Anybus® CompactCom™ 40

Modbus-TCP ®

NETWORK GUIDE

SCM-1202-027 1.1 ENGLISH

HMS Connecting Devices™

Important User Information

Liability

Every care has been taken in the preparation of this document. Please inform HMS Industrial Networks AB of any inaccuracies or omissions. The data and illustrations found in this document are not binding. We, HMS Industrial Networks AB, reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks AB. HMS Industrial Networks AB assumes no responsibility for any errors that may appear in this document.

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

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

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

Intellectual Property Rights

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

Table of Contents

Page

This page intentionally left blank

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 Modbus-TCP. The document describes the features that are specific to Anybus CompactCom 40 Modbus-TCP. 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 Modbus-TCP specific functionality is to be used, in-depth knowledge of Modbus-TCP networking internals and/or information from the official Modbus-TCP specifications may be required. In such cases, the persons responsible for the implementation of this product should either obtain the Modbus-TCP 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

1.3 Document History

Version	Date	Description
1.0	2017-01-17	FM to DOX, change of document number from HMSI-27-294 to SCM-1202-027. Version numbering restarted. M12 connectors added Minor corrections
1.1	2017-07-11	Added appendix on backward compatibility Updated description of module status LED

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 Trademark Information

2 About the Anybus CompactCom 40 Modbus-TCP

2.1 General

The Anybus CompactCom 40 Modbus-TCP communication module provides instant Ethernet and Modbus-TCP 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 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 ports
- Ethernet connectors or M12 connectors
- 10/100 Mbit, full/half duplex operation
- Modbus-TCP server/slave (up to 4 simultaneous connections)
- Max. read process data: 1536 bytes
- Max. write process data: 1536 bytes
- Max. process data (read + write, in bytes): 3072 bytes
- Web server w. customizable content
- FTP server
- Email client
- Server Side Include (SSI) functionality
- JSON functionality
- Customizable Identity Information
- Transparent Socket Interface
- Modular Device functionality

3 Basic Operation

3.1 Software Requirements

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

- The total number of ADIs that can be represented on the network depends on their size. By default, ADIs with instance numbers 1...3839 can be accessed from the network, each with a size of up to 32 bytes.
- ADI names, types and similar attributes cannot be accessed viaModbus-TCP. They are however represented on the network through the built in web server.
- A network write access of an ADI mapped to process data will result in a corresponding write access of the process data buffer of the Anybus CompactCom 40 Modbus-TCP.
- A network read access of an ADI, even if it is mapped to process data, will result in a corresponding Get_Attribute command towards the application.
- Modbus-TCP reset requests are not supported.
- Up to 5 diagnostic instances (See Diagnostic Object) can be created by the host application during normal conditions. An additional 6th instance may be created in event of a major fault.
- Modbus-TCP 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 (exactly what this means in practice depends on the implementation and the actual installation).
- The use of advanced Modbus-TCP specific functionality may require in-depth knowledge in Modbus-TCP networking internals and/or information from the official Modbus-TCP 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...

- *Application Data (ADIs), p. 13*
- *Diagnostic Object (02h), p. 63*
- 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.

3.2 Device Customization

3.2.1 Modbus-TCP Implementation

By default, a “Read Device Identification” request returns the following information:

Vendor Name	“HMS”
Product Code:	“Anybus CompactCom 40 Modbus-TCP”
Major Minor Rev.:	The current firmware version of the product
Vendor URL:	(no information returned by default)
Product Name:	(no information returned by default)
Model Name:	(no information returned by default)
User Application Name:	(no information returned by default)

It is possible to customize this information by implementing the Modbus Host Object. See *Modbus Host Object (FAh), p. 99* for more information.

3.2.2 Web Interface

The web interface can be fully customized to suit a particular application. Data and web pages are stored in a flash-based file system, which can be accessed using any standard FTP client or the File System Interface Object.

See also...

- *File System, p. 14*
- *FTP Server, p. 22*
- *Web Server, p. 24*

3.2.3 Socket Interface (Advanced Users Only)

The built in socket interface allows additional protocols to be implemented on top of TCP/IP. Data is structured by the application and is then embedded within the Ethernet frames. The host application can open network connections of its own to other nodes on the network, e.g. if you want to connect to another server or use a web server of your own.

See also..

- *Socket Interface Object (07h), p. 74* (Anybus Module Object)
- *Message Segmentation, p. 89*

3.3 Communication Settings

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

In this case, this includes...

Ethernet Interface Settings	By default, the module is set to autonegotiate the physical link settings. It is, however, possible to force the module to use a specific setting if necessary.
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.
Modbus-TCP Connection Timeout	This setting specifies how long a Modbus-TCP connection may be idle before it is closed by the module (default is 60 seconds).
Process Active Timeout	This value specifies how long the module shall stay in the 'PROCESS_ACTIVE' state after receiving a Modbus-TCP request. See*Network Configuration Object (04h), p. 65* for more information. **Note**: This value can be accessed from the Modbus registers. **Note**: This value affects the behavior of the SUP-bit. See *SUP-Bit Definition, p. 108*.

See also...

Web Server, p. 24

Network Configuration Object (04h), p. 65 (Anybus Module Object)

Secure HICP (Secure Host IP Configuration Protocol), p. 109

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.

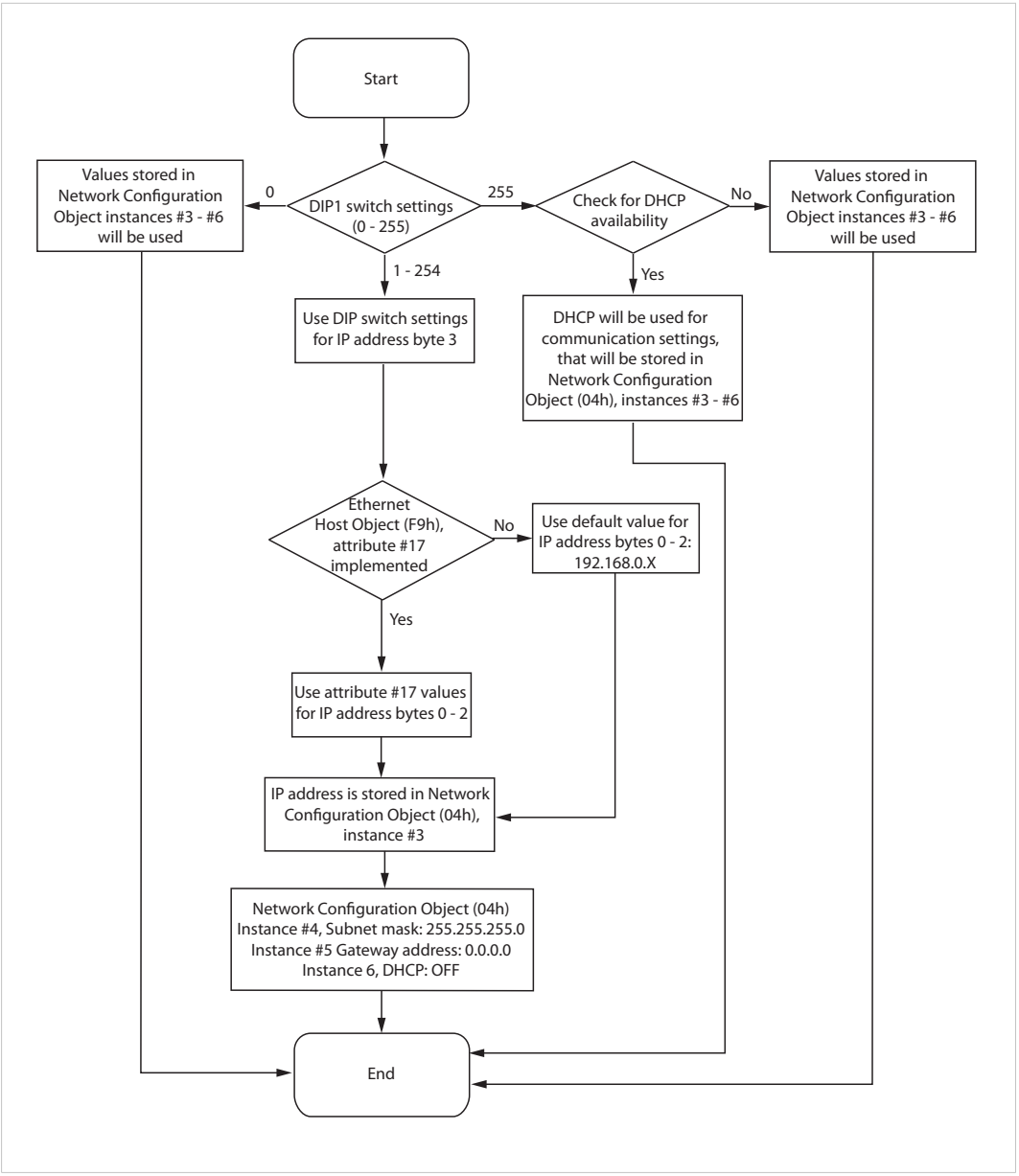


Fig. 1

See also ...

- *Ethernet Host Object (F9h), p. 102*
- Anybus CompactCom M40 Hardware Design Guide
- *Network Configuration Object (04h), p. 65*

3.4 Diagnostics

Each instance within the Diagnostic Object (02h) is represented on the network as a dedicated entry in the Modbus register map (see *Input Registers (3x), p. 17*).

Note that since each entry corresponds *directly* to a specific diagnostic instance, it is possible to have “empty” diagnostic entries in the register map (when read, such entries will return zeroes).

See also...

Input Registers (3x), p. 17

Diagnostic Object (02h), p. 63

3.5 Network Data Exchange

3.5.1 General

It is important to notice that various register areas might have different response times. Generally queries directed at the process data registers will be answered more quickly than those directed at the ADI-related registers since the former are directly processed by the module itself whereas the latter are forwarded to the application, which must respond before the module can respond to the master. In the latter case this will influence the allowable timeout time for the master to use for these registers.

3.5.2 Application Data (ADIs)

As mentioned previously, the total number of ADIs that can be represented on the network depends on their size. By default, ADIs with instance numbers 1...3839 can be accessed from the network, each with a size of up to 32 bytes. It is possible to alter this ratio by changing the number of ADI indexing bits. See attribute #9, in the *Modbus Host Object (FAh), p. 99*.).

Example 1 (Default Settings)

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

Holding Register #	ADI No.
1010h... 101Fh	1
1020h... 102Fh	2
1030h... 103Fh	3
1040h... 104Fh	4
...	...
FFE0h... FFEFh	3838
FFF0h... FFFFh	3839

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

Example 2 (Customized Implementation)

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

Holding Register #	ADI No.
1010h... 102Fh	1
1030h... 104Fh	2
1050h... 106Fh	3
1070h... 108Fh	4
...	...
FFB0h... FFCFh	1918
FFD0h... FFEFh	1919

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

3.5.3 Process Data

Modbus does not feature a dedicated cyclic data channel in the same sense as many other networks. In the Anybus CompactCom 40 implementation, process data can however still be accessed from the network via dedicated entries in the Modbus register map.

Process data can be accessed on a bit by bit basis (as Coils & Discrete Inputs) - *or* - as 16 bit entities (Holding Registers & Input Registers).

For natural reasons, writing to the write process data register area has no effect, and reading unused register locations will return zeroes.

Example

Each 16-bit Modbus register contains 2 bytes from the process data at the corresponding address, i.e. Modbus register N holds process data byte (N*2) in the low byte and (N*2 + 1) in the high byte.

Process Data			**Modbus Register**		
Byte	**Type**	**Value**	**Register**	**Value**	**Comment**
0	UINT16	1234h	0	1234h	-
1					
2	UINT8	00h	1	FF00h	Two bytes from the process data in one register.
3	UINT8	FFh			
4	UINT32	112233-44h	2	3344h	LSB * A 32-bit type occupies two Modbus * registers. MSB
5					
6			3	1122h	
7					
8	BOOL[3]	01h	4	0001h	-
9		00h			
10		01h	5	3401h	-
11	UINT16	1234h			
12			6	0012h	High byte from unmapped process data is set to zero.

3.6 File System

3.6.1 Overview

The fAnybus CompactCom 40 Modbus-TCP 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.

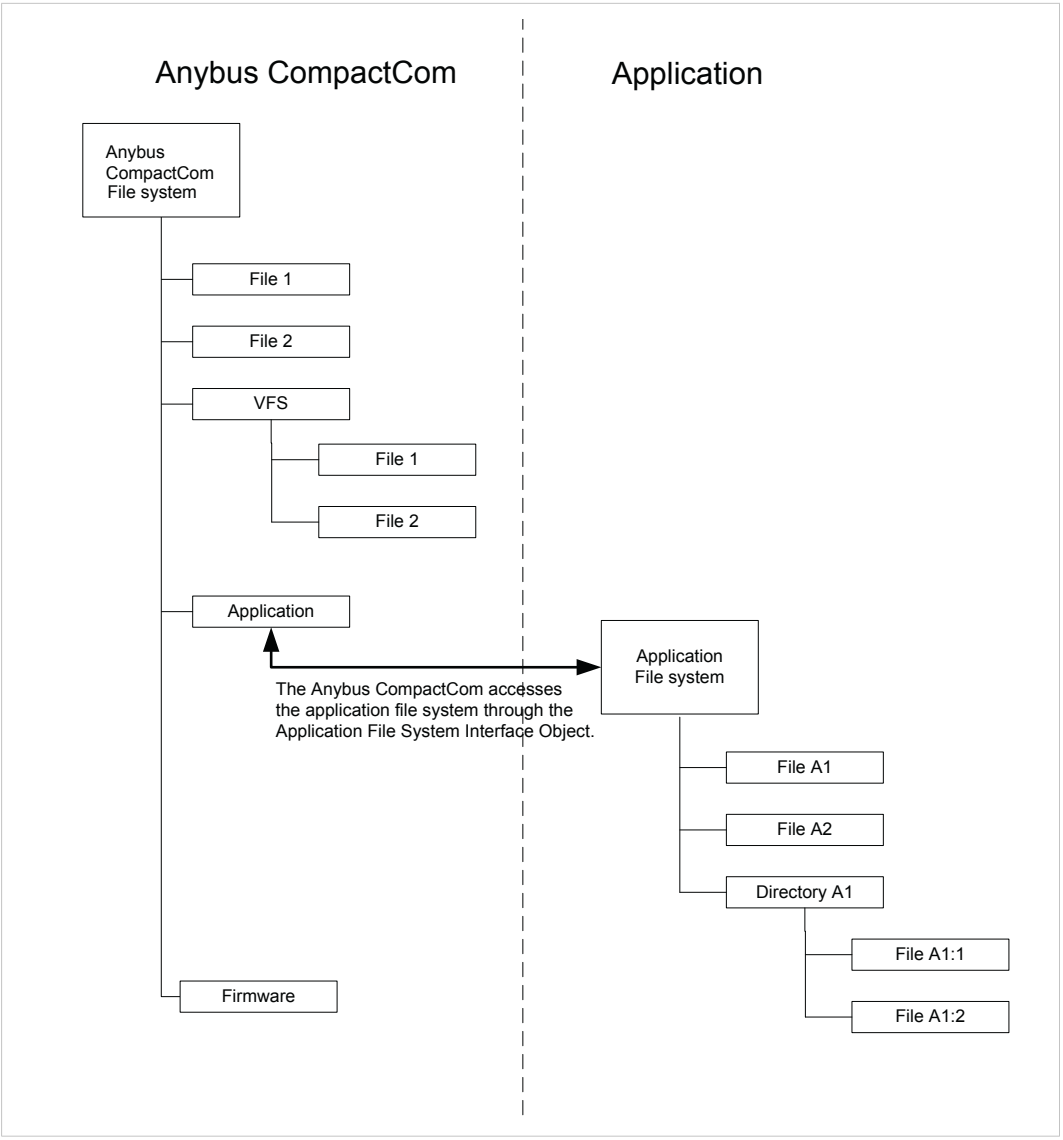


Fig. 2

3.6.2 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). If longer filenames are used, less than 511 directories and files can be stored. 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. 22*
- *Web Server, p. 24*
- *E-mail Client, p. 31*
- *Server Side Include (SSI), p. 41*
- *Anybus File System Interface Object (0Ah), p. 96*
- *Application File System Interface Object (EAh), p. 98*

> ! *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

3.6.3 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 Modbus-TCP Register Implementation

4.1 Holding Registers (4x)

Range	Contents	Notes
0000h...02FFh	Read Process Data (1536 bytes)	-
0300h...07FFh	Reserved	-
0800h...0AFFh	Write Process Data (1536 bytes)	-
0B00h...0FFFh	Reserved	-
1000h...1002h	Reserved	-
1003h	Process Active Timeout	See *Network Configuration Object (04h), p. 65*
1004h	Enter/Exit Idle Mode	0: Not Idle, >0: Idle
1005h...100Fh	Reserved	-
1010h...101Fh	ADI Number 1	See *Application Data (ADIs), p. 13*
1020h...102Fh	ADI Number 2	
...	...	
FFF0h...FFFFh	ADI Number 3839	

4.2 Input Registers (3x)

Range	Contents	Notes
0000h...02FFh	Write Process Data	-
0300h...07FFh	Reserved	-
0800h	Diagnostic Event Count	Number of pending diagnostic events. There may be "gaps" between active diagnostic events. Inactive diagnostic events return 0000h when read.
0801h	Diagnostic Event #1	These registers corresponds to instances in the Diagnostic Object (02h), see *Diagnostic Object (02h), p. 63*. High byte = Severity Low byte = Event Code
0802h	Diagnostic Event #2	
0803h	Diagnostic Event #3	
0804h	Diagnostic Event #4	
0805h	Diagnostic Event #5	
0806h	Diagnostic Event #6	

4.3 Coils (0x)

Range	Contents	Notes
0000h...2FFFh	Read Process Data	-
3000h...7FFFh	Reserved	-

4.4 Discrete Inputs (1x)

Range	Contents	Notes
0000h...2FFFh	Write Process Data	-
3000h...07FFh	Reserved	-

5 Modbus-TCP Functions

The following Modbus-TCP functions are implemented in the module:

#	Function
1	Read Coils
2	Read Discrete Inputs
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Register
15	Write Multiple Coils
16	Write Multiple Registers
23	Read/Write Multiple Registers
43/14	Read Device Identification

Exception Codes:

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

See also...

Modbus Host Object (FAh), p. 99

5.1 Read Coils

Function Code: 1

Register Type: 0x (Coils)

Details

This function is mapped to the Read Process data as follows:

Coil #	Process Data Byte #	Bit #
0000h	0000h	0
0001h		1
0002h		2
0003h		3
...		...
0007h		7
0008h	0001h	0
0009h		1
000Ah		2
000Bh		3
...		...
000Fh		7
...	...	...
2FF8h	05FFh	0
2FF9h		1
2FFAh		2
2FFBh		3
...		...
2FFFh		7

5.2 Read Discrete Inputs

Function Code: 2

Register Type: 1x (Discrete Inputs)

Details

This function is mapped to the Write Process data; the mapping is otherwise identical to that of the “Read Coils” function described above.

5.3 Read Holding Registers

Function Code: 3

Register Type: 4x (Holding Registers)

Details

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

5.4 Read Input Registers

Function Code: 4

Register Type: 3x (Input Registers)

Details

Mapped to Write Process Data and diagnostic events.

5.5 Write Single Coil

Function Code: 5

Register Type: 0x (Coils)

Details

This function is mapped to the Read Process data, and the mapping is identical to that of the “Read Coils” function described above.

5.6 Write Single Register

Function Code: 6

Register Type: 4x (Holding Registers)

Details

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

5.7 Write Multiple Coils

Function Code: 15

Register Type: 0x (Coils)

Details

This function is mapped to the Read Process data, and the mapping is identical to that of the “Read Coils” function described above.

5.8 Write Multiple Registers

Function Code: 16

Register Type: 4x (Holding Registers)

Details

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

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

5.9 Read/Write Multiple Registers

Function Code: 23

Register Type: 4x (Holding Registers)

Details

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

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

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

The write operation is performed before the read. If there is an overlap in the read and write ranges, the newly written data will be returned by the read operation.

5.10 Read Device Identification

Function Code: 43 (subcode 14)

Register Type: -

Details

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

Identification strings are extracted from the host application via the *Modbus Host Object (FAh), p. 99*. If this object is not implemented, the default identification strings will be returned

6 FTP Server

6.1 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.

6.2 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).

6.3 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.

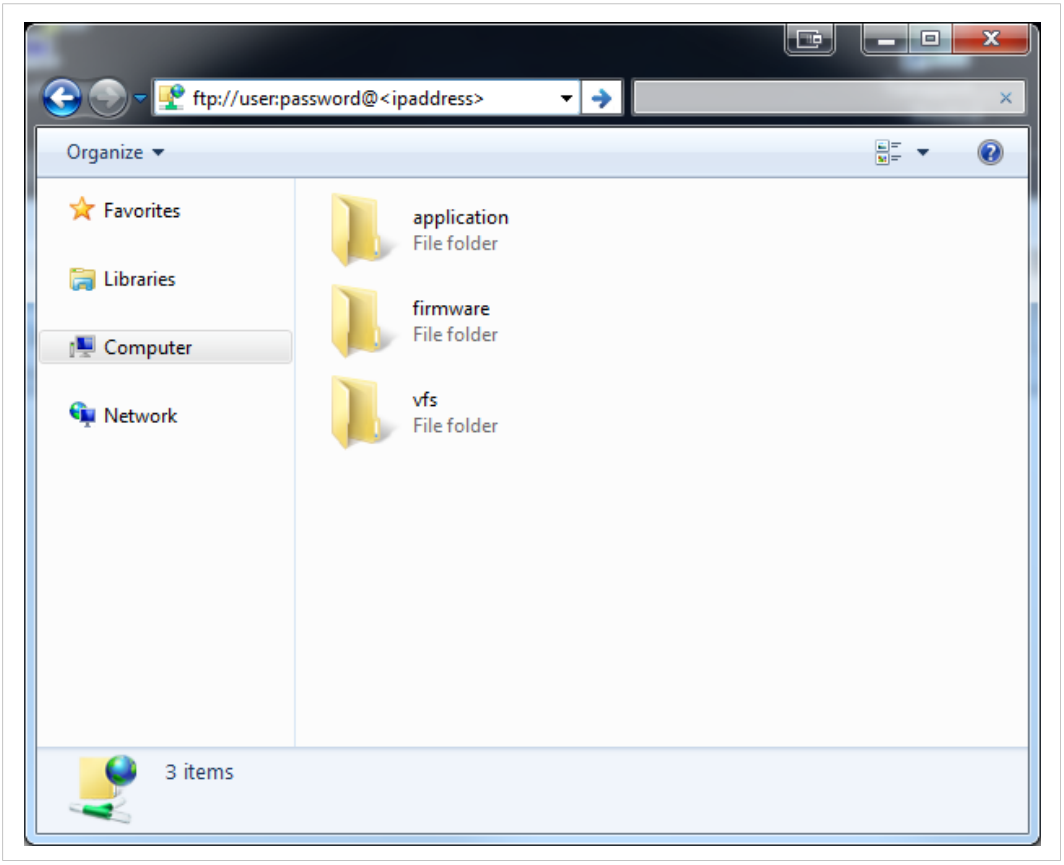


Fig. 3

7 Web Server

7.1 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. 22*
- *Server Side Include (SSI), p. 41*
- *JSON, p. 32*
- *Ethernet Host Object (F9h), p. 102*

7.2 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. 14*
- *Anybus File System Interface Object (0Ah), p. 96*

7.2.1 Network Configuration

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

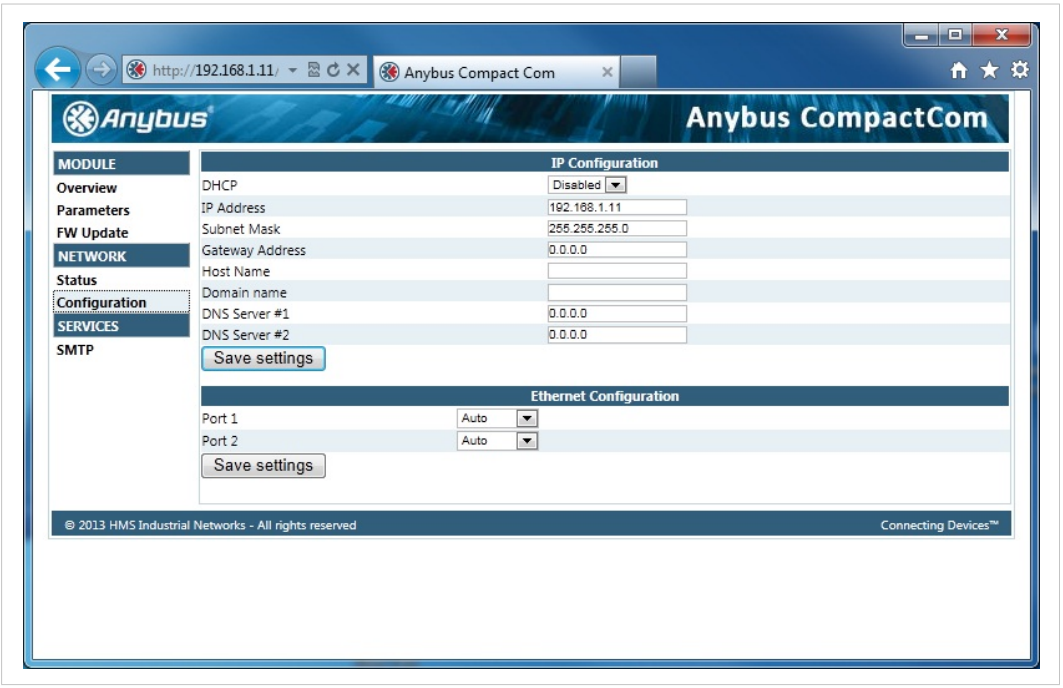


Fig. 4

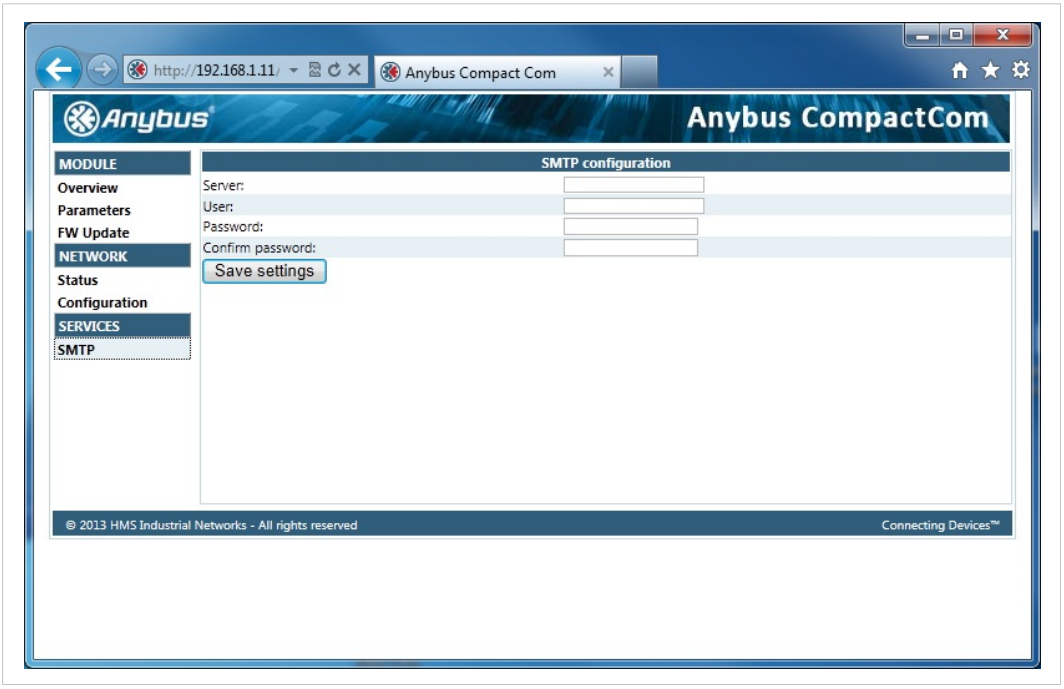


Fig. 5

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	

7.2.2 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.

Modbus Statistics	Description
Modbus connections	Number of active Modbus connections.
Connection ACKs	Number of accepted Modbus connections.
Connection NACKs	Number of refused Modbus connections.
Connection timeouts	Number of Modbus connections closed due to connection timeout.
Process active timeouts	Number of times a "process active timeout" has occurred.
Processed messages	Processed Modbus messages.
Incorrect messages	Incorrect Modbus messages.

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.

7.3 Server Configuration

7.3.1 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.

7.3.2 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

7.3.3 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.

7.3.4 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.

8 E-mail Client

8.1 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).

8.2 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).

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. 40* 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>. The argument <port> is either 1 or 2.

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 Server Side Include (SSI)

10.1 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.

10.2 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)

10.3 Command Functions

10.3.1 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

10.3.2 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’.

10.3.3 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”.

10.3.4 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. 60*

10.3.5 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)

10.3.6 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”.

10.3.7 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:

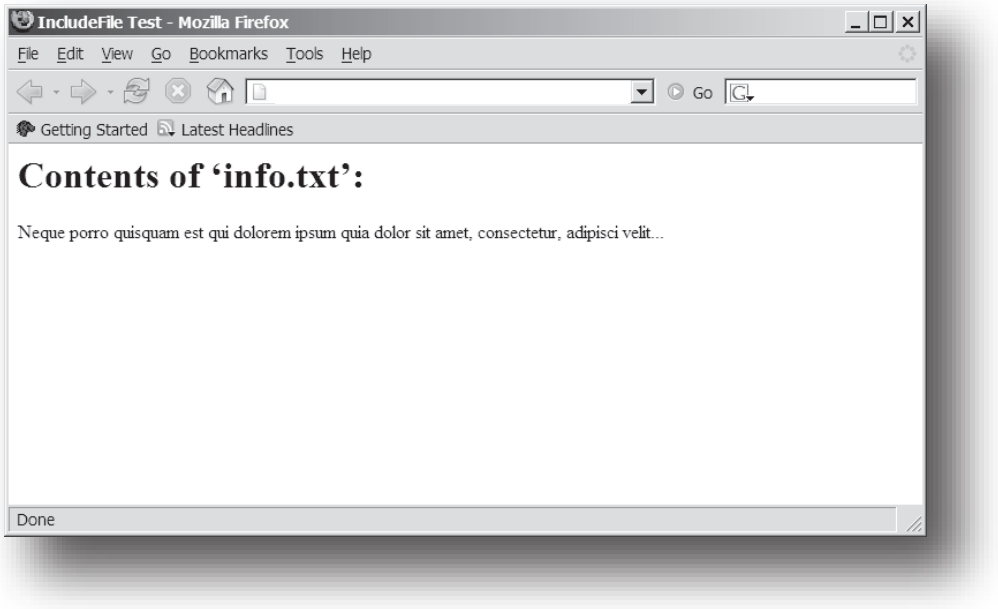


Fig. 6

See also...

- *Include File, p. 41*

10.3.8 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”.

10.3.9 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. 56*

Default Output

Scenario	Default Output
Success	(printf() result)
ABCCMessage error	ABCCMessage error string (*Errors, p. 59*)

Example

See ..

- *ABCCMessage(), p. 56*
- *Example (Get_Attribute):, p. 58*

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

10.3.10 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. 56*

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. 59*)

Example

See also...

ABCCMessage(), p. 56

Example (Set_Attribute):, p. 58

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

10.4 Argument Functions

10.4.1 General Information

Argument functions are supplied as parameters to certain command functions.

General Syntax:

(Syntax depends on context)

Argument Functions:

Function	Description
ABCCMessage()	-

10.4.2 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. 60

10.5 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. 59*
[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. 46*

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 Modbus-TCP.

Standard Objects:

Network Specific Objects:

11.2 Anybus Object (01h)

Category

Basic, extended

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*.

See also ...

- *Diagnostics, p. 12*

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	

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	0093h
2	Network type string	Get	Array of CHAR	“Ethernet Modbus-TCP”
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. 11*
- *E-mail Client, p. 31*

Supported Commands

Object:	Get_Attribute
	Reset
Instance:	Get_Attribute
	Set_Attribute
	Get_Enum_String

Object Attributes (Instance #0)

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Network Configuration”
2	Revision	Get	UINT8	01h
3	Number of instances	Get	UINT16	0012h (18)
4	Highest instance number	Get	UINT16	0016h (22)

(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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after reset. (Multilingual, see page 73) 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 (default)

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 73)
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 73) 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 73) 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

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 73)
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 73) 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 73) 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

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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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 73)
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	Any change is valid after 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”
6	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” (default) 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”
6	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” (default) 01h “MDI” 02h “MDIX”

Instance Attributes (Instances #18 and #19)

These instances are reserved for future attributes.

Instance Attributes (Instance #20, Modbus connection timeout)

This instance holds the settings for the Modbus connection timeout. Changes will be applied to new connections. Existing connections will use the previous timeout value.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Conn tmo” (Multilingual, see page *73*)
2	Data type	Get	UINT8	05h (= UINT16)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	UINT16	Value: Meaning (seconds): 0 Timeout disabled 60 Default
6	Configured Value	Get	UINT16	Holds the configured value, which will be written to attribute #5. Value: Meaning (seconds): 0 Timeout disabled 60 Default

Instance Attributes (Instance #21, Process active timeout)

This instance holds the settings for the Process active timeout. Changes have immediate effect. See *Communication Settings, p. 11* for more information.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Process tmo” (Multilingual, see page *73*)
2	Data type	Get	UINT8	05h (= uint16)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	UINT16	Default = 0 (milliseconds, disable timeout)
6	Configured Value	Get	UINT16	Holds the configured value, which will be written to attribute #5. Default = 0 (milliseconds, disable timeout)

Instance Attributes (Instance #22, Word order)

This instance holds the Word order settings. Value is used after module reset.

#	Name	Access	Data Type	Description
1	Name	Get	Array of CHAR	“Word order” (Multilingual, see page *73*)
2	Data type	Get	UINT8	04h (= UINT8)
3	Number of elements	Get	UINT8	01h (one element)
4	Descriptor	Get	UINT8	07h (read/write/shared access)
5	Value	Get/Set	UINT8	Value: — Meaning: 0 — Little endian (default) 1 — Big endian Other values will be translated to 0 (default).
6	Configured Value	Get	UINT8	Holds the configured value, which will be written to attribute #5. Value: — Meaning: 0 — Little endian (default) 1 — Big endian Other values will be translated to 0 (default).

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
20	Conn tmo	Verb. Tmo	Tout Conexion	Tout Conn.	Conn tmo
21	Process tmo	Prozess Tmo	Tout Proceso	Tout Processo	Process tmo
22	Word order	Wortfolge	Orden palabra	Ordine "word"	Ordre - mots

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. 89*.

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: 00h 01h 02h	Mode: Shutdown receive channel Shutdown send channel 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.

Application 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 receive 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. 89*).

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. 89*
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. 89*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control bits	For more information, see *Message Segmentation, p. 89*
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. 89*).

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. 89*
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. 89*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control bits	For more information, see *Message Segmentation, p. 89*
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. 89*).

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. 89*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control	(For more information, see *Message Segmentation, p. 89*)
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. 89*).

- 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. 89*).

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]	Segmentation Control	For more information, see *Message Segmentation, p. 89*
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.

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	-
4	Highest instance no.	Get	UINT16	-

Instance Attributes (Instance #1)

#	Name	Access	Data Type	Description
1	MAC Address	Get	Array of UINT8	Current MAC address. See also “Ethernet Host Object (F9h)”

12 Host Application Objects

12.1 General Information

This chapter specifies the host application object implementation in the module. The Application Data Object is mandatory to implement. The other objects listed here may optionally be implemented within the host application firmware to expand the implementation.

Standard Objects:

- Application Object (FFh) - (see Anybus CompactCom 40 Software Design Guide)
- Application Data Object (FEh) - (see Anybus CompactCom 40 Software Design Guide)

Network Specific Objects:

- *Application File System Interface Object (EAh), p. 98*
- *Ethernet Host Object (F9h), p. 102*
- *Modbus Host Object (FAh), p. 99*

12.2 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*.

12.3 Modbus Host Object (FAh)

Category

Extended

Object Description

This object implements Modbus related settings in the host application.

The implementation of this object is optional; the host application can support none, some, or all of the attributes specified below. The module will attempt to retrieve the values of these attributes during 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]”).

See also ...

- Anybus CompactCom 40 Software Design Guide, “Error Codes”

Supported Commands

Object:	Get_Attribute
	Process-modbus-message
Instance:	Get_Attribute

Object Attributes (Instance #0)

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

Instance Attributes (Instance #1)

Extended

Changes to these attributes during runtime will have no effect. If an attribute is not implemented the default value will be used.

#	Name	Access	Data Type	Default Value	Comment
1	Vendor name	Get	Array of CHAR	“HMS”	These settings will be returned in response to a “Read Device Identification” request. Attribute 2 and 3 will also be used as identification on the web site and for SHICP (IPconfig). The maximum allowed length of each string is 244 bytes; strings exceeding this length will be ignored and their default value will be used instead. See also ... • *Web Server, p. 24* • *Read Device Identification, p. 21*
2	Product Code	Get	Array of CHAR	“Anybus CompactCom 40 Modbus-TCP”	
3	Major Minor Revision	Get	Array of CHAR	(firmware rev.)	
4	Vendor URL	Get	Array of CHAR	“ ”	
5	Product name	Get	Array of CHAR	“ ”	
6	Model name	Get	Array of CHAR	“ ”	
7	User Application Name	Get	Array of CHAR	“ ”	
8	Device ID	Get	Array of UINT8	-	Not used
9	No. of ADI indexing bits	Get	UINT8	04h	Value: / Meaning: 00h each ADI = 1 Modbus register 01h each ADI = 2 Modbus registers 02h each ADI = 4 Modbus registers 03h each ADI = 8 Modbus registers 04h each ADI = 16 Modbus registers 05h each ADI = 32 Modbus registers 06h each ADI = 64 Modbus registers 07h each ADI = 128 Modbus registers (other) (invalid) (see *Application Data (ADIs), p. 13*)
10	Enable Modbus message forwarding	Get	Bool	False	If true, all Modbus messages, addressed (or broadcast) to this node are routed to the application.
11	Modbus read/write registers command offset	Get	SINT16[2]	[0x0000, 0x0000]	These values provides possibility to use offsets for the various read/write holding register commands Format: [READ, WRITE]

Command Details: Process-modbus-message

Category

Extended

Details

Command Code:	10h
Valid for:	Object

Description

If enabled, this command routes Modbus/TCP communication to the host application.

- Command Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(ignore)
CmdExt[1]		
MsgData[0... n]	Modbus message frame (Query)	-

- Response Details

Field	Contents	Comments
CmdExt[0]	(reserved)	(set to zero)
CmdExt[1]		
MsgData[0... n]	Modbus message frame (Response)	-

The response data size must not exceed 254 bytes, if more data is returned, no Modbus response message will be sent to the originator of the request.

If the response contains no data, no Modbus response will be sent to the originator of the request.

12.4 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	Access	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 Ethernet/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 only valid for Ethernet/IP 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)

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.

This page intentionally left blank

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.

This bit is set when in PROCESS_ACTIVE, and only if the Process active timeout value is greater than zero (0).

B.2 Anybus State Machine

The table below describes how the Anybus state machine relates to the Modbus-TCP network

Anybus State	Implementation	Comment
WAIT_PROCESS	Waiting for Modbus requests. The module shifts to PROCESS_ACTIVE when a Modbus request is received.	-
ERROR	IP address conflict.	This state is only possible if Address Conflict Detection (ACD) is enabled in the Ethernet Host object (enabled by default).
PROCESS_ACTIVE	The module shifts to WAIT_PROCESS if no requests are received within the time stated by Process Active Timeout (see Instance #21 in *Network Configuration Object (04h), p. 65*.	-
IDLE	The IDLE state can be entered/exited by writing to the Modbus Enter/Exit idle state register at address 1004h.	
EXCEPTION	Any Modbus requests will be ignored.	

B.3 Application Watchdog Timeout Handling

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

C Secure HICP (Secure Host IP Configuration Protocol)

C.1 General

The Anybus CompactCom 40 Modbus-TCP 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.

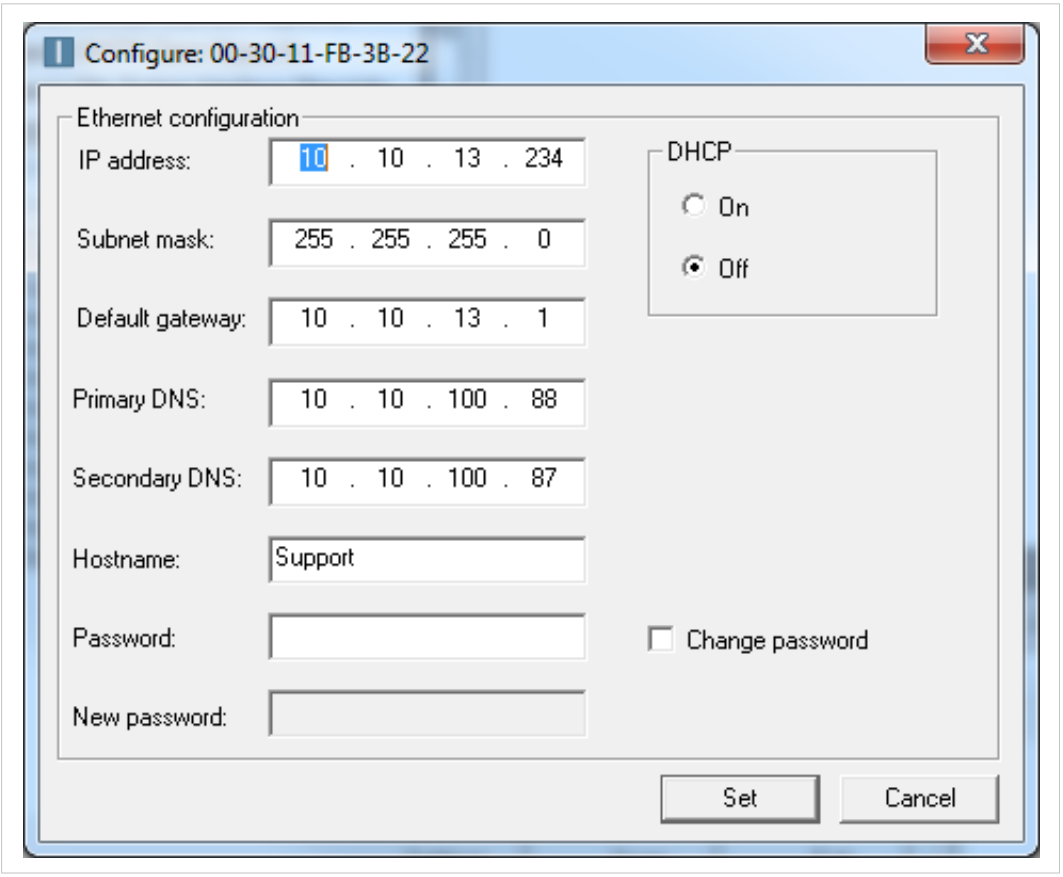


Fig. 7

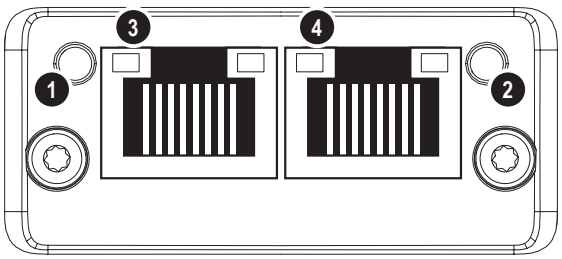
Validate the new settings by clicking **Set**, or click **Cancel** to cancel all changes. Optionally, the configuration can be protected from unauthorized access by a password. To enter a password, check the **Change password** checkbox and enter the password in the **New password** text field.

D Technical Specification

D.1 Front View

D.1.1 Front View (Ethernet Connectors)

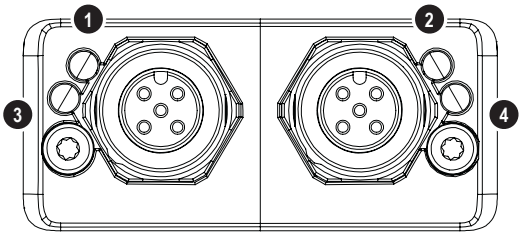
#	Item	Connector
1	Network Status LED	Ethernet, 45
2	Module Status LED	
3	Link/Activity LED (port 1)	
4	Link/Activity LED (port 2)	



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 IP address or in state EXCEPTION
Green	At least one Modbus message received
Green, flashing	Waiting for first Modbus message
Red	IP address conflict detected, FATAL ERROR
Red, flashing	Connection timeout. No Modbus message has been received within the configured “process active timeout” time

A test sequence is performed on this LED during startup.

D.1.4 Module Status LED

LED State	Description
Off	No power
Green	Normal operation
Red	Major fault (including Anybus exception), FATAL
Red, flashing	Minor fault
Alternating red/green	Firmware update from file system in progress

A test sequence is performed on this LED during startup.

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 Fatal Error

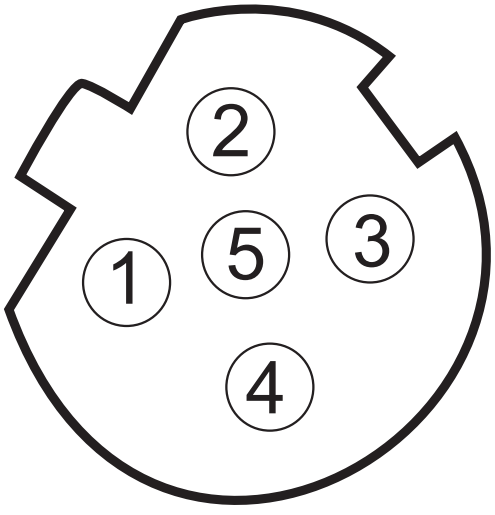
If both the Network Status LED and the Module Status LED are red, a fatal error has occurred.

D.1.7 Ethernet Interface

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

D.1.8 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 Modbus-TCP 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 Modbus-TCP is designed to fulfil the requirements of a Class B module.

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 Modbus-TCP will remain as a Class B module.

For more information about the power consumption classification used on the Anybus CompactCom 40 platform, consult the general *Anybus CompactCom 40 Hardware Design Guide*.

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 40 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 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 Modbus-TCP, when designing with both series in one application, or when adapting a 30 series application for the 40 series.

E.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.

E.2 Hardware Compatibility

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

E.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. 8 Anybus CompactCom M30/M40

E.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.

E.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.

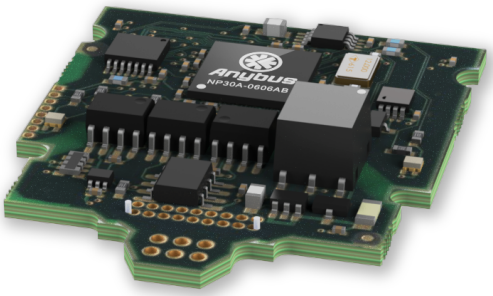


Fig. 9 Anybus CompactCom B30

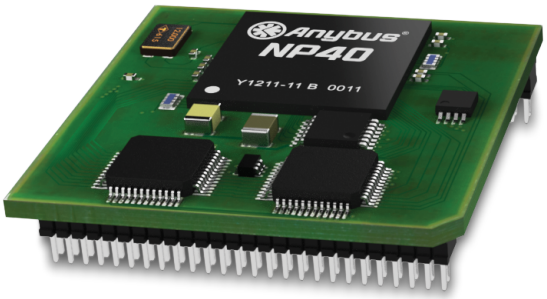


Fig. 10 Anybus CompactCom B40–1 (not for migration)

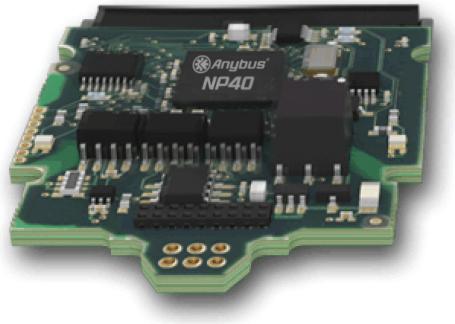


Fig. 11 Anybus CompactCom B40–2

E.2.4 Host Application Interface

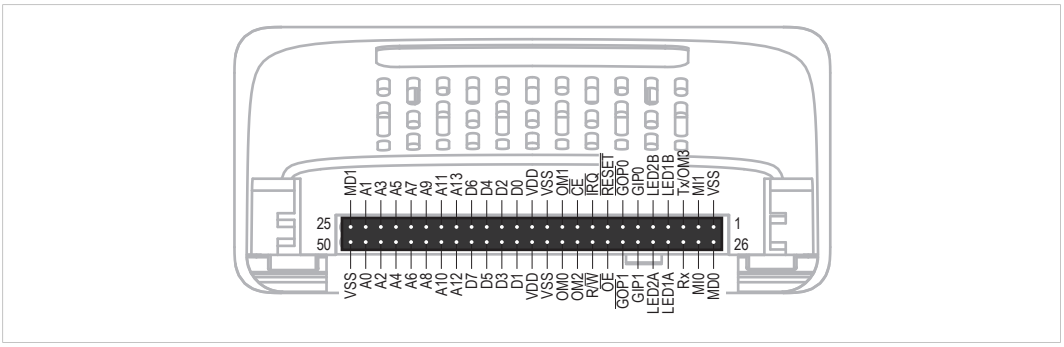


Fig. 12

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.

E.3 General Software

E.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”

E.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.

E.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.

E.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. 117* • *GOP[0..1]/LED4[A..B], p. 117*

E.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”.

E.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”.

E.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”.

E.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.

E.4 Network Specific — Modbus-TCP

E.4.1 Modbus Registers

Rearrangements have been made in the Modbus register map, because process data sizes have been increased to 1536 bytes in each direction. An existing PLC configuration need to be changed to use the new addresses. **No difference on the application side.**

Contents	30-series Modbus Address	40-series Modbus Address
Holding Registers (4x)		
Read Process Data	0000h-00FFh	0000h-02FFh
Write Process Data	0100h-01FFh	0800h-0AFFh
Process Active Timeout	0203h	1003h
Enter/Exit Idle Mode	0204h	1004h
ADI Number 1	0210h-021Fh	1010h-101Fh
ADI Number 2	0220h-022Fh	1020h-102Fh
ADI Number 3839		FFF0h-FFFFh
Input Registers (3x)		
Write Process Data	0000h-00FFh	0000h-02FFh
Diagnostic Event Count	0100h	0800h
Diagnostic Event #1	0101h	0801h
Diagnostic Event #2	0102h	0802h
Diagnostic Event #3	0103h	0803h
Diagnostic Event #4	0104h	0804h
Diagnostic Event #5	0105h	0805h
Diagnostic Event #6	0106h	0806h
Coils (0x)		
Read Process Data	0000h-0FFFh	0000h-2FFFh
Discrete Inputs (1x)		
Write Process Data	0000h-0FFFh	0000h-2FFFh

E.4.2 BOOL arrays

Process data mapped BOOL arrays are not compressed to bit-fields on the network in the 40-series, but handled as a normal 8-bit datatype. To create bit-arrays in the 40-series, use the new datatypes BITx instead.

E.4.3 Network Configuration Object (04h)

The instances in the Network Configuration Object have been rearranged for the Ethernet based modules for consistency. Network specific instances are moved to instance number 20 and onwards. This is done to increase the number of instances in the section that is not network specific.

If the host application is using any of the parameters below, the software must be updated to use the new instance numbers.

Parameter Name	30-series Instance #	40-series Instance #
Modbus Connection Timeout	9	20
Process Active Timeout	10	21
DNS1	11	9
DNS2	12	10
Host Name	13	11
Domain Name	14	12
SMTP Server	15	13
SMTP User	16	14
SMTP Password	17	15
Word Order	18	22

E.4.4 Modbus Host Object (FAh)

Attribute	30-series	40-series	Change/Action/Comment
#2, Product Code	Default: “Anybus-CC Modbus-TCP (2-Port)”	Default: “Anybus CompactCom 40 Modbus TCP”	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.
#11,Modbus read/ write registers command offset	-	-	In the 30-series, this register address offset is only applied when accessing holding registers with the command Read/Write Multiple registers (23). The 40-series applies this register offset to all holding register access, i.e. commands 3, 6, 16 and 23.

E.4.5 Ethernet Host Object (F9h)

Attribute	30-series	40-series	Change/Action/Comment
#4, Enable Modbus-TCP	Available	Removed	Attribute removed in the 40-series. The CompactCom will never request this attribute. Nothing needs to be changed in the host application.

E.4.6 Process data

In the 30-series modules, writing to the ADI register area would only result in a Set_Attribute command to the application (Application Data Object (FEh)) if the ADI was not mapped to read process data. For the 40-series, all register writes to the ADI area also results in a corresponding Set_Attribute command to the host application (Application Data Object (FEh)), as well as updating of the process data.

F Copyright Notices

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

****

FatFs - FAT file system module R0.09b (C)ChaN, 2013

FatFs module is a generic FAT file system module for small embedded systems. This is a free software that opened for education, research and commercial developments under license policy of following trems.

Copyright (C) 2013, ChaN, all right reserved.

The FatFs module is a free software and there is NO WARRANTY. No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY. Redistributions of source code must retain the above copyright notice.

****

Copyright (c) 2002 Florian Schulze.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in the

documentation and/or other materials provided with the distribution.

3. Neither the name of the authors nor the names of the contributors

may be used to endorse or promote products derived from this software

without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

claim that you wrote the original software. If you use this software
in a product, an acknowledgment in the product documentation would be
appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.
L. Peter Deutsch
ghost@aladdin.com

This page intentionally left blank

Box 4126
300 04 Halmstad, Sweden

info@hms.se
SCM-1202-027 1.1.4726 / 2017-07-11T09:25:39