socket (low byte)
Data[1]	Instance number for the connected socket (high byte)
Data[2]	Host IP address byte 4
Data[3]	Host IP address byte 3
Data[4]	Host IP address byte 2
Data[5]	Host IP address byte 1
Data[6]	Host port number (low byte)
Data[7]	Host port number (high byte)

Command Details: Connect

Category

Extended

Details

Command Code	14h
Valid for:	Instance

Description

For SOCK-DGRAM-sockets, this command specifies the peer with which the socket is to be associated (to which datagrams are sent and the only address from which datagrams are received).

For SOCK_STREAM-sockets, this command attempts to establish a connection to a host.

SOCK_STREAM-sockets may connect successfully only once, while SOCK_DGRAM-sockets may use this service multiple times to change their association. SOCK-DGRAM-sockets may dissolve their association by connecting to IP address 0.0.0.0, port 0 (zero).

NON-BLOCKING mode:	This command must be issued repeatedly (polled) until a connection is connected, rejected or timed out. The first connect-attempt will be accepted, thereafter the command will return error code 22 (EINPROGRESS) on poll requests while attempting to connect.
BLOCKING mode:	This command will block until a connection has been established or the connection request is cancelled due to a timeout or a connection error.

- Command Details

Field	Contents
CmdExt[0]	(reserved, set to zero)
CmdExt[1]	
Data[0]	Host IP address byte 4
Data[1]	Host IP address byte 3
Data[2]	Host IP address byte 2
Data[3]	Host IP address byte 1
Data[4]	Host port number (low byte)
Data[5]	Host port number (high byte)

- Response Details

  (no data)

Command Details: Receive

Category

Extended

Details

Command Code	15h
Valid for:	Instance

Description

This command receives data from a connected socket. Message segmentation may be used to receive up to 1472 bytes (for more information, see *Message Segmentation, p. 125*).

For SOCK-DGRAM-sockets, the module will return the requested amount of data from the next received datagram. If the datagram is smaller than requested, the entire datagram will be returned in the response message. If the datagram is larger than requested, the excess bytes will be discarded.

For SOCK_STREAM-sockets, the module will return the requested number of bytes from the received data stream. If the actual data size is less than requested, all available data will be returned.

NON-BLOCKING mode:	If no data is available on the socket, the error code 0006h (EWOULDBLOCK) will be returned.
BLOCKING mode:	The module will not issue a response until the operation has finished.

If the module responds successfully with 0 (zero) bytes of data, it means that the host has closed the connection. The send channel may however still be valid and must be closed using **Shutdown** and/or **Delete**.

• Command Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control bits	For more information, see *Message Segmentation, p. 125*
Data[0]	Receive data size (low)	Only used in the first segment
Data[1]	Receive data size (high)	

• Response Details

The data in the response may be segmented (For more information, see *Message Segmentation, p. 125*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control bits	For more information, see *Message Segmentation, p. 125*
Data[0...n]	Received data	-

Command Details: Receive_From

Category

Extended

Details

Command Code	16h
Valid for:	Instance

Description

This command receives data from an unconnected SOCK_DGRAM-socket. Message segmentation may be used to receive up to 1472 bytes (For more information, see *Message Segmentation, p. 125*).

The module will return the requested amount of data from the next received datagram. If the datagram is smaller than requested, the entire datagram will be returned in the response message. If the datagram is larger than requested, the excess bytes will be discarded.

The response message contains the IP address and port number of the sender.

NON-BLOCKING mode:	If no data is available on the socket, the error code 0006h (EWOULDBLOCK) will be returned.
BLOCKING mode:	The module will not issue a response until the operation has finished.

• Command Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control bits	For more information, see *Message Segmentation, p. 125*
Data[0]	Receive data size (low byte)	Only used in the first segment
Data[1]	Receive data size (high byte)	

• Response Details

The data in the response may be segmented (For more information, see *Message Segmentation, p. 125*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control bits	For more information, see *Message Segmentation, p. 125*
Data[0]	Host IP address byte 4	The host address/port information is only included in the first segment. All data thereafter will start at Data[0]
Data[1]	Host IP address byte 3	
Data[2]	Host IP address byte 2	
Data[3]	Host IP address byte 1	
Data[4]	Host port number (low byte)	
Data[5]	Host port number (high byte)	
Data[6...n]	Received data	

Command Details: Send

Category

Extended

Details

Command Code	17h
Valid for:	Instance

Description

This command sends data on a connected socket. Message segmentation may be used to send up to 1472 bytes (For more information, see *Message Segmentation, p. 125*).

NON-BLOCKING mode:	If there isn't enough buffer space available in the send buffers, the module will respond with error code 0006h (EWOULDBLOCK)
BLOCKING mode:	If there isn't enough buffer space available in the send buffers, the module will block until there is.

- Command Details

  To allow larger amount of data (i.e. >255 bytes) to be sent, the command data may be segmented (For more information, see *Message Segmentation, p. 125*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control	(For more information, see *Message Segmentation, p. 125*)
Data[0...n]	Data to send	-

- Response Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(ignore)
CmdExt[1]		
Data[0]	Number of sent bytes (low)	Only valid in the last segment
Data[1]	Number of sent bytes (high)	

Command Details: Send_To

Category

Extended

Details

Command Code	18h
Valid for:	Instance

Description

This command sends data to a specified host on an unconnected SOCK-DGRAM-socket. Message segmentation may be used to send up to 1472 bytes (For more information, see appendix For more information, see *Message Segmentation, p. 125*).

- Command Details

  To allow larger amount of data (i.e. >255 bytes) to be sent, the command data may be segmented (For more information, see *Message Segmentation, p. 125*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control	For more information, see *Message Segmentation, p. 125*
Data[0]	Host IP address byte 4	The host address/port information shall only be included in the first segment. All data thereafter must start at Data [0]
Data[1]	Host IP address byte 3	
Data[2]	Host IP address byte 2	
Data[3]	Host IP address byte 1	
Data[4]	Host port number (low byte)	
Data[5]	Host port number (high byte)	
Data[6...n]	Data to send	

- Response Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(ignore)
CmdExt[1]		
Data[0]	Number of sent bytes (low byte)	Only valid in the last segment
Data[1]	Number of sent bytes (high byte)	

Command Details: IP_Add_Membership

Category

Extended

Details

Command Code	19h
Valid for:	Instance

Description

This command assigns the socket an IP multicast group membership. The module always joins the “All hosts group” automatically, however this command may be used to specify up to 20 additional memberships.

- Command Details

Field	Contents
CmdExt[0]	(reserved, set to zero)
CmdExt[1]	
Data[0]	Group IP address byte 4
Data[1]	Group IP address byte 3
Data[2]	Group IP address byte 2
Data[3]	Group IP address byte 1

- Response Details

  (no data)

Command Details: IP_Drop_Membership

Category

Extended

Details

Command Code	1Ah
Valid for:	Instance

Description

This command removes the socket from an IP multicast group membership.

- Command Details

Field	Contents
CmdExt[0]	(reserved, set to zero)
CmdExt[1]	
Data[0]	Group IP address byte 4
Data[1]	Group IP address byte 3
Data[2]	Group IP address byte 2
Data[3]	Group IP address byte 1

- Response Details

  (no data)

Command Details: DNS_Lookup

Category

Extended

Details

Command Code	1Bh
Valid for:	Object

Description

This command resolves the given host name and returns the IP address.

- Command Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]		
Data[0... N]	Host name	Host name to resolve

- Response Details (Success)

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]		
Data[0]	IP address byte 4	IP address of the specified host
Data[1]	IP address byte 3	
Data[2]	IP address byte 2	
Data[3]	IP address byte 1	

Socket Interface Error Codes (Object Specific)

The following object-specific error codes may be returned by the module when using the socket interface object.

Error Code	Name	Meaning
1	ENOBUFS	No internal buffers available
2	ETIMEDOUT	A timeout event occurred
3	EISCONN	Socket already connected
4	EOPNOTSUPP	Service not supported
5	ECONNABORTED	Connection was aborted
6	EWOULDBLOCK	Socket cannot block because unblocking socket type
7	ECONNREFUSED	Connection refused
8	ECONNRESET	Connection reset
9	ENOTCONN	Socket is not connected
10	EALREADY	Socket is already in requested mode
11	EINVAL	Invalid service data
12	EMSGSIZE	Invalid message size
13	EPIPE	Error in pipe
14	EDESTADDRREQ	Destination address required
15	ESHUTDOWN	Socket has already been shutdown
16	(reserved)	-
17	EHAVEOOB	Out of band data available
18	ENOMEM	No internal memory available
19	EADDRNOTAVAIL	Address is not available
20	EADDRINUSE	Address already in use
21	(reserved)	-
22	EINPROGRESS	Service already in progress
28	ETOOMANYREFS	Too many references
101	Command aborted	If a command is blocking on a socket, and that socket is closed using the Delete command, this error code will be returned to the blocking command.
102	DNS name error	Failed to resolve the host name (name error response from DNS server.
103	DNS timeout	Timeout when performing a DNS lookup.
104	DNS command failed	Other DNS error.

Message Segmentation

General

Category: Extended

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. The maximum socket message size is 1472. To ensure support for socket interface messages larger than 255 bytes a segmentation protocol is used.

The segmentation bits have to be set for all socket interface messages, in the commands where segmentation can be used, whether the messages have to be segmented or not.

The segmentation protocol is implemented in the message layer and must not be confused with the fragmentation protocol used on the serial host interface. Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.

The module supports 1 (one) segmented message per instance

Command Segmentation

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

Command segmentation is used for the following commands (Socket Interface Object specific commands):

- Send
- Send To

When issuing a segmented command, the following rules apply:

- When issuing the first segment, FS must be set.
- When issuing subsequent segments, both FS and LS must be cleared.
- When issuing the last segment, the LF-bit must be set.
- For single segment commands (i.e. size less or equal to the message channel size), both FS and LS must be set.
- The last response message contains the actual result of the operation.
- The command initiator may at any time abort the operation by issuing a message with AB set.
- If a segmentation error is detected during transmission, an error message is returned, and the current segmentation message is discarded. Note however that this only applies to the current segment; previously transmitted segments are still valid.

Segmentation Control Bits (Command)

Bit	Contents	Meaning
0	FS	Set if the current segment is the first segment
1	LS	Set if the current segment is the last segment
2	AB	Set if the segmentation shall be aborted
3...7	(reserved)	Set to 0 (zero)

Segmentation Control Bits (Response)

Bit	Contents	Meaning
0... 7	(reserved)	Ignore

Response Segmentation

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

Response segmentation is used for responses to the following commands (Socket Interface Object specific commands):

- Receive
- Receive From

When receiving a segmented response, the following rules apply:

- In the first segment, FS is set.
- In all subsequent segment, both FS and LS are cleared.
- In the last segment, LS is set.
- For single segment responses (i.e. size less or equal to the message channel size), both FS and LS are set.
- The command initiator may at any time abort the operation by issuing a message with AB set.

Segmentation Control bits (Command)

Bit	Contents	Meaning
0	(reserved)	(set to zero)
1		
2	AB	Set if the segmentation shall be aborted
3...7	(reserved)	Set to 0 (zero)

Segmentation Control bits (Response)

Bit	Contents	Meaning
0	FS	Set if the current segment is the first segment
1	LS	Set if the current segment is the last segment
2...7	(reserved)	Set to 0 (zero)

11.7 SMTP Client Object (09h)

Category

Extended

Object Description

This object groups functions related to the SMTP client.

Supported Commands

Object:	Get_Attribute
	Create
	Delete
	Send e-mail from file (see below)
Instance:	Get_Attribute
	Set_Attribute
	Send e-mail (see below)

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“SMTP Client”
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	-
4	Highest instance no.	Get	UINT16	-
11	Max. no. of instances	Get	UINT16	0006h
12	Success count	Get	UINT16	Reflects the no. of successfully sent messages
13	Error count	Get	UINT16	Reflects the no. of messages that could not be delivered

Instance Attributes (Instance #1)

Instances are created dynamically by the application.

#	Name	Access	Data Type	Description
1	From	Get/Set	Array of CHAR	e.g. “someone@somewhere.com”
2	To	Get/Set	Array of CHAR	e.g.“ someone.else@anywhere.net”
3	Subject	Get/Set	Array of CHAR	e.g. “Important notice”
4	Message	Get/Set	Array of CHAR	e.g.“Shut down the system”

Command Details: Create

Category

Extended

Details

Command Code	03h
Valid for:	Object

Description

This command creates an e-mail instance.

- Command Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]		

- Response Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(ignore)
CmdExt[1]		
Data[0]	Instance number	low byte
Data[1]		high byte

Command Details: Delete

Category

Extended

Details

Command Code	04h
Valid for:	Object

Description

This command deletes an e-mail instance.

- Command Details

Field	Contents	Comments
CmdExt[0]	E-mail instance number	low byte
CmdExt[1]		high byte

- Response Details

  (no data)

Command Details: Send E-mail From File

Category

Extended

Details

Command Code	11h
Valid for:	Object

Description

This command sends an e-mail based on a file in the file system.

The file must be a plain ASCII-file in the following format:

```
[To]
recipient

[From]
sender

[Subject]
email subject

Se  [Headers]
extra headers, optional

[Message]
actual email message
```

- Command Details

Field	Contents
CmdExt[0]	(reserved, set to zero)
CmdExt[1]	
Data[0... n]	Path + filename of message file

- Response Details

  (no data)

Command Details: Send E-mail

Category

Extended

Details

Command Code	10h
Valid for:	Instance

Description

This command sends the specified e-mail instance.

- Command Details

  (no data)

- Response Details

  (no data)

Object Specific Error Codes

Error Codes	Meaning
1	SMTP server not found
2	SMTP server not ready
3	Authentication error
4	SMTP socket error
5	SSI scan error
6	Unable to interpret e-mail file
255	Unspecified SMTP error
(other)	(reserved)

11.8 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, e.g. when application specific web pages are to be installed.

Instances are created and deleted dynamically during runtime.

This object is thoroughly described in *Anybus CompactCom 40 Software Design Guide*.

11.9 Network Ethernet Object (0Ch)

Category

Extended

Object Description

This object provides Ethernet-specific information to the application.

The object has three instances, each corresponding to a port:

Instance #	Port
1	Internal port
2	Port 1
3	Port 2

Each instance provides statistic counters for the port. This information can e.g be presented on internal web pages, if present, using the JSON script language.

Instance attribute #1 is reserved and used for backwards compatibility with earlier applications.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute

Object Attributes (Instance #0)

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

Instance Attributes (Instance #1)

#	Name	Access	Data Type	Description
1	MAC Address	Get	Array of UINT8	Reserved, used for backwards compatibility. (Device MAC address.) (See also *Ethernet Host Object (F9h), p. 159*)
2	(Reserved)			
3	(Reserved)			
4	MAC Address	Get	Array of UINT8	Device MAC address
5	Interface Counters	Get	Array of UINT32	Array containing MIB-II interface counters (rfc1213) See table below for array indices.
6	(Reserved)			

Instance Attributes (Instances #2 - #3)

#	Name	Access	Data Type	Description
1 - 4	(Reserved)			
5	Interface Counters	Get	Array of UINT32	Array containing MIB-II interface counters (rfc1213) See table below for array indices.
6	Media Counters	Get	Array of UINT32	Array containing Ethernet-Like MIB counters for the port. See table below for array indices.

Interface Counters

: Array indices of Interface Counters attribute (#5)

Index	Name	Description
0	In octets	Octets received on the interface
1	In Unicast Packets	Unicast packets received on the interface
2	In Non-Unicast Packets	Non-unicast packets (multicast/broadcast) packets received on the interface
3	In Discards	Inbound packets received on the interface but discarded
4	In Errors	Inbound packets that contain errors (does not include In Discards)
5	In Unknown Protos	Inbound packets with unknown protocol
6	Out Octets	Octets transmitted on the interface
7	Out Unicast packets	Unicast packets transmitted on the interface
8	Out Non-Unicast Packets	Non-unicast (multicast/broadcast) packets transmitted on the interface
9	Out Discards	Outbound packets discarded
10	Out Errors	Outbound packets that contain errors

Media Counters

: Array indices of Media Counters attribute (#6)

Index	Name	Description
0	AlignmentErrors;	Frames received that are not an integral number of octets in length
1	FCSErrors;	Frames received that do not pass the FCS check
2	SingleCollisions;	Successfully transmitted frames which experienced exactly one collision
3	MultipleCollisions;	Successfully transmitted frames which experienced more than one collision
4	SQETestErrors;	Number of times SQE test error is generated
5	DeferredTransmissions;	Frames for which first transmission attempt is delayed because the medium is busy
6	LateCollisions;	Number of times collision is detected later than 512 bit-times into the transmission of a packet
7	ExcessiveCollisions;	Frames for which transmission fails due to excessive collisions
8	IMACTransmitErrors;	Frames for which transmission fails due to an internal MAC sublayer transmit error
9	ICarrieSenseErrors;	Times that the carrier sense condition was lost or never asserted when attempting to transmit a frame
10	IFrameTooLong;	Frames received that exceed the maximum permitted frame size
11	IMACRecieveErrors;	Frames for which reception on an interface fails due to an internal MAC sublayer receive error

11.10 CIP Port Configuration Object (0Dh)

Category

Extended

Object Description

This object is used to populate and enumerate the CIP Port Object (see *Port Object (F4h), p. 86*) on the network side. Basically, this is a matter of creating and updating instances and attributes which shall represent a CIP Port within the host application. This process is necessary in case support for Unconnected CIP Routing has been enabled (see *EtherNet/IP Host Object (F8h), p. 150*, Instance Attribute #17).

Each instance within this object corresponds to an instance in the CIP Port Object. The object supports up to 8 instances, where instance #1 is dedicated to the local TCP port, enabling the host application to implement up to 7 additional ports. Instance #1 will automatically be populated with default values, however it is possible for the host application to customize instance attributes #2 and #4.

Apart from attribute #7, it is possible to write to the instance attributes only during setup. The host application is responsible for keeping instance attribute #7 updated for all ports located within the host application.

! Note that the module does not take over the host application responsibility for error control; the module will not verify that the data set by the host application is correct.

Supported Commands

Object:	Get_Attribute
	Create
	Delete
Instance:	Get_Attribute
	Set_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“Network Ethernet”
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	-
4	Highest instance no.	Get	UINT16	-
11	Max. no. of instances	Get	UINT16	0008h

Instance Attributes (Instance #1)

#	Name	Access	Data Type	Description
1	Port Type	Set	UINT16	Enumerates the port (See CIP specification, available from www.odva.org)
2	Port Number	Set	UINT16	CIP port number associated with this port
3	Link Path	Set	Array of UINT8	Logical path segments which identify the object for this port.
4	Port Name	Set	Array of CHAR	String (max. no. of characters is 64) which names the port.
5	-	-	-	(reserved)
6	-	-	-	(reserved)
7	Node Address	Set	Array of UINT8	Node number of this device on port. The data type restricts the range to a Port Segment. The encoded port number must match the value specified in attribute #2. A device which does not have a node number on the port can specify a zero length node address within the Port Segment (i.e. 10h 00h). In case the node address changes during runtime, the host application is responsible for updating this attribute as well.
8	Port Node Range	Set	Struct of: UINT16 (Min) UINT16 (Max)	Minimum and maximum node number on port. Support for this attribute is conditional; the attribute shall be supported provided that the node number can be reported within the range of the data type (e.g. DeviceNet). If not (as is the case with networks such as EtherNet/IP which uses a 4 byte IP address), the attribute shall not be supported.

11.11 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.

#	Name	Access	Data Type	Description
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

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 EtherNet/IP 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 EtherNet/IP 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 Host Application Objects

12.1 General Information

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

Standard Objects:

- Assembly Mapping Object (EBh) - (see Anybus CompactCom 40 Software Design Guide)
- Modular Device Object (ECh) - (see Anybus CompactCom 40 Software Design Guide)
- *Sync Object (EEh), p. 149*
- Energy Control Object (F0h) - (see Anybus CompactCom 40 Software Design Guide)
- Application Data Object (FEh) - (see Anybus CompactCom 40 Software Design Guide)
- Application Object (FFh) - (see Anybus CompactCom 40 Software Design Guide)

Network Specific Objects:

- *Functional Safety Object (E8h), p. 144*
- *Application File System Interface Object (EAh), p. 146*
- *CIP Identity Host Object (EDh), p. 147*
- *EtherNet/IP Host Object (F8h), p. 150*
- *Ethernet Host Object (F9h), p. 159*

12.2 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.

12.3 Application File System Interface Object (EAh)

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 allows the user to download software through the FTP server to the application. The application decides the available memory space.

This object is thoroughly described in *Anybus CompactCom 40 Software Design Guide*.

12.4 CIP Identity Host Object (EDh)

Category

Extended

Object Description

This object allows for applications to support additional CIP identity instances. It is used to provide additional product identity information, e.g. concerning the software installed.

The first instance in the CIP identity object will not change its behavior. When implementing instances in the CIP identity host object, they will be mapped to the CIP identity object starting at instance 2. Instance no. 1 in the CIP identity host object will be mapped to instance no. 2 in the CIP identity object and so on.

See also ...

- *Identity Object (01h), p. 64* (CIP object)

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute
	Get_Attribute_All

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“CIP Identity”
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	Depends on application
4	Highest instance no.	Get	UINT16	Depends on application

Instance Attributes (Instance #1)

#	Name	Access	Data Type	Description
1	Vendor ID	Get	UINT16	These values replace the values for the CIP identity object instance #2 and upwards. See also... *Identity Object (01h), p. 64* (CIP-object)
2	Device Type	Get	UINT16	
3	Product Code	Get	UINT16	
4	Revision	Get	struct of: UINT8 Major UINT8 Minor	
5	Status	Get	UNIT16	
6	Serial Number	Get	UINT32	
7	Product Name	Get	Array of CHAR	

Command Details: Get_Attribute_All

Category

Extended

Details

Command Code:	10h
Valid for:	Object

Description

This service must be implemented by the application for all instances that exist in the CIP identity host object. If identity data is requested from the network the Anybus module will issue this command to the application. The application will then respond with a message containing a struct of all attributes in the requested instance.

- Command Details

  (no data)

- Response Details

Field	Contents	Comments
MsgData[0, 1]	Vendor ID	ABCC CIP identity data
MsgData[2, 3]	Device type	
MsgData[4, 5]	Product code	
MsgData[6]	Major revision	
MsgData[7]	Minor revision	
MsgData [8,9]	Status	
MsgData[10 .. .13]	Serial number	
MsgData[14 .... n]	Product name	

12.5 Sync Object (EEh)

Category

Extended

Object Description

The Anybus CompactCom 40 EtherNet/IP does not support CIP Sync. This object is only used to store the cycle time for the last established IO connection that consumes data.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute
	Set_Attribute

Object Attributes (Instance #0)

(Consult the general *Anybus CompactCom 40 Software Design Guide* for further information.)

Instance Attributes (Instance #1)

The attributes are represented on EtherNet/IP as follows:

#	Name	Access	Data Type	Description
1	Cycle time	Get/Set	UINT32	The RPI for the last established IO connection that consumes data (O→T RPI)
2–8	(not implemented)			

12.6 EtherNet/IP Host Object (F8h)

Category

Basic, Extended

Object Description

This object implements EtherNet/IP specific features in the host application. Note that this object must not be confused with the Ethernet Host Object, see *Ethernet Host Object (F9h), p. 159*.

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

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

Note that some of the commands used when accessing this object may require segmentation. For more information, see *Message Segmentation, p. 125*.

If the module is configured to use EIP QuickConnect functionality, the EDS file has to be changed. As the EDS file is changed, the identity of the module has to be changed and the module will require certification..

See also ...

- *Identity Object (01h), p. 64* (CIP object)
- *Assembly Object (04h), p. 68* (CIP object)
- *Port Object (F4h), p. 86* (CIP object)
- *CIP Port Configuration Object (0Dh), p. 135*
- Anybus CompactCom 40 Software Design Guide, “Error Codes”

Supported Commands

Object:	Get_Attribute
	Process_CIP_Object_Request
	Set_Configuration_Data
	Process_CIP_Routing_Request
	Get_Configuration_Data
Instance:	Get_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“EtherNet/IP”
2	Revision	Get	UINT8	02h
3	Number of instances	Get	UINT16	0001h
4	Highest instance no.	Get	UINT16	0001h

Instance Attributes (Instance #1)

Basic

#	Name	Access	Data Type	Default Value	Comment
1	Vendor ID	Get	UINT16	005Ah	These values are set in the Identity Object (CIP) at startup. See also... • *Network Identity, p. 11* • *Identity Object (01h), p. 64* Please note that changing any of these attributes requires a new Vendor ID.
2	Device Type	Get	UINT16	002Bh	
3	Product Code	Get	UINT16	0037h	
4	Revision	Get	struct of: UINT8 Major UINT8 Minor	(software revision)	
5	Serial Number	Get	UINT32	(set at production)	
6	Product Name	Get	Array of CHAR	“Anybus CompactCom 40 EtherNet/IP(TM)”	

Extended

#	Name	Access	Data Type	Default Value	Comment
7	Producing Instance No.	Get	Array of UINT16	-	The values in this array are the EtherNet/IP Assembly instance numbers that matches the host application Assembly Mapping Ojbect instances that are listed in attribute #11 (Write PD Instance List). If the Assembly Mapping Object is not implemented, one element in this array is allowed, to set the producing instance number. The maximum number of entries in the array is 6. See “Multiple Assembly Instances” below for an example.
8	Consuming Instance No.	Get	Array of UINT16	-	The values in this array are the EtherNet/IP Assembly instance numbers that matches the host application Assembly Mapping Ojbect instances that are listed in attribute #12 (Read PD Instance List). If the Assembly Mapping Object is not implemented, one element in this array is allowed, to set the consuming instance number. The maximum number of entries in the array is 6. See “Multiple Assembly Instances” below for an example.
9	Enable communication settings from Net	Get	BOOL	True	Value / Meaning True: Can be set from network False: Cannot be set from network See also ... • *TCP/IP Interface Object (F5h), p. 88* (CIP-object) • *Ethernet Link Object (F6h), p. 92* CIP-object) • *Network Configuration Object (04h), p. 101*(Anybus Module Object)
11	Enable CIP forwarding	Get	BOOL	False	Value / Meaning True: Requests to unknown CIP objects and unknown assembly object instances are routed to the application. False: Requests to unknown CIP objects and unknown assembly object instances are not routed to the application. See also.command deails for Process _CIP_Object_ Request below
12	Enable Parameter Object	Get	BOOL	True	Value / Meaning True: Enable CIP Parameter Object False: Disable CIP Parameter Object
13	Input-Only heartbeat instance number	Get	UINT16	0003h	See “Instance 03h Attributes (Heartbeat, Input-Only)” in *Assembly Object (04h), p. 68* (CIP-object).

#	Name	Access	Data Type	Default Value	Comment
14	Listen-Only heart-beat instance number	Get	UINT16	0004h	See “Instance 04h Attributes (Heartbeat, Listen-Only)” in *Assembly Object (04h), p. 68* (CIP-object).
15	Assembly object Configuration in-stance number	Get	UINT16	0005h	See “Instance 05h Attributes (Configuration Data)” in *Assembly Object (04h), p. 68* (CIP-object).
16	Disable Strict IO Match	Get	BOOL	False	If true, the module will accept Class1 connection re-quests that have sizes that’s less than or equal to the configured IO sizes.
17	Enable unconnected routing	Get	BOOL	False	If true, the module enables unconnected CIP routing. This also triggers an initial upload of the contents of the CIP Port Mapping object.
18	Input-Only extended heartbeat instance number	Get	UINT16	0006h	See “Instance 06h Attributes (Heartbeat, Input-Only Ex-tended)” in *Assembly Object (04h), p. 68* (CIP-object).
19	Listen-Only extended heartbeat instance number	Get	UINT16	0007h	See “Instance 06h Attributes (Heartbeat, Listen-Only Extended)” in *Assembly Object (04h), p. 68* (CIP-object).
20	Interface label port 1	Get	Array of CHAR	Port 1	The value of this attribute is used to change the inter-face label for Ethernet Link Object Instance #1
21	Interface label port 2	Get	Array of CHAR	Port 2	The value of this attribute is used to change the inter-face label for Ethernet Link Object Instance #2
22	Interface label inter-nal port	Get	Array of CHAR	Internal	The value of this attribute is used to change the inter-face label for Ethernet Link Object Instance #3
23 - 25	(reserved)				
26	Enable EtherNet/IP QuickConnect	Get	BOOL	False	<u>Value</u> <u>Meaning</u> True EtherNet/IP QuickConnect functionality enabled. False False EtherNet/IP QuickConnect function-ality disabled. If the module is configured to use EIP QuickConnect functionality, the EDS file has to be changed. As the EDS file is changed, the identity of the module has to be changed and the module will require certification.
27 - 28	(reserved)				
29	Ignore Sequence Count Check	Get	BOOL	False	Setting this attribute to “true” makes the module ignore the Sequence Count Check for consumed Class 1 data. This means that all data, not just changed/new data, re-ceived from the Originator, will be copied to the applica-tion. Copying all data and not just changed data is a violation of the CIP specification. It will also affect the performance of the module. Use precaution when setting this flag to“true”. HMS Industrial Networks AB will do NO performance measurements and states NO guarantees about how performance will be affected when copying all data.
30	ABCC ADI Object Number	Get	UINT16	00A2h	This attribute either changes the object number of the-ADI Object (CIP object) or disables the ADI Object (CIP object). Valid object numbers are within the vendor spe-cific ranges (0064h - 00C7h and 0300h - 04FFh). Any other value will disable the ADI object.
31	Enable DLR	Get	BOOL	True	<u>Value</u> <u>Meaning</u> True DLR functionality enabled False DLR functionality disabled

Multiple Assembly Instances

The Assembly Mapping Object has two arrays on class level (Write PD Instance List and Read PD Instance List) listing instances defined by the application. The arrays of attributes 7 and 8 in the EtherNet/IP host object (Producing Instance Number and Consuming Instance number) are bound to the instance lists in the Assembly Mapping Object. The arrays list the corresponding CIP instance numbers representing each assembly instance defined by the application.

For more information, see

- *Using the Assembly Mapping Object (EBh), p. 19*
- Anybus CompactCom 40 Software Design Guide, “Assembly Mapping Object (EBh)”

Command Details: Process_CIP_Object_Request

Category

Extended

Details

Command Code:	10h
Valid for:	Object

Description

By setting the ‘Enable CIP Request Forwarding’-attribute (#11), all requests to unimplemented CIP-objects and unknown assembly object instances, will be forwarded to the host application through this command. The application then has to evaluate the request and return a proper response. The module supports one CIP-request; additional requests will be rejected by the module.

Note that since the telegram length on the host interface is limited, the request data size must not exceed 255 bytes. If it does, the module will send a ‘resource unavailable’ response to the originator of the request and the message will not be forwarded to the host application.

This command is similar - but not identical - to the ‘Process_CIP_Request’-command in the Anybus CompactCom 40 DeviceNet.

- Command Details

Field	Contents	Comments
CmdExt[0]	CIP Service Code	CIP service code from original CIP request
CmdExt[1]	Request Path Size	Number of 16-bit words in the Request Path field
MsgData[0... m]	Request Path	CIP EPATH (Class, Instance, Attr. etc.)
MsgData[m... n]	Request Data	Service-specific data

- Response Details

Field	Contents	Comments
CmdExt[0]	CIP Service Code	(Reply bit set)
CmdExt[1]	00h	(reserved, set to zero)
MsgData[0]	General Status	CIP General Status Code
MsgData[1]	Size of Additional Status	Number of 16-bit words in Additional Status array
MsgData[2... m]	Additional Status	Additional Status, if applicable
MsgData[m... n]	Response data	Actual response data, if applicable

! When using this functionality, make sure to implement the common CIP Class Attribute (attribute #1, Revision) for all objects in the host application firmware. Failure to observe this will prevent the module from successfully passing conformance tests.

Command Details: Set_Configuration_Data

Category

Extended

Details

Command Code:	11h
Valid for:	Object

Description

If the data segment in the CIP "Forward_Open" service contains Configuration Data, this will be forwarded to the host application through this command. If implemented, the host application should evaluate the request and return a proper response. Segmentation is used, see "Message Segmentation" on page 189 for more information. The maximum total amount of configuration data that will be accepted by the module is 458 bytes.

This command must be implemented in order to support Configuration Data. If not implemented, the CIP "Forward_Open"-request will be rejected by the module.

• Command Details

Field	Contents	Comments
CmdExt[0]	-	(reserved, ignore)
CmdExt[1]	Segmentation Control bits	See *Message Segmentation, p. 125*
MsgData[0 - 1]	Producing connection point	Producing connection point, requested by the originator.
MsgData[2 - 3]	Consuming connection point	Consuming connecition point, requested by the originator.
MsgData[4... n]	Data	Actual configuration data

MsgData[0 - 1]and MsgData[2 - 3] can both be 0. Normally, the Set_Configuration_Data command is sent to the application when an I/O connection is setup on the network. Producing connection point and consuming connection point are available and will be forwarded with the command. But if the configuration data originates from a set attribute single request or a not matching NULL forward open request, there is no information on the connection points and 0 (zero) will be forwarded to the application.

• Response Details (Success)

Field	Contents	Comments
CmdExt[0]	00h	(reserved, set to zero)
CmdExt[1]	00h	(reserved, set to zero)

• Response Details (Error)

Field	Contents	Comments
CmdExt[0]	00h	(reserved, set to zero)
CmdExt[1]	00h	(reserved, set to zero)
MsgData[0]	Error code	Anybus error code
MsgData[1]	Extended error code	If the Anybus error code is set to FFh, the extended error code shall be translated as shown in the table below.
MsgData[2... 3]	Index	If the Extended error code is set to 02h (invalid configuration), this parameter points to the attribute that failed.

Extended Error Code

If the Error code equals FFh (Object specific error), the extended code will be translated as below:

Code	Contents	CIP no.	CIP status code	Additional Information
01h	Ownership conflict	01h	Connection failure	The configuration data was supplied in a forward open request.
		10h	Device State conflict	The configuration data was supplied in a set request to the Assembly object.
02h	Invalid configuration	09h	Bad attribute data	CIP extended error code: Use value from MsgData[2 - 3]. The extended error code shall only be used if the request originated from a Forward Open request, not for explicit set requests.

- *Connection Manager (06h), p. 71* (CIP object)
- Message segmentation

Command Details: Process_CIP_Routing_Request

Category

Extended

Details

Command Code:	12h
Valid for:	Object

Description

The module will strip the first path within the “Unconnected_Send” service and evaluate whether or not it’s possible to continue with the routing (e.g. check that the requested port exists within the port object). If the stripped path was the last path the contents delivered to the application will be the CIP request sent to the destination node, otherwise it will be an “Unconnected_Send” service with updated route path information.

The module supports one pending request. Additional requests will be rejected by the module.

Please not that since the telegram length on the host interface is limited, the data must not exceed 255 bytes in length. If it does, the module will reject the originator of the request (“Resource unavailable”), and this command will not be issued towards the host application.

- Command Details

Field	Contents	Comments
CmdExt[0]	-	(reserved, ignore)
CmdExt[1]	-	(reserved, ignore)
MsgData[0... n]	Destination Path	Destination path encoded as an EPATH.
MsgData[n+1]	Time_tick	Valid after timeout parameters have been updated
MsgData[n+2]	Time-out_ticks	Valid after timeout parameters have been updated
MsgData[n+3... m]	CIP message	CIP message to route

- Response Details

Field	Contents	Comments
CmdExt[0]	00h	(reserved, set to zero)
CmdExt[1]	00h	(reserved, set to zero)
MsgData[0]	CIP Service	Actual CIP service code, response bit set
MsgData[1]	00h	(reserved, set to zero)
MsgData[2]	General Status	Actual CIP General status code
MsgData[3]	Size of Additional Status	No. of 16-bit words in Additional Status Array
MsgData[4... n]	Additional Status Array	Additional status, if applicable
MsgData[n+1... m]	Response Data	Actual response data

See also..

- *Port Object (F4h), p. 86* (CIP object)
- *CIP Port Configuration Object (0Dh), p. 135*

Command Details: Get_Configuration_Data

Category

Extended

Details

Command Code:	13h
Valid for:	Object

Description

If the configuration data is requested from the network, the Anybus will issue this command to the application. The application shall send the stored configuration data in the response message.

Segmentation is used since the telegram length on the host interface is limited. The maximum total amount of configuration data that will be accepted by the module is 458 bytes.

This command must be implemented in order to support Configuration Data. If not implemented, the request will be rejected by the Anybus module.

- Command Details

Field	Contents	Comments
CmdExt[0]	00h	-
CmdExt[1]	00h	-
MsgData[0... n]	-	No extended message data

- Response Details (Success)

Field	Contents	Comments
CmdExt[0]	00h	(reserved, set to zero)
CmdExt[1]	Segmentation Control bits	See *Message Segmentation, p. 125*
MsgData[0 - n]	Status	Configuration data from the application

- Response Details (Error)

Field	Contents	Comments
CmdExt[0]	00h	(reserved, set to zero)
CmdExt[1]	Segmentation Control bits	See *Message Segmentation, p. 125*
MsgData[0]	Status	Anybus protocol error code

12.7 Ethernet Host Object (F9h)

Object Description

This object implements Ethernet features in the host application.

Supported Commands

Object:	Get_Attribute
Instance:	Get_Attribute Set_Attribute

Object Attributes (Instance #0)

#	Name	Access	Data Type	Value
1	Name	Get	Array of CHAR	“Ethernet”
2	Revision	Get	UINT8	02h
3	Number of instances	Get	UINT16	0001h
4	Highest instance no.	Get	UINT16	0001h

Instance Attributes (Instance #1)

- If an attribute is not implemented, the default value will be used.
- The module is preprogrammed with a valid MAC address. To use that address, do not implement attribute #1.
- Do not implement attributes #9 and #10, only used for PROFINET devices, if the module shall use the preprogrammed MAC addresses.
- If new MAC addresses are assigned to a PROFINET device, these addresses (in attributes #1, #9, and #10) have to be consecutive, e.g. (xx:yy:zz:aa:bb:01), (xx:yy:zz:aa:bb:02), and (xx:yy:zz:aa:bb:03) with the first five octets not changing.

#	Name	Access	Data Type	Default Value	Comment
1	MAC address	Get	Array of UINT8	-	6 byte physical address value; overrides the preprogrammed Mac address. Note that the new Mac address value must be obtained from the IEEE. Do not implement this attribute if the preprogrammed Mac address is to be used.
2	Enable HICP	Get	BOOL	True (Enabled)	Enable/Disable HICP
3	Enable Web Server	Get	BOOL	True (Enabled)	Enable/Disable Web Server (Not used if Transparent Ethernet is enabled.)
4	(reserved)				Reserved for Anybus CompactCom 30 applications.
5	Enable Web ADI access	Get	BOOL	True (Enabled)	Enable/Disable Web ADI access (Not used if Transparent Ethernet is enabled.)
6	Enable FTP server	Get	BOOL	True (Enabled)	Enable/Disable FTP server (Not used if Transparent Ethernet is enabled.)
7	Enable admin mode	Get	BOOL	False (Disabled)	Enable/Disable FTP admin mode (Not used if Transparent Ethernet is enabled.)
8	Network Status	Set	UINT16	-	See below.

#	Name	Ac-cess	Data Type	Default Value	Comment
9	Port 1 MAC address	Get	Array of UINT8	-	**Note**: This attribute is only valid for PROFINET devices. 6 byte MAC address for port 1 (mandatory for the LLDP protocol). This setting overrides any Port MAC address in the host PROFINET IO Object. Do not implement this attribute if the preprogrammed Mac address is to be used.
10	Port 2 MAC address	Get	Array of UINT8	-	**Note**: This attribute is only valid for PROFINET devices. 6 byte MAC address for port 2 (mandatory for the LLDP protocol). This setting overrides any Port MAC address in the host PROFINET IO Object. Do not implement this attribute if the preprogrammed Mac address is to be used.
11	Enable ACD	Get	BOOL	True (Enabled)	Enable/Disable ACD protocol. If ACD functionality is disabled using this attribute, the ACD attributes in the CIP TCP/IP object (F5h) are not available.
12	Port 1 State	Get	ENUM	0 (Enabled)	The state of Ethernet port 1. • This attribute is not read by EtherCAT devices, where Port 1 is always enabled. • This attribute is not used by PROFINET 00h: Enabled 01h: Disabled. The port is treated as existing. References to the port can exist, e.g. in network protocol or on website.
13	Port 2 State	Get	ENUM	0 (Enabled)	The state of Ethernet port 2. • This attribute is not read by EtherCAT devices, where Port 2 is always enabled. • This attribute is not used by PROFINET 00h: Enabled 01h: Disabled. The port is treated as existing. References to the port can exist, e.g. in network protocol or on website. 02h: Inactive. The attribute is set to this value for a device that only has one physical port. All two-port functionality is disabled. No references can be made to this port. **Note:** This functionality is available for Ether-net/IP and Modbus-TCP devices.
14	(reserved)				
15	Enable reset from HICP	Get	BOOL	0 = False	Enables the option to reset the module from HICP.
16	IP configuration	Set	Struct of: UINT32 (IP address) UINT32 (Subnet mask) UINT32 (Gateway)	N/A	Whenever the configuration is assigned or changed, the Anybus CompactCom module will update this attribute.

#	Name	Access	Data Type	Default Value	Comment
17	IP address byte 0–2	Get	Array of UINT8[3]	[0] = 192 [1] = 168 [2] = 0	First three bytes in IP address. Used in standalone shift register mode if the configuration switch value is set to 1-245. In that case the IP address will be set to: Y[0].Y[1].Y[2].X Where Y0-2 is configured by this attribute and the last byte X by the configuration switch.
18	Ethernet PHY Configuration	Get	Array of BITS16	0x0000 for each port	Ethernet PHY configuration bit field. The length of the array shall equal the number of Ethernet ports of the product. Each element represents the configuration of one Ethernet port (element #0 maps to Ethernet port #1, element #1 maps to Ethernet port #2 and so on). **Note**: Only valid for EtherNet/IP and Modbus-TCP devices. Bit 0: Auto negotiation fallback duplex 0 = Half duplex 1 = Full duplex Bit 1–15: Reserved
20	SNMP read-only community string	Get	Array of CHAR	“public”	**Note**: This attribute is only valid for PROFINET devices. Sets the SNMP read-only community string. Max length is 32.
21	SNMP read-write community string	Get	Array of CHAR	“private”	**Note**: This attribute is only valid for PROFINET devices. Sets the SNMP read-write community string. Max length is 32.
22	DHCP Option 61 source	Get	ENUM	0 (Disabled)	**Note**: This attribute is currently only valid for Ethernet/IP devices. See below (DHCP Option 61, Client Identifier)
23	DHCP Option 61 generic string	Get	Array of UINT8	N/A	**Note**: This attribute is currently only valid for Ethernet/IP devices. See below (DHCP Option 61, Client Identifier)
24	Enable DHCP Client	Get	BOOL	1 = True	**Note**: This attribute is currently valid for Ethernet/IP and PROFINET devices. Enable/disable DHCP Client functionality 0: DHCP Client functionality is disabled 1: DHCP Client functionality is enabled

Network Status

This attribute holds a bit field which indicates the overall network status as follows:

Bit	Contents	Description	Comment
0	Link	Current global link status 1= Link sensed 0= No link	
1	IP established	1 = IP address established 0 = IP address not established	
2	(reserved)	(mask off and ignore)	
3	Link port 1	Current link status for port 1 1 = Link sensed 0 = No link	EtherCAT only: This link status indicates whether the Anybus CompactCom is able to communicat using Ethernet over EtherCAT (EoE) or not. That is, it indicates the status of the logical EoE port link and is not related to the link status on the physical EtherCAT ports.
4	Link port 2	Current link status for port 2 1 = Link sensed 0 = No link	Not used for EtherCAT
5... 15	(reserved)	(mask off and ignore)	

DHCP Option 61 (Client Identifier)

Only valid for EtherNet/IP devices

The DHCP Option 61 (Client Identifier) allow the end-user to specify a unique identifier, which has to be unique within the DHCP domain.

Attribute #22 (DHCP Option 61 source) is used to configure the source of the Client Identifier. The table below shows the definition for the Client identifier for different sources and their description.

Value	Source	Description
0	Disable	The DHCP Option 61 is disabled. This is the default value if the attribute is not implemented in the application.
1	MACID	The MACID will be used as the Client Identifier
2	Host Name	The configured Host Name will be used as the Client Identifier
3	Generic String	Attribute #23 will be used as the Client Identifier

Attribute #23 (DHCP Option 61 generic string) is used to set the Client Identifer when Attribute #22 has been set to 3 (Generic String). Attribute #23 contains the Type field and Client Identifier and shall comply with the definitions in RFC 2132. The allowed max length that can be passed to the module via attribute #23 is 64 octets.

Example:

If Attribute #22 has been set to 3 (Generic String) and Attribute #23 contains 0x01, 0x00, 0x30, 0x11, 0x33, 0x44, 0x55, the Client Identifier will be represented as an Ethernet Media Type with MACID 00:30:11:33:44:55.

Example 2:

If Attribute #22 has been set to 2 (Host Name) Attribute #23 will be ignored and the Client Identifier will be the same as the configured Host Name.

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 Implementation Details

B.1 SUP-Bit Definition

The supervised bit (SUP) indicates that the network participation is supervised by another network device. In the case of EtherNet/IP, this means that the SUP-bit is set when one or more CIP (Class 1 or Class 3) connections has been opened towards the module.

B.2 Anybus State Machine

The table below describes how the Anybus Statemachine relates to the EtherNet/IP network

Anybus State	**Implementation**	**Comment**
WAIT_PROCESS	The module stays in this state until a Class 1 connection has been opened.	-
ERROR	Class 1 connections errors Duplicate IP address detected	-
PROCESS_ACTIVE	Error free Class 1 connection active (RUN-bit set in the 32-bit Run/Idle header of an Exclusive-Owner connection)	Only valid for consuming connections.
IDLE	Class 1 connection idle.	
EXCEPTION	Unexpected error, e.g. watchdog timeout etc.	MS LED turns red (to indicate a major fault) NS LED is off

B.3 Application Watchdog Timeout Handling

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

C Secure HICP (Secure Host IP Configuration Protocol)

C.1 General

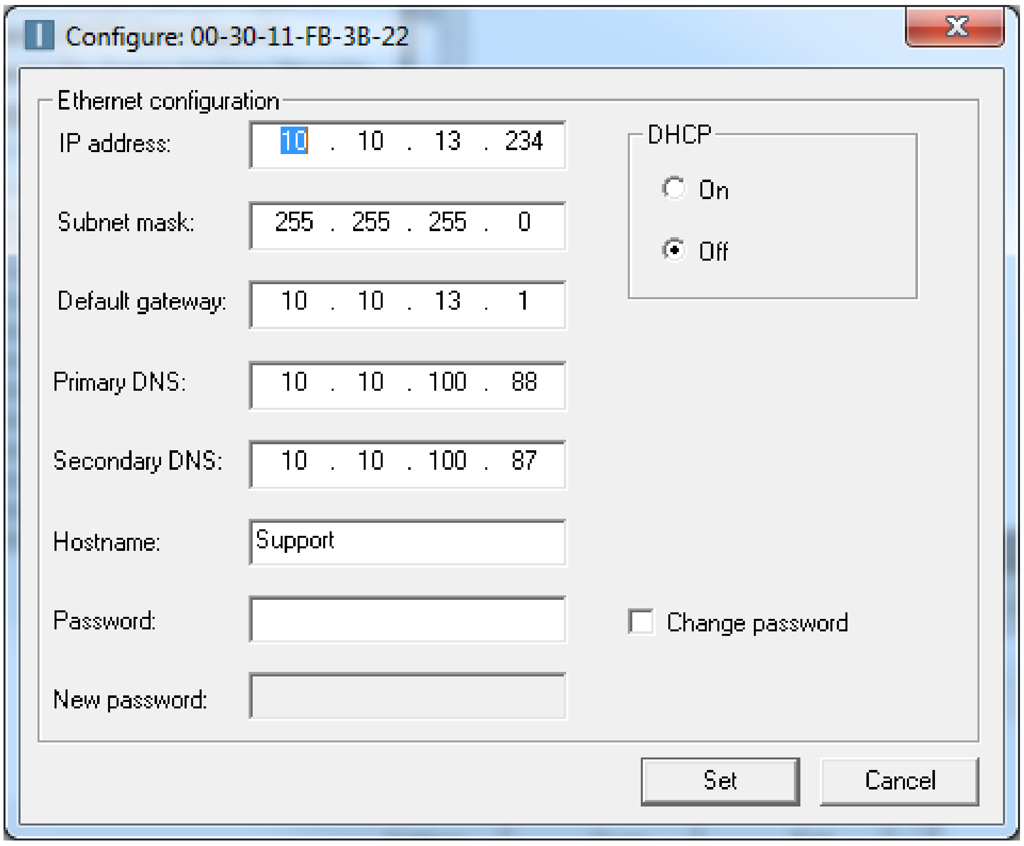
The Anybus CompactCom 40 EtherNet/IP supports the Secure HICP protocol used by the Anybus IPconfig utility for changing settings, e.g. IP address, Subnet mask, and enable/disable DHCP. Anybus IPconfig can be downloaded free of charge from the HMS website, www.anybus.com. This utility may be used to access the network settings of any Anybus product connected to the network via UDP port 3250.

The protocol offers secure authentication and the ability to restart/reboot the device(s).

C.2 Operation

When the application is started, the network is automatically scanned for Anybus products. The network can be rescanned at any time by clicking **Scan**.

To alter the network settings of a module, double-click on its entry in the list. A window will appear, containing the settings for the module.



IP address:	10 . 10 . 13 . 234
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	10 . 10 . 13 . 1
Primary DNS:	10 . 10 . 100 . 88
Secondary DNS:	10 . 10 . 100 . 87
Hostname:	Support
Password:	
New password:	

DHCP: On [✓ Off]

☐ Change password

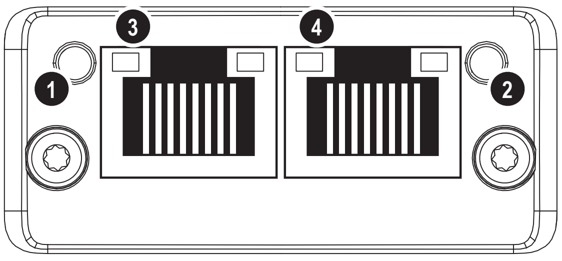
Fig. 7

Validate the new settings by clicking **Set**, or click **Cancel** to cancel all changes. Optionally, the configuration can be protected from unauthorized access by a password. To enter a password, check the **Change password** checkbox and enter the password in the **New password** text field.

D Technical Specification

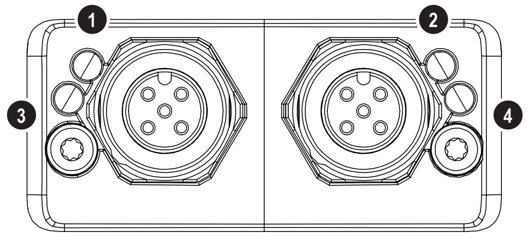
D.1 Front View

D.1.1 Front View (Ethernet Connectors)

#	Item	Connector	
1	Network Status LED	Ethernet, RJ45	
2	Module Status LED		
3	Link/Activity LED (port 1)		
4	Link/Activity LED (port 2)		

Test sequences are performed on the Network and Module Status LEDs during startup.

D.1.2 Front View (M12 Connectors)

#	Item	Connector	
1	Network Status LED	M12	
2	Module Status LED		
3	Link/Activity LED (port 1)		
4	Link/Activity LED (port 2)		

Test sequences are performed on the Network and Module Status LEDs during startup.

D.1.3 Network Status LED

LED State	Description
Off	No power or no IP address
Green	Online, one or more connections established (CIP Class 1 or 3)
Green, flashing	Online, no connections established
Red	Duplicate IP address, FATAL error
Red, flashing	One or more connections timed out (CIP Class 1 or 3)

D.1.4 Module Status LED

LED State	Description
Off	No power
Green	Controlled by a Scanner in Run state
Green, flashing	Not configured, or Scanner in Idle state
Red	Major fault (EXCEPTION-state, FATAL error etc.)
Red, flashing	Recoverable fault(s). Module is configured, but stored parameters differ from currently used parameters.

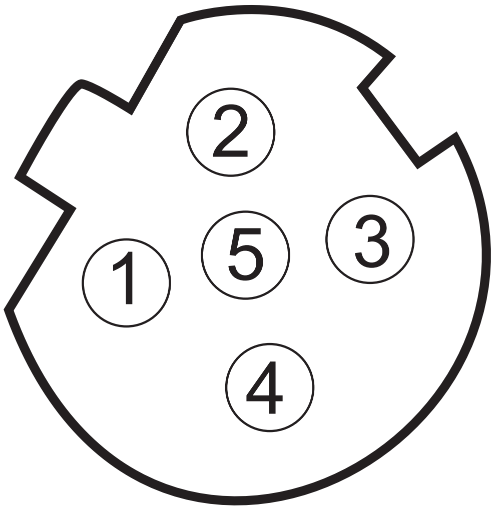
D.1.5 LINK/Activity LED 3/4

LED State	Description
Off	No link, no activity
Green	Link (100 Mbit/s) established
Green, flickering	Activity (100 Mbit/s)
Yellow	Link (10 Mbit/s) established
Yellow, flickering	Activity (10 Mbit/s)

D.1.6 Ethernet Interface

The Ethernet interface 10/100Mbit, full or half duplex operation.

D.1.7 M12 Connectors, Code D

Pin	Name	Description	
1	TXD+	Transmit positive	
2	RXD+	Receive positive	
3	TXD-	Transmit negative	
4	RXD-	Receive negative	
5 (Thread)	Shield	Shield	

D.2 Functional Earth (FE) Requirements

In order to ensure proper EMC behavior, the module must be properly connected to functional earth via the FE pad/FE mechanism described in the *Anybus CompactCom 40 Hardware Design Guide*. Proper EMC behavior is not guaranteed unless these FE requirements are fulfilled.

D.3 Power Supply

D.3.1 Supply Voltage

The Anybus CompactCom 40 EtherNet/IP requires a regulated 3.3 V power source as specified in the general *Anybus CompactCom 40 Hardware Design Guide*.

D.3.2 Power Consumption

TheAnybus CompactCom 40 EtherNet/IP is designed to fulfil the requirements of a Class B module. The current hardware design consumes up to 360 mA

In line with HMS policy of continuous product development, we reserve the right to change the exact power requirements of this product without prior notification. However, in any case, the Anybus CompactCom 40 EtherNet/IP will remain as a Class B module.

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

D.4 Environmental Specification

Consult the *Anybus CompactCom 40 Hardware Design Guide* for further information.

D.5 EMC Compliance

Consult the *Anybus CompactCom 40 Hardware Design Guide* for further information.

E Timing & Performance

E.1 General Information

This chapter specifies timing and performance parameters that are verified and documented for the Anybus CompactCom 40 EtherNet/IP.

Category	Parameters	Page
Startup Delay	T1, T2	*169*
NW_INIT Handling	T100	*169*
Event Based WrMsg Busy Time	T103	*169*
Event Based Process Data Delay	T101, T102	*170*

For further information, please consult the Anybus CompactCom 40 Software Design Guide.

E.2 Internal Timing

E.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	The Anybus CompactCom 40 EtherNet/IP module generates the first application interrupt (parallel mode)	64	ms
T2	The Anybus CompactCom 40 EtherNet/IP module is able to receive and handle the first application telegram (serial mode)	64	ms

E.2.2 NW_INIT Handling

This test measures the time required by the Anybus CompactCom 40 EtherNet/IP module to perform the necessary actions in the NW_INIT-state.

Parameter	Conditions
No. of network specific commands	Max.
No. of ADIs (single UINT8) mapped to Process Data in each direction. (If the network specific maximum is less than the value given here, the network specific value will be used.)	32
Event based application message response time	> 1 ms
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	Event based modes	58	ms

E.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	Min.	Max.	Unit.
T103	H_WRMSG area busy time	6	9	µs

E.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 has been received by the network interface, to when the RDPDI interrupt is asserted to the application.

“Write process data delay” 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.

The tests were run in 16-bit parallel event mode, with interrupts triggered only for new process data events. Eight different IO sizes (2, 16, 32, 64, 128, 256, 512 and 1024 bytes) were used in the tests, all giving the same test results.

The delay added by the PHY circuit has not been included, as this delay is insignificant compared to the total process data delay.

Parameter	Description	Delay (min.)	Delay (typ.)	Delay (max.)	Unit
T101	Read process data delay Measured at an IO size of 32 bytes	-	-	84	µs
T102	Write process data delay Measured at an IO size of 32 bytes	-	-	106	µs

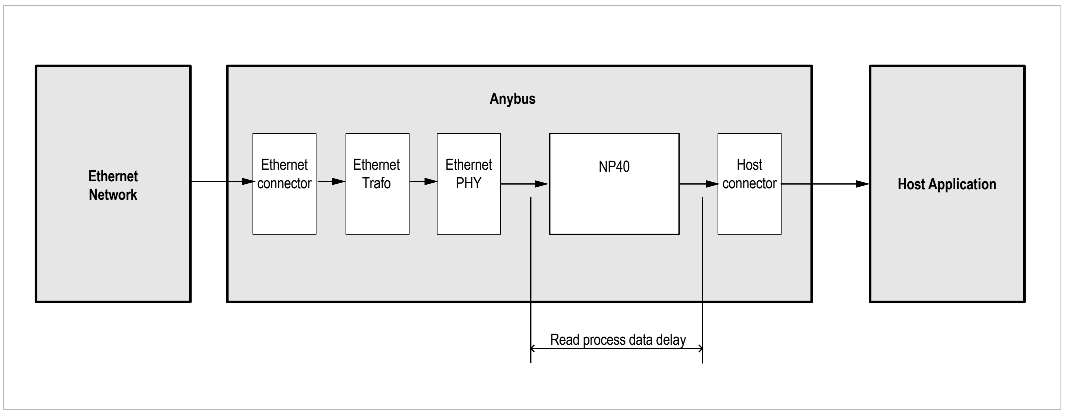


Fig. 8

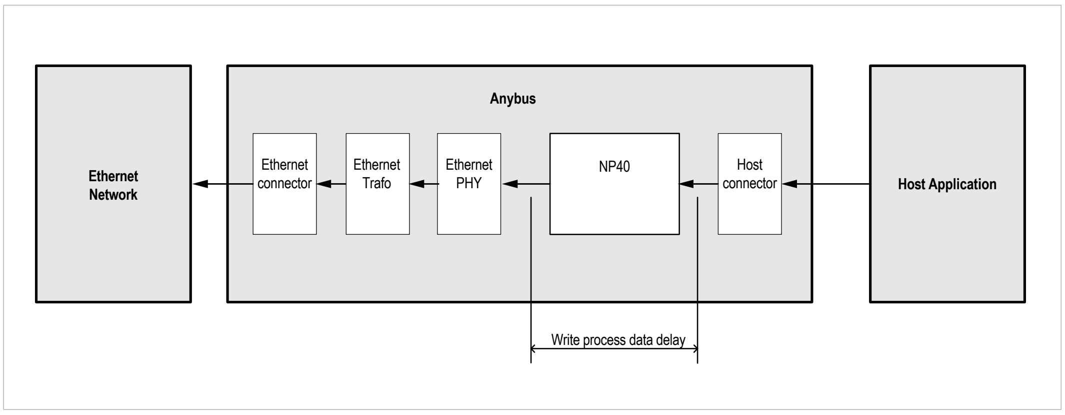


Fig. 9

F 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 EtherNet/IP, when designing with both series in one application, or when adapting a 30 series application for the 40 series.

F.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.

This documentation only deals with differences between the 30-series and the 40-series. For a description of new and enhanced functionality in the Anybus CompactCom 40-series, please consult our support pages, where you can find all documentation.

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

F.2 Hardware Compatibility

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

F.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.

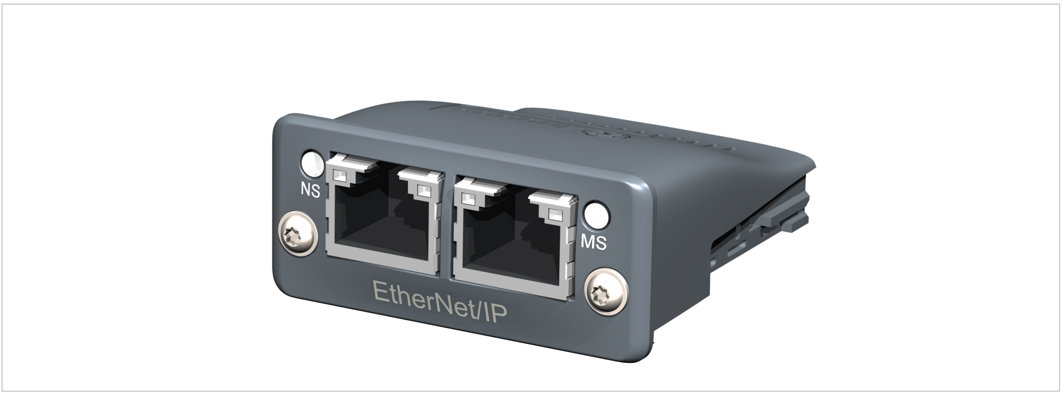


Fig. 10 Anybus CompactCom M30/M40

F.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.

F.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 AB 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 AB for more information on the Anybus CompactCom B40-2.

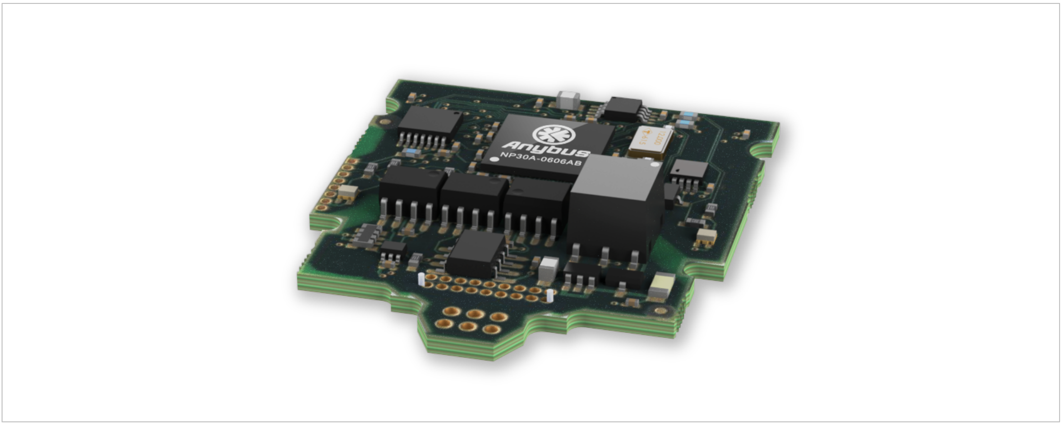


Fig. 11 Anybus CompactCom B30

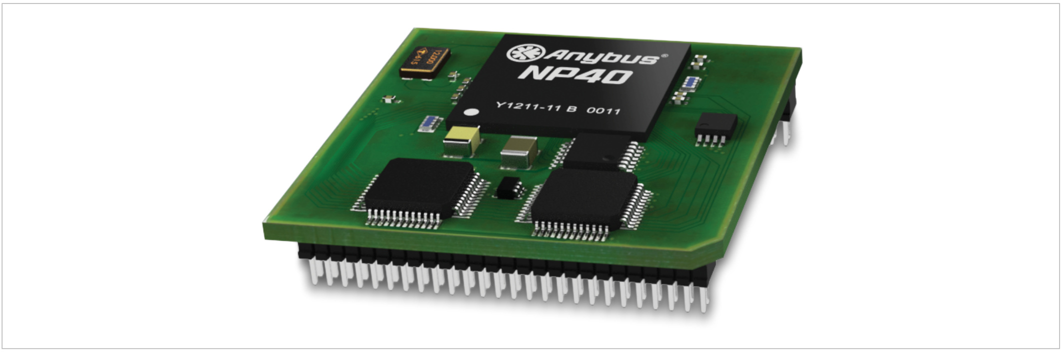


Fig. 12 Anybus CompactCom B40–1 (not for migration)

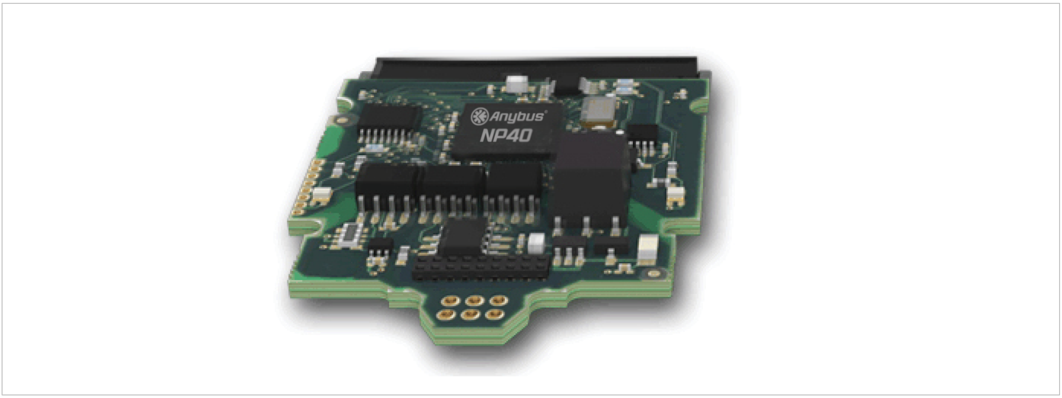


Fig. 13 Anybus CompactCom B40–2

F.2.4 Host Application Interface

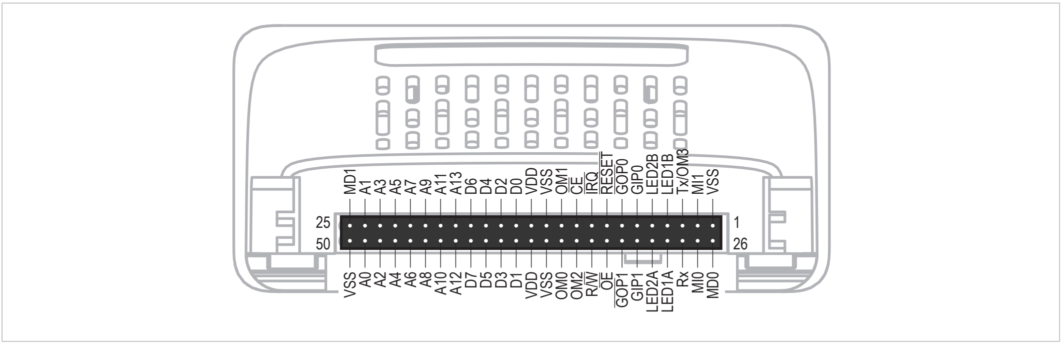


Fig. 14

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

This pin is Tx only in the 30-series. 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, the 40-series module or brick 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 3.45V for a logic high level.

F.3 General Software

F.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.

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”

F.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 “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.

F.3.3 Requests from 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.

F.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. 175* • *GOP[0..1]/LED4[A..B], p. 175*

F.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 HMS comply with this difference.

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

F.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 comply with this difference.

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

F.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 CompactCom 30 example drivers from HMS comply with this difference.

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

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

F.4 Network Specific — EtherNet/IP

F.4.1 Network Object (03h)

Attribute #1, Network Type	The 30-series module is available in two network type versions, either with “Beacon based DLR” (Highest performance) or with “Announce based DLR” which both are Ethernet redundancy protocols. The 40-series is only available with “Beacon based DLR”. The network type value differs between the versions.

Value	Network Type	Anybus CompactCom Product
0085h	EtherNet/IP, No DLR	30-series 1-port
009Ch	EtherNet/IP, Announce Based DLR	30-series 2-port
009Bh	EtherNet/IP, Beacon Based DLR	30-series and 40-series
00ABh	EtherNet/IP, Beacon Based DLR + IIoT	40–series

F.4.2 EtherNet/IP Host Object (F8h)

Attribute	Default	Anybus CompactCom Product	Comment
#2, Device Type	0000h	30-series, EtherNet/IP, No DLR	If the attribute is implemented in the host application, it overrides the default value and there is no difference between the 30-series and the 40-series. If the attribute is not implemented, the default value is used.
	0000h	30-series, EtherNet/IP, Announce Based DLR	
	002Bh	30-series, EtherNet/IP, Beacon Based DLR	
	002Bh	40-series, EtherNet/IP, Beacon Based DLR	
#3, Product Code	0063h	30-series, EtherNet/IP, No DLR	If the attribute is implemented in the host application, it overrides the default value and there is no difference between the 30-series and the 40-series. If the attribute is not implemented, the default value is used.
	002Eh	30-series, EtherNet/IP, Announce Based DLR	
	0036h	30-series, EtherNet/IP, Beacon Based DLR	
	0037h	40-series, EtherNet/IP, Beacon Based DLR	
#6, Product Name	Anybus-CC EtherNet/IP	30-series, EtherNet/IP, No DLR	If the attribute is implemented in the host application, it overrides the default value and there is no difference between the 30-series and the 40-series. If the attribute is not implemented, the default value is used.
	CompactCom EtherNet/IP(TM) 2P	30-series, EtherNet/IP, Announce Based DLR	
	Anybus-CC EIP (2-Port) BB DLR	30-series, EtherNet/IP, Beacon Based DLR	
	Anybus CompactCom 40 EtherNet/IP(TM)	40-series, EtherNet/IP, Beacon Based DLR	
Attribute #27, Producing Instance Map	See comment		Attribute removed in the 40-series (only available in the 30-series EtherNet/IP Beacon Based DLR). The CompactCom will never request this attribute. Replaced by the functionality in the Assembly Mapping Object (EBh). If this attribute is used, the Assembly Mapping object must be implemented instead.
Attribute #28, Consuming Instance Map	See comment		Attribute removed in the 40-series (only available in the 30-series EtherNet/IP Beacon Based DLR). The CompactCom will never request this attribute. Replaced by the functionality in the Assembly Mapping Object (EBh). If this attribute is used, the Assembly Mapping object must be implemented instead.

EtherNet/IP functionality

Max Message Connections	The maximum number of simultaneous Class 3 connections are 16 in the 30-series and 6 in the 40-series. No change is needed in the host application.
EtherNet/IP Encapsulation Sessions	The maximum number of simultaneous encapsulation sessions are 48 in the 30-series and 15 in the 40-series. No change is needed in the host application.

F.4.3 EDS file (Electronic Datasheet file used by configuration tool)

EDS file Generator Tool

An EDS-generator for automatic EDS-file generation up to date with the differences below. The EDS-generator only works with the 40-series, version 1.30 and later.

The generator can be downloaded from www.anybus.com/starterkit40: .

Keywords

The following keywords differs between the 30-series and the 40-series. The EDS generator reflects this change.

Keyword	Comments
Capacity->MaxCIPConnections	Removed in 40-series – replaced by: MaxMsgConnections and MaxIOConnections (see below)
Capacity->MaxMsgConnections	New keyword in the 40-series, Value: 6
Capacity->MaxIOConnections	New keyword in the 40-series, Value: 4

G Copyright Notices