

User Manual

Anybus[®] Communicator CAN

Modbus-TCP

Doc.Id. HMSI-168-60
Rev. 1.20



HALMSTAD • CHICAGO • KARLSRUHE • TOKYO • BEIJING • MILANO • MULHOUSE • COVENTRY • PUNE • COPENHAGEN

HMS Industrial Networks
Mailing address: Box 4126, 300 04 Halmstad, Sweden
Visiting address: Stationsgatan 37, Halmstad, Sweden

E-mail: info@hms-networks.com
Web: www.anybus.com

Important User Information

This document is intended to provide a good understanding of the functionality offered by the Anybus Communicator CAN - Modbus-TCP.

The reader of this document is expected to be familiar with high level software design, and communication systems in general. The use of advanced Modbus-TCP specific functionality may require in-depth knowledge of 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 should either obtain the Modbus-TCP specification to gain sufficient knowledge or limit their implementation in such a way that this is not necessary. Also knowledge of CANopen (slave) is expected.

Liability

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

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

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

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

Intellectual Property Rights

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

The "Silk" icon set, used in the Anybus Configuration Manager tool, is created by Mark James, Birmingham, England. The complete icon set is found at <http://famfamfam.com/lab/icons/silk/>. The icon set is licensed under the Creative Commons Attribution 2.5 License (<http://creativecommons.org/licenses/by/2.5>).

Trademark Acknowledgements

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

Warning: This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

ESD Note: This product contains ESD (Electrostatic Discharge) sensitive parts that may be damaged if ESD control procedures are not followed. Static control precautions are required when handling the product. Failure to observe this may cause damage to the product.

Anybus Communicator CAN - Modbus-TCP User Manual

Rev 1.20

Copyright© HMS Industrial Networks AB

Mar 2013 Doc Id HMSI-168-60

Table of Contents

| | | |
|----------------------|--|----|
| Preface | About This Document | |
| | Related Documents | 1 |
| | Document History | 1 |
| | Conventions & Terminology | 2 |
| | Sales and Support | 3 |
| Chapter 1 | About the Anybus Communicator CAN | |
| | Introduction | 4 |
| | Anybus Communicator CAN Concept..... | 5 |
| | <i>General</i> | 5 |
| | <i>Data Exchange Model</i> | 6 |
| Chapter 2 | About the Module | |
| | External view..... | 7 |
| | Mounting..... | 8 |
| | Status LEDs | 9 |
| | Connectors | 10 |
| | <i>Ethernet Connectors</i> | 10 |
| | <i>USB Connector</i> | 10 |
| | <i>CAN Connector</i> | 10 |
| | Power Connector..... | 11 |
| | Software Installation | 12 |
| | <i>Anybus Configuration Manager</i> | 12 |
| | <i>Firmware Update</i> | 12 |

| | | |
|------------------|---|----|
| Chapter 3 | Getting Started | |
| Chapter 4 | CAN Network Communication | |
| | General..... | 14 |
| | Types of Messages..... | 14 |
| | <i>Query-Response</i> | 14 |
| | <i>Produce and Consume</i> | 15 |
| | Protocol Building Blocks..... | 15 |
| | Control/Status Word..... | 17 |
| | Transaction Live List..... | 18 |
| Chapter 5 | Data Representation on Modbus-TCP | |
| | General..... | 19 |
| | Data Representation..... | 19 |
| | <i>Memory Layout (Internal Memory Buffer)</i> | 20 |
| | Modbus-TCP..... | 21 |
| | <i>General</i> | 21 |
| | <i>Addressing Modes</i> | 21 |
| | <i>Supported Exception Codes</i> | 21 |
| | <i>Modbus Addressing Mode</i> | 22 |
| | <i>Anybus Addressing Mode</i> | 23 |
| Chapter 6 | Configuration | |
| | Configuring the Anybus Communicator CAN..... | 24 |
| | Configuring the Modbus-TCP Network..... | 24 |
| Chapter 7 | Anybus Configuration Manager | |
| | Main Window..... | 25 |
| | <i>Pull-down Menus</i> | 26 |
| Chapter 8 | Basic Settings | |
| | Project..... | 29 |
| | Network Settings..... | 29 |
| | <i>TCP/IP Settings</i> | 29 |
| | <i>Modbus Settings</i> | 30 |
| | Communicator Settings..... | 31 |
| | Subnetwork Settings..... | 32 |
| Chapter 9 | Groups and Transactions | |
| | General..... | 33 |
| | Groups..... | 33 |
| | Transactions..... | 33 |
| | <i>Produce</i> | 34 |
| | <i>Consume</i> | 35 |
| | <i>Query/Response</i> | 36 |

| | |
|--|----|
| <i>Dynamic Produce</i> | 37 |
| <i>Dynamic Consume</i> | 38 |
| Chapter 10 Configuration of CAN Frames | |
| General | 39 |
| <i>CAN Identifiers</i> | 39 |
| Produce/Query CAN Frame | 40 |
| Consume/Response CAN Frame | 40 |
| CAN Frames in Dynamic Transactions | 41 |
| Chapter 11 Online | |
| Select Connection | 42 |
| Connect/Disconnect | 43 |
| Download and Upload Configuration | 43 |
| Chapter 12 Anybus Configuration Manager Tools | |
| Monitor/Modify | 44 |
| CAN Line Listener | 45 |
| Address Overview | 46 |
| Diagnostics/Status | 47 |
| Reassign Addresses | 48 |
| Project Summary | 49 |
| Password | 50 |
| Options | 51 |

Appendix A Technical Specification

| | |
|--|----|
| Protective Earth (PE) Requirements | 52 |
| Power Supply | 52 |
| Environmental Specification | 52 |
| <i>Temperature</i> | 52 |
| <i>Relative Humidity</i> | 52 |
| EMC (CE) Compliance | 53 |

Appendix B Configuration Example

Appendix C Advanced IT Functionality

| | |
|---|----|
| File System | 58 |
| <i>General</i> | 58 |
| <i>File System Overview</i> | 59 |
| <i>System Files</i> | 59 |
| Basic Network Configuration | 60 |
| <i>General Information</i> | 60 |
| <i>Ethernet Configuration File ('ethcfg.cfg')</i> | 61 |
| <i>IP Access Control</i> | 62 |
| <i>Anybus IPconfig (HICP)</i> | 63 |
| FTP Server | 64 |
| <i>General</i> | 64 |
| <i>FTP Connection Example (Windows Explorer)</i> | 64 |
| Telnet Server | 66 |
| <i>General</i> | 66 |
| <i>General Commands</i> | 66 |
| <i>Diagnostic Commands</i> | 67 |
| <i>File System Operations</i> | 68 |
| Web Server | 71 |
| <i>General</i> | 71 |
| <i>Authorization</i> | 72 |
| <i>Content Types</i> | 72 |
| Server Side Include (SSI) | 74 |
| <i>General</i> | 74 |
| <i>Functions</i> | 75 |
| <i>Changing SSI output</i> | 81 |
| E-mail Client | 84 |
| <i>General</i> | 84 |
| <i>E-mail Definitions</i> | 84 |

Appendix D Copyright Notices

P. About This Document

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

P.1 Related Documents

| Document | Author |
|-------------------------------|----------------------|
| CAN protocol specification | www.can-cia.org |
| Open Modbus-TCP Specification | Schneider Automation |
| | |
| | |
| | |

P.2 Document History

Summary of Recent Changes (1.10... 1.20)

| Change | Page(s) |
|--|---------|
| Corrected description of Module Status LED | 9 |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Revision List

| Revision | Date | Author(s) | Chapter(s) | Description |
|----------|------------|-----------|---------------------------|-------------------------------|
| 1.00 | 2011-09-22 | KeL | - | First official release |
| 1.01 | 2012-02-13 | KeL | 12 | Minor updates |
| 1.10 | 2012-09-10 | KaD, KeL | 2, 4, 7, 8, 9, 10, 11, 12 | Service pack 1 updates |
| 1.20 | 2013-03-14 | KeL | 2 | Minor corrections and updates |
| | | | | |
| | | | | |
| | | | | |

P.3 Conventions & Terminology

The following conventions are used throughout this manual:

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

P.4 Sales and Support

| Sales | | Support | |
|---------------------------------|---------------------------|------------|-----------------------------|
| HMS Sweden (Head Office) | | | |
| E-mail: | sales@hms-networks.com | E-mail: | support@hms-networks.com |
| Phone: | +46 (0) 35 - 17 29 56 | Phone: | +46 (0) 35 - 17 29 20 |
| Fax: | +46 (0) 35 - 17 29 09 | Fax: | +46 (0) 35 - 17 29 09 |
| Online: | www.anybus.com | Online: | www.anybus.com |
| HMS North America | | | |
| E-mail: | us-sales@hms-networks.com | E-mail: | us-support@hms-networks.com |
| Phone: | +1-312 - 829 - 0601 | Phone: | +1-312-829-0601 |
| Toll Free: | +1-888-8-Anybus | Toll Free: | +1-888-8-Anybus |
| Fax: | +1-312-629-2869 | Fax: | +1-312-629-2869 |
| Online: | www.anybus.com | Online: | www.anybus.com |
| HMS Germany | | | |
| E-mail: | ge-sales@hms-networks.com | E-mail: | ge-support@hms-networks.com |
| Phone: | +49 (0) 721-989777-000 | Phone: | +49 (0) 721-989777-300 |
| Fax: | +49 (0) 721-989777-010 | Fax: | +49 (0) 721-989777-010 |
| Online: | www.anybus.de | Online: | www.anybus.de |
| HMS Japan | | | |
| E-mail: | jp-sales@hms-networks.com | E-mail: | jp-support@hms-networks.com |
| Phone: | +81 (0) 45-478-5340 | Phone: | +81 (0) 45-478-5340 |
| Fax: | +81 (0) 45-476-0315 | Fax: | +81 (0) 45-476-0315 |
| Online: | www.anybus.jp | Online: | www.anybus.jp |
| HMS China | | | |
| E-mail: | cn-sales@hms-networks.com | E-mail: | cn-support@hms-networks.com |
| Phone: | +86 (0) 10-8532-3183 | Phone: | +86 (0) 10-8532-3023 |
| Fax: | +86 (0) 10-8532-3209 | Fax: | +86 (0) 10-8532-3209 |
| Online: | www.anybus.cn | Online: | www.anybus.cn |
| HMS Italy | | | |
| E-mail: | it-sales@hms-networks.com | E-mail: | it-support@hms-networks.com |
| Phone: | +39 039 59662 27 | Phone: | +39 039 59662 27 |
| Fax: | +39 039 59662 31 | Fax: | +39 039 59662 31 |
| Online: | www.anybus.it | Online: | www.anybus.it |
| HMS France | | | |
| E-mail: | fr-sales@hms-networks.com | E-mail: | fr-support@hms-networks.com |
| Phone: | +33 (0) 3 68 368 034 | Phone: | +33 (0) 3 68 368 033 |
| Fax: | +33 (0) 3 68 368 031 | Fax: | +33 (0) 3 68 368 031 |
| Online: | www.anybus.fr | Online: | www.anybus.fr |
| HMS UK & Eire | | | |
| E-mail: | uk-sales@hms-networks.com | E-mail: | support@hms-networks.com |
| Phone: | +44 (0) 1926 405599 | Phone: | +46 (0) 35 - 17 29 20 |
| Fax: | +44 (0) 1926 405522 | Fax: | +46 (0) 35 - 17 29 09 |
| Online: | www.anybus.co.uk | Online: | www.anybus.com |
| HMS Denmark | | | |
| E-mail: | dk-sales@hms-networks.com | E-mail: | support@hms-networks.com |
| Phone: | +45 (0) 35 38 29 00 | Phone: | +46 (0) 35 - 17 29 20 |
| Fax: | +46 (0) 35 17 29 09 | Fax: | +46 (0) 35 - 17 29 09 |
| Online: | www.anybus.com | Online: | www.anybus.com |
| HMS India | | | |
| E-mail: | in-sales@hms-networks.com | E-mail: | in-support@hms-networks.com |
| Phone: | +91 (0) 20 40111201 | Phone: | +91 (0) 20 40111201 |
| Fax: | +91 (0) 20 40111105 | Fax: | +91 (0) 20 40111105 |
| Online: | www.anybus.com | Online: | www.anybus.com |

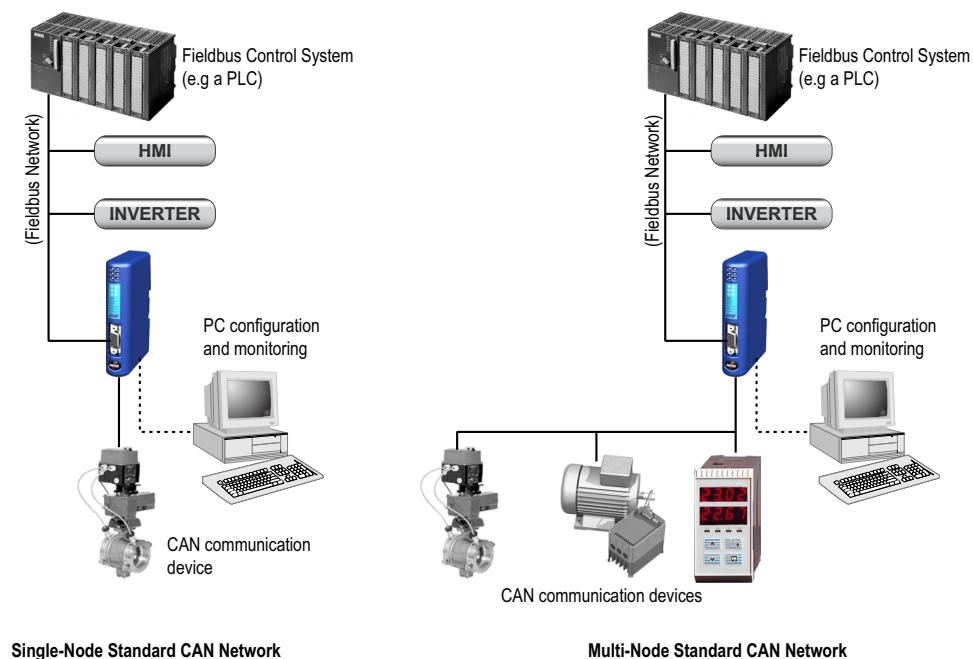
1. About the Anybus Communicator CAN

1.1 Introduction

The Anybus Communicator CAN is a series of products that acts as a gateway between a subnetwork, running the standard CAN protocol, and a number of popular industrial networks. Integration of industrial devices is enabled without loss of functionality, control and reliability, both when retro-fitting to existing equipment as well as when setting up new installations.

The Anybus Communicator CAN is based on patented Anybus technology, a proven industrial communication solution used all over the world by leading manufacturers of industrial automation products. Each module offers integration of industrial CAN devices to one of these industrial networks: EtherCAT, PROFIBUS, ControlNet, Modbus-RTU, Modbus-TCP, PROFINET, PROFINET IRT, EtherNet/IP, DeviceNet CC-Link and CANopen. The scope of this manual is the Anybus Communicator CAN for Modbus-TCP. The manual primarily describes the functionality and the configuration of the CAN network and the connection between the CAN network and the Modbus-TCP network. Relevant information on the Modbus-TCP interface of the module is given, with the intention to facilitate the configuration of the interface into a Modbus-TCP network. For more information about Modbus-TCP, please refer to official specifications.

No proprietary configuration software is needed. All necessary configuration is performed using the Anybus Configuration Manager that accompanies the product.



Subnetwork

The Anybus Communicator CAN recognizes and supports communication that conforms to the CAN standards 2.0A and 2.0B. The Communicator can adapt to any predefined network using CAN frames as means for data exchange, using the Anybus Configuration Manager tool, that is included with the product.

- 0 - 8 bytes of data in each frame
- 11-bit (CAN 2.0A) identifier or 29-bit (CAN 2.0B) identifier
- Bit rates supported: 20, 50, 100, 125, 200, 250, 500, 800 and 1000 kbit/s.

Modbus-TCP Interface

Modbus-TCP connectivity is provided through patented Anybus technology:

- 10 and 100 Mbit operation, full or half duplex
- Flexible file system providing both volatile and non-volatile storage areas
- Security framework
- IP Access Control
- DHCP/HICP support

1.2 Anybus Communicator CAN Concept

1.2.1 General

The Anybus Communicator is designed to exchange data between a subnetwork, running CAN, and a higher level network. The CAN protocol uses frames, that are individually configurable, offering great flexibility.

Through the configuration of the CAN frames, the Communicator will adapt to a predefined CAN network. It will be possible to send data to and receive data from the subnetwork, but also to act as a relay for data on the CAN subnetwork.

The Communicator can issue frames cyclically, on change of data, or based on trigger events issued by the control system of the higher level network (i.e. the fieldbus master or PLC) or by the CAN network. It can also monitor certain aspects of the subnetwork communication and notify the higher level network when data has changed.

An essential part of the Anybus Communicator package is the Anybus Configuration Manager, a Windows™ application which is used to supply the Communicator with a description of the subnetwork protocol. No programming skills are required; instead, a visual protocol description-system is used to specify the different parts of the CAN frames.

1.2.2 Data Exchange Model

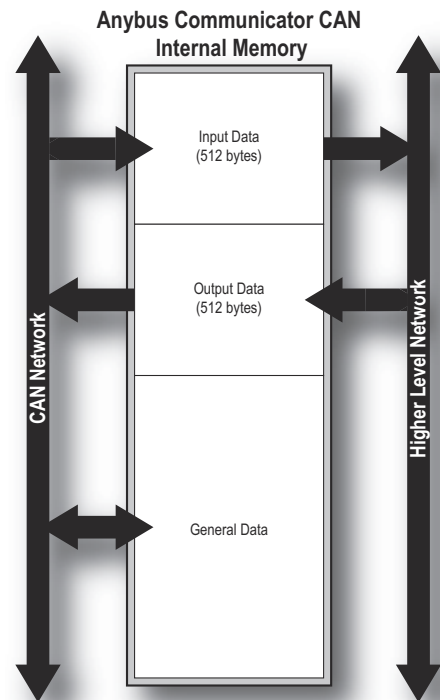
Internally, the data exchanged on the subnetwork, and the data exchanged on the higher level network, resides in the same memory.

This means that in order to exchange data with the sub-network, the higher level network simply reads and writes data to memory locations specified using the Anybus Configuration Manager. The very same memory locations can then be exchanged on the subnetwork.

The internal memory buffer is divided into three areas based on their function:

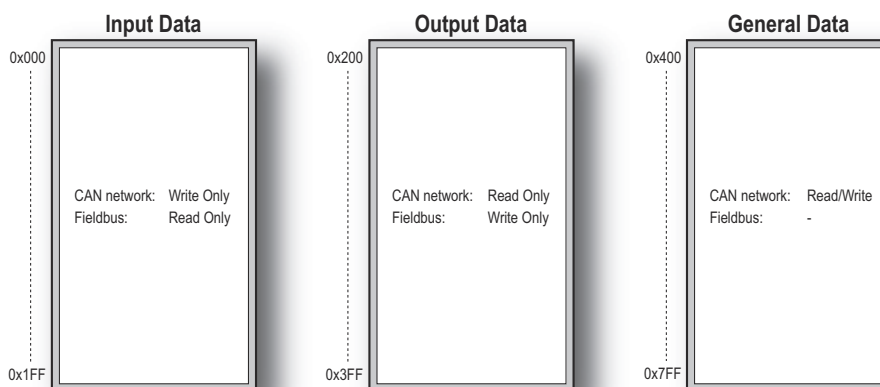
- **Input Data (Up to 512 bytes)**
This area can be read by the higher level network.
- **Output Data (Up to 512 bytes)**
This area can be written to by the higher level network.

- **General Data**
This area can not be accessed from the higher level network, but may be used for transfers between individual nodes on the subnetwork, or as a general “scratch pad” for data. The size of the General Data area is 1024 bytes. How much data of the area that is used for subnetwork communication is decided by the configuration.



Memory Map

When building the subnetwork configuration using the Anybus Configuration Manager, the different areas described above are mapped to the memory locations (addresses) specified below.



2. About the Module

2.1 External view

A: Status LEDs

See also...

- “Status LEDs” on page 9

B: Fieldbus Specific Connectors

These connectors are used to connect the Anybus Communicator CAN module to the Modbus-TCP network. They are described in “Ethernet Connectors” on page 10.

C: USB connector

This connector is used for uploading and downloading the configuration and for software upgrade of the module.

See also...

- “USB Connector” on page 10

D: CAN Connector

This connector is used to connect the communicator to the CAN network.

See also...

- “CAN Connector” on page 10

E: Power Connector

This connector is used to apply power to the communicator.

See also...

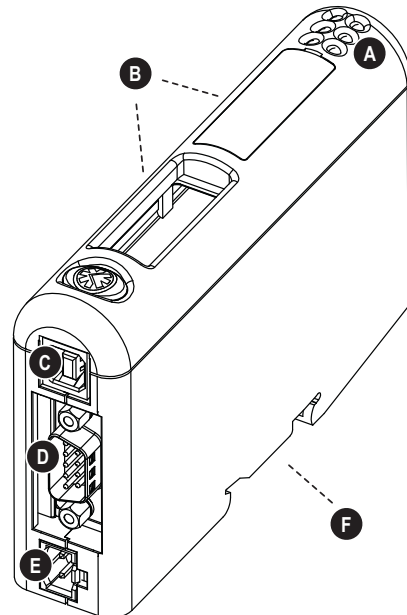
- “Power Connector” on page 11

F: DIN-rail Connector

The DIN-rail mechanism connects the communicator to PE (Protective Earth).

See also...

- “Mounting” on page 8

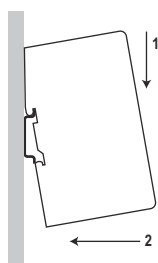


2.2 Mounting

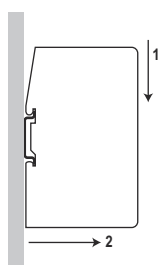
Perform the following steps when physically installing the Communicator:

1. Snap the Communicator on to the DIN-rail (See “External view” on page 7).

The DIN-rail mechanism works as follows:



To snap the Communicator *on*, first press the it downwards (1) to compress the spring in the DIN-rail mechanism, then push it against the DIN-rail as to make it snap on (2).



To snap the Communicator *off*, push the it downwards (1) and pull it out from the DIN-rail (2), as to make it snap off from the DIN-rail.

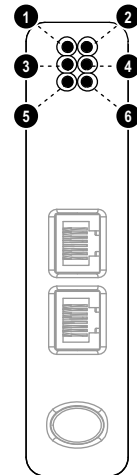
2. Connect the Communicator to the CAN network.
3. Connect the Communicator to the Modbus-TCP network.
4. Connect the power cable and apply power.

2.3 Status LEDs

The status LEDs on the front indicate the status of the module as shown in the table below.

Status LEDs 1 - 4 indicate the status of the Modbus-TCP network and status LEDs 5 - 6 indicate the status of the CAN subnetwork and the device.

| # | State | Status |
|---------------------------|-----------------------|--|
| 1 - Module Status | Off | No power or not initialized |
| | Flashing green (1 Hz) | Initialized, normal operation |
| | Flashing red (1 Hz) | Invalid MAC address (internal error) |
| | Flashing red (2 Hz) | Failed to load Ethernet configuration from FLASH |
| | Flashing red (4 Hz) | Internal error (fatal) |
| | Red | Duplicate IP address detected |
| 2 - Network status | Flashing green | Indicates the number of Modbus-TCP connections (each connection is represented by a single green flash) |
| 3 - Link activity 1 | Off | No link sensed on port 1/2 |
| 4 - Link activity 2 | Flashing green | Activity, receiving/transmitting Ethernet Packets at 100 Mbit |
| | Flashing yellow | Activity, receiving/transmitting Ethernet Packets at 10 Mbit |
| 5 - CAN subnetwork status | Off | Power off/no CAN communication |
| | Green | Running with no transaction error/timeout |
| | Flashing green | Not all transactions have been executed at least once since startup and no transaction error/timeout has occurred. |
| | Flashing red | Transaction error/timeout or subnetwork stopped |
| | Red | Fatal error |
| 6 - Device status | Off | Power off |
| | Alternating red/green | Invalid or missing configuration |
| | Green | Operation mode Run |
| | Flashing green | Operation mode Idle |
| | Red | Fatal error |

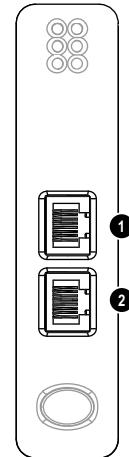
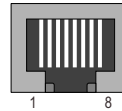


2.4 Connectors

2.4.1 Ethernet Connectors

Modbus-TCP Ports 1 and 2

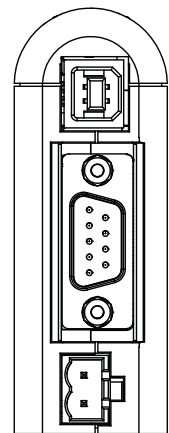
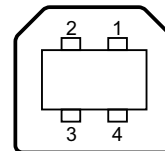
| Pin no | Description |
|------------|-----------------|
| 1 | TD+ |
| 2 | TD- |
| 3 | RD+ |
| 4, 5, 7, 8 | (not connected) |
| 6 | RD- |
| Housing | Cable Shield |



2.4.2 USB Connector

At the bottom of the module you find a USB connector used for software upgrade of the module and for uploading and downloading configurations.

| Pin no. | Description |
|---------|-----------------------------------|
| 1 | +5 V input |
| 2 | USBDM (USB communication signals) |
| 3 | USBDP (USB communication signals) |
| 4 | Signal GND |
| Housing | Cable Shield |

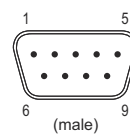


Note: USB is used for configuration and software upgrade only. Remove the USB cable when the configuration of the module is finished.

2.4.3 CAN Connector

Next to the USB connector the CAN connector is found.

| Pin no. | Description |
|------------|---------------------------|
| 2 | CAN_L |
| 5 | Housing, CAN cable shield |
| 7 | CAN_H |
| 1, 4, 8, 9 | (not connected) |
| 3, 6 | CAN GND |



2.5 Power Connector

| Pin no. | Description |
|---------|-------------|
| 1 | +24V DC |
| 2 | GND |



Notes:

- Use 60/75 or 75×C copper (CU) wire only.
- The terminal tightening torque must be between 5... 7 lbs-in (0.5... 0.8 Nm)

See also...

- “Power Supply” on page 52

2.6 Software Installation

2.6.1 Anybus Configuration Manager

System Requirements

- Pentium 233 MHz or higher (300 MHz recommended)
- 64 MB RAM or more (128 MB recommended)
- Microsoft Windows XP, Windows Vista, or Windows 7

Installation

- **Anybus Communicator CAN resource CD**

Insert the CD and follow the onscreen instructions. If the installation does not start automatically right-click on the CD-drive icon and select Explore. Execute 'setup.exe' and follow the onscreen instructions.

- **From website**

Download and execute the self-extracting .exe file from the HMS website (www.anybus.com).

2.6.2 Firmware Update

Updates of the Communicator firmware will be published on the support pages at www.anybus.com. Also available is the tool Firmware Download TP, that is used to download the updated firmware to the Communicator.

Note: Before downloading the new firmware, save a copy of the configuration, as the configuration in the module will be erased during the installation process of the new firmware. When download of the firmware is finished, the configuration can be restored from the safety copy.

3. Getting Started

The purpose of this chapter is to give a short description of how to install the module and get it up and running, transferring I/O data between the CAN network and the Modbus-TCP network. Before starting, make sure that you have access to knowledge about the CAN protocol to be configured, e.g. access to the CAN protocol specification.

Perform the following steps when installing the Communicator:

1. Download the Anybus Configuration Manager from the product pages at www.anybus.com or copy it from the CD that accompanies the product. Install it on your PC.
2. Build your configuration in the Anybus Configuration Manager tool, for an example see “Configuration Example” on page 54, for a description of the tool see chapters 7 to 12.
3. Connect the Communicator to your PC using the USB connector.
4. Connect the power cable and apply power.
5. Download the configuration from the Anybus Configuration Manager to the Communicator. See “Online” on page 42.
6. Remove the USB cable, turn off the power and disconnect the power cable.
7. Snap the Communicator on to the DIN-rail (See “Mounting” on page 8).
8. Connect the Communicator to the CAN network with proper termination and shielding.
9. If necessary, configure the other nodes in the CAN network.
10. Connect the Communicator to the Modbus-TCP network.
11. Connect the power cable and apply power.
12. Configure the Modbus-TCP network. Please adapt the configuration to the one stored in the Communicator.

4. CAN Network Communication

4.1 General

The CAN protocol is message-based and offers the possibility to exchange up to 8 bytes of data in each message. How these bytes are interpreted, is defined in each application. The CAN protocol is a transparent protocol, meaning that it only acts as a data carrier, and it is up to the users (the application) to define and interpret the data content of the messages.

Data on CAN is exchanged using frames. Each frame has a unique identifier for the data it exchanges. The identifier also represents the message priority on the CAN network. The Anybus Communicator CAN supports either 11-bit (CAN 2.0A) or 29-bit (CAN 2.0B) identifiers, depending on what is defined during configuration.

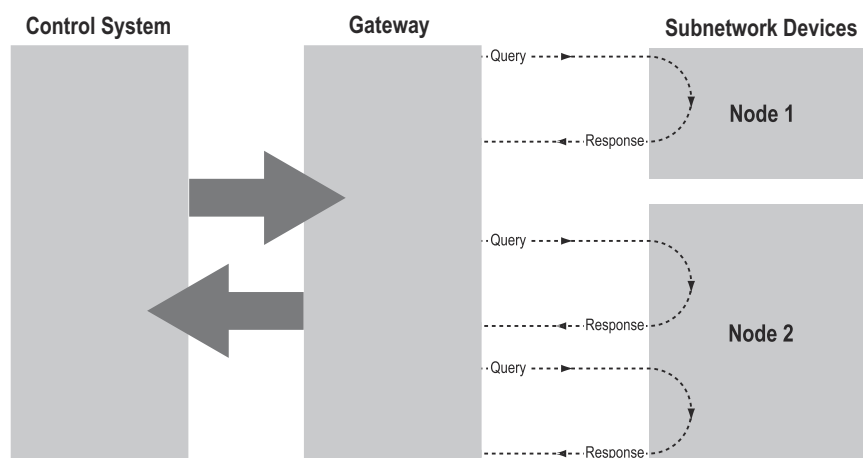
CAN is essentially a produce-consume network, where all nodes listen to all messages. The devices recognize what data to collect by what identifier the CAN frame carries. The Communicator is also able to act as a Master and issue queries that demand responses. It is possible to use both methods in the same configuration of the module.

4.2 Types of Messages

The Anybus Communicator CAN features three different message types regarding the subnetwork communication, called 'Query/Response', 'Produce' and 'Consume'. Note that these messages only specify the basic communication model, not the actual CAN protocol. All three types of messages can be used in the same configuration.

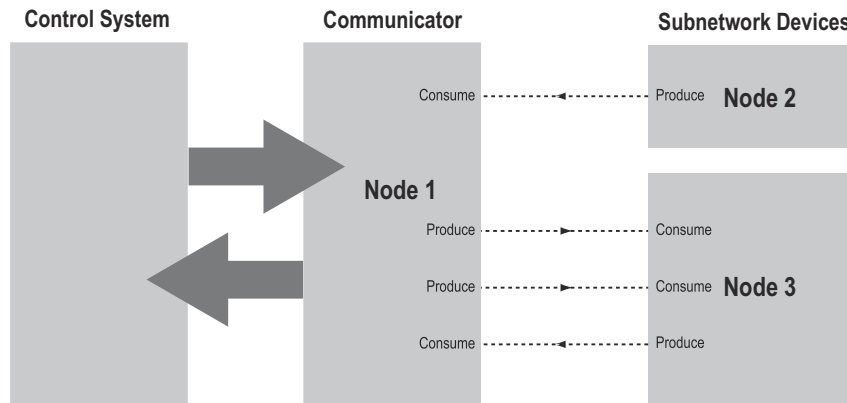
4.2.1 Query-Response

The Communicator acts as a master on the subnetwork, and the CAN communication takes place in a query-response fashion. The Communicator sends a query and expects an answer within the specified timeout.



4.2.2 Produce and Consume

When using these messages, there is no master-slave relationship between the Communicator and the nodes on the subnetwork. Any node, including the Communicator, may spontaneously produce a message. The message is sent on the network. The nodes on the network listen to all traffic and decide independently which messages to consume (read). Nodes do not have to respond to messages, nor do they have to wait for a query to send a message on the network.



In the figure above, the Communicator ‘consumes’ data that is ‘produced’ by a node on the subnetwork. This ‘consumed’ data can then be accessed from the higher level network. This also works the other way around; the data received from the higher level network is used to ‘produce’ a message on the subnetwork to be ‘consumed’ by a node.

Note: When configuring the Communicator using the Anybus Configuration Manager, ‘produce’ and ‘consume’ are defined from the Communicator’s perspective.

4.3 Protocol Building Blocks

The following building blocks are used in Anybus Configuration Manager to describe the subnetwork communication. How these blocks apply to the two modes of operation will be described later in this document.

- **Group**

A group in the Anybus Configuration Manager does not represent any special device on the CAN network. It is a means to structure the transactions that are defined for the Communicator. Each group can be associated with a number of transactions, see below.

- **Transaction**

A transaction consists of one or more CAN frames. Each transaction is associated with a set of parameters controlling how and when to use it on the subnetwork. There are five kinds of transactions: produce, consume, query-response, dynamic produce and dynamic consume. A group can contain transactions of all three types simultaneously. A total of 128 transactions can be configured.

- **Dynamic Transaction**

In normal transactions, all parameters are changed using the Anybus Configuration Manager. A dynamic transaction makes it possible for a network master to change selected parameters during runtime. The parameters are mapped to the output data area or to the general data area and it will not be possible to change them using the Anybus Configuration Manager. A dynamic trans-

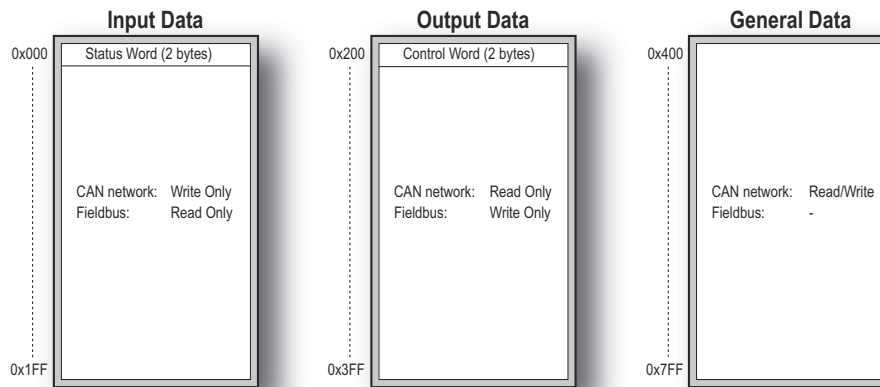
action may only consist of one CAN frame that can hold up to one data object. Also only one produce and one consume dynamic transaction are allowed.

- **CAN Frames**

The CAN frames are low level entities used to compose transactions (see above). Each frame carries an 11-bit or 29-bit identifier and can hold up to 8 bytes of data. See “Configuration of CAN Frames” on page 39. A total of 256 CAN frames can be configured.

4.4 Control/Status Word

An optional control/status word can be used to control the startup mode of the module and to read the status of the CAN network. The control word is always mapped to the first two bytes of the output data area, and the status word is mapped to the first two bytes of the input data area. It is not possible to change these locations.



Note 1: The picture shows the maximum available data areas in the Communicator. Not all fieldbuses can access all addresses in the input and output data areas, please see section Data Exchange Model in chapter 1.

Note 2: The control/status words are stored in the first two bytes of the data areas, with the least significant byte (bit 0-7) in the first byte (byte #0).

Through the control word it is possible to reset the CAN controller, reboot the module and decide the start-up mode of the Communicator:

| Bit | Name | Description |
|--------|----------------|--|
| 15 - 3 | (Reserved) | |
| 2 | Reset CAN | A transition from 0 to 1 resets the CAN controller (used when the CAN interface is bus off). |
| 1 | Reboot module | A transition from 0 to 1 reboots the Communicator (software reset) |
| 0 | Operation mode | This bit sets the start-up operation mode of the Communicator: 0 - Idle (No new data issued to the CAN network. Data received from the CAN network is sent on the Modbus-TCP network.) 1 - Run (Data is exchanged between the CAN network and Modbus-TCP.) |

The status word holds status information from the CAN network:

| Bit | Name | Description |
|--------|--------------------|--|
| 15 - 6 | (Reserved) | |
| 5 | CAN overrun | 0 - OK 1 - CAN reception overrun |
| 4 | Error passive | 0 - The CAN interface is NOT in error passive state 1 - The CAN interface is in error passive state |
| 3 | Bus off | 0 - Bus running 1 - Bus off |
| 2 | Reset CAN complete | If set, the CAN controller has been reset (used when the CAN interface is bus off). |
| 1 | (Reserved) | |
| 0 | Operation mode | 0 - Idle 1 - Run |

4.5 Transaction Live List

An optional transaction live list is available. It consists of a bit array where each bit corresponds to a transaction on the CAN subnetwork. (bit 0 corresponds to transaction 1 etc.). A set bit indicates normal functionality. The bit is not set if the transaction is non-working or non-existent. The live list is mapped in the input data area of the memory, either at the start of the area or directly after the status word. From 8 transactions up to 128 transactions in steps of 8 can be monitored using the live list. Thus, up to 16 bytes of the input data area of the memory can be occupied by the live list.

The latest live list is always available from the Anybus Configuration Managers Diagnostics/Status window, whether the live list is mapped in the input data area or not, see “Diagnostics/Status” on page 47.

5. Data Representation on Modbus-TCP

5.1 General

The Anybus Communicator CAN acts as a Modbus-TCP slave on the Modbus-TCP network. As such, it does not initiate communication towards other nodes by itself, but can be read from/written to by a Modbus-TCP master.

The Modbus-TCP protocol is an implementation of the standard Modbus protocol running on top of TCP/IP. The built in Modbus /TCP server provides access to the Input- and Output Data areas via a subset of the functions defined in the Modbus-TCP specification.

All Modbus-TCP messages are exchanged through TCP port no. 502 and the Modbus-TCP server can handle a maximum of 8 simultaneous connections. For detailed information regarding the Modbus-TCP protocol, consult the Open Modbus Specification.

5.2 Data Representation

The Input and Output Data areas in the internal memory buffer are used for Modbus-TCP data. The amount of data exchanged on the bus depends on the configuration of the Communicator. The data sizes can be viewed at any time, see “Address Overview” on page 46.

5.2.1 Memory Layout (Internal Memory Buffer)

The data in the input and output data areas is represented as continuous blocks of I/O data. Each data object in a CAN frame adds to the total amount. Usage of the memory can always be seen in the Address Overview, see page 46.

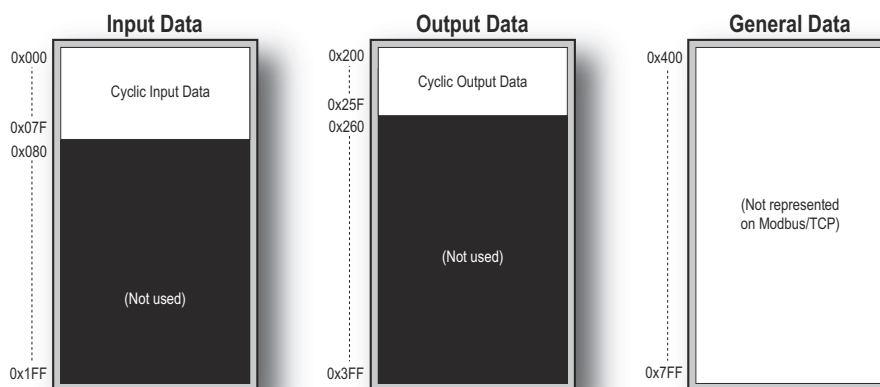
Example:

In this example, the I/O sizes for the Communicator has been set to the following values:

IO Size In = 128 bytes (0x0080)

IO Size Out = 96 bytes (0x0060)

Resulting memory layout:



Please note that the two first bytes of the Output and Input Data areas are occupied by the Control/Status Word if this is enabled. Live List and Transmit and Receive Counters are by default mapped to the input memory area from the start address or after the Status Word.

5.3 Modbus-TCP

5.3.1 General

The Modbus-TCP protocol is an implementation of the standard Modbus protocol running on top of TCP/IP. The built-in Modbus-TCP server provides access to the input and output data areas via a subset of the functions defined in the Modbus-TCP specification.

The server supports up to 8 simultaneous connections and communicates over TCP port 502. For detailed information regarding the Modbus-TCP protocol, consult the Open Modbus Specification.

5.3.2 Addressing Modes

The Anybus Communicator features two different modes of operation regarding the Modbus communication:

- **Modbus Addressing Mode**

In this mode, the input and output data areas are mapped to different function codes. The Modbus addressing mode is disabled/enabled in the Anybus Configuration Manager.

Note that coil addressing is not possible in this mode.

See also...

- “Modbus Addressing Mode” on page 22
- “Settings specified in Anybus Configuration Manager” on page 30

- **Anybus Addressing Mode (default)**

Compared to Modbus Addressing Mode, this mode allows data to be addressed in a more flexible way. Note however that several function codes can be used to access the same data in the Communicator. While this may appear confusing at first, it allows data to be manipulated in ways not possible in Modbus Addressing Mode (e.g. it is possible to manipulate individual bits of a register by accessing coils associated with the same memory location).

See also...

- “Anybus Addressing Mode” on page 23

5.3.3 Supported Exception Codes

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

5.3.4 Modbus Addressing Mode

Supported Function Codes

The following function codes can be used in this mode:

| Modbus Function | Function Code | Associated with Area |
|--------------------------|---------------|----------------------------------|
| Read Holding Registers | 3 | Output Data area (0x200...0x3FF) |
| Read Input Registers | 4 | Input Data area (0x000...0x1FF) |
| Write Single Register | 6 | Output Data area (0x200...0x3FF) |
| Force Multiple Registers | 16 | Output Data area (0x200...0x3FF) |
| Mask Write Register | 22 | Output Data area (0x200...0x3FF) |
| Read/Write Registers | 23 | Output Data area (0x200...0x3FF) |

Input Register Map

The input data area is mapped to input registers as follows:

| Register # | Memory Location in the Communicator | Comments |
|------------|-------------------------------------|--|
| 1 | 0x000... 0x001 | Each register corresponds to two bytes in the input data area. |
| 2 | 0x002... 0x003 | |
| 3 | 0x004... 0x005 | |
| 4 | 0x006... 0x007 | |
| 5 | 0x008... 0x009 | |
| 6 | 0x00A... 0x00B | |
| ... | ... | |
| 255 | 0x1FC... 0x1FD | |
| 256 | 0x1FE... 0x1FF | |

Holding Register Map

The output data area is mapped to holding registers as follows:

| Register # | Memory Location in the Communicator | Comments |
|------------|-------------------------------------|---|
| 1 | 0x200... 0x201 | Each register corresponds to two bytes in the output data area. |
| 2 | 0x202... 0x203 | |
| 3 | 0x204... 0x205 | |
| 4 | 0x206... 0x207 | |
| 5 | 0x208... 0x209 | |
| 6 | 0x20A... 0x20B | |
| ... | ... | |
| 255 | 0x3FC... 0x3FD | |
| 256 | 0x3FE... 0x3FF | |

5.3.5 Anybus Addressing Mode

Supported Function Codes

The following function codes can be used in this mode:

| Modbus Function | Function Code | Associated with Area(s) | No. of I/Os or data points per command |
|--------------------------|---------------|---|--|
| Read Coil | 1 | Input and Output Data Area (0x000... 0x3FF) | 1 - 2000 bits |
| Read Input Discretes | 2 | | 1 - 2000 bits |
| Read Holding Registers | 3 | | 1 - 125 registers |
| Read Input Registers | 4 | | 1 - 125 registers |
| Write Coil | 5 | Output Data Area (0x200... 0x3FF) | 1 bit |
| Write Single Register | 6 | | 1 register |
| Force Multiple Coils | 15 | | 1 - 800 bits |
| Force Multiple Registers | 16 | | 1 - 800 registers |
| Mask Write Register | 22 | | 1 register |
| Read/Write Registers | 23 | Input and Output Data Area (0x000... 0x3FF) | 125 registers read/100 registers write |

Coil & Register Map

The input and output data areas are mapped to coils and registers as follows:

| Register # | Coil # | Memory Location in ABC | Area | Comments |
|------------|----------------|------------------------|------------------|----------|
| 1 | 1... 16 | 0x000... 0x001 | Input Data area | - |
| 2 | 17... 32 | 0x002... 0x003 | | |
| 3 | 33... 48 | 0x004... 0x005 | | |
| 4 | 49... 64 | 0x006... 0x007 | | |
| ... | ... | ... | | |
| 255 | 4065... 4080 | 0x1FC... 0x1FD | | |
| 256 | 4081... 4096 | 0x1FE... 0x1FF | Output Data area | - |
| 257 | 4097... 4112 | - | | |
| ... | ... | ... | | |
| 1024 | 16369... 16384 | - | | |
| 1025 | 16385... 16400 | 0x200... 0x201 | | |
| 1026 | 16401... 16416 | 0x202... 0x203 | (reserved) | |
| 1027 | 16417... 16432 | 0x204... 0x205 | | |
| 1028 | 16433... 16448 | 0x206... 0x207 | | |
| ... | ... | ... | | |
| 1279 | 20449... 20464 | 0x3FC... 0x3FD | | |
| 1280 | 20465... 20480 | 0x3FE... 0x3FF | | |

Note 1: The table above applies to all function codes.

Note 2: Coils are mapped MSB first, i.e. coil 0 corresponds to bit 15 of register 0.

6. Configuration

6.1 Configuring the Anybus Communicator CAN

The configuration of the Anybus Communicator CAN is performed using the configuration tool Anybus Configuration Manager for Communicator CAN (ACM). The tool is included on the CD that accompanies the module, and it is also available for download at www.anybus.com. Chapters 7 to 12 in this manual describe the configuration tool and its features. A configuration example is given in Appendix B on page 54.

The USB connector at the bottom of the module is used for uploading and downloading the configuration. Please remove the USB cable when the configuration of the Communicator is finished.

6.2 Configuring the Modbus-TCP Network

The Anybus Communicator CAN - Modbus-TCP is a Modbus-TCP slave on the Modbus-TCP network. The general settings for the adapter interface are configured using the ACM (see “Network Settings” on page 28). Please note that the size of the I/O data that can be read from and written to the module is defined when configuring the Communicator using the ACM tool.

There are a number of different configuration tools for Modbus-TCP available on the market. The choice of tool depends on the application and the Modbus-TCP master of the network.

An application note, describing how to configure an Anybus Modbus-TCP slave interface, is available on the support pages for the Anybus Communicator CAN - Modbus-TCP module at www.anybus.com.

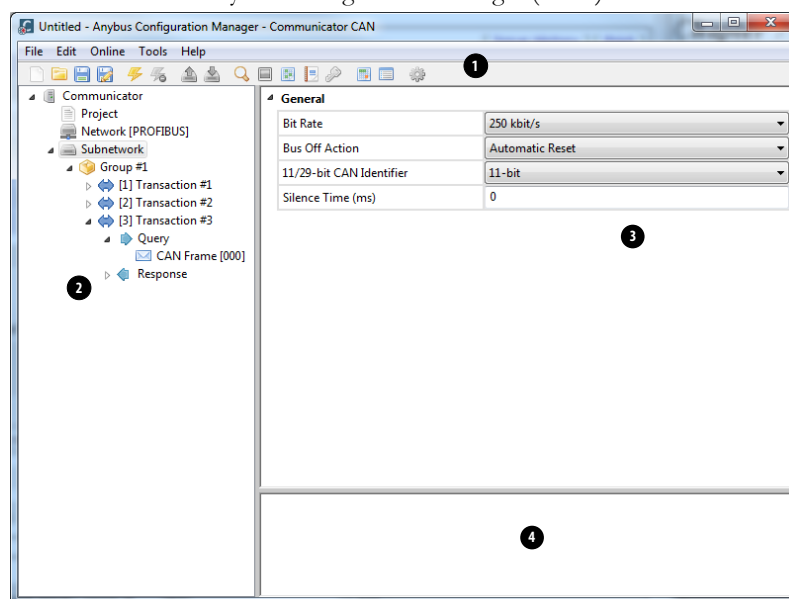
7. Anybus Configuration Manager

The Anybus Configuration Manager (ACM) is used to configure all aspects of the Communicator CAN. It also provides different tools for monitoring the module and the CAN subnetwork.

Note: The configuration manager automatically allocates addresses and memory space in the input and output areas of the Communicator for the data objects that are configured. It is possible to change these addresses, but it is recommended to finish the configuration using default addresses before starting to change any addresses. A valid address range is always shown in the information section of the main window.

7.1 Main Window

The main window in the Anybus Configuration Manager (ACM) is divided into 4 sections as follows:



1. Pull-down Menus & Toolbar

The toolbar provides quick access to frequently used functions.

2. Navigation Section

This section is the main tool for building, selecting and altering different levels of the subnetwork configuration. On most entries, right-clicking will give access to the different selections related to that particular entry.

3. Parameter Section

This section holds a list of parameters or options related to the currently selected entry in the Navigation Section.

The parameter value may be specified either using a selection box or entering a value manually, depending on parameter.

4. Information Section





This section presents information related to the parameter where the pointer is hovering.

7.1.1 Pull-down Menus

Some of these entries are available directly on the toolbar as well. The toolbar icon is shown next to these entries.

File

This menu features the following entries:

- **New**
Create a new configuration. 
- **Open...**
Open a previously created configuration.
A configuration is saved with the file extension .hcg. 
- **Save**
Save the current configuration. 
- **Save As...**
Save the current configuration under a new name. 
- **Recent Files**
Displays a list of recently accessed configurations
- **Exit**
Close the Anybus Configuration Manager.

Edit

This menu features the following entries:

- **Undo**
Undo the most recent action. Repeat to undo more actions.
- **Redo**
Redo the most recent undo.

Online

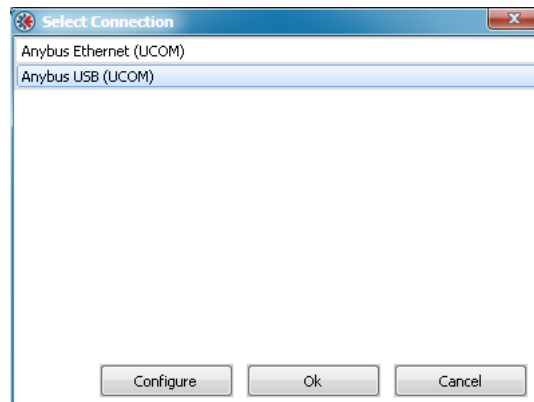
This menu features the following entries:

- **Select Connection**

This entry gives the opportunity to select connection for the module.

See also

- “Select Connection” on page 42



- **Connect/Disconnect**

This entry connects/disconnects the configuration tool to the module.



- **Upload Configuration**

This entry uploads a previously downloaded configuration to the Anybus Configuration Manager.



- **Download Configuration**

This entry downloads the configuration to the Anybus Communicator CAN. Any previously downloaded configuration will be overwritten.¹



1. ‘Download Configuration’ will only be available if there is a valid configuration to download. Please check the Diagnostics/Status page for information about warnings and faults. See “Diagnostics/Status” on page 47.

Tools

This menu features the following entries:

- **Monitor/Modify**

This entry opens the Monitor/Modify window that gives easy access to monitoring and modifying the transaction data.

- See “Monitor/Modify” on page 44



- **CAN Line Listener**

Listen in on the CAN communication on the subnetwork.

- See “CAN Line Listener” on page 45



- **Address Overview**

Displays the usage of the different parts of the internal memory of the module.

- See “Address Overview” on page 46



- **Diagnostics/Status**

Displays diagnostics and status of the Communicator and the present configuration.

- See “Diagnostics/Status” on page 47



- **Change Module Password**

Gives the opportunity to change the download and upload passwords for the module.

- See “Password” on page 50



- **Project Summary**

Displays information and a summary of the present configuration. The information is saved in html format and can be displayed in any browser.

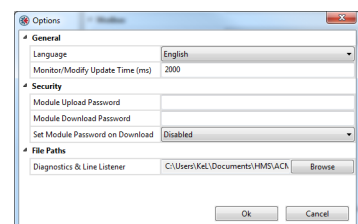
- See “Project Summary” on page 49



- **Options**

Selecting this entry gives access to more settings, that can be used to adapt the behavior of the Communicator.

- See “Options” on page 51



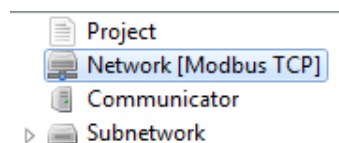
Help

This menu features the following entry:

- **About...**

Displays information about the Anybus Configuration Manager.

8. Basic Settings



8.1 Project

Selecting 'Project' will give the opportunity to enter and store project information.

Project name, project creator, version and description can be entered.

| Project Information | |
|---------------------|-----------------------------|
| Name | Example |
| Creator | Appl. Dev. |
| Version | 1.0 |
| Description | This project is an example. |

8.2 Network Settings

Select 'Network' in the Navigation Section to gain access to the parameters described in this section.

General

During start-up the fieldbus interface of the Communicator is initialized to fit the configuration created in the Anybus Configuration Manager. Optionally, some initialization parameters can be set manually to provide better control over how the data shall be treated by the Communicator.

Network Type

The Anybus Configuration Manager supports a wide range of networking systems. Make sure that this parameter is set to the correct network type.

Selecting Modbus-TCP will give access to the TCP/IP Settings, see below.

| Network | |
|--------------|----------------|
| Network Type | PROFIBUS DP-V1 |
| | CANopen |
| | CC-Link |
| | ControlNet |
| | DeviceNet |
| | EtherCAT |
| | EtherNet/IP |
| | Modbus RTU |
| | Modbus TCP |

8.2.1 TCP/IP Settings

The Anybus Communicator CAN offers different modes of operation regarding the network settings.

DHCP/BootP

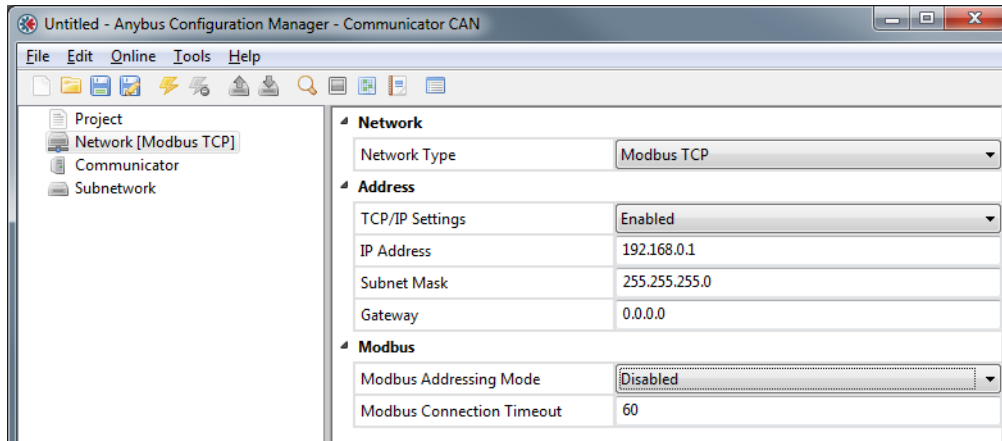
At the first startup of the module 'TCP/IP settings' are disabled and the Anybus Communicator CAN retrieves the settings from a DHCP or BootP server by default. If no DHCP server is found, the network configuration can be accessed via HICP, see "Anybus IPconfig (HICP)" on page 63.

At subsequent startups the TCP/IP settings will only be retrieved from a DHCP or BootP if 'TCP/IP settings' are disabled.

If no current settings are available the Communicator will halt and indicate an error on the on-board status LEDs (the network configuration may however still be accessed via HICP, see "Anybus IPconfig (HICP)" on page 63.

Settings specified in Anybus Configuration Manager

The network settings can be configured in the Anybus Configuration Manager if ‘TCP/IP Settings’ are enabled.



Note that in such cases, the contents of the system file ‘ethcfg.cfg’ will be ignored completely, causing the following behavior:

- DNS services will not be available
- Domain and Host name cannot be set
- Settings received from the network (i.e. via HICP or DHCP) will be lost in the event of a power loss or reset.

This behavior will persist for as long as ‘TCP/IP Settings’ is enabled, also if the module is restarted, i.e. the settings will be valid after a restart.

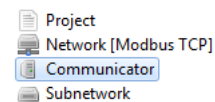
8.2.2 Modbus Settings

Also available are settings for Modbus:

- The Modbus Addressing Mode is disabled by default, see “Addressing Modes” on page 21.
- The Modbus Connection Timeout configures the timeout value used for Modbus-TCP connections. If a connection does not receive a request within the specified time, the connection will be shut down by the Modbus-TCP server. If the value is set to zero no timeout is used for Modbus connections. Valid values are 0, 10 - 65535 (s).

8.3 Communicator Settings

Select 'Communicator' in the Navigation Section to gain access to the parameters described in this section. The figure shows the available parameters.



| | |
|--------------------------|-------------------------------------|
| General | |
| Control/Status Word | Enabled |
| Start-up Operation Mode | Idle |
| Transaction Live List | Map 16 transactions (2 bytes) |
| Statistics | |
| Counters | Enable Receive and Transmit Counter |
| Receive Counter Address | 0x004 |
| Transmit Counter Address | 0x006 |
| Fatal Event | |
| Action | Stay in Safe-State |

General

| Parameter | Comment |
|------------------------------------|--|
| Control/Status Word ^a | If the Control/Status word is enabled it occupies the first two bytes of the out/in area of the memory. See also.. "Control/Status Word" on page 17 |
| Start-up Operation Mode | If the Control Word is enabled, it is possible to decide the start-up mode of the subnetwork. The start-up mode can be either 'Run' or 'Idle'. |
| Transaction Live List ^a | If the Transaction Live List is enabled it is mapped from the beginning of the input area or, if the Control/Status Word is enabled, after the Status Word. It is possible to map from 8 to 128 transactions, in steps of 8. Each transaction is represented by a bit that tells the system whether the transaction is alive or not. See also ... "Transaction Live List" on page 18 |

a. If the Control/Status Word or the Transaction Live List are going to be used, it is recommended to enable these before any frames are added when building the configuration, to avoid memory address collisions.

Statistics

| Parameter | Comment |
|--------------------------|--|
| Counters ^a | The receive counter and the transmit counter count successful CAN messages ^b on the subnetwork. If enabled, the counters can be mapped to the input data area. The first free address in the input data area is selected by default. The counters can be disabled and enabled separately. |
| Receive Counter Address | Enter the address in the input data area where the receive counter shall be mapped. The receive counter occupies 2 bytes. |
| Transmit Counter Address | Enter the address in the input data area where the transmit counter shall be mapped. The transmit counter occupies 2 bytes. |

a. If any counters are going to be used, it is recommended to enable these before any frames are added when building the configuration, to avoid address collisions in the memory.

b. The messages are counted only if they are configured in the Anybus Configuration Manager

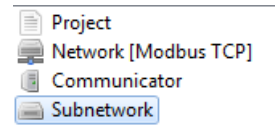
Fatal Event

The action in case of a fatal software event is decided by this parameter

| Parameter | Values | Comment |
|-----------|--------------------|---|
| Action | Stay in Safe-State | The Communicator will be locked in the safe state |
| | Software Reset | The software will be reset and the Communicator will be restarted automatically |

8.4 Subnetwork Settings

Select 'Subnetwork' in the Navigation Section to gain access to the settings described in this section.



General

| Parameter | Values | Comment |
|--------------------------|---|---|
| Bit Rate | 20 kbit/s 50 kbit/s 100 kbit/s 125 kbit/s 200 kbit/s 250 kbit/s 500 kbit/s 800 kbit/s 1000 kbit/s | Select CAN bit rate on the subnetwork. |
| Bus Off Action | No action Automatic Reset | Select what will happen to the CAN controller when the CAN network goes bus off. Available only when the Control/Status Word is not used. Please note that when enabling the Control/Status Word, this parameter will automatically be set to 'No Action'. |
| 11/29-bit CAN Identifier | 11 bit 29 bit | Select CAN identifier size on the subnetwork If there are transactions configured when this parameter is changed, the following will happen: - a change from 11 bit to 29 bit identifier will cause the identifier to be padded with zeroes up to 29 bits, keeping the 11 bits at the same location. - a change from 29 bit to 11 bit identifier will cause the upper 18 bits to be deleted and the lower 11 bits kept. WARNING! This may in some cases cause faulty CAN identifiers. |
| Silence Time (ms) | 0 - 65535 | Default = 0 (disabled) The minimum time that must elapse between the end of a message and the beginning of the next message. If, for example, a device on the subnetwork is slow and/or does not have a queue for messages, it may be necessary to enter a pause in between messages to ensure that all messages are handled correctly. |

9. Groups and Transactions

9.1 General

The configuration of the Communicator is set up in groups, each containing one or more transactions. Please note that the groups do not represent a physical device on the CAN network. They are a means for structuring the application, and maintaining an overview of it. The maximal number of groups is 128.

A transaction can be either a Produce, a Consume or a Query/Response transaction. Each transaction holds one or more CAN frames, which transport the data on the network. A total of 128 transactions is allowed, and a total of 256 CAN frames.

Each CAN frame can hold up to 8 bytes of data.

Groups and transactions as well as frames and objects (described in the next section) can be copied and pasted in the configuration tree, but only at the same level as they were copied from, or their parent.

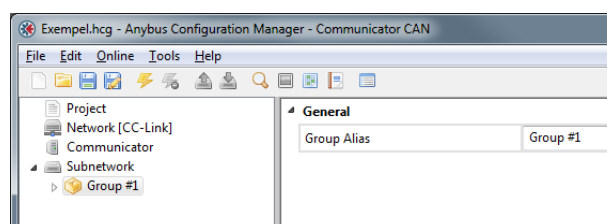
9.2 Groups

To create a group, right click on 'Subnetwork' and select 'Add Group'. The name of the group can be changed by selecting 'Group' and then entering a new name at 'Group Alias'.

If you want to insert another group, right click on 'Subnetwork' once more. The new group will be added to the end of the list of groups.

If you right click on a group and select 'Insert Group', the new group will be inserted before the selected group.

It is recommended to change the group name, to better present the configuration.

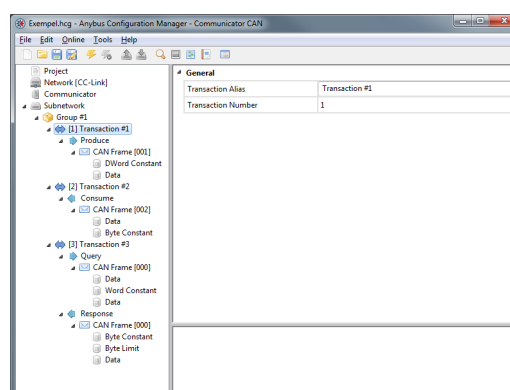


9.3 Transactions

There are five kinds of transactions: Produce, Consume, Query/Response, Dynamic Produce and Dynamic Consume.

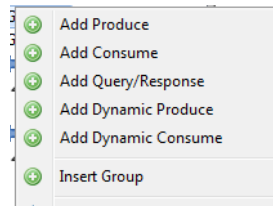
Selecting the transaction will give the option to give the transaction an alias. The order of the transactions in the tree is given as the transaction number in the parameter section. Each transaction number corresponds to a bit in the transaction live list that can be mapped to the input data area.

Note: The transaction live list is always available in the Diagnostics/Status window, even when it is not mapped to the input data area in the memory.



To add a transaction to the group, right click on the group and select either Add Produce, Add Consume, Add Query/Response, Add Dynamic Produce or Add Dynamic Consume.

Each transaction holds one CAN frame by default when added to a group. The dynamic transactions can not hold more than one CAN frame.



9.3.1 Produce

A produce transaction transmits CAN frames on the CAN network for all devices on the network to listen to. A CAN device on the network will use the identifier of the produce transaction to decide if the data is meant for it or not. The Communicator operates as any other device on the CAN network, that produces and transmits data on the network. Selecting 'Produce' gives access to the following parameters:

| General | |
|-------------------------------|------------|
| Produce Alias | Produce |
| Offline Options | Clear Data |
| Update Mode | Cyclically |
| Update on RTR | Enabled |
| Transmission Complete Byte | Enabled |
| Transmission Complete Address | 0x000 |
| Timing | |
| Update Time (ms) | 10 |

| Parameter | Value | Comment |
|-------------------------------|-----------------------------------|---|
| Produce Alias | - | An alias for the produce transaction (max 16 characters) |
| Offline Options | Clear Data | Select what will happen to the output data if the Modbus-TCP network goes offline |
| | Freeze Data | |
| | Stop Transaction | |
| Update Mode | Cyclically | Defines how the transmission of the transaction is triggered |
| | On Data Change | |
| | Single Shot | |
| | Trigger Byte | |
| Update on RTR | Disabled | If a message on the configured CAN identifier for a produce transaction is received with the RTR (Remote Transmission Request) bit set, the produce transaction is triggered to be sent. Only available if only one CAN frame is configured in the transaction. |
| | Enabled | |
| Transmission Complete Byte | Disabled | When enabled, the Transmission Complete Byte is incremented each time a produce transmission is completed. |
| | Enabled | |
| Transmission Complete Address | First available address (default) | If the Transmission Complete Byte is enabled, enter the address here. |
| Update Time (ms) | 1000 (default) | When Update Mode 'Cyclically' is selected, this parameter defines the time interval (ms) between two transmissions. Valid range: 5 - 65535 |
| Trigger Byte Address | First available address (default) | When Update Mode 'Trigger Byte' is selected, this parameter specifies the address of the trigger byte. The transaction will be triggered on a change in this byte. |

Right click on 'Produce' to add another CAN frame. For the setup of CAN frames see "Configuration of CAN Frames" on page 39.

9.3.2 Consume

A consume transaction listens to CAN frames on the CAN network and collects data from a frame with a matching CAN identifier. The Communicator operates as any other device on the CAN network that listens to all data that is available on the network. Selecting 'Consume' gives access to the following parameters:

| General | |
|----------------------------|------------|
| Consume Alias | Consume |
| Offline Options | Clear Data |
| Consistency Check | Enabled |
| Timing | |
| Offline Timeout (ms) | 100 |
| Trigger | |
| Reception Trigger Byte | Enabled |
| Reception Trigger Address | 0x010 |
| Status | |
| Transaction Status Byte | Enabled |
| Transaction Status Address | 0x001 |

| Parameter | Value | Comment |
|----------------------------|-----------------------------------|---|
| Consumer Alias | - | An alias for the consume transaction (max 16 characters). |
| Offline Options | Clear Data | Select what will happen to the input data if the CAN subnetwork goes offline. |
| | Freeze Data | |
| Consistency Check | Disabled | When enabled, all frames in the transaction must be received before evaluation. The frames are verified to contain expected data according to the configuration. Once verified, the fieldbus process data is updated with the received data. When disabled, all frames will be evaluated individually and the fieldbus process data is updated directly. The Offline Timeout will be set to 0. |
| | Enabled | |
| Offline Timeout | 0 (default) | The maximum time before the transaction is considered to be lost. Use 0 to disable the timeout. Valid Range: 0, 10 - 65535. |
| Reception Trigger Byte | Disabled | When enabled, the Reception Trigger Byte is incremented each time a consume transaction is received. |
| | Enabled | |
| Reception Trigger Address | First available address (default) | If the Reception Trigger Byte is enabled, enter the address here. |
| Transaction Status Byte | Disabled | When enabled, the Transaction Status Byte is updated every time the status of the transaction is changed. |
| | Enabled | |
| Transaction Status Address | First available address (default) | If the Transaction Status Byte is enabled, enter the address here. |

Right click on 'Consume' to add another CAN frame. For the setup of CAN frames see "Configuration of CAN Frames" on page 39.

Transaction Status Byte

If enabled, the transaction status byte holds the following status information for each separate transaction.

| Bit | Name | Description |
|-------|--------------|--|
| 0 | Timeout | 0 - The transaction has not timed out. 1 - The transaction has timed out. |
| 1 | Data error | 0 - The transaction does not have data errors. 1 - The transaction has data errors. ^a |
| 2 | Not executed | 0 - The transaction has been executed at least once. 1 - The transaction has not been executed yet. |
| 3 - 7 | (Reserved) | (Always 0) |

a. Data errors that can be detected are 'data out of range', 'invalid data size' and 'non-matching constant bytes' (in Constant object).

9.3.3 Query/Response

In Query/Response mode the Communicator operates as a master and issues queries to the CAN network. The Communicator will then expect a response within the specified timeout. A Query/Response transaction includes both query CAN frames and response CAN frames.

Selecting Query will give the same options as selecting Produce, except 'Update on RTR', see:

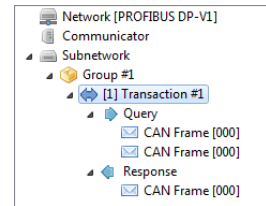
- “Produce” on page 34.

Selecting Response will give the same options as selecting Consume, see

- “Consume” on page 35.

Please note that the Offline Timeout value indicates the maximum time that the Communicator will wait for an answer before an error is issued. For a cyclic query, the offline timeout must be lower than the update time.

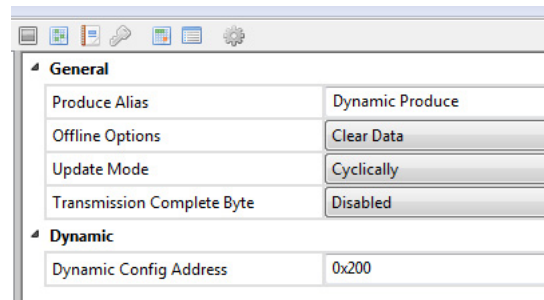
Right click on either 'Query' or 'Response' to add a new CAN frame. For the setup of CAN frames see “Configuration of CAN Frames” on page 39.



9.3.4 Dynamic Produce

Only one dynamic produce transaction can be added to a configuration. The function and parameters are similar to a produce transaction with the exception that some parameters can be accessed by the network master in the output data area or the general data area. At the same time, these parameters are not accessible via the Anybus Configuration Manager.

Parameters that are available in the Anybus Configuration Manager are given in the table below.



| Parameter | Value | Comment |
|-------------------------------|-----------------------------------|--|
| Produce Alias | - | An alias for the dynamic produce transaction (max 16 characters) |
| Offline Options | Clear Data | Select what will happen to the output data if the Modbus-TCP network goes offline |
| | Freeze Data | |
| | Stop Transaction | |
| Update Mode | Cyclically | Defines how the transmission of the transaction is triggered |
| | On Data Change | |
| | Single Shot | |
| | Trigger Byte | |
| Transmission Complete Byte | Disabled | When enabled, the Transmission Complete Byte is incremented each time a produce transmission is completed. |
| | Enabled | |
| Transmission Complete Address | First available address (default) | If the Transmission Complete Byte is enabled, enter the address here. |
| Trigger Byte Address | First available address (default) | When Update Mode 'Trigger Byte' is selected, this parameter specifies the address of the trigger byte. The transaction will be triggered on a change in this byte. |
| Dynamic Config Address | First available address (default) | This parameter specifies the memory address for the dynamically configurable parameters. |

Parameters that can be changed dynamically are stored at the specified memory address in the order given in the table below. These parameters are initialized at 0.

| Parameter | Size | Comment | |
|-----------------------|--------------|---|--|
| CAN-ID | 2 or 4 bytes | 11 bit or 29 bit CAN identifier | |
| Update time | 2 bytes | When Update Mode 'Cyclically' is selected, this parameter defines the time interval (ms) between two transmissions. Valid range: 5 - 65535. Cyclic update is stopped if this parameter is set to 0. | |
| Data length (bit 0-3) | 1 byte | The data length is given in bits 0-3 in this byte. At initialization this value is set to 0 and can later be changed up to the maximal data length entered for the data object in the Anybus Configuration Manager. | |
| RTR bit (bit 4) | | | Signals a remote transmission request. |
| Reserved (bit 5 - 7) | | | |

9.3.5 Dynamic Consume

Only one dynamic consume transaction can be added to a configuration. The function and parameters are similar to a consume transaction with the exception that some parameters can be accessed by the network master in the output data area or the general data area. At the same time, these parameters are not accessible via the Anybus Configuration Manager.

Parameters that are available in the Anybus Configuration Manager are given in the table below.

| Parameter | Value | Comment |
|----------------------------|-----------------------------------|--|
| Consumer Alias | - | An alias for the dynamic consume transaction (max 16 characters). |
| Offline Options | Clear Data | Select what will happen to the input data if the CAN subnetwork goes offline. |
| | Freeze Data | |
| Reception Trigger Byte | Disabled | When enabled, the Reception Trigger Byte is incremented each time a consume transaction is received. |
| | Enabled | |
| Reception Trigger Address | First available address (default) | If the Reception Trigger Byte is enabled, enter the address here. |
| Transaction Status Byte | Disabled | When enabled, the Transaction Status Byte is updated every time the status of the transaction is changed. ^a |
| | Enabled | |
| Transaction Status Address | First available address (default) | If the Transaction Status Byte is enabled, enter the address here. |
| Dynamic Config Address | First available address (default) | This parameter specifies the memory address for the dynamically configurable parameters. |

a. See "Transaction Status Byte" on page 35.

Parameters that can be changed dynamically are stored at the specified memory address in the order given in the table below. These parameters are initialized at 0.

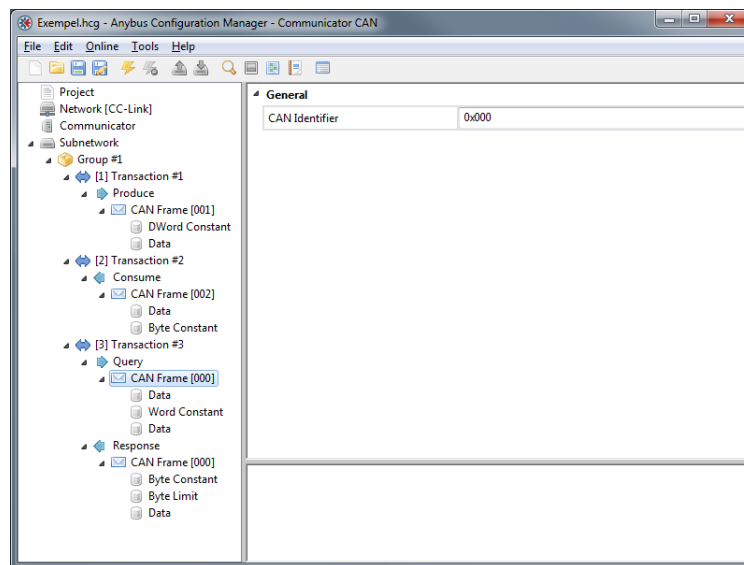
| Parameter | Size | Comment |
|-----------------------|--------------|---|
| CAN-ID | 2 or 4 bytes | 11 bit or 29 bit CAN identifier |
| Offline Timeout | 2 bytes | The maximum time before the transaction is considered to be lost. Use 0 to disable the timeout. Valid Range: 0, 10 - 65535. |
| Data length (bit 0-3) | 1 byte | The data length is given in bits 0-3 in this byte. At initialization this value is set to 0 and can later be changed up to the maximal data length entered for the data object in the Anybus Configuration Manager. |
| Reserved (bit 4 - 7) | | |

10. Configuration of CAN Frames

10.1 General

Each transaction includes one or more CAN frames. A total of 256 CAN frames is allowed. Right-clicking on a transaction will give the opportunity to add another frame to the transaction.

The Anybus Configuration Manager makes it possible to decide the configuration of the 8 bytes of data that can be included in each frame. The configuration manager automatically allocates memory space in the input and output areas of the Communicator for the data objects that are configured in the frames. The result can be seen in the Address Overview, see page 46. Any address conflicts will turn up red in this view.



Note: A CAN frame can not contain more than 8 bytes of data. It is possible to configure the data area in each frame, but the size of the combination of objects must not exceed 8 bytes.

10.1.1 CAN Identifiers

Each frame has a CAN identifier, to make it possible for each node on the CAN network to recognize data meant for it. Default identifier is '0'. It can be changed by selecting the CAN Frame and enter the new CAN Identifier in the Parameter window.

The CAN frame has either a 11-bit identifier or a 29-bit identifier. If the size of the identifier is changed, an 11-bit identifier will have the 11 original bits padded with zeroes in front. A 29 bit identifier will have its 18 highest bits cut, which may cause a not valid 11-bit identifier.

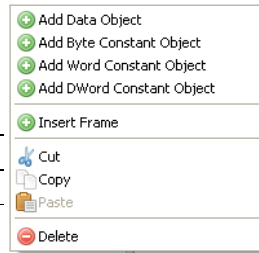
It is possible to have several frames in one transaction. The first frame in a Consume or Response transaction must have a CAN identifier that does not appear in any other Consume or Response transaction. Consecutive frames within a received transaction may have the same identifier, on two conditions:

- The first part of the data area in the frame is a byte, word or Dword constant with a unique value compared to other frames with the same identifier within the transaction.
- If any frame with another identifier is added to the transaction, it must not break the sequence of frames with identical identifiers.

10.2 Produce/Query CAN Frame

The following objects and parameters are configurable in a CAN frame in a produce transaction, or when used in the query part of a query/response transaction. To add objects to the 8 byte data area of the frame, right-click on CAN Frame.

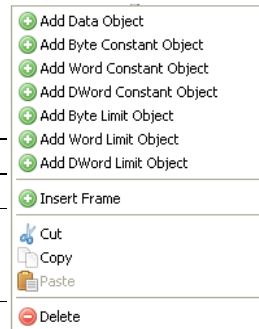
| Object | Parameters | Description/Comment |
|----------------|---|---|
| Data | Data Length (Bytes) | A data object can occupy 1 - 8 bytes (default =1). |
| | Data Address | Address in the data area where the object shall be mapped. Default: The first available position is used. |
| | Swap | Values: Result (original value = 0102 0304): No Swapping (default) 0102 0304 Word Swap 0201 0403 Double Word Swap 0403 0201 |
| Byte Constant | Value (1 byte, valid range: 0x00 - 0xFF) | Constant value to be transmitted (little endian). |
| Word Constant | Value(2 bytes, valid range: 0x0000 - 0xFFFF) | Constant value to be transmitted (little endian). |
| Dword Constant | Value (4 bytes, valid range: 0x00000000 - 0xFFFFFFFF) | Constant value to be transmitted (little endian). |



10.3 Consume/Response CAN Frame

The following objects and parameters are configurable in a CAN frame in a consume transaction, or when used in the response part of a query/response transaction. To add objects to the 8 byte data area of the frame, right-click on CAN Frame.

| Object | Parameters | Description/Comment |
|----------------|---|---|
| Data | Data Length (Bytes) | A data object can occupy 1 - 8 bytes (default =1). |
| | Data Address | Address in the data area where the object shall be mapped. Default: The first available position shall be used. |
| | Swap | Values: Result (original value = 0102 0304): No Swapping (default) 0102 0304 Word Swap 0201 0403 Double Word Swap 0403 0201 |
| Byte Constant | Value (1 byte, valid range: 0x00 - 0xFF) | When receiving a message with a constant, the received value will be checked against this value. If the values differ, the message will be ignored (little endian). |
| Word Constant | Value(2 bytes, valid range: 0x0000 - 0xFFFF) | When receiving a message with a constant, the received value will be checked against this value. If the values differ, the message will be ignored (little endian). |
| Dword Constant | Value (4 bytes, valid range: 0x00000000 - 0xFFFFFFFF) | When receiving a message with a constant, the received value will be checked against this value. If the values differ, the message will be ignored (little endian). |

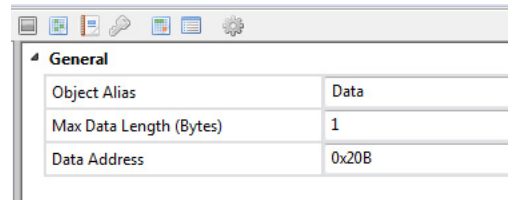


| Object | Parameters | Description/Comment |
|--|---------------|--|
| Byte Limit (1 byte, valid range: 0x00 - 0xFF) | Minimum Value | When receiving a message with a limit object, the received value will be checked against the minimum value. If the received value is lower than the minimum value, the message will be ignored. |
| | Maximum Value | When receiving a message with a limit object, the received value will be checked against the maximum value. If the received value is larger than the maximum value, the message will be ignored. |
| Word Limit (2 bytes, valid range: 0x0000 - 0xFFFF) | Minimum Value | When receiving a message with a limit object, the received value will be checked against the minimum value. If the received value is lower than the minimum value, the message will be ignored. |
| | Maximum Value | When receiving a message with a limit object, the received value will be checked against the maximum value. If the received value is larger than the maximum value, the message will be ignored. |
| Dword Limit (4 bytes, valid range: 0x00000000 - 0xFFFFFFFF) | Minimum Value | When receiving a message with a limit object, the received value will be checked against the minimum value. If the received value is lower than the minimum value, the message will be ignored. |
| | Maximum Value | When receiving a message with a limit object, the received value will be checked against the maximum value. If the received value is larger than the maximum value, the message will be ignored. |

10.4 CAN Frames in Dynamic Transactions

A dynamic transaction must consist of one frame. Only one object, a data object, can be added to the 8 byte data area of this frame. .

The CAN Identifier of the CAN frame in a dynamic transaction can not be set in the Anybus Configuration Manager. The identifier is stored in the output data area or the general data area.

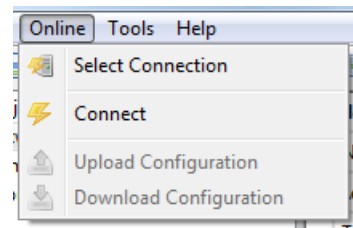


The Max Data Length of the data object is entered in the configuration manager, but the actual data length is given by the parameter mapped to the output data area or the general data area.

11. Online

The entries in the Online menu are used to select and connect to a Anybus Communicator CAN module and to upload/download the configuration.

- Select Connection
- Connect/Disconnect
- Upload Configuration
- Download Configuration



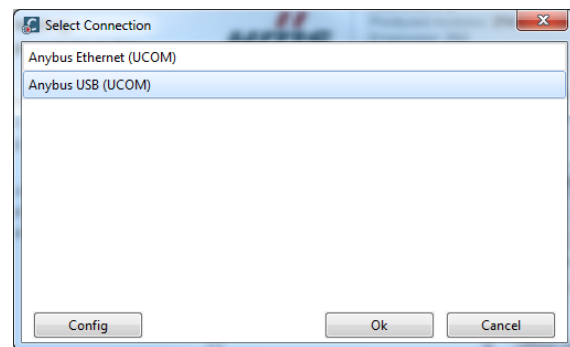
Connect the Communicator that is to be configured to your PC. The configuration can always be downloaded to the Communicator using the USB type 2 connector and the included USB cord. If the industrial network interface supports Ethernet, a suitable LAN cable can be used to download the configuration. Apply power to the module.

11.1 Select Connection

To be able to access the module, start by choosing 'Select Connection'.

The module supports Ethernet and USB connections.

Note: Although the Anybus Ethernet (UCOM) connection is available in the connection list, it can in practice only be used if the module has an RJ45 connector for Ethernet.



General

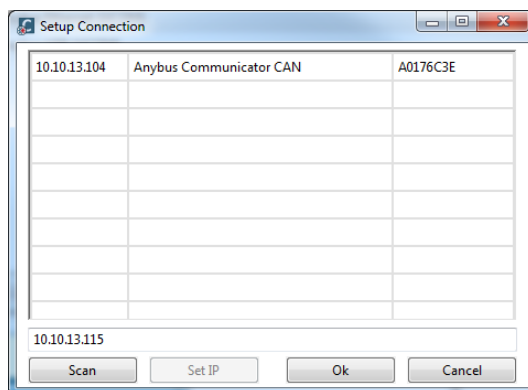
When a connection is selected, the PC running the ACM will lock to that specific Communicator. If the USB connection is used, the Communicator will be identified by its serial number. If the Ethernet connection is used, the IP address will be used for identification. If the configuration is to be downloaded to another module, using the same PC, the process of selecting connection will have to be repeated for that specific module.

It is recommended to select a specific Communicator for the connection, as this will diminish the risk of downloading the wrong configuration.

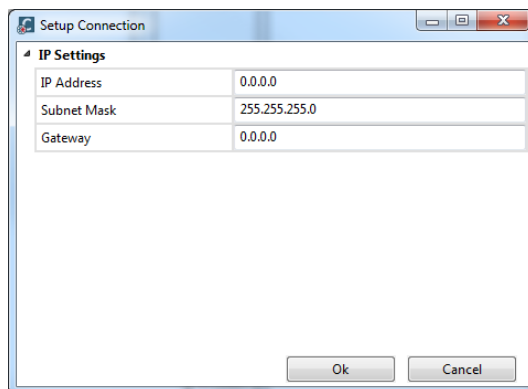
Anybus Ethernet (UCOM) Connection

Selecting ‘Anybus Ethernet (UCOM)’ and pressing ‘Configure’ opens a window where available Anybus Communicator CAN modules are listed.

To scan the network for further modules, press the ‘Scan’ button at the bottom of the window.



If the IP settings for a module are not set, it is possible to set these by pressing the ‘Set IP’ button. The module can be identified by the MAC Id listed in the rightmost column.

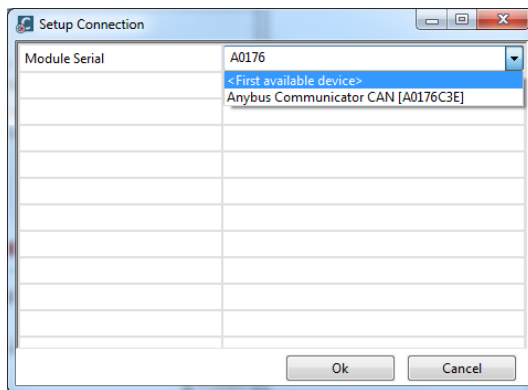


Note: These IP settings will be overwritten at the next power up of the module, if the settings are changed within the configuration.

Anybus USB (UCOM) Connection

To use a USB connection, select ‘Anybus USB (UCOM)’. Continue by pressing ‘Configure’ to open the ACM USB Connection window. The dropdown menu in this window shows available Anybus Communicator CAN modules.

There are also the options to either manually enter the serial number of a desired device or to select ‘first available device’ to download a configuration to.



11.2 Connect/Disconnect

The Communicator is connected/disconnected using this entry in the menu.

11.3 Download and Upload Configuration

Selecting ‘Download Configuration’ downloads the configuration to the Communicator. Any configuration previously present in the ACM will be overwritten.

Selecting ‘Upload Configuration’ will fetch the configuration in the connected Communicator to the Anybus Configuration Manager.

If the configuration is to be downloaded to another Communicator, change the connection, see ‘Select Connection’ on page 42.

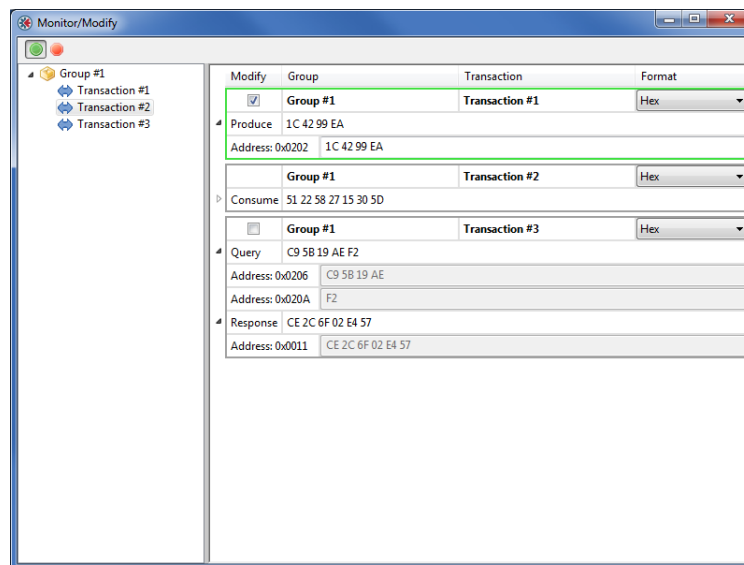
12. Anybus Configuration Manager Tools

The Anybus Configuration Manager (ACM) gives access to different tools for monitoring and controlling the module and the CAN subnetwork:

- Monitor/Modify
- CAN Line Listener
- Address Overview
- Diagnostics/Status
- Reassign Addresses
- Project Summary
- Password
- Options

12.1 Monitor/Modify

Selecting this option in the Tools menu opens this window, where the data areas of the transactions can be monitored. If the configuration downloaded to the Communicator is the same as is open in the ACM, it is possible to monitor and modify the transactions. Pressing the green button on the left starts the monitoring/modifying:



If Modify is enabled, it is possible to change the data values during runtime in Produce transactions and in the Query part of Query/Response transactions, i.e. only the out area of the Communicator can be modified. This will inhibit any data from the industrial network (Modbus-TCP), but input data from the CAN network will still be updated.

Note 1: Addresses in the general area range can not be modified. If a transaction only has addresses in the general area, the Modify check box will be disabled.

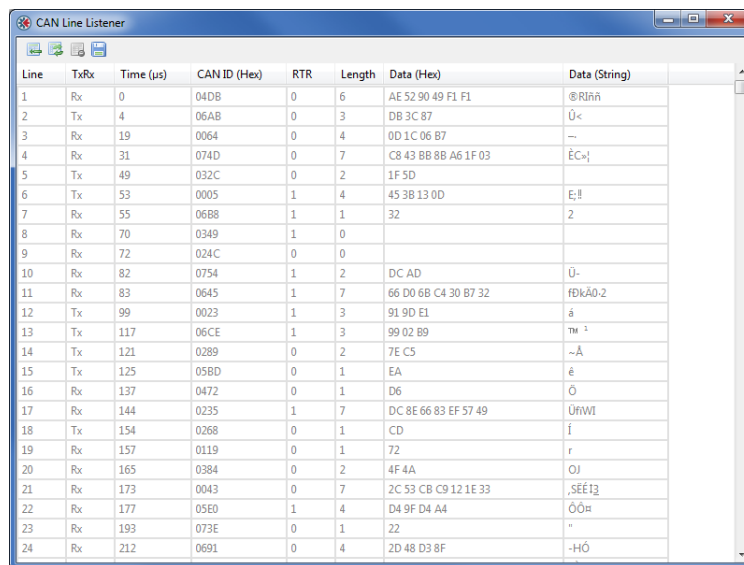
Note 2: If a transaction is defined to transmit on a change of state in a trigger byte, this transaction can not be modified by this tool.

12.2 CAN Line Listener

The CAN Line Listener gives the opportunity to log the traffic on the CAN network. Any log can be saved for later use. The 5000 latest frames are logged. This is done continuously, or it is possible to stop logging after 5000 frames from a defined time.

The CAN Line Listener shows all CAN frames present on the CAN network, not only those sent or received by the Communicator. Information about CAN frames, that have identifiers present in the configuration, that is downloaded to the Communicator, is shown in black text. Information about all other frames is shown in gray text. Clicking on the save icon will save the log at the location entered in the Tools/Options dialogue, see “Options” on page 51.

Please note that the configuration in the ACM and the configuration in the Communicator have to match.

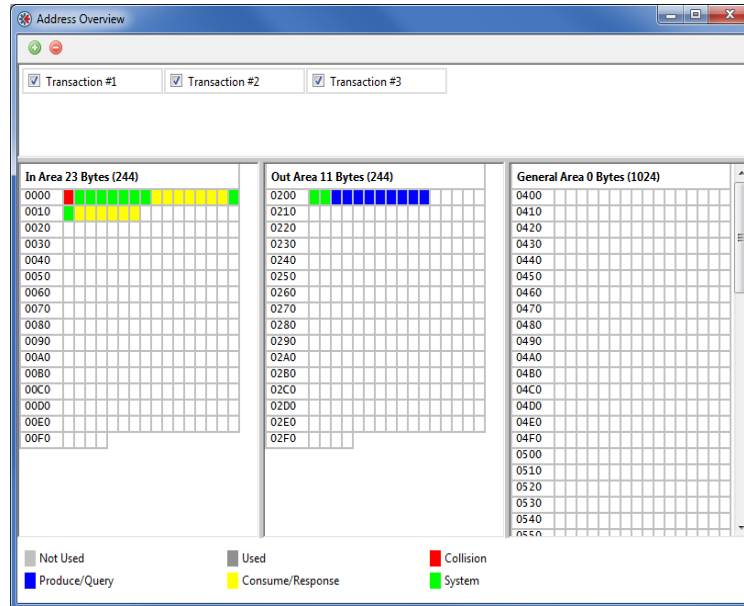


| Line | TxRx | Time (µs) | CAN ID (Hex) | RTR | Length | Data (Hex) | Data (String) |
|------|------|-----------|--------------|-----|--------|----------------------|---------------|
| 1 | Rx | 0 | 04DB | 0 | 6 | AE 52 90 49 F1 F1 | @R!ññ |
| 2 | Tx | 4 | 06AB | 0 | 3 | DB 3C 87 | Ù< |
| 3 | Rx | 19 | 0064 | 0 | 4 | 0D 1C 06 B7 | -- |
| 4 | Rx | 31 | 074D | 0 | 7 | C8 43 8B 8B A6 1F 03 | ËC-; |
| 5 | Tx | 49 | 032C | 0 | 2 | 1F 5D | |
| 6 | Tx | 53 | 0005 | 1 | 4 | 45 3B 13 0D | E;! |
| 7 | Rx | 55 | 0688 | 1 | 1 | 32 | 2 |
| 8 | Rx | 70 | 0349 | 1 | 0 | | |
| 9 | Rx | 72 | 024C | 0 | 0 | | |
| 10 | Rx | 82 | 0754 | 1 | 2 | DC AD | Ù- |
| 11 | Rx | 83 | 0645 | 1 | 7 | 66 D0 6B C4 30 B7 32 | fBkÄ0-2 |
| 12 | Tx | 99 | 0023 | 1 | 3 | 91 9D E1 | ä |
| 13 | Tx | 117 | 06CE | 1 | 3 | 99 02 B9 | ™ º |
| 14 | Tx | 121 | 0289 | 0 | 2 | 7E C5 | --Ä |
| 15 | Tx | 125 | 058D | 0 | 1 | EA | ë |
| 16 | Rx | 137 | 0472 | 0 | 1 | D6 | Ö |
| 17 | Rx | 144 | 0235 | 1 | 7 | DC 8E 66 83 EF 57 49 | ÙñW! |
| 18 | Tx | 154 | 0268 | 0 | 1 | CD | ï |
| 19 | Rx | 157 | 0119 | 0 | 1 | 72 | r |
| 20 | Rx | 165 | 0384 | 0 | 2 | 4F 4A | OJ |
| 21 | Rx | 173 | 0043 | 0 | 7 | 2C 53 CB C9 12 1E 33 | ,SEE!3 |
| 22 | Rx | 177 | 05E0 | 1 | 4 | D4 9F D4 A4 | ÖÖπ |
| 23 | Rx | 193 | 073E | 0 | 1 | 22 | " |
| 24 | Rx | 212 | 0691 | 0 | 4 | 2D 48 D3 8F | -HÖ |

Note: The CAN Line Listener will only display data if the RTR bit is NOT set.

12.3 Address Overview

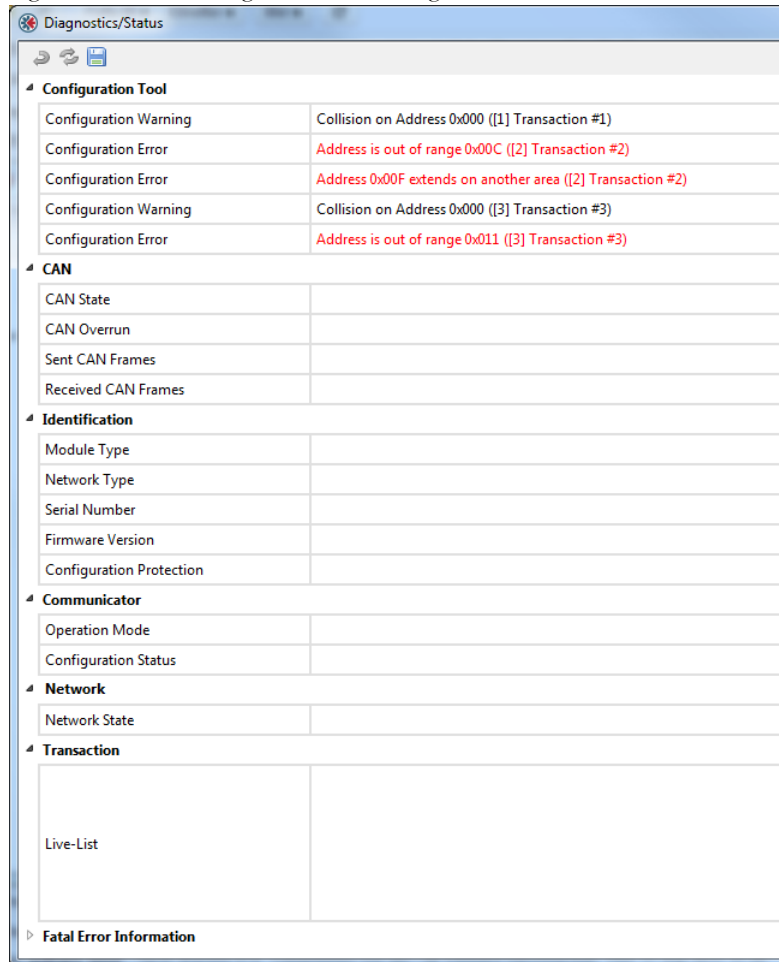
The Address Overview tool shows the usage of the different memory areas in the module. It gives an easy view of any collisions of data that are present in the different memory areas. If needed, the memory location for the data of one transaction at a time, can be shown.



Note: The Address Overview is an offline tool with no reference to the module. It shows the memory usage of the the configuration that is present in the Anybus Configuration Manager at the moment.

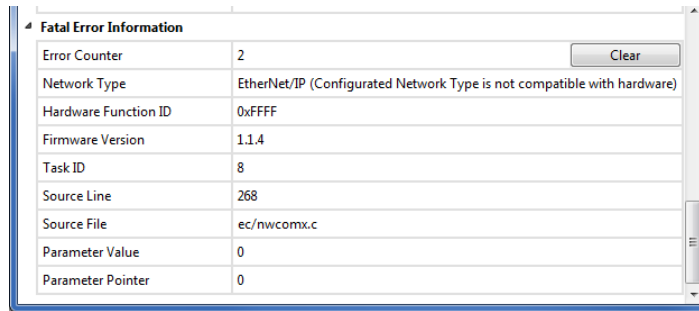
12.4 Diagnostics/Status

The Diagnostics/Status tool gives access to diagnostics and status information of different kinds.



| Item | Description |
|-------------------------|---|
| Configuration Tool | The configuration is validated by the ACM and any errors will be reported here, e.g. if the some address has been used for several transactions or if the same CAN identifier is used for more than one transaction. This is the only section of the Diagnostic/Status window that can be used when the configuration tool is not connected to a Communicator. Note: This information is valid for the configuration in the tool only, and does not relate to any configuration stored in the module. |
| CAN | Information on the status of the CAN subnetwork |
| Identification | Information on the module |
| Communicator | This item gives the operation mode and the configuration status of the Communicator |
| Network | Network state |
| Transaction | The live list will be shown here. It can also be kept in the input memory area, see 4-18 "Transaction Live List". |
| Fatal Error Information | If the Communicator is subject to a fatal error, this information is used by HMS support when troubleshooting the module. Please contact HMS support at www.anybus.com if a fatal error occurs. |

The figure below shows an example of what the section Fatal Error Information may look like.



| Fatal Error Information | |
|-------------------------|---|
| Error Counter | 2 |
| Network Type | EtherNet/IP (Configurated Network Type is not compatible with hardware) |
| Hardware Function ID | 0xFFFF |
| Firmware Version | 1.1.4 |
| Task ID | 8 |
| Source Line | 268 |
| Source File | ec/nwcomx.c |
| Parameter Value | 0 |
| Parameter Pointer | 0 |

The information in the Diagnostics/Status window can be saved by clicking on the save icon (file format CSV). The file will be saved at the location entered in the Tools/Options dialogue, see “Options” on page 51.

12.5 Reassign Addresses


This tool sorts all assigned data and puts it in order, from the beginning of the memory area and on. It also removes any collisions. The result can be seen using the tool “Address Overview”.

Note that there will be no confirmation notice after clicking the “Reassign Addresses” button.

12.6 Project Summary

Project information and a summary of the configuration is saved as a html file and can be read in any browser. The file is saved as the location entered in the Tools/Options dialogue, see “Options” on page 51.

Selecting ‘Project Summary’ will open a browser window that displays the summary:



Anybus Configuration Manager
Communicator CAN
Version 1.1.1.3

Information

| | |
|---------------------|---------------------|
| Project Name | Example |
| Project Creator | Appl. Dev. |
| Project Version | 1.0 |
| Project Description | Example Project |
| Document Date | 2011-09-16 13:52:45 |

Configuration

Network

| | |
|----------------------------|---------------|
| Network Type: | EtherNet/IP |
| TCP/IP Settings: | Enabled |
| IP Address: | 192.168.0.1 |
| Subnet Mask: | 255.255.255.0 |
| Gateway: | 0.0.0.0 |
| Modbus Addressing Mode: | Disabled |
| Modbus Connection Timeout: | 60 |
| Exact IO Match: | Disabled |

Communicator

| | |
|---------------------------|-------------------------------|
| Control/Status Word: | Enabled |
| Start-up Operation Mode: | Idle |
| Transaction Live List: | Map 16 transactions (2 bytes) |
| Receive Counter Address: | 0x004 |
| Transmit Counter Address: | 0x006 |
| Action: | Stay in Safe-State |

Subnetwork

| | |
|---------------------------|------------|
| Bit Rate: | 250 kbit/s |
| Bus Off Action: | No Action |
| 11/29-bit CAN Identifier: | 11 bit |

Group #1 > [1] Transaction #1

| | |
|----------------------------------|-------------------|
| Transaction: | Produce |
| Offline Options: | Clear Data |
| Update Mode: | Cyclically |
| Update on RTR: | Enabled |
| Transmission Complete Address: | 0x00F |
| Update Time (ms): | 10 |
| CAN Frame [001], DWord Constant: | Value: 0xABCDEF00 |
| CAN Frame [001], Data (4 bytes): | Address: 0x202 |

Group #1 > [2] Transaction #2

| | |
|----------------------------------|----------------|
| Transaction: | Consume |
| Offline Options: | Clear Data |
| Offline Timeout (ms): | 100 |
| Reception Trigger Address: | 0x010 |
| CAN Frame [002], Data (7 bytes): | Address: 0x008 |
| CAN Frame [002], Byte Constant: | Value: 0x22 |

Group #1 > [3] Transaction #3

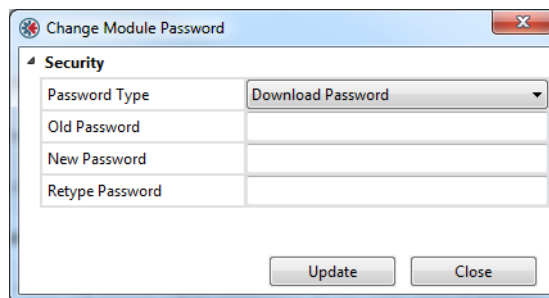
| | |
|----------------------------------|--------------------|
| Transaction: | Query |
| Offline Options: | Clear Data |
| Update Mode: | Cyclically |
| Transmission Complete Address: | 0x000 |
| Update Time (ms): | 10 |
| CAN Frame [000], Data (4 bytes): | Address: 0x206 |
| CAN Frame [000], Word Constant: | Value: 0x0000 |
| CAN Frame [000], Data (1 byte): | Address: 0x20A |
| Transaction: | Response |
| Offline Options: | Clear Data |
| Offline Timeout (ms): | 0 |
| CAN Frame [000], Byte Constant: | Value: 0x00 |
| CAN Frame [000], Byte Limit: | Value: 0x00 - 0x00 |
| CAN Frame [000], Data (6 bytes): | Address: 0x011 |

12.7 Password

It is possible to password protect a configuration. Passwords can be set both for uploading and downloading a configuration.

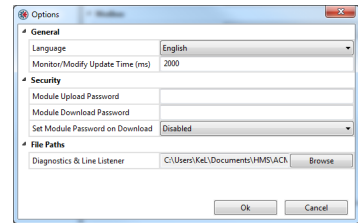
Passwords are set in the Options window, see “Options” on page 51. The same password can be used for uploading a configuration and for downloading a configuration. If the “Set Module Password on Download” parameter is enabled, the password will be downloaded to the module along with the configuration. When a configuration is protected by passwords you can still use the tools that are listed in this chapter. It is only the configuration by itself that is protected.

The passwords in a connected module can be changed directly from the ‘Change Module Password’ entry in the Tools menu. If no password has been set previously, the “Old Password” box should be left empty.



12.8 Options

Selecting this entry gives access to more settings, that can be used to adapt the behavior of the Communicator.



| Item | Subitem | Comment |
|-----------------------|---------------------------------|---|
| General | Language | |
| | Monitor/Modify Update Time (ms) | Enter the time between monitor/modify updates in milliseconds ^a . Valid range: 1000 to 60000 Default: 2000 |
| Security ^b | Module Upload Password | |
| | Module Download Password | |
| | Set Module Password on Download | Default: Disabled |
| File Paths | Diagnostics & Line Listener | By default the logs and the project summary are saved in the user catalog in Windows (My Documents\HMS\ACM Communicator CAN). To change this, browse to or enter the name of the folder where the logs shall be saved. ^c If the folder entered does not exist, the ACM will use the default address. |

- a. If a low value is entered, it may affect the performance of the Communicator, i.e. the data throughput delay will be longer.
- b. See "Password" on page 50.
- c. The log files contain time stamped versions of the CAN Line Listener, Diagnostics/Status and Project Summary. When the "Project Summary" button is pressed a time stamped version is automatically saved in a subfolder named "Project Summary Logs".
When the Save button in the CAN Line Listener window is pressed a time stamped version is automatically saved in a subfolder named "Line Listener Logs".
When the Save button in the Diagnostics/Status window is pressed a time stamped version is automatically saved in a subfolder named "Diagnostics Logs".

A. Technical Specification

A.1 Protective Earth (PE) Requirements

The product must be connected to protective earth (PE) via the DIN-rail connector in order to achieve proper EMC behavior.

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

A.2 Power Supply

Supply Voltage

The Communicator requires a regulated 24 V \pm 10% DC power source.

Power Consumption

The typical power consumption is 150 mA at 24 V.

A.3 Environmental Specification

A.3.1 Temperature

Operating

-25° to +55° Celsius

(Test performed according to IEC-60068-2-1 and IEC 60068-2-2.)

Non Operating

-40° to +85° degrees Celsius

(Test performed according to IEC-60068-2-1 and IEC 60068-2-2.)

A.3.2 Relative Humidity

The product is designed for a relative humidity of 5 to 95% non-condensing.

Test performed according to IEC 60068-2-30.

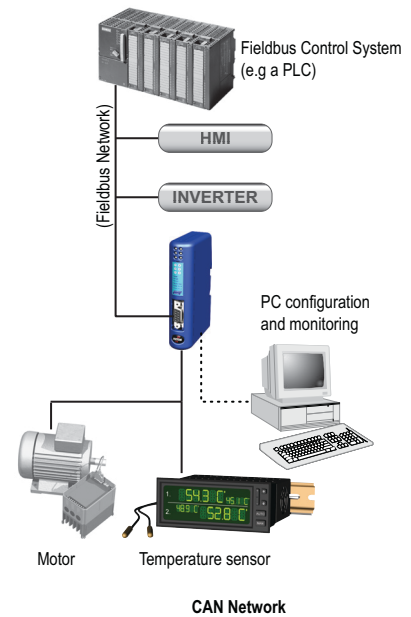
A.4 EMC (CE) Compliance

EMC compliance testing has been conducted according to the Electromagnetic Compatibility Directive 2004/108/EC. For more information please consult the EMC compliance document, see product/support pages for Anybus Communicator CAN to CANopen (slave) at www.anybus.com.

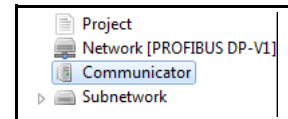
B. Configuration Example

This appendix gives an example of the configuration of an Anybus Communicator CAN to collect data from a temperature sensor and to control and monitor a motor.

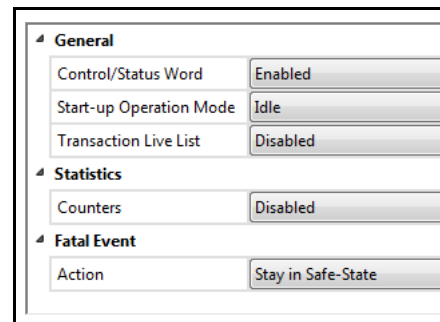
1. Start the Anybus Configuration Manager - Communicator CAN (ACM).
2. Choose industrial network. The example is the same irrespective of industrial network, but in an application it is important to choose network first, as the ACM will show the amount of data that can be transferred.



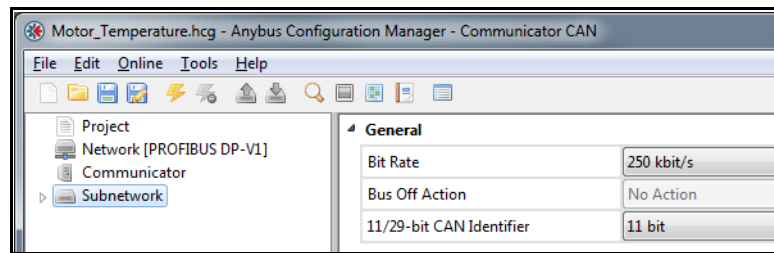
3. Select 'Communicator'.



- Enable the Control/Status Word.
If the Control/Status Word is to be used in a configuration, it is recommended to enable it before adding any transactions to the configuration. The Control/Status Word is positioned at the start of the memory, and this may cause address conflicts if any data objects have been configured previously.
For more information on the Control/Status Word, see page 18.
- Leave the rest of the parameters at default values.



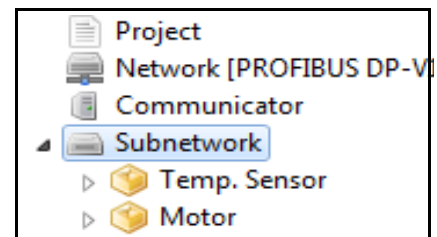
4. Select 'Subnetwork'.



If the Control/Status Word is enabled no Bus Off Action can be defined.

5. Add Groups.

- Right-click on 'Subnetwork' and add two groups to the navigation tree, one for each device on the CAN network.
- Rename them e.g. Temp. Sensor and Motor. Renaming is essential to enable other users than the designer of the application to comfortably monitor and modify the application.



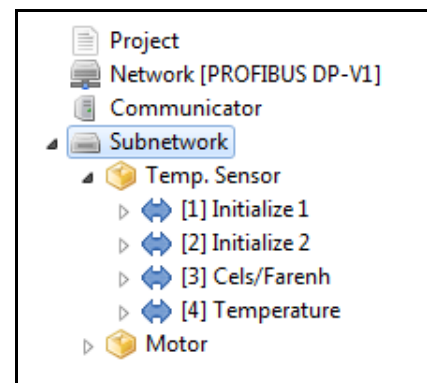
The CAN network is message-based, but using the group to structure the transactions will make it conceptually easier to build a configuration.

6. Add transactions to Temp. Sensor group.

The temperature sensor needs to be initialized. It needs instructions during runtime, and it will deliver temperature data to the Communicator.

A suitable transaction for an initialization is a query-response transaction which is run once at start up. A query-response transaction ensures an acknowledgement of a successful initialization. In this example, the initialization is performed in two steps.

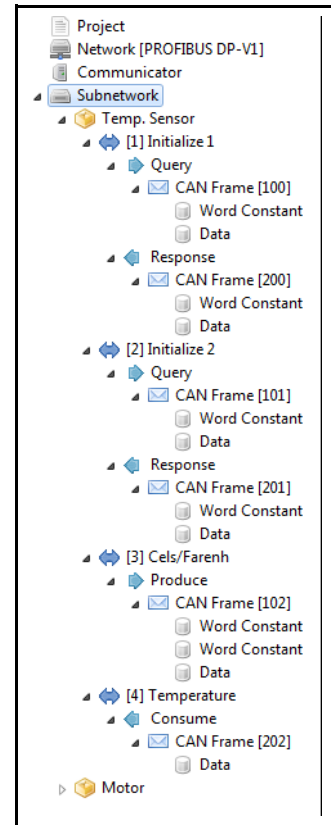
Also, instructions and information need to be sent to the sensor and data collected. A produce transaction sends information to the network and a consume transaction will collect information.



- Add two query-response transactions and rename them 'Initialize 1' and 'Initialize 2'.
- On each, select Query and change Update Mode to Single Shot. The transactions will be run once at startup to initialize the communication with the temperature sensor.
- Leave the rest of the parameters at default values.
- Add one Produce transaction to send information and instructions to the temperature sensor. Rename the transaction to 'Cels/Farenh' and set Update Mode to Cyclically.
- Leave the rest of the parameters at default values.
- Finally add one Consume transaction to collect the data cyclically from the temperature sensor. Rename the transaction to 'Temperature' and set Update Mode to Cyclically.
- Leave the rest of the parameters at default values.

7. Add frames to the transactions.

- Right-click on 'Query' in 'Initialize 1' and add a CAN frame.
- Select the frame.
- Set a unique CAN identifier to the frame. The CAN identifier shall be recognized on the network by the temperature sensor.
- Right click on the frame to define the components of the 8 byte data area in the frame, see figure to the right.
- Enter constant values where applicable.
- Right-click on 'Response' in 'Initialize 1' and repeat the procedure.



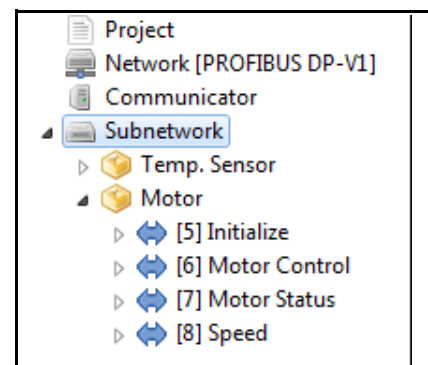
Addresses in the input and output areas of the internal memory will automatically be allocated to the data objects. It is possible to change these addresses, but it is recommended to finish configuration using default values. If any collisions appear, the addresses can be changed at a later stage. The ACM will not allow you to add a data or a constant object, that is larger than the remaining data area in the selected frame.

8. Repeat according to step 7 to add frames and contents to 'Initialize 2', 'Cels/Farenh' and 'Temperature'.

9. Add transactions to Motor group.

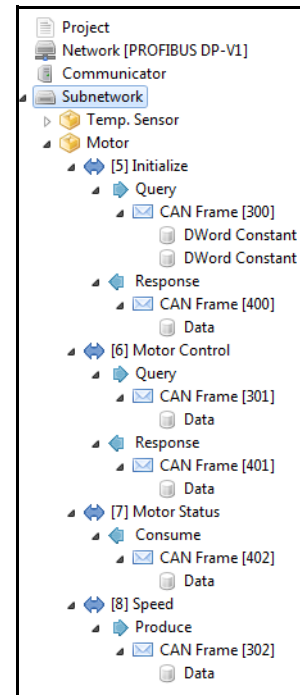
The motor needs to be initialized. It also needs instructions during runtime, and it will return status information to the Communicator. It is also possible to remotely set the speed of the motor.

- Add a query-response transaction and rename it 'Initialize'.
- Select Query and change Update Mode to Single Shot.

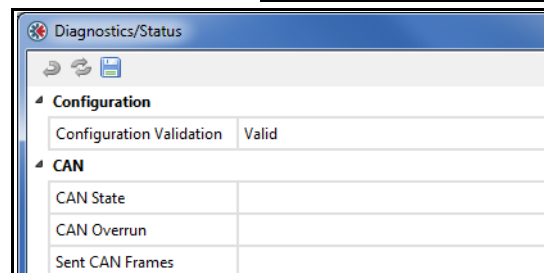


- Add one Query-Response transaction ("Motor Control") to control the motor during runtime.
- Set Update Mode to On Data Change.
- Add one Consume transaction ("Motor Status") to collect status cyclically from the motor.
- Finally add a Produce transaction ("Speed") to be able to change the speed of the motor.

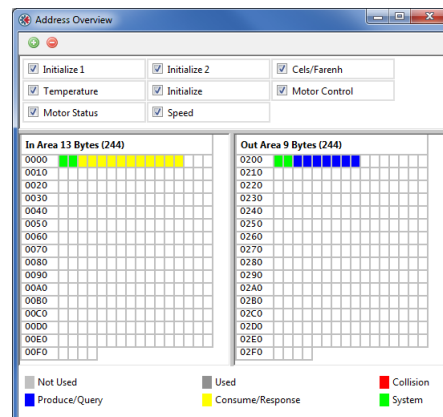
10. Add frames to the transactions, see step 7 above.



11. Check the validity of configuration in the Diagnostics/Status window.



If address conflicts are present, check the Address Overview to see which transactions cause the conflict. Change the addresses of the data objects in the frames to remove conflicts.



12. Download the configuration to the Communicator using the USB connection. Remove the USB cable when finished

A configuration can be saved at any time and opened at a later time for editing. Once it is valid it can be downloaded to the Anybus Communicator CAN.

C. Advanced IT Functionality

C.1 File System

C.1.1 General

General

The Anybus Communicator features a built-in file system, which is used to store information such as web files, network communication settings, e-mail messages etc.

Storage Areas

The file system consists of the different storage areas:

- **Non-volatile area (approx. 1.4 Mb)**
This section is intended for static files such as web files, configurations files etc.
- **Volatile area (approx. 1 Mb)**
This area is intended for temporary storage; data placed here will be lost in case of power loss or reset.

Conventions

- ‘\’ (backslash) is used as a path separator
- A ‘path’ originates from the system root and as such must begin with a ‘\’
- A ‘path’ must not end with a ‘\’
- Names may contain spaces (‘ ’) but must not begin or end with one.
- Names must not contain one of the following characters: ‘\ / : * ? “ < > |’
- Names cannot be longer than 48 characters (plus null termination)
- A path cannot be longer than 256 characters (filename included)
- The maximum number of simultaneously open files is 40
- The maximum number of simultaneously open directories is 40

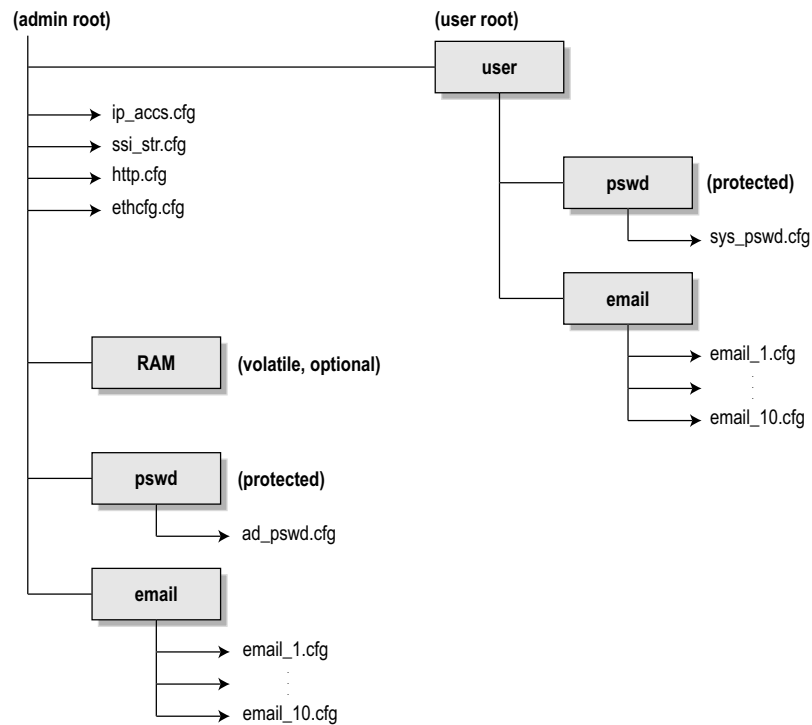
Important Note:

The non-volatile storage is located in FLASH memory. Each FLASH segment can only be erased approximately 100000 times due to the nature of this type of memory.

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

C.1.2 File System Overview



C.1.3 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™). Note that some of these files may also be altered by the Communicator itself, e.g. when using SSI (see “Server Side Include (SSI)” on page 73).

The format of the system files are based on the concept of ‘keys’, where each ‘key’ can be assigned a value, see example below.

Example:

```
[Key1]
value of key1

[Key2]
value of key2
```

The exact format of each system file is described in detail later in this document.

The contents of the above files can be redirected:

Example:

In this example, the contents will be loaded from the file ‘here.cfg’.

```
[file path]
|i\put\it\over\here.cfg
```

Note: Any directory in the file system can be protected from web access by placing the file `web-accs.cfg` in the directory, see “Authorization” on page 71.

C.2 Basic Network Configuration

C.2.1 General Information

The Anybus Communicator offers two modes of operation regarding the network settings (see below). Which mode to use is determined by the ‘TCP/IP Settings’ parameter in Anybus Configuration Manager, see “TCP/IP Settings” on page 29.

- **TCP/IP Settings: Enabled**

When operating in this mode, the contents of the system file ‘ethcfg.cfg’ will be ignored completely, causing the following behavior:

- DNS services will not be available
- Domain and Host name cannot be set
- E-mail services will not be available
- Settings received from the network (i.e. via HICP or DHCP) will be lost in the event of a power loss or reset.

- **TCP/IP Settings: Disabled**

When operating in this mode, the Anybus Communicator module will use the settings stored in the system file ‘ethcfg.cfg’. If this file is missing, the module will attempt to retrieve its settings via DHCP or HICP for 30 seconds. If no configuration has been received within this period, the module will halt and indicate an error on its status LEDs.

DHCP/BootP

The Anybus Communicator can retrieve the TCP/IP settings from a DHCP or BootP server. If no DHCP server is found, the module will fall back on its current settings (i.e. the settings currently stored in ‘ethcfg.cfg’).

If no current settings are available (i.e. ‘ethcfg.cfg’ is missing, or contains invalid settings), the module will halt and indicate an error on the onboard status LEDs. The network configuration may however still be accessed via HICP, see “Anybus IPconfig (HICP)” on page 63.

C.2.2 Ethernet Configuration File ('ethcfg.cfg')

General

To be able to participate on the network, the Anybus Communicator needs a valid TCP/IP configuration. These settings are stored in the system file '\ethcfg.cfg'.

File Format:

```
[IP address]
xxx.xxx.xxx.xxx

[Subnet mask]
xxx.xxx.xxx.xxx

[Gateway address]
xxx.xxx.xxx.xxx

[DHCP/BOOTP]
ON or OFF

[SMTP address]
xxx.xxx.xxx.xxx

[SMTP username]
username

[SMTP password]
password

[DNS1 address]
xxx.xxx.xxx.xxx

[DNS2 address]
xxx.xxx.xxx.xxx

[Domain name]
domain

[Host name]
anybus

[HICP password]
password
```

IP address

Subnet mask

Gateway address

DHCP/BootP
ON - Enabled
OFF - Disabled

SMTP server/login settings
Username and Password is only necessary if required by the server.

Primary and Secondary DNS
Needed to be able to resolve host names

Default domain name for not fully qualified host names

Host name

HICP password

The settings in this file may also be affected by...

- DCP (See "DHCP/BootP" on page 60).
- HICP (See "Anybus IPconfig (HICP)" on page 63)
- SSI (See "Server Side Include (SSI)" on page 73)
- DHCP/BootP (See "DHCP/BootP" on page 60)

See also...

- "FTP Server" on page 64
- "TCP/IP Settings" on page 29

C.2.3 IP Access Control

It is possible to specify which IP addresses that are permitted to connect to the Anybus Communicator. This information is stored in the system file ‘\ip_accs.cfg’.

File Format:

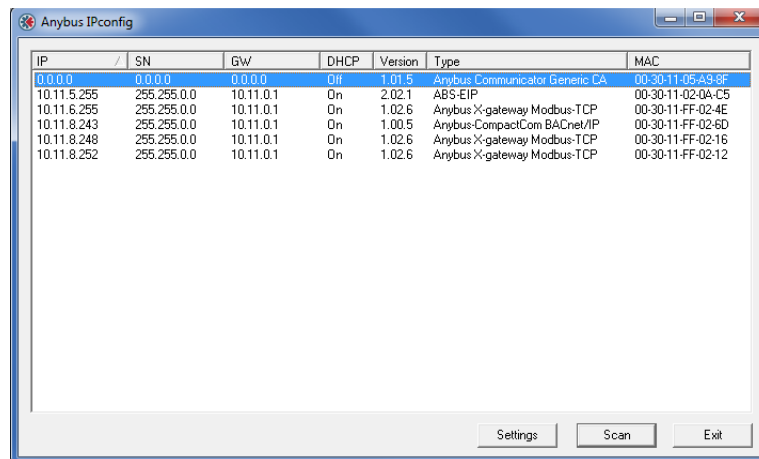
| | | |
|---------------------------------|---|--|
| [Web] xxx.xxx.xxx.xxx | • | Nodes listed here may access the web server |
| [FTP] xxx.xxx.xxx.xxx | • | Nodes listed here may access the FTP server |
| [Modbus-TCP] xxx.xxx.xxx.xxx | • | Nodes listed here may access the module via Modbus-TCP |
| [All] xxx.xxx.xxx.xxx | • | Fallback setting, used by the module when one or several of the keys above are omitted |

Note: ‘*’ may be used as a wildcard to select IP series.

C.2.4 Anybus IPconfig (HICP)

The Anybus Communicator supports the HICP protocol used by the Anybus IPconfig utility from HMS, which is included in the installer. This utility may be used to configure the network settings of any Anybus product connected to the network. Note that if successful, this will replace the settings currently stored in the configuration file (“ethcfg.cfg”).

Upon starting the program, the network is scanned for Anybus products. The network can be rescanned at any time by clicking ‘Scan’. In the list of detected devices, the Communicator will appear as ‘Anybus Communicator Generic CAN’. To alter its network settings, double-click on its entry in the list.

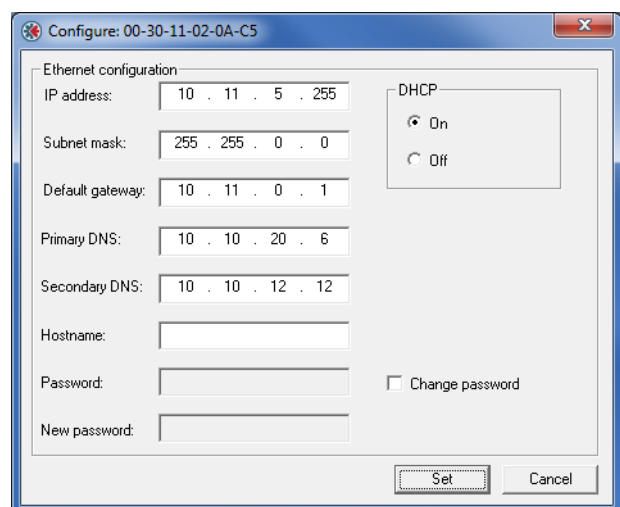


A window will appear, containing the IP configuration and password settings. Validate the new settings by clicking ‘Set’, or click ‘Cancel’ to abort.

Optionally, the TCP/IP settings may be protected from unauthorized access by a password. To enter a password, click on the ‘Change password’ checkbox, and enter the password under ‘New password’. When protected, any changes in the configuration requires that the user supplies a valid password.

When done, click ‘Set’. The new IP configuration will now be stored in the configuration file (“ethcfg.cfg”).

Note that if ‘TCP/IP Settings’ has been enabled in the Anybus Configuration Manager, any settings received via HICP will be lost in the event of a power loss or reset.



C.3 FTP Server

C.3.1 General

The built-in FTP server provides a way to access the file system using a standard FTP client.

The following port numbers are used for FTP communication:

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

Security Levels

The FTP server features two security levels; admin and normal.

- **Normal level users**
The root directory will be ‘\user’.
- **Admin level users**
The root directory will be ‘\’, i.e. the user has unrestricted access to the file system.

User Accounts

The user accounts are stored in two files, which are protected from web access:

- ‘\user\pswd\sys_pswd.cfg’
This file holds the user accounts for normal level users.
- ‘\pswd\ad_pswd.cfg’
This file holds the user accounts for admin level users.

File Format:

The format of these files are as follows:

```
Username1:Password1
Username2:Password2
Username3:Password3
```

Note 1: If no valid user accounts have been defined, the gateway will grant admin level access to all users. In such cases, the FTP accepts any username/password combination, and the root directory will be ‘\’.

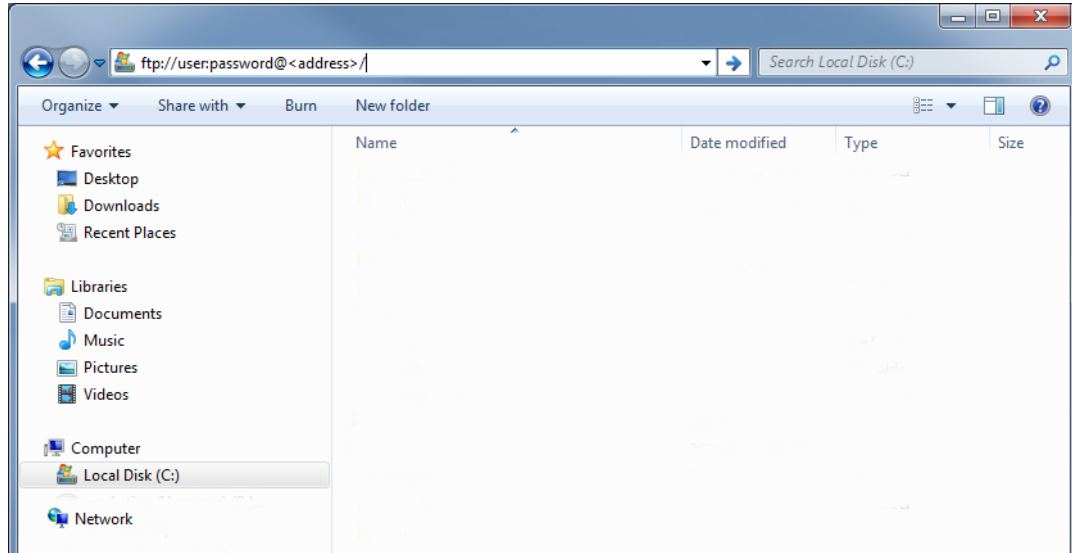
Note 2: The FTP server shares user accounts with the Telnet server.

C.3.2 FTP Connection Example (Windows Explorer)

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

1. Open the Windows Explorer by right-clicking on the ‘Start’ button and selecting ‘Explore’.
2. In the address field, type FTP://<user>:<password>@<address>
 - Substitute <address> with the IP address of the Anybus Communicator
 - Substitute <user> with the username

- Substitute <password> with the password
3. Press enter. The Explorer will now attempt to connect to the gateway using the specified settings. If successful, the built in file system is displayed in the Explorer window.



C.4 Telnet Server

C.4.1 General

The built-in Telnet server provides a way to access the file system using a standard Telnet client. The server communicates through TCP port 23.

Security Levels

Just like the FTP server, the Telnet server features two security levels; admin and normal.

- **Normal level users**
The root directory will be ‘\user’.
- **Admin level users**
The root directory will be ‘\’, i.e. the user has unrestricted access to the file system.

User Accounts

The Telnet server shares user accounts with the FTP server. If no valid user accounts have been defined, the gateway will grant admin level access to all users. In such case, no login is required, and the root directory will be ‘\’.

For more information, see “User Accounts” on page 64

C.4.2 General Commands

admin

- **Syntax**
admin
- **Description**
Provided that the user can supply a valid admin username/password combination, this command provides admin access rights to normal level users.

exit

- **Syntax**
exit
- **Description**
This command closes the Telnet session.

help

- **Syntax**
help [general|diagnostic|filesystem]

- **Description**

If no argument is specified, the following menu will be displayed.

General commands:

```
help          - Help with menus
version       - Display version information
exit         - Exit station program
```

Also try 'help [general|diagnostic|filesystem]'

version

- **Syntax**

version

- **Description**

This command will display version information, serial number and MAC ID of the Ethernet module, in the Communicator.

C.4.3 Diagnostic Commands

arps

- **Syntax**

arps

- **Description**

Display ARP stats and table.

iface

- **Syntax**

iface

- **Description**

Display net interface stats.

routes

- **Syntax**

routes

- **Description**

Display IP route table.

sockets

- **Syntax**

sockets

- **Description**

Display socket list.

C.4.4 File System Operations

For commands where filenames, directory names or paths shall be given as an argument the names can be written directly or within quotes. For names including spaces the filenames must be surrounded by quotes. It is also possible to use relative pathnames using '.', '\', and '..'

append

- **Syntax**
append [file] ["The line to append"]
- **Description**
Appends a line to a file.

cd

- **Syntax**
cd [path]
- **Description**
Changes current directory.

copy

- **Syntax**
copy [source] [destination]
- **Description**
This command creates a copy of the source file at a specified location.

del

- **Syntax**
del [file]
- **Description**
Deletes a file.

dir

- **Syntax**
dir [path]
- **Description**
Lists the contents of a directory. If no path is given, the contents of the current directory is listed.

df

- **Syntax**
df
- **Description**
Displays filesystem info.

format

- **Syntax**
format

- **Description**

Formats the file system. This is a privileged command and can only be called in administration mode.

md

- **Syntax**
md [directory]
- **Description**
Creates a directory. If no path is given, the directory is created in the current directory.

mkfile

- **Syntax**
mkfile [filename]
- **Description**
Creates an empty file.

move

- **Syntax**
move [source] [destination]
- **Description**
This command moves a file or directory from the source location to a specified destination.

rd

- **Syntax**
rd [directory]
- **Description**
Removes a directory. The directory can only be removed if it is empty.

ren

- **Syntax**
ren [old name] [new name]
- **Description**
Renames a file or directory.

type

- **Syntax**
type [filename]
- **Description**
Types the contents of a file.

C.5 Web Server

C.5.1 General

The Anybus Communicator features a flexible web server with SSI capabilities. The built-in web pages can be customized to fit a particular application and allow access to I/O data and configuration settings.

The web server communicates through port 80.

See also...

- “Server Side Include (SSI)” on page 73

Protected Files

For security reasons, the following files are protected from web access:

- Files located in ‘\’
- Files located in ‘\pswd’
- Files located in a directory which contains a file named ‘web_accs.cfg’

Default Web Pages

The Anybus Communicator contains a set of virtual files which can be used when building a web page for configuration of network parameters. These virtual files can be overwritten (not erased) by placing files with the same name in the root of disc 0.

This makes it possible to, for example, replace the HMS logo by uploading a new logo named ‘\logo.jpg’. It is also possible to make links from a web page to the virtual configuration page. In such case the link shall point to ‘\config.htm’.

These virtual files are:

C.5.2 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.

File Format:

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

List of approved users.

```
[AuthName]
(message goes here)
```

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.

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

Example:

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

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

[AuthName]
Yeah. Whatsda passwoid?
```

Note that when using this feature, make sure to put the user/password files in a directory that is protected from web access, see "Protected Files" on page 71.

C.5.3 Content Types

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

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

It is possible to configure/reconfigure the reported content types, and which files that shall be scanned for SSI. This is done in the system file '\http.cfg'.

File Format:

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

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

Note: Up to 50 content types and 50 SSI file types may be specified in this file.

C.6 Server Side Include (SSI)

C.6.1 General

Server Side Include (from now on referred to as SSI) functionality enables dynamic content to be used on web pages and in e-mail messages.

SSI are special commands embedded in the source document. When the Anybus Communicator encounters such a command, it will execute it, and replace it with the result (when applicable).

Syntax

The 'X's below represents a command opcode and parameters associated with the command.

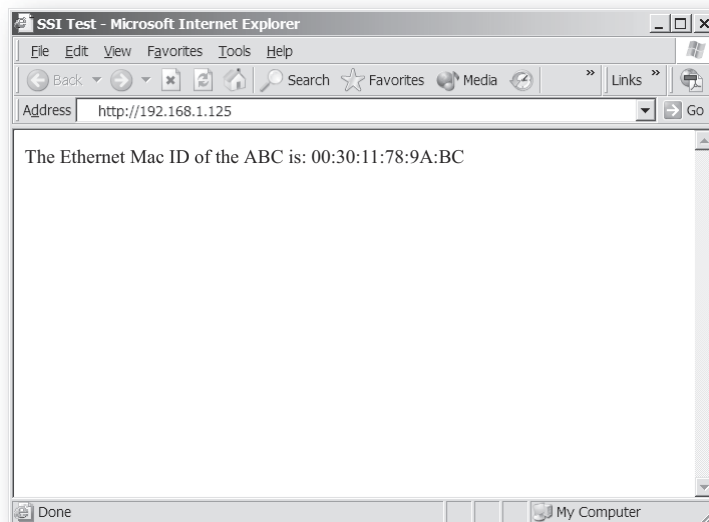
```
<?--#exec cmd_argument='XXXXXXXXXXXXXXXXXXXXXXXXXX'-->
```

Example

The following example causes a web page to display the Ethernet Mac ID of the gateway:

```
<HTML>
<HEAD><TITLE>SSI Test</TITLE></HEAD>
<BODY>
The Ethernet Mac ID of the ABC is:
<?--#exec cmd_argument='DisplayMacID'-->
</BODY>
</HTML>
```

Resulting webpage:



C.6.2 Functions

DisplayMacID

This function returns the MAC ID in format xx:xx:xx:xx:xx:xx.

Syntax:

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

DisplaySerial

This function returns the serial number of the network interface.

Syntax:

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

DisplayFWVersion

This function returns the main firmware revision of the network interface.

Syntax:

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

DisplayBLVersion

This function returns the bootloader firmware revision of the network interface.

Syntax:

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

DisplayIP

This function returns the currently used IP address.

Syntax:

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

DisplaySubnet

This function returns the currently used Subnet mask.

Syntax:

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

DisplayGateway

This function returns the currently used Gateway address.

Syntax:

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

.....**DisplayDhcpState()**

This function returns whether DHCP/BootP is enabled or disabled.

Syntax:

```
<?--#exec cmd_argument='DisplayDhcpState( "Output when ON", "Output when OFF" )'-->
```

.....**StoreConfig**

Note: This function cannot be used in e-mail messages.

This function stores a passed IP configuration in the configuration file '.cfg'.

Syntax:

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

Include this line in a HTML page and pass a form with new IP settings to it.

Accepted fields in form:

```
SetIp  
SetSubnet  
SetGateway  
SetDhcpState - value "on" or "off"  
SetDNS1  
SetDNS2  
SetHostName  
SetDomainName
```

Default output:

```
Invalid IP address!  
Invalid Subnet mask!  
Invalid Gateway address!  
Invalid IP address or Subnet mask!  
Invalid DHCP state!  
Invalid DNS1!  
Invalid DNS2!  
Configuration stored correctly.  
Failed to store configuration.
```

GetText()

Note: This function cannot be used in e-mail messages.

This function retrieves a text string from an object and stores it in the output data area.

Syntax:

```
<?--#exec cmd_argument='GetText( "ObjName", OutWriteString ( offset ), n )'-->
```

ObjName- Name of object.

offset - Specifies the destination offset from the beginning of the output data area.

n - Specifies maximum number of characters to read (optional)

Default output:

```
Success - Write succeeded
Failure - Write failed
```

printf()

This function includes a formatted string, which may contain data from the input and output data areas, on a web page. The formatting of the string is similar to the C-language function printf().

Syntax:

```
<?--#exec cmd_argument='printf("String to write", Arg1, Arg2, ..., ArgN)'-->
```

Like the C-language function printf() the "String to write" for this SSI function contains two types of objects: Ordinary characters, which are copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive argument to printf. Each conversion specification begins with the character % and ends with a conversion character. Between the % and the conversion character there may be, in order:

- Flags (in any order), which modify the specification:
 - which specifies left adjustment of the converted argument in its field.
 - + which specifies that the number will always be printed with a sign
 - (space) if the first character is not a sign, a space will be prefixed.
 - 0 for numeric conversions, specifies padding to the field with leading zeroes.
 - # which specifies an alternate output form. For o, the first digit will be zero. For x or X, 0x or 0X will be prefixed to a non-zero result. For e, E, f, g and G, the output will always have a decimal point; for g and G, trailing zeros will not be removed.
- A number specifying a minimum field width. The converted argument will be printed in a field at least this wide, and wider if necessary. If the converted argument has fewer characters than the field width it will be padded on the left (or right, if left adjustment has been requested) to make up the field width. The padding character is normally space, but can be 0 if the zero padding flag is present.
- A period, which separates the field width from the precision.
- A number, the precision, that specifies the maximum number of characters to be printed from a string, or the number of digits to be printed after the decimal point for e, E, or F conversions, or the number of significant digits for g or G conversion, or the minimum number of digits to be printed for an integer (leading 0s will be added to make up the necessary width)
- A length modifier h, l (letter ell), or L. "h" Indicates that the corresponding argument is to be printed as a short or unsigned short; "l" indicates that the argument is long or unsigned long.

The conversion characters and their meanings are shown below. If the character after the % is not a conversion character, the behavior is undefined.

| Character | Argument type, Converted to |
|-----------|---|
| d, i | byte, short; decimal notation (For signed representation. Use signed argument) |
| o | byte, short; octal notation (without a leading zero). |
| x, X | byte, short; hexadecimal notation (without a leading 0x or 0X), using abcdef for 0x or ABCDEF for 0X. |
| u | byte, short; decimal notation. |
| c | byte, short; single character, after conversion to unsigned char. |
| s | char*; characters from the string are printed until a "\0" is reached or until the number of characters indicated by the precision have been printed |
| f | float; decimal notation of the form [-]mmm.ddd, where the number of d's is specified by the precision. The default precision is 6; a precision of 0 suppresses the decimal point. |
| e, E | float; decimal notation of the form [-]m.ddddd e+-xx or [-]m.dddddE+-xx, where the number of d's specified by the precision. The default precision is 6; a precision of 0 suppresses the decimal point. |
| g, G | float; %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 zeros and trailing decimal point are not printed. |
| % | no argument is converted; print a % |

The arguments that can be passed to the SSI function *printf* are:

| Argument | Description |
|--------------------------------|---|
| InReadSByte(<i>offset</i>) | Read a signed byte from position <i>offset</i> in the Input Data area |
| InReadUByte(<i>offset</i>) | Read an unsigned byte from position <i>offset</i> in the Input Data area |
| InReadSWord(<i>offset</i>) | Read a signed word from position <i>offset</i> in the Input Data area |
| InReadUWord(<i>offset</i>) | Read an unsigned word from position <i>offset</i> in the Input Data area |
| InReadSLong(<i>offset</i>) | Read a signed longword from position <i>offset</i> in the Input Data area |
| InReadULong(<i>offset</i>) | Read an unsigned longword from position <i>offset</i> in the Input Data area |
| InReadString(<i>offset</i>) | Read a string (char*) from position <i>offset</i> in the Input Data area |
| InReadFloat(<i>offset</i>) | Read a floating point (float) value from position <i>offset</i> in the Input Data area |
| OutReadSByte(<i>offset</i>) | Read a signed byte from position <i>offset</i> in the Output Data area |
| OutReadUByte(<i>offset</i>) | Read an unsigned byte from position <i>offset</i> in the Output Data area |
| OutReadSWord(<i>offset</i>) | Read a signed word (short) from position <i>offset</i> in the Output Data area |
| OutReadUWord(<i>offset</i>) | Read an unsigned word (short) from position <i>offset</i> in the Output Data area |
| OutReadSLong(<i>offset</i>) | Read a signed longword (long) from position <i>offset</i> in the Output Data area |
| OutReadULong(<i>offset</i>) | Read an unsigned longword (long) from position <i>offset</i> in the Output Data area |
| OutReadString(<i>offset</i>) | Read a null-terminated string from position <i>offset</i> in the Output Data area |
| OutReadFloat(<i>offset</i>) | Read a floating point (float) value from position <i>offset</i> in the Output Data area |

scanf()

Note: This function cannot be used in e-mail messages.

This function reads a string passed from an object in a HTML form, interprets the string according to the specification in format, and stores the result in the output data area according to the passed arguments. The formatting of the string is equal to the standard C function call scanf().

Syntax:

```
<?--#exec cmd_argument='scanf( "ObjName", "format", Arg1, ..., ArgN), ErrVal1,
..., ErrvalN'-->
```

ObjName - The name of the object with the passed data string
format - Specifies how the passed string shall be formatted
Arg1 - ArgN - Specifies where to write the data
ErrVal1 -ErrValN - Optional; specifies the value/string to write in case of an error.

| Character | Input, Argument Type |
|-----------|---|
| d | Decimal number; byte, short |
| i | Number, byte, short. The number may be in octal (leading 0(zero)) or hexadecimal (leading 0x or 0X) |
| o | Octal number (with or without leading zero); byte, short |
| u | Unsigned decimal number; unsigned byte, unsigned short |
| x | Hexadecimal number (with or without leading 0x or 0X); byte, short |
| c | Characters; char*. The next input characters (default 1) are placed at the indicated spot. The normal skip over white space is suppressed; to read the next non-white space character, use %1s. |
| s | Character string (not quoted); char*, pointing to an array of characters large enough for the string and a terminating "\0" that will be added. |
| e, f, g | Floating-point number with optional sign, optional decimal point and optional exponent; float* |
| % | Literal %; no assignment is made. |

The conversion characters d, i, o, u and x may be preceded by l (letter ell) to indicate that a pointer to 'long' appears in the argument list rather than a 'byte' or a 'short'

The arguments that can be passed to the SSI function scanf are:

| Argument | Description |
|---------------------------------|--|
| OutWriteByte(<i>offset</i>) | Write a byte to position <i>offset</i> in the Output Data area |
| OutWriteWord(<i>offset</i>) | Write a word to position <i>offset</i> in the Output Data area |
| OutWriteLong(<i>offset</i>) | Write a long to position <i>offset</i> in the Output Data area |
| OutWriteString(<i>offset</i>) | Write a string to position <i>offset</i> in the Output Data area |
| OutWriteFloat(<i>offset</i>) | Write a floating point value to position <i>offset</i> in the Output Data area |

Default output:

```
Write succeeded
Write failed
```

IncludeFile()

This function includes the contents of a file on a web page.

Syntax:

```
<?--#exec cmd_argument='IncludeFile( "File name" )'-->
```

Default output:

| | |
|---------|-----------------------------|
| Success | - <File content> |
| Failure | - Failed to open <filename> |

SaveToFile()

Note: This function cannot be used in e-mail messages.

This function saves the contents of a passed form to a file. The passed name/value pair will be written to the file "File name" separated by the "Separator" string. The [Append|Overwrite] parameter determines if the specified file shall be overwritten, or if the data in the file shall be appended.

Syntax:

```
<?--#exec cmd_argument='SaveToFile( "File name", "Separator", [Append|Overwrite] )'-->
```

Default output:

| | |
|---------|-----------------------|
| Success | - Form saved to file |
| Failure | - Failed to save form |

SaveDataToFile()

Note: This function cannot be used in e-mail messages.

This function saves the data of a passed form to a file. The "Object name" parameter is optional, if specified, only the data from that object will be stored. If not, the data from all objects in the form will be stored.

The [Append|Overwrite] parameter determines if the specified file shall be overwritten, or if the data in the file shall be appended.

Syntax:

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

Default output:

| | |
|---------|-----------------------|
| Success | - Form saved to file |
| Failure | - Failed to save form |

C.6.3 Changing SSI output

There are two methods of changing the output strings from SSI functions:

1. Changing SSI output defaults by creating a file called "\ssi_str.cfg" containing the output strings for all SSI functions in the system
2. Temporarily changing the SSI output by calling the SSI function "SsiOutput()

SSI Output String File

If the file "\ssi_str.cfg" is found in the file system and the file is consistent with the specification below, the SSI functions will use the output strings specified in this file instead of the default strings.

The files shall have the following format:

```
[StoreEtnConfig]
Success: "String to use on success"
Invalid IP: "String to use when the IP address is invalid"
Invalid Subnet: "String to use when the Subnet mask is invalid"
Invalid Gateway: "String to use when the Gateway address is invalid"
Invalid IP or Subnet: "String to use when the IP address and Subnet mask does
not match"
Invalid DNS1: "String to use when the primary DNS cannot be found"
Invalid DNS2: "String to use when the secondary DNS cannot be found"
Save Error: "String to use when storage fails"
Invalid DHCP state: "String to use when the DHCP state is invalid"

[scanf]
Success: "String to use on success"
Failure: "String to use on failure"

[IncludeFile]
Failure: "String to use when failure"1

[SaveToFile]
Success: "String to use on success"
Failure: "String to use on failure"1

[SaveDataToFile]
Success: "String to use on success"
Failure: "String to use on failure"1

[GetText]
Success: "String to use on success"
Failure: "String to use on failure"
```

The contents of this file can be redirected by placing the line '[File path]' on the first row, and a file path on the second.

Example:

```
[File path]
\user\ssi_strings.cfg
```

In this example, the settings described above will be loaded from the file 'user\ssi_strings.cfg'.

Temporary SSI Output change

The SSI output for the next called SSI function can be changed with the SSI function "SsiOutput()" The next called SSI function will use the output according to this call. Thereafter the SSI functions will use

1. '%s' includes the filename in the string

the default outputs or the outputs defined in the file '\ssi_str.cfg'. The maximum size of a string is 128 bytes.

Syntax:

```
<?--#exec cmd_argument='SsiOutput( "Success string", "Failure string" )'-->
```

Example:

This example shows how to change the output strings for a scanf SSI call.

```
<?--#exec cmd_argument='SsiOutput ( "Parameter1 updated", "Error" )'-->  
<?--#exec cmd_argument='scanf( "Parameter1", "%d", OutWriteByte(0) )'-->
```

C.7 E-mail Client

C.7.1 General

The built-in e-mail client can send predefined e-mail messages based on trigger-events in input and output data areas. The client supports SSI, however note that some SSI functions cannot be used in e-mail messages (specified separately for each SSI function).

See also...

- “Server Side Include (SSI)” on page 73

Server Settings

The Anybus Communicator needs a valid SMTP server configuration in order to be able to send e-mail messages. These settings are stored in the system file ‘\ethcfg.cfg’.

See also...

- “Ethernet Configuration File (‘ethcfg.cfg’)” on page 61

Event-Triggered Messages

As mentioned previously, the e-mail client can send predefined messages based on events in the input and output data areas. In operation, this works as follows:

1. The trigger source is fetched from a specified location
2. A logical AND is performed between the trigger source and a mask value
3. The result is compared to a reference value
4. If the result is true, the e-mail is sent to the specified recipient(s).

Which events that shall cause a particular message to be sent, is specified separately for each message. For more information, see “E-mail Definitions” on page 82.

Note that the input and output data areas are scanned twice per second, i.e. to ensure that an event is detected by the gateway, it must be present longer than 0.5 seconds.

C.7.2 E-mail Definitions

The e-mail definitions are stored in the following two directories:

- ‘\user\email’
This directory holds up to 10 messages which can be altered by normal level FTP users.
- ‘\email’
This directory holds up to 10 messages which can be altered by admin level FTP users.

E-mail definition files must be named 'email_1.cfg', 'email_2.cfg'... 'email_10.cfg' in order to be properly recognized by the gateway.

File Format:

```
[Register]
Area, Offset, Type

[Register Match]
Value, Mask, Operand

[To]
recipient

[From]
sender

[Subject]
subject line

[Headers]
Optional extra headers

[Message]
message body
```

| Key | Value | Scanned for SSI |
|---------|--|-----------------|
| Area | Source area. Possible values: 'IN' (Input Data area) or 'OUT' (Output Data area) | No |
| Offset | Source offset, written in decimal or hexadecimal. | |
| Type | Source data type. Possible values are 'byte', 'word', and 'long' | |
| Value | Used as a reference value for comparison. | |
| Mask | Mask value, applied on the trigger source prior to comparison (logical AND). | |
| Operand | Possible values are '<', '=', or '>' | |
| To | E-mail recipient | Yes |
| From | Sender e-mail address | |
| Subject | E-mail subject. One line only. | |
| Headers | Optional; may be used to provide additional headers. | |
| Message | The actual message. | |

Note: Hexadecimal values must be written with the prefix '0x' in order to be recognized by the Anybus Communicator.

Copyright Notices

This product includes software developed by Carnegie Mellon, the Massachusetts Institute of Technology, the University of California, and RSA Data Security:

Copyright 1986 by Carnegie Mellon.

Copyright 1983,1984,1985 by the Massachusetts Institute of Technology

Copyright (c) 1988 Stephen Deering.

Copyright (c) 1982, 1985, 1986, 1992, 1993

The Regents of the University of California. All rights reserved.

This code is derived from software contributed to Berkeley by Stephen Deering of Stanford University.

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 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.
- Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS 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 REGENTS 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.

Copyright (C) 1990-2, RSA Data Security, Inc. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD4 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD4 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.

Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.