12005 Technology Drive
Eden Prairie, MN 55344-3620 USA

Application Guide

Using a Hilscher PKV 30-DNS DeviceNet™ to Modbus RTU Protocol Converter for Communication with Modbus RTU Compatible Servo Drives

Originator: Thomas J Schrauth
(612) 995-8168

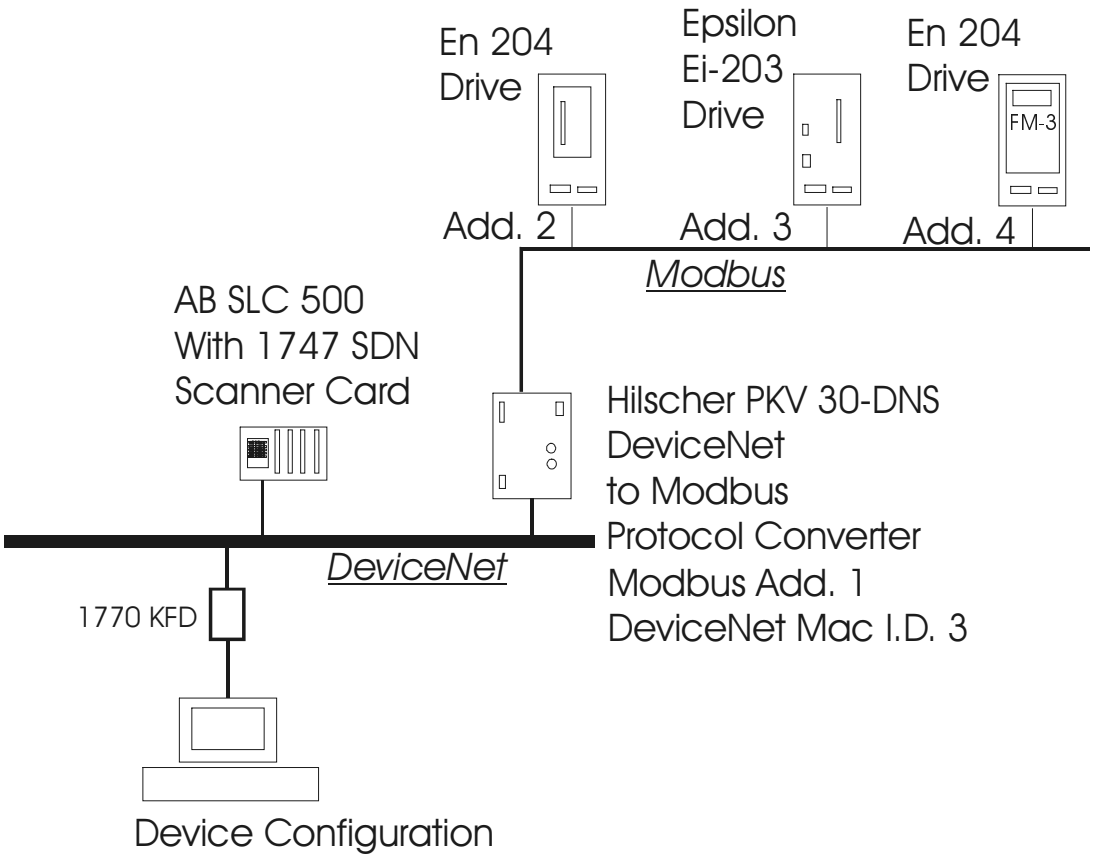
Emerson EMC
Date: 7-6-00

Revision Level: 01

Table of Contents

Application Overview:

This Application Guide describes the setup and implementation of connecting a DeviceNet network with Modbus RTU compatible products through the use of a protocol converter. The DeviceNet system described will consist of an Allen Bradley SLC500 with DeviceNet scanner card 1747-SDN, a Hilscher PKV 30-DNS DeviceNet to Modbus protocol converter, an En204, an Epsilon Ei-203, and an En 204 with a FM-3.



Equipment Used:

- Hilscher PKV 30-DNS protocol converter (Modbus address #1)

Dimensions:
105 X 105 X 80 mm
Din Rail Mountable
24VDC Input

- En 204 Digital Drive with MG 316 Motor (Modbus address #2)
- Epsilon Ei-203 Drive with a NT 212 Motor (Modbus address #3)
- En 204 Digital Drive with a Function Module #3 and a MG 316 Motor (Modbus address #4)
- Allen Bradley SLC 500 PLC; 1747-L542, 4 Slot Chassis, P1 Power Supply, 1747 SDN Scanner Card
- Qnt: 2 Emerson DDS-001 Multi-drop drive to drive serial cable (daisy chain cables)
- 24 VDC Power Supply for DeviceNet Network
- 1770 KFD RS 232 to DeviceNet Interface module with 96881501 RS232 cable (included)
- DeviceNet thin cable with four open style connectors
- RS Logix 500 Industrial Programming Software (PLC)
- Rs NetworX for DeviceNet Programming Software (Scanner)
- COMPRO Hilscher PKV 30-DNS Programming Software

Configuring Emerson Drives

This Application Example will involve three Emerson Motion Control Drives. Data transfer will involve most Modbus functions. Because the PLC communicates with the drives via DeviceNet, any reference to the PLC implies the PLC / 1747 SDN scanner card.

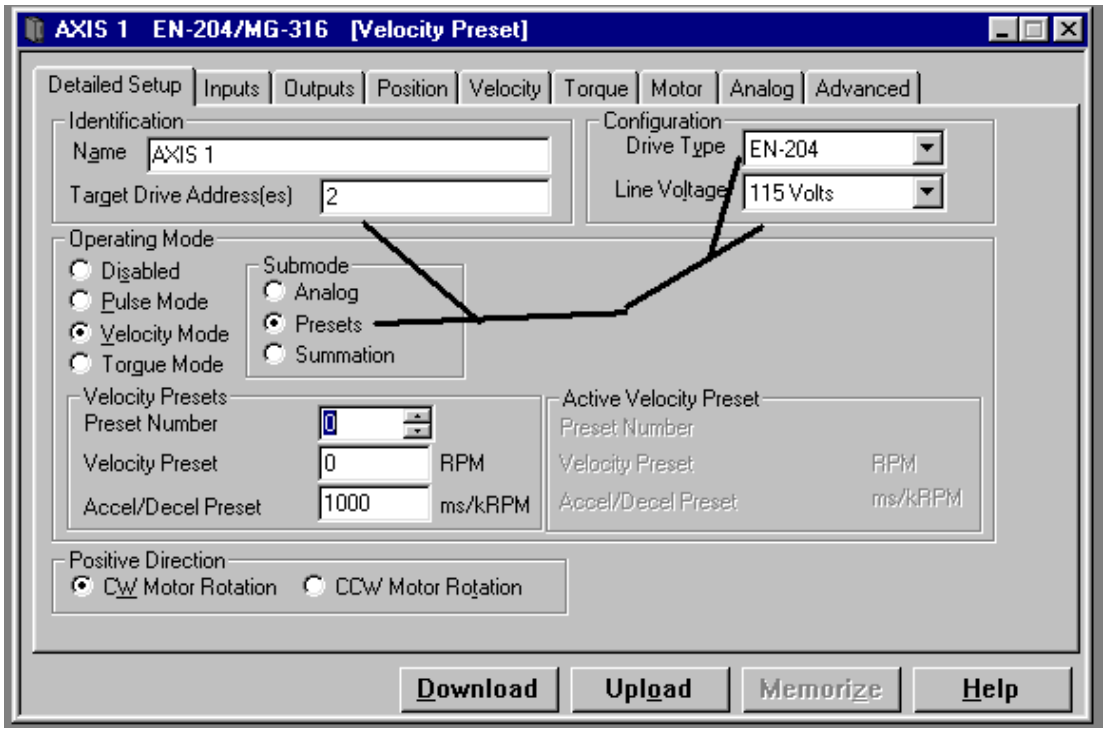
En 204 (Modbus Address #2)

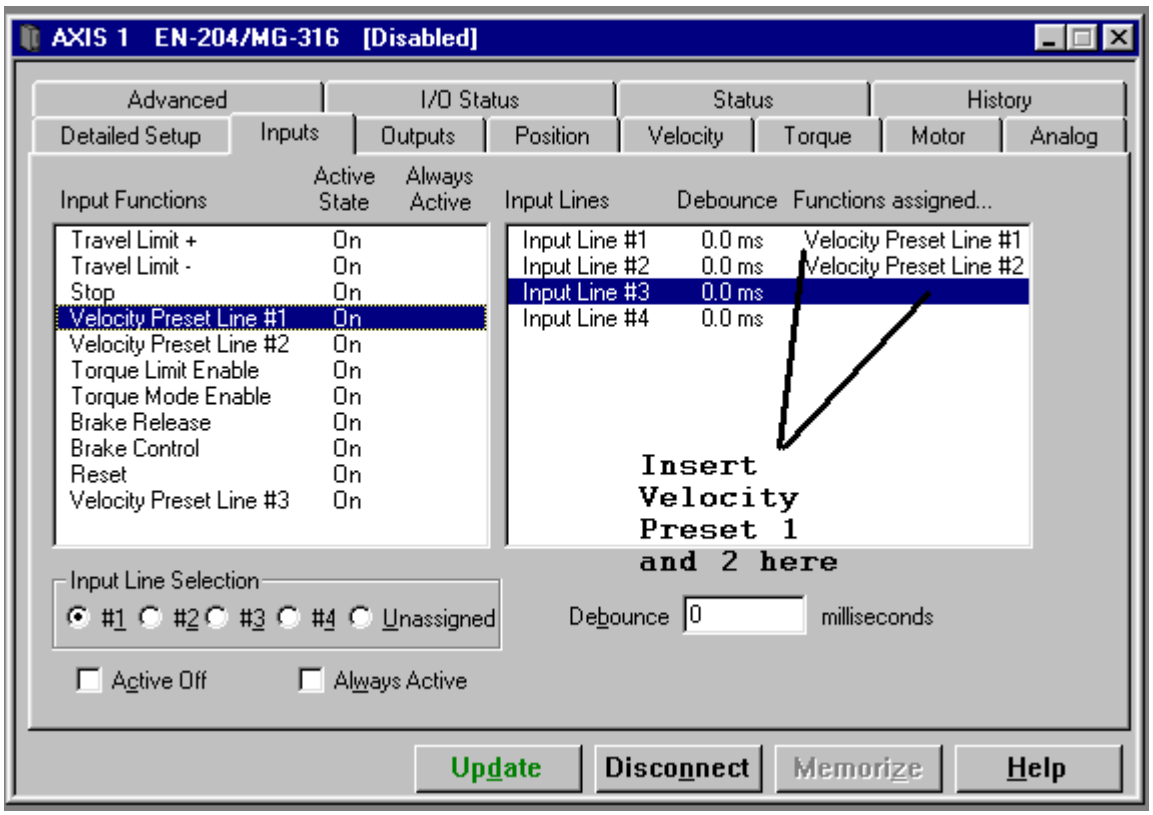
Drive #2 En 204 will be set up so the PLC will be able to force inputs to initiate a Preset Velocity. In addition to this a monitor of Velocity, Position, and Following Error Feedback will be implemented as well as a bit defined in the PLC to clear any faults on the drive. Velocity Preset #1 and Velocity Preset #2 will be user definable in the PLC.

- Create a new analog velocity setup for the En 204.
    - Making sure that the Drive type and line voltage match the items in use, set the target address for 2 and click the radio button for Presets.

- Set Input Line #1 and Line #2 for Velocity Preset Line #1 and #2.

Application Example:





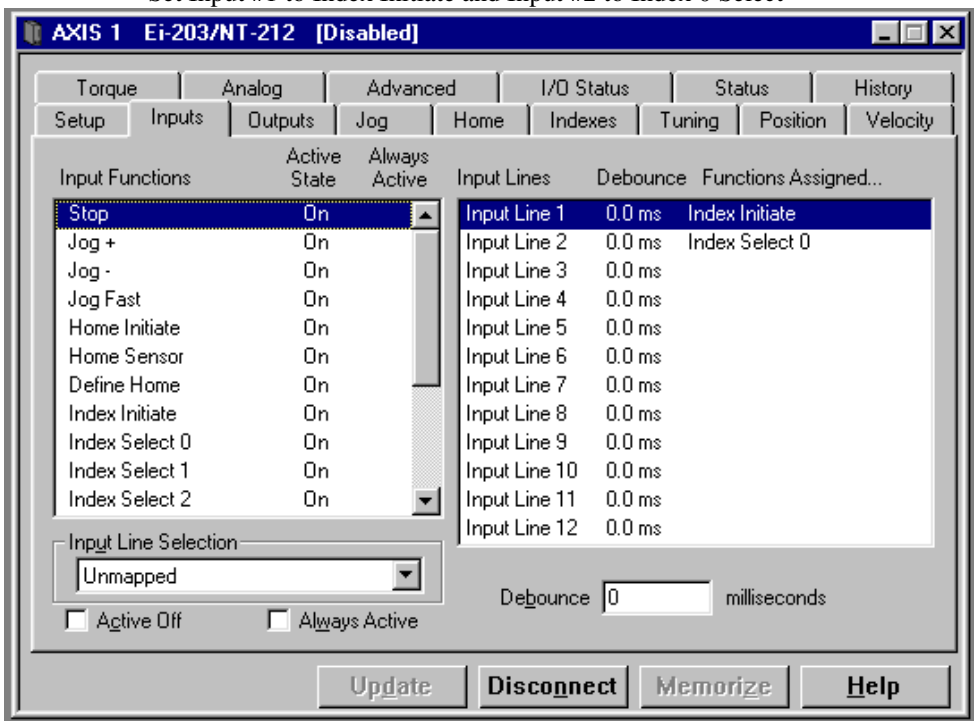
- Save and Download this File.

Epsilon Ei-203 (Modbus Address #3)

Drive #3 Epsilon Ei-203 will be configured to force inputs to initiate Index 0 as well as Index Select 0. Because of the drive flexibility, the rest of the parameters related with Index 0 will be changed without having to define them. Example; Index 0 Distance, Index 0 Velocity, Clear Faults, and reading the Position Command.

Create a new configuration for the Ei-203 as follows:

- Set Input #1 to Index Initiate and Input #2 to Index 0 Select

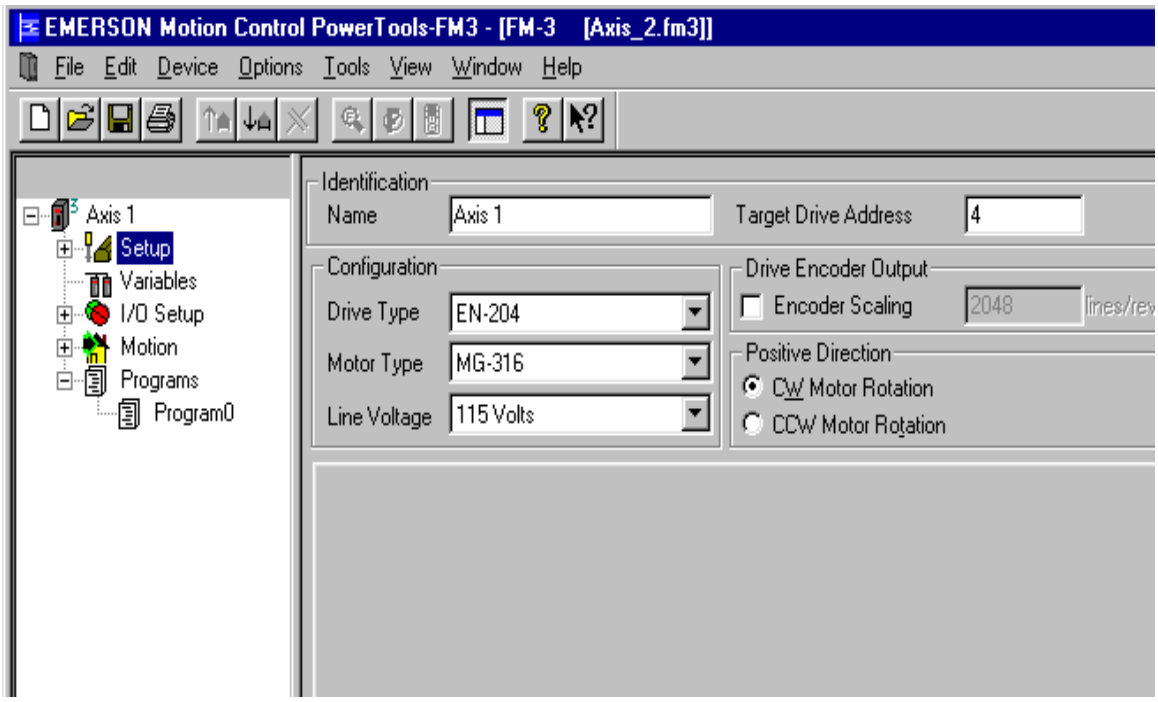


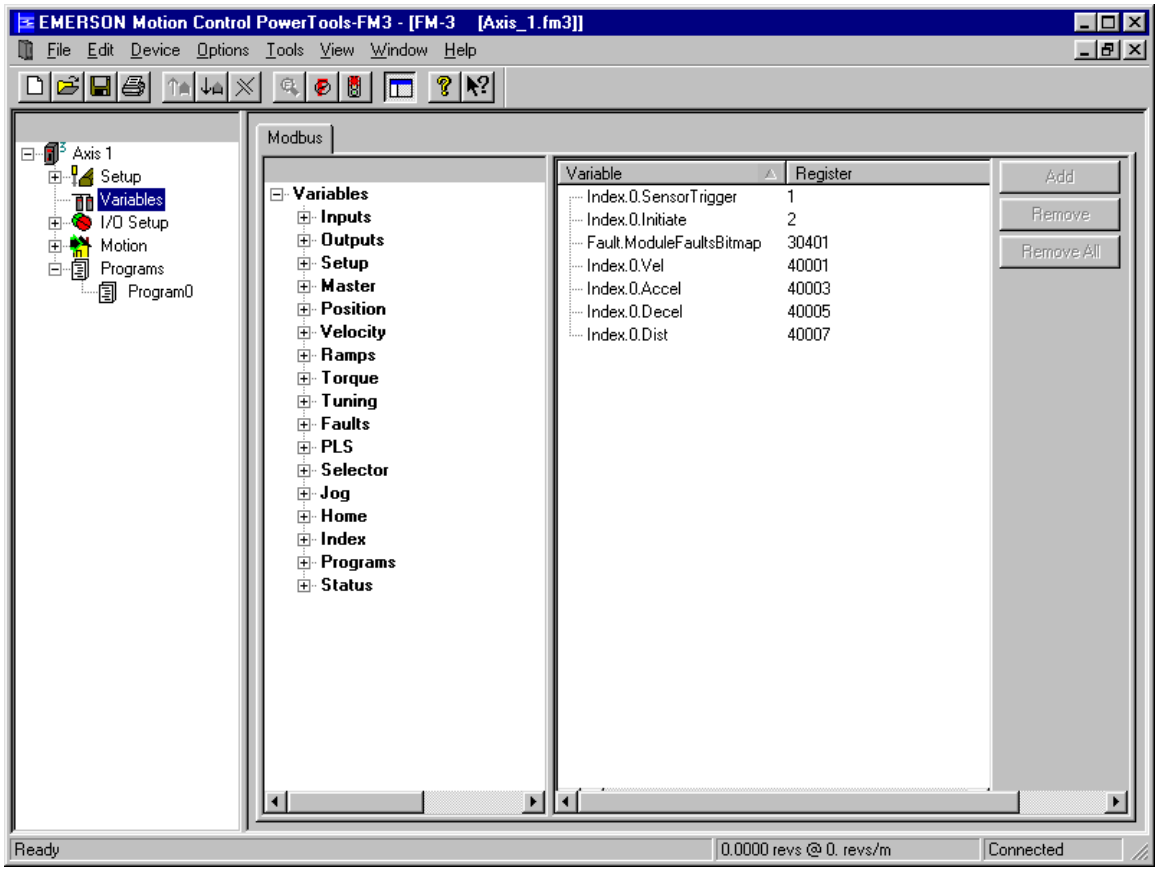
En 204 with FM 3 (Modbus address #4)

Drive #4 En 204 with Function Module #3 has Modbus addresses that are predefineable, therefore Powertools FM3 must be used to set these Modbus addresses. The PLC will be in charge of setting an Index up with an Accel, Decel, Distance, Index Initiate, and Index Sensor. In addition to this, the PLC will read in the Fault Status Bitmap.

Create a new configuration in Powertools FM-3 as follows:

- Set the Target Drive Address for 4
- Set the motor type and line voltage
- Using the variables screen, set the following Modbus addresses
  - Index 0 sensor trigger Mod Address 1
  - Index 0 Initiate Mod Address 2
  - Fault Module Bitmap Mod Address 30001
  - Index 0 Accel Mod Address 40003
  - Index 0 Decel Mod Address 40005
  - Index 0 Distance Mod Address 40007





- Save and Download Project

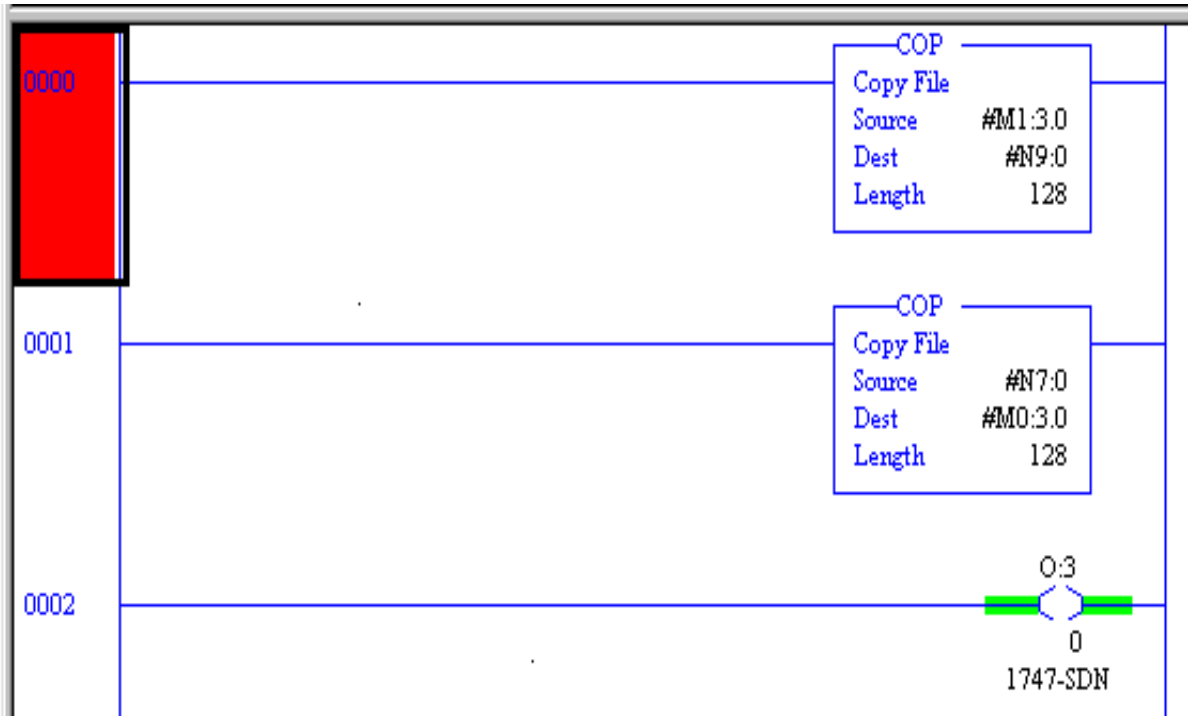
PLC Configuration

1. Configure RS-Linx for the PLC setup and then start RS-Logix.
2. In order to see the M0 and M1 Explicit Messaging files of the scanner card a copy command must be issued in the ladder logic to an integer table. Following is an example of a copy for the inputs and outputs of the scanner card respectively:
3. In addition (O: "SLOT". 0) for the scanner module must be set high to enable the scanner card.

.

Application Example:

Partial PLC Ladder Program



NOTE: *For an application example of the full program, refer to* ***Appendix B***

Data Format Transformation

1. When data is transferred through the Hilscher module, the lower and upper bytes of each word are exchanged.
2. The following chart designates how data is transformed in the conversion between Modbus RTU and DeviceNet.

(b15), (b14), (b13), (b12), (b11), (b10), (b9), (b8), (b7), (b6), (b5), (b4), (b3), (b2), (b1), (b0)
TRANSFORMS TO
(b7), (b6), (b5), (b4), (b3), (b2), (b1), (b0), (b15), (b14), (b13), (b12), (b11), (b10), (b9), (b8)

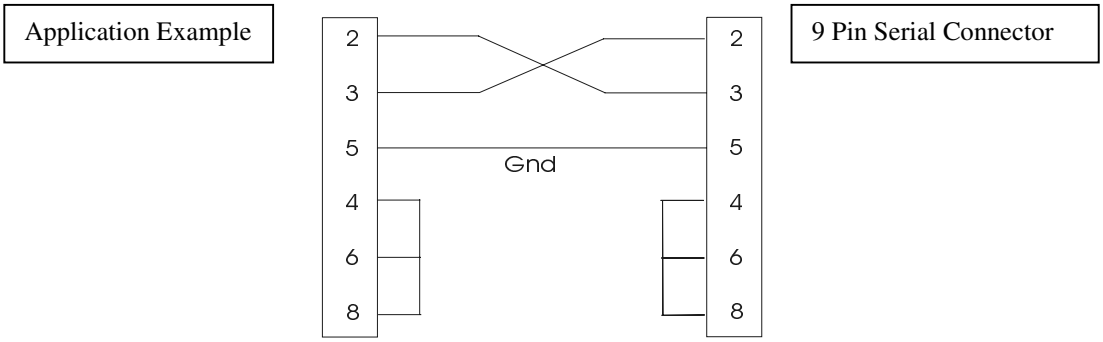
Modbus Data	Binary	DeviceNet Data	Binary
1000	OOOOOO11111O1OOO	-6141	111O1OOOOOOOOO11
5500	OOO1O1O1O11111OO	31765	O11111OOOOO1O1O1

NOTE: When setting individual bits or registers, the preceding chart should be used. I.E. Bit 8 DeviceNet = Bit 0 Modbus Register.

Setting up the Hilscher PKV 30-DNS DeviceNet to Modbus Protocol Converter

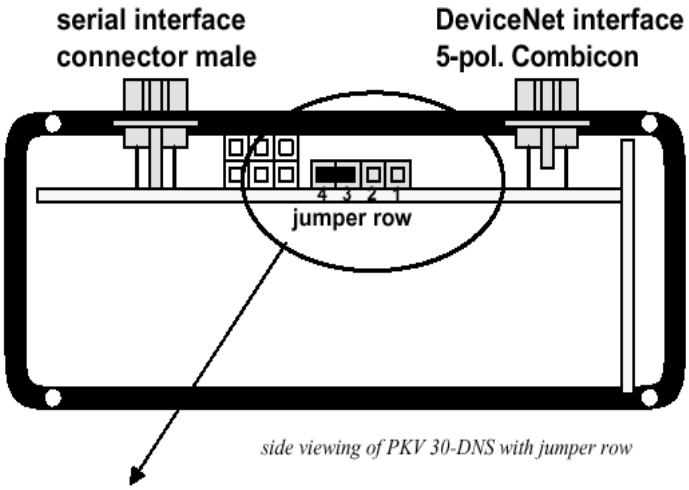
Commissioning Lead

A commissioning lead with the connections shown below is required to connect the PC communications port to the Hilscher module.



Internal Jumpers

The Hilscher PKV 30-DNS is shipped with jumpers which set the unit for RS422 serial interface. In order to change the jumper status, the lid of the PKV 30-DNS must be removed by means of four screws, and jumpers set as below.



side viewing of PKV 30-DNS with jumper row

selection	jumper J5	Interfacetyp
	open	RS232
	3-4	RS485
	1-2, 3-4	RS422

4 3 2 1

Selection of serial interface type

Important:

The PKV 30 DNS can be programmed with jumpers in any position using COMPRO. This could lead the user to believe that the serial communication is set up correctly on the Modbus side. When using a new unit, *always* check jumper configuration.

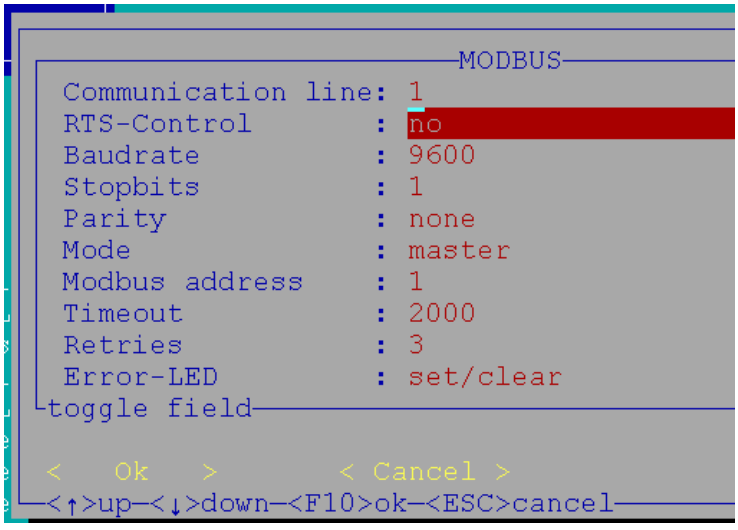
Configuration Mode

1. Ensure the converter module is switched off.
2. Connect the "Diagnose X3" port to the PC COM port using the commissioning lead.
3. If PC COM port is set to anything except COM 1, enter DOS and edit CPRUN.BAT to read "S(com port #)". Ex. "S1" "S2" "S3"
4. Enter Windows Explorer and double click on CPRUN.BAT.
5. Press Return
6. If running COMPRO for the first time select Dnsmbr, otherwise select the appropriate project.
7. Select ONLINE, SYSTEM and BOOTSTART. Select OK, and the message "**wait for hardware receipt**" will be displayed.
8. Switch the converter on and the message should disappear. The module is now in configuration mode.

Modbus Network Settings

1. Select DATABASE, EDIT and MODBUS. This brings up the configuration network for the Modbus network.
2. To change settings, move the cursor bar to highlight the setting, and press the SPACE BAR until the desired setting appears.
3. Set the stop bit to "1", and the baud rate and parity as required. (Usually, this will be 9600 or 19200 bits/sec, no parity, 8bits, but this may depend on the other devices to be connected to the network.)
4. Set the mode to MASTER.
5. Press ESCAPE and select OK to save and exit.

Application Example:



Configure the Modbus Network Command List

1. Every node on the Modbus network MUST be assigned a unique node address before connecting to the Modbus network.
2. Select DATABASE, EDIT and COMMAND. This brings up the Modbus network Communications configuration window.
3. For each transfer set the following information. Use the "+" and "-" keys on the **NUMERIC KEYPAD** to add and remove lines as required.
   a. SLAVE target node for the command
   b. FUNCTION specifies READ or WRITE command.
      1. Read Coil Status
      2. Read Input Status
      3. Read Holding Registers
      4. Read Input Registers
      5. Force Single Coil
      6. Preset Single Register
      15. Force Multiple Coils
      16. Preset Multiple Registers
   c. ADDRESS specifies the source/destination register in the target slave node
   d. QUANTITY number of consecutive registers to read or write
4. Press Escape and select OK to save any changes made.

Application Example:

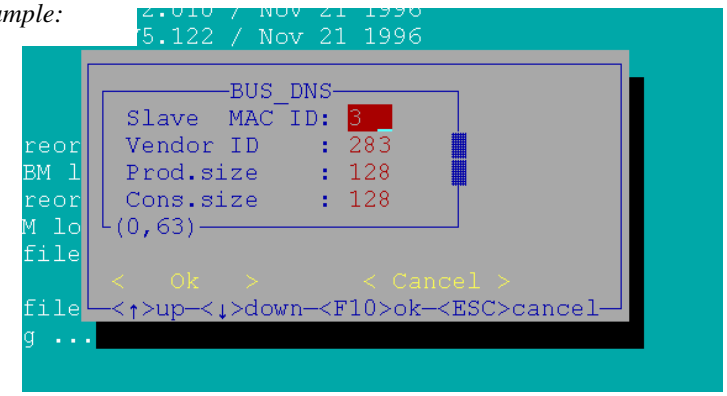
SLAVE	FUNC	ADDRESS	QUANT	MEM ADD	WRITE	*Function*
		(Modbus)		(Hilscher)		
2	15	2	29	1	CYCLIC	*Input Force Command and Enable*
2	5	1007	1	30	CYCLIC	*Clear Faults*
2	4	32021	2	30001	CYCLIC	*Write Velocity Feedback*
2	4	32026	2	30003	CYCLIC	*Position Feedback*
2	4	32028	2	30005	CYCLIC	*Following Error Feedback*
2	16	41105	2	40007	CYCLIC	*Vel Preset #1*
2	16	41109	2	40009	CYCLIC	*Vel Preset #2*
3	15	2	29	32	CYCLIC	*Input Force Command and Enable*
3	5	1007	1	61	CYCLIC	*Clear Faults*
3	4	32036	2	30007	CYCLIC	*Position Command*
3	4	32021	2	30009	CYCLIC	*Velocity Feedback*
3	6	43001	1	40012	CYCLIC	*Index Type 0*
3	6	43002	1	40013	CYCLIC	*Index 0 Dist*
3	16	43004	2	40015	CYCLIC	*Index 0 Vel*
4	5	2	1	63	CYCLIC	*Index 0 Initiate*
4	5	1	1	62	CYCLIC	*Jog 0 Select*
4	16	40001	2	40017	CYCLIC	*Index 0 Vel*
4	5	3	1	80	CYCLIC	*Fault Reset*
4	16	40003	2	40019	CYCLIC	*Index 0 Accel*
4	16	40005	2	40021	CYCLIC	*Index 0 Decel*
4	16	40007	2	40023	CYCLIC	*Index 0 Dist*
4	2	10001	1	10336	CYCLIC	*Read Ind 0 @ Vel*
4	4	30401	2	30012	CYCLIC	*Fault Status bit Map*

Gray area does not appear in Software Configuration.

Configure PKV 30-DNS as the DeviceNet Slave

1. Select DATABASE, EDIT and BUS_DNS. This brings up the DeviceNet slave configuration window.
2. Highlight a line and press SPACE BAR until required data format appears.
3. Choose a unique MAC I.D. number for the Hilscher between (1-63). Using the rotary switches on the Hilscher module, dial in the same MAC I.D.
4. Choose number of bytes in and out of DeviceNet under "CONS" and "PROD"

Application Example:



Important:

The .eds file which is stored on the floppy in the same directory as CPRUN tells the DeviceNet MASTER the exact settings of the slave. The Hilscher's .eds file is shipped at 8 bytes in and 8 bytes out. If anything except 8 bytes "CONS" and 8 bytes "PROD" is set when configuring the DeviceNet slave, the .eds file must be edited and changed before attempting to place it into RS Networx.

Save and Download the Project

1. Select DATABASE, SAVE, and NEW. Specify a new name for the project and press ENTER.
2. Select ONLINE, DATABASE, and DOWNLOAD, and the configured project file will be downloaded to the module.
   a. If error 44 "Connection could not be established" occurs, check cables and bootstart the system again. Then repeat DOWNLOAD.
3. When download is complete, switch power to the module off, disconnect the commissioning lead, and connect the Modbus network to the DIAGNOSE X3 port. Turn power to the module on, and if the Modbus network configuration matches the Hilscher module configuration, the ERR2 LED will turn solid green. If the DeviceNet network is already configured and working correctly, the ERR1 LED will also turn solid green and the RDY and RUN LEDs will be solid on.

NOTE: *The Hilscher PKV 30-DNS module will not run until the DeviceNet network has been correctly configured.*

FOR MORE INFORMATION ON THE HILSCHER PKV 30-DNS CONSULT USER GUIDES APPENDIX C, D, AND E.

Setting up the DeviceNet Master

Uploading System

1. Attach all DeviceNet nodes and turn system power on.
2. With 1770-KFD attached to the system and to the active COM port start RS-Linx and go to COMMUNICATION and CONFIGURE DRIVERS.
3. Add a new DeviceNet Driver. Configure it for the correct serial port settings for computer communications and give the 1770-KFD a unique node address then set the communications rate at 125k to start with.

Note:

All devices must be set up on the DeviceNet side to communicate at the same baud rate. It is recommended for the ease of startup that communications be set at the lowest possible baud rate to start out (125 k). In most applications this rate is sufficient. After the network is functional the user may implement a higher baud rate if needed.

4. Open RS-NetWorx and go to FILE and NEW to open a new RS-NetWorx file.
5. Select NETWORK and ONLINE.
6. After uploading the system, select NETWORK and UPLOAD FROM NETWORK.
7. WAIT FOR THE FUNCTION TO UPLOAD ALL 64 NODES.

Installing .eds Files

1. Select TOOLS and EDS WIZARD. Follow the instructions and insert the correct file from the files that came with the PKV 30-DNS
2. If RS-NetWorx does not allow an install of the .eds file, then edit the file with Notepad to make it work. An error message should direct the editor to the .eds mistake.

Note:

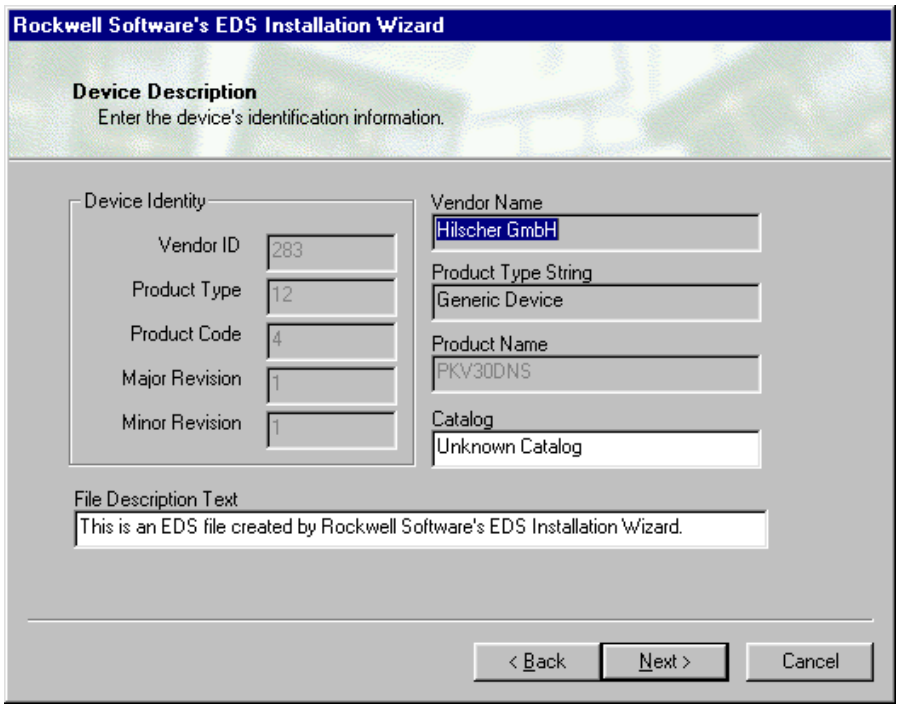
Remember to change the .eds to reflect the amount of data produced and consumed by the Hilscher. This editing can be accomplished using Notepad.

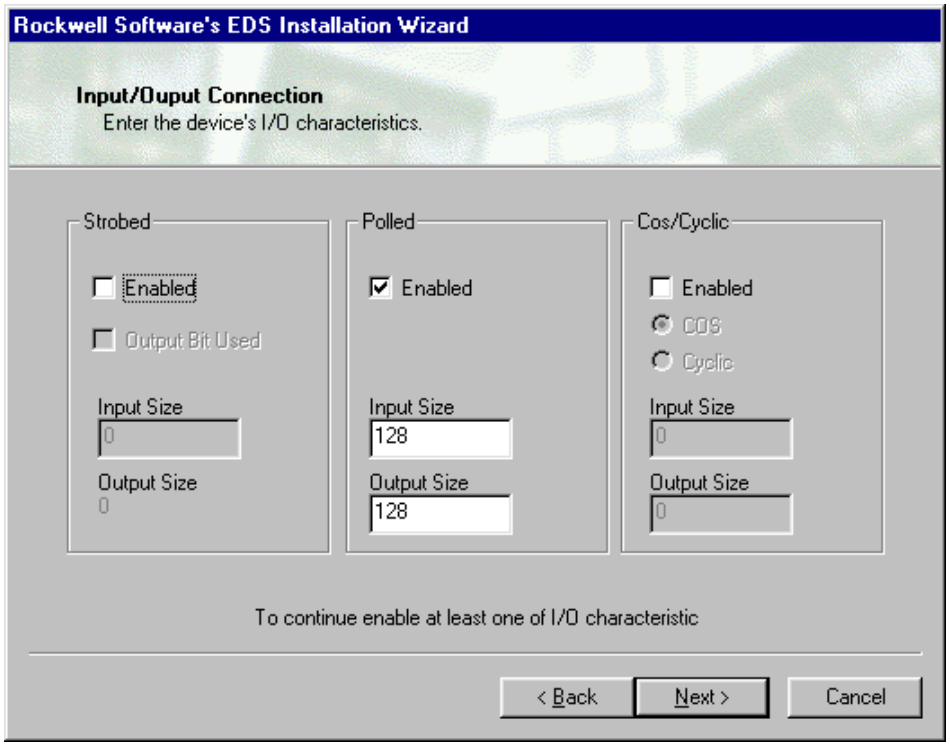
3. After completion the Hilscher PKV 30-DNS should have an icon on the main screen of RS-NetWorx.

Alternate Installation for .eds Files

1. Select TOOLS and EDS WIZARD.
2. Click the box for CREATE EDS STUB.
3. Enter in the correct package size for data in and out.

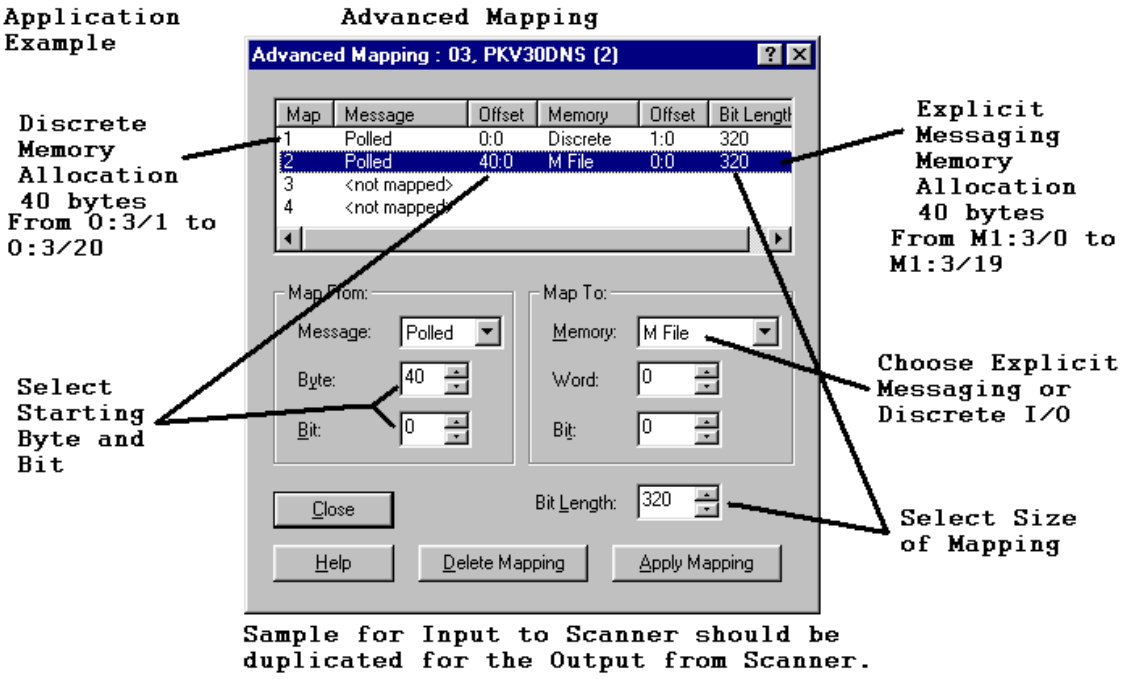
Application Example:





Commissioning the Scanner Card

1. Select VIEW and GRAPH, a scanner card, a globe (representing the Hilscher), and the 1770-KFD should be seen. Double click on the scanner card.
2. Under the GENERAL tab, the scanner module should be set to (node) address 0. Click on APPLY then download these changes to the PLC.
3. Under the MODULE tab, the slot that the scanner card is in should be set. After it is set, click on APPLY and download this change to the processor.
4. Select the INPUT tab and choose the ADVANCED button.
5. For each mapping number (1 – 4) choose the address and number of bytes mapped in that location as shown in the example below.
6. APPLY MAPPING and close the window.
7. Select the OUTPUT tab and choose the ADVANCED button.
8. For each mapping number (1 – 4) choose the address and number of bytes mapped in that location as shown in the example below.
9. Check the INPUT and OUTPUT tabs to make sure that the data has been mapped.
10. Click O.K. to get to the main screen and select, NETWORK and DOWNLOAD TO NETWORK.



Mapping Data to the PLC

The following is an example of where data will be mapped in this application:

SLAVE	FUNC	ADDRESS (Modbus)	QUANT	MEM ADD (Hilscher)	WRITE	*Function*	Variable
2	15	2	29	1	CYCLIC	*Input Force Command and Enable*	A
2	5	1007	1	30	CYCLIC	*Clear Faults*	B
2	4	32021	2	30001	CYCLIC	*Write Velocity Feedback*	C
2	4	32026	2	30003	CYCLIC	*Position Feedback*	D
2	4	32028	2	30005	CYCLIC	*Following Error Feedback*	E
2	16	41105	2	40007	CYCLIC	*Vel Preset #1*	F
2	16	41109	2	40009	CYCLIC	*Vel Preset #2*	G
3	15	2	29	32	CYCLIC	*Input Force Command and Enable*	H
3	5	1007	1	61	CYCLIC	*Clear Faults*	I
3	4	32036	2	30007	CYCLIC	*Position Command*	J
3	4	32021	2	30009	CYCLIC	*Velocity Feedback*	K
3	6	43001	1	40012	CYCLIC	*Index Type 0*	L
3	6	43002	1	40013	CYCLIC	*Index 0 Dist*	M
3	16	43004	2	40015	CYCLIC	*Index 0 Vel*	N
4	5	2	1	63	CYCLIC	*Index 0 Initiate*	O
4	5	1	1	62	CYCLIC	*Jog 0 Select*	P
4	16	40001	2	40017	CYCLIC	*Index 0 Vel*	Q
4	5	3	1	80	CYCLIC	*Fault Reset*	R
4	16	40003	2	40019	CYCLIC	*Index 0 Accel*	S
4	16	40005	2	40021	CYCLIC	*Index 0 Decel*	T
4	16	40007	2	40023	CYCLIC	*Index 0 Dist*	U
4	2	10001	1	10336	CYCLIC	*Read Ind 0 @ Vel*	V
4	4	30401	2	30012	CYCLIC	*Fault Status bit Map*	W

PLC READ DATA MAPPING CONFIGURATION

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I:3.1	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
I:3.2	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
I:3.3	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
I:3.4	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
I:3.5	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
I:3.6	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
I:3.7	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J
I:3.8	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J
I:3.9	K	K	K	K	K	K	K	K	K	K	K	K	K	K	K	K
I:3.10	K	K	K	K	K	K	K	K	K	K	K	K	K	K	K	K
I:3.11									V							
I:3.12	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
I:3.13	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
I:3.14																
I:3.15																
I:3.16																
I:3.17																
I:3.18																
I:3.19																
I:3.20																
MI:3.0																
MI:3.1																
MI:3.2																
MI:3.3																

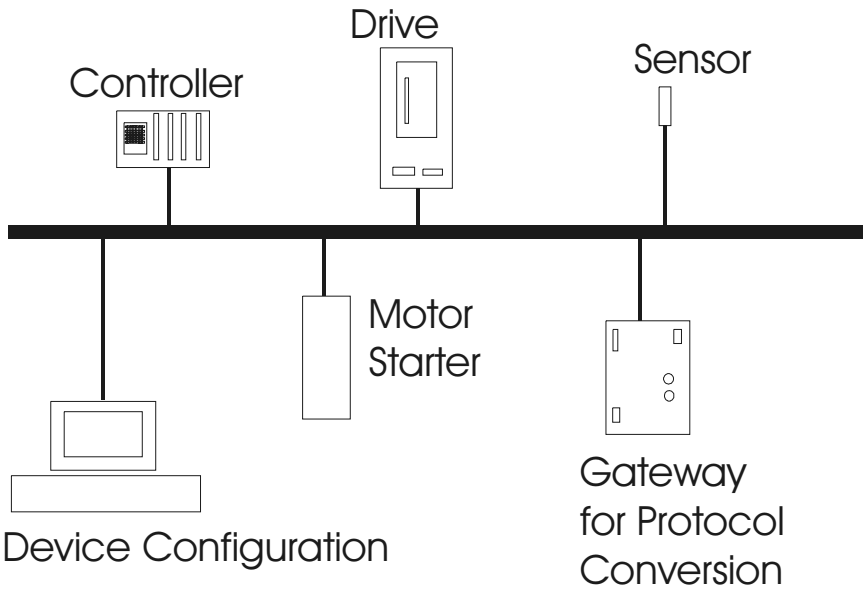
PLC WRITE DATA MAPPING CONFIGURATION

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
O:3.1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
O:3.2	A	A	A	A	A	A	A	A	H		B	A	A	A	A	A
O:3.3	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
O:3.4	H	H	H	H	H	H	H	H		O	P	I	H	H	H	H
O:3.5									R							
O:3.6																
O:3.7	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
O:3.8	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
O:3.9	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
O:3.10	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
O:3.11																
O:3.12	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
O:3.13	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
O:3.14	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
O:3.15	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
O:3.16	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
O:3.17	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
O:3.18	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
O:3.19	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
O:3.20	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
MO:3.0	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
MO:3.1	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
MO:3.2	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
MO:3.3	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Appendix A: What is DeviceNet?

DeviceNet is a low-level network that provides connections between simple industrial devices (sensors, actuators) and higher-level devices (controllers). DeviceNet standards and specifications are managed by ODVA (Open DeviceNet Vendors Assoc.) which is an independent supplier organization that manages the DeviceNet specification and supports the worldwide growth of DeviceNet.

Example DeviceNet Communications Link



DeviceNet has two primary purposes.

1. Transport of control-orientated information associated with low-level devices.
2. Transport of other information that is indirectly related to the system being controlled, such as configuration parameters.

The list below presents a summary of the Physical/Media specific characteristics of DeviceNet:

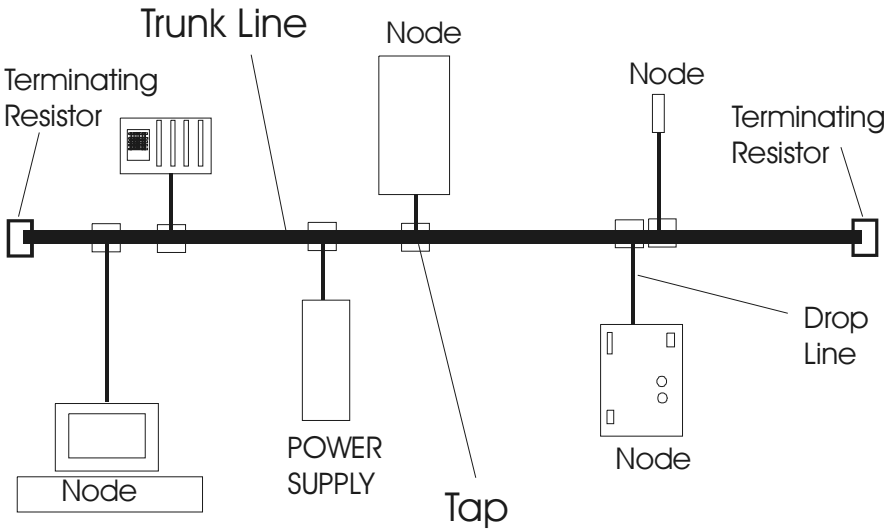
1. Trunk-line / drop-line configuration
2. Support for up to 64 nodes
3. Node removal without severing the network
4. Simultaneous support for both network-powered (sensors) and self-powered (actuators) devices.
5. Use of sealed or open-style connectors
6. Protection from wiring errors
7. Selectable data rates of 125k baud, 250k baud, and 500k baud
8. Adjustable power configuration to meet individual application needs
9. High current capability (up to 16 amps per supply)
10. Operation with off-the-shelf power supplies
11. Power taps that allow the connection of several power supplies from multiple vendors that comply with DeviceNet standards
12. Built-in overload protection
13. Power available along the bus: both signal and power lines contained in the trunk line

Appendix A

DeviceNet Hardware Components

The following components are necessary to design a DeviceNet cable system:

- Cables
  - -- Trunk line
  - -- Drop line
- Nodes/devices
- Connectors
- Power supplies
- Terminating resistors



Cabling

Specific cable is necessary for a DeviceNet cable system. The following cable types can be used:

Cable Type:

Round (thick)

- Typically used for trunk line
- Outside diameter of 12.2 mm (0.48 in.)
- 5 wire cable

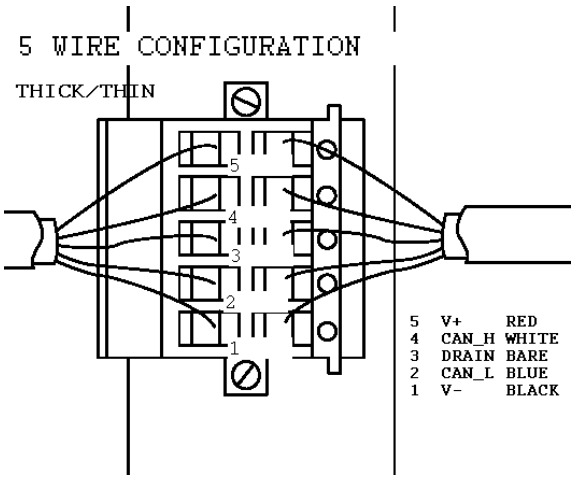
Round (thin)

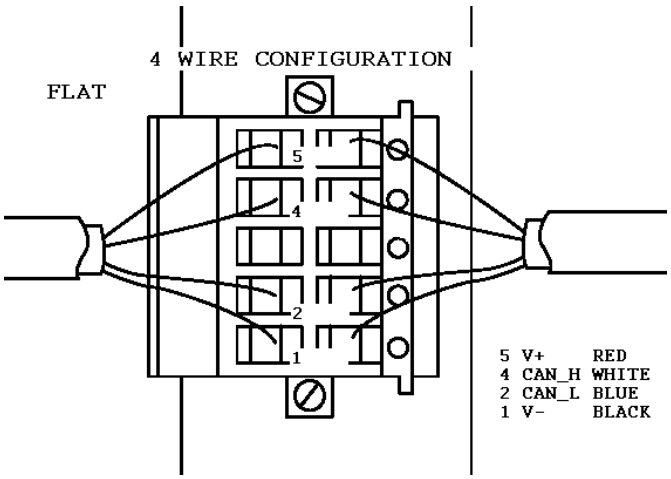
- Typically used for drop line
- More flexible than thick cable
- Outside diameter of 6.9 mm (0.27 in.)
- 5 wire cable

Appendix A

Flat

- Typically used for trunk line when flexible cable or modular design is needed
- No predetermined cable lengths
- Can have connections anywhere on cable
- 4 wire cable (NO SHIELD)





Trunk Line:

The trunk line connects all nodes to each other. The trunk line configuration must meet the following requirements:

- No more than 4A of current at any point on the trunk
- Maximum distance requirements based on the data rate of the network

The distance between any two points on a DeviceNet network cannot exceed the maximum cable distance for the data rate used.

Data Rate	Maximum Distance (Flat Cable)	Maximum Distance (Thick Cable)
125k bit/s	420m (1378 ft)	500m (1640 ft)
250k bit/s	200m (656 ft)	250m (820 ft)
500k bit/s	75m (246 ft)	100m (328 ft)

Drop Line:

Thin cable is typically used to create a drop line. The drop line is responsible for connecting nodes on the network to the trunk line. Drop line specifications include:

- 3 A of Current capability
- Length requirements based on the data rate of the network

The cumulative drop line length refers to the sum of all drop lines in the cable system. This sum cannot exceed the maximum cumulative length allowed for the data rate used.

Appendix A

Data Rate	Cumulative Drop Line Rate
125k bit/s	156m (512 ft)
250k bit/s	78m (256 ft)

The maximum cable distance from any device on a branching drop line to the trunk line is 6 meters (20 feet).

Nodes/Devices

A DeviceNet node is any device that is addressable through DeviceNet and contains DeviceNet communications circuitry. Nodes must comply with the following rules:

- A node must be connected to the network by a tap and drop-line.
- Nodes must be DeviceNet-compatible devices.
- 64 nodes can be supported on one network.
- Each node must be assigned a number. (MAC I.D.)
- Node numbers cannot be duplicated

Connectors

Connectors are used for attaching cables to devices. Connectors can be either Open-style (wires exposed), or Sealed.

Power Requirements

- The power supply must have its own current limit protection.
- Fuse protection must be provided for each segment of the cable system.
- The power supply must be correctly sized to provide each device with its required power.
- Power supplies should be distributed throughout the DeviceNet network to maintain a maximum of 4 amps per trunk branch.

Terminating Resistors

Terminating resistors are used to reduce the reflection of signals over the network. They must be 121Ω and installed on both ends of the network.

DeviceNet Messaging and Communications

MASTER/SLAVE RELATIONSHIP

ODVA defines a DeviceNet Master as:

> The device that gathers and distributes I/O data for the process controller. A Master scans its Slave devices based on a scan list it contains. With respect to the network, the Master is a Group 2 Client or a Group 2 Only Client.

ODVA defines a DeviceNet Slave as:

> A Slave returns I/O data to its Master when it is scanned. With respect to the network, the Slave is a Group 2 Server or a Group 2 Only Server.

Appendix A

ODVA defines Predefined Master/Slave Connection Set as:

> A set of Connections that facilitate communications typically seen in a Master/Slave relationship. Many of the steps involved in the creation and configuration of an Application to Application connection have been removed within the Predefined master/Slave Connection Set definition. This, in turn, presents the means by which a communication environment can be established using less network and device resources.

Rules governing the Master/Slave relationship

- Each Slave may report to only 1 Master.
- The Master of one sub-network may be Slave of a different sub-network.
- The MAC I.D. of every device (all sub-networks) must be unique.

DEVICENET GROUP 2 PREDEFINED MESSAGE CONNECTIONS

Connection Objects *MODEL THE COMMUNICATION* characteristics of a particular Application-to-Application relationship. DeviceNet supports two types of connections:

- I/O Connections

> I/O connections provide dedicated, special-purpose communication paths between a producing application and one or more consuming applications. Application-specific I/O data moves through these ports. I/O messages can fall into three categories.
>
> 1. I/O Bit-Strobe
> 2. I/O Poll
> 3. I/O Change of State

I/O Bit-Strobe:

The Bit-Strobe command is an I/O message that is transmitted by the Master. A Bit-Strobe Command Message has multi-cast capabilities. Multiple Slaves can receive and react to the same Bit-Strobe command (multi-cast capabilities). The bit-Strobe Response is an I/O Message that a Slave transmits back to the Master when the Bit-Strobe Command is received. Within a slave, the bit-Strobe Command and Response Messages are received/transmitted by a single connection Object.

I/O Poll:

The Poll Command is an I/O Message that is transmitted by the Master. A Poll Command is directed towards a single, specific Slave (point-to-point). A Master must transmit a separate Poll command Message for each one of its Slaves that is to be polled. The Poll Response is an I/O Message that a Slave transmits back to the Master when the Poll Command is received. Within a Slave, the Poll command and Response Messages are received/transmitted by a single Connection Object.

I/O Change of State:

The Change of State/Cyclic Message is transmitted by either the Master or the Slave. A change of State/Cyclic Message is directed towards a single specific node (point-to-point). An Acknowledge Message may be returned in response to this message. Within either the Master or the Slave, the producing Change of State Message and consuming Acknowledge Message are received/transmitted by a second connection object.

Appendix A

- Explicit Messaging Connections

Explicit Messaging Connections provide generic, multi-purpose communication paths between two devices. Explicit Messages provide the typical request/response-oriented network communications.

Explicit Response/Request Messages:

Explicit Request Messages are used to perform operations such as reading and writing attributes. Explicit Response Messages indicate the results of the attempt to service an Explicit Request Message. Within a Slave, Explicit Requests and Responses are received/transmitted by a single Connection Object.

For more information consult DeviceNet communication Model and Protocol Volume I and II.

INPUT FORCE COMMAND AND ENABLE BITS DRIVE #2

0000
N10:0
0
O:3
24
1747-SDN
O:3
40
1747-SDN

0001
N10:0
1
O:3
25
1747-SDN
O:3
41
1747-SDN

INPUT FORCE COMMAN AND ENABLE BITS DRIVE #2

0002
N10:0
0
N10:0
1
MOV
Move
Source 768
768<
Dest O:3.2
0<

Clear Faults on Drive 2

0003
N10:0
3
O:3
37
1747-SDN

Velocity Preset #1 set for 1000. Drive #2

0004
MOV
Move
Source -6141
-6141<
Dest O:3.8
-6141<

Velocity Preset #2 set for 2000. Drive #2

0005
MOV
Move
Source -12281
-12281<
Dest O:3.10
-12281<

INDEX TYPE DRIVE #3 (EPSILON)

0006
N10:0
0
MOV
Move
Source 0
0<
Dest O:3.12
0<

INDEX 0 DRIVE #3 (EPSILON) INDEX

0007

N10:0 0 | N10:0 1 | O:3 39 1747-SDN | O:3 55 1747-SDN

INDEX 0 DISTANCE MOST SIG WORD DRIVE #3 (EPSILON)

0008

MOV Move Source 1024 1024< Dest O:3.13 1024<

INDEX 0 DISTANCE LEAST SIG WORD DRIVE #3 (EPSILON)

0009

MOV Move Source 128 128< Dest O:3.14 128<

INDEX 0 VELOCITY MOST SIGNIFICANT WORD DRIVE #3 (EPSILON)

0010

MOV Move Source 0 0< Dest O:3.15 0<

INDEX 0 VELOCITY LEAST SIGNIFICANT WORD DRIVE #3 (EPSILON) SPEED 2000

0011

N10:0 0

MOV Move Source -12281 -12281< Dest O:3.16 -18421<

INDEX 0 VELOCITY LEAST SIGNIFICANT WORD DRIVE #3 (EPSILON) SPEED 3000

0012

N10:0 1

MOV Move Source -18421 -18421< Dest O:3.16 -18421<

INDEX 0 INITIATE DRIVE #4 (FM-3)

00

N10:0 0 | O:3 70 1747-SDN

JOG 0 SELECT
DRIVE #4 (FM-3)

0014

N10:0
5

O:3
69
1747-SDN

FAULT RESET
DRIVE #4 (FM-3)

0015

N10:0
6

O:3
87
1747-SDN

INDEX 0 VELOCITY MOST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0016

MOV
Move
Source 0
0<
Dest O:3.17
0<

INDEX 0 VELOCITY LEAST SIGNIFICANT WORD
DRIVE #4 (FM-3) SET AT 1000 RPM

0017

MOV
Move
Source -6141
-6141<
Dest O:3.18
-6141<

INDEX 0 ACCEL MOST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0018

MOV
Move
Source 0
0<
Dest O:3.19
0<

INDEX 0 ACCEL LEAST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0019

MOV
Move
Source -6141
-6141<
Dest O:3.20
-6141<

INDEX 0 DECEL MOST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0020

MOV
Move
Source 0
0<
Dest N7:0
0<

INDEX 0 DECEL LEAST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0021

MOV
Move
Source -6141
-6141<
Dest N7:1
-6141<

INDEX 0 DISTANCE MOST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0022

MOV
Move
Source 2048
2048<
Dest N7:2
2048<

INDEX 0 DISTANCE LEAST SIGNIFICANT WORD
DRIVE #4 (FM-3)

0023

MOV
Move
Source 128
128<
Dest N7:3
128<

0024

COP
Copy File
Source #M1:3.0
Dest #N9:0
Length 128

0025

COP
Copy File
Source #N7:0
Dest #M0:3.0
Length 128

0026

O:3
0
1747-SDN

0027

END

Operation instructions manual

ComPro

Project planning and diagnostic

DOS program

Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 78
D-65795 Hattersheim
Germany

Tel. +49 (0) 6190/9907-0
Fax. +49 (0) 6190/9907-50

Sales: +49 (0) 6190/9907-0
Hotline and support: +49 (0) 6190/9907-99

e-mail: hilscher@hilscher.com
Web: http://www.hilscher.com

Index	Date	Version	Chapter	Revision
1	1.12.93	1.00	all	draw up; based on V 1.00 of the German manual
6	13.09.95	2.12	all	draw up; based on V 2.11 of the German manual timestamp using with trace. Not compatible to previous versions of ComPro
7	21.02.96 13.06.96	from 2.20	all all	Improved system conditions Expanded menu ONLINE with submenu driver Expanded menu ONLINE-SYSTEM with submenu device
8	21.11.96	from 2.224	 all	New command line parameter /?, /H and /HELP Some small additional features Bug fixes Spelling corrections
9	26.10.97	from 2.224	1, 2	Using the program ComPro under Windows NT

Although this program has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this program for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this program or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifiactions at any time in as far as this contributes to technical progress. The version of the manual supplied with the program applies.

1 General

The 'ComPro' program is the configuration and diagnostic program for all equipment supplied by Hilscher.
The configuration data for the devices are stored in a database on the PC. With the aid of ComPro, this database can be processed and transferred to the target hardware.
Extensive status functions allow the user to monitor the device status at any time and change it where necessary. In addition, ComPro includes functions for supervision and manipulation of message traffic.

The databases have their own format, and can only be processed with ComPro. They are stored in the COMPRO directory as '*.DBM' files.

1.1 System conditions

The PC should have a main memory capacity of 2 MByte and a free disk memory capacity of 1 MByte. It is recommended to have a 386 as a processor or larger. It is a executable program in a DOS-Box for Windows and OS/2.

Following points are mind by configuration at the PC:

- Every PC should have at least 510 kByte DOS memory free.
- By using of ComPro with a CIF card the memory manager (QEMM, 386MAX, EMM386 etc.) must exlude a memory utilization of 2 kByte for **every** CIF card.
  (for example: QEMM => X=CA00-CA7F by CIF-Basicaddress=CA00:0000).

It will be checked how many memory is available while editing the data base. If there is no more memory available there will be displayed an error notifiction. Use in this case the external data base editor DBEDIT.EXE.

1.2 Installation and program start

To install ComPro, make the drive with the installation disk the current drive. Then starte the installation program INSTALL.EXE and follow the menue.

To start ComPro, switch to the '*target directory*' and enter

CPRUN <*RETURN*>

1.3 DOS-adjustment for OS/2

Following DOS-adjustments for OS/2 are to use:

parameter	adjustment
COM_DIRECT_ACCESS	ON
COM_HOLD	ON
COM_SELECT	COM1 or COM2
DOS_BACKROUND_EXECUTION	ON
DOS_HIGH	ON
DOS_RAMSIZE	640
(VIDEO_MODE_RESTRICTION	CGA)

DOS-adjustment for OS/2

1.4 Windows NT

The program ComPro can only run under Windows NT using a serial connection between the PC and the CIF for parameterization or diagnostic purposes.

The requirements are that the CIF has a diagnostic interface and a RS232 cable. The you can connect via the RS232 cable COM1 of the PC to the diagnostic interface of the CIF. Alternatively you can use COM2 of the PC.

The necessary parameter for starting program ComPro is described in chapter *The command line* in section *Parameter for the CIF* in this manual.

The pinning of the RS232 cable is desribed in the device manual of the device. With order number KAB-SRV this cable is available.

2 The command line

The batch ile stored all specific ComPro parameters for all different devices!

As ComPro can be installed for all our equipments (PKV, KPO, CIF and so on) equally. Because of this the various parameters have to be entered when the program is started. The program must know the connected device. **For every device we deliver a batch file called 'CPRUN.BAT', in which your settings are stored.**

2.1 General parameters

parameter	meaning
/? /H /HELP	Displays all command line parameters
/B:Keyfile	Name of the keyfiles (Macros), automatic execution by starting the program
/D:Datenbank.DBM	Name of the database, automatic loading by starting the program. The database must be installed in the COMPRO directory.
/MONO	Independent of the used graphic adapter the monochrome mode switched on.
/RT:xx	set up device basis address (see SETUP).
/CB:xx	set up ComPro basis address (see SETUP).

General call parameters

2.2 Parameter for the PKV and the KPO

parameter	meaning
/S:1	Selection of the PCs serial interface COM1 for the connection to the device diagnostic interface.
/S:2	Selection of the PCs serial interface COM2 for the connection to the device diagnostic interface.

Call parameters for PKV and KPO

2.3 Parameter for the CIF

parameter	meaning
/A:CIF-SEGMENT	Memory address of CIF jumpers on the Hardware side. For example: */A:CA00*
/S:1	Selection of the PCs serial interface COM1 for the connection to the device diagnostic interface.
/S:2	Selection of the PCs serial interface COM2 for the connection to the device diagnostic interface.
/DPM:xx	Size of the dual port memory in kByte. Possible parameters are 1, 2, 4, 8, 16. Default-parameter is 2 kByte.
/T:Data base.DBM	Name of the data base, which automatically downloaded on the CIF at start time. The data base must be in the COMPRO-directory.
/R:1	Force a warm start on the CIF.
/R:2	Force a cold start on the CIF.

Call parameters for CIF

The program ComPro has access to the CIF over the dual-port memory. The parameter is /A: and the segment address for example /A:CA00. This parameter is used in file CPRUN.BAT as default parameter.

As an alternativ a serial connection between ComPro and the CIF is usable, if the CIF has a diagnostic interface. The parameter for ComPro is in this case /S:1 (for COM1) or /S:2 (for COM2). To use a serial connection the parameter /A: does not be used anymore. The batch file CPRUN has to be modified to COMPRO /S:1 or COMPRO /S:2.

To build up a serial connection between CIF and PC a RS232 cable is necessary.

3 Description of ComPro functions

The ComPro main window consists of the menu line and the protocol window. The protocol window notes all the actions by the user, thus providing full traceability of all user actions and how the connected system or ComPro itself reacted to them.

User dialogue box:

'F10' key: jump to <ok-enter>

'ESC' key: jump to <cancel-esc>

In a user dialog box, the fields <ok-enter> und <cancel-esc> are permanently available. The <ok-enter> field can be reached with the 'F10' key, and the <cancel-esc> field with the 'ESC' key. Pressing these keys does **not** however terminate the dialog, but merely places the cursor on the relevant field. In order to terminate the dialog, the selected field has to be further acknowledged with *<enter>*.

The program is structured in six parts:

- *database* Off-line configuration and parameterization of the link (*edit, save, load, print*)
- *online* On-line configuration and status functions (*system, task, database, software, driver*)
- *diagnostic* Diagnostic functions (*filter, trace, protocol, monitor*)
- *setup* setting the address *(address, password)*
- *macro* macro functions *(recording, playing, delay)*
- *exit* Terminate ComPro session.

3.1 Offline processing of a database - *database*

The database records are the specific configuration data of a device. This is a file with the ending *.DBM, it is kept in the COMPRO directory. Thus it is possible to manage all configuration datas for several devices with one ComPro.

Every database subdivides serveral database tables, they contain information about the whole system (device hardware, device firmware and ComPro). With the database ComPro can adapt to the connected devices.

The first menu option provides all the functions for offline processing of a database.

- *edit* Editing of a database file
- *save* Saving a database to the hard disk
- *load* Loading a database from the hard disk
- *print* Printing a database table to a printer or a file

3.1.1 Editing a database - *edit*

A internal special database editor is provided for editing of configuration data. The editor checks all entries, and only permits the user to exit from an input field when the values entered in that field are within the valid range. The value range for the field concerned is shown in the last line of the editor window. All entries made are only stored in the PC's internal database when the <ok> field has been selected and acknowledged. If the user exit the editor with <cancel>, all entries are cancelled. There is no safety prompt!

By selecting of the menu option 'edit' all available tables of the loaded database were displayed. By means of <*cursor up*> respectively <*cursor down*> can be choosed a table of a list and opened with the <*enter*> key.

The key assignment in the editor is as follows:

- *TAB* - one field to the right
- *shift-TAB* - one field to the left
- *cursor up* - one field up
- *cursor down* - one field down
- *grey +* - insert protocol (not available on every tables)
- *grey -* - delete protocol (not available on every tables)
- *escape* - Cursor to <cancel> field
- *F10* - Cursor to <ok> field

3.1.2 Saving a database - *save*

A database from the hard-disk would always saved in topical directory. The file is stored with the extension 'DBM'.

A selection of the files for existing databases is displayed. The database can be saved by a existing or a new name.

By selection of a name from the list the current database will saved to the hard-disk by this name. The old database from the hard disk will be over written, after creating a **.BAK*-File.

If the database is to be saved with a new name, select the 'new' option. Then enter a file name with which the database is to be stored in the current directory.

The database is only saved when the entry is confirmed by pressing *'Enter'*.

3.1.3 Loading a database - *load*

A selection of the files for existing databases is displayed. If one of these databases is to be loaded, the name should be selected with *'cursor up'* or *'cursor down'* keys and *'enter'* pressed to confirm.

If there is a database in RAM which has not been saved, a prompt first appears enquiring whether the current database should be saved before a new database is loaded. If this prompt is answered with *<cancel-esc>*, the database in RAM is cancelled.

3.1.4 Printing a database - *print*

There is a facility for printing out a database table as formatted text respectively printing out to a file. First select a table of a list. The name of the file is stored with the extension **.PRN*. The default line length is 80 characters and the default page length 65 lines.

3.2 The online functions - *online*

The functions under this menu option directly influence (online) the connected device (CIF, PKV, KPO etc.). If an online function is activated and no device is connected, an error message is generated.

The menu option is subdivided into four submenus:

- *system* Reading of equipment configuration, firmware version, display of the modules, reading of global status, reading and setting the time of the device, coldstart, warmstart and bootstart.
- *task* Display the status of all tasks, display or edit the status of individual tasks, display the task versions.
- *database* Overview of the existing segments, transfer of the data base between the connected device and the ComPro, deletion of individual database segments.
- *software* Overview of the existing segments, loading of the device firmware or individual device functions (e.g SPC module) into the connected device.
- *driver* Switch driver free and display licensed drivers.

3.2.1 The device system functions- *system*

3.2.1.1 Device information - *device*

The device number (GNR), the serial number of the device (SNR) and the manufacturing date will be displayed.

name	display
date	21.02.1996
device number	9 2 0 9 0 5 4 0
serial number	0 0 0 0 0 5 4 3

Device information

3.2.1.2 Display of the firmware version - *firmware*

This function can be used to read out the name and the version of the firmware running on the device and its checksum.
Firmware indicates the file name of the firmware loaded and the device.
The date in the version is identical to the file date.

name	display
firmware	MODBUS CIF10/11
version	V1.234 01.02.95
checksum	0X1234

A firmware display for example

3.2.1.3 Display of the device configuration - *configuration*

This function is used to display the configuration of the connected device. Following selection of the *configuration* menu, the operator can choose between *hardware* and *software*.

On selection of the hardware configuration, the following table appears. The *values* are example values.

The unit code contains the file extension of the firmware.

parameter	display
unit code	H41
processor type	80C188
ram type	1 x 32kByte RAM
res. memory type	no present
boot memory	1 x 128k-FLASH (29F010)
SCC-count	2
system frequency	16 MHz
realtime clock	no present
software number	0
SCA-type	no present

A hardware configuration for example

On selection of the software configuration, the following table appears. The *values* are example values.

parameter	display
cycletime [ms]	2
max. taskcount	8
segment count	31
segment size	288
message size	255
RCS-version	01.100
device address	0
first SW number	0
cnt. SW-number	0

A software configuration for example

3.2.1.4 The device system functions - *modules*

This function can be used to display the modules composing the firmware. The firmware is composed of RCS, LIB and MCL modules. Together with the module name, the version, start address and status of the module are also displayed.

3.2.1.4.1 List of RCS modules

The modules of the RCS operating system present in the firmware are shown in a list.

number	name	version	start address	state
01:	RCS_SYS	1.007	D2AD:2CED	ok.
02:	RCS_FNC	1.006	DDAD:3CC3	ok.
03:	RCS_COM	1.006	D78A:622C	ok.
04:	RCS_SCC	1.007	E17D:2819	ok.
05:	RCS_DBM	1.001	E402:0175	ok.
06:		0.000	0000:0000	
07:		0.000	0000:0000	

List of RCS modules for example

3.2.1.4.2 List of LIB modules

The library modules present in the firmware are shown in a list.

number	name	version	start address	state
01:	LIB_SOR	1.003	F324:0038	ok
02:		0.000	0000:0000	
03:		0.000	0000:0000	
04:		0.000	0000:0000	
05:		0.000	0000:0000	
06:		0.000	0000:0000	
07:		0.000	0000:0000	

List of LIB modules for example

3.2.1.4.3 List of MCL modules

The communications modules present in the firmware are shown in a list.

number	name	version	start address	state
01:	MCL_KPO	2.002	E5A9:0048	ok.
02:	MCL_SCC	1.006	E4FE:071E	ok.
03:		0.000	0000:0000	

List of MCL modules for example

3.2.1.5 The device system function - *state*

The states of global variables are displayed. The number of the segments in the system, the free RAM length and the number of running tasks are also displayed. The idle task of the operating system is also counted.

free segment	29
free ram length	3.248
running task	2

Status display for example

3.2.1.6 Setting the system time - *time*

The device system time is not available on every device.

The system time is read out and can be changed.

3.2.1.7 Cold start - *coldstart*

A cold start is practicable. It is equivalent to switching the device off and on. During a cold start, all the data in RAM are deleted and the system was started new. A cold start interrupt the communication.

There is a safety prompt before this action is executed.

3.2.1.8 Warm start - *warmstart*

During a warm start, the hardware is reinitialized. In contrast to a cold start, the current parameters are retained. The system was started new with the current parameters. A warm start interrupts the communication.

There is a safety prompt before this action is executed.

3.2.1.9 The initial start - *bootstart*

On an bootstart, the connected device is placed in its booting condition. In this condition, the device only accepts the commands load firmware, database overview and configuration.

The system is reset and the initial program loader is activated without any possibly present firmware being started.

There is a safety prompt before this action is executed.

3.2.2 The task functions - *task*

The activities of the tasks on the device can be monitored.

3.2.2.1 Reading the version data - *version*

The version data for all tasks running on the connected device are displayed.

	name	version	prio	startidx	state
00:	RCS	1.000	0	0	ok
01:	TASK1	1.000	1	1	ok
02:	TASK2	1.000	2	2	ok
03:					
04:					
05:					
06:					
07:					

task - version for example

Prio indicates the priority of the task, and *Startidx* the index of the task.

3.2.2.2 Displaying the task status - *status read*

The call up of select status is dependent on the device software and not always possible. Please consult your protocol description.

On selection of the submenu *all* the current status of all tasks on the connected device is displayed online. This can be used to check what action the individual task is currently performing. In addition, it is possible to determine whether there is a connection between the diagnostic PC and the device.

On selection of *all*, the window appears with the task numbers, the individual task names and the status of the relevant task. The task names and task status indicated in the following table are intended as examples only.

number of task	task name	task state
00:	RCS	running
01:	TASK1	send
02:	TASK2	ready
03:		
04:		
05:		
06:		
07:		

Taskstate for example

In addition to the *all* status for the device, each task provides one or more individual states. These individual states contain, among other data, send, receive and error counters, and further task specific status information. Further details on the task status can be found in your protocol description.

Apart from the 'read only' task status, some devices also have writable task status fields. These are displayed on the left hand half of the screen, with the indication that edit mode can be entered with key '*F9'*. The values displayed can then be edited in the edit window and transferred to the connected device. The user must indicate which values in the status field are to be written to the hardware. Selection takes place by setting the selection field with the '*space bar*'. On confirmation of the dialog with <*ok*> the data are transferred. It is to be noted that no current status information is displayed in the left hand status field during edit mode.

3.2.2.3 Resetting the task status - *status reset*

With this function, status fields for individual tasks or all tasks can be reset.

3.2.3 Device configuration - *database*

In most devices, the configuration data are stored in a flash EPROM, protected from zero voltage. A number of CIF cards which store the configuration data in a dual port memory are an exception.

The flash EPROM is divided into several segments. These segments are used by the firmware for different purposes. The segmentation is specified by the firmware and can only be altered by exchanging the firmware EPROM.

3.2.3.1 Overview of the database segments - *overview*

With this submenu, the segmentation of the flash EPROM and the status of the individual segments can be viewed.

Segments of type *DBM* are used for storage of configuration data. Segments of type *etc* (for 'etcetera') form add on segments and thus enlarge the previous segment. This is the case, for example, with large configuration databases. The segmentation is specified by the firmware and can only be altered by replacing the firmware. The current length of the first entry is the total length.

	name	type	mode	address	max. len	act. len	state
0	PROFIBUS	DBM	0x00	4000:0000	16.384	12.230	ok
1			0x00	0000:0000	16.384	0	ok
2			0x00	0000:0000	16.384	0	ok
3			0x00	0000:0000	16.384	0	ok
4			0x00	0000:0000	16.384	0	ok
5			0x00	0000:0000	16.384	0	ok
6			0x00	0000:0000	16.384	0	ok
7			0x00	0000:0000	16.384	0	ok

Overview for example

Segment errors can occur if an illegal database or no database at all is loaded. An incorrect flash module or a defective configuration can also lead to errors. Incorrect configuration data such as incorrect baud rate or parity, etc. are then not detected!

The maximum permissible length of the configuration data is determined from the tables displayed. If an attempt is made to load a longer database than permissible into the device, this is acknowledged with an error message on downloading. The configuration data in the device are then invalid!

3.2.3.2 Transferring the configuration data into the device - *download*

Transfer of the database into the connected device is activated in this submenu. All the tasks in the device have to be stopped for transfer of the database. This is effected by the ComPro after a safety prompt. Transfer is then initialized and the data then downloaded. The device is then re-initialized. The parameters from the database transferred are accepted by the firmware as new parameters.

Prior to downloading, the ComPro checks whether the database loaded is suitable for the device firmware. If the database and device firmware are incompatible, error message *Message (53): RCS_FLASH_FILE* appears.

3.2.3.3 Reading the configuration data from the device - *upload*

The database is transferred from the connected device to the PC. The function of the device remains unaffected. If there is a database in the PC memory which has not been saved, the user has an opportunity to save this prior to uploading.

3.2.3.4 Deletion of the configuratoin data in the device - *delete*

Individual segments in the flash EPROM can be deleted with this function. Depending on the firmware, the device will stop operation immediately, or following the next warm start/cold start at the latest.

3.2.4 The device system functions - *software*

3.2.4.1 Overview of the firmware modules - *overview*

With this submenu, the segmentation of the flash EPROM and the status of the individual segments can be viewed.

3.2.4.2 Modification of the firmware modules - *program load*

With this function, an individual module is downloaded from the PC into the connected device. Individual functions of the firmware can thus be systematically modified or adapted.

There is a safety prompt before the action is executed.

3.2.4.3 Loading the firmware modules - *firmware load*

This function downloads the complete firmware from the PC into the connected device. It is dependent on the firmware being stored in a flash EPROM in the device.

This function allows firmware modification to be performed at any time.

3.2.5 Driver activation - *driver*

3.2.5.1 Display drivers - *display*

With this menu item all licensed drivers will be displayed.

3.2.5.2 Activate driver - *activate*

It appears the request to enter the activation code. The input of the code is in hexadecimal description. The code can have a length up to 16 bytes and is only for licensed drivers.

First, licensed drivers must be acquired from us. Therefore we need the serial number of the device hardware (CIF communication interface, COM Communication module). Then the activation code is spent by us. Finally, enter the activation code to free the driver.

3.3 The diagnostic functions

The diagnostic functions described below can be used for monitoring and manipulation of the function of the communications device.

3.3.1 The trace filter - *filter*

The filter dialog provides for an indication of what information the tasks should enter in the trace buffer. The trace filters can be activated individually for each task or globally for all tasks.

With the activation of the trace filters, the selected tasks enter the required information in the trace buffer. This information can be displayed using the submenu *trace.*

To activate a trace filter, an *event* and an *entry* are to be specified. Under *event*, the user sets the conditions under which a task is to effect an entry in the trace. Under *entry*, the user selects the extent to which the task stores data in the trace.

Specification of the event, on the basis of which entries are made:

- *off* no entries
- *send/receive* on sending and receipt of a message
- *sending* on sending of a message
- *receiving* on receipt of a message
- *error* on occurrence of an error
- *always* on any event

Specification of the information to be entered in the trace buffer (Entry):

With some protocols, no distinction is made between header and user data, and therefore the settings for header data/user data/data are identical.

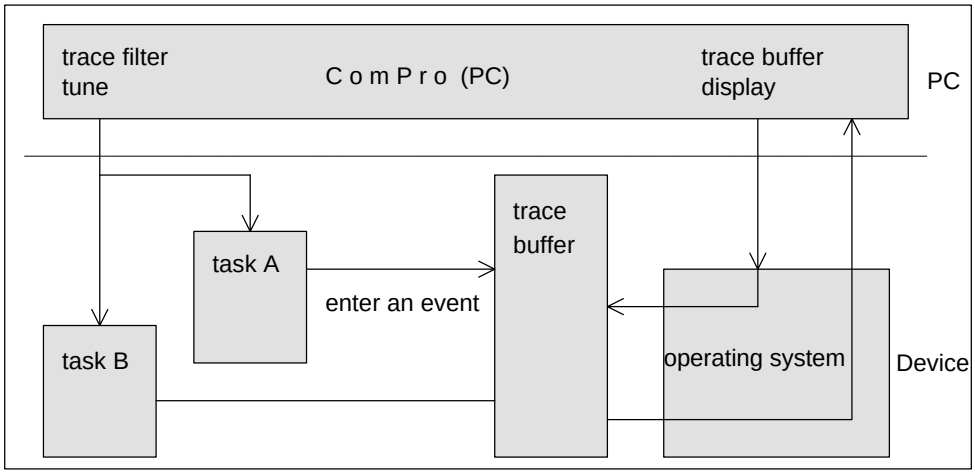
- *off* no data
- *telegram header* Only the header data of the telegrams transmitted/ received
- *telegram data* Only the user datas of a telegram transmitted/received
- *telegram* Only a telegram transmitted/received with its user and header data
- *message header* Only the header data of a system message input or output
- *message data* Only the user data of a system message input or output
- *message* A system message input or output complete with its user and header data

3.3.2 The trace - *trace*

The setting of which task enters signals in the trace buffer on what event is effected with the FILTER submenu.

The diagnostic functions of the ComPro are used to test a link and to locate errors. For this purpose, the device has a trace buffer, in which the tasks enter certain data on occurrence of an event. **The entries are also made when the ComPro is not connected!** Entry of data in the trace buffer is started by the setting of a trace filter.

The operating system on the device transfers the contents of the trace buffer to the ComPro when the *Trace* submenu has been selected and the subsequent prompt answered. As the trace buffer is designed as a ring buffer, a buffer overflow may occur. This means that the trace protocol is incomplete. A trace buffer overflow is displayed on reading by ComPro.



Course of the trace function

Following selection of the trace submenu, there is a prompt as to whether the contents already stored in the trace buffer are to be displayed.

Trace buffer **output** is activated with <*ok*> without previously deleting the trace buffer.

An entry of <*no*> also activates the trace buffer output. In this case, however, all trace entries, i.e. the entire trace buffer, are deleted! The entries from the tasks which then follow are output.

Stopping acts on the output only, and trace buffer entries continue to be made by the tasks.

All entries in the trace buffer are transmitted to the PC and displayed. Trace buffer output is terminated with '*esc*'. The **Output** can be stopped with the key combination '*ctrl-S*' and startes again with '*ctrl-Q*'.

3.3.3 Recording trace outputs in a file - *protocol*

On activation of the protocol function, all trace buffer entries displayed are recorded in a file. On opening of a protocol file, any existing file of the same name is deleted! The file created has the name <*filename*>.*PRT* .

The protocol file is an ASCII file and can be viewed with any ASCII editor and with the DOS command *type* <*filename*> .

If recording is active, repeated selection of the submenu activates the recording function.

3.3.4 The message monitor - *monitor*

With the message monitor, you have the opportunity to intervene actively in the working sequence of the device software by sending messages to individual tasks and receiving messages from them. For this purpose, you can use the message monitor to compile, edit, save, load and of course send and receive messages.

Each task on the device 'understands' certain commands, which are transmitted to it in the form of a message. On receipt of a command, the task will execute and acknowledge that command. The transmission of a command to a task is initially independent from the sending of a message via an interface. Each interface task does however know commands which cause it to send a message.

The task commands can be found in the manual on the protocol.

The structure of a command is dependent on the task which performs it. Please consult the manual for the protocol for the command supported by your software.

3.3.4.1 The windows of the message monitor

The monitor screen is divided into four windows. The two left hand windows are output windows for the messages received (receive windows), and the two right hand windows are used to enter, edit and send messages (send windows). Messages in a send window can be saved to the hard disk and loaded again.

Messages are always stored in the subdirectory *MSG*. This directory must be created before messages are saved.

Both windows are divided into message headers and message data. The message header has a fixed structure, which is defined as follows:

Rx Receiver of the message

Tx Sender of the message

Ln Length of the utilization data (calculated!)

Nr Message number for identification of the message

A Answer identifier

F Error identifier

B Command identifier

E Extension

The contents of a message are described in detail in the manual on the protocol. Entry and display are hexadecimal.

3.3.4.2 The functions of the monitor

Key	Function
'Esc'	exit from the monitor
'TAB'	toggle between message header and message data windows
'F1'	send message to the selected receiver
'F2'	load a stored message from the hard disk. The message is searched for in the subdirectory *MSG*.MSG* Subdirectory *MSG* must exist.
'F3'	save message to hard disk. The message is stored in the subdirectory *MSG* under the name *name.MSG* . The subdirectory *MSG* must exist.
'F4'	remove a stored message from the hard disk. A selected message *Name.MSG* is deleted from the subdirectory *MSG*. The subdirectory *MSG* must exist.
'F5'	activate protocol When the protocol is activated, **all** messages sent and received are stored in an ASCII file on disk. The file has the extension **.ASC*.
'F6'	Reset Counter The counters for messages sent and messages received are reset to zero.
'F7'	Nr.Inc The user can toggle between automatic incrementation of the message number on each transmission and no incrementation.

Keys for the monitor

3.4 Settings - *setup*

3.4.1 Setting the basic addresses - *address*

A device can contain several subsystems. Intelligent communications cards for the PROFIBUS are an example. These subsystems can be described as a device within the device. Each system is addressed by ComPro by means of a unique basic address. Device address zero is reserved for the main system and the default settings. The addresses of the subsystems can be found in the relevant device description.

The basic address of ComPro is only of significance for devices with several diagnostic interfaces. In such a case, a unique identification of the ComPro must be set.

3.4.2 Function - *password*

Prepared for a future function.

3.5 Macro

There is a facility for recording keystrokes with the macro recorder and replaying them. A recorded macro is always stored on the hard disk as an ASCII file.

3.5.1 Recording macros - *recording*

In order to start recording the macro, it must first be given a name. The name of the macro corresponds to the file name under which it is stored on the hard disk. When entry of the name has been acknowledged with '*enter*', the following message appears on the screen:

**** Macro recording active: Name.MAK ****
Macro end with 'CTRL-A'
Insert pause with 'CTRL-P'
Insert wait key with 'CTRL-K'

When recording is active, all keystrokes are stored in the macro apart from the control keys mentioned above.

Key	Function
'Ctrl-X'	End macro recording Completion of recording is acknowledged with the message ****Macro recording ended**** even if the main screen is not activated (e.g. diagnostic, trace).
'Ctrl-P'	When playing the macro, a pause of approx. 1 second is inserted at the point where these keys have been pressed. Longer pauses can be inserted by pressing the keys several times.
'Ctrl-K'	When playing the macro, it waits for a key to be pressed at this point

These control keys can be used at any time, even if for example the message monitor is active.

During recording, the menu options *Play* and *Delay* are not available.

3.5.2 Playing the macro - *playing*

On selection of a macro, it is played with the delay set. **No** interruption is currently possible! A macro can be automatically executed on the start of the ComPro program (see 'The command line').

3.5.3 Delay macro - *delay*

A pause period can be specified, with the result that the pause occurs between re-playing of the individual keystrokes.

3.6 Terminate ComPro - *exit*

ComPro is terminated, a prompt as to whether the user wishes to exit from the program follows.

If the current database has not yet been saved there is then a safety prompt.

Device manual

PKV 30-DNS

Protocol converter for DeviceNet Slave

Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 78
D-65795 Hattersheim
Deutschland

Tel. +49 (0) 6190/9907-0
Fax. +49 (0) 6190/9907-50

Sales: +49 (0) 6190/9907-0
Hotline and Support: +49 (0) 6190/9907-99

e-mail: hilscher@hilscher.com
Homepage: http://www.hilscher.com

Index	Date	Device	Device number	Chapter	Revision
1	23.02.99	PKV 30-DNS	G981700, Rev A	all	created

Although this appliance has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this appliance for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this appliance or its documentation shall be limited to cases of intent.We reserve the right to modify our products and their specifiactions at any time in as far as this contributes to technical progress. The version of the manual supplied with the appliance applies.

1 Introduction

1.1 Purpose

There is often a need to transfer data between controllers from different manufacturers or exchange them with a higher level host computer. Each system has its own transfer protocol. Implementation of an alien protocol is often impossible or only possible at high cost. This results from the following background conditions:

- The interface drivers do not match
- The interface controller does not meet the requirements
- Insufficient or lacking computer power
- Real time requirements cannot be fulfilled
- No facility for making additional configuration data available
- Missing or insufficient commissioning and diagnosis aids

In some cases, the protocol is only required for a single system. Nevertheless the implementation has to satisfy extreme quality demands. The effects of a software error in the transfer protocol can lead to faults ranging up to the standstill of the entire system and thus cause unpredictable costs.

Experience shows that implementation testing and inspection in particular only take place in the laboratory or on a system which is hardly capable of functioning. In spite of the greatest possible care, then it happens that an error only becomes apparent when the plant is in service. Particularly when the error occurs sporadically or depends on particular plant conditions, location of the error without an integrated diagnostic function is a matter of luck.

Practical experience shows that most problems are created not by errors in the implementation but by inadequate agreements on the user level. Message matching between the linked units is in some cases incomplete or is not complied with. As a result, messages which have not been sent are expected, or messages are sometimes mixed up. If the message traffic can be transcribed, the problem can be rapidly rectified.

The protocol converter is a device developed specifically for these problems, whose operating system provides all functions necessary for the rational and reliable implementation of coupling protocols.

The device is supplied in various versions. The protocol converter PKV 30-DNS has two communication interfaces and one of them is for DeviceNet slave and the other one is for general protocols.

2 General Device Discription

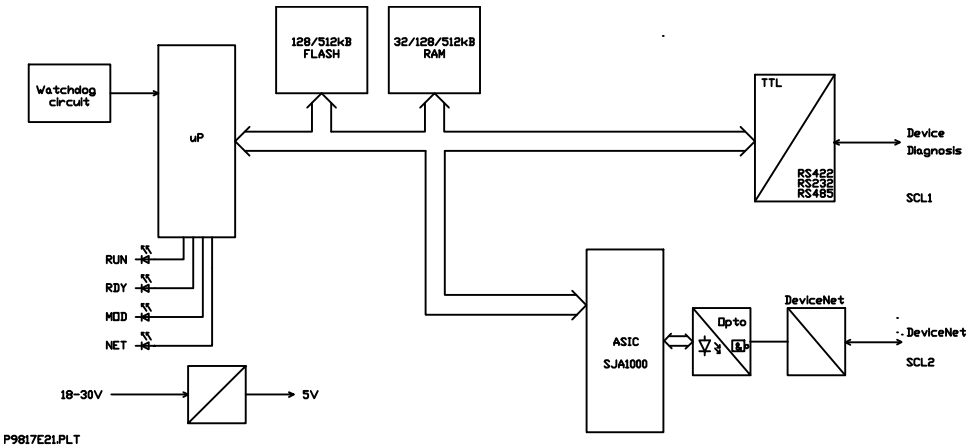
The protocol converter consists of a main board and a power supply board, with the DC/DC converter which generates all internal voltages.

The main board has a 80C188 microprocessor. It has an internal timer, interrupt and DMA controllers, and therefore requires only a few external components. The computing power is sufficient to process even large quantities of data. In addition, the 16-bit processor facilitates efficient software development in a high level language. The firmware and the configuration data are stored in a flash EPROM. This can be programmed in the circuit and retains its data even when the power supply is switched off.
The serial interfaces are realized by the SCC controller AM85C30 and the ASIC SJA1000. For the connection of a device with the converter there is a non-isolated interface wich can be configured as RS232, RS422 or RS485-type. The second isolated interface is designed for the DeviceNet.

The correct function of the protocol converter and its internal power supply are monitored by a watchdog circuit with the MAX 809 component. In the case of error, this triggers a reset on the processor.

The internal power supply is provided by a switching controller. Its input voltage is filtered through a current compensated annular core reactor and filter capacitors. A transient diode is provided as overvoltage and reverse polarity protection. In the case of a fault, the internal semiconductor fuse switch off until the fault disappears. In addition, there is a charging capacitor which blocks voltage drops such as those which occur on switching of contactors.

Valid operation and an error state from the serial interfaces are displayed by LEDs.

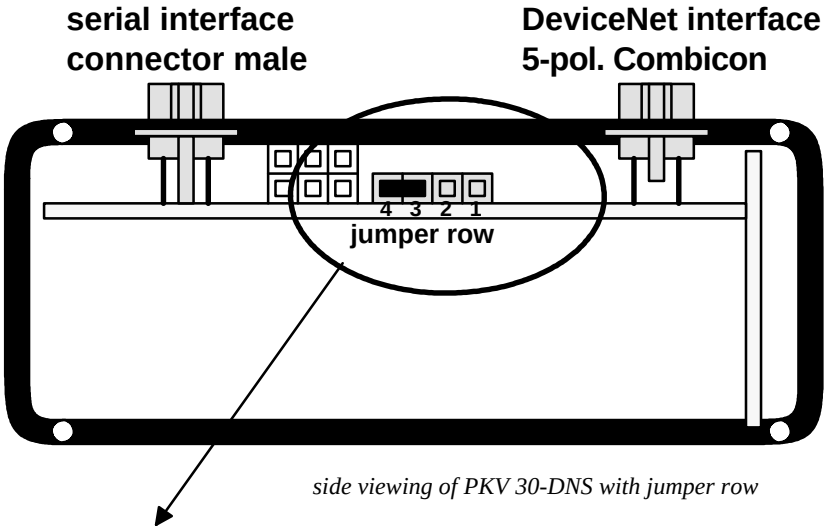


Block diagram of the protocol converter PKV 30-DNS

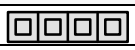
2.1 Configuration

2.1.1 Selection of Serial Interface Type

The hardware of the converter has jumper J5, to switch between the different types of serial interfaces. This jumper is accessable if you open the side plate on the side where there are both serial interfaces. This is be done by removing four screws. The following picture will show the position of the jumper seeing from the side of the converter. With the help of the table you can choose the right type of interface.



side viewing of PKV 30-DNS with jumper row

selection	jumper J5	Interfacetyp
	open	RS232
	3-4	RS485
	1-2, 3-4	RS422

4 3 2 1

Selection of serial interface type

2.1.2 Selection of the DeviceNet Slave Station Address

The selcetion of the slave address (0-63) is done with two code switches. Please refer to chapter 'Physical dimensions' in this manual.

2.2 Connection of the Power Supply (X1)

The protocol converter requires a 24V power supply. For maximum load of the current see chapter 'Technical data' in this manual. A threephase rectifier or simple rectifier circuit with charging capacitor are sufficient. The power supply must be earth grounded.

The power supply are connected by means of a plug-in screw terminal. A 3-way COMBICON plug from PHOENIX (MSTB 2.5/3-ST-5.08) is used.

Pin	Symbol	Signal
1	+24V	+24V supply voltage
2	0V	reference potential
3	PE	earth ground

Pin assignment of the X1 power supply connector

2.3 Serial Interfaces

The PKV 30-DNS has two independent serial interfaces.

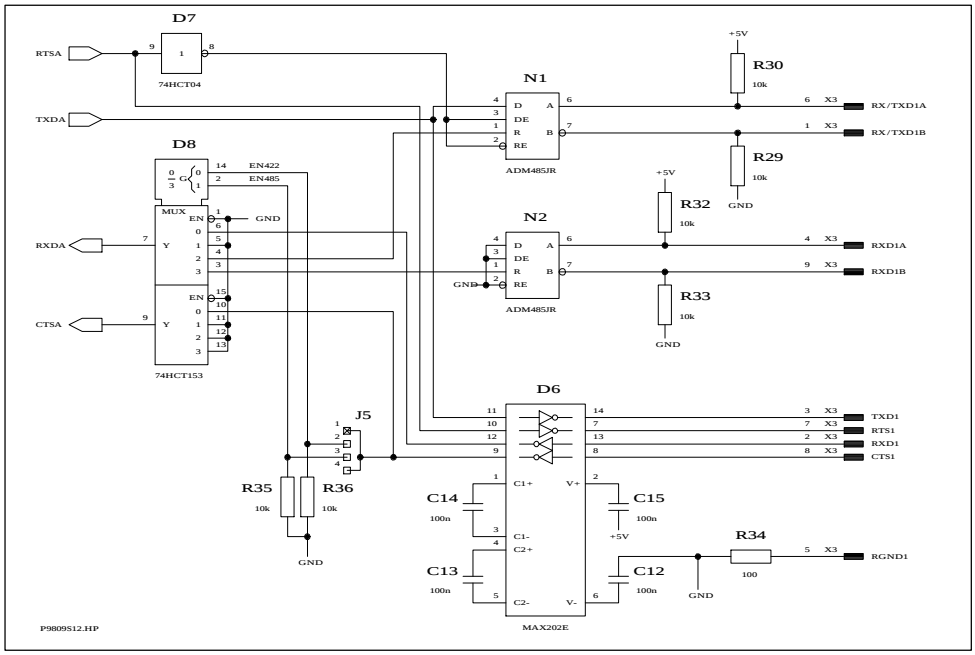
First serial interface (X3). The first interface can be used for communication to another device as RS232, RS422, or RS485 interface or for diagnostic/configuration. The settings of the communication interface as RS232, RS422, or RS485 interface is done with jumper J5. (see chapter configuration).

Second serial interfcae (X2). The second interface is the connection for the DeviceNet slave.

2.3.1 First Serial Interface (X3)

The serial interface uses a pin D-Sub connector for the connection to a device with a serial RS232, RS422, or RS485 interfaces. The interface is configurable with jumpers (jumper row J5).

This interface is also used for diagnostic/configuration of the protocol converter. The activation of the diagnostic/configuration mode is described in chapter activation of the diagnostic/configuration mode in this manual.



Output schematic of the serial device interface at connector X3

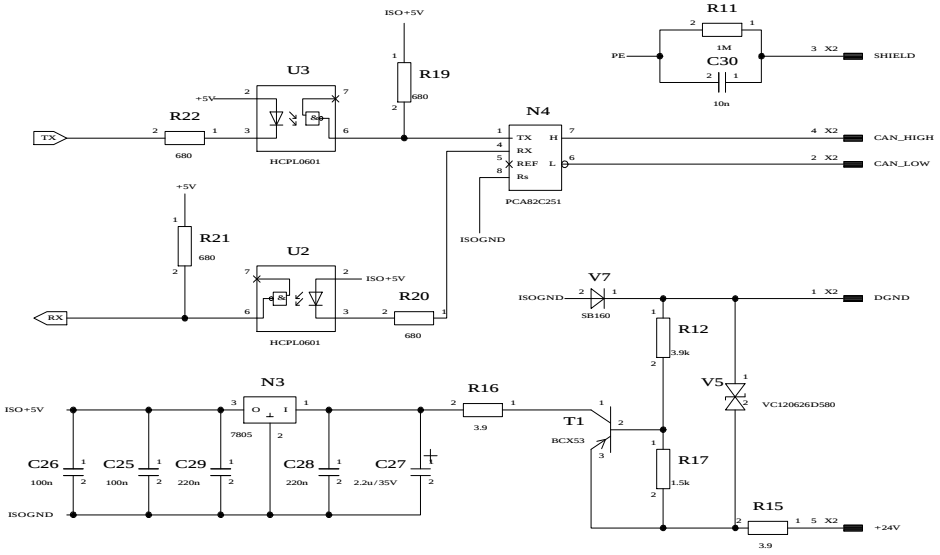
The resistors R29, R30 and R32, R33 are necessary to get a defined potential on the RS485/RS422-bus if there is no active transmitter or the connector is left open. All resistance have a value of 10 kOhm.
The following table describes the signals of the different kind of interfaces. It is not allowed to make any connections to the other signals of unused kind of interfaces.

pin	input/output	signal name	signal	RS 485	RS 422	RS 232
1	input/output output	RXD/TXD-N TXD-N	transmit data inverted RS 422 data inverted RS485	✓	✓	
2	input	RXD	receive data RS 232			✓
3	output	TXD	transmit data RS 232			✓
4	input	RXD-P	receive data RS 422		✓	
5		RGND	reference potential accros 100 Ohm	✓	✓	✓
6	input/output output	RXD/TXD-P TXD-P	transmit data RS 422 data RS 485	✓	✓	
7	output	RTS	ready to send			✓
8	input	CTS	clear to send			✓
9	input	RXD-N	receive data inverted RS 422		✓	

Pin assignment of the serial device interface X3

Using the RS232 interface use the pins only which are marked with a check in the table.

2.3.2 DeviceNet Interface (X2)



Schematic of DeviceNet interface X2

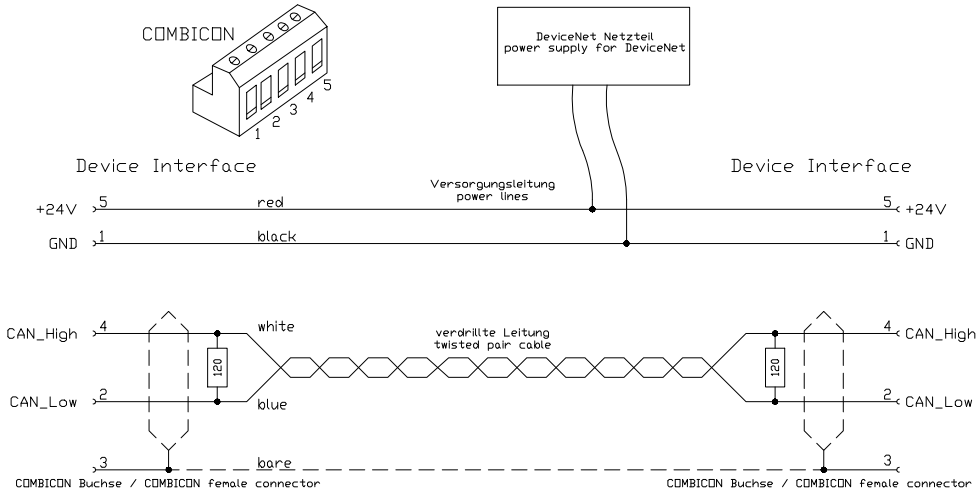
DeviceNet fieldbus connection

Pin COMBICON	Input/ Output	Symbol	Signal
1		DGND	Reference potential
2	Input/Output	CAN_Low	Receive/ Send Data-Low
3		Shield	Cable Shield
4	Input/Output	CAN_High	Receive/ Send Data-High
5		+24V	Power supply of the interface

Pin assignment of the DeviceNet interface X2

2.3.3 DeviceNet Cable and Connection

Please use only special cable types according to the DeviceNet standard for the connection of DeviceNet devices. The power supply of the fieldbus interface of the different devices and the data lines will be connected with the cable. So it is possible to guarantee proper operationm of the network, when a device is switched off.



2.3.4 Bus Termination

The fieldbus termination resistors of 120 Ohm must be fit in between the two data lines at the end of the network for proper data transmission.

2.4 Diagnostic Interface

This interface facilitates the connection of a PC to the protocol converter. It conforms to the RS232C standard to CCITT or DIN. Only the necessary signals are provided.

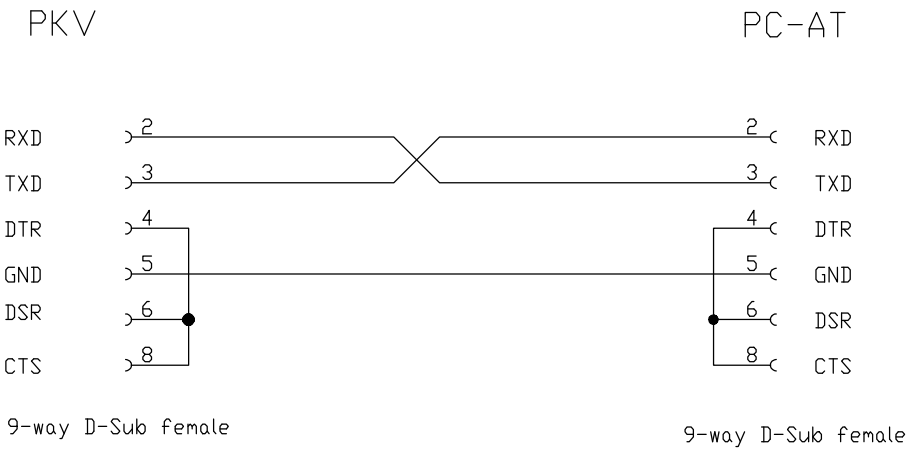
The control signals are produced and evaluated as follows:

- RTS is switched to high on readiness for operation and not changed.
- CTS must be connected internal with pin 4 and 8 of D-Sub connector

A 9-way D-Sub connector to DIN 41652 is used.

Data transfer between the PC and the protocol converter takes place at 9600 Baud and uses the following data format: 8 data bits, 1 stop bit and even parity. The 3964R protocol is used as the transfer protocol.

The PC is connected by a three-way cable which must not be longer than 15 m. The wiring of the cable is shown below.



KABPKVE2

Diagnostic cable between the protocol converter and the PC

The interface on connector X3 of the PKV 30-DNS is designed for

- communication

respectively for

- diagnostic/configuration

If the interface is configured as RS422 or RS485 it is **not necessary** to change for the diagnostic mode.

If the power of the protocol converter is switched on, it checks the serial interface if the diagnostic/configuration mode should be activated, otherwise it starts the communication mode.

2.4.1 Activation of the Diagnostic/Configuration Mode

Connect the diagnostic cable on X3 of the protocol converter and to COM1 (or COM2) of the PC.

Start the program ComPro

`COMPRO /S:1` respectively. `COMPRO /S:2`

(for COM1 respectively COM2).

Select menu *online - system - bootstart.* A window appears that shows *The system will be reseted and the bootloader becomes active without starting any firmware*.

Turn off power of the protocol converter and wait at least 10 seconds.

Accept the message *The system will be reseted and the bootloader becomes active without starting any firmware* and press the Enter key. A red window appears that shows *Waiting for hardware receipt*.

Power on the protocol converter.

The red window disappears.

To test if the protocol converter is in diagnostic/configuration mode select menu *online system - firmware.* If a window appears and shows the name of the firmware, then the diagnostic/configuration mode is active. If the message *Connection could not be established or connection lost* appears, then try to activate the diagnostic/configuration mode again according to the steps described above.

2.5 Status Displays LED

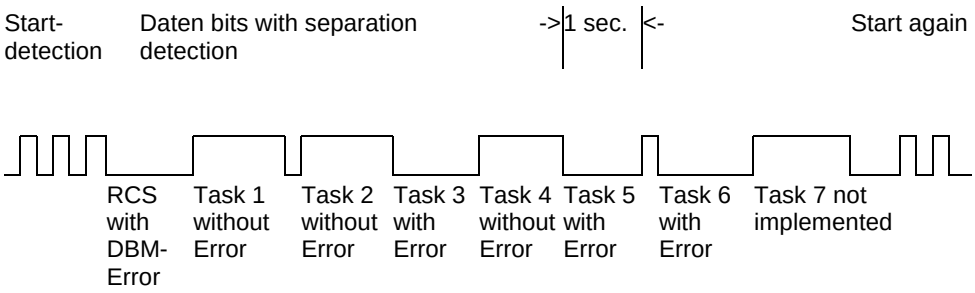
On the PKV 30-DNS exists 4 LEDs.

LED	Colour	State	Meaning
RDY	Yellow	on flashing cyclic flashing non cyclic off	PKV ready bootstrap loader active hardware or system error hardware error
RUN	Green	on flashing non cyclic (see below) off	communication running parameter error no communication
NET	Red	on	duplicate Device detected
		flashing	Connection time out
		off	Device not powered
	Green	on	Connection established
		flashing	no Connection established
		off	Device not powered
MOD	Red	on	Configuration Error
		off	Device not powered
	Green	on	CIF Normal Operation
		flashing	Configuration is missing
		off	Device not powered

If the device is defective, it is possible that the watchdog responds cyclically, what results also in a cyclically flashing of the RDY-LED.

When switched on, the PKV performs a self-test. If this is performed without error, the yellow RDY LED is switched on. Otherwise, the LED starts to flash, and further running of the program is aborted.
If no firmware has been loaded in the PKV, the bootstraploader displays this by regular flashing of the RDY LED at one second intervals. The flash frequency is increased to approx. 5 Hz while the firmware is being loaded.
Stays the LED off the may be a defect.

If a parameterization error is detected by a protocol task, the task is displayed by the RUN LED as shown in the illustration below.
If no error occurs and communication has started, the RUN LED is switched on. Is this LED blinking cyclic, no parameterization error has been detected, but the communication on the bus has not been established.
If communication is blocked, e.g. by the 'System start' parameter, the RUN LED remains off.



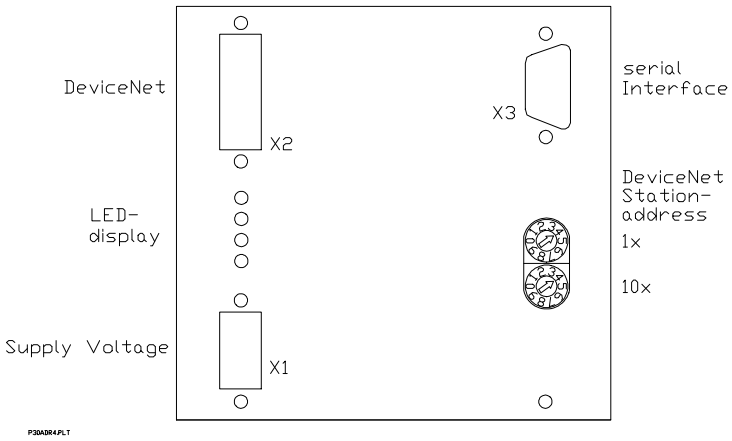
Display by the LED is from left to right

Display of the task which reports a parameterization error

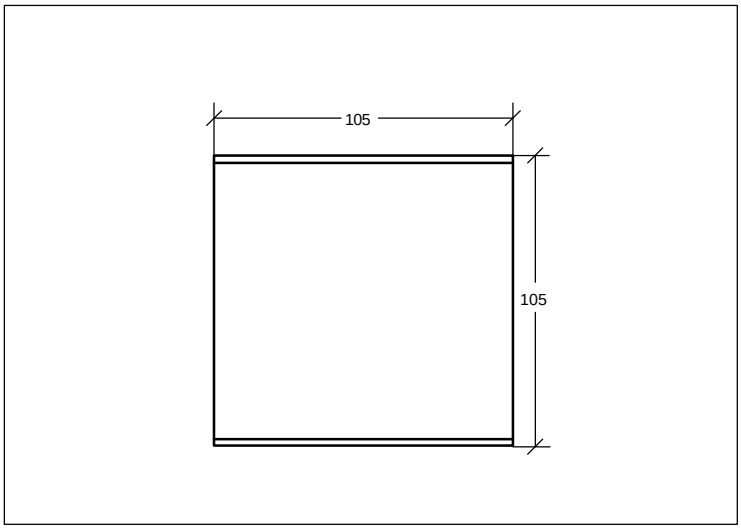
2.6 Physical Dimensions

The protocol converter is installed in an aluminium enclosure. This facilitates direct installation in the cabinet on a mounting rail (TS35 to DIN EN 50022). As an alternative, an enclosure variant for direct screw fitting on an assembly wall is available.
The physical dimensions and the arrangement of the plug connectors are shown in the drawings below.

Open the top of the protocol converter if you want to change the serial interface type
---------->



Connector arrangement on the protocol converter



Physical dimensions of the protocol converter for snap mounting to a mounting rail

The overall height of the converter is approx. 80 mm.

3 Appendix

3.1 Technical Data

Processor	16 Bit with timer, interrupt and DMA controllers, two serial interfaces, watchdog
Memory range	512 kByte RAM, 512 kByte FLASH
Serial interface	configurable as RS232C, RS422, RS485-interface, non-isolated, max. data transmission 19.2 kBaud, asynchronous, synchronous
DeviceNet interface	ISO High speed, potential free, 500 kBaud
Display-LEDs	system running and communication running, or communication error on serial interface, status of DeviceNet
Operating voltages	18V - 30V, max. 0,15 A at 24 V
Operating temperature	0 to 50 degrees Celsius
Safety type	IP50
Dimensions (LxHxD)	105 x 105 x 80 mm
Mounting	rail mounting DIN EN 50022

Bridge manual

Conversion DeviceNet Slave to Modbus RTU

Hilscher Gesellschaft für Systemautomation mbH
D-65795 Hattersheim
Rheinstraße 78
Germany

Tel. +49 (0) 6190/9907-0
Fax. +49 (0) 6190/9907-50

Sales: +49 (0) 6190/9907-0
Hotline and Support: +49 (0) 6190/9907-99

e-mail: hilscher@hilscher.com
Web: http://www.hilscher.com

Index	Date	Version	Chapter	Revision
1	25.02.99	V1.000	all	draw up
2	10.05.99	V1.002	all	Errors removed / Improvements

Although this protocol implementation has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this protocol implementation for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this protocol implementation or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifications at any time in as far as this contributes to technical progress. The version of the manual supplied with the protocol implementation applies.

1 Introduction

This manual describes the linking of devices with the Modbus RTU and DeviceNet protocol based on the PKV30-DNS protocol converter.

The Modbus RTU protocol can be configured as master or slave. At the DeviceNet the protocol converter is a slave.

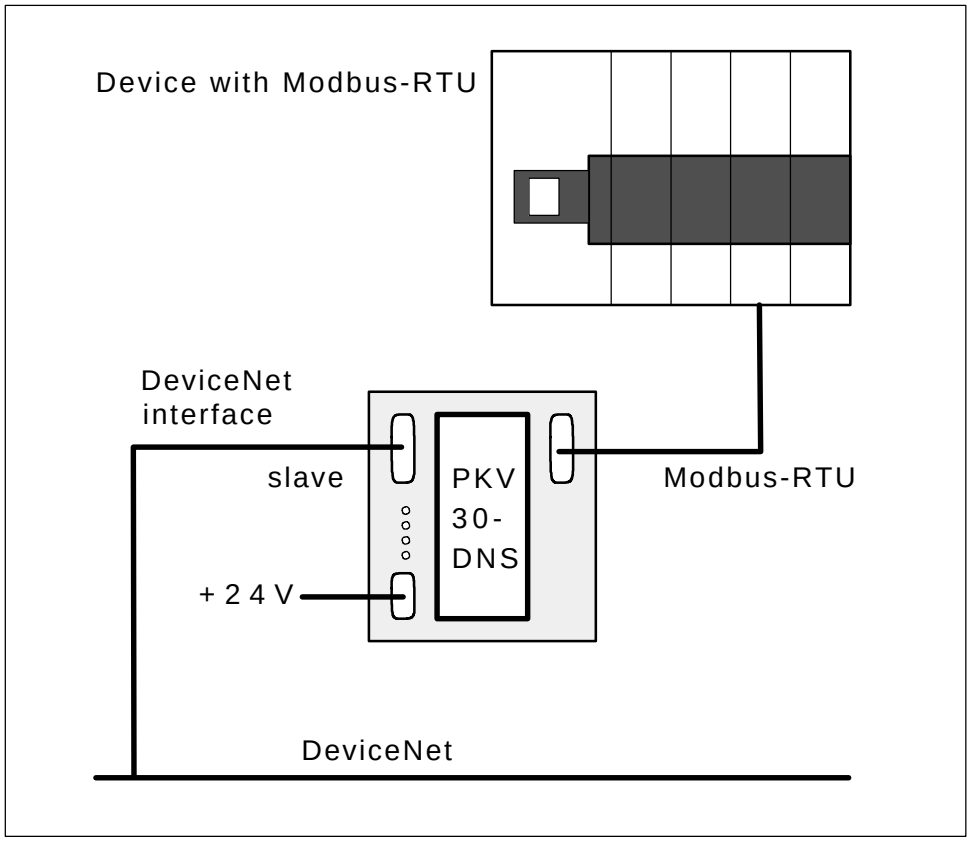
The data exchange is carried out by means of two common process images in the protocol converter. These process images correspond to the input and output data of the DeviceNet and are cyclically exchanged with the DeviceNet master.
The memories can be accessed via the Modbus RTU protocol. The access is different depending on whether the Modbus RTU protocol is operated as a master or a slave:

Modbus RTU as slave	The Modbus RTU master can access the process images directly by means of the function codes 1, 2, 3, 4, 5, 6, 15 and 16.
Modbus RTU as master	A job list is defined on the protocol converter, which the protocol converter processes cyclically, and thus exchanges data between connected Modbus slaves and its process image. For all function codes that write data to the slaves (FC 5, 6, 15 and 16), it can be additionally parametrized whether the data is to be written cyclically or only when the data is changed.

When the Modbus RTU protocol works as a master, then a **supervision** of the Modbus RTU slaves is possible. A bit-coded status field in the input data to the DeviceNet master then provides information on the current condition of the Modbus RTU slaves. The slave supervision is described in the *Status and error bits of the Modbus slave participants* chapter.

Furthermore, in the master operation of the Modbus RTU protocol, a **Watchdog supervision** of the Modbus RTU is possible. In this, the protocol converter reads a particular register (Watchdog register) of a Modbus RTU slave with a cycling time that can be parametrized. If this reading process is unsuccessful, then the failure of the whole Modbus RTU or all the devices connected to it, is assumed. In this case **all** input data to the DeviceNet master are set to **zero**. The inputs to the DeviceNet master are again served when the Watchdog register becomes available again. The Watchdog supervision is described in *The Watchdog supervision* chapter.

The parametrizing and diagnostic program ComPro is used for the configuration. This tool is described in its own manual.



Connection of the devices to the PKV 30-DNS

2 Data image of the data in the protocol converter

The data exchange is carried out via the process images in the protocol converter. This consists of a maximum of 128 input words and 128 output words. How much data is exchanged over the bus depends on the configuration of the DeviceNet and is independent of the accesses on the part of the Modbus RTU. Here is just tested whether the access is within the data region. It is up to the user to configure the DeviceNet such that relevant process data is available at the corresponding positions of the process image. Up to 255 bytes can be exchanged for input and output data at the DeviceNet.

2.1 Data formation from the viewpoint of the DeviceNet

The meaning of inputs and outputs must always be seen from the viewpoint of the DeviceNet master.

Input data	is always data that is received from the DeviceNet master.
Output data	is always data that is output from the DeviceNet master.

The input and output data of the common memory is exchanged right from the start. Only the amount of data is determined by the configuration. The arrangement for the function codes 3, 6 and 16 between the Modbus address (Mem.Adr. in the COMMAND table) and the process images looks as follows:

	DeviceNet - Bytes in the common memory		Mem.Adr.
Input data of the DeviceNet (Write from the Modbus side with FC 6 and 16)	1. Byte	2. Byte	40.001
	3. Byte	4. Byte	40.002
	5. Byte	6. Byte	40.003
	...	...	...
	...	Last byte	
	...	...	40.128
Output data of the DeviceNet (Read from the Modbus side with FC 3 and 4)	1. Byte	2. Byte	40.001
	3. Byte	4. Byte	40.002
	5. Byte	6. Byte	40.003
	...	...	...
	...	Last byte	
	...	...	40.128

Imageing of the DeviceNet data on the Modbus data

This means that there is always depicted the **lowest valid Modbus address** (here 40.001) **for the function code used** (here 3, 6 or 16) at the start of the process image! Correspondingly, the function code 4 in connection with the Modbus address 30.001 is **also** depicted at the start of the process image. The same is valid to the access to bit markers with the function codes 1, 2, 5 and 15.

The access to the process image is also possible bit-wise (bit marker, coils) with the function codes 1, 2, 5, and 15. The following picture shows how the individual bits are arranged in the common process image.

		Register	Arrangement of the bit markers / coils in the common memory															
Input data of the DeviceNet (Write with FC 5 and 15)	1st word	40.001	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	2nd word	40.002	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
	3rd word	40.003	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	Last word	40.128	2048	2047	2046	2045	2044	2043	2042	2041	2040	2039	2038	2037	2036	2035	2034	2033
Output data of the DeviceNet (Read with FC 1)	1st word	40.001	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	2nd word	40.002	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
	3rd word	40.003	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	Last word	40.128	2048	2047	2046	2045	2044	2043	2042	2041	2040	2039	2038	2037	2036	2035	2034	2033

Arrangement of the bit markers in the common process images

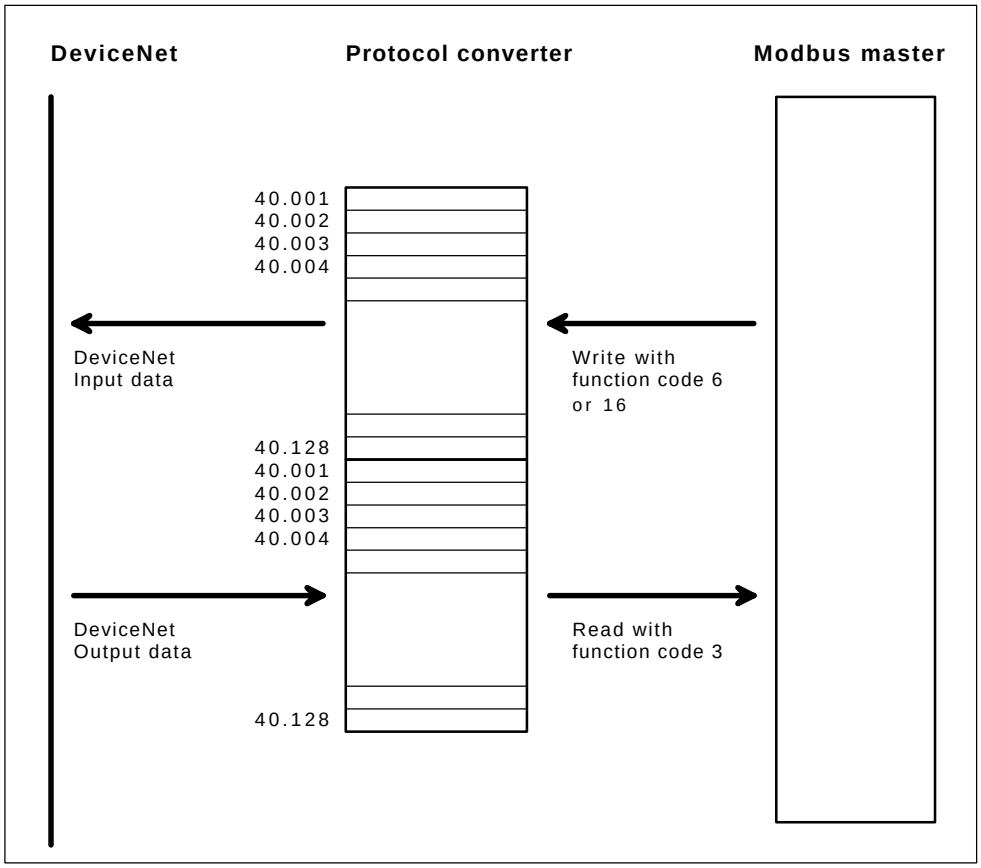
The access to the output data of the DeviceNet via the function code 2 thus begins at the Modbus address 10.001.

In contrast to controllers from the Modicon family, the registers and the coils (bit markers) in the protocol converter are situated above each other in the process image. Thus, word- or bit-wise access of the same data can be selected!

2.2 Data image from the viewpoint of the Modbus RTU Slave

If the protocol converter is a slave (server) at the Modbus RTU, then reading and writing can occur on the common memory with the functions of the Modbus RTU.

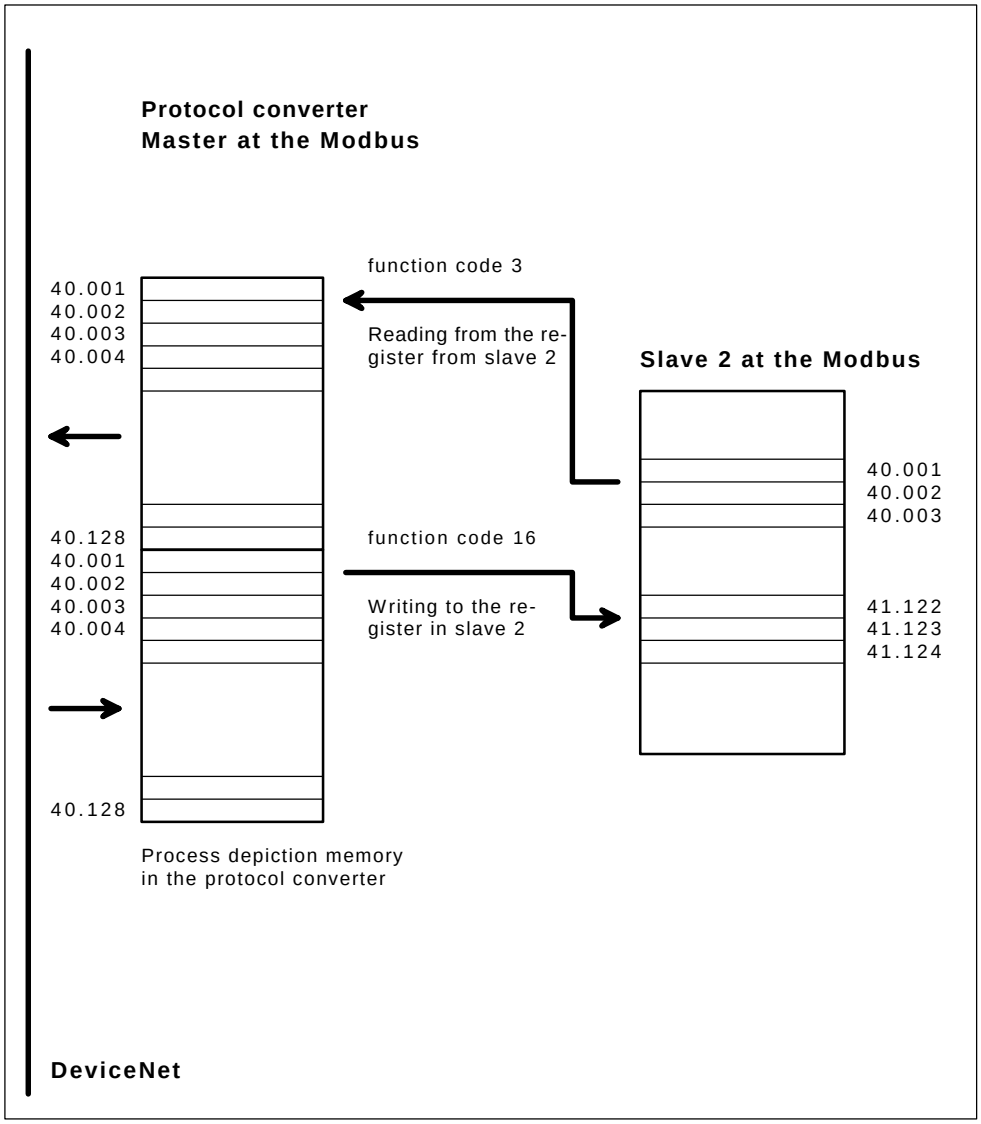
The data image between the addresses on the DeviceNet and the Modbus RTU is described with the following picture as an example of the function codes 3, 6 and 16 for the process image.



Data transfer between protocol converter as slave at the Modbus RTU and Modbus master

2.3 Data image from the viewpoint of the Modbus RTU master

If the protocol converter is a master at the Modbus RTU, then, on the basis of the commands defined in the COMMAND list, it independently generates read and write commands between the process image and the slaves connected to it.



Data exchange between the protocol converter and slave devices at the Modbus RTU

2.4 Status and error bits of the Modbus slave participants

If the protocol converter is working as a **master** at the Modbus, a status or error field can be utilized in the process image (Supervision). This must be configured in the SUPERVIS table. The following options are available:

Off	No status/error field is utilized. This means that all slave participants must work without error, otherwise the protocol converter closes down the communication at the DeviceNet.
CommandStatus	A separate status bit is utilized for each command of the COMMAND table. The bit is set when the command has been carried out without error. The Bit is deleted in case of error.
CommandError	An error bit is utilized for each command of the COMMAND table. The bit is set when an error has occurred in carrying out the command. The bit is deleted again when the next error-free command has been carried out.
SlaveStatus	A separate status bit is utilized for each slave. The bit is set when a command to this slave is carried out without error. The bit is deleted in the case of error.
SlaveError	An error bit is utilized for each slave. The bit is set when an error occurs during a command to this slave. The bit is deleted again when the next command has been carried out without error.

This defines whether a bit that is set in the process image denotes an error-free or an error transmission. Furthermore, the bits can represent alternatively a slave or a command. Thus the lowest value bit is allocated to the slave with the address 1 or to the first command in the COMMAND table.
The allocation of the bits is taken from the interpretation in the process image.

The starting address of the bit field in the process image is defined by means of the *Start Register* parameter. In the *Quantity Register* parameter is indicated how many registers are included in the status field. If a bit lies outside the defined field, then it is not considered. If the **quantity is set to 0,** then no status field is utilized.

The following interpretation shows which data is shown in the bit field with the various configurations.

As an example, the following configuration is assumed:

1st slave	Address 2
2nd slave	Address 3
3rd slave	Address 4
4th slave	Address 20

1st command	Read from 1st slave
2nd command	Read from 2nd slave
3rd command	Read from 3rd slave
4th command	Write to 3rd slave
5th command	Read from 4th slave

The first register for the bit field given is 40.010 (*Start Register* = 40.010) and two register is given as the length of the bit field (*Register Quantity* = 2); the 2nd slave is switched off:

	Register	Arrangement of the bit field in the process image														
Input data	40.009	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1st word	**40.010**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**
2nd word	**40.011**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	40.012	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Outputs in the bit field during supervision mode = CommandStatus

	Register															
Input data	40.009	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1st word	**40.010**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
2nd word	**40.011**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	40.012	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Outputs in the bit field during supervision mode = CommandError

	Register															
Input data	40.009	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1st word	**40.010**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
2nd word	**40.011**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
	40.012	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Outputs in the bit field during supervision mode = SlaveStatus

	Register															
Input data	40.009	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1st word	**40.010**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
2nd word	**40.011**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	40.012	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Outputs in the bit field during supervision mode = SlaveError

2.5 The Watchdog supervision

A Watchdog supervision is possible in the **master operation** of the Modbus RTU protocol. Here, the protocol converter reads out a particular register (Watchdog register) of a Modbus RTU slave with a cycling time that can be parametrized. If this read process is flawed, then the failure of the Modbus RTU or its connected devices is assumed. In this case **all** input data (with the exception for Supervision, see below) to the DeviceNet master are set to **zero.** When the Watchdog register is available again, then the inputs to the DeviceNet master are served again.

Remarks:

- If registers are reserved for the Slave supervision (status and error bits), then these are **not** altered!
- The Watchdog supervision is carried out cyclically with the cycling time parametrized in the *Watchdog time* in the WATCHDOG table. However the cycling time cannot be guaranteed for the following reasons:
    - It can occur that the Modbus slave used for the Watchdog supervision answers too late. The protocol converter waits, according to its setting (MODBUS table *Timeout* parameter), for example, for one second for the answer from the Modbus slave. This fact must be considered in the choice of the *Watchdog time*. Normally, the Watchdog supervision requires approx. 20..50ms for a baud rate of 9,600 baud.
    - A priority switching guarantees that, **at least** one command of the COMMAND table is processed after every Watchdog supervision. This means at the same time that the Watchdog Supervision cannot completely block the data transfer. However, the Modbus side data throughput can be greatly reduced if the *Watchdog time* is selected too be too short (e.g. 100 ms) as a Watchdog supervision is inserted between every command of the COMMAND table.

3 Configuration

The transfer parameters of the two protocols are defined by the configuration:

Modbus RTU	The parameters of the MODBUS table must be entered. This determines the whole behavior of the Modbus RTU.
DeviceNet	The configuration is carried out by means of the ComPro. Thus, only the data model needs to be defined in the BUS_DNS table. The Bus address is set on the address switches on the protocol converter.

For configuration, the protcol converter must be switched into the configuration and diagnostic mode.
This is described in the device manual of the PKV30-DNS.

Only a configuration as such is necessary for the protocol conversion, if the Modbus RTU protocol is to be operated in the master mode. Then the cyclical commands and commands of the protocol converter generated by the changes in the data must be defined in the COMMAND table. If a status or error field is to be utilized, then the SUPERVIS table must be defined. If the Watchdog supervision is required, it must be defined in the WATCHDOG table.

3.1 Editing the MODBUS table

Parameter	Meaning	Value range
Communication line	Gives the device interface that is served by the protocol.	**1**
RTS Control	If the RTS control is switched on, then the control lines RTS and CTS are served by an RS232C interface or by an RS485/RS422 interface that switches through the data drivers only while sending. This does not alter the protocol sequence.	**no** yes
Baud rate	Determines the rate of data transfer.	50 Baud 100 Baud 110 Baud 150 Baud 200 Baud 300 Baud 600 Baud 1200 Baud 2400 Baud 4800 Baud **9600 Baud** 19200 Baud
Stop bits	Defines the number of stop bits.	**1**, 2
Parity	Defines the parity bit.	none **even** odd
Mode	Defines the operating mode.	slave **master**
Modbus address	Gives the own address at the Modbus.	**1**,2...247
Timeout	Master mode: Gives the maximum waiting time in milliseconds for an answering telegram from a slave. Slave mode: Gives the maximum waiting time for an answer from the application.	10...**1000**...10000
Retries	Defines the number of retries of the telegram repeats in an error case. Relevant only in the Master operation.	0...**3**...10
Error-LED	This defines the operating mode of the error LEDs	**set/clear** only set

Parametrizing the Modbus RTU protocol in the MODBUS table.

3.2 Editing the BUS_DNS table

Parameter	Meaning	Value range
Slave MAC ID	Defines the DeviceNet address in the network.	**0** - 63
Vendor ID	DeviceNet specific manufacturer number, allocated to every manufacturer. Here: 283 for Hilscher; not changeable.	**283**
Prod.size	Defines the number of the input data in bytes.	0 - **8** - 255
Cons.size	Defines the number of output data in bytes.	0 - **8** - 255

Parametrizing the DeviceNet protocol in the BUS_DNS table

3.3 Editing the COMMAND table

This table is only relevant when the protocol converter operates as master on the Modbus RTU. A maximum of 300 commands can be defined. Each command consists of the following parameters. The default values are shown in bold and are underlined:

Parameter	Meaning	Value range
Slave	Gives the slave address from which, or into which the protocol converter conveys the data.	0 - **2** - 255
Function	Gives the function code for this access.	1 / 2 / **3** / 4 / 5 / 6 / 15 / 16
Address	Gives the address in the slave device.	1..9.999 10.001..19.999 30.001...39.999 **40.001**..49.999
Quantity	Gives the amount of data to be transmitted.	1 - **2** - 255
Mem.Addr.	Gives the address in the process image of the protocol converter in which the protocol converter stores the data or from which the protocol converter takes the data. The respective starting address (e.g. 40.001) is then always given at the start of the process image.	1..128 10.001..10.128 30.001..30.128 **40.001**..40.128
Write	Defines for all writing function codes (FC 5, 6, 15 and 16) whether the command is to be triggered cyclically or only by a change of data. This parameter has no significance for the remaining function codes.	**Cyclic** Change

Attention:
Addresses in accordance with Modbus convention are valid.

Defining the Master commands in the COMMAND table

When entering the parameters, care must be taken that these, in fact, address valid registers.

A delay time can be parametrized between the individual commands. This can be necessary in order to prevent the loading becoming too high on the connected Modbus slaves through uninterrupted communication.
The parametrizing is carried out in the SUPERVIS table.

3.4 Editing the SUPERVIS table

In this table, there is defined whether a bit field is utilized and if so, its function.

Parameter	Meaning	Value range
Supervision Mode	No bit field Status bit field for commands Error bit field for commands Status bit field for Slaves Error bit field for Slaves	**off** CommandStatus CommandError SlaveStatus SlaveError
Start Register	Start address of the bit field. The first 16 bits of the bit field are entered into this register	**40.001**..40.128
Quantity Register	Number of Registers of the Bit field. 0 = No bit field is entered.	**0**..128
Delay [msec]	Delay time between the individual commands in the Modbus master mode in milliseconds.	**0**..65535

Parameter list

If the Supervision mode is set to *off*, then, the DeviceNet master has **no possibility** to check the data exchange with the Modbus slaves. In the other modes, the bit field is transfered to the DeviceNet master. This is then responsible for the evaluation and display of the error condition at the Modbus.

3.5 Editing the WATCHDOG table

In this table the Watchdog supervision is parametrized.

Parameter	Meaning	Value range
Watchdog	off = No Watchdog supervision The following parameters then have no significance. on = Watchdog supervision active.	**off** on
Watchdog address	Address of the Modbus slave.	1..**2**..247
Watchdog Register	Modbus address of the Watchdog register	**40001**..49999
Watchdog time	Cycling time of the Watchdog supervision in milliseconds.	10..**1000**..60000

Parameter list

4 Error messages

The following tables show the error messages of the individual protocols. These can be displayed with the aid of the ComPro program. Errors are also displayed by means of two error LEDs on the protocol converter (see chapter *Setting up the converter as a Slave at the DeviceNet*).

4.1 Error messages from the Modbus RTU

Error number	Error
10	Serial interface occupied Interface occupied The serial interface has already been initialized by another task. Error in the “Interface” parameter.
11	Sum of all baudrates to high Sum of all baud rates has been exceeded. The sum of all baud rates on all initialized interfaces is too high.
12	Error 'Communication line' Error “Interface” Parametrized interface on the device is not available.
13	Error 'Baudrate' Error “Baudrate”. Invalid value for the “Baud rate” inititializing parameter.
14	Error 'Parity' Error “Parity” Invalid value for the “Parity” inititializing parameter.
15	Error 'Databits' Error “Data Bits” Invalid value for the “Data Bits” inititializing parameter.
16	Error 'Stopbits' Error “Stop Bits” Invalid value for the “Stop Bits” inititializing parameter.
17	Error 'RTS-Control' Error “RTS control” Invalid value for the "RTS-Control" inititializing parameter.
50	Error 'Mode' Error “Mode” Invalid value for the "Mode" inititializing parameter.
51	Error 'Modbus address' Error “Modbus address” Invalid value for the "Modbus address" inititializing parameter.
52	Error 'Timeout' Error “Timeout” Invalid value for the "Timeout" inititializing parameter.
53	Error 'Retries' Error “Retries” Invalid value for the "Retries" inititializing parameter.
54	Error 'Error-LED' Error “LED Operation”' Invalid value for the "Error-LED" inititializing parameter.

Initializing errors

Error number	Error
100	Parity error Parity error The interface controller has detected a parity error.
101	Framing error Character frame error The interface controller has detected a parity error.
102	Overrun error Loss of received data The interface controller has detected an “overrun” error..
103	To much/less data received Too much/too little data received. More data has been received than can be accommodated in the receiving buffer or too little to make a proper telegram testing possible.
104	CRC error CRC error A checksum error has been recognized in the receiving telegram.
105	Timeout telegram Time monitoring error. A time monitoring error has occurred as no answer has been received from the Slave within the defined Slave waiting time.
110	Unknown exception received Unknown exception received. a non-defined “exception response code”, e.g. 0 or greater than 9 has been received.
111 ... 119	Exception 1 ... 9 received Exception 1 ... 9 received The accessed Slave has answered with an “exception response code”.
120	Invalid 'slave address' received Invalid Slave address received The Slave address in the answering telegram is not that of the addressed Slave.
121	Invalid 'function code' received Invalid function code received. The function information in the received Modbus telegram does not correspond with the issued function.
122	Invalid 'bytecount' received Incorrect “bytecount” received. The displayed amount of data in the received Modbus telegram (bytecount) does not correspond with the amount of data transferred.
123	Too much/less data received Too little/ too much data received. Incorrect amount of data in the received Modbus telegram.
124	Invalid data address received Invalid data address received. Incorrect data address in the Modbus telegram.
125	Invalid answer data received Invalid answer data received. The answering telegram from the Slave has been correctly received but does not correspond with the command telegram.
126	Invalid diagnostic code received Invalid diagnostic code received. Only the diagnostic code 0, i.e. “loopback test” is permissible.

Communication errors

Error number	Meaning
151	Invalid length of message Invalid length of message The transferred amount of useful data for output as Modbus telegram is incorrect.
152	Unknown message command Unknown message command. The message command Msg.B is invalid.
154	Message error received Message error is set. In Slave mode, the coupling partner returns an error instead of the data asked for or a confirmation.
155	Timeout message Time monitoring error message. A time monitoring error has occurred as no answer has arrived from the application program within the configured Timeout period.
160	Invalid telegram header in acknowledge message A different telegram header in the answering message. The application Program has returned a different telegram head in the command confirmation.
161	Error 'device address' Error “device address” The Slave address in Msg.DeviceAdr was entered incorrectly by the application program.
162	Error 'data area' Error “data area”. The data area in Msg.DataArea was entered incorrectly by the application program.
163	Error 'data address' Error “data address”,
165	Error 'data quantity' Error “data quantity” The data address in Msg.DataAdr was entered incorrectly by the application program.
166	Error 'data type' Error “data type” The data type in Msg.DataType was entered incorrectly by the application program.
167	Error 'function' Error ”function” The function code in Msg.Function was entered incorrectly by the application program.

Protocol and message errors

4.2 Error messages of the DNSMBR bridge

Error number	Error
10	Serial interface occupied Interface occupied The serial interface has already been initialized by another task. Error in the “interface” parameter.
11	Sum of all baudrates to high The sum of all baud rates has been exceeded. The sum of all baud rates on all initialized interfaces is too high.
50	Modbus data base missing Error in opening the data base - table MODBUS.
51	Command error 'Function' Incorrect Modbus function code in COMMAND table.
52	Command error 'Quantity' Incorrect amount of data in COMMAND table.
53	Command error 'Range of Address/Quantity' COMMAND table: the combination of Modbus address “address” and the amount of data “quantity” has exceeded the permitted area.
54	Command error 'Mem.Add.' Incorrect address in the process image of the protocol converter in the COMMAND table.
55	Command error 'Address' Incorrect Modbus address in COMMAND table.
60	SUPERVIS data base missing Error when opening the database - SUPERVIS table.
61	WATCHDOG data base missing Error when opening the database - WATCHDOG table.

Initializing errors

Error number	Error
200	Task not initialized The task could not be initialized.
210	Error at opening the data base The parameter database is not available.
212	Error at reading the data base The parameter database is inconsistent.
213	System error 'RcsPutStructure' Internal error.

Internal system errors

5 Setting up the converter as a Slave at the DeviceNet

The following sequence must be adhered to in the set up of the protocol converter as a slave:

- A valid configuration database must be saved on the protocol converter. In the supply, the indicated default parameters have been set.
- The bus address must be set on the protocol converter at the address switches.
- The coupling partner on the Modbus-RTU must be connected. Please consult the device manual for configuring the physical interface and fitting the cable.
- Configuring the DeviceNet master with the aid of the included EDS file. In the master configuration, exactly the same information must be input as in the BUS_DNS table.
- Connecting the DeviceNet cable and the supply voltage.
- The LED RDY and RUN must light up and must not blink.
- The LEDs signal the following conditions:

Display	Color	Condition	Meaning
RDY	yellow	on blinks cyclically blinks irregularly off	PKV ready Bootstraploader active Hardware or system error Hardware defective
RUN	green	on blinks irregularly blinks regularly off	Communication running Parametrizing error Ready for communication No communication
NET	red	on	Critical link failure
		blinking	Connection time out
		off	Device not powered
	green	on	On-line, link OK
		blinking	On-line, not connected
		off	Device not powered
MOD	red	on	Unrecoverable fault
		blinking	Minor fault
		off	No power
	green	on	
		blinking	Configuration failure
		off	No Power

LED indications of the PKV30-DNS