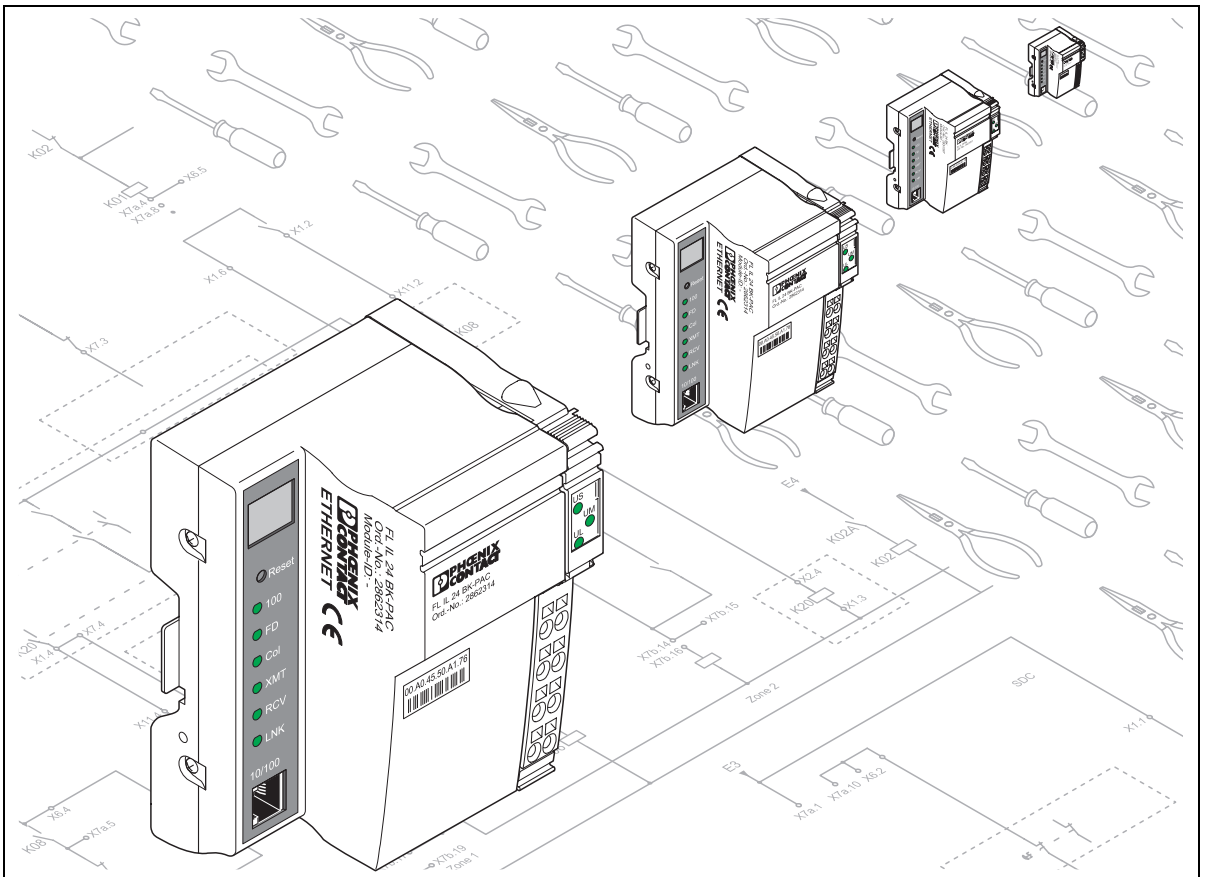


User Manual

Hardware and Firmware User Manual for the
FL IL 24 BK / FL IL 24 BK-PAC
Ethernet/Inline Bus Coupler

Designation: FL IL 24 BK-PAC UM E

Order No.: 90 14 20 5



Factory Line

User Manual

Hardware and Firmware User Manual

FL IL 24 BK / FL IL 24 BK-PAC Ethernet/Inline Bus Coupler

Designation: FL IL 24 BK-PAC UM E

Revision: 05

Order No.: 90 14 20 5

This user manual is valid for:

FL IL 24 BK

FL IL 24 BK-PAC

© Phoenix Contact 09/2004

615605



Please Observe the Following Notes:

In order to ensure the safe use of your device, we recommend that you read this manual carefully. The following notes provide information on how to use this manual.

Requirements of the User Group

The use of products described in this manual is oriented exclusively to qualified electricians or persons instructed by them, who are familiar with applicable national standards. Phoenix Contact assumes no liability for erroneous handling or damage to products from Phoenix Contact or external products resulting from disregard of information contained in this manual.

Explanation of Symbols Used



The *note* symbol informs you of conditions that must strictly be observed to achieve error-free operation. It also gives you tips and advice on the efficient use of hardware and on software optimization to save you extra work.



The *attention* symbol refers to an operating procedure which, if not carefully followed, could result in damage to equipment or personal injury.



The *text* symbol refers to detailed sources of information (manuals, data sheets, literature, etc.) on the subject matter, product, etc. This text also provides helpful information for the orientation in the manual.

Statement of Legal Authority

This manual, including all illustrations contained herein, is copyright protected. Use of this manual by any third party deviating from the copyright provision is forbidden. Reproduction, translation, or electronic or photographic archiving or alteration requires the express written consent of Phoenix Contact. Violators are liable for damages.

Phoenix Contact reserves the right to make any technical changes that serve the purpose of technical progress.

Phoenix Contact reserves all rights in the case of patent award or listing of a registered design. Third-party products are always named without reference to patent rights. The existence of such rights shall not be excluded.



Warning

The FL IL 24 BK(-PAC) module is designed exclusively for SELV operation according to IEC 60950/EN 60950/VDE 0805.



Shielding

The shielding ground of the twisted pair cables that can be connected is electrically connected with the socket. When connecting network segments, avoid ground loops, potential transfers, and voltage equalization currents using the braided shield.



ESD

The modules are fitted with electrostatically sensitive components. Exposure to electric fields or charge imbalance may damage or adversely affect the life of the modules.

The following safety equipment must be used when using electrostatically sensitive modules:

Create an electrical equipotential bonding between yourself and your surroundings, e.g., using an ESD wristband, which is connected to the grounded DIN rail to which the module will be connected.



Housing

Only authorized Phoenix Contact personnel are permitted to open the housing.

About This Manual

Purpose of this manual	This manual illustrates how to configure an Ethernet/Inline station to meet application requirements.
Who should use this manual	Use this manual if you are responsible for configuring and installing an Ethernet/Inline station. This manual is written based on the assumption that the reader possesses basic knowledge about Inline systems.
Related documentation	For specific information on the individual Inline terminals see the corresponding terminal-specific data sheets.
Latest documentation on the Internet	Make sure you always use the latest documentation. Changes in or additional information on present documentation can be found on the Internet at www.phoenixcontact.com or www.factoryline.de . The homepages are updated daily. You can also contact us by sending an e-mail to factoryline-service@phoenixcontact.com .

Orientation in this manual

For easy orientation when looking for specific information the manual offers the following help:

- The manual starts with the main table of contents that gives you an overview of all manual topics.
- Each manual section starts with an overview of the section topics.
- On the left side of the pages within the sections you will see the topics that are covered in the section.
- In the Appendix you will find a list of figures and a list of tables.

This user manual includes

In the first section you are introduced to Inline basics and general information that applies to all terminals or terminal groups of the Inline product range. Topics are, for example:

- Overview of the Inline product groups
- Terminal structure
- Terminal installation and wiring
- Common technical data

Validity of documentation

Phoenix Contact reserves the right to make any technical extensions and changes to the system that serve the purpose of technical progress. Up to the time that a new manual revision is published, any updates or changes will be documented on the Internet at www.phoenixcontact.com or www.factoryline.de.

Table of Contents

1	FL IL 24 BK(-PAC).....	1-3
1.1	General Functions	1-3
1.1.1	Product Description	1-3
1.2	Structure of the FL IL 24 BK(-PAC) Bus Coupler	1-5
1.2.1	Local Status and Diagnostic Indicators	1-6
1.3	Connecting the Supply Voltage	1-7
1.4	Connector Assignment	1-8
1.5	Supported Inline Modules.....	1-9
1.6	Basic Structure of Low-Level Signal Modules	1-16
1.6.1	Electronics Base	1-17
1.6.2	Connectors	1-17
1.7	Function Identification and Labeling	1-21
1.8	Dimensions of Low-Level Signal Modules.....	1-24
1.9	Electrical Potential and Data Routing	1-27
1.10	Circuits Within an Inline Station and Provision of the Supply Voltages.....	1-29
1.10.1	Supply of the Ethernet Bus Coupler	1-30
1.10.2	Logic Circuit U_L	1-30
1.10.3	Analog Circuit U_{ANA}	1-31
1.10.4	Main Circuit U_M	1-32
1.10.5	Segment Circuit	1-33
1.11	Voltage Concept	1-35
1.12	Diagnostic and status indicators.....	1-42
1.12.1	LEDs on the Ethernet Bus Coupler	1-42
1.12.2	Indicators on the Supply Terminal	1-44
1.12.3	Indicators on the Input/Output Modules	1-45
1.12.4	Indicators on Other Inline Modules	1-46
1.13	Mounting/Removing Modules and Connecting Cables.....	1-47
1.13.1	Installation Instructions	1-47
1.13.2	Mounting and Removing Inline Modules	1-47
1.13.3	Mounting	1-48
1.13.4	Removing	1-50
1.13.5	Replacing a Fuse	1-52

1.14	Grounding an Inline Station	1-54
1.14.1	Shielding an Inline Station	1-56
1.14.2	Shielding Analog Sensors and Actuators	1-56
1.15	Connecting Cables	1-59
1.15.1	Connecting Unshielded Cables	1-59
1.15.2	Connecting Shielded Cables Using the Shield Connector ...	1-61
1.16	Connecting the Voltage Supply	1-64
1.16.1	Power Terminal Supply	1-64
1.16.2	Provision of the Segment Voltage Supply at Power Terminals	1-65
1.16.3	Demands on the Power Supply Units	1-65
1.17	Connecting Sensors and Actuators	1-66
1.17.1	Connection Methods for Sensors and Actuators	1-66
1.17.2	Connection Examples for Digital I/O Modules	1-67
2	Startup/Operation	2-3
2.1	Default Upon Delivery/Default Settings	2-3
2.2	Firmware Start	2-3
2.3	Transmitting BootP Requests	2-4
2.4	Assigning an IP Address Using the Factory Manager	2-4
2.4.1	BootP	2-4
2.5	Manual Addition of Devices Using The Factory Manager	2-5
2.6	Selecting IP Addresses	2-5
2.6.1	Possible Address Combinations	2-7
2.6.2	Subnet Masks	2-8
2.6.3	Structure of the Subnet Mask	2-8
2.7	Web-Based Management	2-10
2.7.1	Calling Web-Based Management (WBM)	2-10
2.7.2	Structure of the Web Pages	2-11
2.7.3	Layout of the Web Pages	2-11
2.7.4	Password Protection	2-12
2.7.5	Process Data Access Via XML	2-12
2.8	Firmware Update	2-17
2.8.1	Firmware Update Using The Factory Manager	2-17

2.9	Firmware Update Using Web-Based Management (WBM) Without Factory Manager	2-19
2.9.1	Trap Generation	2-19
2.9.2	Representation of Traps in the Factory Manager	2-19
2.9.3	FL IL 24 BK(-PAC) Traps	2-20
2.9.4	Defining the Trap Manager	2-20
2.10	Factory Line I/O Configurator	2-21
2.10.1	Factory Line I/O Browser	2-21
2.10.2	OPC Configurator	2-22
2.11	Management Information Base - MIB	2-24
2.11.1	Standard MIBs:	2-24
2.12	Interface Group (1.3.6.1.2.1.2)	2-25
2.12.1	Private MIBs	2-30
2.13	Meaning of the 7-Segment Display	2-44
3	Driver Software	3-3
3.1	Documentation	3-3
3.1.1	Hardware and Firmware User Manual	3-3
3.2	Software Structure	3-3
3.2.1	Ethernet/Inline Bus Coupler Firmware	3-4
3.2.2	Driver Software	3-4
3.3	Support and Driver Update	3-5
3.4	Transfer of I/O Data	3-6
3.4.1	Position of the Process Data (Example)	3-7
3.5	Startup Behavior of the Bus Coupler	3-8
3.5.1	Plug & Play Mode	3-8
3.5.2	Expert Mode	3-9
3.5.3	Possible Combinations of the Modes	3-9
3.5.4	Startup Diagrams of the Bus Coupler	3-10
3.5.5	Changing and Starting a Configuration in P&P Mode	3-12
3.6	Changing a Reference Configuration Using the Software	3-13
3.6.1	Effects of Expert Mode	3-13
3.6.2	Changing a Reference Configuration	3-13
3.7	Description of the Device Driver Interface (DDI) Introduction	3-15
3.7.1	Overview	3-15
3.7.2	Working Method of the Device Driver Interface	3-16
3.7.3	Description and Functions of the Device Driver Interface	3-18

3.8	Monitoring Functions	3-33
3.8.1	Process Data Monitoring/ Process Data Watchdog	3-35
3.8.2	Connection Monitoring (Host Checking)	3-37
3.8.3	Data Interface (DTI) Monitoring	3-40
3.8.4	I/O Fault Response Mode	3-44
3.8.5	Treatment of the NetFail Signal/Testing With ETH_SetNetFail	3-46
3.9	IN Process Data Monitoring.....	3-52
3.10	Notification Mode	3-56
3.11	Programming Support Macros.....	3-58
3.11.1	Introduction	3-58
3.12	Description of the Macros.....	3-60
3.12.1	Macros for Converting the Data Block of a Command	3-61
3.12.2	Macros for Converting the Data Block of a Message	3-63
3.12.3	Macros for Converting Input Data	3-65
3.12.4	Macros for Converting Output Data	3-66
3.13	Diagnostic Options for Driver Software	3-68
3.13.1	Introduction	3-68
3.14	Positive Messages.....	3-69
3.15	Error Messages	3-70
3.15.1	General Error Messages	3-70
3.15.2	Error Messages When Opening a Data Channel	3-72
3.15.3	Error Messages When Transmitting Messages/Commands	3-73
3.15.4	Error Messages When Transmitting Process Data	3-75
3.16	Example Program.....	3-78
3.16.1	Demo Structure Startup	3-78
3.16.2	Example Program Source Code	3-80
4	Firmware Services.....	4-3
4.1	Overview.....	4-3
4.1.1	Services Available in Both Operating Modes	4-3
4.1.2	Services Available Only in Expert Mode	4-4
4.2	Notes on Service Descriptions	4-4
4.2.1	"Name_of_the_Service" Service	4-5
4.3	Services for Parameterizing the Controller Board	4-7
4.3.1	"Control_Parameterization" Service	4-7

4.3.2	"Set_Value" Service	4-9
4.3.3	"Read_Value" Service	4-11
4.3.4	"Initiate_Load_Configuration" Service	4-13
4.3.5	"Load_Configuration" Service	4-15
4.3.6	"Terminate_Load_Configuration" Service	4-18
4.3.7	"Read_Configuration" Service	4-20
4.3.8	"Complete_Read_Configuration" Service	4-26
4.3.9	"Delete_Configuration" Service	4-29
4.3.10	"Create_Configuration" Service	4-30
4.3.11	"Activate_Configuration" Service	4-32
4.3.12	"Control_Device_Function" Service	4-34
4.3.13	"Reset_Controller_Board" Service	4-36
4.4	Services for Direct INTERBUS Access	4-38
4.4.1	"Start_Data_Transfer" Service	4-38
4.4.2	"Alarm_Stop" Service	4-40
4.5	Diagnostic Services	4-41
4.5.1	"Get_Error_Info" Service	4-41
4.5.2	"Get_Version_Info" Service	4-47
4.6	Error Messages for Firmware Services	4-50
4.6.1	Overview	4-50
4.6.2	Positive Messages	4-51
4.6.3	Error Messages	4-51
5	PCP Communication	5-3
5.1	Transmission of Parameter Data	5-3
5.1.1	PCP Configuration in the Web-Based Management	5-4
5.1.2	Configuration of the PCP PDU Size	5-4
5.2	Supported PCP Commands	5-5
6	Modbus/TCP Protocol	6-3
6.1	Modbus Protocol	6-3
6.1.1	Modbus Connections	6-3
6.1.2	Modbus Port	6-4
6.1.3	Modbus Conformance Classes	6-4
6.1.4	Modbus Message Format	6-4
6.1.5	Modbus Byte Order	6-4
6.1.6	Modbus Bit Order	6-5
6.2	Modbus Function Codes	6-5

Table of Contents

6.3	Modbus Table	6-5
6.3.1	Dynamic Modbus/TCP Process Data Table	6-6
6.3.2	Example: Location of the Input/Output Data	6-7
6.3.3	Location of the Process Data in Dynamic Tables	6-8
6.4	Applicable Functions	6-9
6.5	Supported Function Codes	6-9
6.5.1	Read Multiple Registers	6-10
6.5.2	Write Multiple Registers	6-11
6.5.3	Read Coils	6-12
6.5.4	Read Input Discretes	6-13
6.5.5	Read Input Registers	6-14
6.5.6	Write Coils	6-15
6.5.7	Write Single Register	6-16
6.5.8	Read Exception Status	6-17
6.5.9	Exception Status Data Format	6-17
6.5.10	Exception Responses	6-18
6.5.11	Write Multiple Coils	6-19
6.5.12	Read/Write Register	6-20
6.6	Reserved Registers for Command and Status Words	6-22
6.6.1	Command Word	6-22
6.6.2	Status Word	6-23
6.6.3	Diagnostics Using the Analog Input Table	6-23
6.6.4	Fault Table	6-24
6.7	Monitoring	6-25
6.8	Modbus Monitoring	6-26
6.9	I/O Fault Response Mode	6-26
6.9.1	Power Up Table	6-27
6.9.2	Connection Monitoring Table	6-28
6.10	Modbus/TCP PCP Registers	6-30
7	Technical Data	7-3
7.1	Ordering Data	7-11

This section provides information about

- the basic structure of low-level signal modules
- the assignment and meaning of the diagnostic and status indicators on the various Inline modules
- potential and data routing
- housing dimensions and labeling options for the modules
- general information on the module circuit diagrams

FL IL 24 BK(-PAC).....	1-3
1.1 General Functions	1-3
1.1.1 Product Description	1-3
1.2 Structure of the FL IL 24 BK(-PAC) Bus Coupler	1-5
1.2.1 Local Status and Diagnostic Indicators.....	1-6
1.3 Connecting the Supply Voltage	1-7
1.4 Connector Assignment	1-8
1.5 Supported Inline Modules.....	1-9
1.6 Basic Structure of Low-Level Signal Modules	1-16
1.6.1 Electronics Base	1-17
1.6.2 Connectors	1-17
1.7 Function Identification and Labeling	1-21
1.8 Dimensions of Low-Level Signal Modules	1-24
1.9 Electrical Potential and Data Routing	1-27
1.10 Circuits Within an Inline Station and Provision of the Supply Voltages.....	1-29
1.10.1 Supply of the Ethernet Bus Coupler	1-30
1.10.2 Logic Circuit U_L	1-30
1.10.3 Analog Circuit U_{ANA}	1-31
1.10.4 Main Circuit U_M	1-32
1.10.5 Segment Circuit	1-33
1.11 Voltage Concept	1-35
1.12 Diagnostic and status indicators.....	1-42
1.12.1 LEDs on the Ethernet Bus Coupler.....	1-42
1.12.2 Indicators on the Supply Terminal	1-44
1.12.3 Indicators on the Input/Output Modules	1-45
1.12.4 Indicators on Other Inline Modules	1-46

1.13	Mounting/Removing Modules and Connecting Cables.....	1-47
1.13.1	Installation Instructions	1-47
1.13.2	Mounting and Removing Inline Modules.....	1-47
1.13.3	Mounting	1-48
1.13.4	Removing.....	1-50
1.13.5	Replacing a Fuse	1-52
1.14	Grounding an Inline Station	1-54
1.14.1	Shielding an Inline Station	1-56
1.14.2	Shielding Analog Sensors and Actuators	1-56
1.15	Connecting Cables	1-59
1.15.1	Connecting Unshielded Cables	1-59
1.15.2	Connecting Shielded Cables Using the Shield Connector	1-61
1.16	Connecting the Voltage Supply	1-64
1.16.1	Power Terminal Supply.....	1-64
1.16.2	Provision of the Segment Voltage Supply at Power Terminals	1-65
1.16.3	Demands on the Power Supply Units	1-65
1.17	Connecting Sensors and Actuators	1-66
1.17.1	Connection Methods for Sensors and Actuators	1-66
1.17.2	Connection Examples for Digital I/O Modules	1-67

1 FL IL 24 BK(-PAC)

1.1 General Functions

1.1.1 Product Description

Ethernet/Inline bus coupler

Features

- Ethernet coupler for the Inline I/O system
- Ethernet TCP/IP
 - 10/100 Base-T(X)
 - Management via SNMP
 - Integrated web server
- Modbus/TCP protocol
- DDI (Device Driver Interface) protocol software interface
- Up to 63 additional Inline modules can be connected (process data channel)
up to eight PCP modules can be connected
- Flexible installation system for Ethernet
- IP parameter setting via BootP, web-based management (WBM) or SNMP
- Driver software for Sun Solaris/Windows NT/2000
- Software interface kit for other Unix systems

Applications

- Connection of sensors/actuators via Ethernet

Exchange of Inline process data via Ethernet using a Unix workstation or a Windows NT/2000 computer.

Front view of the FL IL 24 BK(-PAC)

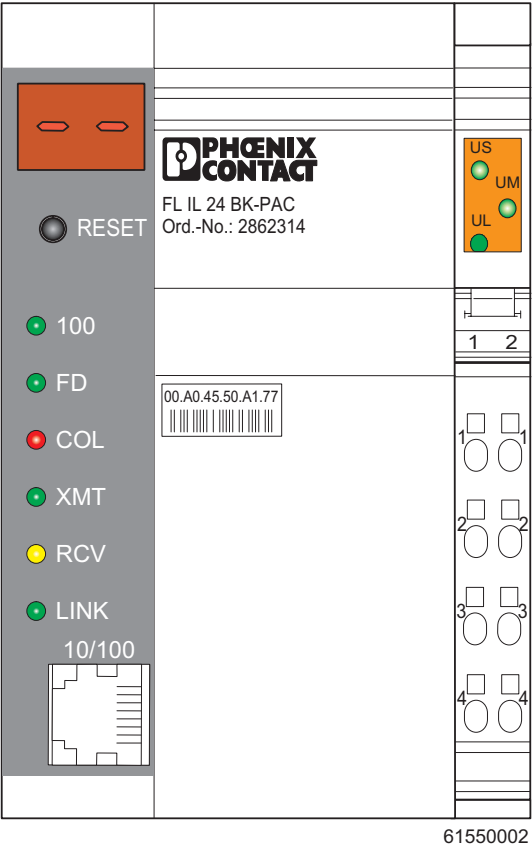


Figure 1-1 Front view of the FL IL 24 BK(-PAC)

1.2 Structure of the FL IL 24 BK(-PAC) Bus Coupler

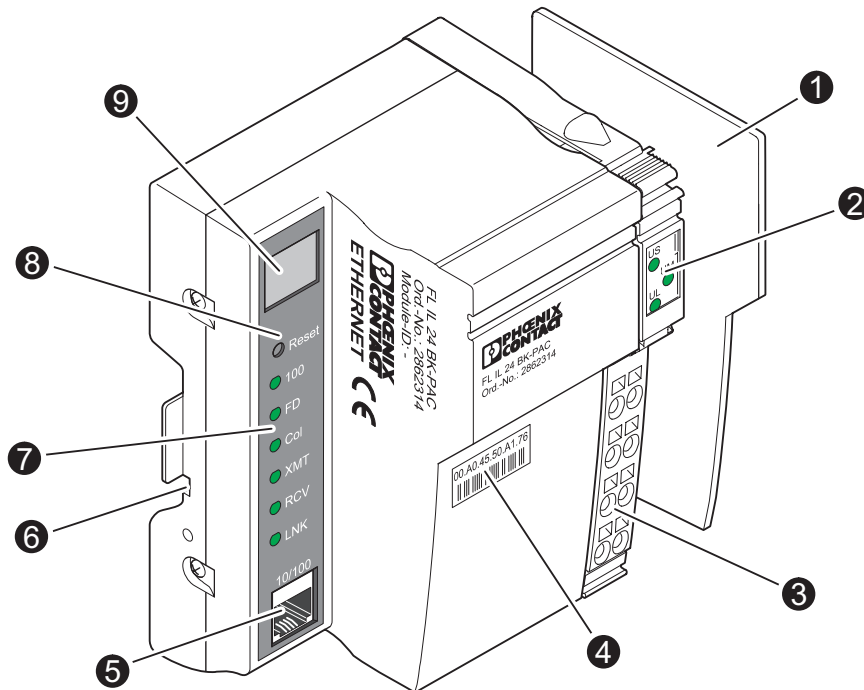


Figure 1-2 Structure of the FL IL 24 BK(-PAC) bus coupler

The bus coupler has the following components:

- 1 End plate to protect the last Inline module
- 2 Inline diagnostic indicators
- 3 24 V DC supply and functional earth ground connector (not supplied as standard - order as an accessory)
- 4 MAC address in clear text and as a barcode
- 5 Ethernet interface (twisted pair cables in RJ45 format)
- 6 Two FE contacts for grounding the bus coupler using a DIN rail (on the back of the module)
- 7 Ethernet status and diagnostic indicators
- 8 Reset button
- 9 7-segment display for the device status (Ethernet communication unit)

1.2.1 Local Status and Diagnostic Indicators

Table 1-1 Local status and diagnostic indicators

Des.	Color	Status	Meaning
Module Electronics			
UL	Green	ON	24 V supply, 7 V communications power/interface supply present
		OFF	24 V supply, 7 V communications power/interface supply not present
UM	Green	ON	24 V main circuit supply present
		OFF	24 V main circuit supply not present
US	Green	ON	24 V segment supply present
		OFF	24 V segment supply not present
Ethernet Port			
100	Green	ON	Operation at 100 Mbps
		OFF	Operation at 10 Mbps (if LNK LED active)
FD	Green	ON	Data transmission in full duplex mode
		OFF	Data transmission in half duplex mode (if LNK LED active)
COL	Red	ON	Collision of data telegrams
		OFF	Transmission of telegrams without a collision (if LNK LED active)
XMT	Green	ON	Data telegrams are being sent
		OFF	Data telegrams are not being sent
RCV	Yellow	ON	Data telegrams are being received
		OFF	Data telegrams are not being received
LNK	Green	ON	Physical network connection ready to operate
		OFF	Physical network connection interrupted or not present

Reset button

The reset button is on the front plate. When the reset button is pressed the bus coupler is completely initialized and booted. Inline system outputs are reset and inputs are not read.

1.3 Connecting the Supply Voltage

The module is operated using a +24 V DC SELV.

Typical Connection of the Supply Voltage

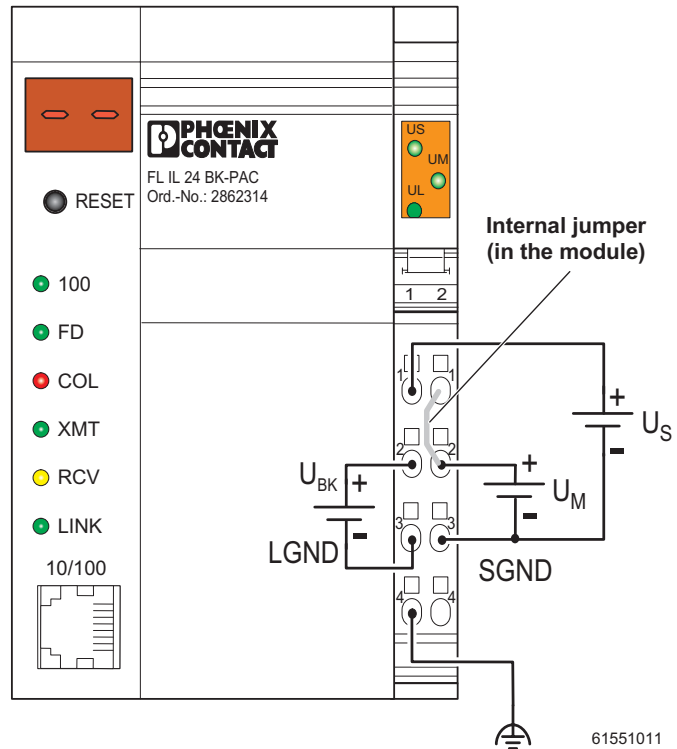


Figure 1-3 Typical connection of the supply voltage

1.4 Connector Assignment

Table 1-2 Connector assignment

Terminal Point/ Connector	Assignment/Power Connector		Wire Color/Remark
1.1	24 V DC (U_S)	24 V segment supply	The supplied voltage is directly led to the potential jumper.
1.2	24 V DC (U_{BK})	24 V supply	The communications power for the bus coupler and the connected local bus devices is generated from this power. The 24 V analog power (U_{ANA}) for the local bus devices is also generated.
2.1, 2.2	24 V DC (U_M)	Main voltage	The main voltage is routed to the local bus devices via the potential jumpers.
1.3	LGND	Reference potential logic ground for U_{BK}	The potential is the reference ground for the communications power U_{BK} .
2.3	SGND	Reference potential for U_S and U_M	The reference potential is directly led to the potential jumper and is, at the same time, ground reference for the main and segment supply.
1.4, 2.4	FE	Functional earth ground (FE)	The functional earth ground must be connected to the 24 V DC supply/functional earth ground connection. The contacts are directly connected to the potential jumper and FE springs on the bottom of the housing. The terminal is grounded when it is snapped onto a grounded DIN rail. Functional earth ground is only used to discharge interference.



The GND potential jumper carries the total current from the main and segment circuits. The total current must not exceed the maximum current carrying capacity of the potential jumper (8 A). If the 8 A limit is reached at one of the potential jumpers U_S , U_M , and GND during configuration, a new power terminal must be used.



Functional earth ground must be connected through the 24 V DC supply/functional earth ground connection.

1.5 Supported Inline Modules

Table 1-3 Digital Input/Output Modules

Designation	Properties	Order No.
IB IL 24 DI 2	2 inputs, 4-wire termination, 24 V DC	27 26 20 1
IB IL 24 DI 2-PAC	2 inputs, 4-wire termination, 24 V DC	28 61 22 1
IB IL 24 DI 2-NPN	2 inputs with negative logic, 4-wire termination, 24 V DC	27 40 11 2
IB IL 24 DI 2-NPN-PAC	2 inputs with negative logic, 4-wire termination, 24 V DC	28 61 48 3
IB IL 24 EDI 2	2 inputs, 4-wire termination, with electronic overload protection and diagnostics	27 42 60 9
IB IL 24 EDI 2-PAC	2 inputs, 4-wire termination, with electronic overload protection and diagnostics	28 61 62 9
IB IL 24 EDI 2-DESINA	2 inputs, 4-wire termination according to Desina specification, with electronic overload protection and diagnostics	27 40 32 6
IB IL 24 EDI 2-DESINA-PAC	2 inputs, 4-wire termination according to Desina specification, with electronic overload protection and diagnostics	28 61 52 2
IB IL 24 DI 4	4 inputs, 3-wire termination, 24 V DC	27 26 21 4
IB IL 24 DI 4-PAC	4 inputs, 3-wire termination, 24 V DC	28 61 23 4
IB IL 24 DI 8	8 inputs, 4-wire termination, 24 V DC	27 26 22 7
IB IL 24 DI 8-PAC	8 inputs, 4-wire termination, 24 V DC	28 61 24 7
IB IL 24 DI 8 T2	8 inputs, 4-wire termination, 24 V DC, according to EN 61131-2 Type 2	28 60 43 9
IB IL 24 DI 8 T2-PAC	8 inputs, 4-wire termination, 24 V DC, according to EN 61131-2 Type 2	28 62 20 4
IB IL 24 DI 16	16 inputs, 3-wire termination, 24 V DC	27 26 23 0
IB IL 24 DI 16-PAC	16 inputs, 3-wire termination, 24 V DC	28 61 25 0
IB IL 24 DI 16-NPN	16 inputs with negative logic, 3-wire termination, 24 V DC	28 63 51 7
IB IL 24 DI 16-NPN-PAC	16 inputs with negative logic, 3-wire termination, 24 V DC	28 63 52 0
IB IL 24 DI 32/HD	32 inputs, 1-wire termination, 24 V DC	28 60 78 5
IB IL 24 DI 32/HD-PAC	32 inputs, 1-wire termination, 24 V DC	28 62 83 5
IB IL 120 DI 1	1 input, 3-wire termination, 120 V AC	28 36 70 6
IB IL 120 DI 1-PAC	1 input, 3-wire termination, 120 V AC	28 61 91 7
IB IL 230 DI 1	1 input, 3-wire termination, 230 V AC	27 40 34 2
IB IL 230 DI 1-PAC	1 input, 3-wire termination, 230 V AC	28 61 54 8
IB IL 24 DO 2	2 outputs, 500 mA, 4-wire termination, 24 V DC	27 40 10 6

Table 1-3 Digital Input/Output Modules

Designation (Contd.)	Properties	Order No.
IB IL 24 DO 2-PAC	2 outputs, 500 mA, 4-wire termination, 24 V DC	28 61 47 0
IB IL 24 DO 2-2A	2 outputs, 2 A, 4-wire termination, 24 V DC	27 26 24 3
IB IL 24 DO 2-2A-PAC	2 outputs, 2 A, 4-wire termination, 24 V DC	28 61 26 3
IB IL 24 DO 2-NPN	2 outputs with negative logic, 500 mA, 4-wire termination, 24 V DC	27 40 11 9
IB IL 24 DO 2-NPN-PAC	2 outputs with negative logic, 500 mA, 4-wire termination, 24 V DC	28 61 49 6
IB IL 24 EDO 2	2 outputs, 500 mA, 4-wire termination, 24 V DC, extended diagnostics, configurable outputs	27 42 59 9
IB IL 24 EDO 2-PAC	2 outputs, 500 mA, 4-wire termination, 24 V DC, extended diagnostics, configurable outputs	28 61 61 6
IB IL 24 DO 4	4 outputs, 500 mA, 3-wire termination, 24 V DC	27 26 25 6
IB IL 24 DO 4-PAC	4 outputs, 500 mA, 3-wire termination, 24 V DC	28 61 27 6
IB IL 24 DO 8	8 outputs, 500 mA, 4-wire termination, 24 V DC	27 26 26 9
IB IL 24 DO 8-PAC	8 outputs, 500 mA, 4-wire termination, 24 V DC	28 61 28 9
IB IL 24 DO 8-NPN	8 outputs with negative logic, 500 mA, 4-wire termination, 24 V DC	28 63 54 6
IB IL 24 DO 8-NPN-PAC	8 outputs with negative logic, 500 mA, 4-wire termination, 24 V DC	28 63 53 3
IB IL 24 DO 8-2A	8 outputs, 2 A, 4-wire termination, 24 V DC	27 42 11 7
IB IL 24 DO 8-2A-PAC	8 outputs, 2 A, 4-wire termination, 24 V DC	28 61 60 3
IB IL 24 DO 16	16 outputs, 500 mA, 3-wire termination, 24 V DC	27 26 27 2
IB IL 24 DO 16-PAC	16 outputs, 500 mA, 3-wire termination, 24 V DC	28 61 29 2
IB IL 24 DO 32/HD	32 outputs, 500 mA, 1-wire termination, 24 V DC	28 60 93 4
IB IL 24 DO 32/HD-PAC	32 outputs, 500 mA, 1-wire termination, 24 V DC	28 62 82 2
IB IL DO 1 AC	1 output, 12 V - 253 V AC, 500 mA, 3-wire termination	28 36 74 8
IB IL DO 1 AC-PAC	1 output, 12 V - 253 V AC, 500 mA, 3-wire termination	28 61 92 0
IB IL DO 4 AC-1A	1 output, 12 V - 253 V AC, 1 A, 3-wire termination	27 42 69 6
IB IL DO 4 AC-1A-PAC	1 output, 12 V - 253 V AC, 1 A, 3-wire termination	28 61 65 8
IB IL 24/230 DOR 1/W	1 SPDT relay contact, 5 V - 253 V AC, 3 A	28 36 43 4
IB IL 24/230 DOR 1/W-PAC	1 SPDT relay contact, 5 V - 253 V AC, 3 A	28 61 88 1
IB IL 24/230 DOR 1/W-PC	1 SPDT relay contact, 5 V - 253 V AC, 3 A, for inductive and capacitive loads	28 60 40 0
IB IL 24/230 DOR 1/W-PC-PAC	1 SPDT relay contact, 5 V - 253 V AC, 3 A, for inductive and capacitive loads	28 62 17 8

Table 1-3 Digital Input/Output Modules

Designation (Contd.)	Properties	Order No.
IB IL 24/230 DOR 4/W	4 SPDT relay contacts, 5 V - 253 V AC, 3 A	28 36 42 1
IB IL 24/230 DOR 4/W-PAC	4 SPDT relay contacts, 5 V - 253 V AC, 3 A	28 61 87 8
IB IL 24/230 DOR 4/W-PC	4 SPDT relay contacts, 5 V - 253 V AC, 3 A, for inductive and capacitive loads	28 60 41 3
IB IL 24/230 DOR 4/W-PC-PAC	4 SPDT relay contacts, 5 V - 253 V AC, 3 A, for inductive and capacitive loads	28 62 18 1
IB IL 24/48 DOR/2W	2 SPDT relay contacts, 5 V - 50 V AC, 5 V - 120 V DC, 2 A	28 62 97 4
IB IL 24/48 DOR/2W-PAC	2 SPDT relay contacts, 5 V - 50 V AC, 5 V - 120 V DC, 2 A	28 63 11 9

Table 1-4 Analog input/output modules

Designation	Properties	Order No.
IB IL AI 2/4-20	2 inputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA	28 60 44 2
IB IL AI 2/4-20-PAC	2 inputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA	28 62 21 7
IB IL AI 2/SF	2 inputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA, ± 40 mA	27 26 28 5
IB IL AI 2/SF-PAC	2 inputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA, ± 40 mA	28 61 30 2
IB IL AI 2/SF-230	2 inputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA, ± 40 mA, 230 Hz	27 40 81 8
IB IL AI 2/SF-230-PAC	2 inputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA, ± 40 mA, 230 Hz	28 61 57 7
IB IL AI 8/SF	8 inputs, 2-wire termination, 24 V DC, 0 - 5 V, 0 - 10 V, ± 10 V, 0 - 25 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA	27 27 83 1
IB IL AI 8/SF-PAC	8 inputs, 2-wire termination, 24 V DC, 0 - 5 V, 0 - 10 V, ± 10 V, 0 - 25 V, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA	28 61 41 2
IB IL AI 8/IS	8 inputs, 3-wire termination, 24 V DC, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA, ± 40 mA	27 42 74 8
IB IL AI 8/IS-PAC	8 inputs, 3-wire termination, 24 V DC, 0 - 20 mA, 4 - 20 mA, ± 20 mA, 0 - 40 mA, ± 40 mA	28 61 66 1
IB IL AI 2-HART	2 inputs, 2-wire termination 24 V DC, 0 - 25 mA, 4 - 20 mA, HART functionality, HART protocol transmission	28 60 26 4
IB IL AI 2-HART-PAC	2 inputs, 2-wire termination 24 V DC, 0 - 25 mA, 4 - 20 mA, HART functionality, HART protocol transmission	28 62 14 9

Table 1-4 Analog input/output modules

Designation (Contd.)	Properties	Order No.
IB IL TEMP 2 RTD	2 inputs, 4-wire termination, 16 bits, resistance sensors	27 26 30 8
IB IL TEMP 2 RTD-PAC	2 inputs, 4-wire termination, 16 bits, resistance sensors	28 61 32 8
IB IL TEMP 2 RTD/300	2 inputs, 4-wire termination, 16 bits, resistance sensors	27 40 76 6
IB IL TEMP 2 RTD/300-PAC	2 inputs, 4-wire termination, 16 bits, resistance sensors	28 61 55 1
IB IL TEMP 2 UTH	2 inputs, 2-wire termination, 16 bits, thermocouples	27 27 76 3
IB IL TEMP 2 UTH-PAC	2 inputs, 2-wire termination, 16 bits, thermocouples	28 61 38 6
IB IL TEMPCON RTD	Multi-channel temperature controller, 6 inputs/6 outputs	28 19 24 4
IB IL TEMPCON RTD-PAC	Multi-channel temperature controller, 6 inputs/6 outputs	28 61 77 1
IB IL TEMPCON UTH	8 inputs, 8 outputs, controller functions	28 19 31 2
IB IL TEMPCON UTH-PAC	8 inputs, 8 outputs, controller functions	28 61 80 7
IB IL AO 1/SF	1 output, 2-wire termination, 24 V DC, 0-20 mA, 4-20 mA, 0-10 V	27 26 29 8
IB IL AO 1/SF-PAC	1 output, 2-wire termination, 24 V DC, 0-20 mA, 4-20 mA, 0-10 V	28 61 31 5
IB IL AO 1/U/SF	1 output, 2-wire termination, 24 V DC, 0-10 V	27 27 77 6
IB IL AO 1/U/SF-PAC	1 output, 2-wire termination, 24 V DC, 0-10 V	28 61 39 9
IB IL AO 2/SF	2 outputs, 2-wire termination, 24 V DC, 0-20 mA, 4-20 mA, 0-10 V	28 62 80 6
IB IL AO 2/SF-PAC	2 outputs, 2-wire termination, 24 V DC, 0-20 mA, 4-20 mA, 0-10 V	28 63 08 3
IB IL AO 2/U/BP	2 outputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V	27 32 73 2
IB IL AO 2/U/BP-PAC	2 outputs, 2-wire termination, 24 V DC, 0 - 10 V, ± 10 V	28 61 46 7

Table 1-5 Special function modules

Designation	Properties	Order No.
IB IL SSI	1 absolute encoder input, 4 digital inputs, 4 digital outputs, 500 mA, 3-wire termination, 24 V DC	28 36 34 0
IB IL SSI-PAC	1 absolute encoder input, 4 digital inputs, 4 digital outputs, 500 mA, 3-wire termination, 24 V DC	28 61 86 5
IB IL SSI-IN	1 absolute encoder input, 24 V DC	28 19 30 9
IB IL SSI-IN-PAC	1 absolute encoder input, 24 V DC	28 19 57 4

Table 1-5 Special function modules

Designation	Properties	Order No.
IB IL INC	1 incremental encoder input, 4 digital inputs, 4 digital outputs, 500 mA, 3-wire termination, 24 V DC	28 36 32 4
IB IL INC-PAC	1 incremental encoder input, 4 digital inputs, 4 digital outputs, 500 mA, 3-wire termination, 24 V DC	28 61 84 9
IB IL INC-IN	1 incremental encoder input with square-wave signal, 1 digital signal for reference signal, 2 digital inputs, 24 V DC	28 19 22 8
IB IL INC-IN-PAC	1 incremental encoder input with square-wave signal, 1 digital signal for reference signal, 2 digital inputs, 24 V DC	28 61 75 5
IB IL CNT	1 counter input, 1 control input, 1 digital output, 500 mA, 3-wire termination, 24 V DC	28 36 33 7
IB IL CNT-PAC	1 counter input, 1 control input, 1 digital output, 500 mA, 3-wire termination, 24 V DC	28 61 85 2
IB IL IMPULSE IN	1 input for magnetostrictive length measuring systems with pulse interface	28 19 23 1
IB IL IMPULSE IN-PAC	1 input for magnetostrictive length measuring systems with pulse interface	28 61 85 2
IB IL POS 200	Inline positioning control	28 19 33 8
IB IL POS 200-PAC	Inline positioning control including accessories	28 61 82 3
IB IL RS 232	Terminal for serial data transmission via RS 232	27 27 34 9
IB IL RS 232-PAC	Terminal for serial data transmission via RS 232	28 61 35 7
IB IL RS 485/422	Terminal for serial data transmission via RS 485/422	28 36 79 3
IB IL RS 485/422-PAC	Terminal for serial data transmission via RS 485/422	28 61 93 3
ASI MA IB IL	AS-i/master	27 41 28 8

Table 1-6 Motor terminal blocks

Designation	Properties	Order No.
IB IL 24 TC	Thermistor terminal	27 27 41 7
IB IL 24 TC-PAC	Thermistor terminal	28 61 36 0
IB IL 400 ELR 1-3A	Electronic direct starter, 1.5 kW (2.01 hp), 400 V AC	27 27 35 2
IB IL 400 ELR R-3A	Electronic reversing-load starter, 1.5 kW (2.01 hp), 400 V AC	27 27 37 8
IB IL 400 MLR 1-8A	Electromechanical direct starter, 3.7 kW (4.96 hp), 400 V AC	27 27 36 5
IB IL DC AR 48/10A	Servo amplifier for DC motors with brushgears	28 19 28 6
IB IL EC AR 48/10A	Servo amplifier for DC motor without brushgears (EC motor)	28 19 25 7

Table 1-6 Motor terminal blocks

Designation	Properties	Order No.
IB IL EC AR 48/10A-PAC	Servo amplifier for DC motor without brushgears (EC motor)	28 19 58 7
IB IL PWM/2	Terminal for pulse width modulation and frequency modulation or stepper motor control, 2 outputs for 5 V or 24 V	27 42 61 2
IB IL PWM/2-PAC	Terminal for pulse width modulation and frequency modulation or stepper motor control, 2 outputs for 5 V or 24 V	28 61 63 2

Table 1-7 Power and segment terminals

Designation	Properties	Order No.
IB IL 24 PRW IN	Power terminal, 24 V DC	27 26 31 1
IB IL 24 PRW IN-PAC	Power terminal, 24 V DC	28 61 33 1
IB IL 24 PRW IN/F	Power terminal, 24 V DC with fuse	27 27 90 9
IB IL 24 PRW IN/F-PAC	Power terminal, 24 V DC with fuse	28 61 43 8
IB IL 24 PRW IN/F-D	Power terminal, 24 V DC with fuse and diagnostics	28 36 66 7
IB IL 24 PRW IN/F-D-PAC	Power terminal, 24 V DC with fuse and diagnostics	28 61 89 4
IB IL 24 PRW IN/2-F	Power terminal, 24 V DC with fuse	28 60 01 5
IB IL 24 PRW IN/2-F-PAC	Power terminal, 24 V DC with fuse	28 62 13 6
IB IL 24 PRW IN/2-F-D	Power terminal, 24 V DC with fuse and diagnostics	28 60 28 0
IB IL 24 PRW IN/2-F-D-PAC	Power terminal, 24 V DC with fuse and diagnostics	28 62 15 2
IB IL 24 PWR IN/M	Power terminal, 24 V DC	28 61 02 7
IB IL 24 PWR IN/R	Power terminal, 24 V DC	27 42 76 4
IB IL 24 PWR IN/R-PAC	Power terminal, 24 V DC	28 61 67 4
IB IL 120 PRW IN	Power terminal, 120 V AC with fuse	27 31 70 4
IB IL 120 PRW IN-PAC	Power terminal, 120 V AC with fuse	28 61 45 4
IB IL 230 PRW IN	Power terminal, 230 V AC with fuse	27 40 33 9
IB IL 230 PRW IN-PAC	Power terminal, 230 V AC with fuse	28 61 53 5
IB IL 24 SEG	Segment terminal, 24 V DC	27 26 32 4
IB IL 24 SEG-PAC	Segment terminal, 24 V DC	28 61 34 4
IB IL 24 SEG/F	Segment terminal, 24 V DC with fuse	27 27 74 7
IB IL 24 SEG/F-PAC	Segment terminal, 24 V DC with fuse	28 61 37 3
IB IL 24 SEG/F-D	Segment terminal, 24 V DC with fuse and diagnostics	28 36 68 3
IB IL 24 SEG/F-D-PAC	Segment terminal, 24 V DC with fuse and diagnostics	28 61 90 4
IB IL 24 SEG-ELF	Segment terminal, 24 V DC with electronic fuse	27 27 78 9

Table 1-7 Power and segment terminals

Designation	Properties	Order No.
IB IL 24 SEG-ELF-PAC	Segment terminal, 24 V DC with electronic fuse	28 61 40 9
IB IL PD GND	Terminal for GND potential distribution	28 63 06 7
IB IL PD GND-PAC	Terminal for potential distribution (GND)	28 62 99 0
IB IL PD 24V	Terminal for potential distribution (main voltage)	28 63 05 4
IB IL PD 24V-PAC	Terminal for potential distribution (main voltage)	28 62 98 7

Table 1-8 Safety

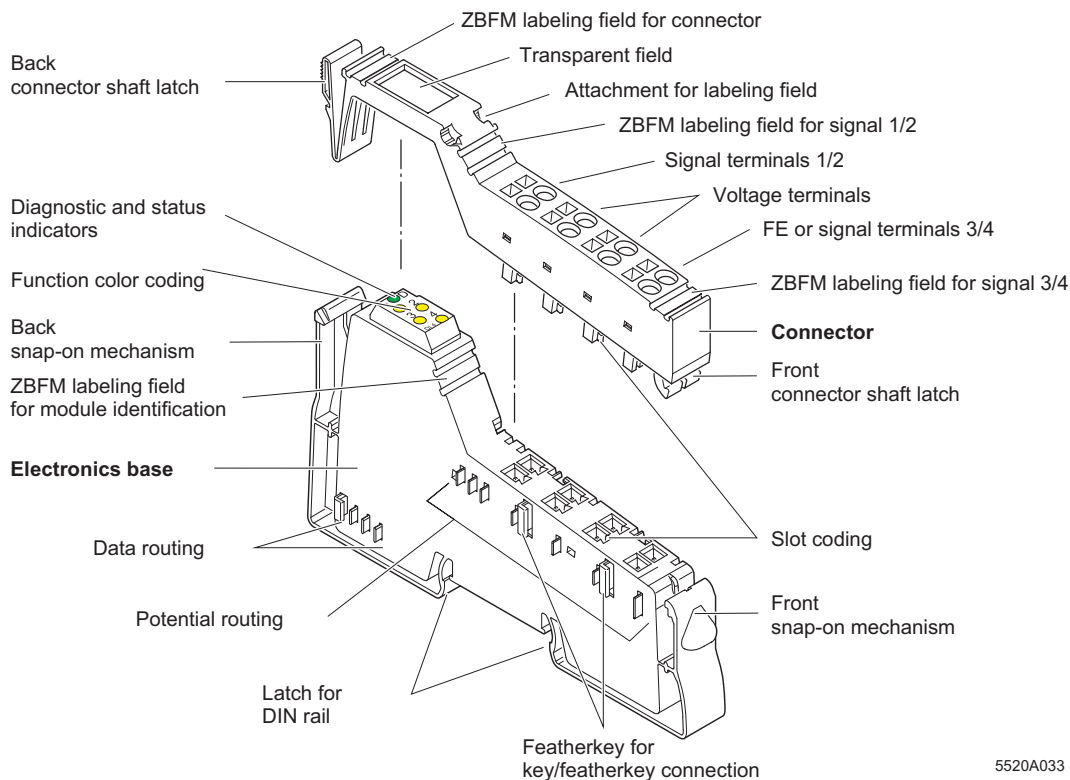
Designation	Properties	Order No.
IB IL 24 SAFE 1	Safety terminal with 2 N/O contacts	27 40 78 2
IB IL 24 SAFE 1-PAC	Safety terminal with 2 N/O contacts	28 61 56 4
IB IL 400 SAFE 2	Safety terminal with 3 N/O contacts and 1 N/C contact, 200 V AC to 600 V AC	27 40 79 5

Table 1-9 Controller / CPU

Designation	Properties	Order No.
ILC 200 UNI	Inline Controller with INTERBUS local bus interface	27 30 90 9
ILC 200 UNI-PAC	Inline Controller with INTERBUS local bus interface	28 62 29 1
IB IL 332-128	Inline CPU	28 19 13 4
IB IL 332-128-PAC	Inline CPU	28 61 73 9
IB IL 332-256	Inline-CPU with 256 kbytes RAM, 1 MB flash, serial interface	28 19 32 5
IB IL 332-256-PAC	Inline-CPU with 256 kbytes RAM, 1 MB flash, serial interface	28 61 81 0

1.6 Basic Structure of Low-Level Signal Modules

Regardless of the function and the design width, an Inline low-level signal module consists of the electronics base (or base for short) and the plug-in connector (or connector for short).



5520A033

Figure 1-4 Basic structure of an Inline module

The most important components shown in Figure 1-4 are described in sections "Electronics Base" on page 1-17 and "Connectors" on page 1-17.

ZBFM: Zack marker strips, flat
(see also Section "Function Identification and Labeling" on page 1-21)



The components required for labeling are listed in the Phoenix Contact "CLIPLINE" catalog.

1.6.1 Electronics Base

The electronics base holds the entire electronics for the Inline module and the potential and data routing.

Design widths

The electronics bases for low-level signal modules are available in a width of 8 terminal points (8-slot terminal) or 2 terminal points (2-slot terminal). Exceptions are combinations of these two basic terminal widths (see also Section "Dimensions of Low-Level Signal Modules" on page 1-24).

1.6.2 Connectors

The I/O or supply voltages are connected using a pluggable connector.

Advantages

This pluggable connection offers the following advantages:

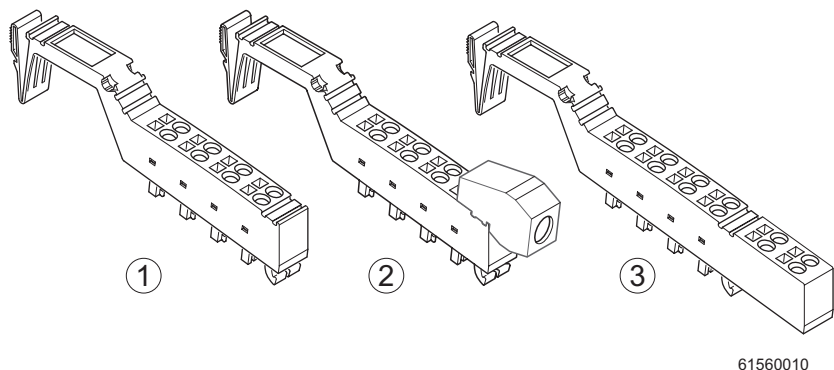
- Simple exchange of module electronics for servicing. There is no need to remove the wiring.
- Different connectors can be used on one electronics base, depending on your requirements.

Connector width

Regardless of the width of the electronics base, the connectors have a width of two terminal points. This means that you must plug 1 connector on a 2-slot base, 2 connectors on a 4-slot base, and 4 connectors on an 8-slot base.

Connector types

The following connector types are available:



61560010

Figure 1-5 Inline connector types

1 Standard connector

The green standard connector is used for the connection of two signals in 4-wire technology (e.g., digital I/O signals).

The black standard connector is used for supply terminals. The adjacent contacts are jumpered internally (see Figure 1-6 on page 1-19).

2 Shield connector

This green connector is used for signals connected using shielded cables (e.g., analog I/O signals).

FE or shielding is connected via a shield connection clamp rather than via a terminal point.

3 Extended double signal connector

This green connector is used for the connection of four signals in 3-wire technology (e.g., digital I/O signals).

**Connector
identification**

All connectors are offered with and without color print. The connectors with color print (marked with CP in the Order Designation) have terminal points that are color-coded according to their functions.

The following colors indicate the signals of the terminal points:

Table 1-10 Terminal point color-coding

Color	Terminal Point Signal
Red	+
Blue	-
Green/yellow	Functional earth ground

Internal structure of the connectors

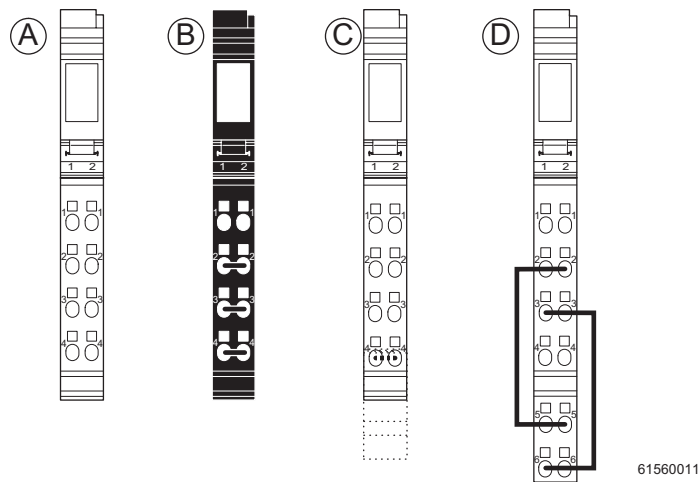


Figure 1-6 Internal structure of the connectors

- A Green connector for I/O connection
- B Black connector for supply terminals
- C Shield connector for analog terminals
- D Double signal connector for I/O connection

Jumpered terminal points already integrated in the connectors are shown in Figure 1-6.

The shield connector is jumpered through the shield connection. All other connectors are jumpered through terminal point connection.



To avoid a malfunction, only snap a connector on a terminal that is appropriate for this connection. Refer to the module-specific data sheet to select the correct connectors.



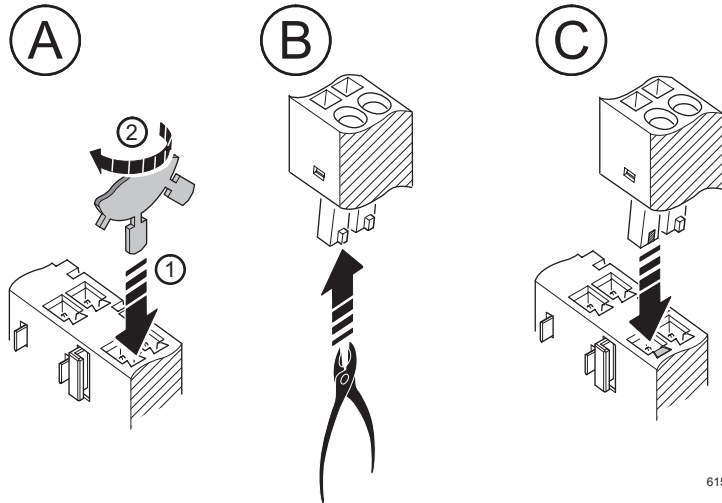
The black connector must not be placed on a module for which a double signal connector is to be used. Mixing this up leads to a short circuit between two signal terminal points (1.4 - 2.4).



Only snap black connectors onto a supply terminal. When the terminal points are jumpered in the black connector, power is carried through the jumpering in the connector and not through the printed circuit board of the module.

Connector keying

You can prevent mismatching of connectors by keying the base and the connector.



61560012

Figure 1-7 Connector keying

- Plug a keying profile (disc) into the keyway in the base (1) and turn it away from the small plate (2) (Figure 1-7, detail A).
- Use a diagonal cutter to cut off the keying tab from the connector (Figure 1-7, detail B).

Now, only the base and connector with the same keying will fit together (Figure 1-7, detail C).

1.7 Function Identification and Labeling

Function identification

The modules are color-coded to enable visual identification of the functions (1 in Figure 1-8).

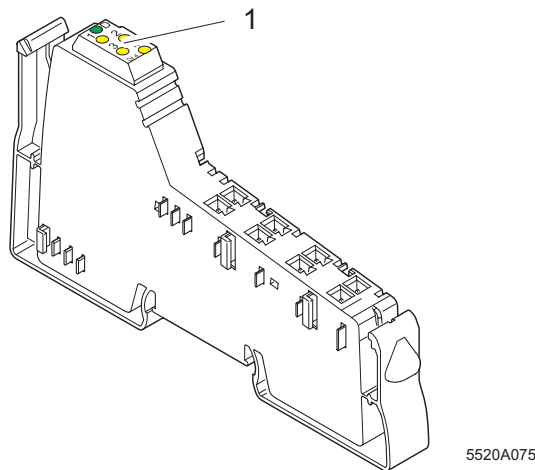


Figure 1-8 Function identification

The following colors indicate the functions:

Table 1-11 Module color-coding

Color	Function of the Module
Light blue	Digital input 24 V DC area
Pink	Digital output 24 V DC area
Blue	Digital input 120/230 V AC area
Red	Digital output 120/230 V AC area
Green	Analog input
Yellow	Analog output
Orange	Fieldbus coupler, special function modules
Black	Power terminal/segment terminal

Connector identification

The color-coding of the terminal points is described on page 1-18.

Labeling/ terminal numbering

Terminal point numbering is illustrated using the example of an 8-slot module.

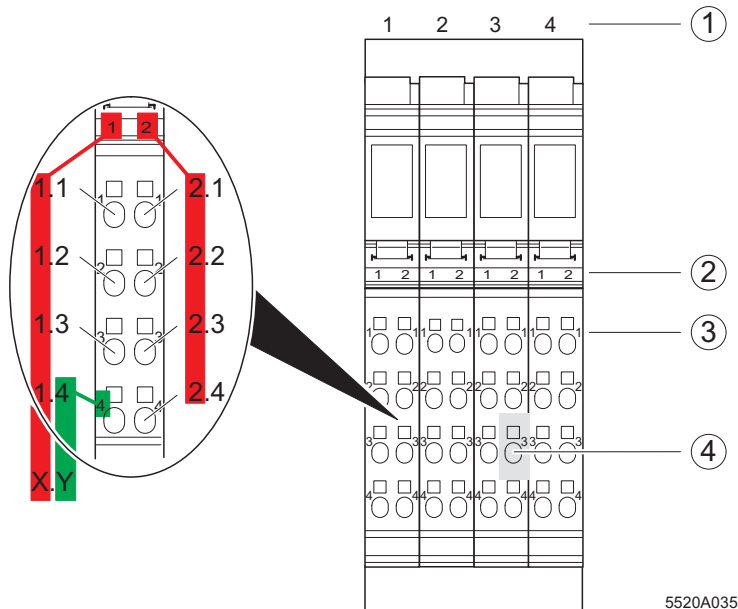


Figure 1-9 Terminal point numbering

Slot/connector

The slots (connectors) on a base are numbered consecutively (1 in Figure 1-9). This numbering is **not** shown on the actual module.

Terminal point

The terminal points on each connector are marked X.Y.

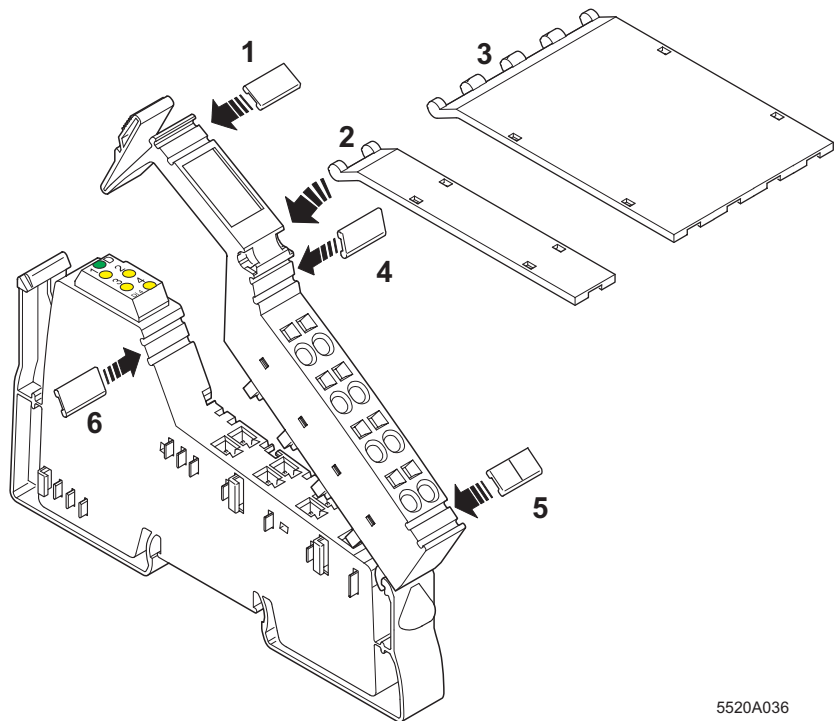
X is the number of the terminal point row on the connector. It is indicated above the terminal point row (2 in Figure 1-9).

Y is the terminal point number in a row. It is directly indicated on the terminal point (3 in Figure 1-9).

The precise designation for a connection point is thus specified by the slot and terminal point. The highlighted terminal point (4 in Figure 1-9) would be numbered as follows: slot 3, terminal point 2.3.

Additional labeling

In addition to this module marking, you can identify the slots, terminal points, and connections using marker strips and labeling fields.



5520A036

Figure 1-10 Labeling of modules

Various options are available for labeling the slots and terminal points:

- 1 Each connector can be labeled individually with Zack marker strips.
- 2 / 3 Another option is to use a large labeling field. This labeling field is available in two widths, either as a labeling field covering one connector (2) or as a labeling field covering four connectors (3). You can label each channel individually with free text. On the upper connector head there is a keyway for attaching this labeling field. The labeling field can be tilted up and down. In each end position there is a small latch which ensures that the labeling field remains in place.
- 4 / 5 Each signal can be labeled individually using Zack markers. On a double signal connector, the upper keyway (4) is designed for labeling signals 1/2 and the lower keyway (5) is for signals 3/4.
- 6 On the electronics base each slot can be labeled individually using Zack markers. These markers are covered when a connector is plugged in.

Using the markers on the connector and on the electronics base, you can clearly assign both connector and slot.



The components required for labeling are listed in the Phoenix Contact "CLIPLINE" catalog.

1.8 Dimensions of Low-Level Signal Modules

Today, small I/O stations are frequently installed in 80 mm (3.150 in.) standard control boxes. Inline modules are designed so that they can be used in this type of control box.

The housing dimensions of a module are determined by the dimensions of the electronics base and the dimensions of the connector.

The electronics bases of the low-level signal modules are available in three design widths (12.2 mm [0.480 in.], 24.4 mm [0.961 in.] and 48.8 mm [1.921 in.]).

It accepts either one, two, or four 12.2 mm (0.480 in.) wide connectors.

When a connector is plugged in, each terminal depth is 71.5 mm (2.815 in.).

The height of the module depends on the connector used. The connectors are available in three different versions (see Figure 1-14).

2-slot housing

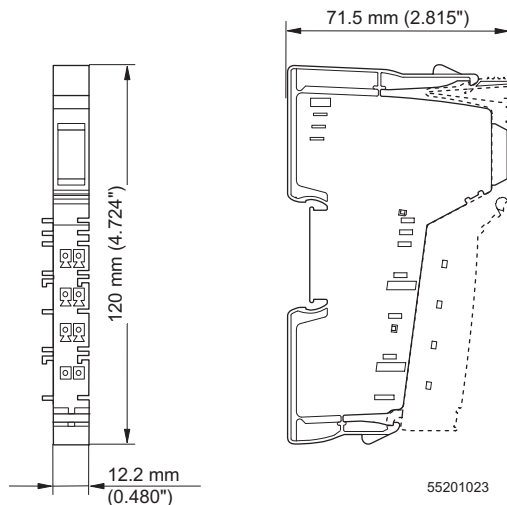


Figure 1-11 Dimensions of the electronics bases (2-slot housing)

4-slot housing

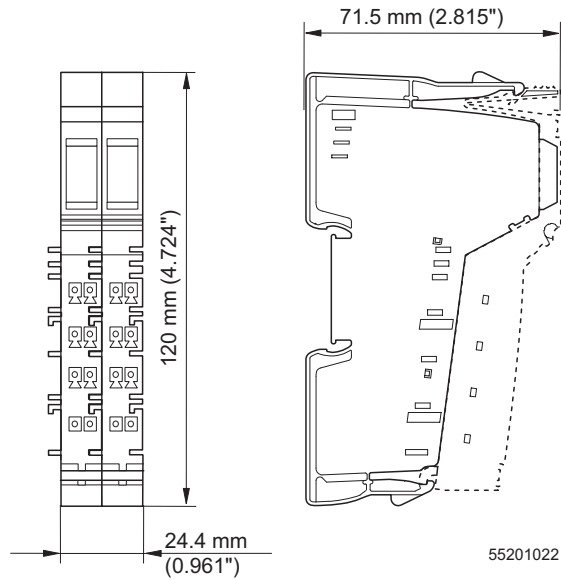


Figure 1-12 Dimensions of the electronics bases (4-slot housing)

8-slot housing

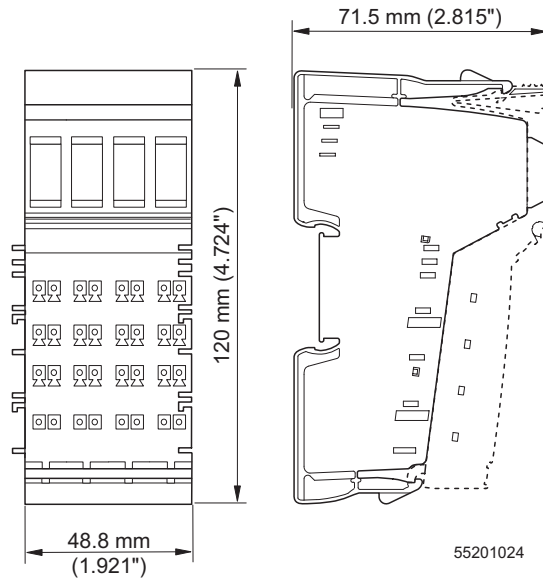


Figure 1-13 Dimensions of the electronics bases (8-slot housing)

Connector

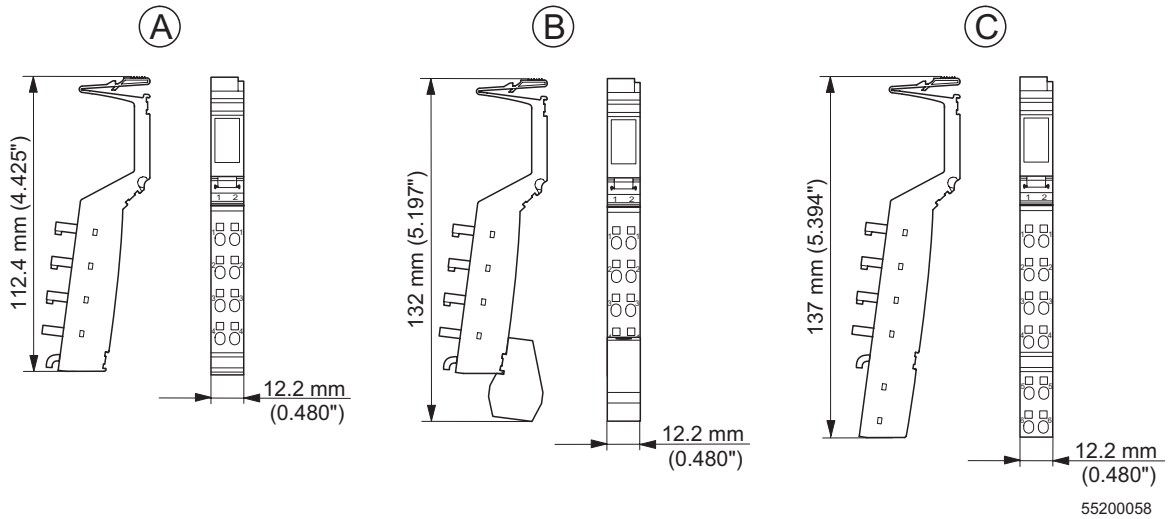


Figure 1-14 Connector dimensions

Key:

- A Standard connector
- B Shield connector
- C Extended double signal connector

The depth of the connector does not influence the overall depth of the module.

1.9 Electrical Potential and Data Routing

An important feature of the INTERBUS Inline and Ethernet/Inline bus coupler product ranges is their internal potential routing system. The electrical connection between the individual station devices is created automatically when the station is installed. When the individual station devices are connected, a power rail is created for the relevant circuit. It is created mechanically through the interlocking of knife and featherkey contacts on the adjacent modules.

A special segment circuit eliminates the need for additional external potential jumpering to neighboring modules.

Two independent circuits are created in one station: the logic circuit and the I/O circuit.

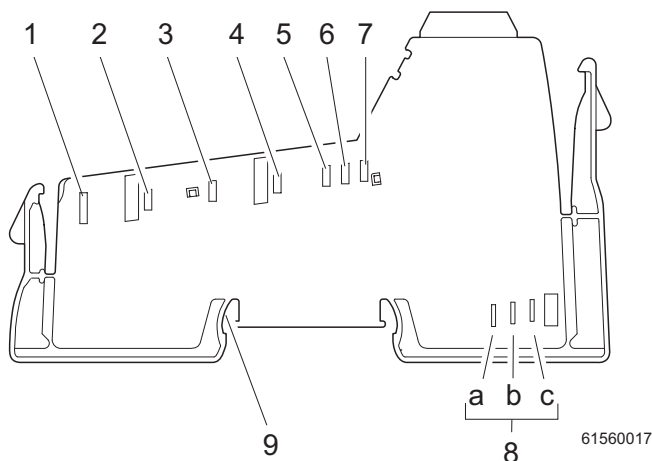


Figure 1-15 Potential and data routing

Table 1-12 Potential jumper (see Figure 1-15)

No.	Function		Meaning
1	FE	FE	Functional earth ground
2	SGND	SGND	Ground of segment supply and main supply
3	24 V	U_M	Supply for main circuit (with overload protection, if necessary)
4	24 V	U_S	Supply for segment circuit (with overload protection, if necessary) This jumper does not exist in the 120/230 V AC power levels.
5	LGND	U_{L-}	Ground of communications power and I/O supply for analog modules
6	24 V	U_{ANA}	I/O supply for analog modules
7	7.5 V	U_{L+}	Supply for module electronics
(9)	FE spring		FE contact to DIN rail



The GND potential jumper carries the total current from the main and segment circuits. The total current must not exceed the maximum current carrying capacity of the potential jumper (8 A). If the 8 A limit is reached at one of the potential jumpers U_S , U_M , and GND during configuration, a new power terminal must be used.



The FE potential jumper must be connected via terminal point 1.4 or 2.4 at the Ethernet bus coupler to a grounding terminal (see Figure 1-9). The FE potential jumper is led through all of the modules and connected via the FE spring to the grounded DIN rail of every supply terminal.

Table 1-13 Data jumper (see Figure 1-15)

No.	Function		Meaning
8a	DI1		Local bus signal (Data IN)
8b	DO1		Local bus signal (Data OUT)
8c	DCLK		Clock signal, local bus

1.10 Circuits Within an Inline Station and Provision of the Supply Voltages

There are several circuits within an Inline station. These are automatically set up when the modules have been properly installed. The voltages of the different circuits are supplied to the connected modules via the potential jumpers.



Load capacity of the jumper contacts

Please refer to the module-specific data sheet for the circuit to which the I/O circuit of a special module is to be connected.

Observe the maximum current carrying capacity of the jumper contacts on the side for each circuit. The load capacities for all potential jumpers are given in the following sections.



The arrangement of the potential jumpers can be found in Section "Electrical Potential and Data Routing" on page 1-27.
For voltage connection, please refer to the notes given in the module-specific data sheets.

1.10.1 Supply of the Ethernet Bus Coupler

The supply voltage U_{BK} and the segment voltage U_S **must** be connected to the Ethernet bus coupler. From the supply voltage U_{BK} , the voltages for the logic circuit U_L (7.5 V) and the supply of the modules for analog signals U_{ANA} (24 V) are internally generated. The segment voltage is used to supply the sensors and actuators.

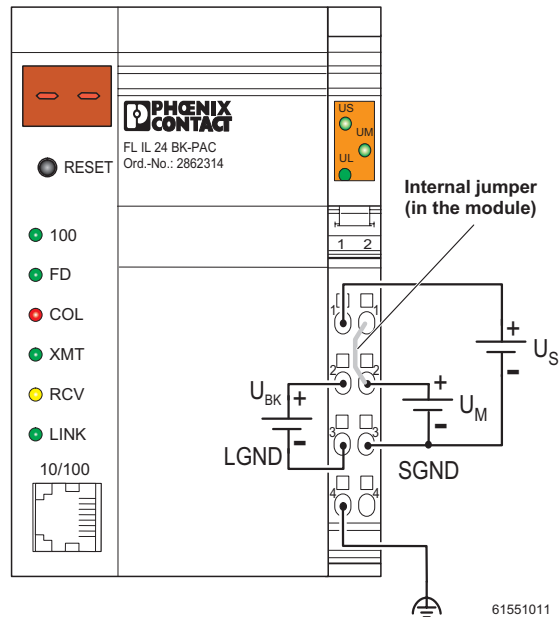


Figure 1-16 Typical connection of the supply voltage

1.10.2 Logic Circuit U_L

The logic circuit with communications power U_L starts at the bus coupler, is led through all modules of a station and cannot be supplied via another supply terminal.

Function

The logic circuit provides the communications power for all modules in the station.

Voltage

The voltage in this circuit is 7.5 V DC.

Provision of U_L

The communications power U_L is generated from the supply voltage U_{BK} of the bus coupler.
The communications power is not electrically isolated from the 24 V input voltage for the bus coupler.

Current carrying capacity

The maximum current carrying capacity of U_L is 2 A.

1.10.3 Analog Circuit U_{ANA}

The analog circuit with the supply for the analog modules (also referred to as analog voltage) U_{ANA} is supplied at the bus coupler and is led through all the modules in an Inline station. Power cannot be supplied by the supply terminals. U_{ANA} is not electrically isolated from U_{BK} .

Function

The module I/O devices for analog signals are supplied from the analog circuit.

Voltage

The voltage in this circuit is 24 V.

Provision of U_{ANA}

The analog voltage U_{ANA} is generated from the main voltage U_{BK} of the bus coupler.

Current carrying capacity

The maximum current carrying capacity of U_{ANA} is 0.5 A.

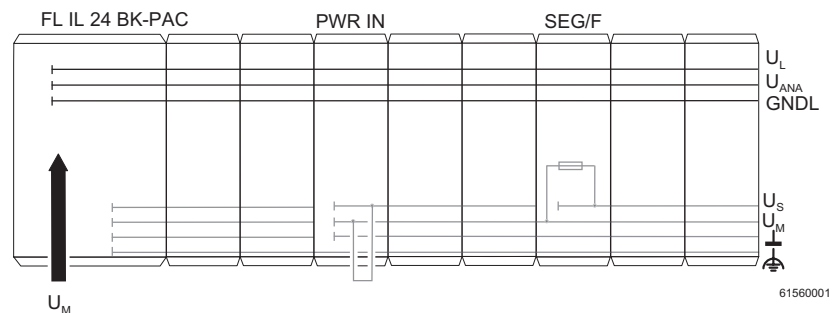


Figure 1-17 Logic and analog circuit

FL IL 24 BK-PAC	Ethernet bus coupler
PWR IN	Power terminal
SEG/F	Segment terminal with fuse as an example of a segment terminal

1.10.4 Main Circuit U_M

The main circuit with the main voltage U_M starts at the bus coupler or a power terminal and is led through all subsequent modules until it reaches the next power terminal. A new circuit that is electrically isolated from the previous one begins at the next power terminal.

Several power terminals can be used within one station.

Function

Several independent segments can be created within the main circuit. The main circuit provides the main voltage for these segments. For example, a separate supply for the actuators can be provided in this way.

Voltage



The maximum voltage in this circuit is 24 V DC. U_M can only be a maximum of 250 V AC when using special PWR-IN modules.

Current carrying capacity

The maximum current carrying capacity is 8 A (total current with the segment circuit). If the limit value of the common GND potential jumper for U_M and U_S is reached (total current of U_S and U_M), a new power terminal must be used.

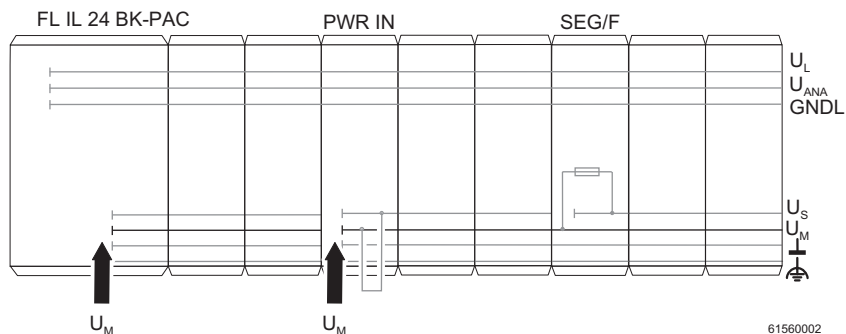


Figure 1-18 Main circuit

FL IL 24 BK-PAC	Ethernet bus coupler
PWR IN	Power terminal
SEG/F	Segment terminal with fuse as an example of a segment terminal

Provision of U_M

In the simplest case, the main voltage U_M can be supplied at the bus coupler. In this case it is 24 V DC.

The main voltage U_M can also be supplied via a power terminal. A power terminal **must** be used if:

- 1 Different voltage areas (e.g., 120 V AC) are to be created.
- 2 Electrical isolation is to be created.
- 3 The maximum current carrying capacity of a potential jumper (U_M , U_S or GND, total current of U_S and U_M) is reached.

1.10.5 Segment Circuit

The segment circuit or auxiliary circuit with the segment voltage U_S starts at the Ethernet bus coupler or a supply terminal (power terminal or segment terminal) and is led through all subsequent modules until it reaches the next supply terminal.

Function

You can use several segment terminals within a main circuit, and in this way segment the main circuit. It has the same reference ground as the main circuit. This means that circuits with different fuses can be created within the station without external cross wiring.

Voltage

The voltage in this circuit should not exceed 24 V DC.

Current carrying capacity

The current carrying capacity is 8 A, maximum (total current with the main circuit). If the limit value of the common potential jumper for U_M and/or U_S is reached (total current of U_S and U_M), a new power terminal must be used.

Generation of U_S

There are various ways of providing the segment voltage U_S :

- 1 You can supply the segment voltage at the Ethernet/Inline bus coupler or a power terminal.
- 2 You can tap the segment voltage from the main voltage at the Ethernet/Inline bus coupler or a power terminal using a jumper or a switch.
- 3 You can use a segment terminal with a fuse. Within this terminal the segment voltage is automatically tapped from the main power.
- 4 You can use a segment terminal without a fuse and tap the segment voltage from the main voltage using a jumper or a switch.



With 120 V/230 V AC voltage levels, segments cannot be created. In this case, only the main circuit is used.

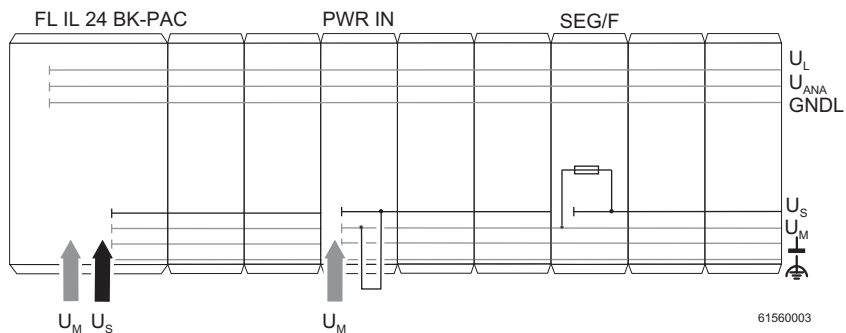


Figure 1-19 Segment circuit

FL IL 24 BK(-PAC)	Ethernet/Inline bus coupler
PWR IN	Power terminal
SEG/F	Segment terminal with fuse as an example of a segment terminal

1.11 Voltage Concept

The Ethernet bus coupler and the Inline local bus system have a defined voltage and grounding concept.

This avoids an undesirable effect on I/O devices in the logic area, suppresses undesirable compensating currents, and increases noise immunity.

Electrical isolation: Ethernet

The Ethernet interface is electrically isolated from the bus coupler logic. The Ethernet cable shield is directly connected to functional earth ground. The device has two functional earth ground springs, which have contact to the DIN rail when they are snapped on. The springs are used to discharge interference, rather than serve as protective earth ground. To ensure effective interference discharge, even for dirty DIN rails, functional earth ground is also led to terminals 1.4 and 2.4. Always ground either terminal 1.4 or 2.4 (see Figure 1-32 on page 1-54). This also grounds the Inline station of the bus coupler sufficiently up to the first power terminal.

A 120 V AC or 230 V AC power terminal interrupts the FE potential jumper. Therefore a 24 V DC power terminal, which is located directly behind such an area, must also be grounded using the FE terminal point.

To avoid the flow of compensating currents, connect a suitably sized equipotential bonding cable parallel to the Ethernet cable.

No electrical isolation of the Inline communications power

The bus coupler does not have electrical isolation for the Inline module communications power. U_{BK} (24 V), U_L (7.5 V), and U_{ANA} (24 V) are not electrically isolated.

Isolated supply for logic and I/O devices

The logic and I/O devices can be supplied by separate power supply units. If you wish to use different potentials for the communications power (U_{BK}) and the segment/main voltage (U_S/U_M), do not connect the GND and GND_{UBK} grounds of the supply voltages.

Option 1

The Fieldbus coupler main voltage U_M and the I/O supply U_S are provided separately with the same ground potential from **two** voltage supplies:

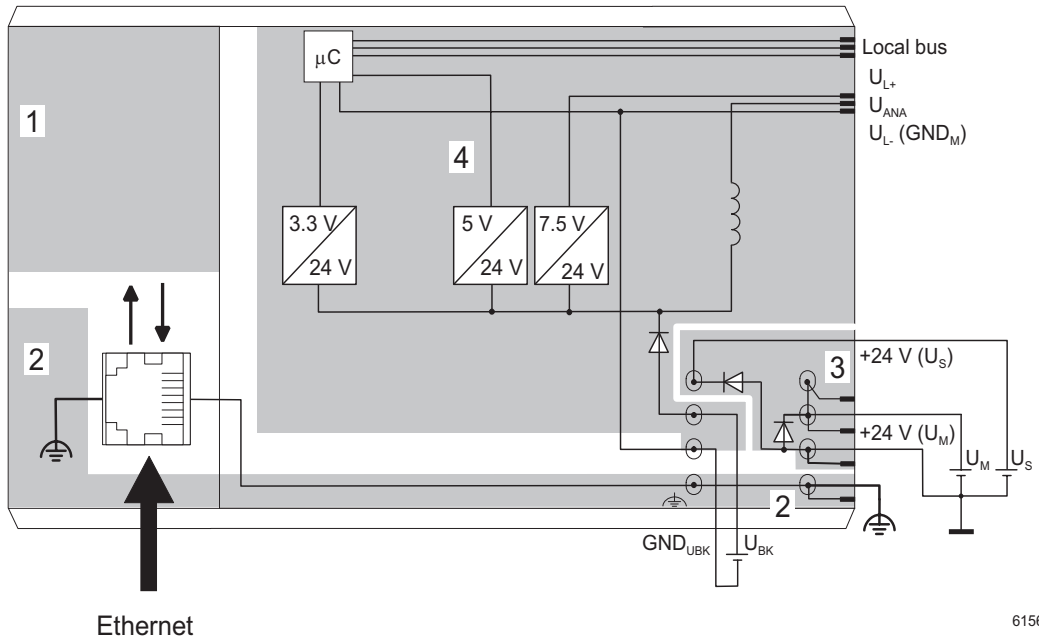


Figure 1-20 Potential areas in the bus coupler (two voltage supplies)

Potential areas:

- 1 Ethernet interface area
- 2 Functional earth ground (PE) and (shield) Ethernet interface area
- 3 Main voltage U_M and I/O voltage U_S area
- 4 Inline communications power

Option 2

Common supply of voltages U_{BK} , U_M , and U_S from **one** voltage supply:

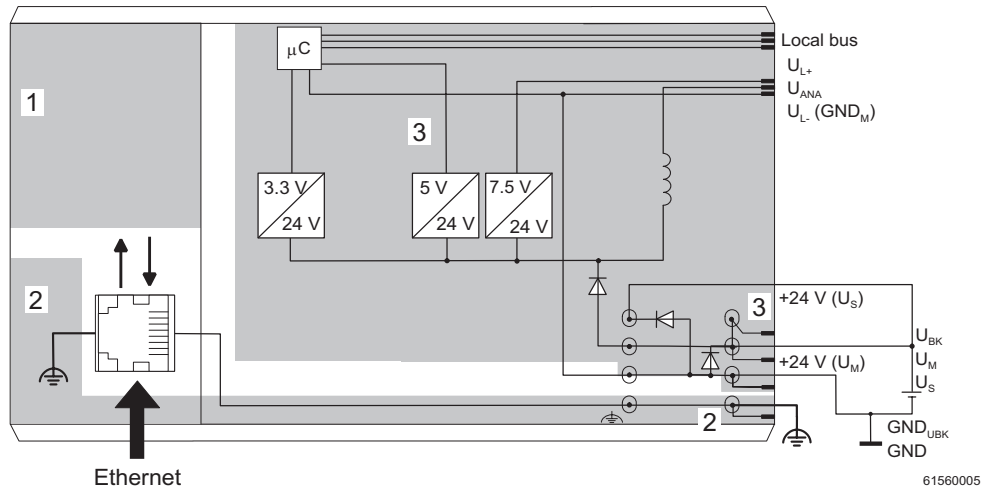


Figure 1-21 Bus coupler potentials (one voltage supply)

Potential areas:

- 1 Ethernet interface area
- 2 Functional earth ground / (shield) Ethernet interface area, bus coupler
- 3 Main voltage U_M and I/O voltage U_S area



The connector on the right can only be used when all the voltages supplied to the bus coupler have the same reference potential. Simply insert the external jumper to correctly connect all the supply points (see "Typical connection of the supply voltage" on page 1-30).

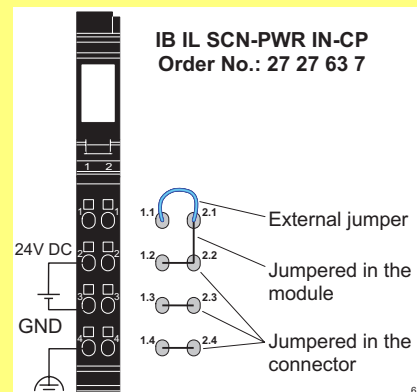
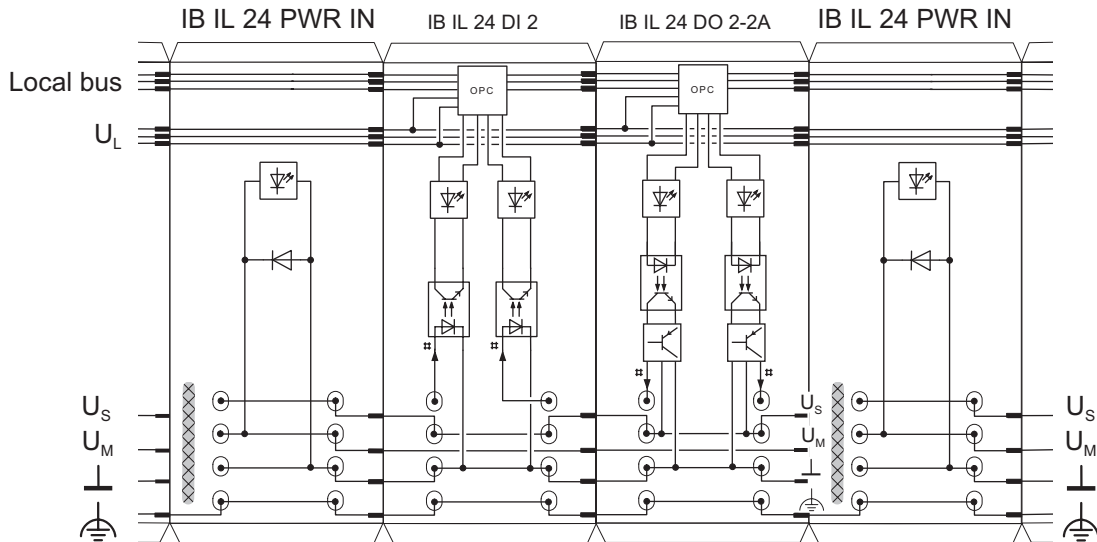


Figure 1-22 Power connector for supply from a single power supply unit

Potentials: Digital module

The isolation of the I/O circuit of a digital module to the communications power is only ensured if U_{BK} and U_M/U_S are provided from separate voltage supplies.

An example of this principle is shown in Figure 1-23 on a section of an Inline station.



61560013

Figure 1-23 Example: Interruption/creation of the potential jumpers using the power terminal

The areas hatched in the figure **XXXXXX** show the points at which the potential jumpers are interrupted.

Potentials: Analog module

The I/O circuit (measurement amplifier) of an analog module receives floating power from the 24 V supply voltage U_{ANA} . The power supply unit with electrical isolation is a component of an analog module. The voltage U_{ANA} is looped through in each module and, in this way, is also available to the next module.

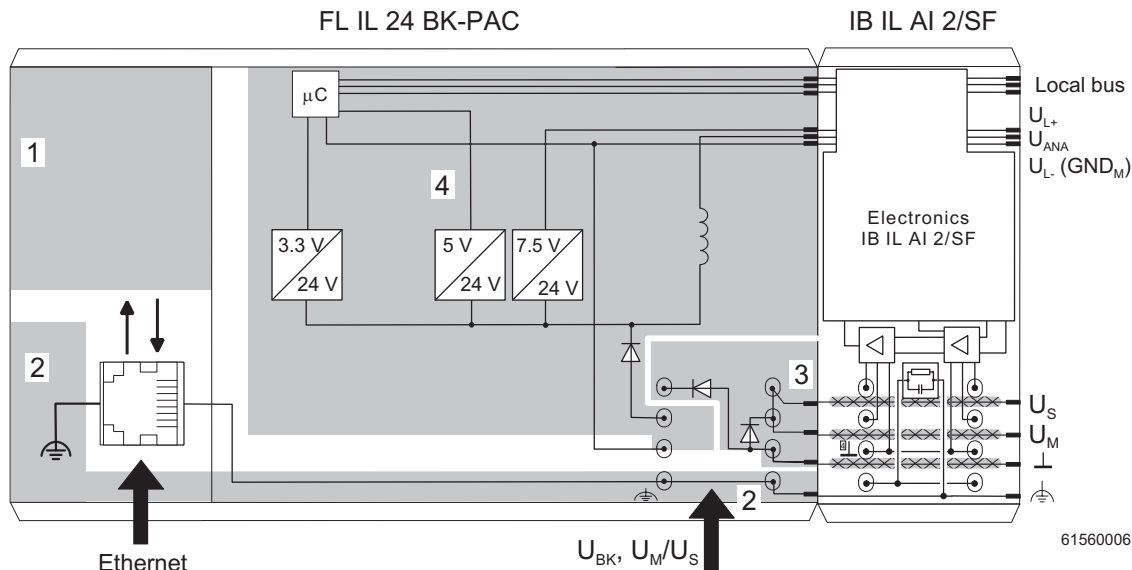


Figure 1-24 Electrical isolation between Ethernet bus coupler and analog module

The potential jumpers hatched **XXXXXX** in the figure are not used in the analog module. This means that the 24 V supply of the bus coupler (U_{BK}) or the power terminal is always electrically isolated from the I/O circuit (measurement amplifier) of the analog module. The I/O circuit of the analog module is supplied from the analog circuit U_{ANA} .

**I/O supply electrically
isolated from one
another**

Several electrically isolated segment or main circuits can be created by using power terminals. A power terminal interrupts the U_S/U_M and GND potential jumpers and has terminal points for another power supply unit. In this way, the I/O circuits of the Inline modules are electrically isolated from one another before and after the power terminal.

During this process the 24 V power supply units on the low voltage side must not be connected to one another.

One method of electrical isolation using a power terminal is illustrated in Figure 1-25. If a number of grounds are connected, e.g., to functional earth ground, electrical isolation is lost.

Because U_S and U_M can be supplied separately, it is possible to create separate segment circuits using a segment terminal. Using a switch, it is possible, for example, to create a switched segment circuit (see Figure 1-25 on page 1-41). U_S and U_M can be protected separately, yet still have a common ground potential. Please observe the maximum total current of 8 A.

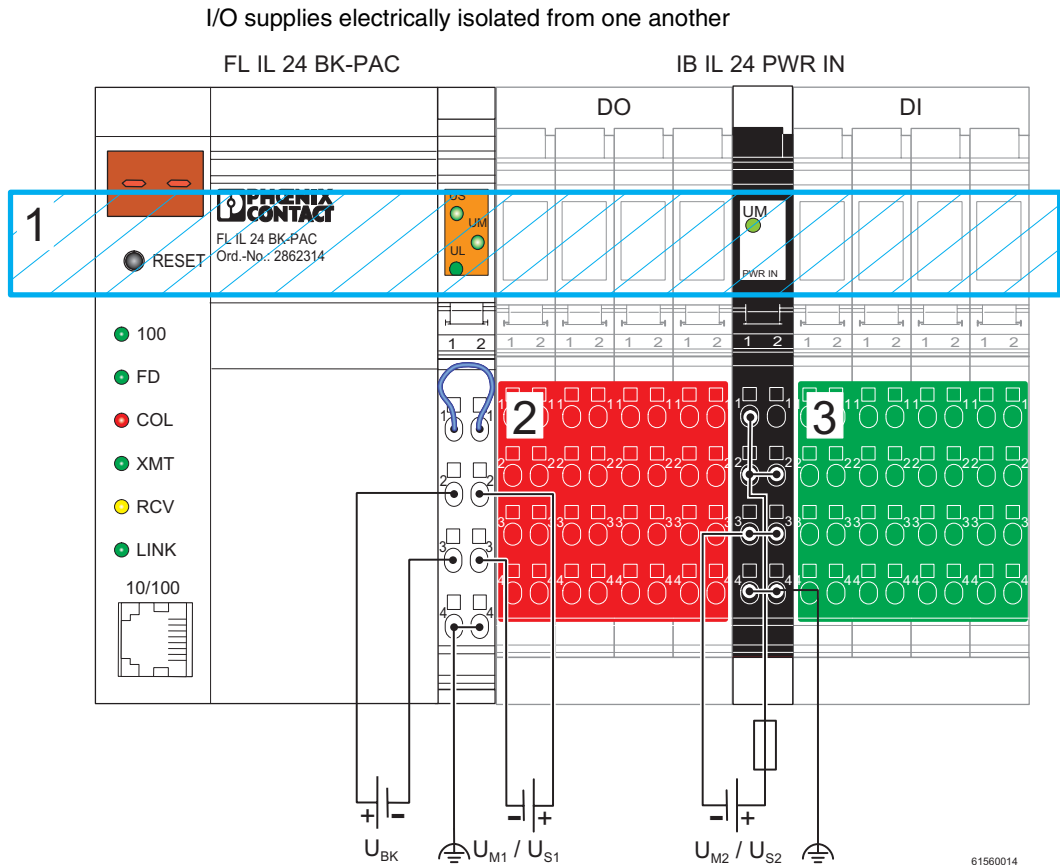


Figure 1-25 Structure of I/O supplies that are electrically isolated from one another

Potentials within the station:

- 1 Bus logic of the station
- 2 I/O (outputs)
- 3 I/O (inputs)

1.12 Diagnostic and status indicators

All modules are provided with LED diagnostic and status indicators for local error diagnostics.

Diagnostics

The diagnostic indicators (red/green) indicate the type and location of the error. The module is functioning correctly if all of the green LEDs are on.

Once an error has been removed, the indicators immediately display the current status.

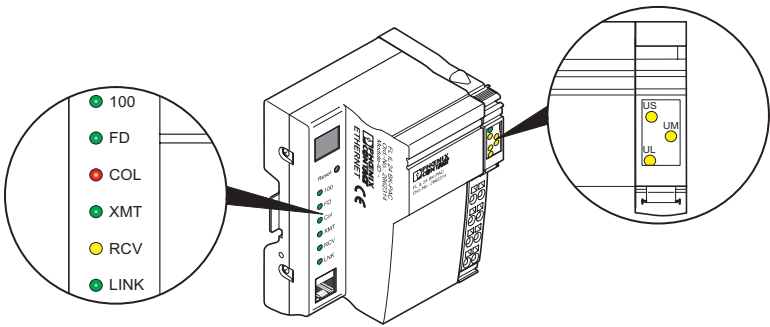
Status

The status indicators (yellow) display the status of the relevant inputs/outputs or the connected device.



For information about the diagnostic and status indicators on each module, please refer to the module-specific data sheet.

1.12.1 LEDs on the Ethernet Bus Coupler



61560015

Figure 1-26 LEDs on the Ethernet bus coupler

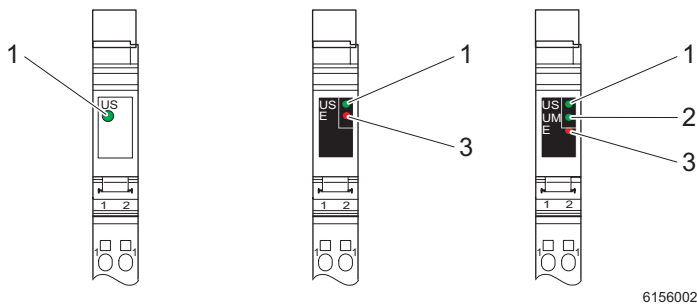
Diagnostics

The following states can be read on the bus coupler:

Table 1-14 Diagnostic LEDs of the bus coupler

Des.	Color	Status	Meaning
Module Electronics			
UL	Green	ON	24 V supply, 7 V communications power/interface supply present
		OFF	24 V supply, 7 V communications power/interface supply not present
UM	Green	ON	24 V main circuit supply present
		OFF	24 V main circuit supply not present
US	Green	ON	24 V segment supply present
		OFF	24 V segment supply not present
Ethernet Port			
100	Green	ON	Operation at 100 Mbps
		OFF	Operation at 10 Mbps (if LNK LED active)
FD	Green	ON	Data transmission in full duplex mode
		OFF	Data transmission in half duplex mode (if LNK LED active)
COL	Red	ON	Collision of data telegrams
		OFF	Transmission of telegrams without a collision (if LNK LED active)
XMT	Green	ON	Data telegrams are being sent
		OFF	Data telegrams are not being sent
RCV	Yellow	ON	Data telegrams are being received
		OFF	Data telegrams are not being received
LNK	Green	ON	Physical network connection ready to operate
		OFF	Physical network connection interrupted or not present

1.12.2 Indicators on the Supply Terminal



61560022

Figure 1-27 Possible indicators on supply terminals (segment terminal with and without fuse and power terminal)

Diagnostics

The following states can be read from the supply terminals:

Table 1-15 Diagnostic LED on the power terminal

LED	Color	State	Description of the LED States
UM (2)	Green	ON	24 V main circuit supply present
		OFF	Main circuit supply not present

Table 1-16 Diagnostic LED on the segment terminal

LED	Color	State	Description of the LED States
US (1)	Green	ON	24 V segment circuit supply present
		OFF	Segment circuit supply not present

Table 1-17 Additional LED on supply terminals with fuse

LED	Color	State	Description of the LED States
E (3)	Red	ON	Fuse not present or blown
		OFF	Fuse OK



On modules with fuses, the green LED indicates that the main or segment voltage is present **at the line side** of the fuse, meaning that if the green LED is on, there is voltage on the line side of the fuse. If the red LED is also on, the voltage is not present on the output side. Either no fuse is available or it is faulty.

1.12.3 Indicators on the Input/Output Modules



Figure 1-28 I/O module indicators

Diagnostics

The following states can be read from the I/O modules:

Table 1-18 Diagnostic LED of the I/O modules

LED	Color	State	Description of the LED States
D (1)	Green	ON	Local bus active
		Flashing:	
		0.5 Hz (slow)	Communications power present, local bus not active
		2 Hz (medium)	Communications power present, I/O error
		4 Hz (fast)	Communications power present, module in front of the flashing module has failed or the module itself is faulty; Modules following the flashing module are not part of the configuration frame.
		OFF	Communications power not present, local bus not active

Status

The status of the input or output can be read on the relevant yellow LED:

Table 1-19 Status LEDs of the I/O terminals

LED	Color	State	Description of the LED States
1, 2, 3, 4 (2)	Yellow	ON	Relevant input/output set
		OFF	Relevant input/output not set

Assignment Between Status LED and Input/Output



For the assignment of a status LED and the corresponding input/output, please refer to the module-specific data sheet.

1.12.4 Indicators on Other Inline Modules



For diagnostic and status indicators on other Inline modules (e.g., special function modules or power modules), please refer to the module-specific data sheet.

1.13 Mounting/Removing Modules and Connecting Cables

1.13.1 Installation Instructions



To ensure installation is carried out correctly, please read the "Installation Instructions for the Electrical Engineer" supplied with the bus coupler.



Do not replace modules while the power is connected

Before removing or mounting a module, disconnect the power to the entire station. Make sure the entire station is reassembled before switching the power back on. Failure to observe this rule may damage the module.

1.13.2 Mounting and Removing Inline Modules

An Inline station can be set up by mounting the individual components side by side. No tools are required. Mounting side by side automatically creates voltage and bus signal connections (potential and data routing) between the individual station components.

The modules are mounted perpendicular to the DIN rail. This ensures that they can be easily mounted and removed even within limited space.

After a station has been set up, individual modules can be exchanged by pulling them out or plugging them in. Tools are not required.

DIN rail

All Inline modules are mounted on 35 mm (1.378 in.) standard DIN rails.

End clamp/CLIPFIX

Mount end clamps on both sides of the Inline station. The end clamps ensure that the Inline station is correctly assembled. End clamps fix the Inline station on both sides and keep it from moving side to side on the DIN rail. Phoenix Contact recommends using the CLIPFIX 35 (Order No. 30 22 21 8) or E/UK end clamps (Order No. 12 01 44 2).



To remove the bus coupler, the left end clamp must be removed first.

End plate

An Ethernet Inline station **must** be terminated with an end plate. It has no electrical function. It protects the station against ESD pulses and the user against dangerous contact voltage. The end plate is supplied with the bus coupler and must not be ordered separately.

1.13.3 Mounting

When mounting a module, proceed as follows (Figure 1-29):

- First snap on the electronics base, which is required for mounting the station, perpendicular to the DIN rail (detail A).



Ensure that **all** featherkeys and keyways of adjacent modules are interlocked (detail B).

The keyway/featherkey connection links adjacent modules and ensures safe potential routing.

- Next, attach the connectors to the corresponding base.

First, place the front connector shaft latching in the front snap-on mechanism (detail C).

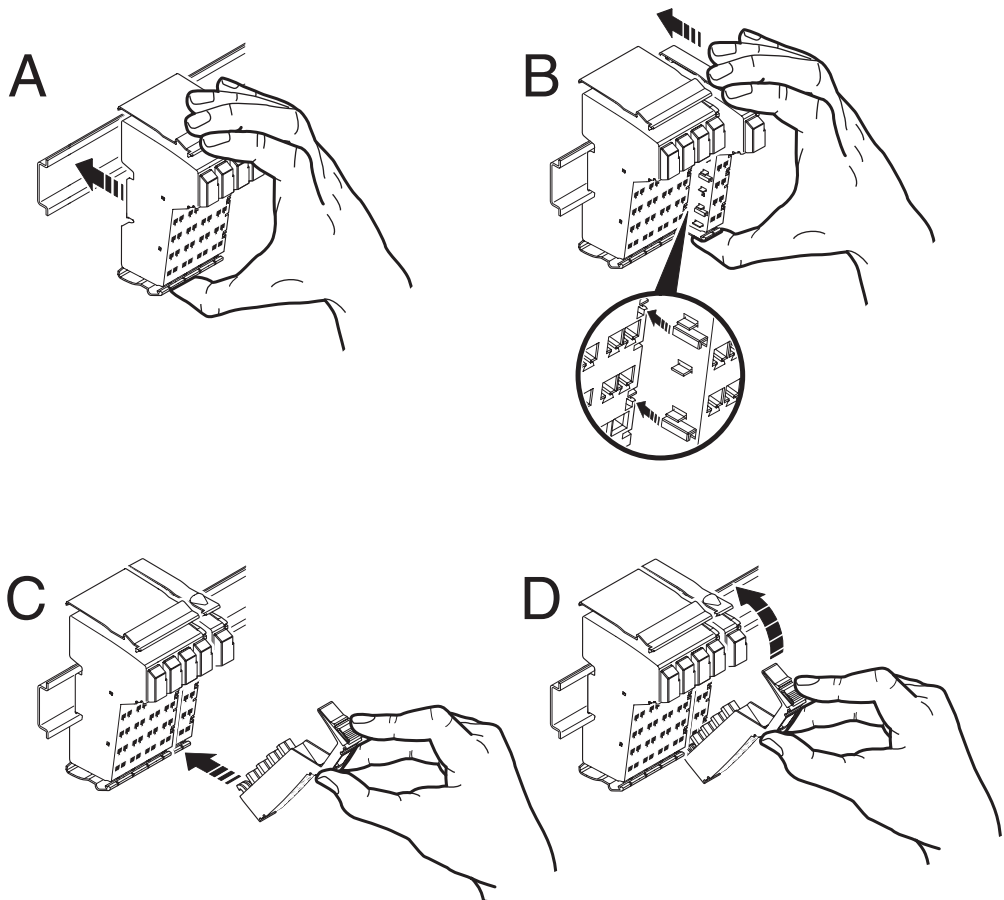
Then press the top of the connector towards the base until it snaps into the back snap-on mechanism (detail D).



The keyways of an electronics base do not continue when a connector has been installed on the base. When snapping on an electronics base, there must be no connector on the left-hand side of the base. If a connector is present, it will have to be removed.



Use end clamps to fix the Inline station to the DIN rail (see Ordering Data).



6138A015

Figure 1-29 Snapping on a module

1.13.4 Removing

When removing a module, proceed as follows (Figure 1-30):

- If there is a labeling field, remove it (A1 in detail A).



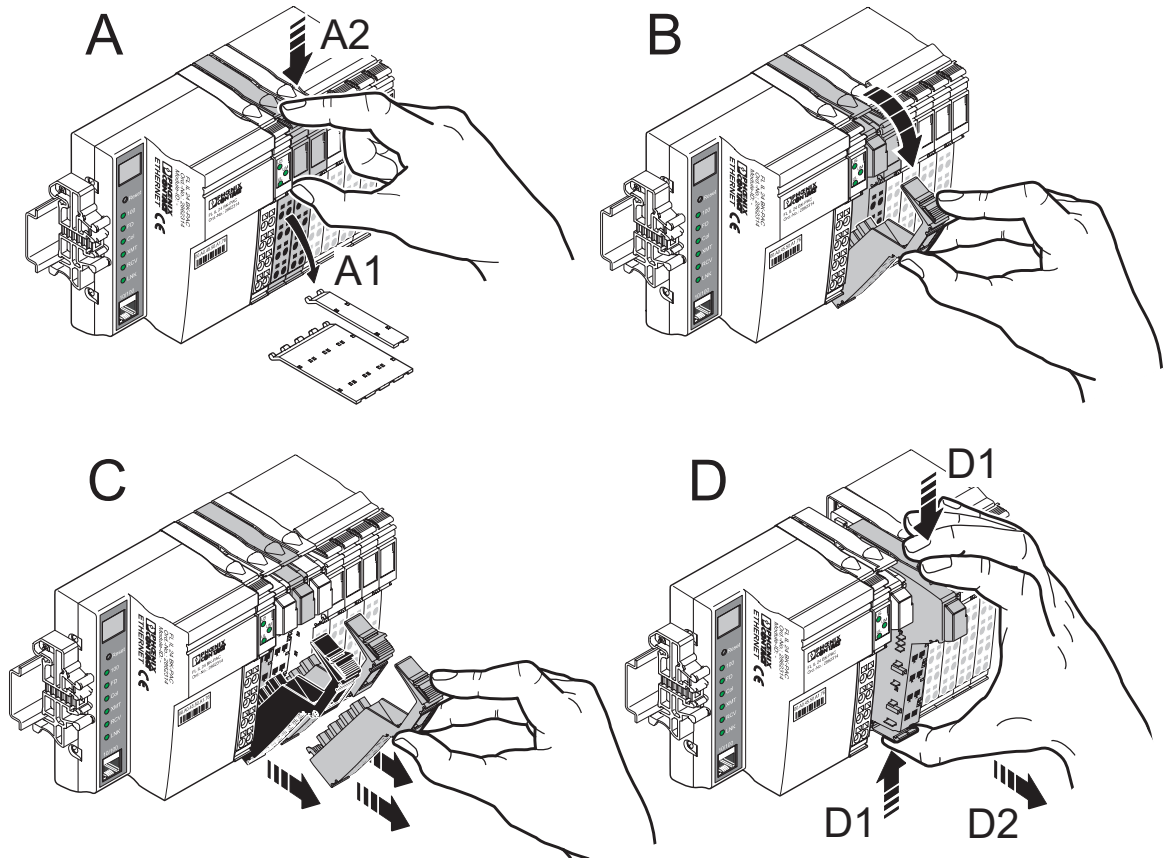
If a module has more than one connector, all of these must be removed. Below is a description of how to remove a 2-slot module.

Lift the connector of the module to be removed by pressing on the back connector shaft latching (A2 in detail A).

- Remove the connector (detail B).
- Remove the left-adjacent and right-adjacent connectors of the neighboring modules (detail C). This prevents the potential routing featherkeys and the keyway/featherkey connection from being damaged. You also have more space available for accessing the module.
- Press the release mechanism, (D1 in detail D) and remove the electronics base from the DIN rail by pulling the base straight back (D2 in detail D). If you have not removed the connector of the next module on the left, remove it now in order to protect the potential routing featherkeys and the keyway/featherkey connection.



To remove the bus coupler, the left end clamp must be removed first.



6156007

Figure 1-30 Removing a module

Replacing a module

If you want to replace a module within the Inline station, follow the removal procedure described above. Do not snap the connector of the module directly to the left back on yet. First, insert the base of the new module. Then reconnect all the connectors.



Use end clamps to fix the Inline station to the DIN rail (see Ordering Data).

1.13.5 Replacing a Fuse

The power and segment terminals are available with or without fuses.

For modules with fuses, the voltage presence and the fuse state are monitored and indicated by diagnostic indicators.

If a fuse is not present or defective, you must insert or replace it.

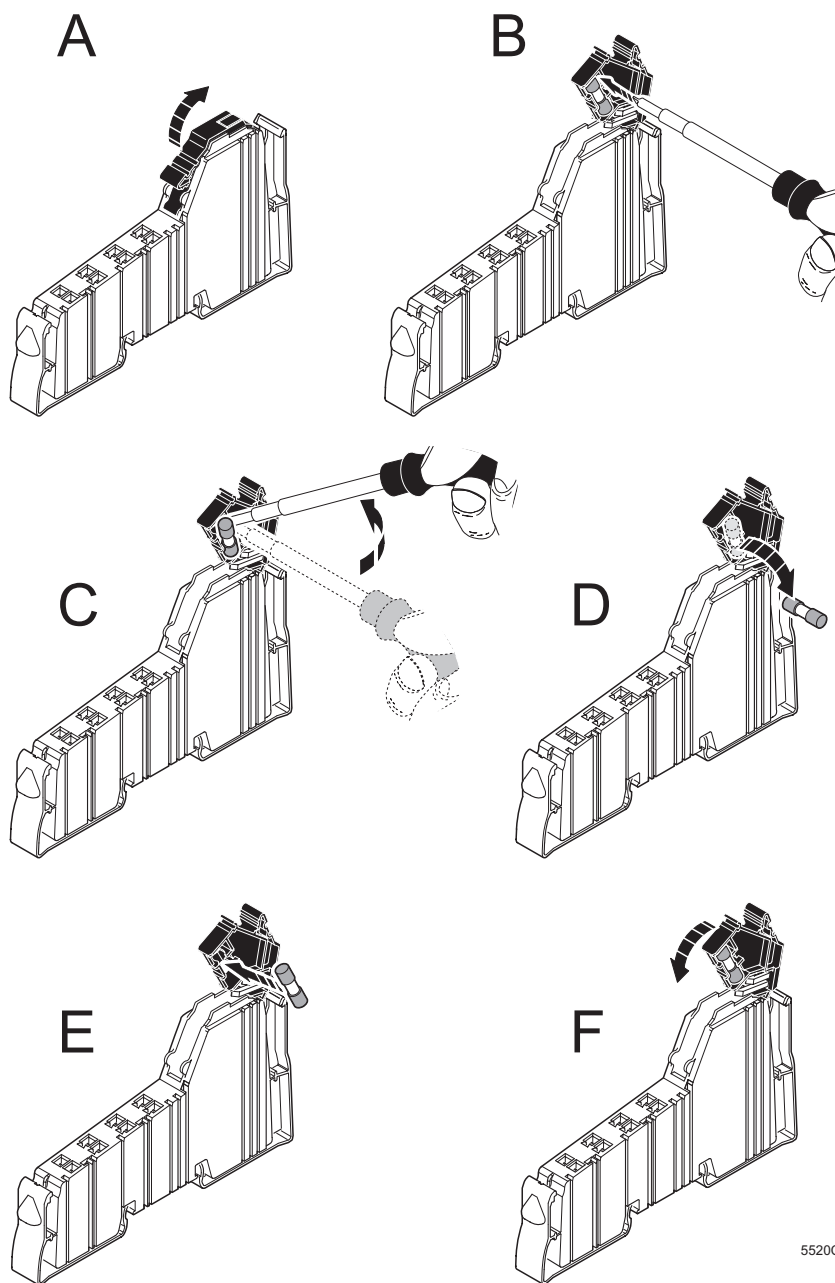


Observe the following when replacing a fuse in order to protect your health and the system.

1. Use the screwdriver carefully to avoid injury.
2. Lift the fuse out at the metal contact. Do not lift the fuse out at the glass part as you may break it.
3. Carefully lift the fuse out at one side and remove it by hand. Make sure the fuse does not fall into your system.

Follow these steps when replacing a fuse (see Figure 1-31):

- Lift the fuse lever (A).
- Insert the screwdriver behind a **metal contact** of the fuse (B).
- Carefully lift the metal contact of the fuse (C).
- Remove the fuse by hand (D).
- Insert a new fuse (E).
- Push the fuse lever down again until it clicks into place (F).



5520C011

Figure 1-31 Replacing a fuse

1.14 Grounding an Inline Station

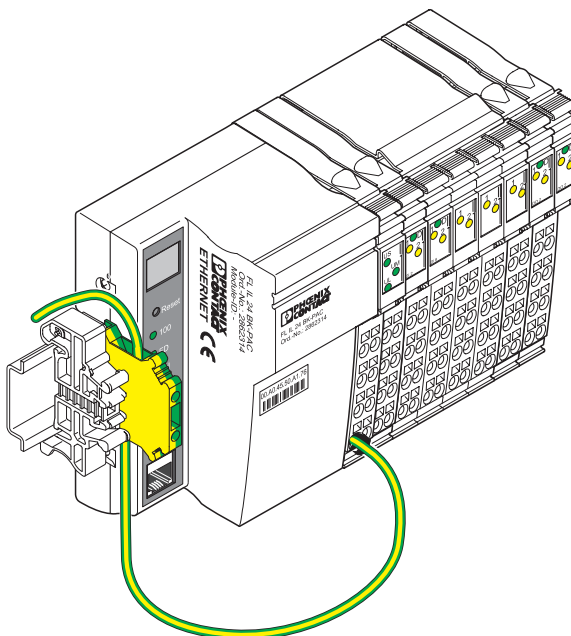
All devices in an Inline station must be grounded so that any possible interference is shielded and discharged to ground potential. A wire of at least 1.5 mm² (16 AWG) must be used for grounding.

Ethernet bus coupler and supply terminals

The bus coupler, power terminals, and segment terminals have FE springs (metal clips) on the bottom of the electronics base. These springs create an electric connection to the DIN rail. Use grounding terminal blocks to connect the DIN rail to protective earth ground. The modules are grounded when they are snapped onto the DIN rail.

Required additional grounding

In order to ensure reliable grounding even if the DIN rail is dirty or the metal clip has been damaged, Phoenix Contact specifies that the bus coupler must also be grounded via the FE terminal point (e.g., with the USLKG 5 universal ground terminal block, Order No. 04 41 50 4, see Figure 1-32).



61560008

Figure 1-32 Additional grounding of the FL IL 24 BK(-PAC)

FE potential jumper

The FE potential jumper (functional earth ground) runs from the bus coupler through the entire Inline station. Ground the DIN rail. FE is grounded when a module is snapped onto the DIN rail correctly. If supply terminals are part of the station, the FE potential jumper is also connected with the grounded DIN rail.



The function of FE is to discharge interference. It does not provide shock protection for people.

Low-level signal

The other Inline low-level signal modules are automatically grounded via the FE potential jumper when they are mounted adjacent to other modules.

Power level

The FE potential jumper is also connected to the power modules.

1.14.1 Shielding an Inline Station

Shielding is used to reduce the effects of interference on the system.

In the Inline station, the Ethernet cable and the module connecting cables for analog signals are shielded.



Observe the following when using shielded cables:

- Fasten the shielding so that as much of the braided shield as possible is held underneath the clamp of the shield connection.
- Make sure there is good contact between the connector and module.
- Do not damage or squeeze the wires. Do not strip off the wires too far.
- Make a clean wire connection.

1.14.2 Shielding Analog Sensors and Actuators



- For maximum noise immunity, always connect analog sensors and actuators using shielded, twisted-pair cables.
- Connect the shielding to the shield connector. The method for connecting the shielding is described in Section 1.15.2, "Connecting Shielded Cables Using the Shield Connector".
- Analog input and output modules require different shielding connections. The cable lengths must also be considered.

Table 1-20 Overview: shield connection of analog sensors/actuators

Module Type	Connection to the Module	Cable Length	Connection to the Sensor/ Actuator
Analog input module IB IL AI 2/SF	Within the module, ground is connected to FE via an RC element.	< 10 m (32.81 ft.)	–
		> 10 m (32.81 ft.)	Connect the sensor directly to PE
Analog output module IB IL AO ...	Via shield connection clamp directly to FE	< 10 m (32.81 ft.)	–
		> 10 m (32.81 ft.)	Isolate the actuator with an RC element and connect it to PE

1.14.2.1 Connecting an IB IL 24 AI 2/SF Analog Input Module

- Connect the shielding to the shield connector (see Section 1.15.2, "Connecting Shielded Cables Using the Shield Connector").
- When connecting the sensor shielding with FE potential, ensure a large surface connection.

Within the module, ground is connected to FE via an RC element.

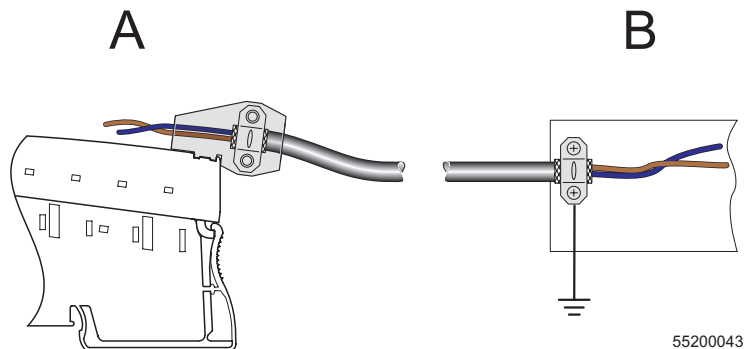


Figure 1-33 Connection of analog sensors, signal cables > 10 m (32.81 ft.)

- A Module side
B Sensor side



If you want to use both channels of the IB IL AI 2/SF module, there are different ways of connecting the shielding, depending on the cross-section.

- 1 Use a multi-wire cable for the connection of both sensors and connect the shielding to the shield connector as described above.
- 2 Use a thin cable for the connection of each sensor and connect the shielding of both cables together to the shield connector.
- 3 Use the standard connector (IB IL SCN-8; without shield connector). Twist the braided shield of each cable and place it on one of the terminal points to be used for FE connection.
You should only use this option if the cross-section is too large and the first two methods are not possible.

1.14.2.2 Connecting an Analog Output Module IB IL AO ...



- Connect the shielding via the shield connector (see Section 1.15.2, "Connecting Shielded Cables Using the Shield Connector").
- When connecting the shielding with the FE potential, ensure a large surface connection.



Danger of creating ground loops

- The shielding must only be directly connected to ground potential at one point.
- For **cable lengths exceeding 10 meters (32.81 ft.)** the actuator side should always be isolated by means of an RC element.
The capacitor C should typically have values of 1 nF to 15 nF. The resistor R should be at least 10 MΩ.

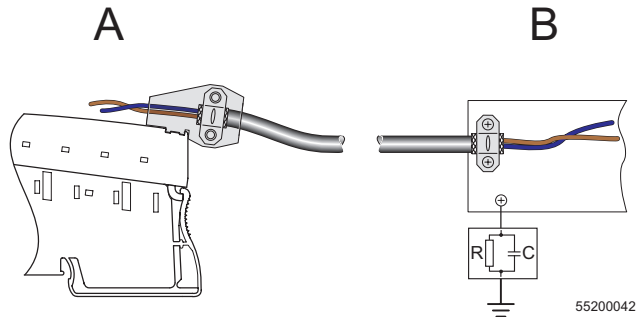


Figure 1-34 Connection of actuators, signal cables > 10 m (32.81 ft.)

- A Module side
B Actuator side

1.15 Connecting Cables

Both shielded and unshielded cables are used in a station.

The cables for the I/O devices and supply voltages are connected using the spring-cage connection method. This means that signals up to 250 V AC/DC and 5 A with a conductor cross section of 0.2 mm² through 1.5 mm² (AWG 25 - 16) can be connected.

The Ethernet cable is connected via an 8-pos. RJ45 connector.

1.15.1 Connecting Unshielded Cables

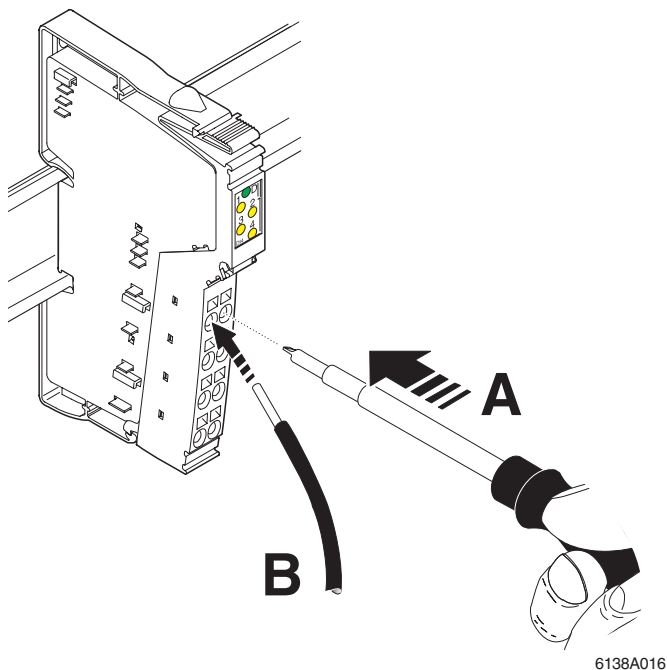


Figure 1-35 Connecting unshielded cables

Wire the connectors as required for your application.



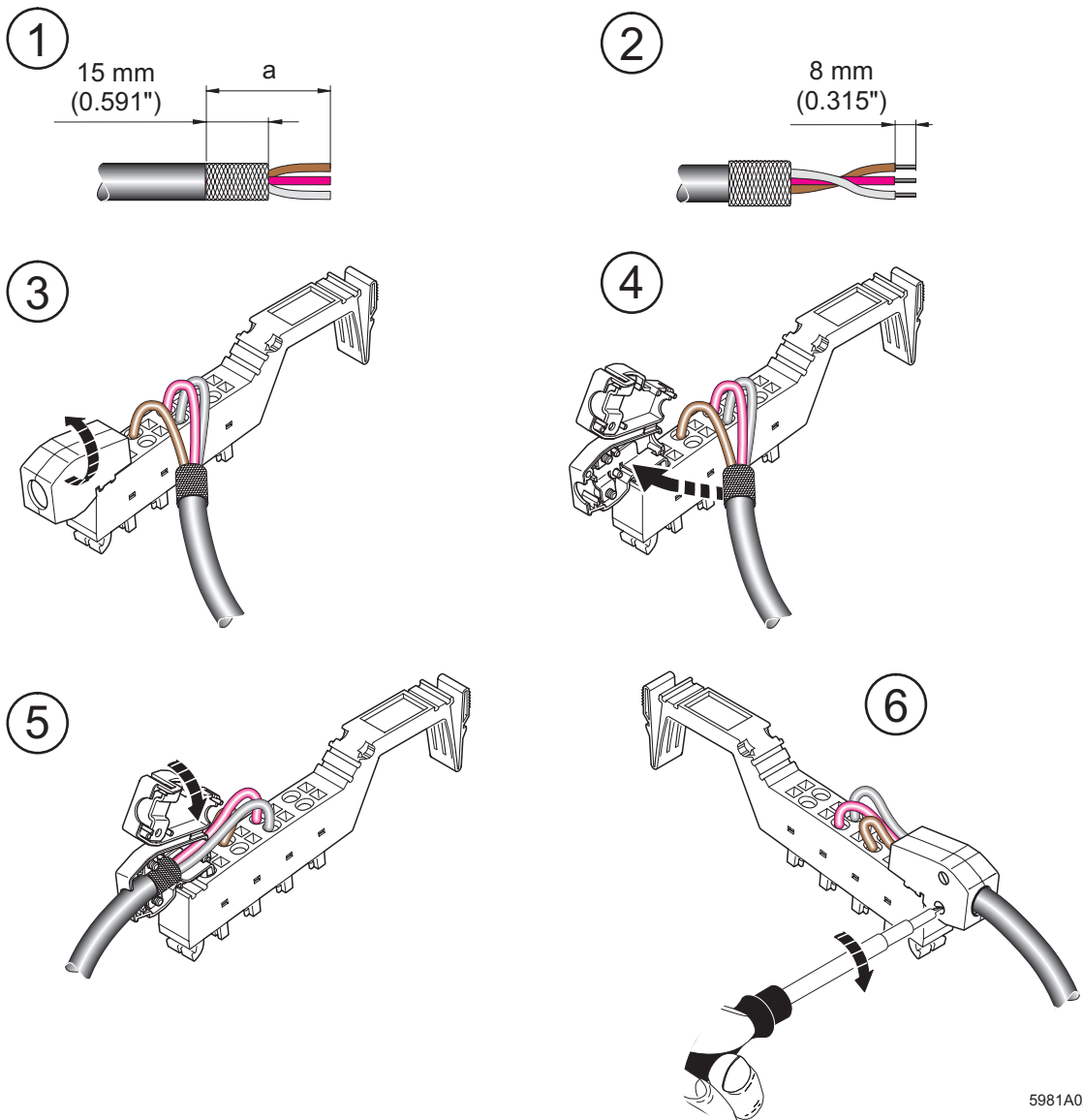
For connector assignment, please refer to the appropriate module-specific data sheet.



When wiring, proceed as follows:

- Strip 8 mm (0.315 in.) off the cable. Fieldbus coupler and Inline wiring is normally done without ferrules. However, it is possible to use ferrules. If using ferrules, make sure they are properly crimped.
- Push a screwdriver into the slot of the appropriate terminal point (Figure 1-35, detail 1), so that you can insert the wire into the spring opening. Phoenix Contact recommends using a SFZ 1 - 0,6 x 3,5 screwdriver (Order No. 12 04 51 7; see Phoenix Contact "CLIPLINE" catalog).
- Insert the wire (Figure 1-35, detail B). Remove the screwdriver from the opening. This clamps the wire.
- After installation, the wires and the terminal points should be labeled.

1.15.2 Connecting Shielded Cables Using the Shield Connector



5981A023

Figure 1-36 Connecting the shield to the shield connector

This section describes the connection of a shielded cable, using an "analog cable" as an example.

Connection should be carried out as follows:

Stripping cables

- Strip the outer cable sheath to the desired length (a). (1)
The desired length (a) depends on the connection position of the wires and whether there should be a large or a small space between the connection point and the shield connection.
- Shorten the braided shield to 15 mm (0.591 in.). (1)
- Fold the braided shield back over the outer sheath. (2)
- Remove the protective foil.
- Strip 8 mm (0.315 in.) off the wires. (2)



Inline wiring is normally done without ferrules. However, it is possible to use ferrules. If using ferrules, make sure they are properly crimped.

Wiring the connectors

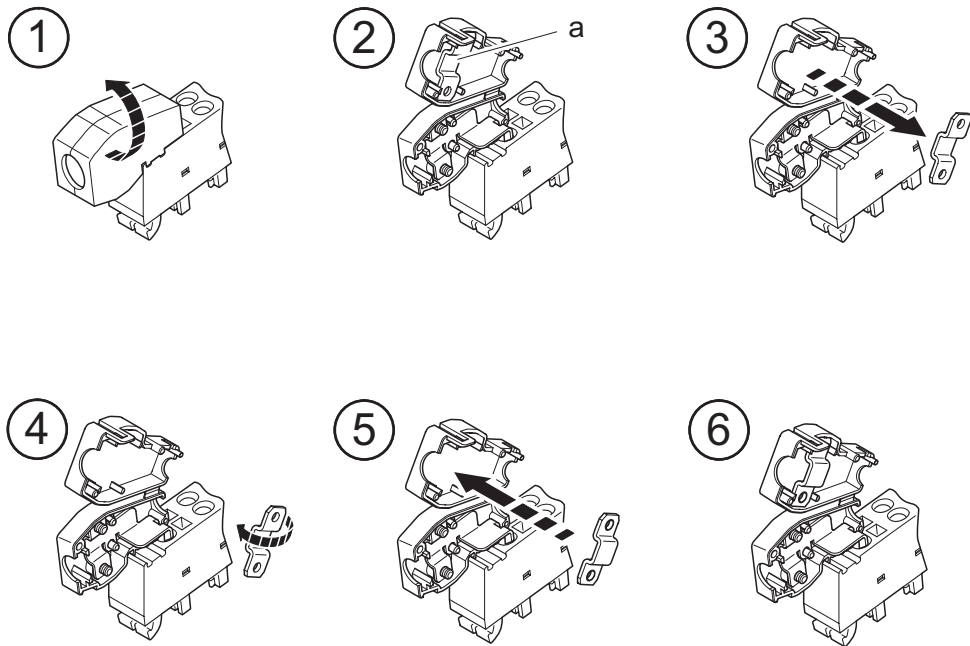
- Push a screwdriver into the slot of the appropriate terminal point (Figure 1-35 on page 1-59, detail 1), so that you can insert the wire into the spring opening. Phoenix Contact recommends using a SFZ 1 - 0,6 x 3,5 screwdriver (Order No. 12 04 51 7; see Phoenix Contact "CLIPLINE" catalog).
- Insert the wire (Figure 1-35 on page 1-59, detail 2). Remove the screwdriver from the opening. This clamps the wire.



For connector assignment, please refer to the appropriate module-specific data sheet.

Connecting the shield

- Open the shield connector. (3)
- Check the direction of the shield connection clamp in the shield connector (see Figure 1-37).
- Place the cable with the folded braided shield in the shield connector. (4)
- Close the shield connector. (5)
- Fasten the screws on the shield connector using a screwdriver. (6)



5520A068

Figure 1-37 Shield connection clamp alignment

Shield connection clamp

The shield connection clamp (a in Figure 1-37, detail 2) in the shield connector can be used in various ways depending on the cross section of the cable. For thicker cables, the dip in the clamp must be turned away from the cable (Figure 1-37, detail 2). For thinner cables, the dip in the clamp is turned towards the cable (Figure 1-37, detail 6).

If you need to change the direction of the shield connection clamp, proceed as shown in Figure 1-37:

- Open the shield connector housing (1).
- The shield connection is delivered with the clamp positioned for connecting thicker cables (2).
- Remove the clamp (3), turn it to suit the cross-section of the cable (4), then reinsert the clamp (5).
- Figure 6 shows the position of the clamp for a thin cable.

1.16 Connecting the Voltage Supply

To operate a station you must provide the supply voltage for the bus coupler, logic of the modules, and the sensors and actuators.

The voltage supplies are connected using unshielded cables (see Section 1.15.1).



For the connector assignment of the supply voltage connections, please refer to the module-specific data sheets for power and segment terminals.



Do not replace terminals while the power is connected

Before removing or mounting a module, disconnect the power to the entire station. Make sure the entire station is reassembled before switching the power back on.

1.16.1 Power Terminal Supply

Apart from supplying the I/O voltage at the Fieldbus coupler, it is also possible to provide the voltage using a power terminal.

U_M

24 V Main Circuit Supply

The main power is reintroduced at the power terminal.

U_S

24 V Segment Circuit Supply

The segment voltage can be supplied at the power terminal or generated from the main power. Install a jumper or create a segment circuit using a switch to tap the voltage U_S from the main circuit U_M .

Electrical isolation

You can create a new potential area through the power terminal.

Voltage areas

Power terminals can be used to create substations with different voltage areas. Depending on the power terminal, it is possible to work with 24 V DC, 120 V AC or 230 V AC.

**Use appropriate power terminals for different voltage areas**

To use different voltage areas within a station, a new power terminal must be used for each area.

**Dangerous voltage**

When the power terminal is removed, the metal contacts are freely accessible. With 120 V or 230 V power terminals, it should be assumed that dangerous voltage is present. You **must** disconnect power to the station **before removing** a terminal.

If these instructions are not followed, there is a danger of damage to health or even of a life-threatening injury.

1.16.2 Provision of the Segment Voltage Supply at Power Terminals

You cannot provide voltage at the segment terminal.

A segment terminal can be used to create a new partial circuit (segment circuit) within the main circuit. This segment circuit permits the separate supply of power outputs and digital sensors and actuators.

You can use a jumper to tap the segment voltage from the main circuit. If you use a switch, you can control the segment circuit externally.

You can create a protected segment circuit without additional wiring by means of a segment terminal with a fuse.

1.16.3 Demands on the Power Supply Units

**Use power supply units with safe isolation**

Only use power supply units that ensure safe isolation between the primary and secondary circuits according to EN 50178.



For additional voltage supply requirements, please refer to the data sheets for the segment and power terminals.

1.17 Connecting Sensors and Actuators

Sensors and actuators are connected using connectors. Each module-specific data sheet indicates the connector(s) to be used for that specific module.

Connect the unshielded cable as described in Section 1.15.1 on page 1-59 and the shielded cable as described in Section 1.15.2 on page 1-61.

1.17.1 Connection Methods for Sensors and Actuators

Most of the digital I/O modules in the Inline product range permit the connection of sensors and actuators in 2, 3 and 4-wire technology.

Because of the different types of connectors, a single connector can support the following connection methods:

- 2 sensors or actuators in 2, 3 or 4-wire technology
- 4 sensors or actuators in 2 or 3-wire technology
- 2 sensors or actuators in 2 or 3-wire technology with shield
(for analog sensors or actuators)



When connecting analog devices, please refer to the module-specific data sheets, as the connection method for analog devices differs from that for digital devices.

1.17.2 Connection Examples for Digital I/O Modules

Various connection options are described below using 24 V DC modules as an example. For the 120 V/230 V AC area, the data changes accordingly. A connection example is given in each module-specific data sheet.

Table 1-21 Overview of the connections used for digital input modules

Connection	Representation in the Figure	2-wire	3-wire	4-wire
Sensor signal IN	IN	X	X	X
Sensor supply U_S / U_M	U_S (+24 V)	X	X	X
Ground GND	GND ($\perp$)	–	X	X
Ground/FE shielding	FE ()	–	–	X

X Used
– Not used

Table 1-22 Overview of the connections used for digital output modules

Connection	Representation in the Figure	2-wire	3-wire	4-wire
Actuator signal OUT	OUT	X	X	X
Actuator supply U_S	U_S (+24 V)	–	–	X
Ground GND	GND ($\perp$)	X	X	X
Ground/FE shielding	FE ()	–	X	X

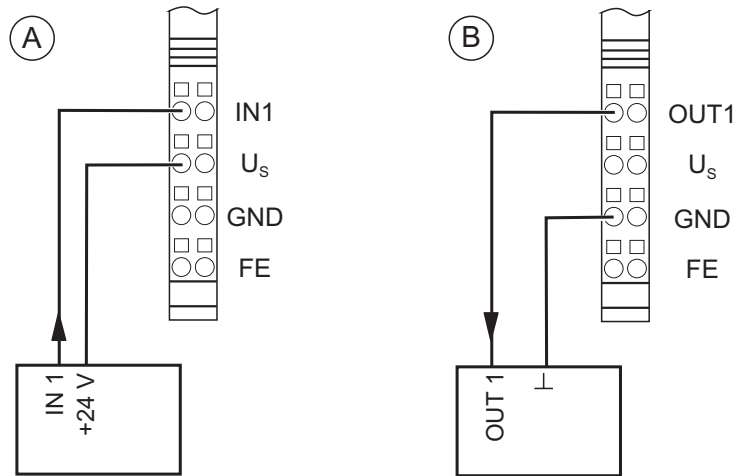
X Used
– Not used



In the following figures U_S designates the supply voltage. Depending on which potential jumper is accessed, the supply voltage is either the main voltage U_M or the segment voltage U_S .

Different Connection Methods for Sensors and Actuators

2-wire technology



55200027

Figure 1-38 2-wire termination for digital devices

Sensor

Figure 1-38, detail A, shows the connection of a 2-wire sensor. The sensor signal is carried to terminal point IN1. The sensor is supplied from the voltage U_s .

Actuator

Figure 1-38, detail B, shows the connection of an actuator. The actuator is supplied through output OUT1. The load is switched directly by the output.



The maximum current carrying capacity of the output must not be exceeded (see the module-specific data sheet).

3-wire technology

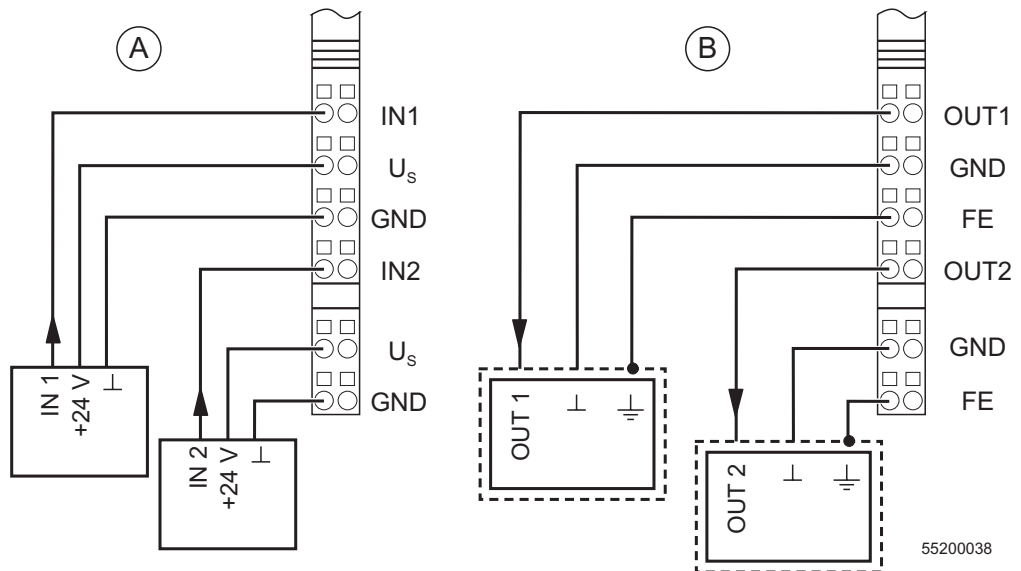


Figure 1-39 3-wire termination for digital devices

Sensor

Figure 1-39, detail A, shows the connection of a 3-wire sensor. The sensor signal is carried to terminal point IN1 (IN2). The sensor is supplied via terminal points U_s and GND.

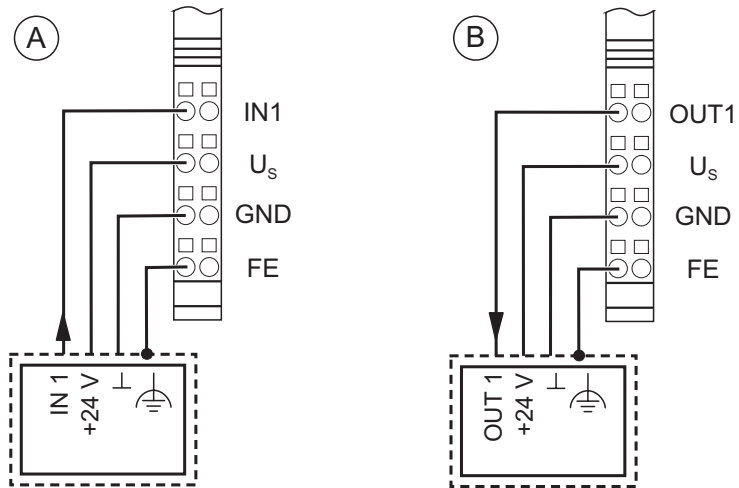
Actuator

Figure 1-39, detail B, shows the connection of a shielded actuator. The actuator is supplied through output OUT1 (OUT2). The load is switched directly by the output.



The maximum current carrying capacity of the output must not be exceeded (see the module-specific data sheet).

4-wire technology



55200037

Figure 1-40 4-wire termination for digital devices

Sensor

Figure 1-40, detail A, shows the connection of a shielded 4-wire sensor. The sensor signal is carried to terminal point IN1. The sensor is supplied via terminal points U_s and GND. The sensor is grounded via the FE terminal point.

Actuator

Figure 1-40, detail B, shows the connection of a shielded actuator. The provision of the supply voltage U_s means that even actuators that require a separate 24 V supply can be connected directly to the terminal.



The maximum current carrying capacity of the output must not be exceeded (see the module-specific data sheet).

This section provides information about

- startup
- assigning IP parameters
- the Management Information Base (MIB)

Startup/Operation	2-3
2.1 Default Upon Delivery/Default Settings	2-3
2.2 Firmware Start	2-3
2.3 Transmitting BootP Requests	2-4
2.4 Assigning an IP Address Using the Factory Manager	2-4
2.4.1 BootP	2-4
2.5 Manual Addition of Devices Using The Factory Manager	2-5
2.6 Selecting IP Addresses	2-5
2.6.1 Possible Address Combinations	2-7
2.6.2 Subnet Masks	2-8
2.6.3 Structure of the Subnet Mask	2-8
2.7 Web-Based Management	2-10
2.7.1 Calling Web-Based Management (WBM)	2-10
2.7.2 Structure of the Web Pages	2-11
2.7.3 Layout of the Web Pages	2-11
2.7.4 Password Protection	2-12
2.7.5 Process Data Access Via XML	2-12
2.8 Firmware Update	2-17
2.8.1 Firmware Update Using The Factory Manager	2-17
2.9 Firmware Update Using Web-Based Management (WBM) Without Factory Manager	2-19
2.9.1 Trap Generation	2-19
2.9.2 Representation of Traps in the Factory Manager	2-19
2.9.3 FL IL 24 BK(-PAC)Traps	2-20
2.9.4 Defining the Trap Manager	2-20
2.10 Factory Line I/O Configurator	2-21
2.10.1 Factory Line I/O Browser	2-21
2.10.2 OPC Configurator	2-22
2.11 Management Information Base - MIB	2-24
2.11.1 Standard MIBs:	2-24

2.12 Interface Group (1.3.6.1.2.1.2) 2-25

 2.12.1 Private MIBs 2-30

2.13 Meaning of the 7-Segment Display 2-44

2 Startup/Operation

2.1 Default Upon Delivery/Default Settings

By default upon delivery the following functions and properties are available:

- The password is "private".
- The bus coupler has no valid IP parameters:
 - IP address: 0.0.0.0
 - Subnet mask: 0.0.0.0
 - Gateway: 0.0.0.0
- Plug & play mode activated
- Expert mode inactive
- System description: Ethernet bus coupler
 - System contact: unknown
 - System name: FL IL 24 BK
 - System location: unknown
- HW watchdog activated (default parameter: 0x00000001).
- No INTERBUS configuration stored. All entries set to 0x0000.
- Fault response mode: 1
- Protocol switch: 0
- Watchdog timeout: 500 ms

2.2 Firmware Start

The firmware is started after the device has been connected to the power supply or the reset key has been pressed. The following sequence is displayed (see also "Startup Behavior of the Bus Coupler" on page 3-8):

Table 2-1 Sequence displayed after the device is switched on

Display	Meaning
01	Boot loader is started, BootP requests are transmitted
bo	Firmware is extracted
02	Firmware is started
PP --	Plug & play mode activated or operation

2.3 Transmitting BootP Requests

Initial Startup:

During initial startup, the device transmits BootP requests without interruption until it receives a valid IP address. The requests are transmitted at varying intervals (2 s, 4 s, 8 s, 2 s, 4 s, etc.) so that the network is not loaded unnecessarily. If valid IP parameters are received, they are saved as configuration data by the device.

Restart:

If the device already has valid configuration data, it only transmits three more BootP requests upon a restart. If it receives a BootP reply, the new parameters are saved. If the device does not receive a reply, it starts with the previous configuration.



If **only** the tftp parameters are modified (see "Firmware Update" on page 2-17) for the existing configuration and the IP parameters remain the same, e.g., using firmware with a new file name, the modifications to the configuration only take effect when the software update flag is enabled on the device web page or via SNMP.

2.4 Assigning an IP Address Using the Factory Manager



Alternatively, the IP address can be entered via any BootP server.

There are two options available when assigning the IP address: reading the MAC address via BootP or manually entering the MAC address in the Add New Ethernet Device dialog box in the Factory Manager.

2.4.1 BootP

- Ensure that the network scanner  and the BootP server  have been started.
- Connect the device to the network and the supply voltage.

- The BootP request for the new device triggered by the device restart/reset appears in the Factory Manager message window. Select the relevant message.
- Click with the right mouse button on the BootP message for the device or on  .
- Enter the relevant data in the Add New Ethernet Device dialog box (see 2.5).
- Save the configuration settings and restart the device (reset key or power up).



If the device is being started for the first time, it is then automatically booted with the specified configuration. If the device is not being started for the first time, save the configuration and restart the device (reset key or power up). The device now transmits another BootP request and receives the specified IP parameters from the BootP server/Factory Manager.

2.5 Manual Addition of Devices Using The Factory Manager

- Open the Add New Ethernet Device dialog box by clicking on  , by selecting "Add Device" from the Device View context menu or by using the Ctrl+A key combination.
- Enter the desired data under "Description" and "TCP/IP Address".
- Activate the "BootP Parameter" by selecting "Reply on BootP Requests".
- Enter the MAC address. It can be found on the sticker on the front of the housing.
- Save the configuration settings and restart the device (reset key or power up).

The device now transmits another BootP request and receives the specified IP parameters from the BootP server.

2.6 Selecting IP Addresses

The IP address is a 32-bit address, which consists of a network part and a user part. The network part consists of the network class and the network address. There are currently five defined network classes; classes A, B, and C are used in modern applications, while classes D and E are hardly ever used. It is therefore usually sufficient if a network device only "recognizes" classes A, B, and C.

With binary representation of the IP address the network class is represented by the first bits. The key factor is the number of "ones" before the first "zero". The assignment of classes is shown in the following table. The empty cells in the table are not relevant to the network class and are already used for the network address.

	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5
Class A	0				
Class B	1	0			
Class C	1	1	0		
Class D	1	1	1	0	
Class E	1	1	1	1	0

The bits for the network class are followed by those for the network address and user address. Depending on the network class, a different number of bits is available, both for the network address (network ID) and the user address (host ID).

	Network ID	Host ID
Class A	7 bits	24 bits
Class B	14 bits	16 bits
Class C	21 bits	8 bits
Class D	28-bit multicast identifier	
Class E	27 bits (reserved)	

IP addresses can be represented in decimal, octal or hexadecimal notation. In decimal notation, bytes are separated by dots (dotted decimal notation) to show the logical grouping of the individual bytes.



The decimal points do not divide the address into a network and user address. Only the value of the first bits (before the first "zero") specifies the network class and the number of remaining bits in the address.

2.6.1 Possible Address Combinations

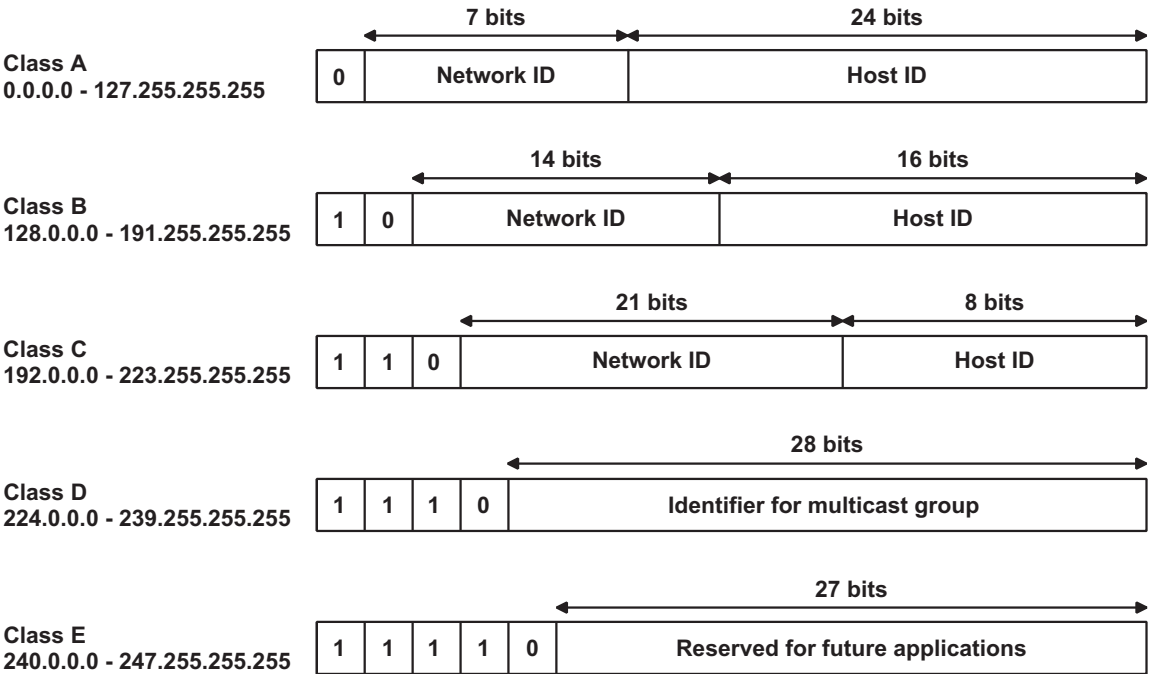


Figure 2-1 Structure of IP addresses

Special IP Addresses for Special Applications

Certain IP addresses are reserved for special functions. The following addresses should not be used as standard IP addresses.

127.x.x.x Addresses

The class A network address "127" is reserved for a loop-back function on all PCs, regardless of the network class. This loop-back function may only be used on networked PCs for internal test purposes.

If a telegram is addressed to a PC with the value 127 in the first byte, the receiver immediately sends the telegram back to the transmitter.

The correct installation and configuration of the TCP/IP software, for example, can be checked in this way.

As the first and second layers of the ISO/OSI reference model are not included in the test, complete testing should be carried out separately using the ping function.

Value 255 in the Byte

Value 255 is defined as a broadcast address. The telegram is sent to all the PCs that are in the same part of the network. Examples: 004.255.255.255, 198.2.7.255 or 255.255.255.255 (all the PCs in all the networks). If the network is divided into subnetworks, the subnet masks must be observed during calculation, otherwise some devices may be omitted.

0.x.x.x Addresses

Value 0 is the ID of the specific network. If the IP address starts with a zero, the receiver is in the same network. Example: 0.2.1.1 refers to device 2.1.1 in this network.

The zero previously signified the broadcast address. If older devices are used, unauthorized broadcast and complete overload of the entire network (broadcast storm) may occur when using the IP address 0.x.x.x.

2.6.2 Subnet Masks

Routers and gateways divide large networks into several subnetworks. The IP addresses for individual devices are assigned to specific subnetworks by the subnet mask. The **network part** of an IP address is **not** modified by the subnet mask. An extended IP address is generated from the user address and subnet mask. Because the masked subnetwork is only recognized by the local PCs, this extended IP address appears as a standard IP address to all the other devices.

2.6.3 Structure of the Subnet Mask

The subnet mask always contains the same number of bits as an IP address. The subnet mask has the same number of bits (in the same position) set to "one", which is reflected in the IP address for the network class.

Example: An IP address from class A contains a 1-byte network address and a 3-byte PC address. Therefore, the first byte of the subnet mask may only contain "ones".

The remaining bits (three bytes) then contain the address of the subnetwork and the PC. The extended IP address is created when the bits of the IP address and the bits of the subnet mask are ANDed. Because the subnetwork is only recognized by local devices, the corresponding IP address appears as a "normal" IP address to all the other devices.

Application

If ANDing of the address bits gives the local network address and the local subnetwork address, the device is located in the local network. If ANDing gives a different result, the data telegram is sent to the subnetwork router.

Example for a class B subnet mask:

Decimal notation: 255.255.192.0

Binary notation: 1111 1111.1111 1111.1100 0000.0000 0000



Using this subnet mask, the TCP/IP protocol software differentiates between the devices that are connected to the local subnetwork and the devices that are located in other subnetworks.

Example: Device 1 wants to establish a connection to device 2 using the above subnet mask. Device 2 has IP address 59.EA.55.32.

IP address representation for device 2:

Hexadecimal notation: 59.EA.55.32

Binary notation: 0101 1001.1110 1010.0101 0101.0011 0010

The individual subnet mask and the IP address for device 2 are then ANDed bit by bit by the software to determine whether device 2 is located in the local subnetwork.

ANDing the subnet mask and IP address for device 2:

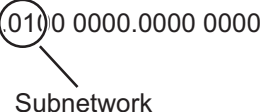
Subnet mask: 1111 1111.1111 1111.1100 0000.0000 0000

AND

IP address: 0101 1001.1110 1010.0101 0101.0011 0010

Result after ANDing: 0101 1001.1110 1010.0100 0000.0000 0000

Subnetwork



After ANDing, the software determines that the relevant subnetwork (01) does not correspond to the local subnetwork (11) and the data telegram is forwarded to a subnetwork router.

2.7 Web-Based Management

The FL IL 24 BK(-PAC) has a web server, which generates the required pages for web-based management and, depending on the requirements of the user, sends them to the "Factory Manager" or a standard web browser.

Web-based management can be used to access static information (e.g., technical data, MAC address) or dynamic information (e.g., IP address, status information) or to change the configuration (password-protected).

2.7.1 Calling Web-Based Management (WBM)

The FL IL 24 BK-PAC web server can be addressed using the IP address if configured correspondingly.

The bus coupler homepage is accessed by entering the URL "<http://ip-address>".

Example: <http://192.168.2.81>

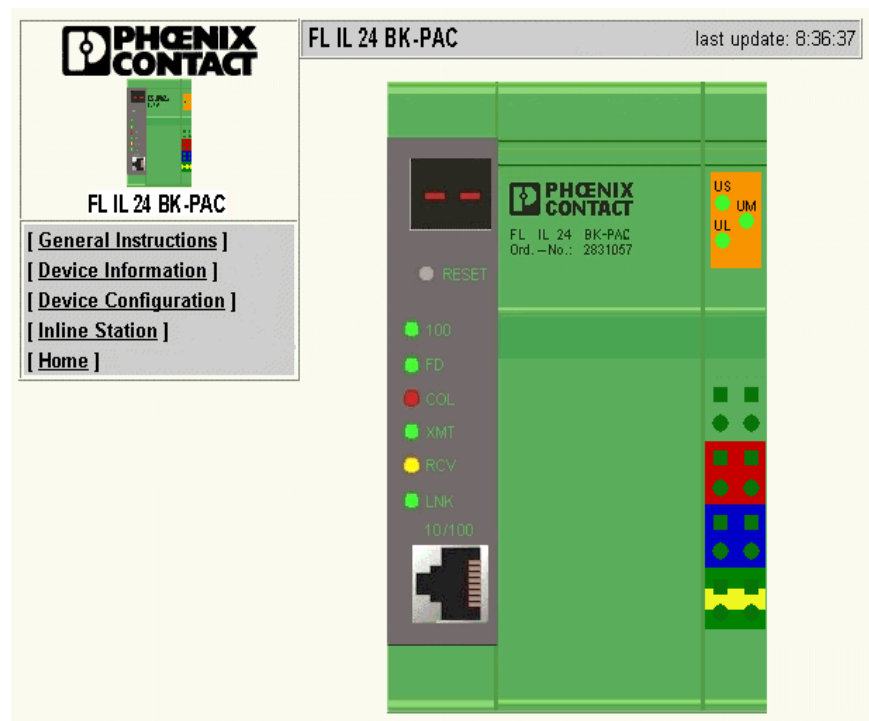


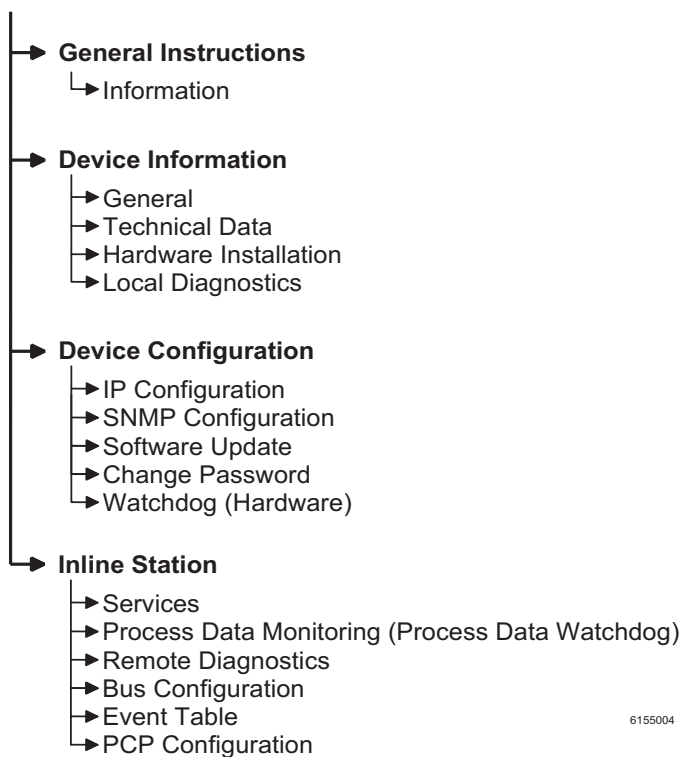
Figure 2-2 WBM homepage

2.7.2 Structure of the Web Pages

The Ethernet bus coupler pages are divided into two parts, with the selection menu and the relevant submenus on the left-hand side, and the corresponding information displayed on the right-hand side. Static and dynamic information about the bus coupler can be found in the following menus.

2.7.3 Layout of the Web Pages

FL IL 24 BK-PAC



6155004

2.7.4 Password Protection

The bus coupler is protected by two passwords (case-sensitive). The password for read access is "public", while the password for read and write access is "private". All status changes to the bus coupler are only possible after the password for read and write access has been entered. The password can be changed at any time. Your unique password must be between four and twelve characters long.



If you forget the password, the device can be re-enabled by Phoenix Contact. Ensure you have the exact device designation and serial number ready when you call the telephone number indicated on the last page.

2.7.5 Process Data Access Via XML

The integrated FL IL 24 BK-PAC web server can be used to access the process data of the connected Inline terminals using a web page in XML format.

Use a standard web browser to access the web pages. Enter the address in the following format in the address line of the browser to call the XML pages containing the process data: "http:// <IP-Adresse>/processdata.xml".

2.7.5.1 Structure of the XML Files

The XML file comprises different data areas:

IL_STATION	Frame for the entire XML file. The obligatory elements of this frame are IL_BUS_TERMINAL and IL_BUS.
IL_BUS_TERMINAL	This data area contains information on the entire Inline station (bus coupler and all connected terminals). This area includes: TERMINAL_TYPE, module NAME, IP_ADDRESS, the number of connected terminals MODULE_NUMBER, the INTERBUS diagnostic register DIAGNOSTIC_STATUS_REGISTER, and the INTERBUS status register DIAGNOSTIC_PARAMETER_REGISTER.
TERMINAL_TYPE	This area contains the module designation, i.e., always FL IL 24 BK(-PAC).
NAME	Contains user-specific station names. The station name can be modified via SNMP or WBM.
IP_ADDRESS	Contains the station IP address.
MODULE_NUMBER	Contains the number of connected Inline terminals. In the event of a bus error the number of the last known operable configuration is specified.

DIAGNOSTIC_STATUS_REGISTER	Contains the INTERBUS status, indicated by all the bits in the diagnostic status register. A detailed description can be found in the diagnostic parameter register. The diagnostic parameter register is always re-written if an error bit is set.
IL_BUS	Frame for the connected Inline terminals.
IL_MODULE	Frame for the data of individual Inline terminals. The terminals are numbered from one to 63, maximum.
MODULE_TYPE	Contains the module type, e.g., DI, DO, DIO, AI, AO, AIO, and PCP.
PD_CHANNELS	Number of process data channels of an Inline terminal. With digital terminals the number of channels is equal to the number of supported bits. With other modules the number of process data words is indicated. Example: An AO 2 has two process data channels and a DO 8 has eight bits and eight process data channels.
PD_WORDS	Number of process data words of an Inline terminal. Note that analog terminals always have the same number of output and input data words. An AO 2 also has two input channels and an AI 2 also has two output channels.
PD_IN	This area is used by all terminals that use input data. The number of process data words depends on the terminal type.

Example:

a) Inline terminal with two active inputs

```
<IL_MODULE number="1">
  <MODULE_TYPE>DI</MODULE_TYPE>
  <PD_CHANNELS>2</PD_CHANNELS>
  <PD_WORDS>1</PD_WORDS>
  <PD_IN word="1">3</PD_IN>
</IL_MODULE>
```

b) Inline terminal with two digital inputs; only the second input is active

```
<IL_MODULE number="3">
  <MODULE_TYPE>DI</MODULE_TYPE>
  <PD_CHANNELS>2</PD_CHANNELS>
  <PD_WORDS>1</PD_WORDS>
  <PD_IN word="1">2</PD_IN>
</IL_MODULE>
```

c) Inline terminal with 16 digital inputs; inputs 13 and 14 are active

```
<IL_MODULE number="7">
  <MODULE_TYPE>DI</MODULE_TYPE>
  <PD_CHANNELS>16</PD_CHANNELS>
  <PD_WORDS>1</PD_WORDS>
  <PD_IN word="1">12288</PD_IN>
</IL_MODULE>
```

The inputs data word returns the value 12288 ($2^{12} + 2^{13}$).

d) Inline terminal with two analog inputs; only the first channel (14970) is active

```
<IL_MODULE number="10">
  <MODULE_TYPE>AI</MODULE_TYPE>
  <PD_CHANNELS>2</PD_CHANNELS>
  <PD_WORDS>2</PD_WORDS>
  <PD_IN word="1">14970</PD_IN>
  <PD_IN word="2">8</PD_IN>
  <PD_OUT word="1">0</PD_OUT>
  <PD_OUT word="2">0</PD_OUT>
</IL_MODULE>
```

PD_OUT This area is used by all terminals with output data. The use of bits is identical to the use of "PD_IN".

2.7.5.2 Validity of Data

The validity of data is identical to the validity via DDI or OPC access.

2.7.5.3 Error in the Inline Station

If the FL IL 24 BK(-PAC) is not able to configure the connected Inline terminals correctly, error code "82" is displayed. The process data is then listed in the XML file as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE IL_STATION SYSTEM "processdata.dtd">
<IL_STATION>
  <IL_BUS_TERMINAL>
    <TERMINAL_TYPE>FL IL 24 BK-PAC</TERMINAL_TYPE>
    <NAME>FL IL 24 BK-PAC</NAME>
    <IP_ADDRESS>172.16.27.37</IP_ADDRESS>
    <MODULE_NUMBER>0</MODULE_NUMBER>
    <DIAGNOSTIC_STATUS_REGISTER>132
    </DIAGNOSTIC_STATUS_REGISTER>
    <DIAGNOSTIC_PARAMETER_REGISTER>65535</
    DIAGNOSTIC_PARAMETER_REGISTER>
  </IL_BUS_TERMINAL>
</IL_BUS>
</IL_BUS>
</IL_STATION>
```

The values of the diagnostic status register and the diagnostic parameter register indicate the error cause. The number of the connected terminals is "zero", which means that the "IL_BUS" area is empty.

In the event of a bus error "bF" is displayed. The process data is invalid because only internal values but no values on INTERBUS are indicated. The status is shown in the diagnostic register.

To ensure that valid data is shown, the diagnostic register must also always be scanned. The same behavior will occur if a wrong configuration is connected. In this case INTERBUS is not running and only internal values can be read in the XML file.

In the event of an I/O error all data is valid, except for the data of the faulty terminal.

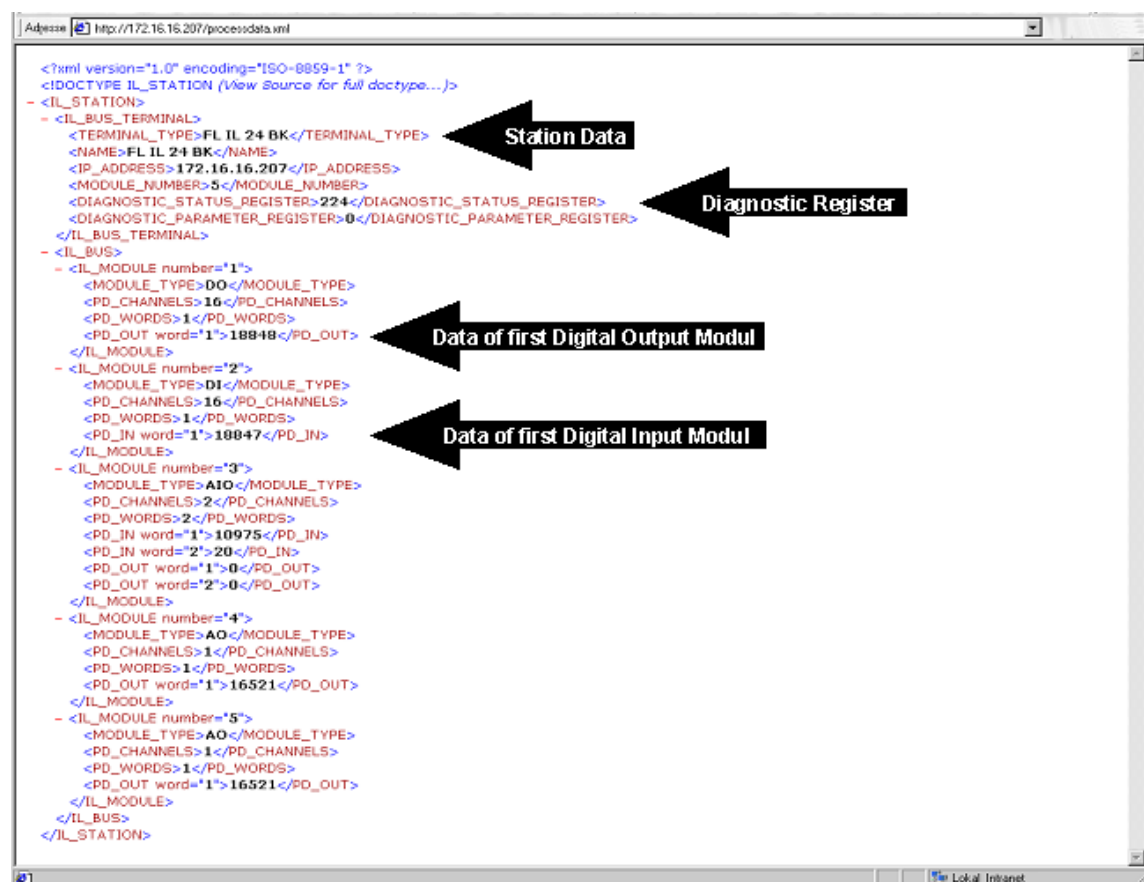


Figure 2-3 Screenshot of the XML data

2.8 Firmware Update



Boot loader <1.80 (see WBM: Device Information/General): When activating a firmware update, ensure that a valid firmware version is available. Otherwise the management part of the device attempts to update repeatedly and is unavailable for management and diagnostic functions.

2.8.1 Firmware Update Using The Factory Manager

The following steps must be carried out when executing a firmware update using the Factory Manager:

1. In the Device View window, right-click on the device whose firmware you want to update. Select "Properties" from the context menu and then the "BootP Parameter" tab.
2. Activate "BootP" and fill in the appropriate fields. For further information, please refer to the Factory Manager help.
3. Ensure that the BootP and tftp server for the Factory Manager are activated.
4. Open the web page for the bus coupler (context menu or Ctrl+W). Click on "Device Configuration" and then "Software Update". In the "Software Update on Next Reboot" field, click on "Enable".
5. Enter your password and click "Apply" to execute a reboot at a later time; click on "Apply and Reboot" for the update to take effect immediately.





FL IL 24 BK-PAC

[\[General Instructions \]](#)
[\[Device Information \]](#)
Device Configuration
[IP Configuration](#)
[SNMP Configuration](#)
☒ **Software Update**
[Change Password](#)
[Watchdog](#)
[\[Inline Station \]](#)
[\[Home \]](#)

FL IL 24 BK-PAC
last update: 10:32:28

Software Update

TFTP Server IP Address	TFTP:// 149.208.18.40
Downloadable File Name	 fl_il_bk_v1_10.bin
Software Update on Next Reboot	☐ Enable ☒ Disable
TFTP Update Status	Firmware Update was successfully executed.

If the software update status is set to enable the FL IL 24 BK will try to load new software within the next reboot. Press the apply button to change the software update status. The settings will take effect after the next reboot of the FL IL 24 BK.

Figure 2-4 WBM firmware update



The display indicates "03" (requesting firmware download at tftp server), then "04" (downloading firmware to memory) and finally "05" (firmware transfer to memory complete). The bus coupler is then automatically restarted.

Boot loader <1.80: If the display indicates "17" after the update, restart the bus coupler and repeat the update using a valid firmware.

Bootloader ≥1.80: If the display indicates "17" after the update, the bus coupler starts using the firmware already available. Check the firmware version on the "Device Information" web page and repeat the update, if necessary.

2.9 Firmware Update Using Web-Based Management (WBM) Without Factory Manager

The following steps must be carried out when executing a firmware update using WBM:

1. Open the web page for the bus coupler, by entering the IP address for the bus coupler in the address line of a standard web browser. After the web page has been loaded, click on "Device Configuration" and then "Software Update". Enter the IP address of the tftp server in the "TFTP Server IP Address" field. Then enter the file name of the firmware and the path name, if necessary, in the "Downloadable File Name" field. In the "Software Update on Next Reboot" field, click on "Enable".
2. Enter your password and click "Apply" to execute a reboot at a later time; click on "Apply and Reboot" for the update to take effect immediately.



The display indicates "03" (requesting firmware download at tftp server), then "04" (downloading firmware to memory) and finally "05" (firmware transfer to memory complete). The bus coupler then restarts automatically.

2.9.1 Trap Generation

When important events occur, e.g., a configuration change, the bus coupler sends a trap to a trap manager defined by the user. This enables the network administrator to react quickly to these events and to ensure network availability. Traps are usually only transmitted once.

2.9.2 Representation of Traps in the Factory Manager

ID	Date	Time	Source	Text
2000	31.1.2001	08:28:32	Trap-Receiver	-->SNMP-Trap (LinkUp an Slice 1, Port 4) received from Inline-I/O Robot (192.168. 2. 80).
2000	31.1.2001	08:28:32	Trap-Receiver	-->SNMP-Trap (LinkUp an Slice 1, Port 3) received from Inline-I/O Robot (192.168. 2. 80).
2000	31.1.2001	08:28:32	Trap-Receiver	-->SNMP-Trap (LinkUp an Slice 1, Port 1) received from Inline-I/O Robot (192.168. 2. 80).
2000	31.1.2001	08:28:31	Trap-Receiver	-->SNMP-Trap (ColdStart) received from Inline-I/O Robot (192.168. 2. 80).
2000	31.1.2001	08:28:31	Trap-Receiver	-->SNMP-Trap (ColdStart) received from Inline-I/O Robot (192.168. 2. 80).

Figure 2-5 Trap representation in the Factory Manager using a few example traps

2.9.3 FL IL 24 BK(-PAC) Traps

The FL IL 24 BK(-PAC) supports five traps:

- ColdStart - sent twice each time the device is restarted.
- PasswordChange - sent after the password is changed successfully.
- FWHealth - sent after any changes to the firmware operating status.
- Configuration - sent after any changes to the hardware configuration.
- Authentification - wrong password for SNMP access.

2.9.4 Defining the Trap Manager

Traps must be sent to a trap manager to be evaluated. Two appropriate network devices can be defined as trap managers. Open the "SNMP Configuration" dialog box in the "Device Configuration" menu, and enter the IP addresses of the trap managers in the "First/Second Trap Manager IP Address" fields. Enter the password for write access and save with "Apply".



FL IL 24 BK-PAC
[\[General Instructions \]](#)
[\[Device Information \]](#)
Device Configuration
IP Configuration
● [SNMP Configuration](#)
[Software Update](#)
[Change Password](#)
[Watchdog](#)
[\[Inline Station \]](#)
[\[Home \]](#)

FL IL 24 BK-PAClast update: 12:48:29

SNMP Configuration

System Information

Name of Device	FL IL 24 BK
Description	Ethernet bus terminal
Physical Location	H10-2/4
Contact	Manager

Writing a new configuration can take several seconds!

Enter Password		Apply
----------------	--------------------------	-------

Trap Configuration

First Trap Manager IP Address	192.168.17.243
Second Trap Manager IP Address	192.168.233.78
Sending traps	☒ Enable ☐ Disable

Please enter IP addresses in dotted decimal notation (e.g., 172.16.16.230).

Enter Password		Apply
----------------	--------------------------	-------

Figure 2-6 Defining the trap manager



Up to five trap managers can be defined via SNMP.

2.10 Factory Line I/O Configurator

The Factory Line I/O configurator is a software package for the easy **configuration**, **startup** and **diagnostics** of Factory Line Ethernet bus couplers. In particular, process data exchange is supported via **OPC** in an easy-to-use manner.

You can find the software on the "CD FL IL 24 BK" CD, Order No. 28 32 06 9. The I/O configurator is divided into two parts: I/O browser and OPC configurator.

2.10.1 Factory Line I/O Browser

The I/O browser is used to create the bus configuration. From all supported modules, select those you want to use in your station later (offline configuration) or those you are using currently (online configuration). During online configuration you have the possibility to read and test an existing bus configuration.

Configuration

During system planning the I/O configurator offers an integrated online product catalog using XML technology to help ensure optimum startup. You have access to all supported Inline terminals which can be integrated into the Inline local bus by using drag and drop. In the following I/O browser window, the bus structure is displayed on the left and the product catalog on the right.

Startup

After installing the hardware you can start up the stations based on the configured data.

Diagnostics

You can test the operating status of the stations at any time and also receive comprehensive support on correcting any errors using the integrated INTERBUS technology.

Configuring an Inline station using the I/O configurator

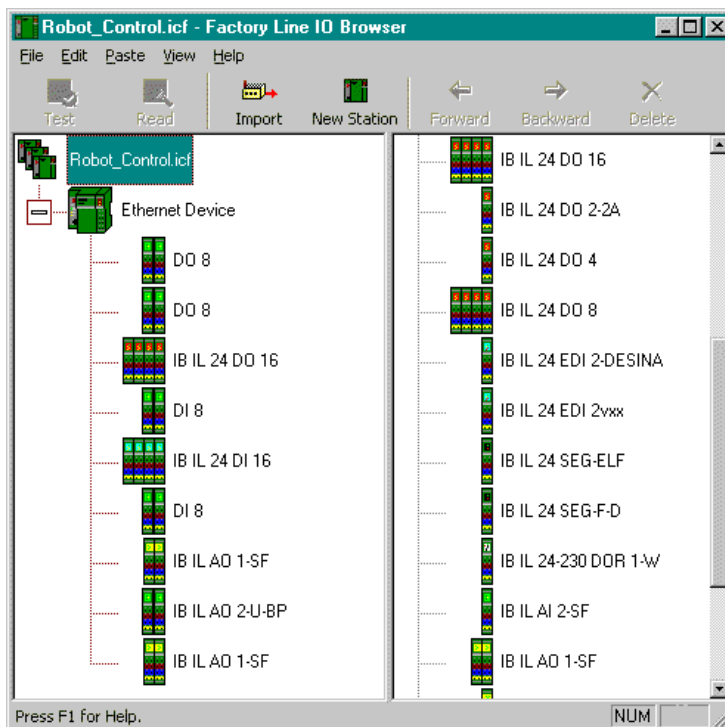


Figure 2-7 I/O browser screen

2.10.2 OPC Configurator

OPC Data Exchange

Process data exchange via OPC is supported in a very easy-to-use manner. OPC items are assigned to the Inline station structure for the relevant terminal points using the OPC configurator. You can configure the INTERBUS OPC server from Phoenix Contact (designation IBS OPC SERVER, Order No. 27 29 12 7) for this bus coupler type using the OPC configurator. The project file and an OPC server provide the application program or the visualization with direct access to the process data for the bus configuration.

Linking Items and Physical Terminal Points

An item can be created for each physical I/O terminal in your bus configuration and the entire configuration can be stored in a project file. The project file and an OPC server provide the application program or the visualization with direct access to the process data for the bus configuration.

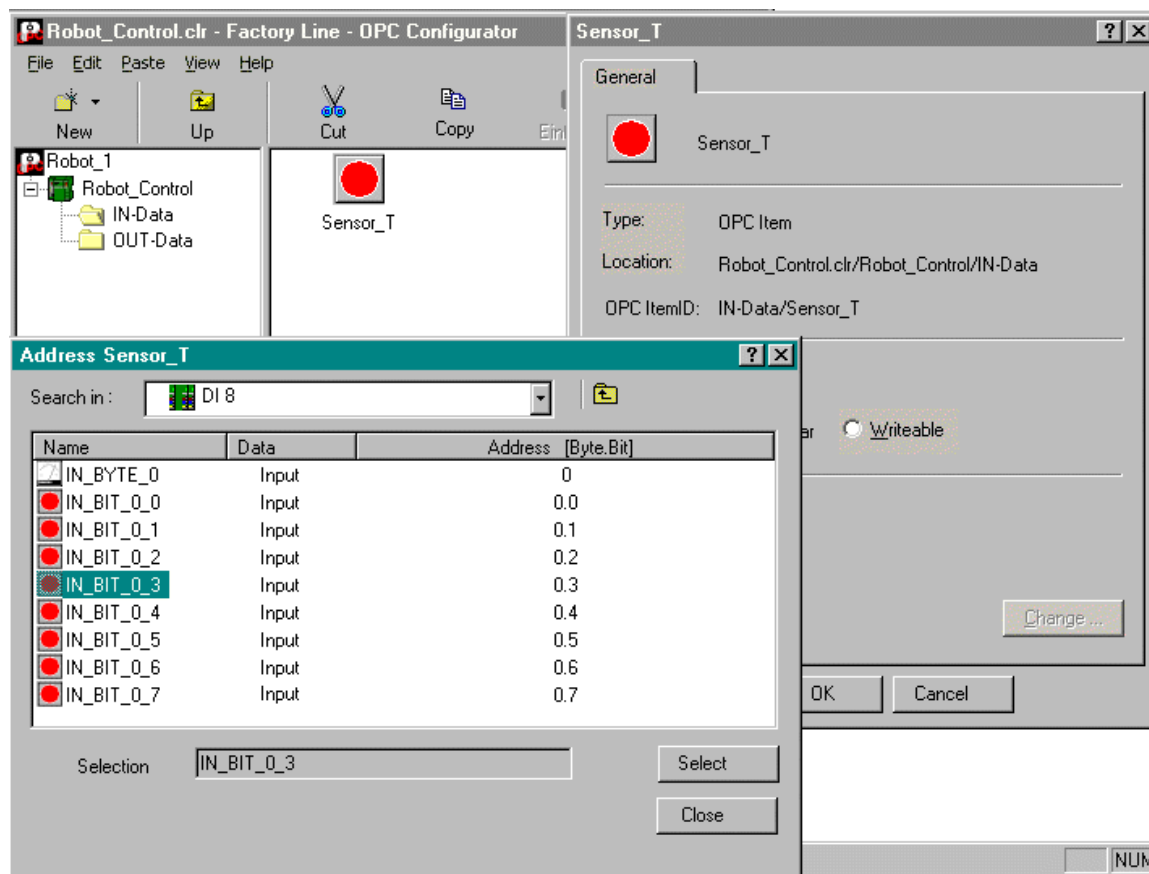


Figure 2-8 Linking items and terminal points



The entire configuration can be carried out offline.

Startup

After the hardware has been installed, the bus configuration can either be configured online or started up using the project file.

Diagnostics

The operating state of the Inline station can be checked at any time. The comprehensive diagnostic functions provide support when removing errors from the local bus (configuration).

OPC Communication

Configure the OPC server from Phoenix Contact for this type of bus coupler using the project file that was created using this software. The project file and an OPC server provide the application program or the visualization with direct access to the process data for the bus configuration.

2.11 Management Information Base - MIB

The FL IL 24 BK(-PAC) supports the following MIBs:

Standard MIB:

RFC 1213 (MIB II)

Private MIBs:

PhoenixContact MIB, FL MIB (Factory Line MIB) and FL DEVICE MIB.

2.11.1 Standard MIBs:

2.11.1.1 RFC-1213 (MIB II)

System Group (1.3.6.1.2.1.1)

The system group contains information about device management.

(1) interfaces

- (1) sysDescr
- (2) sysObjektID
- (3) sysUpTime
- (4) sysContact
- (5) sysName
- (6) sysLocation
- (7) sysServices

2.12 Interface Group (1.3.6.1.2.1.2)

The interface group contains information about device interfaces.

```
(2) interfaces
-- (1) ifNumber
-- (2) ifTable
-- (1) if Entry
-- (1) ifIndex
-- (2) ifDescr
-- (3) ifType
-- (4) ifMtu
-- (5) ifSpeed
-- (6) ifPhysAddress
-- (7) ifAdminStatus
-- (8) ifOperStatus
-- (9) ifLastChange
-- (10) ifInOctets
-- (11) ifInUcastPkts
-- (12) ifInNUcastPkts
-- (13) ifInDiscards
-- (14) ifInErrors
-- (15) ifInUnknownProtos
-- (16) ifOutOctets
-- (17) ifOutUcastPkts
-- (18) ifOutNUcastPkts
-- (19) ifOutDiscards
-- (20) ifOutErrors
-- (21) ifOutQLen
-- (22) ifSpecific
```

Address Translation Group - AT (1.3.6.1.2.1.3)

The address translation group is mandatory for all systems. It contains information about the address assignment.

```
(3) at
-- (1) atTable
-- (1) atEntry
-- (1) atIfIndex
-- (2) atPhysAddress
-- (3) atNetAddress
```

Internet Protocol Group - IP (1.3.6.1.2.1.4)

The Internet protocol group is mandatory for all systems. It contains information concerning IP switching.

- (4) ip
 - (1) ipForwarding
 - (2) ipDefaultTTL
 - (3) ipInReceives
 - (4) ipInHdrErrors
 - (5) ipInAddrErrors
 - (6) ipForwDatagrams
 - (7) ipInUnknownProtos
 - (8) ipInDiscards
 - (9) ipInDelivers
 - (10) ipOutRequests
 - (11) ipOutDiscards
 - (12) ipOutNoRoutes
 - (13) ipReasmTimeout
 - (14) ipReasmReqds
 - (15) ipReasmOKs
 - (16) ipReasmFails
 - (17) ipFragOKs
 - (18) ipFragFails
 - (19) ipFragCreates
 - (20) ipAddrTable
 - (1) ipAddrEntry
 - (1) ipAdEntAddr
 - (2) ipAdEntIfIndex
 - (3) ipAdEntNetMask
 - (4) ipAdEntBcastAddr
 - (5) ipAdEntReasmMaxSize
 - (21) ipRouteTable
 - (1) ipRouteEntry
 - (1) ipRouteDest
 - (2) ipRouteIfIndex
 - (3) ipRouteMetric1
 - (4) ipRouteMetric2
 - (5) ipRouteMetric3
 - (6) ipRouteMetric4
 - (7) ipRouteNextHop
 - (8) ipRouteType
 - (9) ipRouteProto
 - (10) ipRouteAge

- (11) ipRouteMask
- (12) ipRouteMetric5
- (13) ipRouteInfo
- (22) ipNetToMediaTable
 - (1) ipNetToMediaEntry
 - (1) ipNetToMediaIflIndex
 - (2) ipNetToMediaPhysAddress
 - (3) ipNetToMediaNetAddress
 - (4) ipNetToMediaType
- (23) ipRoutingDiscards

ICMP Group (1.3.6.1.2.1.5)

The Internet control message protocol group is mandatory for all systems. It contains information about error treatment and control in Internet data traffic.

- (5) icmp
 - (1) icmpInMsgs
 - (2) icmpInErrors
 - (3) icmpInDestUnreachs
 - (4) icmpInTimeExcds
 - (5) icmpInParmProbs
 - (6) icmpInSrcQuenchs
 - (7) icmpInRedirects
 - (8) icmpInEchos
 - (9) icmpInEchoReps
 - (10) icmpInTimestamps
 - (11) icmpInTimestampReps
 - (12) icmpInAddrMasks
 - (13) icmpInAddrMaskReps
 - (14) icmpOutMsgs
 - (15) icmpOutErrors
 - (16) icmpOutDestUnreachs
 - (17) icmpOutTimeExcds
 - (18) icmpOutParmProbs
 - (19) icmpOutSrcQuenchs
 - (20) icmpOutRedirects
 - (21) icmpOutEchos
 - (22) icmpOutEchoReps
 - (23) icmpOutTimestamps
 - (24) icmpOutTimestampReps
 - (25) icmpOutAddrMasks
 - (26) icmpOutAddrMaskReps

Transfer Control Protocol Group - TCP (1.3.6.1.2.1.6)

The transfer control protocol group is mandatory for all systems that implement TCP. Instances for objects which provide information about a specific TCP connection apply as long as the connection is established.

- (6) tcp
 - (1) tcpRtoAlgorithm
 - (2) tcpRtoMin
 - (3) tcpRtoMax
 - (4) tcpMaxConn
 - (5) tcpActiveOpens
 - (4) ipRouteMetric2
 - (6) tcpPassiveOpens
 - (7) tcpAttemptFails
 - (8) tcpEstabResets
 - (9) tcpCurrEstab
 - (10) tcpInSegs
 - (11) tcpOutSegs
 - (12) tcpRetransSegs
 - (13) tcpConnTable
 - (1) tcpConnEntry
 - (1) tcpConnState
 - (2) tcpConnLocalAddress
 - (3) tcpConnLocalPort
 - (4) tcpConnRemAddress
 - (5) tcpConnRemPort
 - (14) tcpInErrs
 - (15) tcpOutRsts

User Datagram Protocol Group - UDP (1.3.6.1.2.1.7)

The user datagram protocol group is mandatory for all systems that implement UDP.

- (7) udp
 - (1) udpInDatagrams
 - (2) udpNoPorts
 - (3) udpInErrors
 - (4) udpOutDatagrams
 - (5) udpTable
 - (1) udpEntry
 - (1) udpLocalAddress
 - (2) udpLocalPort

EGP (1.3.6.1.2.1.8)

The EGP group is mandatory for all systems that implement EGP.

- (8) egp
 - (1) egpInMsgs
 - (2) egpInErrors
 - (3) egpOutMsgs
 - (4) egpOutErrors
 - (5) egpNeighTable
 - (1) egpNeighEntry
 - (1) egpNeighState
 - (2) egpNeighAddr
 - (3) egpNeighAs
 - (4) egpNeighInMsgs
 - (5) egpNeighInErrs
 - (6) egpNeighOutMsgs
 - (7) egpNeighOutErrs
 - (8) egpNeighInErrMsgs
 - (9) egpNeighOutErrMsgs
 - (10) egpNeighStateUps
 - (11) egpNeighStateDowns
 - (12) egpNeighIntervalHello
 - (13) egpNeighIntervalPoll
 - (14) egpNeighMode
 - (15) egpNeighEventTrigger
 - (6) egpAs

Simple Network Management Protocol Group (1.3.6.1.2.1.11)

The simple network management protocol group is mandatory for all systems. In SNMP devices, which are optimized to support either a single agent or a single management station, some of the listed objects will be overwritten with the value "0".

- (11) snmp
 - (1) snmpInPkts
 - (2) snmpOutPkts
 - (3) snmpInBadVersions
 - (4) snmpInBadCommunityNames
 - (5) snmpInBadCommunityUses
 - (6) snmpInASNParseErrs
 - (7) not used
 - (8) snmpInTooBig
 - (9) snmpInNoSuchNames
 - (10) snmpInBadValues

- (11) snmplnReadOnlys
- (12) snmplnGenErrs
- (13) snmplnTotalReqVars
- (14) snmplnTotalSetVars
- (15) snmplnGetRequests
- (16) snmplnGetNexts
- (17) snmplnSetRequests
- (18) snmplnGetResponses
- (19) snmplnTraps
- (20) snmpOutTooBig
- (21) snmpOutNoSuchNames
- (22) snmpOutBadValues
- (23) not used
- (24) snmpOutGenErrs
- (25) snmpOutGetRequests
- (26) snmpOutGetNexts
- (27) snmpOutSetRequests
- (28) snmpOutGetResponses
- (29) snmpOutTraps
- (30) snmpEnableAuthenTraps

2.12.1 Private MIBs

2.12.1.1 PhoenixContact MIB

The PhoenixContact MIB contains manufacturer information.

The pxcModules (OID = 1.3.6.1.4.1.4346.1) and pxcGlobal (OID = 1.3.6.1.4.1.4346.2) groups are described in this private Phoenix Contact MIB (OID = 1.3.6.1.4.1.4346).

MIB structure:

- (1) pxcModules
 - (1) pxcRootModule
- (2) pxcGlobal
 - (1) pxcBasic
 - (1) pxcBasicName
 - (2) pxcBasicDescr
 - (3) pxcBasicURL

pxcBasicName

OID	1.3.6.1.4.1.4346.2.1.1
Syntax	Display string
Access	Read
Description	Contains the manufacturer name, Phoenix Contact GmbH & Co. KG

pxcBasicDescr

OID	1.3.6.1.4.1.4346.2.1.2
Syntax	Display string
Access	Read
Description	Contains the manufacturer name and address, Phoenix Contact GmbH & Co. KG P.O. Box 1341 D-32819 Blomberg

pxcBasicURL

OID	1.3.6.1.4.1.4346.2.1.3
Syntax	Display string
Access	Read
Description	Contains the URL of the manufacturer, http://www.phoenixcontact.com

2.12.1.2 FL MIB

The FL MIB contains information about the Factory Line product group.

This private FL MIB (OID = 1.3.6.1.4.1.4346) describes the pxcFactoryLine (OID = 1.3.6.1.4.1.4346.11) group.

MIB structure:

- (1) pxcModules
 - (2) pxcFLModule
- (11) pxcFactoryLine
 - (1) flGlobal
 - (1) flBasic
 - (1) flBasicName
 - (2) flBasicDescr
 - (3) flBasicURL
 - (4) flBasicCompCapacity

- (2) flComponents
 - (1) flComponentsTable
 - (1) flComponentsEntry
 - (1) flComponentsIndex
 - (2) flComponentsName
 - (3) flComponentsDescr
 - (1) flComponentsURL
 - (1) flComponentsOrderNumber

flBasicName

OID	1.3.6.1.4.1.4346.11.1.1.1
Syntax	Display string
Access	Read
Description	Contains the name of the product group, Factory Line

flBasicDescr

OID	1.3.6.1.4.1.4346.11.1.1.2
Syntax	Display string
Access	Read
Description	Contains a brief description of the product group, Ethernet installation system

flBasicURL

OID	1.3.6.1.4.1.4346.11.1.1.3
Syntax	Display string
Access	Read
Description	Contains a URL for the product group, http://www.factoryline.de

flBasicCompCapacity

OID	1.3.6.1.4.1.4346.11.1.1.4
Syntax	Integer32 (1...1024)
Access	Read
Description	Contains the number of different components that can be controlled by this device.

flComponentsTable - flComponentsEntry

OID	1.3.6.1.4.1.4346.11.1.2.1.1
Syntax	
Access	
Description	Generates a table with descriptions for components in the "Factory Line" product group, which can be controlled by this management device.
flComponentsIndex	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.1
Syntax	Integer32 (1 ... 1024)
Access	Read
Description	Contains the component product index
flComponentsName	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.2
Syntax	Display string
Access	Read
Description	Contains the designation of the component
flComponentsDescr	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.3
Syntax	Display string
Access	Read
Description	Contains a brief description of the component
flComponentsURL	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.4
Syntax	Display string
Access	Read
Description	Contains the URL of a web page with additional information: www.factoryline.de
flComponentsOrderNumber	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.5
Syntax	Display string
Access	Read
Description	Contains the order number of the component

2.12.1.3 FL Device MIB

The FL Device MIB contains general information about components from the Factory Line product group.

This private FL Device MIB (OID = 1.3.6.1.4.1.4346) describes one part of the pxcFactoryLine (OID = 1.3.6.1.4.1.4346.11) group.

MIB structure:

- (1) pxcModules
 - (3) flDeviceModule
- (11) pxcFactoryLine
 - (11) flWorkDevice
 - (1) flWorkBasic
 - (1) flWorkBasicName
 - (2) flWorkBasicDescr
 - (3) flWorkBasicUrl
 - (4) flWorkBasicSerialNumber
 - (5) flWorkBasicHWRevision
 - (11) flWorkBasicCompMaxCapacity
 - (12) flWorkBasicCompCapacity
 - (2) flWorkComponents
 - (1) flWorkComponentsTable
 - (1) flWorkComponentsEntry
 - (1) flWorkComponentsIndex
 - (2) flWorkComponentsOID
 - (3) flWorkComponentsURL
 - (4) flWorkComponentsDevSign
 - (5) flWorkComponentsPowerStat
 - (11) flWorkComponentsStrongReset
 - (3) flWorkTraps
 - (0) flWorksTrapsDelemeter
 - (1) flWorkFWPasswdAccess
 - (2) flWorkFWHealth
 - (3) flWorkFWConf
 - (11) flWorkFirmware
 - (1) flWorkFWInfo
 - (1) flWorkFWInfoVersion
 - (2) flWorkFWInfoState
 - (3) flWorkFWInfoDate
 - (4) flWorkFWInfoTime
 - (5) flWorkFWInfoCopyright
 - (6) flWorkFWInfoBootVersion
 - (7) flWorkFWInfoBootState
 - (8) flWorkFWInfoBootDate

```

-- (9) flWorkFWInfoBootTime
-- (11) flWorkFWInfoOperStatus
-- (12) flWorkFWInfoHealthText
-- (2) flWorkFWCtrl
-- (1) flWorkFWCtrlBasic
-- (1) flWorkFWCtrlReset
-- (2) flWorkFWCtrlTrapDestCapacity
-- (2) flWorkFWCtrlTrapDest
-- (1) flWorkFWCtrlTrapDestTable
-- (1) flWorkFWCtrlTrapDestEntry
-- (1) flWorkFWCtrlTrapDestIndex
-- (2) flWorkFWCtrlTrapDestIPAddr
-- (3) flWorkFWCtrlPasswd
-- (1) flWorkFWCtrlPasswdSet
-- (2) flWorkFWCtrlPasswdSuccess
-- (4) flWorkFWCtrlUpdate
-- (1) flWorkFWCtrlUpdateEnable
-- (2) flWorkFWCtrlTftpIPAddr
-- (3) flWorkFWCtrlTftpFile
-- (5) flWorkFWCtrlConf
-- (1) flWorkFWCtrlConfStatus
-- (11) flWorkFWInfo
-- (1) flWorkFWParamSaveConfig
-- (12) flWorkRprr

```

flWorkBasicName

OID	1.3.6.1.4.1.4346.11.11.1.1
Syntax	Display string
Access	Read/write
Description	Contains the device name (corresponds to "sysName" from MIB2)

flWorkBasicDescr

OID	1.3.6.1.4.1.4346.11.11.1.2
Syntax	Display string
Access	Read/write
Description	Contains a brief description (corresponds to "sysDescr" from MIB2)

flWorkBasicName

OID	1.3.6.1.4.1.4346.11.11.1.3
Syntax	Display string
Access	Read
Description	Contains the URL of the device-specific web page for WBM

flWorkBasicSerialNumber

OID	1.3.6.1.4.1.4346.11.11.1.4
Syntax	Octet string (12)
Access	Read
Description	Contains the serial number of the device

flWorkBasicHWRevision

OID	1.3.6.1.4.1.4346.11.11.1.5
Syntax	Octet string (4)
Access	Read
Description	Contains the hardware version of the device

flWorkBasicCompMaxCapacity

OID	1.3.6.1.4.1.4346.11.11.1.11
Syntax	Integer32 (1...1024)
Access	Read
Description	Contains the maximum possible number of devices that can be connected

flWorkBasicCompCapacity

OID	1.3.6.1.4.1.4346.11.11.1.12
Syntax	Integer32 (1...1024)
Access	Read
Description	Contains the actual number of connected devices

flWorkComponentsTable - flWorkComponentsEntry

OID	1.3.6.1.4.1.4346.11.11.2.1.1
Syntax	
Access	
Description	Generates a table with the description of individual components

flWorkComponentsIndex	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.1
Syntax	Integer32 (1 ... 1024)
Access	Read
Description	Contains the index of the component
flWorkComponentsOID	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.2
Syntax	OBJECT IDENTIFIER
Access	Read
Description	Contains the designation of OIDs/complete path entries
flComponentsURL	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.3
Syntax	Display string
Access	Read
Description	Contains the URL of the web page of this component with additional information
flWorkComponentsDevSign	
OID	1.3.6.1.4.1.4346.11.11.2.1.1.4
Syntax	INTEGER (0 ... 255)
Access	Read
Description	Contains the index entry of the component
flWorkComponentsPowerStat	
OID	1.3.6.1.4.1.4346.11.11.2.1.1.5
Syntax	INTEGER
Access	Read
Description	Contains status information about the connected supply voltages:
	- Unknown 1
	- No voltage present 2
	- Supply voltage 1 OK 3
	- Supply voltage 2 OK 4
	- Supply voltages 1 and 2 OK 5

flWorkComponentsStrongReset

OID	1.3.6.1.4.1.4346.11.11.2.1.1.11
Syntax	INTEGER
Access	Read/write
Description	With write access, a reset can be executed with "2". With read access, the value is always "1" - no reset.

flWorkFWInfoVersion

OID	1.3.6.1.4.1.4346.11.11.11.1.1
Syntax	Octet string (4)
Access	Read
Description	Contains the firmware version as a string. Example for version "3.97": 0x33, 0x2e, 0x39, 0x37

flWorkFWInfoState

OID	1.3.6.1.4.1.4346.11.11.11.1.2
Syntax	Octet string (6)
Access	Read
Description	Contains the firmware release as a string. Example for "beta": 0x62, 0x65, 0x64, 0x61

flWorkFWInfoDate

OID	1.3.6.1.4.1.4346.11.11.11.1.3
Syntax	Octet string (6)
Access	Read
Description	Contains the creation date of the firmware version as a string. Example for "21.05.2001": 0x32, 0x31, 0x30, 0x35, 0x30, 0x31

flWorkFWInfoTime

OID	1.3.6.1.4.1.4346.11.11.11.1.4
Syntax	Octet string (6)
Access	Read
Description	Contains the creation time of the firmware version as a string. Example for "14:10:20": 0x31, 0x34, 0x31, 0x30, 0x32, 0x30

flWorkFWInfoCopyright

OID	1.3.6.1.4.1.4346.11.11.11.1.5
Syntax	Display string (6)
Access	Read
Description	Contains the owner of the firmware copyright. Copyright by Phoenix Contact GmbH & Co. KG, 2000

flWorkFWInfoBootVersion

OID	1.3.6.1.4.1.4346.11.11.11.1.6
Syntax	Octet string (4)
Access	Read
Description	Contains the version of the Boot loader as a string. Example for version "2.65": 0x32, 0x2e, 0x36, 0x35

flWorkFWInfoBootState

OID	1.3.6.1.4.1.4346.11.11.11.1.7
Syntax	Octet string (6)
Access	Read
Description	Contains the Boot loader release as a string. Example for "beta": 0x62, 0x65, 0x64, 0x61

flWorkFWInfoBootDate

OID	1.3.6.1.4.1.4346.11.11.11.1.8
Syntax	Octet string (6)
Access	Read
Description	Contains the creation date of the Boot loader version as a string. Example for "09.03.2001": 0x30, 0x39, 0x30, 0x33, 0x30, 0x31

flWorkFWInfoBootTime

OID	1.3.6.1.4.1.4346.11.11.11.1.7
Syntax	Octet string (6)
Access	Read
Description	Contains the creation time of the Boot loader version as a string. Example for "14:10:20": 0x31, 0x34, 0x31, 0x30, 0x32, 0x30

flWorkFWInfoBootStatus

OID	1.3.6.1.4.1.4346.11.11.11.1.11
Syntax	Integer
Access	Read
Description	Contains the operating state of the firmware. - Problem 1 - No problem 2

flWorkFWInfoHealthText

OID	1.3.6.1.4.1.4346.11.11.11.1.12
Syntax	Display string
Access	Read
Description	Contains additional information/error states of the firmware.

flWorkFWCtrlReset

OID	1.3.6.1.4.1.4346.11.11.11.2.1.1
Syntax	Integer
Access	Read/write
Description	With write access, a reset can be executed with "2". With read access, the value is always "1".

flWorkFWCtrlTrapDestCapacity

OID	1.3.6.1.4.1.4346.11.11.11.2.1.2
Syntax	Integer32 (1 ... 1024)
Access	Read
Description	Contains the number of devices to which the traps are sent.

flWorkFWCtrlTrapDestTable - flWorkFWCtrlTrapDestEntry

OID	1.3.6.1.4.1.4346.11.11.11.2.2.1.1
Syntax	
Access	
Description	Generates a table with the IP addresses of the trap managers

flWorkFWCtrlTrapDestIndex	
OID	1.3.6.1.4.1.4346.11.11.11.2.2.1.1.1
Syntax	Integer32 (1 ... 1024)
Access	Read
Description	Contains the index of the target component, which is to receive the traps
flWorkFWCtrlTrapDestIPAddr	
OID	1.3.6.1.4.1.4346.11.1.2.1.1.2
Syntax	IP address
Access	Read/write
Description	Contains the IP address of the target component, which is to receive the traps

flWorkFWCtrlPasswdSet

OID	1.3.6.1.4.1.4346.11.11.11.2.3.1
Syntax	Octet String (2 ... 24)
Access	Read/write



For security reasons, the response is always "*****" with read access.

Description	A new password can be entered here with a maximum of 12 characters. Example: - Your new password should be "<i>factory3</i>". - The password must be entered a second time for confirmation. - Your entry "<i>factory3factory3</i>". - Your password for write access now is: "<i>factory3</i>"
-------------	---

flWorkFWCtrlPasswdSuccess

OID	1.3.6.1.4.1.4346.11.11.11.2.3.2						
Syntax	Integer						
Access	Read						
Description	A message is displayed, which informs you whether the last change of password was successful. <table> <tr> <td>- Unknown</td><td>1</td></tr> <tr> <td>- Failed</td><td>2</td></tr> <tr> <td>- Successful</td><td>3</td></tr> </table>	- Unknown	1	- Failed	2	- Successful	3
- Unknown	1						
- Failed	2						
- Successful	3						

fiWorkFWCtrlUpdateEnable

OID	1.3.6.1.4.1.4346.11.11.11.2.4.1
Syntax	Integer
Access	Read/write
Description	A firmware update can be executed here on the next manual restart/reset of the device: - Start with existing firmware 1 - Update firmware 2

fiWorkFWCtrlTftpIPAddr

OID	1.3.6.1.4.1.4346.11.11.11.2.4.2
Syntax	IP address
Access	Read/write
Description	Enter the IP address of the tftp server where the (new) firmware can be found.

fiWorkFWCtrlTftpFile

OID	1.3.6.1.4.1.4346.11.11.11.2.4.3
Syntax	Octet String (0 ... 64)
Access	Read/write
Description	Enter the file name of the (new) firmware here.

fiWorkFWCtrlConfStatus

OID	1.3.6.1.4.1.4346.11.11.11.2.5.1
Syntax	INTEGER
Access	Read
Description	Contains a status message about the current hardware configuration: - Configuration OK 1 - Configuration faulty 2 - Configuration saved 3

fiWorkFWParamSaveConfig

OID	1.3.6.1.4.1.4346.11.11.11.11.1
-----	--------------------------------

Syntax	INTEGER
Access	Read/write
Description	The current configuration can be saved in the EEPROM: - Do not save configuration 1 (has no effect) - Save configuration 2 With read access, the value is always "1".

2.13 Meaning of the 7-Segment Display

Table 2-2 During startup/operation:

Display	Meaning
01	Boot loader is started, BootP requests are sent
bo	Firmware is extracted
02	Firmware is started
--	Operating

Table 2-3 Additional information:

Display	Meaning
PP	P&P mode is activated

Table 2-4 During firmware update:

Display	Meaning
03	The firmware is downloaded from the tftp server
04	The firmware is downloaded to the memory
05	The firmware transfer to the memory is complete

Table 2-5 Boot loader error messages:

Display	Meaning	Remedy
17	The transfer of the firmware failed during tftp download (display changes from "03" to "17")	- Check the physical connection - Establish a point-to-point connection - Make sure that the file (with the specified file name) exists and is in the correct directory - Check the IP address of the tftp server - Activate the tftp server - Repeat the download Boot loader <1.80: Restart the bus coupler and repeat the update using a valid firmware. Boot loader ≥1.80: The bus coupler starts with the firmware already available. Check the firmware version on the "Device Information" web page and repeat the update, if necessary.
19	The tftp download was completed successfully, but the file is not a valid firmware version for the bus coupler	- Provide a valid firmware version with the previously specified file name - Repeat the download Boot loader <1.80: Restart the bus coupler and repeat the update using a valid firmware. Boot loader ≥1.80: By a reset the bus coupler starts with the firmware already available. Check the firmware version on the "Device Information" web page and repeat the update, if necessary.



The points under "Remedy" are recommendations; they do not all have to be carried out for every error.

Table 2-6 Firmware error messages:

Display	Meaning	Remedy
80	An error occurred in the firmware	- Restart the device (Power up or reset)
81	An error occurred when accessing the EEPROM	- Restart the device (Power up or reset)
82	The known configuration could not be activated	- Use "Get_Error_Info" to check whether any faulty modules are present

Table 2-6 Firmware error messages: (Contd.)

Display	Meaning	Remedy
83	The known configuration could not be activated because the known configuration and the reference configuration are not the same	- Create a configuration which corresponds to the reference configuration - Activate P&P mode - Store a new reference configuration (Create_Configuration)
8A	Process data watchdog triggered	Safe operation of the station (e.g., due to excessive network load) can no longer be guaranteed. A station reset must be carried out.
bF	Bus error, the bus was stopped due to an error	- Check the INTERBUS devices (Get_Diag_Info)
nF	Network error	- Check the Ethernet connection (Process data watchdog activated)
nC	No MODBUS client connected to bus coupler; no process data watchdog active	- Check the Ethernet connection - The application program was stopped
PF	I/O error	- The I/O error must be removed and acknowledged via DDI or web page



The points under "Remedy" are recommendations; they do not all have to be carried out for every error.



For all other error codes displayed, please contact Phoenix Contact (see final page).

Display in Modbus/TCP Operation

On the 7-segment display a connected Modbus/TCP device is indicated by a decimal point. If the connection to a Modbus/TCP device fails and no other error is present, "nC" ("not connected") is displayed. In addition, the bus coupler activates the fault response mode specified by the user.

If no Modbus/TCP connections was activated, neither a decimal point nor "nC" are displayed.

If several errors occur at the same time, the error with the highest priority is displayed. For the priority of the individual errors, please refer to the following table.

Table 2-7 Priority of the error messages

Priority	Display	Meaning
1	8x	Firmware error
2	bF	Bus error, the bus was stopped due to an error
3	nF	Network error; the monitoring function detected an error - Modbus/TCP connection interrupted - No process data control watchdog active
4	PF	I/O error of the Inline modules
5	PP	Plug & play mode activated
6	nC	No MODBUS client connected to bus coupler; no process data watchdog active

The differences on the display for DDI mode and Modbus/TCP mode are shown in the following table. The remedy in Modbus/TCP mode corresponds to the remedy in DDI mode.

Table 2-8 Differences on the display in DDI mode and Modbus/TCP mode

DDI Mode		Modbus/TCP Mode	
Display	Meaning	Display	Meaning
--	Normal operation without errors	--.	Normal operation without errors, one Modbus/TCP device connected
bF	Bus error	bF.	Bus error, one Modbus/TCP device connected
		bF	Bus error, connection to Modbus/TCP device interrupted
nF	Network error - the monitoring function detected an error or the process data watchdog became active	nC	All Modbus/TCP connections interrupted (the connection was closed, the cable has been removed, or a communication error occurred). No process data watchdog active.
PF	I/O error of the Inline modules	nF.	Network error - the monitoring function detected an error or the process data watchdog became active
PP	Plug & play mode activated	PF.	I/O error of the Inline modules, one Modbus/TCP device connected
		PP.	Plug & play mode activated, one Modbus/TCP device connected

This section provides information about

- the driver software
- an example program

Driver Software	3-3
3.1 Documentation	3-3
3.1.1 Hardware and Firmware User Manual	3-3
3.2 Software Structure	3-3
3.2.1 Ethernet/Inline Bus Coupler Firmware	3-4
3.2.2 Driver Software	3-4
3.3 Support and Driver Update	3-5
3.4 Transfer of I/O Data	3-6
3.4.1 Position of the Process Data (Example)	3-7
3.5 Startup Behavior of the Bus Coupler	3-8
3.5.1 Plug & Play Mode	3-8
3.5.2 Expert Mode	3-9
3.5.3 Possible Combinations of the Modes	3-9
3.5.4 Startup Diagrams of the Bus Coupler	3-10
3.5.5 Changing and Starting a Configuration in P&P Mode	3-12
3.6 Changing a Reference Configuration Using the Software	3-13
3.6.1 Effects of Expert Mode	3-13
3.6.2 Changing a Reference Configuration	3-13
3.7 Description of the Device Driver Interface (DDI) Introduction	3-15
3.7.1 Overview	3-15
3.7.2 Working Method of the Device Driver Interface	3-16
3.7.3 Description and Functions of the Device Driver Interface	3-18
3.8 Monitoring Functions	3-33
3.8.1 Process Data Monitoring/ Process Data Watchdog	3-35
3.8.2 Connection Monitoring (Host Checking)	3-37
3.8.3 Data Interface (DTI) Monitoring	3-40
3.8.4 I/O Fault Response Mode	3-44
3.8.5 Treatment of the NetFail Signal/Testing With ETH_SetNetFail	3-46
3.9 IN Process Data Monitoring	3-52

3.10	Notification Mode	3-56
3.11	Programming Support Macros.....	3-58
3.11.1	Introduction	3-58
3.12	Description of the Macros.....	3-60
3.12.1	Macros for Converting the Data Block of a Command.....	3-61
3.12.2	Macros for Converting the Data Block of a Message.....	3-63
3.12.3	Macros for Converting Input Data	3-65
3.12.4	Macros for Converting Output Data	3-66
3.13	Diagnostic Options for Driver Software	3-68
3.13.1	Introduction	3-68
3.14	Positive Messages.....	3-69
3.15	Error Messages	3-70
3.15.1	General Error Messages.....	3-70
3.15.2	Error Messages When Opening a Data Channel.....	3-72
3.15.3	Error Messages When Transmitting Messages/Commands..	3-73
3.15.4	Error Messages When Transmitting Process Data.....	3-75
3.16	Example Program.....	3-78
3.16.1	Demo Structure Startup	3-78
3.16.2	Example Program Source Code	3-80

3 Driver Software

3.1 Documentation

This „Hardware and Firmware User Manual for FL IL 24 BK/FL IL 24 BK-PAC Ethernet/Inline Bus Coupler“ (Order No. 90 14 20 5) describes the hardware and software functions in association with an Ethernet network and the functions of the Device Driver Interface (DDI) software.

3.2 Software Structure

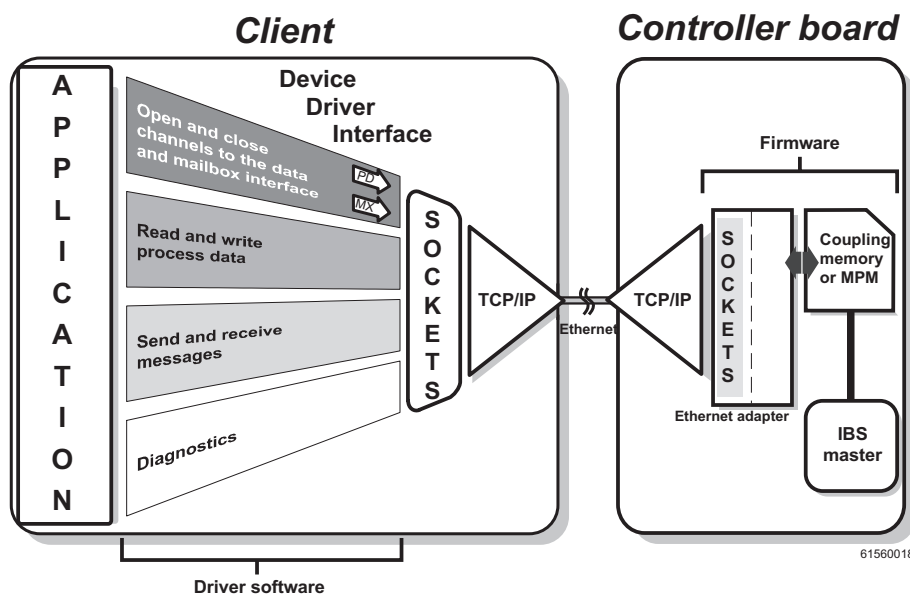


Figure 3-1 Software structure

3.2.1 Ethernet/Inline Bus Coupler Firmware

The Ethernet/Inline bus coupler firmware controls the Inline functions and Ethernet communication, shown on the right-hand side in Figure 3-1.

The bus coupler provides a basic interface for using services via the Ethernet network. The software primarily encodes and decodes the data telegrams for addressing the bus coupler services. The firmware also ensures the network-specific addressing of the bus coupler in the network, i.e., the management of IP parameters.

3.2.2 Driver Software

The driver software (DDI) enables the creation of an application program, shown on the left-hand side in Figure 3-1. A library is available for Sun Solaris 2.4. Due to the large variety of different operating systems, the driver software is available as source code in *IBS ETH DDI SWD E* (Order No. 27 24 19 3).

The driver software can be divided into three groups. The Device Driver Interface functions form the first group, which controls the bus coupler via the Ethernet network. Using these functions, firmware services can be called and started, and results can be requested on the bus coupler. The second group contains functions for monitoring the bus coupler and the workstation with the application program. The third group contains macro functions for the conversion of data between Intel and Motorola data format.

Figure 3-2 illustrates the creation of an application program from the parts of the driver software.

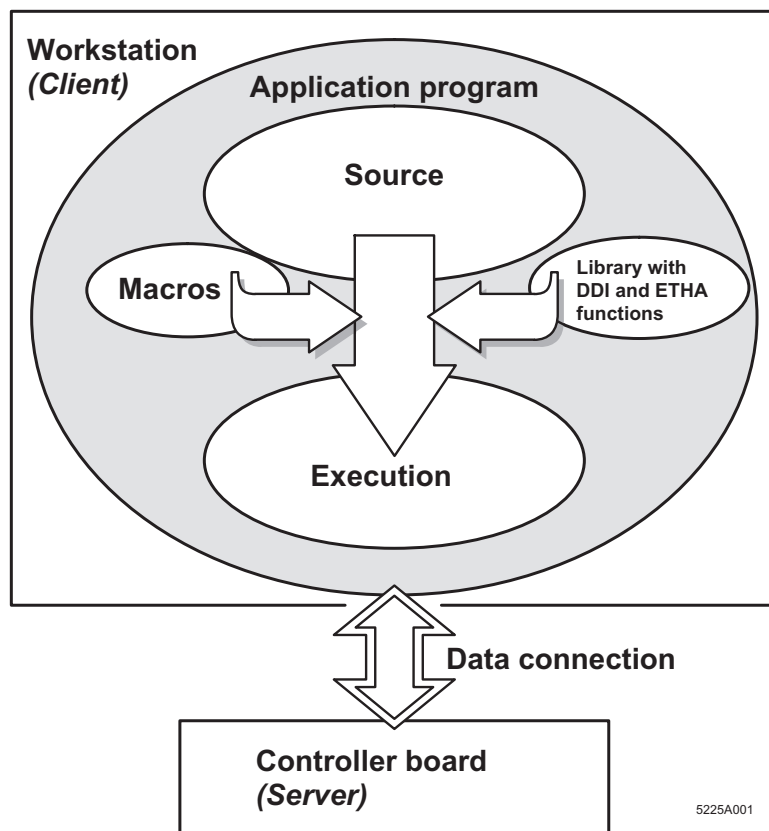


Figure 3-2 Using the driver software in the application program

3.3 Support and Driver Update

In the event of problems, please phone our 24-hour hotline on +49 - 52 35 - 34 18 88.

Driver updates and additional information are available on the Internet at www.phoenixcontact.com.

Training Courses

Our bus coupler training courses enable you to take advantage of the full capabilities of the connected Inline system. For details and dates, please see our seminar brochure, which your local Phoenix Contact representative will be happy to mail to you.

3.4 Transfer of I/O Data

The I/O data of individual Inline modules is transferred via memory areas organized in a word-oriented way (separate memory areas for input and output data). The Inline modules use the memory according to their process data width. User data is stored in word arrays in the order of the connected modules. The assignment of the individual bits is shown in the following diagram:

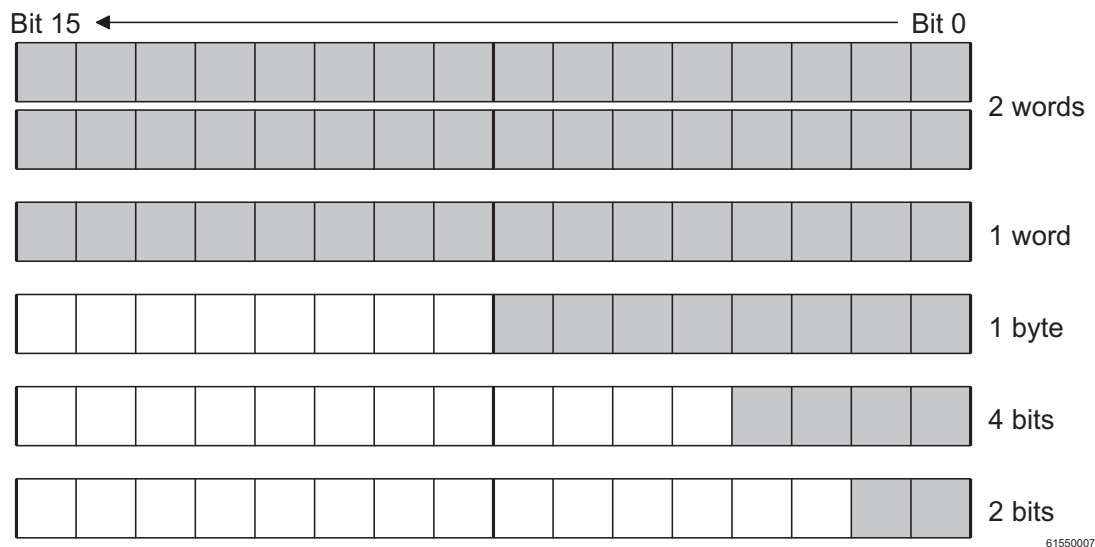


Figure 3-3 Position of the user data for individual devices in the word array

To achieve cycle consistency between input/output data and the station bus cycle, the bus coupler uses an exchange buffer mechanism. This mechanism ensures that the required I/O data is available at the correct time and is protected during writing/reading by appropriate measures.

The following diagram shows the position of the user data for several devices in the word array.

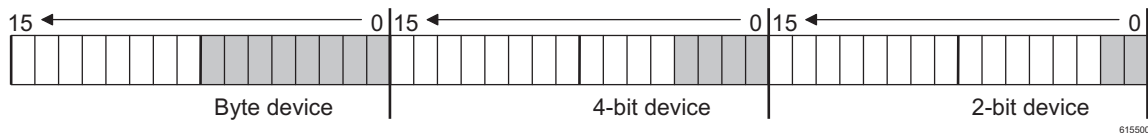


Figure 3-4 Position of the user data for several devices in the word array

3-7



Figure 3-5 Position of the process data according to the physical bus configuration

3.5 Startup Behavior of the Bus Coupler

Startup behavior of the bus coupler is determined via two system parameters: plug & play mode and expert mode. By default upon delivery P&P mode is activated and expert mode deactivated.

3.5.1 Plug & Play Mode



Observe that the following description applies with inactive expert mode. Possible combinations of both modes and their behavior are described on page 3-9.

P&P mode active

The FL IL 24 BK(-PAC) supports plug & play mode (P&P). This mode enables Inline modules connected in the field to be started up using the FL IL 24 BK-PAC bus coupler without a higher-level computer. The P&P status (active or inactive) is stored retentively on the bus coupler. In P&P mode the connected Inline terminals are recognized and checked for their function. If this physical configuration is ready to operate it is stored retentively on the bus coupler.

If the connected configuration could be stored as the reference configuration "PP" is displayed on the bus coupler.

In order not to overwrite the reference configuration again on the next bus coupler start, P&P mode must be deactivated again. Deactivation of P&P mode also acknowledges the reference configuration and enables process data exchange.

P&P mode inactive

With inactive P&P mode the reference configuration is compared with the physical configuration. If both are the same the bus coupler can be set to the "RUN" state.

If the reference configuration does not match the physical configuration "83" is displayed and process data cannot be exchanged for safety reasons.

However, to operate the bus there are two possibilities:

1. Restore the original configuration again in order for the reference configuration to match the physical configuration.
2. Activate P&P mode in order for the known physical configuration to be stored as the reference configuration.

3.5.2 Expert Mode



Observe that the following description applies with inactive P&P mode. Possible combinations of both modes and their behavior are described on page 3-9.

Expert mode inactive

If expert mode is inactive (default upon delivery) an error-free configuration is automatically set to the "RUN" state. If the configuration has a technical fault or if it does not match the reference configuration "83" is displayed and process data cannot be exchanged.

Expert mode active

If expert mode is active an error-free configuration is set to the "READY" state but not automatically to the "RUN" state. The station must be set to the "RUN" state using appropriate firmware commands, such as ACTIVATE_CONFIGURATION, 0x0711, or START_DATA_TRANSFER, 0x0701.

3.5.3 Possible Combinations of the Modes

Table 3-1 Possible combination of the modes and their effect

P&P Mode	Expert Mode	Description/Effect	Diagram
Inactive	Inactive	Normal status - If the connected configuration matches the configuration in the memory the station sets the valid configuration to the "RUN" state. Process data exchange is possible.	Figure 3-6 on page 3-10
Inactive	Active	A valid configuration is set to the "READY" state. Process data can only be exchanged if the station was set to the "RUN" state by means of firmware commands.	Figure 3-7 on page 3-10
Active	Inactive	The connected configuration is stored as the reference configuration. The station is set to the "RUN" state. Process data exchange is not possible.	Figure 3-8 on page 3-11
Active	Active	A physical configuration is stored as the reference configuration and set to the "READY" state. Process data can only be exchanged if P&P mode is inactive and the station was set to the "RUN" state by means of firmware commands.	Figure 3-9 on page 3-11

3.5.4 Startup Diagrams of the Bus Coupler

"Normal" mode / P&P mode and expert mode inactive

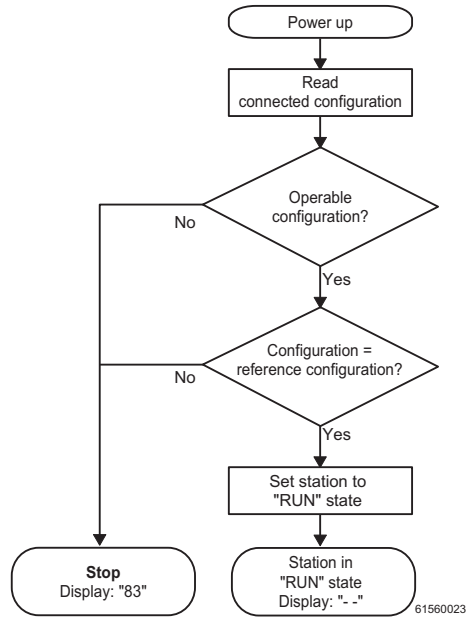


Figure 3-6 "Normal" mode / expert mode and P&P mode inactive

P&P mode inactive - expert mode active

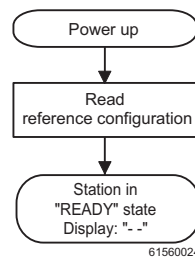


Figure 3-7 P&P mode inactive - expert mode active

P&P mode active - expert mode inactive

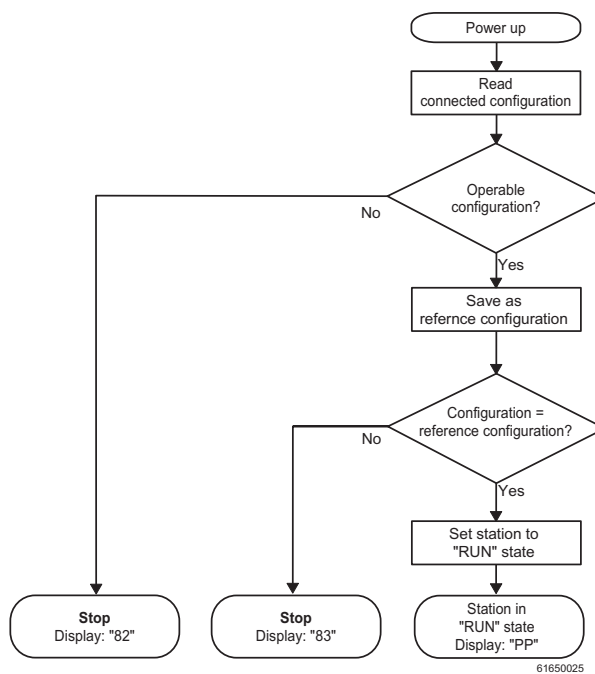


Figure 3-8 P&P mode active - expert mode inactive

P&P mode active and expert mode active

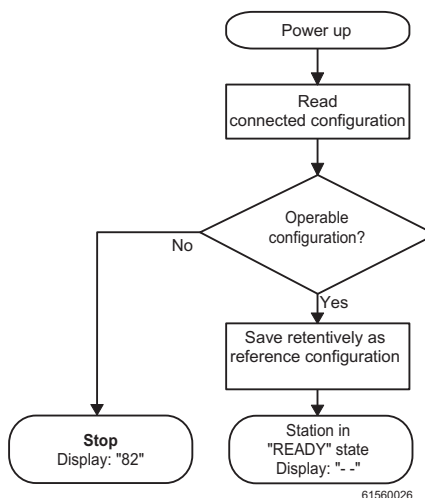


Figure 3-9 P&P mode active and expert mode active

3.5.5 Changing and Starting a Configuration in P&P Mode

The following steps must be carried out when **changing** an existing configuration:



Ensure that plug & play mode is active and expert mode is inactive.

- Switch the power supply off.
- Change the configuration.
- Switch the power supply on.

A configuration is **started** as shown in the flowchart (see Figure 3-6 to Figure 3-9). During startup, please observe the following:

- Once the coupler has been switched on, the previously found configuration is read and started, as long as no errors are present. In addition, the known configuration is saved in the EEPROM as the reference configuration.
- All connected Inline devices are integrated in the known configuration if the "DIAG" LEDs are continuously lit on all modules.
- To prevent the accidental use of the wrong configuration, process data can only be accessed when P&P mode has been deactivated.



When P&P mode is active, access to process data is rejected with the error message 00A9_{hex} (ERR_PLUG_PLAY). The outputs of the entire Inline station are reset in P&P mode.

P&P mode is activated using the I/O browser or the "Set_Value" command via Ethernet. Once P&P mode has been switched off, the bus is only started if the existing configuration and the reference configuration are the same. In addition, the existing configuration will no longer be saved automatically as the reference configuration after a bus coupler restart.

3.6 Changing a Reference Configuration Using the Software

3.6.1 Effects of Expert Mode



Only switch to expert mode if you want to deactivate automatic configuration and activate manual configuration using the firmware commands.

If expert mode (object 2275_{hex}) is active, automatic startup of the connected local bus is prevented.

The user must manually place the bus to the RUN state by activating the configuration (Activate_Configuration/0711_{hex} object or Create_Configuration/0710_{hex} object) and by starting the local bus (Start_Data_Transfer/0701_{hex} object).

In expert mode, the bus coupler behaves in the same way as the gateways (IBS SC/I-T or IBS 24 ETH DSC/I-T).

3.6.2 Changing a Reference Configuration

- Deactivate P&P mode.
- Activate expert mode (for access to all firmware commands).
- Place the bus to the "Active" or "Stop" state (e.g., using the "Alarm_Stop" command).
- The reference configuration can be downloaded or deleted.
- The connected bus can be read using the "Create_Configuration" command and saved as the reference configuration, as long as the bus can be operated.
- The bus is started using the "Start_Data_Transfer" command. If access to process data is rejected with an error message, this means that no reference configuration is present.

Table 3-2 System parameters for the "Set_Value" service (750_{hex})

Variable ID	System Parameter	Value/Comment
0104 _{hex}	Diagnostic status register	Read only
0105 _{hex}	Diagnostic parameter register	Read only
2216 _{hex}	Current PD cycle time	Read only

Table 3-2 System parameters for the "Set_Value" service (750_{hex})

Variable ID	System Parameter	Value/Comment
2240 _{hex}	Plug & play mode	0: Plug & play mode inactive 1: Plug & play mode active (takes effect after reboot)
2275 _{hex}	Expert mode	0: Expert mode inactive (default) 1: Expert mode active
2277 _{hex}	Fault response mode	1: Fault reset mode (default) 2: Hold last state 0: Standard fault mode
2293 _{hex}	Process data monitoring timeout	0: Process data watchdog inactive 200 - 65000: Timeout value

3.7 Description of the Device Driver Interface (DDI) Introduction

The Device Driver Interface (DDI) is provided for using the bus coupler services. The functions of the DDI are combined in a library, which must be linked.

3.7.1 Overview

Table 3-3 Overview of the functions in the DDI

Functions	Page
DDI_DevOpenNode	3-18
DDI_DevCloseNode	3-20
DDI_DTI_ReadData	3-22
DDI_DTI_WriteData	3-24
DDI_DTI_ReadWriteData	3-26
DDI_MXI_SndMessage	3-28
DDI_MXI_RcvMessage	3-30
GetIBSDiagnostic	3-32
ETH_SetHostChecking	3-38
ETH_ClearHostChecking	3-40
ETH_SetDTITimeoutCtrl	3-42
ETH_ClearDTITimeoutCtrl	3-43
ETH_SetNetFail	3-47
ETH_GetNetFailStatus	3-47
ETH_ClrNetFailStatus	3-49
DDI_SetMsgNotification	3-52
DDI_ClrMsgNotification	3-52
ETH_ActivatePDInMonitoring	3-53
ETH_DeactivatePDInMonitoring	3-55
ETH_SetNetFailMode	3-50
ETH_GetFailMode	3-51

3.7.2 Working Method of the Device Driver Interface

- Remote procedure call** The entire Device Driver Interface (DDI) for the bus coupler operates as remote procedure calls. It does not use the standard libraries due to time constraints. A remote procedure call means that the relevant function is not executed on the local computer or the local user workstation (client), but on another computer in the network. In this case, this is the bus coupler for Ethernet. The user does not notice anything different about this working method except that it is faster. The sequence of a remote procedure call is shown in Figure 3-10.
- Editing data telegrams** When a function is called, the transfer parameters for the DDI function and an ID for the function to be executed are copied into a data telegram (network telegram) on the client and sent to the host (bus coupler) via the Ethernet network (TCP/IP). The host decodes the received data telegram, accepts the parameters for the function, and calls the function using these parameters. The *DDI_DTI_ReadData(nodeHd, dtiAcc)* function is called as an example in Figure 3-10.
- During function execution by the server (bus coupler), the thread (process) is in *sleep* state on the client until a reply is received from the server.
- Once the function has been executed on the server, the read data and the return value for the function are copied into a data telegram on the host and sent back to the client (user workstation). The workstation decodes this data telegram and makes the return value of the function available to the user.
- This working method is the same for each DDI function, which is executed on the server as a remote procedure call.

Remote Procedure Call Process

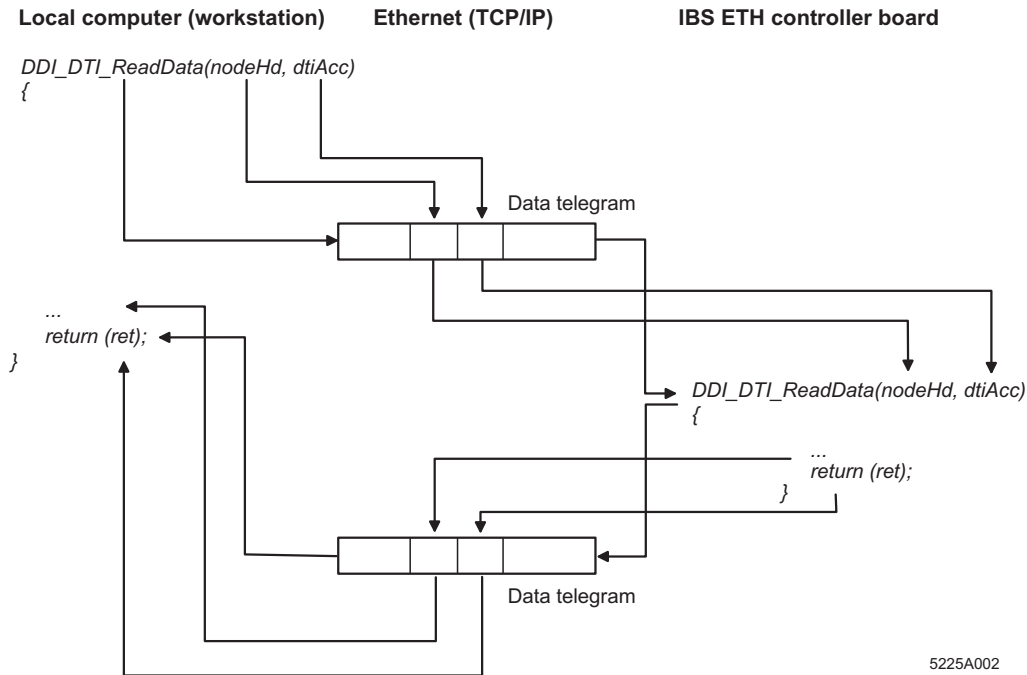


Figure 3-10 Execution of a remote procedure call

3.7.3 Description and Functions of the Device Driver Interface

DDI_DevOpenNode

UNIX

Task:

In order for the Device Driver Interface (DDI) to be able to find and address the desired bus coupler in the Ethernet network using the device name, a file called *ibsetha* must be created. This file contains the assignment between the device name and the IP address or the server name of the bus coupler.



A different name cannot be used for the file.

The structure of the file and its entries is as follows:

```
192.168.5.76      IBETH01N1_M IBETH01N1_D
etha2             IBETH02N1_M IBETH02N1_D
```

Several device names can be assigned to a single IP address or server name. The individual device names are separated by spaces. The address of the bus coupler can be entered in dotted notation: 192.168.5.76 or as server name: etha2. If a device name is used several times, only the first occurrence in the file is evaluated.

Windows NT/2000

The following entries should be created in the registry so that the Device Driver Interface (DDI) can find the selected bus coupler. Entry creation is done by the driver. You can find the driver in the download area on the Internet at www.phoenixcontact.com or the "CD FL IL 24 BK" CD, Order No. 28 32 06 9.

The following registry entry is created:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Phoenix
Contact\IBSETH\Parameters\1]
ConnectTimeout=08,00,00,00
DeviceNames=IBETH01N1_M IBETH01N0_M@01 IBETH01N1_D
IBETH01N0_D IBETH01N1_M@00 IBETH01N1_M@05
InUse=YES
ReceiveTimeout=08,00,00,00
IPAddress=192.168.36.205
```

Function:

The *DDI_DevOpenNode* function opens a data channel to the bus coupler specified by the device name or to a node.

The function receives the device name, the desired access rights, and a pointer to a variable for the node handle as arguments. If the function was executed successfully, a handle is entered in the variable referenced by the pointer, and this handle is used for all subsequent access to this data channel. In the event of an error, a valid value is not entered in the variable.

An appropriate error code is instead returned by the *DDI_DevOpenNode* function, which can be used to determine the cause of the error.

The node handle, which is returned to the application program, is automatically generated by the DDI or bus coupler. This node handle has direct reference to an internal control structure, which contains all the corresponding data for addressing the relevant bus coupler.

The local node handle is used to obtain all the necessary parameters for addressing the bus coupler, such as the IP address, socket handle, node handle on the bus coupler, etc. from this control structure when it is subsequently accessed.

A control structure is occupied when the data channel is opened and is not released until the *DDI_DevCloseNode* function has been executed or the connection has been aborted. The maximum number of control structures is determined when the library is compiled and cannot subsequently be modified. In Windows NT/2000 there are eight control structures per device, with a maximum of 256.

If all the control structures are occupied, another data channel cannot be opened. In this case, if *DDI_DevOpenNode* is called, it is rejected locally with the appropriate error message.

Syntax:	IBDDIRET IBDDIFUNC DDI_DevOpenNode (CHAR *devName, INT16 perm, IBDDIHND *nodeHd);	
Parameters:	CHAR *devName	Pointer to a string with the device name.
	INT16 perm	Access rights to the data channel to be opened. This includes read, write, and read/write access.
	IBDDIHND *nodeHd	Pointer to a variable for the node handle (MXI or DTI).
Return value:	IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.
Constants for the <i>perm</i> parameter	DDI_READ	0x0001 /* Read only access */
	DDI_WRITE	0x0002 /* Write only access */
	DDI_RW	0x0003 /* Read and write access */

Example**Windows NT/2000 / UNIX:**

```
IBDDIHND ddiHnd;
{
    IBDDIRET ddiRet;

    ddiRet=DDI_DevOpenNode ("IBETH01N1_D", DDI_RW,
        &ddiHnd);

    if (ddiRet !=ERR_OK)
    {
        /* Error treatment */
        .
        .
        return:
    }
    .
}
```

DDI_DevCloseNode**Task:**

If a data channel is no longer needed, it can be closed using the *DDI_DevCloseNode* function. This function uses only the node handle as a parameter, which determines the data channel that is to be closed. If the data channel cannot be closed or the node handle is invalid, an appropriate error code is returned by the function.



All active connections should be closed before calling the *DDI_DevCloseNode* function.

Syntax:

IBDDIRET IBDDIFUNC DDI_DevCloseNode(IBDDIHND nodeHd);

Parameter:

IBDDIHND nodeHd

Node handle (MXI or DTI) for the connection that is to be closed.

Return value:

IBDDIRET

If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Example**UNIX / Windows NT/2000**

```
IBDDIHND ddiHnd;

{
    IBDDIRET ddiRet;

    .
    .
    .
    ddiRet=DDI_DevCloseNode (ddiHnd);

    if (ddiRet !=ERR_OK)
    {
        /* Error treatment */
        .
        .
        .
    }
    return;
}
```

DDI_DTI_ReadData**Task**

The *DDI_DTI_ReadData* function is used to read process data from the Inline bus coupler. The function is assigned the node handle and a pointer to a *T_DDI_DTI_ACCESS* data structure.

The *T_DDI_DTI_ACCESS* structure contains all the parameters that are needed to access the process data area of the bus coupler and corresponds to the general DDI specification. A plausibility check is not carried out on the user side, which means that the parameters are transmitted via the network just as they were transferred to the function.

The *nodeHd* parameter specifies the bus coupler in the network to which the request is to be sent. The node handle must also be assigned to a process data channel, otherwise an appropriate error message is generated by the bus coupler.

Syntax:

```
IBDDIRET IBDDIFUNC DDI_DTI_ReadData(IBDDIHND nodeHd,  
                                     T_DDI_DTI_ACCESS *dtiAcc);
```

Parameters:

IBDDIHND nodeHd Node handle (DTI) for the connection from which data is to be read. The node handle also determines the bus coupler, which is to be accessed.

T_DDI_DTI_ACCESS *dtiAcc
Pointer to a T_DDI_DTI_ACCESS data structure. This structure contains all the parameters needed for access.

Return value:

IBDDIRET If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

**Format of the
T_DDI_DTI_
ACCESS structure:**

```
typedef struct {  
    USIGN16 length;  
    /* Amount of data to be read in bytes */  
    USIGN16 address;  
    /* Address in the DTI area (byte address) */  
    USIGN16 dataCons;  
    /* Desired data consistency area */  
    USIGN8 *data;  
    /* Pointer to the data (read and  
    write) */  
} T_DDI_DTI_ACCESS;
```

Example**UNIX / Windows NT/2000**

```
IBDDIHND ddiHnd;

{
    IBDDIRET ddiRet;
    T_DDI_DTI_ACCESS dtiAcc;
    USIGN8 iBuf[512];

    dtiAcc.length = 512;
    dtiAcc.address = 0;
    dtiAcc.data = iBuf;
    dtiAcc.dataCons = DTI_DATA_BYTE;

    ddiRet = DDI_DTI_ReadData (ddiHnd, &dtiAcc);

    if (ddiRet != ERR_OK)
    {
        /* Error treatment */
        .
        .
        .
    }
    .
    .
    .
}
```

DDI_DTI_WriteData**Task:**

The *DDI_DTI_WriteData* function is used to write process data to the bus coupler.



By default upon delivery, the watchdog is activated with 500 ms timeout. The first write process activates the process data watchdog. The next write process is expected during timeout (default: 500 ms).

The function is assigned the node handle and a pointer to a *T_DDI_DTI_ACCESS* data structure.

The *T_DDI_DTI_ACCESS* structure contains all the parameters that are needed to access the process data area of the bus coupler and corresponds to the general DDI specification. A plausibility check is not carried out on the user side, which means that the parameters are transmitted via the network just as they were transferred to the function.

The *nodeHd* parameter specifies the bus coupler in the network to which the request is to be sent. The node handle must also be assigned to a process data channel, otherwise an appropriate error message is generated by the bus coupler.

Syntax:

```
IBDDIRET IBDDIFUNC DDI_DTI_WriteData(IBDDIHND nodeHd,
                                       T_DDI_DTI_ACCESS *dtiAcc);
```

Parameter:

IBDDIHND nodeHd Node handle (DTI) for the connection to which data is to be written. The node handle also determines the bus coupler, which is to be accessed.

T_DDI_DTI_ACCESS *dtiAcc

Pointer to a *T_DDI_DTI_ACCESS* data structure. This structure contains all the parameters needed for access.

Return value:

IBDDIRET

If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.



Cycle-consistent data is written for all data consistency areas higher than byte.

**Format of the
T_DDI_DTI_
ACCESS structure**

```
typedef struct {
    USIGN16 length;
        /* Amount of data to be written in bytes */
    USIGN16 address;
        /* Address in the DTI area (byte address) */
    USIGN16 dataCons;
        /* Desired data consistency area */
};
```

```
        USIGN8 *data;
        /* Pointer to the data (read and
           write) */
    } T_DDI_DTI_ACCESS;
```

Example**UNIX / Windows NT/2000**

```
IBDDIHND ddiHnd;

{
    IBDDIRET ddiRet;
    T_DDI_DTI_ACCESS dtiAcc;
    USIGN8 oBuf[512];

    dtiAcc.length = 512;
    dtiAcc.address = 0;
    dtiAcc.data = oBuf;
    dtiAcc.dataCons = DTI_DATA_BYTE;

    oBuf[0] = 0x12;
    oBuf[1] = 0x34;

    ddiRet = DDI_DTI_WriteData (ddiHnd, &dtiAcc);

    if (ddiRet != ERR_OK)
    {
        /* Error treatment */
    }
}
}
```

DDI_DTI_ReadWriteData

Task: The *DDI_DTI_ReadWriteData* function is used to read and write process data in one call. This function increases performance considerably, especially when using process data services via the network, because process data is read and written in a single sequence.



By default upon delivery, the watchdog is activated with 500 ms timeout. The first write process activates the process data watchdog. The next write process is expected during timeout (default: 500 ms).

The function is assigned the node handle and two pointers to *T_DDI_DTI_ACCESS* data structures. One structure contains the parameters for read access and the other structure contains the parameters for write access. The *T_DDI_DTI_ACCESS* structure corresponds to the general DDI specification. A plausibility check is not carried out on the user side, which means that the parameters are transmitted via the network just as they were transferred to the function.

The *nodeHd* parameter specifies the bus coupler in the network to which the request is to be sent. The node handle must be assigned to a process data channel, otherwise an appropriate error message is generated by the bus coupler.

Syntax: IBDDIRET IBDDIFUNC DDI_DTI_ReadWriteData (IBDDIHND nodeHd,
T_DDI_DTI_ACCESS *writeDTIAcc,
T_DDI_DTI_ACCESS *readDTIAcc);

Parameters:

IBDDIHND nodeHd	Node handle (DTI) for the connection to which data is to be written. The node handle also determines the bus coupler, which is to be accessed.
T_DDI_DTI_ACCESS *writeDTIAcc	Pointer to a T_DDI_DTI_ACCESS data structure with the parameters for write access.
T_DDI_DTI_ACCESS *readDTIAcc	Pointer to a T_DDI_DTI_ACCESS data structure with the parameters for read access.

Return value: IBDDIRET If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Format of the T_DDI_DTI_ ACCESS structure

```
typedef struct {
    USIGN16 length;
        /* Amount of data to be read in bytes */
    USIGN16 address;
        /* Address in the DTI area (byte address) */
    USIGN16 dataCons;
        /* Desired data consistency area */
    USIGN8 *data;
        /* Pointer to the data (read and
        write) */
} T_DDI_DTI_ACCESS;
```

Example

UNIX / Windows NT/2000

```
IBDDIHND ddiHnd;

{
    IBDDIRET ddiRet;
    T_DDI_DTI_ACCESS dtiReadAcc;
    T_DDI_DTI_ACCESS dtiWriteAcc;
    USIGN8 oBuf[512];
    USIGN8 iBuf[512];

    dtiWriteAcc.length = 512;
    dtiWriteAcc.address = 0;
    dtiWriteAcc.data = oBuf;
    dtiWriteAcc.dataCons = DTI_DATA_BYTE;

    dtiReadAcc.length = 512;
    dtiReadAcc.address = 0;
    dtiReadAcc.data = iBuf;
    dtiReadAcc.dataCons = DTI_DATA_BYTE;

    oBuf[0] = 0x12
    oBuf[1] = 0x34

    ddiRet=DDI_DTI_ReadWriteData (ddiHnd,
    &dtiWriteAcc, &dtiReadAcc);

    if (ddiRet!=ERR_OK)
    {
        /* Error treatment */
        .
        .
    }
    .
    .
}
```

DDI_MXI_SndMessage**Task:**

The *DDI_MXI_SndMessage* function is used to send a message to the bus coupler. The function receives a node handle and a pointer to a *T_DDI_MXI_ACCESS* data structure as parameters. The *T_DDI_MXI_ACCESS* structure contains all the parameters that are needed to send the message.

These parameters are transmitted to the bus coupler via the network without a plausibility check, which means that invalid parameters are first detected on the bus coupler and acknowledged with an error message. The *IBDDIHND nodeHd* parameter specifies the bus coupler in the network to which the request is to be sent.

The node handle must be assigned to a mailbox interface data channel, otherwise an appropriate error message is generated by the bus coupler.

Syntax:

```
IBDDIRET IBDDIFUNC DDI_MXI_SndMessage (IBDDIHND nodeHd,  
                                         T_DDI_MXI_ACCESS *mxiAcc);
```

Parameters:

IBDDIHND nodeHd Node handle (MXI) for the connection via which a message is to be written to the mailbox interface. The node handle also determines the bus coupler, which is to be accessed.

T_DDI_MXI_ACCESS *dtiAcc Pointer to a *T_DDI_MXI_ACCESS* data structure. This structure contains all the parameters needed for access.

Return value:

IBDDIRET If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

**Format of the
T_DDI_MXI_
ACCESS structure**

```
typedef struct {  
    USIGN16 msgType;  
    /* Message type (see DDI description) */  
    USIGN16 msgLength;  
    /* Length of the message in bytes */  
    USIGN16 DDIUserID; /* Message ID */  
    USIGN8 *msgBlk;  
    /* Pointer to the message data */  
} T_DDI_MXI_ACCESS;
```

Example**UNIX / Windows NT/2000**

```
IBDDIHND mxiHnd;

.
.
{
    IBDDIRET ddiRet;
    T_DDI_MXI_ACCESS mxiAcc;
    USIGN8 oBuf[256];

    mxiAcc.msgLength = 4;
    mxiAcc.DDIUserID = 0;
    mxiAcc.msgType = 0;
    mxiAcc.msgBlk = oBuf;

    IB_SetCmdCode (oBuf, S_CREATE_CFG_REQ);
    IB_SetParaCnt (oBuf, 1);
    IB_SetParaN (oBuf, 1, 1);

    ddiRet = DDI_MXI_SndMessage (mxiHnd, &mxiAcc);

    if (ddiRet!=ERR_OK)
    {
        /* Error treatment */
        .
        .
        .
    }
    .
    .
    .
}
```

DDI_MXI_RcvMessage

The *DDI_MXI_RcvMessage* function reads a message from the bus coupler. The function receives a node handle and a pointer to a *T_DDI_MXI_ACCESS* data structure as parameters. The *T_DDI_MXI_ACCESS* structure contains all the parameters that are needed to read the message.

These parameters are transmitted to the bus couplers via the network without a plausibility check, which means that invalid parameters are first detected at the bus coupler and acknowledged with an error message. The *nodeHd* parameter specifies the bus coupler in the network to which the request is to be sent. The node handle must be assigned to a mailbox interface data channel, otherwise an appropriate error message is generated by the bus coupler.

The function does not wait until a message is received in the coupling memory, instead it returns immediately. If no message is present, the error code *ERR_NO_MSG* is returned.



To prevent excessive mailbox interface requests, special modes can be activated for reading the message, which enable the system to wait for a message from the bus coupler.

Syntax:

```
IBDDIRET IBDDIFUNC DDI_MXI_RcvMessage(IBDDIHND nodeHd,
                                       T_DDI_MXI_ACCESS *mxiAcc);
```

Parameters:

IBDDIHND nodeHd Node handle (MXI) for the connection via which a message is to be read from the mailbox interface. The node handle also determines the bus coupler, which is to be accessed.

T_DDI_MXI_ACCESS *dtiAcc Pointer to a *T_DDI_MXI_ACCESS* data structure. This structure contains all the parameters needed for access.

Return value:

IBDDIRET If the function is executed successfully, the value 0 (*ERR_OK*) is returned. Otherwise the return value is an error code.

**Format of the
T_DDI_MXI_
ACCESS structure**

```
typedef struct {
    USIGN16 msgType;
        /* Message type */
    USIGN16 msgLength;
        /* Length of the message in bytes */
    USIGN16 DDIUserID;
        /* Message ID */
    USIGN8 *msgBlk;
        /* Pointer to the message data */
} T_DDI_MXI_ACCESS;
```

Example**UNIX / Windows NT/2000**

```
IBDDIHND mxiHnd;
.
.
{
    IBDDIRET ddiRet;
    T_DDI_MXI_ACCESS mxiAcc;
    USIGN8 iBuf[256];
    USIGN16 msgCode;
    USIGN16 paraCounter;
    USIGN16 parameter[128];
    unisnet int i;

    mxiAcc.msgLength = 256;
    mxiAcc.DDIUserID = 0;
    mxiAcc.msgType = 0;
    mxiAcc.msgBlk = iBuf;

    ddiRet = DDI_MXI_RcvMessage (mxiHnd, &mxiAcc);

    if (ddiRet != ERR_OK)
    {
        /* Evaluation of the message */

        msgCode = IB_GetMsgCode (iBuf);
        paraCounter = IB_GetParaCnt (iBuf);

        for (i=0; i<paraCounter; i++)
        {
            parameter[i] = IB_GetParaN (iBuf, i);
        }
    }
}
```

GetIBSDiagnostic

Task: The *DDI_GetIBSDiagnostic* function reads the diagnostic bit register and the diagnostic parameter register. The function receives a valid node handle and a pointer to a *T_IBS_DIAG* data structure as parameters. After the function has been called successfully, the structure components contain the contents of the diagnostic bit register and the diagnostic parameter register in processed form.

Syntax: IBDDIRET IBDDIFUNC DDI_GetIBSDiagnostic(IBDDIHND nodeHd, T_IBS_DIAG *infoPtr);

Parameters:

IBDDIHND nodeHd	Node handle (MXI or DTI) of the bus coupler from which the diagnostic bit register and diagnostic parameter register are to be read.
T_IBS_DIAG *infoPtr	Pointer to a T_IBS_DIAG data structure. The contents of the register are entered in this structure.

Format of the T_IBS_DIAG structure

```
typedef struct {
    USIGN16 state; /* Status of the local bus*/
    USIGN16 diagPara;
                /* Type of error (controller,
                user, etc.) */
} T_IBS_DIAG;
```

Return value: IBDDIRET

If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Example UNIX / Windows NT/2000

```
IBDDIHND ddiHnd;
{
    T_IBS_DIAG infoPtr;
    IBDDIRET ddiRet;
    USIGN16 stateAB;
    USIGN16 diagAB;
    .
    {
        Sleep (20) /* Dependent on operating system */;
        ddiRet = GetIBSDiagnostic (ddiHnd, &infoPtr);
        stateAB = infoPtr.state;
        diagAB = infoPtr.diagPara;
    } while (...).
    .
}
```

3.8 Monitoring Functions

Monitoring functions with different features are available for monitoring Ethernet communication and the connected devices.

- Process data watchdog (process data monitoring)
- Host checking
- DTI monitoring

There are monitoring functions according to the features/functions that need to be monitored. According to the application requirements the appropriate monitoring function can be activated. By default upon delivery, process data watchdog is activated.

Table 3-4 Monitoring functions

Monitoring Mechanism	Monitoring ...			
	... the client application	... the individual channels	... the Ethernet connection	... process data exchange
Process data watchdog (process data monitoring)	X	- - -	X	X
Host checking	- - -	- - -	X	- - -
DTI/Modbus monitoring	X	X	X	- - -

In the event of an error the system reacts with a fault response. The user determines the required fault response mode.

Setting the Required Fault Response Mode

The required fault response mode can be set to the object ID 0x2277 using the web-based management or by writing to the Modbus register 2002 or using the "Set_Value" (0x0750) service. The following fault response modes are available:

Table 3-5 Available fault response modes

Fault Response Mode	Value	Function
Reset fault mode (default)	1	The digital outputs are set to "0". The analog outputs are set to a value (default = "0") programmed by the user.
Standard fault mode	0	All outputs are set to "0".
Hold last state mode	2	All outputs hold the last value.

Reasons for Fault Response

The web-based management, the Modbus register 2004 or the "ETH_GetNetFailState" service allow to request the causes for fault response mode and for setting the NetFail signal.

Causes

The following reasons are possible:

DDI_NF_TASK_CREAT_ERR	0x0001
/* Error when starting a task */	
DDI_NF_LISTENER_ERR	0x0002
/* Listener task error */	
DDI_NF_RECEIVER_ERR	0x0003
/* Receiver task error */	
DDI_NF_ACCEPT_ERR	0x0004
/* Accept function error */	
DDI_NF_ECHO_SERVER_ERR	0x0005
/* Echo server task error */	
DDI_NF_HOST_CONTROLLER_ERR	0x0006
/* Host controller task error */	
DDI_NF_DTI_TIMEOUT	0x0007
/* DTI timeout occurred */	
DDI_NF_HOST_TIMEOUT	0x0008
/* Host timeout occurred */	
DDI_NF_USER_TEST	0x0009
/* NetFail set by user */	
DDI_NF_CONN_ABORT	0x000A
/* Connection aborted */	
DDI_NF_INIT_ERR	0x000B
/* Initialization error */	
DDI_NF_DTI_WATCHDOG	0x000C
/* Process data watchdog triggered */	
DDI_NF_MBUS_TIMEOUT	0x000D
/* Modbus timeout occurred */	

Acknowledgement of the NetFail Signal

The NetFail signal can be acknowledged using the web-based management, by setting bit 1 in the command word of the Modbus register 4076, or using the "ETH_ClrNetFailState" service.

3.8.1 Process Data Monitoring/ Process Data Watchdog

3.8.1.1 Process Data Watchdog Function



By default upon delivery, process data watchdog is activated with 500 ms timeout.

A process data watchdog is integrated into the bus coupler to avoid uncontrolled setting/resetting of the Inline station outputs in the event of an error.

If outputs of the stations are set, ensure access of the controlling process to the station. In the event of an error, e.g., network line interrupted or function error in the controlling process, the bus coupler can react appropriately via the process data watchdog. By default upon delivery, the watchdog is activated with 500 ms timeout. The first write process activates the process data watchdog. The next write process is expected during timeout (default: 500 ms). During error-free operation, the write process is performed during timeout and the watchdog is restarted (triggered).



Read calls do not trigger the process data watchdog.

If there is no triggering during timeout, an error occurred. Two reactions follow:

- The selected fault response mode is executed
- The NetFail signal is set

The reason for setting the NetFail signal is listed in the reason code (see page 3-34).

For safety reasons, the user cannot stop the watchdog once it has been activated. In case the user terminates the controlling application, there is no watchdog triggering; when timeout has expired, the NetFail signal is set and the selected fault response mode is executed.

The NetFail signal is acknowledged using the web-based management or the "ETH_ClrNetFailState" command and the fault response mode is reset.



By acknowledging the error, the watchdog is restarted. This means that it must be triggered during timeout, otherwise an error is detected again.

3.8.1.2 Configuring the Process Data Watchdog and the Fault Response Modes



Timeout can only be changed if the watchdog is in "INIT" state. The "INIT" state occurs after a power up as long as no process data exchange has taken place or in the event of a timeout when fault response was activated and no acknowledgment of the NetFail has yet taken place.

Process data watchdog timeout can be configured from 200 ms to 65000 ms. Timeout can be set to the object ID 0x2293 using the web-based management, by writing to the Modbus register 2000, or using the "Set_Value" (0x0750) service.

Deactivating the Process Data Watchdog

The process data watchdog can only be deactivated if the bus coupler is in "INIT" state. For deactivation, the timeout time is set to "zero". The required fault response mode can also be set to the object ID 0x2277 using the web-based management, by writing to the Modbus register 2002, or using the "Set_Value" (0x0750) service.

Status Diagram of the Process Data Watchdog

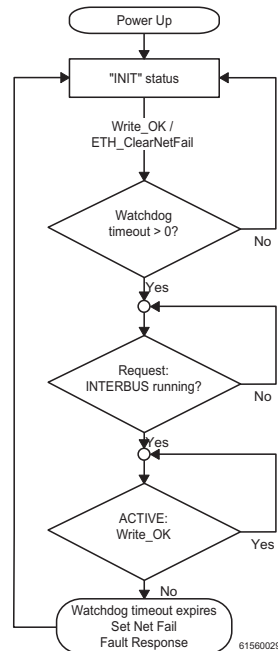


Figure 3-11 Status diagram of the process data watchdog

3.8.2 Connection Monitoring (Host Checking)

Application

Connection monitoring can be used to determine whether there is still a connection between the bus coupler (server) and the computer (client) and whether this computer responds to requests. With this monitoring function it is also possible to detect the following error causes:

- Cable broken, not connected or short circuited
- Transceiver faulty
- Errors or defects in the Ethernet adapter of the bus coupler or in the client
- System crash of the client (workstation)
- Error in the TCP/IP protocol stack

Activating Monitoring

The *ETH_SetHostChecking* function activates the mode for monitoring the connection and the status of the client. The function is assigned a valid node handle (DTI or MXI data channel) and a pointer (*time*) to a variable with the timeout time.

This mode can be activated for all clients (workstations) with a DDI connection. A connection to a client, which only uses Ethernet management cannot be monitored. If several connections to a client are activated simultaneously, the client is only addressed once during a cycle. If the connection no longer exists, monitoring is also reset.

Echo Port

Monitoring uses the echo port, which is provided on all systems that support TCP/IP. Each data telegram to this port is sent back from the receiver to the sender. The port is used for both connection-oriented TCP and connectionless UDP. In the case of the bus coupler, the echo port is used with UDP, to keep the resources used to a minimum.

Detecting an Error

Connection monitoring sends a short data telegram to a client every 500 ms. This interval is predefined and does not even change with to the number of clients that are addressed. This means that the frequency with which each client is "addressed" decreases with the number of connected clients. After the data telegram has been sent, the Inline bus coupler waits for a user-defined time for the reply to be received. If the reply is not received within this time, the bus coupler sends another data telegram to the relevant client. This process is repeated a maximum of three times. Connection monitoring then assumes that a serious error has occurred and sets the NetFail signal (outputs are set to zero).

Deactivating Monitoring

If connection monitoring is no longer required, it can be deactivated using the *ETH_ClearHostChecking* function. Monitoring is only deactivated for the client and the connection, which are specified by the node handle. If the same client has additional DDI connections to the bus coupler and connection monitoring was also activated for these connections, this client is still monitored via the other connections.

If a DDI connection is closed using *DDI_DevCloseNode*, monitoring for this client is also deactivated. Additional connections are treated as above; they are not reset and monitoring for these connections is not deactivated.

Echo Port on the Client (Computer)



An echo server runs on a PC using the Windows operating system if TCP/IP services are installed. You can find these services under ...\\Control Panel\\Network\\Services. Ensure that the echo server responds within 500 ms in every operating state. The echo server implemented as standard on Windows 2000 does not meet these requirements. Therefore the user should use DTI monitoring for monitoring the connection.

ETH_SetHostChecking

Task: After the *ETH_SetHostChecking* function has been called successfully, the client (user workstation) is addressed by the bus coupler at regular intervals.

If the client does not respond within the predefined time (timeout), three additional attempts are made to address the client. If there is still no response, the NetFail signal is set and the TCP connection is aborted by the bus coupler.

Syntax: IBDDIRET IBDDIFUNC ETH_SetHostChecking (IBDDIHND nodeHd, USIGN16 *time);

Parameters:	IBDDIHND nodeHd	Node handle (MXI or DTI) for the bus coupler that is to be monitored.
	USIGN16 *time	Pointer to a variable, which contains the desired timeout time when called. If the function has been called successfully, the actual timeout time is then entered in this variable. The shortest value for the timeout time is 330 ms, the longest value is 65000 ms. If a shorter value is entered, the error code ERR_INVLD_PARAM is returned and "Host Checking" is not activated.

Return value:	IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.
----------------------	----------	--

Example**Unix / Windows NT/2000**

```
IBDDIHND ddiHnd;
{
void CAU00yxDlg::OnButtonSetHostCheckingOn()
{
    IBDDIRET ddiRet;
    USIGN16 hcTime = 1000;
    .
    .
    .
    {
        ddiRet = ETH_SetHostChecking
        (ddiHnd, &hcTime);
        if (ddiRet == ERR_INVLD_PARAM)
        {
            /* hcTime selected too short!
            (min. 330ms) */
            .
            .
            .
        }
    }
    UpdateData (FALSE)
}
```

ETH_ClearHostChecking

Task:	The <i>ETH_ClearHostChecking</i> function deactivates the node used to monitor the client. This function only receives the node handle as a parameter, which is also used to activate monitoring with <i>ETH_SetHostChecking</i> . After the function has been called successfully, monitoring via this channel and for this client is deactivated. Other activated monitoring channels are not affected.	
Syntax:	IBDDIRET IBDDIFUNC ETH_ClearHostChecking (IBDDIHND nodeHd);	
Parameter:	IBDDIHND nodeHd	Node handle (MXI or DTI) of the bus coupler for which monitoring is to be deactivated. The same node handle that was used for activating monitoring must also be used here.
Return value:	IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

3.8.3 Data Interface (DTI) Monitoring

Error Detection and Response

Client monitoring using connection monitoring can only determine whether a client can still be addressed. It is not possible to determine whether the process that controls the bus coupler (application program) is still operating correctly. An extremely serious error occurs when the controlling process is no longer operating correctly, i.e., the bus coupler is no longer supplied with up-to-date process data and as a result incorrect output data is sent to the local bus devices.

DTI monitoring can detect if a message to the data interface of the bus coupler has failed to arrive and the appropriate safety measures can be implemented. In this case, the failure of the DTI data telegram sets the NetFail signal and resets the output data for the local bus devices to zero.

Activating Monitoring

Monitoring of the data interface (DTI) is not activated immediately after the *ETH_SetDTITimeoutCtrl* has been called, but only after data is written to or read from the DTI for the first time using the node handle, which was also used when activating monitoring. Writing to or reading from the DTI via a connection or a node handle for which no monitoring is set does therefore not enable monitoring for another connection.

Once access has been enabled for the first time, all subsequent access must be enabled within the set timeout, otherwise the NetFail signal is activated.

Deactivating Monitoring

Monitoring is deactivated by calling the *ETH_ClearDTITimeoutCtrl* function or by closing the relevant DTI node using the *DDI_DevCloseNode* function.

If a connection is interrupted by the bus coupler as a result of DTI monitoring, the monitoring mode for this connection is deactivated and the corresponding DDI node is closed (see also "ETH_SETDTITimeoutCTRL").

If the bus coupler detects that a connection has been interrupted without the node having been closed, the NetFail signal is set. This applies especially if the controlling process (application program) is closed with an uncontrolled action (e.g., pressing Ctrl+C) and all the open data channels are closed by the operating system.

Status of the NetFail Signal

The user can read the status of the NetFail signal using the *ETH_GetNetFailStatus* function. In addition to the status of the NetFail signal, a second parameter is returned, which indicates the reason if the NetFail signal has been set. An additional function for the controlled setting of the NetFail signal is provided for test purposes. This enables the behavior of the system in the event of a NetFail to be tested, especially during program development. The *ETH_SetNetFail* function only needs a valid node handle as a parameter, so that the corresponding module can be addressed in the network.

The NetFail signal can only be reset by calling the *ETH_ClrSysFailStatus* function or by executing a reset on the bus coupler.

ETH_SetDTITimeoutCtrl

Task:	The <i>ETH_SetDTITimeoutCtrl</i> function activates the node for monitoring the DTI data channel specified by the node handle. After this function has been called, monitoring checks whether process data is received regularly. The function is assigned a valid node handle for a DTI data channel and a pointer (<i>*time</i>) to a variable with the desired timeout time. After the function has been called, the timeout time calculated by the bus coupler can be found in the <i>USIGN16 *time</i> variable.	
Syntax:	IBDDIRET IBDDIFUNC ETH_SetDTITimeoutCtrl (IBDDIHND nodeHd, USIGN16 *time);	
Parameters:	IBDDIHND nodeHd	Node handle (DTI) for the bus coupler that is to be monitored.
	USIGN16 *time	Pointer to a variable, which contains the desired timeout time when called. If the function has been called successfully, the actual timeout time is then entered in this variable. The timeout time can be set to a value in the range from 110 ms to 65000 ms.
Return value:	IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

ETH_ClearDTITimeoutCtrl

Task:	The <i>ETH_ClearDTITimeoutCtrl</i> function deactivates the node for monitoring process data activity. This function only receives the node handle as a parameter, which is also used to activate monitoring. After the function has been called successfully, monitoring via this channel and for this client is deactivated. Other activated monitoring channels are not affected.	
Syntax:	IBDDIRET IBDDIFUNC ETH_ClearDTITimeoutCtrl(IBDDIHND nodeHd);	
Parameter:	IBDDIHND nodeHd	Node handle (DTI) for the bus coupler for which monitoring is to be deactivated. The same node handle that was used for activating monitoring must also be used here.
Return value:	IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Example**Unix / Windows NT/2000**

```
IBDDIHND ddiHnd;
{
    IBDDIRET ddiRet;
    .
    .
    .
    ddiRet = ETH_ClearDTITimeoutCtrl (ddiHnd);
    .
    .
}
```

3.8.4 I/O Fault Response Mode

In case the communication connection is disrupted, the user can select the reaction of the FL IL 24 BK-PAC beforehand. Use the DDI command "Set_Value" on the object ID 2277_{hex}. The following table shows the three possible reactions:

Table 3-6 Available fault response modes

Fault Response Mode	Value	Function
Reset fault mode (default)	1	The digital outputs are set to "0". The analog outputs are set to a value (default = "0") programmed by the user.
Standard fault mode	0	All outputs are set to "0".
Hold last state mode	2	All outputs hold the last value.

The FL IL 24 BK(-PAC) only has one internal volatile memory where the process data is stored during runtime. This memory image is cyclically mapped to the appropriate Inline modules.

3.8.4.1 Power-up Table

Table 3-7 Power-up sequence

Power-up Sequence				
Status of the FL IL 24 BK-PAC	Configuration: Reset Fault Mode		Configuration: Last State Fault Mode	
	Internal memory	Actual output	Internal memory	Actual output
Power up	"0"	"0"	"0"	"0"
First write access to the internal memory after power up	"0" plus the new values	Internal memory	"0" plus the new values	Internal memory
Operating	"0" plus the sum of all new values	Internal memory	"0" plus the sum of all new values	Internal memory

Example: A station consists of a total of three I/O modules: one 16-bit analog output module (AO), one 16-bit digital output module (DO16), and one 2-bit digital output module (DO 2). After power up all outputs are set to "0":

Module	AO	DO 16	DO 2
Value	0x0000	0x0000	0x0000

Writing 0x0200 to the DO 16 module as the first value results in the following output values:

Module	AO	DO 16	DO 2
Value	0x0000	0x0200	0x0000

The state is "0' plus the new values".

Writing the following values to the relevant modules, e.g., 0x0010 on the AO, 0x0001 on the DO 2 and 0xACDC on the DO 16, results in the following output values:

Module	AO	DO 16	DO 2
Value	0x0010	0xACDC	0x0001

The state is "0' plus the sum of all new values".

3.8.4.2 Connection Monitoring Table

This table shows the output values after connection monitoring or the process data watchdog detected an error, e.g., connection interrupted or communication error, while the power supply was kept.

Table 3-8 Connection monitoring table

Connection monitoring table after connection abort, cable interrupt or communication error				
Configuration of the FL IL 24 BK-PAC	Configuration: "Reset Fault Mode "		Configuration: "Last State Fault Mode"	
	Internal memory	Actual output	Internal memory	Actual output
After connection abort, cable interrupt or communication error	Last value in internal memory	The digital outputs are set to "0".	Last value in internal memory	Values in internal memory
First write access in the output table after the connection has been re-established	Last values in internal memory plus newly written values	Internal memory	Last values in the output table plus newly written values	Internal memory
Operating	Last values in internal memory plus all newly written values	Internal memory	Last values in internal memory plus all newly written values	Internal memory

Example: The last entries in the internal memory have the following values:

Module	AO	DO 16	DO 2
Value	0x0123	0x4321	0x0002

Writing the value 0x00A1 to the internal memory of the DO 16 after the connection has been re-established, results in the following actual output value:

Module	AO	DO 16	DO 2
Value	0x0123	0x00A1	0x0002

The state is "Last values in internal memory plus newly written values".

Writing the following values to the internal memory, e.g., 0x0010 on the AO, 0x0001 on the DO 2 and 0xACDC on the DO 16, results in the following output values:

Module	AO	DO 16	DO 2
Value	0x0010	0xACDC	0x0001

The state is "Last values in internal memory plus all newly written values".

3.8.5 Treatment of the NetFail Signal/Testing With ETH_SetNetFail

The NetFail signal is set by writing a register in the coupling memory of the bus coupler. As soon as this signal is detected by the bus coupler, all local bus device outputs are reset and the PCP connections to the devices are interrupted.

The NetFail signal must be set to zero before process data can be output again. The NetFail signal is always set if the connection to the client is interrupted, the bus coupler does not write data to the DTI within the specified time or a general malfunction has been detected on the bus coupler, which prevents safe operation.

Setting the NetFail signal is indicated by setting the NetFail bit in the control word of each data telegram, which is sent by the bus coupler. The NetFail signal can be reset using the appropriate command or, if this is no longer possible, by pressing the reset key on the front plate of the bus coupler.

ETH_SetNetFail

Task:	The <i>ETH_SetNetFail</i> function sets the NetFail signal on the bus coupler and thus prevents further output of process data to the local bus devices. The function is assigned a node handle for a DTI or mailbox data channel of the relevant bus coupler as a parameter.	
Syntax:	IBDDIRET IBDDIFUNC ETH_SetNetFail (IBDDIHND nodeHd);	
Parameter:	IBDDIHND nodeHd	Node handle (MXI or DTI) for the bus coupler on which the NetFail signal is to be executed.
Return value:	IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Example**Unix / Windows NT/2000**

```

IBDDIHnd ddiHnd;
{
    IBDDIRET ddiRet;
    .
    .
    .
    ddiRet = ETH_SetNetFail (ddiHnd);
    .
    .
    .
}

```

ETH_GetNetFailStatus

Task:	The <i>ETH_GetNetFailStatus</i> function sends the NetFail status to the user, which is determined by the node handle of the bus coupler. The function is assigned a node handle for an open DTI or MXI data channel and a pointer to a <i>T_ETH_NET_FAIL</i> structure as parameters. After the function has been called successfully, the structure components contain the status (<i>status</i>) of the NetFail signal and an error code (<i>reason</i>) if the NetFail signal has been set. If the NetFail signal is not set, the <i>status</i> structure component has the value 0. Otherwise <i>status</i> has the value 0xFFFF. The <i>reason</i> structure component is only valid if the NetFail signal is set. The possible values for <i>reason</i> can be found in the IOCTL.H file.	
Syntax:	IBDDIRET IBDDIFUNC ETH_GetNetFailStatus (IBDDIHND nodeHd, T_ETH_NET_FAIL *netFailInfo);	

Parameters: IBDDIHND nodeHd Node handle (MXI or DTI) for the bus coupler on which the NetFail status is to be read.

T_ETH_NET_FAIL *netFailInfo

Pointer to a structure, which contains the NetFail status and the reason for the NetFail, if applicable.

Return value: IBDDIRET If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

**Format of the
T_ETH_NET_FAIL
structure**

```
typedef struct {  
    USIGN16 status; /* NetFail status */  
    USIGN16 reason; /* Reason for the NetFail */  
} T_ETH_NET_FAIL;
```

**Possible values for the
status structure
component:**

```
ETH_NET_FAIL_ACTIVE 0xFFFF  
/* NetFail triggered */  
(See also "Reasons for Fault Response" on page 3-34)  
ETH_NET_FAIL_INACTIVE 0x0000  
/* NetFail signal not triggered */
```

Example

Unix / Windows NT/2000

```
IBDDIHND ddihnd;  
{  
    IBDDIRET ddiRet;  
  
    T_ETH_NET_FAIL netFailInfo  
    USIGN16 nfStatus;  
    USIGN16 nfReason;  
    .  
    .  
    .  
    ddiRet = ETH_GetNetFailStatus (ddiHnd,  
    &netFailInfo);  
  
    if (ddiRet == ERR_OK)  
    {  
        nfStatus = netFailInfo.status  
        nfReason = netFailInfo.reason;  
    }  
    .  
    .  
    .  
}
```

ETH_ClrNetFailStatus

Task:

The *ETH_ClrNetFailStatus* function resets the NetFail signal. This means that process data can be output again and the status of the NetFail signal is set to 0. The function is assigned a valid node handle for a DTI or MXI data channel as a parameter.



Observe that this task may depend on other active monitoring functions.

Syntax:

IBDDIRET IBDDIFUNC ETH_ClrNetFailStatus (IBDDIHND nodeHd);

Parameter:

IBDDIHND nodeHd

Node handle (MXI or DTI) for the bus coupler on which the NetFail status is to be reset.

Return value:

IBDDIRET

If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Example**Unix / Windows NT/2000**

```
IBDDIHND ddiHnd;
{
    IBDDIRET ddiRet;
    .
    .
    .
    ddiRet = ETH_ClrNetFailStatus (ddiHnd);
    .
    .
    .
}
```

ETH_SetNetFailMode

Task:

The *ETH_SetNetFailMode* routine is used to change the behavior of the controller board in the event of a NetFail. After startup, the controller board is in standard mode (*ETH_NF_STD_MODE*), which means that if a NetFail occurs, all outputs of the modules connected to the INTERBUS system are set to zero and the bus continues to run. This behavior can be changed by calling the routine. At present, the controller board supports two different modes:

- Standard mode: The controller board behavior remains the same, i.e., the outputs are set to zero in the event of an error.
- Alarm stop mode: Not only are the outputs set to zero but an alarm stop command is also sent to the controller board.

If the function is executed successfully, the routine returns the return value 0 (*ERR_OK*). In the event of an error, the return value is an error code (see *DDI_ERR.H*).



In alarm stop mode, a command is sent to the controller board but the return value is not obtained. That means that an application program will receive this message on its next read attempt.

Syntax:

```
IBDDIRET IBDDIFUNC ETH_SetNetFailMode(IBDDIHND nodeHd,
T_ETH_NET_FAIL_MODE *netFailModelInfo);
```

The routine receives a valid node handle and a pointer to the structure described below as parameters. In addition to a component in which the mode to be set is entered, the structure contains a pointer to an optional parameter block, the size of which is also entered in the structure. This parameter block is purely optional and is not used for the modes that exist at present. Thus, the structure component *numOfBytes* should be set to zero.

Parameters:

IBDDIHND nodeHd Node handle of a controller board for which the NetFail mode is to be changed.

T_ETH_NET_FAIL_MODE *netFailModelInfo

Pointer to a *T_ETH_NET_FAIL_MODE* data structure. This structure contains the parameters for setting the *NetFail mode* and, if necessary, optional parameters.

Format of the T_ETH_NET_FAIL_ MODE

```
typedef struct {
    USIGN16 mode;                                      /* NetFail mode */
    USIGN16 numOfBytes; /* Size of the parameter block in bytes */
    VOID *miscParamPtr; /* Parameter for the
                        relevant NetFail mode */
} T_ETH_NET_FAIL_MODE;
```

The function prototypes, the type definition of the data structure, and the symbolic constants can be found in the *IOCTRL.H* file.

ETH_GetNetFailMode

Task: The *ETH_GetNetFailMode* function can be used to read the set *NetFail mode*. The routine expects a valid node handle and a pointer to a *T_ETH_NET_FAIL_MODE* data structure (see above) as parameters. After the routine has been called successfully, the user can read the set NetFail mode from the structure. If there are no additional parameters for this mode, this is indicated by the *numOfBytes* structure component, which contains the value zero in this case.

Syntax: IBDDIRET IBDDIFUNC ETH_GetNetFailMode(IBDDIHND nodeHd,
T_ETH_NET_FAIL_MODE *netFailModelInfo)

Parameters:

IBDDIHND nodeHd	Node handle of a controller board from which information on the set NetFail mode is to be read.
T_ETH_NET_FAIL_MODE *netFailModelInfo	Pointer to a T_ETH_NET_FAIL_MODE data structure. If the function is called successfully, the parameters of the NetFail mode set on the controller board as well as the mode itself are entered in this structure.

Format of the structure

```
typedef struct {
    USIGN16 mode;           /* NetFail mode */
    USIGN16 numOfBytes;     /* Size of the parameter
                           block in bytes */
    VOID *miscParamPtr;     /* Parameters for the
                           relevant NetFail mode */
} T_ETH_NET_FAIL_MODE;
```

Constants of the different NetFail modes

```
#define ETH_NF_STD_MODE           0
#define ETH_NF_ALARMSTOP_MODE    1
#define ETH_NF_HOLD_LAST_STATE_MODE 2
```

The function prototypes, the type definition of the data structure, and the symbolic constants can be found in the IOCTL.H file.

3.9 IN Process Data Monitoring

Functions that automatically monitor the IN process data area for changes can be used to reduce the load on the Ethernet network. In systems in which input signals only change slowly or rarely change, the same process data is often transmitted in successive read cycles.

Transmission of the same data loads the network and the client (user workstation) but does not provide any additional information. That is why it is possible to only transmit the IN process data to the client if this data has changed.

The user now has the option to define an area to be monitored by the controller board. This area is read by the controller board firmware cyclically and compared with a reference image of the process data. The comparison of the defined area with the process image of the reference data and the transmission of the data to the relevant client takes place within a period of ≥ 22 ms.

If it is established that the data that has been read differs from the reference image, the read data is automatically sent to the relevant client and entered as the new reference image.

In addition, areas in which changes are not taken into account can be specified. This provides an easy option for masking out the low-order bits of an analog input that change frequently. The modified data is sent by an unconfirmed service.

ETH_ActivatePDInMonitoring

Task: The *ETH_ActivatePDInMonitoring* function activates the mode for monitoring the IN process data for potential changes. This mode can only be activated once on each controller board.

The function is assigned a valid node handle for a DTI data channel and a pointer to a T_ETH_PD_IN_MON structure as parameters. The T_ETH_PD_IN_MON structure contains all the information needed to parameterize the IN process data monitoring:

mode	Mode in which the monitoring is to be executed.
address	Start address (in bytes) from which the input data is to be monitored.
numOfBytes	Size of the area to be monitored in bytes (it must not exceed 1024 bytes).
*maskData	Pointer to a vector with the masking data.
*notifyFuncPtr	Zero (is not supported)

Function: The masking data is combined bit by bit with the data that has been read and determines whether a change in the associated IN data bit will lead to notification of the client. A set bit (1) means that this bit is of significance for monitoring. A bit that is not set (0) means that a change in the associated bit in the IN process data area is insignificant.



If the "IN process data monitoring" function is used, it must be deactivated again before closing the connection (DDI_DevCloseNode).

Syntax: IBDDIRET IBDDIFUNC ETH_ActivatePDInMonitoring(IBDDIHND nodeHd, T_ETH_PD_IN_MON *infoPtr);

Parameters:

IBDDIHND nodeHd	Node handle (DTI) for the controller board for which process data monitoring is to be activated.
T_ETH_PD_IN_MON *infoPtr	Pointer to a T_ETH_PD_IN_MON data structure. This structure contains all the parameters needed to activate monitoring.

Return value: IBDDIRET

If the function is executed successfully, the value 0 (ERR_OK) is returned. Otherwise the return value is an error code.

Format of the data structure:

```
typedef struct {
    USIGN16 mode;           /* Selects the monitoring
                             mode*/
    USIGN16 address;        /* Start address of the
                             area to be monitored*/
    USIGN16 numOfBytes;     /* Size of the
                             area to be monitored*/
    USIGN8 *maskData;       /* Pointer to buffer with the
                             masking data*/
                             /* The size of the buffer
                             corresponds to numOfBytes*/
    VOID (*notifyFuncPtr)(IBDDIHND nodeHd, T_DDI_DTI_ACCESS
                           *dtiAcc);
                             /* Pointer to a function
                             that is called if there is
                             a change in the IN PD data.*/
    USIGN32 timeout;        /* Timeout time in ms
                             */
} T_ETH_PD_IN_MON;
```

Constants for the different modes

```
#define ETH_PD_IN_CHK_INACTIVE 0x0000 /* PD In Check is not activated */
#define ETH_PD_IN_CHK_MODE_UDP 0x0002 /* Send data over UDP port */
```

Description of the constants**ETH_PD_IN_CHK_INACTIVE**

Not in use at present.

ETH_PD_IN_CHK_MODE_UDP

The controller board sends the process data to the client using UDP. The routine automatically determines which port is used, i.e., the user does not normally have any information about the port used. For this reason, the user is provided with one routine that carries out all necessary tasks, thus ensuring that this function is easy to use:

```
IBDDIRET IBDDIFUNC WaitForPDInIndication(IBDDIHND nodeHd,
T_DDI_DTI_ACCESS *dtiAcc)
```

The WaitForPDInIndication function is only assigned the node handle of a valid data channel and a pointer to a *T_DDI_DTI_ACCESS* structure. The routine returns as soon as process data is received or the timeout time that was preset in *timeout* (see *T_ETH_PD_IN_MON*) has elapsed. The components of the *T_DDI_DTI_ACCESS* structure are used to access the process data. The routine returns an integer value, which indicates whether process data has been received and is ready to be evaluated or whether a timeout or another error caused the routine to be terminated. A return value that is not zero always indicates an error that can be defined more specifically using the value.

Proceed as follows:

- Activate process data monitoring with `ETH_ActivatePDInMonitoring`
- Wait for process data (input data) with `WaitForPDInIndication`

The standard DTI functions can be used to read and write input and output values at any time, even if `WaitForPDInIndication` has been used in another thread to wait for an indication.

If the controller board transmits data more quickly than the client retrieves it, the client saves a certain amount of this data to prevent it from being lost immediately. The amount of data saved by the client depends on the system used and the settings in its TCP/IP protocol stack.



As UDP is used as the transmission protocol, it is not clear whether data packets sent by the controller board actually reach the receiver. The controller board does not repeat packets that are lost on the way to the client.

The `T_DDI_DTI_ACCESS` structure is not explained here because it has already been described in detail in the standard DTI routines.

ETH_DeactivatePDInMonitoring

Task:	The <code>ETH_DeactivatePDInMonitoring</code> function deactivates IN process data monitoring. The function is only assigned the node handle as a parameter, which is also used to activate monitoring with <code>ETH_ActivatePDInMonitoring</code> .	
Syntax:	<code>IBDDIRET IBDDIFUNC ETH_DeactivatePDInMonitoring (IBDDIHND nodeHd);</code>	
Parameter:	<code>IBDDIHND nodeHd</code>	Node handle (DTI) for the controller board for which process data monitoring is to be deactivated.
Return value:	<code>IBDDIRET</code>	If the function is executed successfully, the value 0 (<code>ERR_OK</code>) is returned. Otherwise the return value is an error code.



The format of the `T_ETH_PD_IN_MON` structure in the function used to activate process data monitoring, `ETH_ActivatePDInMonitoring`, has been modified in client software version 1.10 or later. A modification was necessary in order to enable additional monitoring modes and transfer the parameters required for these modes. Active "IN process data monitoring" must always be deactivated before closing the connection (`DDI_DevCloseNode`).

3.10 Notification Mode

General: Notification mode enable messages received in the MPM (e.g., a message from the INTERBUS controller board) to be made available to the application program immediately.

This reduces the load on the network and the computer because messages do not have to be scanned cyclically. Data is only transmitted via the network if there is actually a message in the MPM or a specified timeout time has elapsed.

Notification Mode

Task: A feature of notification mode is that the message is awaited on the controller board. A *DDI_MXI_RcvMessage* call waits on the controller board until there is a message or the preset timeout time has elapsed. No other requests can be sent via the channel during this period. Thus, the data channel is practically blocked.

When notification mode is activated, the timeout time is entered in the *T_ETH_NOTIFY_INFO* structure and transmitted to the controller board. The timeout time is endless if the value `FFFF FFFFhex` is entered.



Not possible under Windows NT: A call that blocks the data channel can be terminated by calling the *DDI_ClrMsgNotification* routine. Another MXI data channel should be used for this or another mailbox connection to the controller board should be created. To remove notification mode using this additional connection, the value `FEDChex` (symbolic constant *ETH_NOTIFY_ABORT*) should be entered in the *mode* structure component. The *DDI_MXI_RcvMessage* function then returns the error message *ERR_BLOCK_TIMEOUT*.

Syntax to activate: `IBDDIRET IBDDIFUNC DDI_SetMsgNotification(IBDDIHND nodeHd, T_ETH_NOTIFY_INFO IBPTR *notifyInfoPtr)`

Syntax to deactivate: `IBDDIRET IBDDIFUNC DDI_ClrMsgNotification(IBDDIHND nodeHd, T_ETH_NOTIFY_INFO IBPTR *notifyInfoPtr)`

UNIX

Parameters:	<code>mode</code>	Notification mode
	<code>processId</code>	
	<code>threadId</code>	
	<code>timeout</code>	Abort time in milliseconds

Format of the structure:

```
typedef struct {
    USIGN32 mode; /* Defines the notification mode */
    USIGN32 threadId; /*Thread identifier */
    USIGN32 processId; /*Process identifier */
    USIGN32 timeout; /*Timeout time in milliseconds */
}T_ETH_NOTIFY_INFO;
```

Constants:

```
#define ETH_NOTIFY_MODE_1
```

Windows NT/2000

Parameters:

```
processId
threadId
timeout                Abort time in milliseconds
```

Format of the structure:

```
typedef struct {
    DWORD threadId; /*Thread identifier */
    DWORD processId; /*Process identifier */
    USIGN32 timeout; /*Timeout time in milliseconds */
}T_IBS_WIN32_NOTIFY;
```



Timeout values can only be integer values.

3.11 Programming Support Macros

3.11.1 Introduction

The macros described in this section make it easier to program the application program. These macros also support data transfer (commands, messages, and data) between Intel format and Motorola 68xxx format if a workstation with Intel format is used to create an application program.

The Inline local bus numbers words (16-bit) according to the conventional counting method of the **P**rogrammable **L**ogic **C**ontroller (PLC). Because consecutive words start on even byte addresses (1 byte = 8 bits), they are also numbered according to the even byte addresses. For example, the word which contains bytes 6 and 7 is assigned the number 4.

The process data is sent to the computer as bytes. Because the data on the bus coupler is in Motorola format, it is also received in this format on the computer. If the processor on the computer is in BigEndian format (Motorola), the data can also be processed further in a word-oriented way without conversion. In a processor in LittleEndian format (Intel), the data must be converted accordingly (word-oriented).

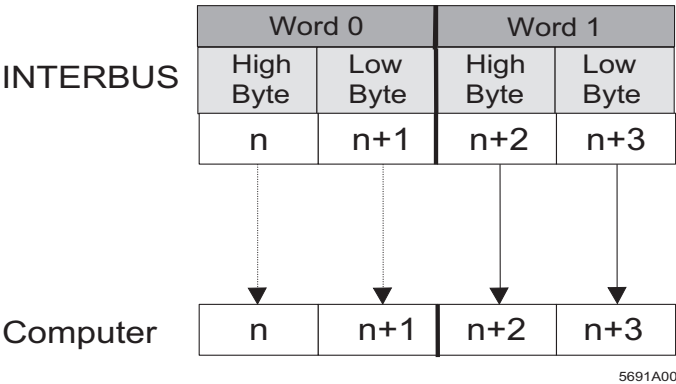


Figure 3-12 Assignment of the process data between the local bus and the computer systems

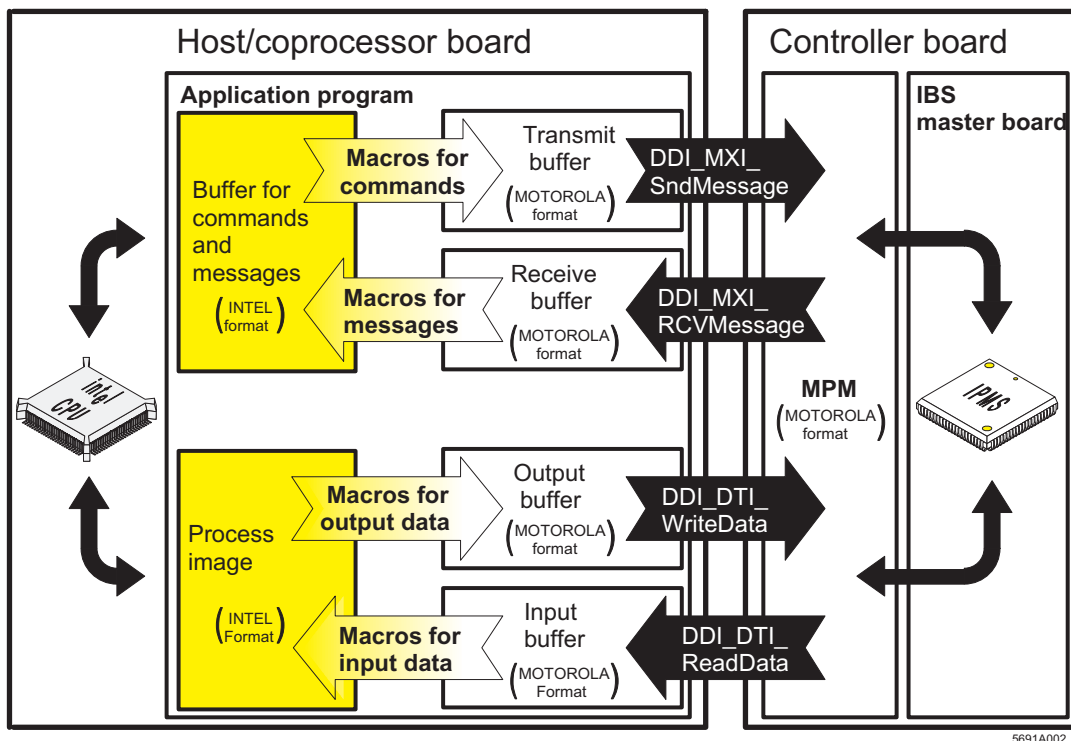


Figure 3-13 Using the macros for programming support



The macros are available for both processor types. However, for processors in Motorola format, the macros have no function.

3.12 Description of the Macros

Table 3-9 Driver software macros

Macro	Task	Page
IB_SetCmdCode	Enters the command code (16-bit) in the specified transmit buffer	3-61
IB_SetParaCnt	Enters the parameter count (16-bit) in the specified transmit buffer	3-61
IB_SetParaN	Enters a parameter (16-bit) in the specified transmit buffer	3-62
IB_SetParaNHiByte	Enters the high-order byte (bit 8 to 15) of a parameter in the specified transmit buffer	3-62
IB_SetParaNLoByte	Enters the low-order byte (bit 0 to 7) of a parameter in the specified transmit buffer	3-62
IB_SetBytePtrHiByte	Returns the address of a parameter entry starting with the high-order byte (bit 8 to 15)	3-62
IB_SetBytePtrLoByte	Returns the address of a parameter entry starting with the low-order byte (bit 0 to 7)	3-63
IB_GetMsgCode	Reads a message code (16-bit) from the specified receive buffer	3-63
IB_GetParaCnt	Reads the parameter count (16-bit) from the specified receive buffer	3-63
IB_GetParaN	Reads a parameter (16-bit) from the specified receive buffer	3-63
IB_GetParaNHiByte	Reads the high-order byte (bit 8 to 15) of a parameter from the specified receive buffer	3-64
IB_GetParaNLoByte	Reads the low-order byte (bit 0 to 7) of a parameter from the specified receive buffer	3-64
IB_GetBytePtrHiByte	Returns the address of a parameter entry starting with the high-order byte (bit 8 to 15)	3-64
IB_GetBytePtrLoByte	Returns the address of a parameter entry starting with the low-order byte (bit 0 to 7)	3-64
IB_PD_GetLongDataN	Reads a double word (32-bit) from the specified position in the input buffer	3-65
IB_PD_GetDataN	Reads a word (16-bit) from the specified position in the input buffer	3-65
IB_PD_GetDataNHiByte	Reads the high-order byte (bit 8 to 15) of a word from the input buffer	3-65

Table 3-9 Driver software macros

Macro	Task	Page
IB_PD_GetDataNLoByte	Reads the low-order byte (bit 0 to 7) of a word from the input buffer	3-65
IB_PD_GetBytePtrHiByte	Returns the address of a word starting with the high-order byte (bit 8 to 15)	3-66
IB_PD_GetBytePtrLoByte	Returns the address of a word starting with the low-order byte (bit 0 to 7)	3-66
IB_PD_SetLongDataN	Writes a double word (32-bit) to the output buffer	3-66
IB_PD_SetDataN	Writes a word (16-bit) to the output buffer	3-66
IB_PD_GetDataNHiByte	Writes the high-order byte (bit 8 to 15) of a word to the output buffer	3-67
IB_PD_GetDataNLoByte	Writes the low-order byte (bit 0 to 7) of a word to the output buffer	3-67
IB_PD_GetBytePtrHiByte	Returns the address of a word starting with the high-order byte (bit 8 to 15)	3-67
IB_PD_GetBytePtrLoByte	Returns the address of a word starting with the low-order byte (bit 0 to 7)	3-67

The macros are defined for different operating systems and compilers in the Device Driver Interface so that they can be used universally.

3.12.1 Macros for Converting the Data Block of a Command

IB_SetCmdCode (n, m)

Task: This macro converts a command code (16-bit) into Motorola format and enters it in the specified transmit buffer.

Parameters: n(USIGN8 *): Pointer to the transmit buffer
m(USIGN16): Command code to be entered

IB_SetParaCnt (n, m)

Task: This macro converts the parameter count (16-bit) into Motorola format and enters it in the specified transmit buffer. The call is only necessary when dealing with a command with parameters. The parameter count specifies the number of subsequent parameters in words.

Parameters: n(USIGN8 *): Pointer to the transmit buffer
m(USIGN16): Parameter count to be entered

IB_SetParaN (n, m, o)

Task: This macro converts a parameter (16-bit) into Motorola format and enters it in the specified transmit buffer. The call is only necessary when dealing with a command with parameters.

Parameters:

n(USIGN8 *):	Pointer to the transmit buffer
m(USIGN16):	Parameter number (counting starts with 1)
o(USIGN16):	Parameter value to be entered

IB_SetParaNHiByte (n, m, o)

Task: This macro converts the high-order byte (bit 8 to 15) of a parameter into Motorola format and enters it in the specified transmit buffer.

Parameters:

n(USIGN8 *):	Pointer to the transmit buffer
m(USIGN16):	Parameter number
o(USIGN8):	Parameter to be entered (byte)

IB_SetParaNLoByte (n, m, o)

Task: This macro converts the low-order byte (bit 0 to 7) of a parameter into Motorola format and enters it in the specified transmit buffer.

Parameters:

n(USIGN8 *):	Pointer to the transmit buffer
m(USIGN16):	Parameter number
o(USIGN8):	Parameter to be entered (byte)

IB_SetBytePtrHiByte (n, m)

Task: This macro returns the address of a parameter entry starting with the high-order byte (bit 8 to 15). The address is a *USIGN8 ** data type.

Parameters:

n(USIGN8 *):	Pointer to the transmit buffer
m(USIGN16):	Parameter number

Return value: (USIGN8 *): Address of the high-order byte of the parameter in the transmit buffer.

IB_SetBytePtrLoByte (n, m)

Task:	This macro returns the address of a parameter entry starting with the low-order byte (bit 0 to 7). The address is a <i>USIGN8</i> * data type.	
Parameters:	n(USIGN8 *):	Pointer to the transmit buffer
	m(USIGN16):	Parameter number
Return value:	(USIGN8 *):	Address of the low-order byte of the parameter in the transmit buffer.

3.12.2 Macros for Converting the Data Block of a Message**IB_GetMsgCode (n)**

Task:	This macro reads the message code (16-bit) from the specified receive buffer and converts it into Intel format.	
Parameter:	n(USIGN8 *):	Pointer to the receive buffer
Return value:	(USIGN16):	Message code

IB_GetParaCnt (n)

Task:	This macro reads the parameter count (16-bit) from the data block of the message and converts it into Intel format. The parameter count specifies the number of subsequent parameters in words.	
Parameter:	n(USIGN8 *):	Pointer to the receive buffer
Return value:	(USIGN16):	Parameter count
Remark:	Only read the parameter count for messages that also have parameters.	

IB_GetParaN (n, m)

Task:	This macro reads a parameter value (16-bit) from the data block of the message and converts it into Intel format.	
Parameters:	n(USIGN8 *):	Pointer to the receive buffer
	m(USIGN16):	Parameter number
Return value:	(USIGN16):	Parameter value
Remark:	Only read parameters for messages that also have parameters.	

IB_GetParaNHiByte (n, m)

Task: This macro reads the high-order byte (bit 8 to 15) of a parameter from the specified receive buffer and converts it into Intel format.

Parameters: n(USIGN8 *): Pointer to the receive buffer
 m(USIGN16): Parameter number

Return value: (USIGN8): Parameter value (byte)

Remark: Only read parameters for messages that also have parameters.

IB_GetParaNLoByte (n, m)

Task: This macro reads the low-order byte (bit 0 to 7) of a parameter from the specified receive buffer and converts it into Intel format.

Parameters: n(USIGN8 *): Pointer to the receive buffer
 m(USIGN16): Parameter number

Return value: (USIGN8): Parameter value (byte)

Remark: Only read parameters for messages that also have parameters.

IB_GetBytePtrHiByte (n, m)

Task: This macro returns the address of a parameter entry starting with the high-order byte (bit 8 to 15). The address is a *USIGN8 ** data type.

Parameters: n(USIGN8 *): Pointer to the receive buffer
 m(USIGN16): Parameter number

Return value: (USIGN8 *): Address of the high-order byte of a parameter in the receive buffer.

IB_GetBytePtrLoByte (n, m)

Task: This macro returns the address of a parameter entry starting with the low-order byte (bit 0 to 7). The address is a *USIGN8 ** data type.

Parameters: n(USIGN8 *): Pointer to the receive buffer
 m(USIGN16): Parameter number

Return value: (USIGN8 *): Address of the low-order byte of a parameter in the receive buffer.

3.12.3 Macros for Converting Input Data

The IBS_MACR.H file contains macros for converting double words, words, and bytes from Motorola to Intel format. Addressing is always word-oriented here.

IB_PD_GetLongDataN (n, m)

Task: This macro reads a double word (32-bit) from the specified position in the input buffer and converts it into Intel format. The word index in the input buffer is used as the position. The macro reads the double word starting from the specified word address over two words.

Parameters: n (USIGN8 *) Pointer to the input buffer
 m (USIGN16) Word number

IB_PD_GetDataN (n, m)

Task: This macro reads a word (16-bit) from the specified position in the input buffer and converts it into Intel format, if necessary.

Parameters: n(USIGN8 *): Pointer to the input buffer
 m(USIGN16): Word number

Return value: (USIGN16): Process data (16-bit)

IB_PD_GetDataNHiByte (n, m)

Task: This macro reads the high-order byte (bit 8 to 15) of a word from the input buffer and converts it into Intel format.

Parameters: n(USIGN8 *): Pointer to the input buffer
 m(USIGN16): Word number

Return value: (USIGN8): Process data (8-bit)

IB_PD_GetDataNLoByte (n, m)

Task: This macro reads the low-order byte (bit 0 to 7) of a word from the input buffer and converts it into Intel format.

Parameters: n(USIGN8 *): Pointer to the input buffer
 m(USIGN16): Word number

Return value: (USIGN8): Process data (8-bit)

IB_PD_GetBytePtrHiByte (n, m)

Task:	This macro returns the address of a word starting with the high-order byte (bit 8 to 15).	
Parameters:	n(USIGN8 *):	Pointer to the input buffer
	m(USIGN16):	Word number
Return value:	(USIGN8 *):	Address of the high-order byte of a word in the input buffer.

IB_PD_GetBytePtrLoByte (n, m)

Task:	This macro returns the address of a word starting with the low-order byte (bit 0 to 7).	
Parameters:	n(USIGN8 *):	Pointer to the input buffer
	m(USIGN16):	Word number
Return value:	(USIGN8 *):	Address of the low-order byte of a word in the input buffer.

3.12.4 Macros for Converting Output Data

The IBS_MACR.H file contains macros for converting double words, words, and bytes from Intel to Motorola format. Addressing is always word-oriented here.

IB_PD_SetLongDataN (n, m, o)

Task:	This macro converts a double word (32-bit) to Motorola format and writes it to the specified position in the output buffer. The word index in the output buffer is used as the position. The macro writes the double word starting from the specified word address over two words.	
Parameters:	n (USIGN8 *)	Pointer to the output buffer
	m (USIGN16)	Word number
	o (USIGN32)	Process data (32-bit)

IB_PD_SetDataN (n, m, o)

Task:	This macro converts a word (16-bit) to Motorola format and writes it to the specified position in the output buffer.	
Parameters:	n(USIGN8 *):	Pointer to the output buffer
	m(USIGN16):	Word number
	o(USIGN16):	Process data (16-bit)

IB_PD_SetDataNHiByte(n, m, o)

Task: This macro converts the high-order byte (bit 8 to 15) of a word to Motorola format and writes it to the specified position in the output buffer.

Parameters:

n(USIGN8 *):	Pointer to the output buffer
m(USIGN16):	Word number
o(USIGN8):	Process data (8-bit)

IB_PD_SetDataNLoByte (n, m, o)

Task: This macro converts the low-order byte (bit 0 to 7) of a word to Motorola format and writes it to the specified position in the output buffer.

Parameters:

n(USIGN8 *):	Pointer to the output buffer
m(USIGN16):	Word number
o(USIGN8):	Process data (8-bit)

IB_PD_SetBytePtrHiByte (n, m)

Task: This macro returns the address of a word starting with the high-order byte (bit 8 to 15).

Parameters:

n(USIGN8 *):	Pointer to the output buffer
m(USIGN16):	Word number

Return value: (USIGN8 *): Address of the high-order byte of a word in the output buffer.

IB_PD_SetBytePtrLoByte (n, m)

Task: This macro returns the address of a word starting with the low-order byte (bit 0 to 7).

Parameters:

n(USIGN8 *):	Pointer to the output buffer
m(USIGN16):	Word number

Return value: (USIGN8 *): Address of the low-order byte of a word in the output buffer.

3.13 Diagnostic Options for Driver Software

3.13.1 Introduction

The driver software diagnostics uses error messages and error codes for the individual functions. These error codes can be used to precisely define the cause of an error. An offset (ERR_BASE) depending on the operating system is added to each code listed here. This offset is already taken into account when using the error message definitions.

Table 3-10 Driver software messages

Code	Error Message	Cause	Page
0000 _{hex}	ERR_OK	The function was executed successfully	3-69
0085 _{hex}	ERR_INVLD_NODE_HD	Invalid node handle specified	3-70
0086 _{hex}	ERR_INVLD_NODE_STATE	Node handle of a data channel that is already closed specified	3-70
0087 _{hex}	ERR_NODE_NOT_READY	Desired node not ready	3-70
0088 _{hex}	ERR_WRONG_DEV_TYP	Incorrect node handle	3-70
0089 _{hex}	ERR_DEV_NOT_READY	Local bus master not ready yet	3-71
008A _{hex}	ERR_INVLD_PERM	Access type not enabled for channel	3-71
008C _{hex}	ERR_INVLD_CMD	Utility function is not supported by driver Version 0.9	3-71
008D _{hex}	ERR_INVLD_PARAM	Command contains invalid parameter	3-71
0090 _{hex}	ERR_NODE_NOT PRES	Node not available	3-72
0091 _{hex}	ERR_INVLD_DEV_NAME	Unknown device name used	3-72
0092 _{hex}	ERR_NO_MORE_HNDL	Device driver resources used up	3-72
0096 _{hex}	ERR_AREA_EXCDED	Access exceeds limit of selected data area	3-75
0097 _{hex}	ERR_INVLD_DATA_CONS	Specified data consistency is not permitted	3-75
009A _{hex}	ERR_MSG_TO_LONG	Message or command contains too many parameters	3-73
009B _{hex}	ERR_NO_MSG	No message present	3-73
009C _{hex}	ERR_NO_MORE_MAILBOX	No further mailboxes of the required size free	3-73
009D _{hex}	ERR_SVR_IN_USE	Send vector register in use	3-73
009E _{hex}	ERR_SVR_TIMEOUT	Invalid node called	3-74
009F _{hex}	ERR_AVR_TIMEOUT	Invalid node called	3-74

Table 3-10 Driver software messages

Code	Error Message	Cause	Page
00A9 _{hex}	ERR_PLUG_PLAY	Invalid write access to process data in P&P mode	3-75
0100 _{hex}	ERR_STATE_CONFLICT	This service is not permitted in the selected operating mode of the controller	3-75
0101 _{hex}	ERR_INVLD_CONN_TYPE	Service called via an invalid connection	3-76
0102 _{hex}	ERR_ACTIVATE_PD_CHK	IN process data monitoring could not be activated	3-76
0103 _{hex}	ERR_DATA_SIZE	The data volume is too large	3-76
0200 _{hex}	ERR_OPT_INVLD_CMD	Unknown command	3-76
0201 _{hex}	ERR_OPT_INVLD_PARAM	Invalid parameter	3-76
1010 _{hex}	ERR_IBSETH_OPEN	The IBSETHA file cannot be opened	3-76
1013 _{hex}	ERR_IBSETH_READ	The IBSETHA file cannot be read	3-77
1014 _{hex}	ERR_IBSETH_NAME	The device name cannot be found in the file	3-77
1016 _{hex}	ERR_IBSETH_INTERNET	The system cannot read the computer name/host address	3-77

3.14 Positive Messages

ERR_OK

0000_{hex}
Meaning:

After successful execution of a function, the driver software generates this message as a positive acknowledgment.

Cause:

No errors occurred during execution of the function.

3.15 Error Messages

If the Device Driver Interface (DDI) generates one of the following error messages as a negative acknowledgment, the function called previously was not processed successfully.

3.15.1 General Error Messages

These error messages can occur when calling any DDI function.

ERR_INVLD_NODE_HD **0085_{hex}**

Cause: An invalid node handle was used when calling the function.

Remedy: Use the valid node handle of a successfully opened data channel.

ERR_INVLD_NODE_STATE **0086_{hex}**

Cause: An invalid node handle was used when calling the function. This is the handle of a data channel that has already been closed.

Remedy: Open the data channel or use one that is already open.

ERR_NODE_NOT_READY **0087_{hex}**

Cause: The node to be used has not yet indicated it is ready, i.e., the node ready bit has not been set in the status register of the coupling memory. The cause of this may, for example, be a hardware fault.

Remedy: Check whether the bus coupler has been started up.

ERR_WRONG_DEV_TYP **0088_{hex}**

Cause: Incorrect node handle. An attempt has been made, e.g., to access the mailbox interface with a node handle for the data interface.

ERR_DEV_NOT_READY**0089_{hex}**

Cause: The local bus master was addressed, even though it was not ready.

Remedy: After a reset, request the local bus master using the *GetIBSDiagnostic()* function on the ready bit in the diagnostic bit register. Once this bit is set, the local bus master can be addressed.

ERR_INVLD_PERM**008A_{hex}**

Cause: An attempt has been made to execute a function on a channel for which the relevant access rights were not logged in when opening the data channel. This error occurs, e.g., if you want to write to the data interface, but read-only rights were specified on opening the channel (DDI_READ constant).

Remedy: Close the channel and open it again with modified access rights.

ERR_INVLD_CMD**008C_{hex}**

Cause: This error message is displayed if you are using an older driver library or an older DLL.

Remedy: Use the latest driver.

ERR_INVLD_PARAM**008D_{hex}**

Cause: This error message is displayed if invalid parameters are used in the command.

Remedy: Check the validity of the parameters used.

3.15.2 Error Messages When Opening a Data Channel

ERR_NODE_NOT_PRESENCE 0090_{hex}

Cause: An attempt was made to open a data channel to a node that does not exist.

Remedy: Select the following node.
IBS ETH: Node 1 = Local bus master

ERR_INVALID_DEVICE_NAME 0091_{hex}

Cause: An unknown device name was specified as a parameter on opening a data channel.

Remedy: Select a correct device name.

ERR_NO_MORE_HANDLES 0092_{hex}

Cause: Device driver resources used up. No further data channels can be opened. If you exit a program without closing the data channels in use, they will stay open. Additional data channels will be opened the next time the program is started. After this program has been started a number of times, the maximum permitted number of data channels that can be opened simultaneously will be reached and no more will be available.

Remedy: Close a data channel that is not required or reinstall the device driver. Always close all data channels used when exiting a program.

3.15.3 Error Messages When Transmitting Messages/Commands

ERR_MSG_TO_LONG

009A_{hex}

Cause 1: If an error message occurs when sending a command, then the length of the command exceeds the maximum number of permitted parameters.

Remedy: Reduce the number of parameters.

Cause 2: If an error message occurs when receiving a message, then the length of the message exceeds the length of the receive buffer specified.

Remedy: Increase the length of the receive buffer.

ERR_NO_MSG

009B_{hex}

Cause: This message occurs if an attempt has been made to retrieve a message using the *DDI_MXI_RcvMessage* function, but no message is present for the node specified by the node handle.

ERR_NO_MORE_MAILBOX

009C_{hex}

Cause 1: You have requested too many mailboxes within a short time.

Remedy: Increase the time interval between individual mailbox requests and restart the *DDI_MXI_SndMessage* service.

Cause 2: No further mailbox of the required size is available. Observe the maximum mailbox size that can be used (1020 bytes).

Remedy: Select a smaller mailbox or wait until a mailbox of the required size is free again.

Cause 3: An attempt was made to address the coprocessor board (COP), but it is faulty.

Remedy: Please get in touch with Phoenix Contact.

ERR_SVR_IN_USE

009D_{hex}

Cause: The send vector register for the node is in use.

Remedy: Address the register again or wait until the register is available again.

ERR_SVR_TIMEOUT**009E_{hex}**

- Meaning:** If a message placed in the MPM by the local bus master is not retrieved by the MPM node addressed, this node does not reset the acknowledge message bit set by the local bus master, i.e., the MPM node addressed does not indicate *Message detected*. After a specific time has elapsed (timeout), the local bus master generates the error message *ERR_SVR_TIMEOUT*. If this error message occurs repeatedly, it must be assumed that the node being addressed is no longer ready to accept the message.
- Cause:** Invalid node called:
An attempt was made, for example, to address the coprocessor board (COP), which is faulty.
- Remedy:** Please get in touch with Phoenix Contact.

ERR_AVR_TIMEOUT**009F_{hex}**

- Meaning:** An acknowledge message bit was set when reading a message to indicate to the communication partner that a message has been processed and the mailbox is free again. This bit must be reset by the communication partner to indicate that it has recognized that the mailbox is free again. If this reset does not take place within a set time, an error message is generated.
- Cause:** Invalid node called, e.g.,:
An attempt was made to address a coprocessor board (COP), which is faulty or not present.
- Remedy:** Please get in touch with Phoenix Contact.

3.15.4 Error Messages When Transmitting Process Data

These errors only occur when accessing the data interface (DTI).

ERR_AREA_EXCDED

0096_{hex}

- Meaning:** Access exceeds the upper limit of the selected data area.
- Cause 1:** The data record to be read or written is too large. The function can read a maximum of 4 kbytes in one call.
- Remedy:** Only read or write data records with a maximum size of 4 kbytes.
- Cause 2:** The upper area limit (4 kbytes over the start of the device area) has been exceeded.
- Remedy:** Make sure that the total of address offset, relative address, and data length to be read does not exceed the upper area limit.

ERR_INVLD_DATA_CONS

0097_{hex}

- Cause:** An invalid value was entered for data consistency (1, 2, 4 or 8 bytes).
- Remedy:** Specify a permissible data consistency with one of the following constants:
- | | |
|----------------|--|
| DTI_DATA_BYTE | : Byte data consistency (1 byte) |
| DTI_DATA_WORD | : Word data consistency (2 bytes) |
| DTI_DATA_LWORD | : Double word data consistency (4 bytes) |
| DTI_DATA_64BIT | : 64-bit data consistency (8 bytes) |

ERR_PLUG_PLAY

00A9_{hex}

- Cause:** An attempt was made to gain write access to process data in plug & play mode. This is not permitted for security reasons.
- Remedy:** Deactivate plug & play mode using the "Set_Value" command with the value "0" or switch to read access.

ERR_STATE_CONFLICT

0100_{hex}

- Cause:** A service was called, which is not permitted in this operating mode.
- Remedy:** Switch to an operating mode in which the desired call can be executed.

ERR_INVLD_CONN_TYPE **0101_{hex}**

Cause: A service was called, which cannot be executed via the selected connection.

Remedy: Select a connection type via which the service can be executed.

ERR_ACTIVE_PD_CHK **0102_{hex}**

Cause: IN process data monitoring failed to activate.

ERR_DATA_SIZE **0103_{hex}**

Cause: The data volume to be transmitted exceeds the maximum permissible size.

Remedy: Transmit the data in several cycles.

ERR_OPT_INVLD_CMD **0200_{hex}**

Cause: An attempt was made to execute an unknown (invalid) command.

Remedy: Select a valid command.

ERR_OPT_INVLD_PARAM **0201_{hex}**

Cause: An attempt was made to execute a command with unknown (invalid) parameters.

Remedy: Enter permitted parameters.

ERR_ETH_RCV_TIMEOUT **1001_{hex}**

Cause: The time limit for receiving a data telegram was exceeded.

Remedy: The Ethernet connection was interrupted or an incorrect IP address was entered.
Increase the timeout time.

ERR_IBSETH_OPEN **1010_{hex}**

Cause: The IBSETHA file cannot be opened.

Remedy: The IBSETHA file does not exist or is in the wrong directory.

ERR_IBSETH_READ**1013_{hex}**

Cause: The IBSETHA file cannot be read.

Remedy: The file exists but cannot be read. You may not have read access.

ERR_IBSETH_NAME**1014_{hex}**

Cause: The device name cannot be found in the file.

Remedy: The name, which was transferred to the DDI_DEVOPEN_NODE () function, is not in the IBSETHA file.

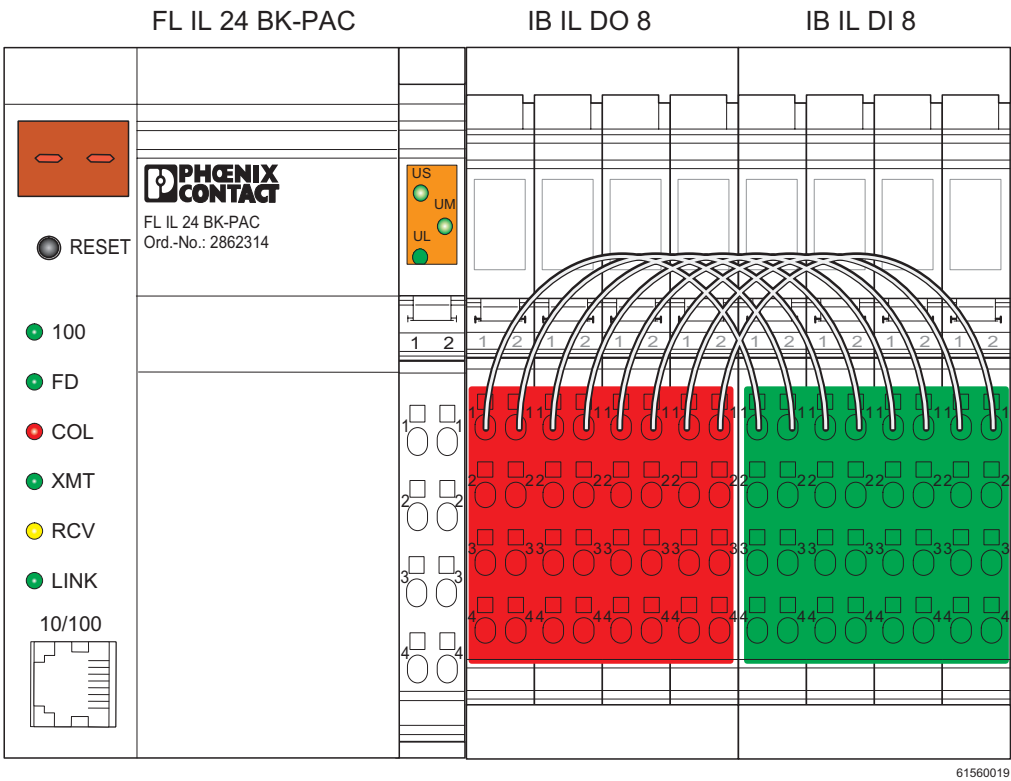
ERR_IBSETH_INTERNET**1016_{hex}**

Cause: The system cannot read the computer name/host address.

Remedy: The IP address entered in the IBSETHA file is incorrect or the symbolic name cannot be found in the host file.

3.16 Example Program

The following diagram illustrates the structure of the station to which the example program refers. One module with 8 digital outputs (IB IL DO 8, Order No. 27 26 26 9) and one module with 8 digital inputs (IB IL DI 8, Order No. 27 26 22 7) are connected to the FL IL 24 BK(-PAC). The inputs are individually jumpered to the outputs. The ground potential is created by the internal potential jumper.



61560019

Figure 3-14 Structure of the station for the example program

3.16.1 Demo Structure Startup

The user is first prompted to specify the bus coupler on which the program is to be executed. This is specified using the registry entries (position 01 to 99). The entry must always be two digits.

Function:

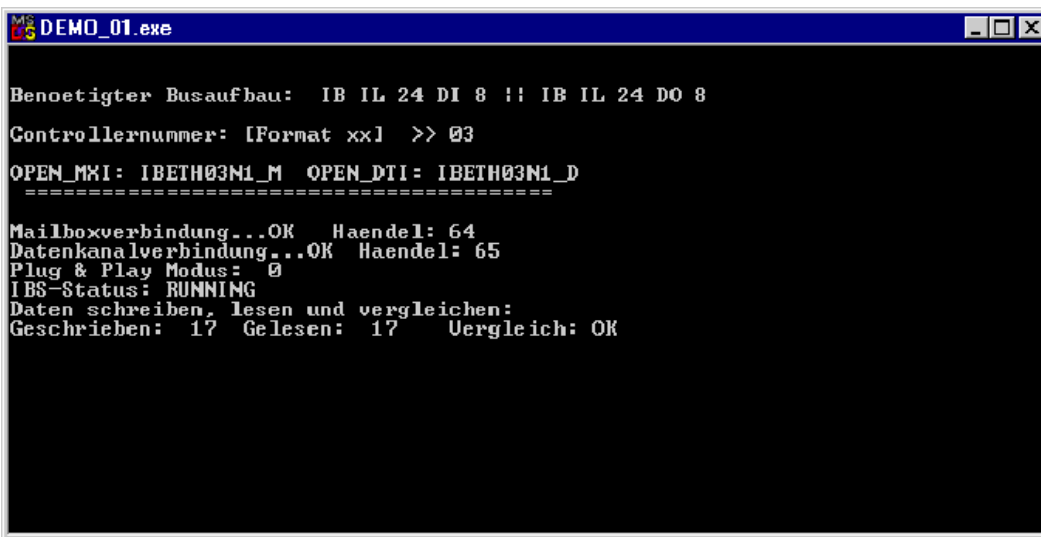
First, the status of plug & play mode is read. If P&P mode is activated (value = 1) the program is terminated with the error message 00A9_{hex} (ERR_PLUG_PLAY), because process data cannot be written in P&P mode for security reasons.

A check then determines whether the local bus in the station is running. If not, the program is also terminated.

If both conditions are met, data items 1 to 255 are output from the output module. Jumping between the outputs and inputs enables the output data to be read in again. The read data is compared with the output data. If they are the same, "Comparison: OK" is output and if they are different, "Comparison: FAILED" is output.

After the process data item "255" has been output, the program is terminated after a 3-second waiting time.

The following figure is a screenshot of the program.



```
MS-DOS DEMO_01.exe
Benoetigter Busaufbau: IB IL 24 DI 8 :: IB IL 24 DO 8
Controllernummer: [Format xx] >> 03
OPEN_MX1: IBETH03M1_M OPEN_DTI: IBETH03M1_D
=====
Mailboxverbindung...OK Haendel: 64
Datenkanalverbindung...OK Haendel: 65
Plug & Play Modus: 0
IBS-Status: RUNNING
Daten schreiben, lesen und vergleichen:
Geschrieben: 1? Gelesen: 1? Vergleich: OK
```

Figure 3-15 Screenshot of the example program

3.16.2 Example Program Source Code

```

/*=====*/
/* INCLUDE FILES AND CONSTANT DEFINITION */
/*=====*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

/*****
 * Include files for the CLIENT library Windows version
 *****/
#include "ethwin32.h"

#define MAX_MSG_LENGTH 100
#define MXI_RCV_TIMEOUT 9

/*****
 * GLOBAL VARIABLES */
/*****
char OPEN_MXI[20] = "IBETH";
char OPEN_DTI[20] = "IBETH";

IBDDIHND mxiHnd, dtiHnd;
          T_DDI_MXI_ACCESS mxiAcc;
T_DDI_DTI_ACCESS dtiAcc;
T_DDI_DTI_ACCESS readAcc;

/*****

/* CreateConnection FUNCTION */
/*
/* Parameters:    NONE */
/* Return value:  INTEGER (0 for OK, 111 for error) */
/*
/*****

int CreateConnection(void)
{
IBDDIRET ret;
/*Mailbox connection*/
ret = DDI_DevOpenNode(OPEN_MXI,DDI_RW,&mxiHnd);
if(ret != ERR_OK)
{
printf("\nError when creating mailbox connection. Error code: %d", ret);
printf("\n TEST ABORTED");

```

```

        fflush(stdout);
        return 111;
    }
else
{
    printf("\nMailbox connection...OK    Handle: %d",mxiHnd);

}
/*Data channel connection*/
ret = DDI_DevOpenNode(OPEN_DTI,DDI_RW,&dtiHnd);
if(ret != ERR_OK)
{
    printf("\nError when creating data channel connection. Error code: %d", ret);
    printf("\n TEST ABORTED");
    fflush(stdout);
    return 111;
}
else
{
    printf("\nData channel connection...OK    Handle: %d",dtiHnd);
}

return 0;
} /
*****/
/* DeleteConnection FUNCTION */
/*
/* Parameters:    NONE */
/* Return value: INTEGER (0 for OK,  111 for error) */
/*
*****/

int DeleteConnection(void)
{
    IBDDIRET ret;
    /* Close mailbox channel */
    ret = DDI_DevCloseNode(mxiHnd);
    if(ret != ERR_OK)
    {
        printf("\nError when closing mailbox channel.  Error code: %d",ret);
        fflush(stdout);
        return 111;
    }
else
{
    printf("\nClose mailbox channel...OK");
}

```

```
    }

    /* Close data channel */
    ret = DDI_DevCloseNode(dtiHnd);
    if(ret != ERR_OK)
    {
        printf("\nError when closing data channel. Error code: %d",ret);
        fflush(stdout);
        return 111;
    }
    else
    {
        printf("\nClose data channel...OK");
    }

return 0;
}

/
*=====*/
/*=====*/
/* M A I N */
/*=====*/
/*=====*/

int main(void)
{

    IBDDIRET locRet = 0;
    char Number[2];
    USIGN8 locMsgBlk[MAX_MSG_LENGTH];
    USIGN8 locReadBlk[MAX_MSG_LENGTH];
    int loci,i;
    USIGN16 ReadData = 0;
    USIGN16 anzahl = 255;
    USIGN16 PlugPlayModus = 111;
    T_IBS_DIAG infoPtr;
    time_t ltime;
    time_t starttime;

    USIGN16 Read1, Read2, Read3, Read4;

    // Display bus configuration
    printf("\n\n Required bus configuration:  IB IL 24 DI 8 || IB IL 24 DO 8\n");

    // Entry of the controller number
    printf("\nController number: [Format xx]  >> ");
```

```

scanf ("%2s",Number);
strcat(OPEN_MXI,Number);
strcat(OPEN_DTI,Number);
strcat(OPEN_MXI,"N1_M");
strcat(OPEN_DTI,"N1_D");
printf("\nOPEN_MXI: %s OPEN_DTI: %s",OPEN_MXI,OPEN_DTI);
printf("\n ===== \n");

// Create connections (DTI and MXI channels) to FL IL 24 BK-PAC
locRet = CreateConnection();

if(locRet != 0){
    printf("\nNo DTI/MXI connection -> Test aborted");
    exit(0);
}

Sleep(500);

// Read plug & play mode
mxiAcc.msgLength = 8;
mxiAcc.msgBlk = locMsgBlk;

IB_SetCmdCode (locMsgBlk, 0x0351);
IB_SetParaCnt (locMsgBlk, 0x0002);
IB_SetParaN (locMsgBlk, 0x01,0x0001);
IB_SetParaN (locMsgBlk, 0x02,0x2240);

locRet = DDI_MXI_SndMessage (mxiHnd, &mxiAcc);
if (locRet != ERR_OK)
{
    printf(" FAIL Error code %x", locRet);
}
// Get service confirmation
mxiAcc.msgLength = 128;
time(&starttime);
locRet = 555;
do
{
    locRet = DDI_MXI_RcvMessage (mxiHnd, &mxiAcc);
    time(&ltime);
}
while (((ltime - starttime) < MXI_RCV_TIMEOUT) && (locRet != ERR_OK));

if (locRet != ERR_OK)
{
    printf("\n\n Incorrect confirmation received, Error code 0x%04X", locRet);
}

```

```
else
{
    PlugPlayModus = IB_GetParaN(locMsgBlk, 0x04);
    printf("\nPlug & Play mode:  %d",PlugPlayModus);
}

// If plug & play mode is active, no data can be written
// -> End of test
if(PlugPlayModus != 0) {
    printf("\nPlug & play mode is active -> End of test\n");
    exit(0);
}

//Read IBS status
locRet = GetIBSDiagnostic(dtiHnd, &infoPtr);
if(locRet != ERR_OK)
{
    printf("\nError when reading INTERBUS status.  Error code: 0x%04X",locRet);
}
else
{
    if(infoPtr.state == 0x00E0) {
        printf("\nIBS status: RUNNING");
    } else {
        printf("\nIBS status: 0x%04X",infoPtr.state);
    }
}

// Reading and writing only permitted when the bus is running
if(infoPtr.state != 0x00E0) {
    printf("\nIBS not in RUN state. -> Abort");
    exit(0);
}

// Write zero to the DI8 module
loci = 1;
printf("\nWrite, read, and compare data:  \n");

// Set buffer to ZERO
dtiAcc.length = MAX_MSG_LENGTH;
dtiAcc.address = 0;
dtiAcc.dataCons = DTI_DATA_WORD;  // Specify data consistency, word consistency
here
dtiAcc.data = locMsgBlk;

for(i = 0;i < MAX_MSG_LENGTH;i++)
```

```

    {
        locMsgBlk[i]=0;
    }

locRet = DDI_DTI_WriteData(dtiHnd,&dtiAcc);

if(locRet != ERR_OK){
    printf("\nError when resetting buffer. Error code: 0x%04X",locRet);
}

Sleep(100);

//Loop for reading and writing 255 data items
do
{
    //Write data
    dtiAcc.length = MAX_MSG_LENGTH;
    dtiAcc.address = 0;
    dtiAcc.dataCons = DTI_DATA_WORD; //Specify data consistency
    dtiAcc.data = locMsgBlk;

    //DO8 is the first DO module
    IB_PD_SetDataN(locMsgBlk,0,loci);

    locRet = DDI_DTI_WriteData(dtiHnd,&dtiAcc);

    if(locRet != ERR_OK){
        printf("\nError when writing data. Error code: 0x%04X",locRet);
    }

    Sleep(500);

    // Read data from module 1 (DI8)
    readAcc.length = MAX_MSG_LENGTH;
    readAcc.address = 0;
    readAcc.data = locReadBlk;

    locRet = DDI_DTI_ReadData(dtiHnd,&readAcc);

    if(locRet != 0){
        printf("\nError when reading data. Error code: 0x%04X", locRet);
    }

    ReadData = IB_PD_GetDataN(locReadBlk,0x00);
    if (ReadData == loci) {
        printf("\rWritten: %3d  Read: %3d    Comparison: OK          ",loci,
ReadData);

```

```
    }
    else {
        printf("\rWritten: %3d  Read: %3d    Comparison: FAILED", loci, ReadData);
    }

    loci++;

}
while(loci < 256);

Sleep(500);

// Close channels to FL IL 24 BK-PAC again
locRet = DeleteConnection();

printf("\nEND\n");

Sleep(3000);

return 0;

}
```

This section provides information about

- firmware functions

Firmware Services.....	4-3
4.1 Overview.....	4-3
4.1.1 Services Available in Both Operating Modes.....	4-3
4.1.2 Services Available Only in Expert Mode	4-4
4.2 Notes on Service Descriptions	4-4
4.2.1 "Name_of_the_Service" Service.....	4-5
4.3 Services for Parameterizing the Controller Board	4-7
4.3.1 "Control_Parameterization" Service.....	4-7
4.3.2 "Set_Value" Service.....	4-9
4.3.3 "Read_Value" Service.....	4-11
4.3.4 "Initiate_Load_Configuration" Service	4-13
4.3.5 "Load_Configuration" Service	4-15
4.3.6 "Terminate_Load_Configuration" Service.....	4-18
4.3.7 "Read_Configuration" Service	4-20
4.3.8 "Complete_Read_Configuration" Service	4-26
4.3.9 "Delete_Configuration" Service.....	4-29
4.3.10 "Create_Configuration" Service	4-30
4.3.11 "Activate_Configuration" Service	4-32
4.3.12 "Control_Device_Function" Service	4-34
4.3.13 "Reset_Controller_Board" Service.....	4-36
4.4 Services for Direct INTERBUS Access	4-38
4.4.1 "Start_Data_Transfer" Service.....	4-38
4.4.2 "Alarm_Stop" Service.....	4-40
4.5 Diagnostic Services	4-41
4.5.1 "Get_Error_Info" Service.....	4-41
4.5.2 "Get_Version_Info" Service	4-47
4.6 Error Messages for Firmware Services	4-50
4.6.1 Overview	4-50
4.6.2 Positive Messages.....	4-51
4.6.3 Error Messages	4-51

4 Firmware Services

As it is not necessary to use each firmware service in both operating modes, the following table indicates the assignment of the services to the operating modes. Not using the services as specified in the table may cause the firmware to behave as follows:

- The service is not permitted in this mode and is rejected with a negative acknowledgment
- The service is executed and terminated with a positive acknowledgment, the effect of this service is removed by the firmware.



Please ensure that only one of the two modes (expert or P&P) is active.

4.1 Overview

4.1.1 Services Available in Both Operating Modes

Table 4-1 Services available in both operating modes

Code	Services	Page
0309 _{hex}	Read_Configuration	4-20
030B _{hex}	Complete_Read_Configuration	4-26
0316 _{hex}	Get_Error_Info	4-41
032A _{hex}	Get_Version_Info	4-47
0351 _{hex}	Read_Value	4-11
0714 _{hex}	Control_Device_Function	4-34
0750 _{hex}	Set_Value	4-9
0956 _{hex}	Reset_Controller_Board	4-36

4.1.2 Services Available Only in Expert Mode

Table 4-2 Services available only in expert mode

Code	Services	Page
0306 _{hex}	Initiate_Load_Configuration	4-13
0307 _{hex}	Load_Configuration	4-15
0308 _{hex}	Terminate_Load_Configuration	4-18
030C _{hex}	Delete_Configuration	4-29
030E _{hex}	Control_Parameterization	4-7
0701 _{hex}	Start_Data_Transfer	4-38
0710 _{hex}	Create_Configuration	4-30
0711 _{hex}	Activate_Configuration	4-32
1303 _{hex}	Alarm_Stop	4-40

4.2 Notes on Service Descriptions

Using the services

The use of a service involves sending a service request and evaluating the service confirmation.

The codes of a service request and the subsequent service confirmation only differ in binary notation in bit 15. Bit 15 of a service confirmation is always set.

Thus, in hexadecimal notation, the code of a service confirmation is always 8000_{hex} higher than the code of the service request which it follows.

Example

"Start_Data_Transfer"

Request:

"Start_Data_Transfer_Request" 0701_{hex}

Confirmation:

"Start_Data_Transfer_Confirmation" 8701_{hex} = 0701_{hex} + 8000_{hex}

– Parameter *Result* = 0000_{hex} ⇒ Service executed successfully

– Parameter *Result* ≠ 0000_{hex} ⇒ Error during service execution

The service confirmation indicates the successful execution of a service via a positive message and provides data, if requested. The service confirmation indicates the error that occurred during service execution via a negative message.

The *Result* parameter of the service confirmation shows if the service was executed successfully (*Result* parameter = 0000_{hex}), or if an error occurred (*Result* parameter ≠ 0000_{hex} describes the error cause).

Structure of a service description

A service request/confirmation consists of a block of data words. The parameters that are contained in this block are given in hexadecimal (_{hex}) or binary (_{bin}) notation.

The structure of all service descriptions is as follows:

4.2.1 "Name_of_the_Service" Service

Task: Describes the functions of the service.

Prerequisite: All conditions, which must be met before a service is called to enable successful processing.

Syntax: Name_of_the_Service_Request Code_{hex}

Word 1	Code
Word 2	Parameter_Count
Word 3	Parameter
Word 4	Parameter
Word 5	Parameter
...	...
	Parameter

Bit | 15 0 |

Key: Code: 0xxx_{hex} Command code of the service request (hexadecimal notation)

Parameter_Count: Number of subsequent words

0000_{hex} If the service request does not have parameters.

xxxx_{hex} Otherwise, length of the parameter data record (number of parameter words).

Parameter: Parameters are described individually. Parameters that are organized byte by byte are separated by a vertical line. If a parameter extends over several data words, this is indicated by a line with three dots.

Parameter blocks: Parameter blocks are marked in bold outline. The individual parameters are described in the following section.

Syntax:

Name_of_the_Service_Confirmation

Code_{hex}

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit

15 0

Key:

Code:

8xxx_{hex}

Message code of the service confirmation

Parameter Count:

Number of subsequent words
with a positive message:

XXXX_{hex}

Number of parameter words that are transferred with a positive message

with a negative message:

XXXX_{hex}

Number of parameter words that are transferred with a negative message

Result:

Result of the service processing

0000_{hex}

indicates a positive message.

The controller board has executed the service successfully.

xxxx_{hex}

indicates a negative message.

The controller board could not execute the service successfully. The *Result* parameter indicates why the service could not be executed.

Add Error Info:

Additional information on the error cause

4.3 Services for Parameterizing the Controller Board

4.3.1 "Control_Parameterization" Service

Task: This service initiates or terminates the parameterization phase. This is necessary in order to ensure a defined startup behavior for the Inline system. During the parameterization phase, for example, the validity of read objects is not ensured. Once the parameterization phase has been terminated, the *MPM_Node_Parameterization_Ready* bit is set in the coupling memory. This means that during startup the host system (computer/PLC) can recognize when the parameterization sequence that is stored on the memory card has been successfully processed.

Syntax:

Control_Parameterization_Request

030E_{hex}

Word 1

Word 2

Word 3

Code
Parameter_Count
Control_Code

Bit

| 15 0 |

Key:

Code:

030E_{hex}

Command code of the service request

Parameter_Count:

Number of subsequent words

0001_{hex}

1 parameter word

Control_Code:

Function of the service

0001_{hex}

initiates the parameterization phase.

0000_{hex}

terminates the parameterization phase.

Syntax: **Control_Parameterization_Confirmation** **830E_{hex}**

Positive message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Negative message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15 0
-----	------------

Key:	Code:	830E _{hex}	Message code of the service confirmation
	Parameter_Count:		Number of subsequent words with a positive message: 0001 _{hex} 1 parameter word with a negative message: 0002 _{hex} 2 parameter words
	Result:		Result of the service processing 0000 _{hex} indicates a positive message. The controller board has executed the service successfully. xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
	Add_Error_Info:		Additional information on the error cause

4.3.2 "Set_Value" Service

Task:

This service assigns new values to INTERBUS system parameters (variables). A new value is only accepted if no error was detected when the value range was checked.

The following system parameters are defined:

Table 4-3 System parameters

Variable ID	System Parameter	Value/Comment
0104 _{hex}	Diagnostic status register	Read only
0105 _{hex}	Diagnostic parameter register	Read only
2216 _{hex}	Current PD cycle time	Read only
2240 _{hex}	Plug & play mode	0: Plug & play mode inactive 1: Plug & play mode active
2275 _{hex}	Expert mode	0: Expert mode inactive 1: Expert mode active
2277 _{hex}	Fault response mode	1: Fault reset mode 2: Standard fault mode 0: Hold last state mode
2293 _{hex}	Process data watchdog timeout	0: Watchdog inactive 200 - 65000: Timeout time in ms

Table 4-4 Available fault response modes

Fault Response Mode	Value	Function
Reset fault mode (default)	1	The digital outputs are set to "0". The analog outputs are set to a value (default = "0") programmed by the user.
Standard fault mode	0	All outputs are set to "0".
Hold last state mode	2	All outputs hold the last value.

Syntax:
Set_Value_Request
0750_{hex}

Word 1

Code

Word 2

Parameter_Count

Word 3	Variable_Count		
Word 4	Variable_ID		1. parameter
Word 5	Value		
Bit	15 0		

Key:	Code:	0750 _{hex}	Command code of the service request
	Parameter_Count:	Number of subsequent words, 0x0003	
	Variable_Count:	Number of system parameters to which new values are to be assigned, 0x0001	
	Variable_ID:	ID of the system parameter to which new values are to be assigned (see Table 4-3), 2240 _{hex}	
	Value:	New value of the system parameter, 0 or 1	

Syntax:

Set_Value_Confirmation

8750_{hex}

Positive message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Negative message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info
Bit	15 0

Key:	Code:	8750 _{hex}	Message code of the service confirmation
	Parameter_Count:	Number of subsequent words	
		with a positive message:	
		0001 _{hex}	1 parameter word
		with a negative message:	
Result:		0002 _{hex}	2 parameter words
		Result of the service processing	
		0000 _{hex}	indicates a positive message.
		The controller board has executed the service successfully.	
		xxxx _{hex}	indicates a negative message.
		The controller board could not execute the	

service successfully. The *Result* parameter indicates why the service could not be executed.

Add_Error_Info: Additional information on the error cause

4.3.3 "Read_Value" Service

Task: This service can be used to read INTERBUS system parameters (variables).



For a list of defined system parameters (variables), please refer to the description of the "Set_Value" service (Table 4-3 on page 4-9).

Syntax: Read_Value_Request 0351_{hex}

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Variable_Count	
Word 4	Variable_ID	1. parameter
Bit	15 0	

Key: Code: 0351_{hex} Command code of the service request
Parameter_Count: Number of subsequent words, 0x002
Variable_Count: Number of system parameters to be read, 0x0001
Variable_ID: ID of the system parameter to be read, 0x2240
0x2275

Syntax:	Read_Value_Confirmation	8351 _{hex}
	Positive message	
Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	Variable_Count	
Word 5	Variable_ID	1. system
Word 6	Value	parameter
	Negative message	
Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	Add_Error_Info	
Bit	15 0	
Key:	Code:	8351 _{hex} Message code of the service confirmation
	Parameter_Count:	Number of subsequent words with a positive message: 0004 _{hex} with a negative message: 0002 _{hex} 2 parameter words
	Result:	Result of the service processing 0000 _{hex} indicates a positive message. The controller board executed the service successfully. xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
	Variable_Count:	Number of read system parameters, 0x0001
	Variable_ID:	ID of the read system parameter
	Value:	Value of the system parameter
	Add_Error_Info:	Additional information on the error cause

4.3.4 "Initiate_Load_Configuration" Service

Task:

The "Initiate_Load_Configuration" service prepares the controller board to transmit a configuration to the INTERBUSmaster using the following services:

- "Load_Configuration" (0307_{hex}) or
- "Complete_Load_Configuration" (030A_{hex}).

To transmit a new configuration frame (*New_Config* parameter = 0001_{hex}), specify the *Frame_Reference* and *Device_Count* parameters (total number of devices).

Prerequisite:

The parameterization phase must have been initiated with the "Control_Parameterization" (030E_{hex}) service before.

Syntax:
Initiate_Load_Configuration_Request
0306_{hex}

Word 1	Code	
Word 2	Parameter_Count	
Word 3	New_Config	
Word 4	Frame_Reference	
Word 5	Device_Count	
Word 6	Extension_Length	Extension
...	...	Extension

Bit	15	8	7	0
-----	----------	---	---------	---

Key:

Code:	0306 _{hex}	Command code of the service request
Parameter_Count:	Number of subsequent words	
	xxxx _{hex}	= 3 + (<i>Extension_Length</i> + 1) / 2
New_Config:	0001 _{hex}	The configuration frame is created again. An existing configuration frame is overwritten.
	0000 _{hex}	updates the existing configuration frame.
Frame_Reference:	0x0001 _{hex}	
Device_Count:	Number of INTERBUS devices, which are included in the existing configuration frame or the new one to be loaded.	
Extension_Length:	0x0000	
Extension:	Not supported. Entries are ignored.	

Syntax: **Initiate_Load_Configuration_Confirmation** **8306_{hex}**

Positive message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Negative message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15 0
-----	------------

Key:	Code:	8306 _{hex}	Message code of the service confirmation
	Parameter_Count:		Number of subsequent words with a positive message: 0001 _{hex} 1 parameter word with a negative message: 0002 _{hex} 2 parameter words
	Result:		Result of the service processing 0000 _{hex} indicates a positive message. The controller board has executed the service successfully. xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
	Add_Error_Info:		Additional information on the error cause

4.3.5 "Load_Configuration" Service

Task:

The configuration frame describes each of the specified INTERBUS devices in a separate numbered entry. The order and the numbering of the entries corresponds to the physical bus configuration.

This service transfers the configuration data to the controller board in the form of a list. Use the *Used_Attributes* parameter to determine which attributes the list should contain.



The "Load_Configuration" service does not check the consistency among the attributes but only whether this data is permitted in principle, e.g., whether it is within the value range.

Prerequisite:

Ensure that the controller board has been prepared for transmission with the following services:

- "Control_Parameterization" (030E_{hex})
- "Initiate_Load_Configuration" (0306_{hex})

Syntax:
Load_Configuration_Request
0307_{hex}

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Used_Attributes	
Word 4	Start_Entry_No	
Word 5	Entry_Count	
Word 6	Configuration_Entry	1. device
...	...	
	Configuration_Entry	nth device

Bit	15	0
-----	----------	---

Key:

Code: 0307_{hex} Command code of the service request

Parameter_Count: Number of subsequent parameter words

xxxx_{hex} The value depends on the *Entry_Count* parameter and *Used_Attributes* parameter.

Used_Attributes: Choice of add-on attributes
The parameter is a 16-bit field in which every bit corresponds to an attribute. Set the corresponding bit to 1 on the attribute that you want to transmit (see the "Configuration_Entry" syntax on page 4-16).

Settings for the *Used_Attributes* parameter:

Bit 0 Device number
Bit 1 Device code

Example:
If the entries are only to consist of the device code, enter the value 0002_{hex} for the *Used_Attributes* parameter (bit 1 is set).

Start_Entry_No: Number of the first device for which attributes are to be transmitted

Entry_Count: Number of devices for which attributes are to be transmitted

Configuration_Entry: Attribute values of the individual devices to be transmitted according to their order in the physical bus configuration (see syntax on page 4-16)



According to the following syntax, only enter attributes in the "Configuration_Entry" parameter block that have been enabled with the *Used_Attributes* parameter (disabled attributes are not entered).



When several entries with several attributes are loaded at the same time, first all the attributes of one entry are loaded, then those of the next entry.

Syntax	"Configuration_Entry"		Attribute
Word x	Bus_Segment_No	Position	Device number
Word x+1	Length_Code	ID_Code	Device code
Bit	15 8 7 0		
Attributes:	Bus_Segment_No:	Number of the bus segment where the device is located Value range: 01 _{hex}	
	Position:	Physical location in the bus segment Value range: 00 _{hex} ... 3F _{hex} (63 _{dec}) for an Inline station The <i>Bus_Segment_No</i> and <i>Position</i> parameters together form the device number.	

Length_Code: The length code refers to the address space required by the device in the host.

ID_Code: The ID code indicates the device type. It is printed as *Module Ident* in decimal notation on the modules. The *Length_Code* and *ID_Code* parameters together form the device number.

Syntax:

Load_Configuration_Confirmation

8307_{hex}

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit

| 15 0 |

Key:

Code:

8307_{hex}

Message code of the service confirmation

Parameter_Count:

Number of subsequent words with a positive message:
0001_{hex} Always 1 parameter word
with a negative message:
0002_{hex} Always 2 parameter words

Result:

Result of the service processing
0000_{hex} indicates a positive message.
 The controller board has executed the service successfully.
xxxx_{hex} indicates a negative message.
 The controller board could not execute the service successfully. The *Result* parameter indicates why the service could not be executed.

Add_Error_Info:

Additional information on the error cause

4.3.6 "Terminate_Load_Configuration" Service

Task: This service terminates the loading of the configuration data in segments. The service also checks the loaded configuration data for permissibility and consistency. If no error is detected, the controller board stores the data in the configuration directory under the *Frame_Reference* given in the "Initiate_Load_Configuration" (0306_{hex}) service. If an error is detected, the service is acknowledged with a negative confirmation.

Remark: The *Default_Parameter* parameter can also be used to indicate whether the process data channel (PD channel) and/or the PCP channel are to be parameterized according to the loaded configuration frame. In this case the firmware automatically creates the process data reference list ("physical addressing") and/or a communication relationship list (CRL).



The "Terminate_Load_Configuration" service does not activate the newly loaded configuration immediately. It is only activated with the "Activate_Configuration" service (0711_{hex}).

Syntax: **Terminate_Load_Configuration_Request** **0308_{hex}**

Word 1
Word 2
Word 3
Bit

Code
Parameter_Count
Default_Parameter
15 0

Key:

Code:	0308 _{hex}	Command code of the service request
Parameter_Count:		Number of subsequent words
	0001 _{hex}	1 parameter word
Default_Parameter:		Indicates whether a default parameterization of the PCP and/or PD channel is to be carried out for the loaded configuration:
	0000 _{hex}	No automatic parameterization
	0001 _{hex}	Automatic parameterization of the process data channel through the creation of the process data reference list
	0002 _{hex}	Automatic parameterization of the PCP channel through the creation of the communication relationship list
	0003 _{hex}	Automatic parameterization of the process data and PCP channel

Syntax: **Terminate_Load_Configuration_Confirmation** 8308_{hex}

Positive message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Negative message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit | 15 0 |

Key:

Code:	8308 _{hex}	Message code of the service confirmation
Parameter_Count:		Number of subsequent words with a positive message: 0001 _{hex} 1 parameter word with a negative message: 0002 _{hex} 2 parameter words
Result:		Result of the service processing 0000 _{hex} indicates a positive message. The controller board has executed the service successfully. xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
Add_Error_Info:		Additional information on the error cause

4.3.7 "Read_Configuration" Service

Task: This service reads various entries of the configuration directory depending on the *Frame_Reference* and *Start_Entry_No* parameters.

Frame_Reference	Start_Entry_No	Entries Read by the Service
0001 _{hex}	0000 _{hex}	Header information of the configuration frame (CFG_Header) selected with the <i>Frame_Reference</i> parameter.
0001 _{hex}	> 0000 _{hex}	Entries of the configuration frame (CFG_Entry) selected with the <i>Frame_Reference</i> parameter. Either the entire configuration frame or only one part, e.g., a single INTERBUS device description, can be read.

Syntax: Read_Configuration_Request 0309_{hex}

Word 1	Code
Word 2	Parameter_Count
Word 3	Frame_Reference
Word 4	Used_Attributes
Word 5	Start_Entry_No
Word 6	Entry_Count

Bit	15 0
-----	------------

Key:

Code:	0309 _{hex}	Command code of the service request
Parameter_Count:		Number of subsequent words
	0004 _{hex}	4 parameter words
Frame_Reference:		Number of the configuration frame
	0001 _{hex}	reads the reference configuration
	0002 _{hex}	reads the physical bus configuration
Only relevant if <i>Frame_Reference</i> > 0000 _{hex}	Used_Attributes:	Attributes to be read
		The parameter is a 16-bit field in which every bit corresponds to an attribute. Set the corresponding bit to 1 on the attributes to be read.
		Settings for the <i>Used_Attributes</i> parameter:
	Bit 0	Device number
	Bit 1	Device code

Start_Entry_No:	Position of the first entry
	0000 _{hex} only reads the header information for the configuration frame.
	xxxx _{hex} reads the entries from the configuration directory from this number onwards
Entry_Count:	Number of entries to be read

The positive message transmits the requested entries from the configuration directory. Depending on the *Frame_Reference* and *Start_Entry_No* parameters in the service request, it has one of the following three structures.

Syntax**Read_Configuration_Confirmation****8309_{hex}**

1. structure

Positive message during service request with:

- Frame_Reference = 0000_{hex}
- Start_Entry_No Not relevant (= 0000_{hex})

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	More_Follows	
Word 5	Frame_Reference	= 0000 _{hex}
Word 6	Current_Configuration	
Word 7	Configuration_Count	
Word 8	Frame_Reference 1	

2. structure

Positive message during service request with:

- Frame_Reference > 0000_{hex}
- Start_Entry_No = 0000_{hex}

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	More_Follows	
Word 5	Frame_Reference	> 0000 _{hex}
Word 6	Used_Attributes	Not relevant
Word 7	Start_Entry_No	= 0000 _{hex}
Word 8	Frame_Device_Count	
Word 9	Active_Device_Count	

Word 10	Frame_IO_Bit_Count
Word 11	Active_IO_Bit_Count
Word 12	Frame_PCP_Device_Count
Word 13	Active_PCP_Device_Count
Word 14	Frame_PCP_Word_Count
Word 15	Active_PCP_Word_Count

Bit	15 0
-----	------------

3. structure

Positive message during service request with:

- Frame_Reference > 0000_{hex}
- Start_Entry_No > 0000_{hex}

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	More_Follows	
Word 5	Frame_Reference	
Word 6	Used_Attributes	
Word 7	Start_Entry_No	
Word 8	Entry_Count	
Word 9	Configuration_Entry	1. device
...	...	
	Configuration_Entry	nth device

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15 0
-----	------------

Key: Code: 8309_{hex} Message code of the service confirmation
Parameter_Count: Number of subsequent words
with a positive message and if *Frame_Reference*
= 0000_{hex}:
xxxx_{hex} = 5 + *Configuration_Count*

	with a positive message and if <i>Frame_Reference</i> > 0000 _{hex} and <i>Start_Entry_No</i> = 0000 _{hex} :
	000D _{hex} 12 parameter words
	with a positive message and if <i>Frame_Reference</i> > 0000 _{hex} and <i>Start_Entry_No</i> > 0000 _{hex} :
	xxxx _{hex} The value depends on the number of devices in the configuration frame and the number of enabled attributes.
	with a negative message:
	0002 _{hex} 2 parameter words
Result:	Result of the service processing
	0000 _{hex} indicates a positive message. The service request has been executed successfully. The data is available in the following parameters.
	xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed (see also <i>Add_Error_Info</i>).
Add_Error_Info:	Additional information on the error cause
More_Follows:	0000 _{hex} indicates that all requested entries are contained in the service confirmation.
	0001 _{hex} indicates that the service confirmation does not contain all requested entries as the amount of data is larger than the mailbox (MXI) that is available for the services. Call the service again to read the remaining data.
Frame_Reference:	Number of the read configuration frame. The parameter contains the value which was transmitted by the service request.
Current_Configuration:	Number of the currently activated configuration frame
Configuration_Count:	Number of configured configuration frames
Frame_Reference x:	Numbers of all stored configuration frames in ascending order
Frame_Device_Count:	Number of configured INTERBUS devices in the selected configuration frame
Active_Device_Count:	Number of active INTERBUS devices in the selected configuration frame
Frame_IO_Bit_Count:	Number of configured I/O bits in the selected configuration frame

Active_IO_Bit_Count:	Number of active I/O bits in the selected configuration frame
Frame_PCP_Device_Count:	Number of configured PCP devices in the selected configuration frame
Active_PCP_Device_Count:	Number of active PCP devices in the selected configuration frame
Frame_PCP_Word_Count:	Number of configured PCP words in the selected configuration frame
Active_PCP_Word_Count:	Number of active PCP words in the selected configuration frame
Used_Attributes:	Read attributes This parameter contains the value which was transmitted by the service request.
Start_Entry_No:	Position of the first entry or 0000 _{hex} if only the header information was read
Entry_Count:	Number of entries that are transmitted by this service confirmation. The <i>More_Follows</i> parameter indicates if there are other entries.
Configuration_Entry:	Selected entries in the order of the physical bus configuration The attributes contained in every entry are enabled in the service request by the <i>Used_Attributes</i> parameter (see the "Configuration_Entry" syntax on page 4-25).



A configuration entry for a device does not have to contain all attributes. If an attribute is not enabled in the service request by the *Used_Attributes* parameter, the configuration entry is reduced by the relevant data words.

In the following, the structure of a configuration entry is shown where **all** attributes are enabled.

Syntax	"Configuration_Entry"		Attribute:
Word x	Bus_Segment_No	Position	Device number
Word x+1	Length_Code	ID_Code	Device code
Bit	15 8 7 0		
Key:	Attribute: Device Number		
	Bus_Segment_No:	Number of the bus segment where the INTERBUS device is located. Value: 00 _{hex}	
	Position:	Physical location in the bus segment Value range: 00 _{hex} to 40 _{hex} for an Inline station	
	Attribute: Device Code		
	Length_Code:	The length code refers to the address space required by the INTERBUS device in the host.	
	ID_Code:	The ID code describes the INTERBUS device function. It is printed as <i>Module Ident</i> in decimal notation on the modules.	

4.3.8 "Complete_Read_Configuration" Service

Task: This service reads entries in the configuration directory in the form of one or more columns which have been selected with the *Used_Attributes* parameter. It is specially adapted to the PLC programming requirements.

Remark: This service can be understood as a meta service for the "Read_Configuration" service (0309_{hex}). The *Start_Entry_No* parameter does not need to be specified, since this service reads all entries of the configuration frame (*Start_Entry_No* = "1").

Syntax: **Complete_Read_Configuration_Request** **030B_{hex}**

Word 1	Code
Word 2	Parameter_Count
Word 3	Used_Attributes

Bit | 15 0 |

Key: Code: 030B_{hex} Command code of the service request
Parameter_Count: Number of subsequent words
0001_{hex} Always 1 parameter word
Used_Attributes: The parameter is a 16-bit field in which every bit corresponds to an attribute. Set the corresponding bits to 1 on the attributes to be read.
Settings for the *Used_Attributes* parameter:
Bit 0 Device number
Bit 1 Device code

Syntax:**Complete_Read_Configuration_Confirmation****830B_{hex}**

Positive message

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	More_Follows	
Word 5	Frame_Reference	
Word 6	Used_Attributes	
Word 7	Start_Entry_No	0001 _{hex}
Word 8	Entry_Count	
Word 9	Configuration_Entry	1. device
...	...	
	Configuration_Entry	nth device

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15 0
-----	------------

Key:

Code:	830B _{hex}	Message code of the service confirmation
Parameter_Count:		Number of subsequent words with a positive message:
	xxxx _{hex}	The value depends on the number of entries and the number and type of attributes that were read.
		with a negative message:
	0002 _{hex}	2 parameter words
Result:		Result of the service processing
	0000 _{hex}	indicates a positive message. The controller board has executed the service successfully.
	xxxx _{hex}	indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.

Add_Error_Info:	Additional information on the error cause
More_Follows:	0000_{hex} indicates that all requested entries are contained in the service confirmation. 0001_{hex} indicates that the service confirmation does not contain all requested entries as the amount of data is larger than the mailbox (MXI) that is available for the services. Call the "Read_Configuration" service (0309_{hex}) to read the remaining data.
Frame_Reference:	Number of the active configuration frame
Used_Attributes:	Read attributes This parameter contains the value which was transmitted by the service request.
Start_Entry_No:	Number of the first entry 0001_{hex} With this service all entries are read out, starting with the first entry.
Entry_Count:	Number of entries that are transferred by the service confirmation.
Configuration_Entry:	Entries in the order of the physical bus configuration The attributes contained in every entry are enabled in the service request by the <i>Used_Attributes</i> parameter. For the description of the <i>Configuration_Entry</i> parameters see "Read_Configuration" service (0309_{hex}) on page 4-20.

4.3.9 "Delete_Configuration" Service

Task: This service deletes an inactive configuration frame from the configuration directory.

Syntax: **Delete_Configuration_Request** **030C_{hex}**

Word 1	Code
Word 2	Parameter_Count
Word 3	Frame_Reference

Bit | 15 0 |

Key: Code: 030C_{hex} Command code of the service request
 Parameter_Count: Number of subsequent words
 0001_{hex} 1 parameter word
 Frame_Reference: 0001_{hex}

Syntax: **Delete_Configuration_Confirmation** **830C_{hex}**

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit | 15 0 |

Key: Code: 830C_{hex} Message code of the service confirmation
 Parameter_Count: Number of subsequent words
 with a positive message:
 0001_{hex} 1 parameter word
 with a negative message:
 0002_{hex} 2 parameter words

Result:	Result of the service processing
	0000 _{hex} indicates a positive message. The controller board has executed the service successfully.
	xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
Add_Error_Info:	Additional information on the error cause

4.3.10 "Create_Configuration" Service

Task: This service causes the controller board to automatically generate a configuration frame from the currently connected configuration and to activate it in order to start the bus. After the execution of the service the controller board is in the *Active* state (display: "*PP*" for *plug & play mode*, "*--*" for *expert mode*).

The new configuration frame and the known configuration are stored in the configuration directory under the number specified in the *Frame_Reference* parameter. If there is already a configuration frame under this number, this frame is overwritten. In addition, the controller board generates default process data description lists, a default process data reference list, and a default communication relationship list (CRL) according to the currently connected bus configuration. In the device descriptions the attributes are initialized as follows:

<i>Device_Number:</i>	According to the known configuration
<i>Length_Code:</i>	According to the known configuration
<i>ID_Code:</i>	According to the known configuration
<i>Device_Level:</i>	According to the known configuration
<i>Group_Number:</i>	For all INTERBUS devices FFFF _{hex} (i.e., no group numbers are supported)
<i>Device_State:</i>	All INTERBUS devices are active

Syntax:

Create_Configuration_Request0710_{hex}

Word 1

Word 2

Word 3

Code
Parameter_Count
Frame_Reference

Bit

15 0

Key:

Code:0710_{hex}Command code of the service request

Parameter_Count:	Number of subsequent words
	0001 _{hex} 1 parameter word
Frame_Reference:	0001 _{hex}

Syntax:

Create_Configuration_Confirmation

8710_{hex}

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit | 15 0

Key:

Code:	8710 _{hex}	Message code of the service confirmation
Parameter_Count:	Number of subsequent words with a positive message: 0001 _{hex}	1 parameter word
	with a negative message: 0002 _{hex}	2 parameter words
Result:	Result of the service processing 0000 _{hex}	indicates a positive message. The controller board has executed the service successfully.
	xxxx _{hex}	indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
Add Error Info:	Additional information on the error cause	

4.3.11 "Activate_Configuration" Service

Task: This service enables the controller board to check the configuration data of the configuration frame for

- conformance with the currently connected configuration
- address overlaps

If no errors are detected, the controller board activates this configuration frame (display: *"PP" for plug & play mode, "--" for expert mode*) and runs ID cycles at regular intervals. The number of the configuration frame is indicated to the controller board by the *Frame_Reference* parameter.

Prerequisite: If you want to activate a configuration frame, another configuration frame cannot be active at the same time. The "Deactivate_Configuration" is not supported.

Syntax:

Word 1

Word 2

Word 3

Bit

15 0

Activate_Configuration_Request

Code
Parameter_Count
Frame_Reference

0711_{hex}

Key:

Code:

Parameter_Count:

Frame_Reference:

0711_{hex}

0001_{hex}

0001_{hex}

Command code of the service request

Number of subsequent words

1 parameter word

Syntax: **Activate_Configuration_Confirmation** **8711_{hex}**

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit | 15 0 |

Key:

Code:	8711 _{hex}	Message code of the service confirmation
Parameter_Count:		Number of subsequent words with a positive message:
	0001 _{hex}	1 parameter word
		with a negative message:
	0002 _{hex}	2 parameter words
Result:		Result of the service processing
	0000 _{hex}	indicates a positive message. The controller board has executed the service successfully.
	xxxx _{hex}	indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
Add_Error_Info:		Additional information on the error cause

4.3.12 "Control_Device_Function" Service

Task: This service can be used to send control commands to one or more INTERBUS Inline devices, for example to acknowledge device status errors or set an alarm output.

Syntax: Control_Device_Function_Request 0714_{hex}

Word 1	Code	List of INTERBUS devices
Word 2	Parameter_Count (n)	
Word 3	Device_Function	
Word 4	Entry_Count	
Word 5	Device_No	
Word 6	Device_No	
	...	
Word n+2	Device_No	

Bit | 15 0 |

Key: Code: 0714_{hex} Command code of the service request
Parameter_Count: Number of subsequent words
Device_Function: 0004_{hex}Conf_Dev_Err_All:
Confirming the peripheral faults (PF) of all devices.
Set **Entry_Count = 0000**_{hex}. The list of INTERBUS
devices is not required.
Entry_Count: 0000_{hex} If Device_Function = 0004_{hex}

Syntax: **Control_Device_Function_Confirmation** **8714_{hex}**

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit | 15 0 |

Key:

Code:	8714 _{hex}	Message code of the service confirmation
Parameter_Count:		Number of subsequent words with a positive message:
	0001 _{hex}	1 parameter word
		with a negative message:
	0002 _{hex}	2 parameter words
Result:		Result of the service processing
	0000 _{hex}	indicates a positive message. The controller board has executed the service successfully.
	xxxx _{hex}	indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
Add_Error_Info:		Additional information on the error cause

4.3.13 "Reset_Controller_Board" Service

Task: This service can be used to initiate a controller board reset.

Prerequisite: Before calling this service, ensure that the state of your system permits a controller board reset.

Syntax: **Reset_Controller_Board_Request** **0956_{hex}**

Word 1	Code
Word 2	Parameter_Count
Word 3	Frame_Reference

Bit | 15 0 |

Key: Code: 0956_{hex} Command code of the service request

Parameter_Count: Number of subsequent words

0001_{hex} 1 parameter word

Reset_Type: 0001_{hex} Cold restart

A cold restart is always carried out.

Syntax: **Reset_Controller_Board_Confirmation** **8956_{hex}**

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit | 15 0 |

Key: Code: 8956_{hex} Message code of the service confirmation

Parameter_Count: Number of subsequent words

with a positive message:

0001_{hex} 1 parameter word

with a negative message:

0002_{hex} 2 parameter words

Result: Result of the service processing

	0000 _{hex}	indicates a positive message. The controller board has executed the service successfully.
	xxxx _{hex}	indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
Add_Error_Info:	Additional information on the error cause	

4.4 Services for Direct INTERBUS Access

4.4.1 "Start_Data_Transfer" Service

Task: This service activates the cyclic data traffic on the bus. After the execution of the service, the controller board is in the *Run* state (display: "*PP*" for *plug & play mode*, "*--*" for *expert mode*).

Prerequisite: Before the service is called, the controller board must be in the *Active* state, i.e., a configuration frame has been activated and ID cycles are already being run at regular intervals.

Syntax: **Start_Data_Transfer_Request** **0701_{hex}**

Word 1	Code
Word 2	Parameter_Count

Bit | 15 0 |

Key: Code: 0701_{hex} Command code of the service request
Parameter_Count: Number of subsequent words
0000_{hex} No parameter word

Syntax: **Start_Data_Transfer_Confirmation** 8701_{hex}

Positive message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Negative message	
Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit		15	0	
-----	--	----------	---	--

Key:	Code:	8701 _{hex}	Message code of the service confirmation
	Parameter_Count:		Number of subsequent words with a positive message: 0001 _{hex} 1 parameter word with a negative message: 0002 _{hex} 2 parameter words
	Result:		Result of the service processing 0000 _{hex} indicates a positive message. The controller board has executed the service successfully. xxxx _{hex} indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
	Add_Error_Info:		Additional information on the error cause

4.4.2 "Alarm_Stop" Service

Task: This service triggers a long reset on the bus. Data traffic is stopped. Modules with process data set their outputs to the value 0. The command is executed directly after the current data cycle has been completed. After the execution of the service, the controller board is in the *Ready* state (display: "PP" for plug & play mode, "--" for expert mode).

Syntax: **Alarm_Stop_Request** **1303_{hex}**

Word 1	Code
Word 2	Parameter_Count

Bit	15 0
-----	------------

Key: Code: 1303_{hex} Command code of the service request
Parameter_Count: Number of subsequent words
0000_{hex} No parameter word

Syntax: **Alarm_Stop_Confirmation** **9303_{hex}**

Positive message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15 0
-----	------------

Key: Code: 9303_{hex} Message code of the service confirmation
Parameter_Count: Number of subsequent words
with a positive message:
0001_{hex} 1 parameter word
with a negative message:
0002_{hex} 2 parameter words
Result: Result of the service processing
0000_{hex} indicates a positive message.
The controller board has executed the service successfully.

xxxx_{hex} indicates a negative message.
The controller board could not execute the service successfully. The *Result* parameter indicates why the service could not be executed.

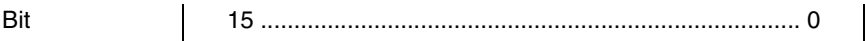
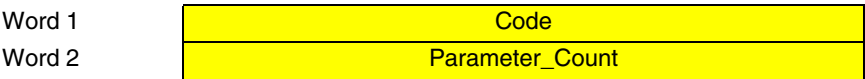
Add_Error_Info: Additional information on the error cause

4.5 Diagnostic Services

4.5.1 "Get_Error_Info" Service

Task: This service can be used to read out the exact error cause and location after a bus error has been indicated. A maximum of ten errors are analyzed.

Syntax: Get_Error_Info_Request 0316_{hex}



Key: Code: 0316_{hex} Command code of the service request
Parameter_Count: Number of subsequent words
0000_{hex} No parameter word

Syntax:

8316_{hex}

Get_Error_Info_Confirmation

Positive message, as long as error localization is still in progress

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	Entry_Count	= 0001 _{hex}
Word 5	Error_Code	= 0BDF _{hex}
Word 6	Add_Error_Info	= FFFF _{hex}

Positive message, if error localization has been completed

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Word 4	Entry_Count	
Word 5	Error_Code	1. error
Word 6	Add_Error_Info	
	Add_Error_Info	

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15	0
-----	----------	---

Key:	Code:	8316 _{hex}	Message code of the service confirmation
	Parameter_Count:		Number of subsequent words
			with positive message (during error localization):
		0004 _{hex}	4 parameter words
			with positive message (after error localization):
		00xx _{hex}	$= 2 + 2 \times \text{Entry_Count}$ (20 words, maximum)
			with a negative message:
		0002 _{hex}	Always 2 parameter words
	Result:		Result of the service processing
		0000 _{hex}	indicates a positive message. The controller board has executed the service successfully.
		xxxx _{hex}	indicates a negative message. The controller board could not execute the service successfully. The <i>Result</i> parameter indicates why the service could not be executed.
	Entry_Count:	0001 _{hex}	
	Error_Code:		Information on the error type
	Add_Error_Info:		with positive message: Error location (<i>Bus segment . Position</i>), if it could be located.
			with negative message: Additional information on the error cause using error codes

Table 4-5 Supported error codes

Code	Error Type	Page
0x0A1C	E_SM_CFG_NUM_OF_DEV_TOO_BIG	4-44
0x0A2D	E_SM_CFG_NUM_OF_PCP_DEV_TOO_BIG	4-44
0x0A2E	E_SM_CFG_IND_ADDR_LIST_TOO_BIG	4-44
0x0B02	E_PNM12_STATE_CONFLICT	4-45

Code	Error Type	Page
0x0BB1	E_PNM12_DEVICE_STATE	4-45
0x0D10	E_PNM12_CONFIG_MISSING_DEVICE	4-45
0x0D20	E_PNM12_CONFIG_MAU_FAIL_DO	4-45
0x0D28	E_PNM12_CONFIG_MAU_FAIL_DI	4-46
0x0D4C	E_PNM12_CONFIG_INVALID_ID	4-46
0x0D80	E_PNM12_CONFIG_MULTI_ERR_OUT	4-46
0x0D9C	E_PNM12_CONFIG_LB_TOO_LONG_OUT	4-46
0xFFFF	CONTROLLER_DEVICE_NUMBER	4-46

Error Code Description**E_SM_CFG_NUM_OF_DEV_TOO_BIG****0A1C_{hex}**

Cause: You exceeded the permitted number of specified or connected INTERBUS devices. The maximum permissible number of INTERBUS Inline devices is 63.

Add_Error_Info: Number of specified or connected INTERBUS devices.

E_SM_CFG_NUM_OF_PCP_DEV_TOO_BIG**0A2D_{hex}**

Meaning: Too many PCP devices.

Cause:

- You connected more than the permitted number of PCP devices.
- You configured more than the permitted number of PCP devices.

Remedy: Reduce the number of connected or configured PCP devices. The maximum permissible number of PCP devices is 8.

E_SM_CFG_IND_ADDR_LIST_TOO_BIG**0A2E_{hex}**

Meaning: The permitted number of internal indirect address list entries was exceeded. You have reached the firmware memory limit.

Cause: You have too many modules that occupy only one byte or one nibble of address space in the data ring.

Remedy:

- Reduce the number of modules occupying only one byte or one nibble of address space. The maximum number of internal permitted indirect address list entries is 384.

Arrange the modules so that the devices that require less than 1 word of address space are next to each other.

E_PNM12_STATE_CONFLICT**0B02_{hex}**

- Cause:**
- It may be that
 - there is an empty configuration frame
 - the first device after the bus coupler is faulty or missing
- Remedy:**
- - Activate a correct configuration frame or
 - Use the first device or an operational device.

E_PNM12_DEVICE_STATE**0BB1_{hex}**

- Meaning:** The specified Inline device indicates a peripheral fault.
- Remedy:** Check the specified Inline device.
- Add_Error_Info:** Device number (Segment . Position) of the Inline device.

E_PNM12_CONFIG_MISSING_DEVICE**0D10_{hex}**

- Meaning:** An Inline device is missing.
- Cause:** A device entered in the active configuration and not marked as switched off is missing from the connected bus configuration.
The active configuration is the quantity of INTERBUS devices connected to the INTERBUS system whose data is within the summation frame during bus cycles.
The active configuration may differ from the connected bus configuration only when physically connected bus segments have been switched off.
- Remedy:** Compare the active configuration with the connected bus configuration, taking any disabled bus segments into account.
- Add_Error_Info:** Error location (Segment . Position).

E_PNM12_CONFIG_MAU_FAIL_DO**0D20_{hex}**

- Meaning:** The Medium Attachment Unit (MAU) firmware component diagnosed an interruption of the data transmission.
- Cause:** Cable break on the data forward path of the incoming bus interface (IN) of the indicated Inline device.
- Remedy:** Check the cables, connectors, and Inline connections for interruptions and repair them, if required.
- Add_Error_Info:** Error location (Segment . Position).

E_PNM12_CONFIG_MAU_FAIL_DI**0D28hex**

- Meaning:** The Medium Attachment Unit (MAU) diagnosed an interruption of the data transmission.
- Cause:** Cable break on the data return path of the incoming bus interface (IN) of the indicated Inline device.
- Remedy:** Check the cables, connectors, and Inline connections for interruptions and repair them, if required.
- Add_Error_Info:** Error location (Segment . Position).

E_PNM12_CONFIG_INVALID_ID**0D4Chex**

- Meaning:** The specified Inline device has an invalid ID code.
- Add_Error_Info:** Error location (Segment . Position).

E_PNM12_CONFIG_MULTI_ERR_OUT**0D80hex**

- Meaning:** Multiple error at the outgoing bus interface (OUT1) of the specified INTERBUS device.
- Cause:** Fault on the bus cable connected to this bus interface, of the following INTERBUS device, or of a device of any subsequent local bus.
- Remedy:** Check this part of the system for:
- Missing or incorrect shielding of the bus cables (connectors)
 - Missing or incorrect grounding/equipotential bonding
 - Poor connections in the connector (loose contact, cold junction)
 - Voltage dips on the communications power for remote bus devices
 - Faulty fiber optic assembly
- Add_Error_Info:** Error location (Segment . Position).

E_PNM12_CONFIG_LB_TOO_LONG_OUT**0D9Chex**

- Meaning:** The local bus connected directly to the controller board consists of more Inline devices than have been entered in the active configuration.
- Remedy:** Check this local bus.
- Add_Error_Info:** Error location (Segment . Position).

CONTROLLER_DEVICE_NUMBER**FFFF**

4.5.2 "Get_Version_Info" Service

Task: This service can be used to read the type, version, manufacturing date, etc. of the hardware and firmware of your controller board.

Syntax: **Get_Version_Info_Request** **032A_{hex}**

Word 1	Code
Word 2	Parameter_Count

Bit | 15 0 |

Key: Code: 032A_{hex} Command code of the service request
 Parameter_Count: Number of subsequent words
 0000_{hex} No parameter word

Syntax: **Get_Version_Info_Confirmation** **832A_{hex}**

Positive message

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Result	
Words 4 + 5	FW_Version (byte 1)	FW_Version (byte 2)
	FW_Version (byte 3)	FW_Version (byte 4)
Words 6 ... 8	FW_State (byte 1)	...
	...	FW_State (byte 6)
Words 9 ... 11	FW_Date (byte 1)	...
	...	FW_Date (byte 6)
Words 12 ... 14	FW_Time (byte 1)	...
	...	FW_Time (byte 6)
Words 15 ... 24	Host_Type (byte 1)	...
	...	Host_Type (byte 20)
Words 25 + 26	Host_Version (byte 1)	Host_Version (byte 2)
	Host_Version (byte 3)	Host_Version (byte 4)
Words 27 ... 29	Host_State (byte 1)	...
	...	Host_State (byte 6)
Words 30 ... 32	Host_Date (byte 1)	...
	...	Host_Date (byte 6)
Words 33 ... 35	Host_Time (byte 1)	...
	...	Host_Time (byte 6)

Words 36 + 37	Start_FW_Version (byte 1)	Start_FW_Version (byte 2)
	Start_FW_Version (byte 3)	Start_FW_Version (byte 4)
Words 38 ... 40	Start_FW_State (byte 1)	...
	...	Start_FW_State (byte 6)
Words 41 ... 43	Start_FW_Date (byte 1)	...
	...	Start_FW_Date (byte 6)
Words 44 ... 46	Start_FW_Time (byte 1)	...
	...	Start_FW_Time (byte 6)
Words 47 ... 50	HW_Art_No (byte 1)	...
	...	HW_Art_No (byte 8)
Words 51 ... 65	HW_Art_Name (byte 1)	...
	...	HW_Art_Name (byte 30)
Words 66 + 67	HW_Motherboard_ID (byte 1)	HW_Motherboard_ID (byte 2)
	HW_Motherboard_ID (byte 2)	HW_Motherboard_ID (byte 4)
Word 68	HW_Version (byte 1)	HW_Version (byte 2)
Words 69 ... 78	HW_Vendor_Name (byte 1)	...
	...	HW_Vendor_Name (byte 20)
Words 79 ... 84	HW_Serial_No (byte 1)	...
	...	HW_Serial_No (byte 12)
Words 85 ... 87	HW_Date (byte 1)	...
	...	HW_Date (byte 6)

Bit	15	0
-----	----------	---

Negative message

Word 1	Code
Word 2	Parameter_Count
Word 3	Result
Word 4	Add_Error_Info

Bit	15	0
-----	----------	---

Key: Code: 832A_{hex} Message code of the service confirmation
Parameter_Count: Number of subsequent words
with a positive message:
0055_{hex} 55 parameter words
with a negative message:

Result: 0002_{hex} 2 parameter words
 Result of the service processing
 0000_{hex} indicates a positive message.
 The controller board has executed the service successfully.
 xxxx_{hex} indicates a negative message.
 The controller board could not execute the service successfully. The *Result* parameter indicates why the service could not be executed.

Add_Error_Info: Additional information on the error cause

Version information for the hardware and firmware. Every byte indicates the ASCII code for a character:

FW_Version: Version of the firmware core (4 bytes)
 (e.g., 33 2E 39 37_{hex} for "Version 3.97")
 FW_State: Firmware status (6 bytes)
 (e.g., 62 65 64 61 00 00_{hex} for "beta" with preliminary versions)
 FW_Date: Creation date of the firmware (6 bytes)
 (e.g., 31 37 30 33 30 31_{hex} for 17.03.01)
 FW_Time: Creation time of the firmware (6 bytes)
 (e.g., 31 34 31 30 32 30_{hex} for 14:10:20)
 Host_Type: Type of the host-specific firmware interface (20 bytes)
 (e.g., FL IL 24 BK-PAC)
 Host_Version: Version of the host-specific firmware interface (4 bytes)
 Host_State: Status of the host-specific firmware interface (6 bytes)
 Host_Date: Creation date of the host-specific firmware interface (6 bytes)
 Host_Time: Creation time of the host-specific firmware interface (6 bytes)
 Start_FW_Version: Version of the start firmware (4 bytes)
 Start_FW_State: Status of the start firmware (6 bytes)
 Start_FW_Date: Creation date of the start firmware (6 bytes)
 Start_FW_Time: Creation time of the start firmware (6 bytes)
 HW_Art_No: Order No. of the controller board (8 bytes)
 HW_Art_Name: Order designation of controller board (30 bytes)
 HW_Motherboard_ID: Motherboard identification (4 bytes)
 (e.g., 32 43_{hex} for "2C")
 HW_Version: Version of the hardware (2 bytes)
 HW_Vendor_Name: Manufacturer of the controller board (20 bytes)

HW_Serial_No:	Serial number of the controller board	(12 bytes)
HW_Date:	Creation date of the controller board	(6 bytes)

4.6 Error Messages for Firmware Services

4.6.1 Overview

Table 4-6 Overview of error messages (according to error codes)

Code	Services	Page
0905 _{hex}	INCORRECT_PARAMETER	4-51
0907 _{hex}	NO_OBJECT	4-51
0918 _{hex}	UNKNOWN_CODE	4-51
0922 _{hex}	ACTION_HANDLER_CONFLICT	4-51
090A _{hex}	INCORRECT_PARACOUNT	4-51
091D _{hex}	ACTION_HANDLER_OVERLAP	4-52
0A02 _{hex}	INCORRECT_STATE	4-52
0A18 _{hex}	INCORRECT_ATTRIB	4-52
0A19 _{hex}	FRAME_NOT_SO_BIG	4-52
0A22 _{hex}	INCORRECT_TN_NUMBER	4-52
0A2F _{hex}	DEVICE_ZERO	4-52
0A51 _{hex}	INCORRECT_FRAME_REF	4-52
0E22 _{hex}	INTERNAL_TIMEOUT	4-53
0E23 _{hex}	FUNCTION_REG_NOT_FREE	4-53
0E24 _{hex}	ACTION_ERROR	4-53

4.6.2 Positive Messages

ERR_OK 0000_{hex}

Meaning	After successful execution of a function, the firmware generates this message as a positive acknowledgment.
Cause	No errors occurred during execution of the function.

4.6.3 Error Messages

If the firmware generates one of the following codes as an acknowledgment, this indicates that an error occurred during execution, and the called function could not be executed successfully.

INCORRECT_PARAMETER 0905_{hex}

Cause	Incorrect parameters were entered when calling the function.
Remedy	Check the specified parameters.

NO_OBJECT 0907_{hex}

Cause	The object called does not exist.
Remedy	Check the object called or select another.

UNKNOWN_CODE 0918_{hex}

Cause	This service is not supported by this device.
Remedy	Select another service.

ACTION_HANDLER_CONFLICT 0922_{hex}

Cause	An internal firmware error has occurred. Additional info 0031 _{hex} :The error_type and/or error_location registers cannot be read. Additional info FFFF _{hex} :Incorrect parameters detected during Read_Configuration.
--------------	--

INCORRECT_PARACOUNT 090A_{hex}

Cause	The number of parameters is incorrect.
Remedy	Correct the number of parameters.

	ACTION_HANDLER_OVERLAP	091D_{hex}
Cause	Cannot read from or write to the EEPROM. Additional info 0001 _{hex} : Write error Additional info 0002 _{hex} : Read error	
	INCORRECT_STATE	0A02_{hex}
Cause	The called service is not permitted in the current status of the device.	
Remedy	Select another service or change the status of the device, so that the desired service can be called.	
	INCORRECT_ATTRIB	0A18_{hex}
Cause	An invalid bit was activated in the Used_Attributes parameter.	
Remedy	Check that the selected attributes are permitted.	
	FRAME_NOT_SO_BIG	0A19_{hex}
Cause	When accessing the configuration frame, the end of the frame was exceeded.	
Remedy	Modify access to the configuration frame.	
	INCORRECT_TN_NUMBER	0A22_{hex}
Cause	You specified inconsistent device numbers.	
Remedy	Enter the device numbers again.	
	DEVICE_ZERO	0A2F_{hex}
Cause	The Initiate_Load_Configuration service could not be executed. The number of connected Inline modules is either zero or greater than 63.	
Remedy	Change the number of connected Inline modules.	
	INCORRECT_FRAME_REF	0A51_{hex}
Cause	The Frame_Reference value is not one (1).	
Remedy	Change the Frame_Reference to 1.	

INTERNAL_TIMEOUT**0E22_{hex}****Cause**

The function_start_reg was not reset within the timeout.
Additional info xxxx_{hex}: Timeout in hex

FUNCTION_REG_NOT_FREE**0E23_{hex}****Cause**

The function_start_reg is not empty.

ACTION_ERROR**0E24_{hex}****Cause**

The service could not be executed correctly.
Additional info 0005_{hex}: Bus data could not be detected
Additional info 00A5_{hex}: The configuration could not be activated.

This section provides information about

- supported PCP commands

PCP Communication	5-3
5.1 Transmission of Parameter Data	5-3
5.1.1 PCP Configuration in the Web-Based Management.....	5-4
5.1.2 Configuration of the PCP PDU Size	5-4
5.2 Supported PCP Commands	5-5

5 PCP Communication

5.1 Transmission of Parameter Data

Besides exchanging process data there are intelligent devices like frequency inverters or controllers that exchange process data with each other and also large data amounts with the control system. Such data can, for example, be needed for the startup phase of a machine. This kind of parameter data seldom changes and must therefore only be transmitted when required.

The INTERBUS protocol can transmit process data and complex data records (parameter data) at the same time. For this, the complex parameter data is divided into small units, transmitted and put together again.

Peripherals Communication Protocol - PCP

In the INTERBUS system the Peripherals Communication Protocol (PCP) divides the parameter data into single segments. After the transmission it recombines the data. PCP refers to the protocol software which makes connection establishment and connection abort possible etc.

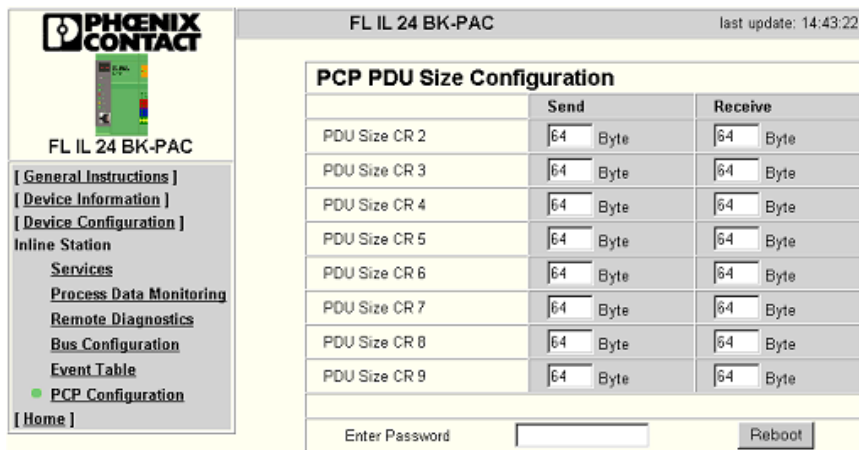


For detailed information on PCP communication, please refer to the IBS SYS PCP G4 UM E user manual, Order No. 27 45 16 9.



Note that a maximum of eight PCP modules can be connected to a bus coupler.

5.1.1 PCP Configuration in the Web-Based Management



PHOENIX CONTACT
FL IL 24 BK-PAC

last update: 14:43:22

PCP PDU Size Configuration

	Send	Receive
PDU Size CR 2	64 Byte	64 Byte
PDU Size CR 3	64 Byte	64 Byte
PDU Size CR 4	64 Byte	64 Byte
PDU Size CR 5	64 Byte	64 Byte
PDU Size CR 6	64 Byte	64 Byte
PDU Size CR 7	64 Byte	64 Byte
PDU Size CR 8	64 Byte	64 Byte
PDU Size CR 9	64 Byte	64 Byte

Enter Password

Navigation Menu:
 [General Instructions]
 [Device Information]
 [Device Configuration]
 Inline Station
 Services
 Process Data Monitoring
 Remote Diagnostics
 Bus Configuration
 Event Table
 PCP Configuration
 [Home]

5.1.2 Configuration of the PCP PDU Size

The standard PDU size for communication with all Phoenix Contact Inline devices is 64 bytes for the transmit and receive direction.

Systems couplers like the ILC 200 UNI are provided with PDU sizes that can be configured. If a different size is configured and if communication is to be with the ILC 200 UNI via an FL IL 24 BK(-PAC), this size must be configured to the values set on the ILC 200 UNI before.

5.2 Supported PCP Commands

Table 5-1 Supported PCP commands

Service	Service Code
Initiate_Request	008B _{hex}
Abort_Request	088D _{hex}
Get_OD_Request	0088 _{hex}
Read_Request	0081 _{hex}
Write_Request	0082 _{hex}
Information_Report_Request	0885 _{hex}
Status_Request	0083 _{hex}
Identify_Request	0087 _{hex}
PNM7_Initiate_Request	00A0 _{hex}
PNM7_Abort_Request	08A1 _{hex}
Load_Kbl_Par_Loc_Request	0264 _{hex}
Service_Execution_Remote_Request	00C1 _{hex}
Read_Kbl_Loc_Request	0203 _{hex}

This section provides information about

- the Modbus/TCP protocol
- supported commands

Modbus/TCP Protocol	6-3
6.1 Modbus Protocol.....	6-3
6.1.1 Modbus Connections	6-3
6.1.2 Modbus Port	6-4
6.1.3 Modbus Conformance Classes.....	6-4
6.1.4 Modbus Message Format	6-4
6.1.5 Modbus Byte Order.....	6-4
6.1.6 Modbus Bit Order.....	6-5
6.2 Modbus Function Codes.....	6-5
6.3 Modbus Table	6-5
6.3.1 Dynamic Modbus/TCP Process Data Table	6-6
6.3.2 Example: Location of the Input/Output Data	6-7
6.3.3 Location of the Process Data in Dynamic Tables	6-8
6.4 Applicable Functions	6-9
6.5 Supported Function Codes.....	6-9
6.5.1 Read Multiple Registers.....	6-10
6.5.2 Write Multiple Registers.....	6-11
6.5.3 Read Coils	6-12
6.5.4 Read Input Discretes	6-13
6.5.5 Read Input Registers	6-14
6.5.6 Write Coils	6-15
6.5.7 Write Single Register	6-16
6.5.8 Read Exception Status	6-17
6.5.9 Exception Status Data Format	6-17
6.5.10 Exception Responses	6-18
6.5.11 Write Multiple Coils	6-19
6.5.12 Read/Write Register	6-20
6.6 Reserved Registers for Command and Status Words	6-22
6.6.1 Command Word.....	6-22
6.6.2 Status Word	6-23
6.6.3 Diagnostics Using the Analog Input Table	6-23
6.6.4 Fault Table	6-24

6.7 Monitoring..... 6-25

6.8 Modbus Monitoring 6-26

6.9 I/O Fault Response Mode..... 6-26

6.9.1 Power Up Table 6-27

6.9.2 Connection Monitoring Table 6-28

6.10 Modbus/TCP PCP Registers 6-30

6 Modbus/TCP Protocol

This section describes the implementation of Modbus/TCP communications on the FL IL 24 BK(-PAC).

Modbus Protocol

- Modbus connections
- Modbus port
- Modbus conformance classes
- Modbus message format

Modbus Tables

- Register/input register table
- Input discrete table
- Coil table

Supported Function Codes

- Read multiple registers
- Write multiple registers
- Read coils
- Read input discretes
- Read input registers
- Write coils
- Write single register
- Read exception status
- Write multiple coils
- Read/write registers

6.1 Modbus Protocol

The bus coupler supports a Modbus/TCP server with the following features:

6.1.1 Modbus Connections

The FL IL 24 BK(-PAC) supports up to eight simultaneous connections. This allows for the fast reestablishment of a connection, i.e., a client is able to re-establish a Modbus connection successfully after it has been aborted.

6.1.2 Modbus Port

Modbus communication on the FL IL 24 BK(-PAC) is supported on the Modbus standard port 502.

6.1.3 Modbus Conformance Classes

The FL IL 24 BK(-PAC) supports Modbus conformance classes 0 and 1.

6.1.4 Modbus Message Format

The Modbus/TCP protocol has a special message format as follows:

Table 6-1 Modbus message format

Byte No.	Meaning
BYTE 0 – 1	Transaction identifier: unique ID generated by the client
BYTE 2 – 3	Protocol identifier = 0
BYTE 4	Length field (upper byte) = 0 (all messages < 256)
BYTE 5	Length field (lower byte) = number of subsequent bytes
BYTE 6	Unit identifier
BYTE 7	Modbus function code
BYTE 8	Data as needed



The "CRC 16" and LRC" check fields normally associated with Modbus are not needed in Modbus/TCP since the checksum mechanisms are used for TCP/IP and data link layers in order to verify the delivery of data packets.

6.1.5 Modbus Byte Order

Modbus uses the "big-endian" representation for addresses and data items. This means that if a numerical quantity larger than a single byte is transmitted (as single or double word), the most significant byte is sent first. Example:

The quantity 0x1234 would be transmitted in the order 0x12 0x34.

The quantity 0x12345678 would be transmitted in the order 0x12 0x34 0x56 0x78.

6.1.6 Modbus Bit Order

If a series of bits is read as a register (e.g., %I1 to %I16), the highest numbered bit (%I16 in this example) is the least significant, and the lowest numbered bit (%I1 in this example) is the most significant.

6.2 Modbus Function Codes

The following function codes are supported:

Table 6-2 Supported function codes

Code No.	Function Code
fc1	Read coils
fc2	Read input discretes
fc3	Read multiple registers
fc4	Read input registers
fc5	Write coil
fc6	Write single register
fc7	Read exception status
fc15	Write multiple coils
fc16	Write multiple registers
fc23	Read/write registers

6.3 Modbus Table

The Modbus protocol's reference table is different from the internal structure of the FL IL 24 BK(-PAC) tables. Modbus refers to a register, input register, discrete input and coil table; the FL IL 24 BK(-PAC) refers to a discrete input (%I), discrete output (%Q), analog inputs (%AI), analog outputs (%AQ), and special register table. The following table shows how each Modbus table is mapped to the FL IL 24 BK(-PAC) tables. Note that all specifications in this table refer to the physical memory inside the FL IL 24 BK(-PAC). In effect, the FL IL 24 BK(-PAC) memory has been given Modbus names. If, for example, we issue a "read input discretes" command to read the inputs in the "read inputs discretes" table, we are actually reading from the internal FL IL 24 BK(-PAC) table %I, which is mapped to the Modbus input discrete table.

Table 6-3 Modbus reference tables

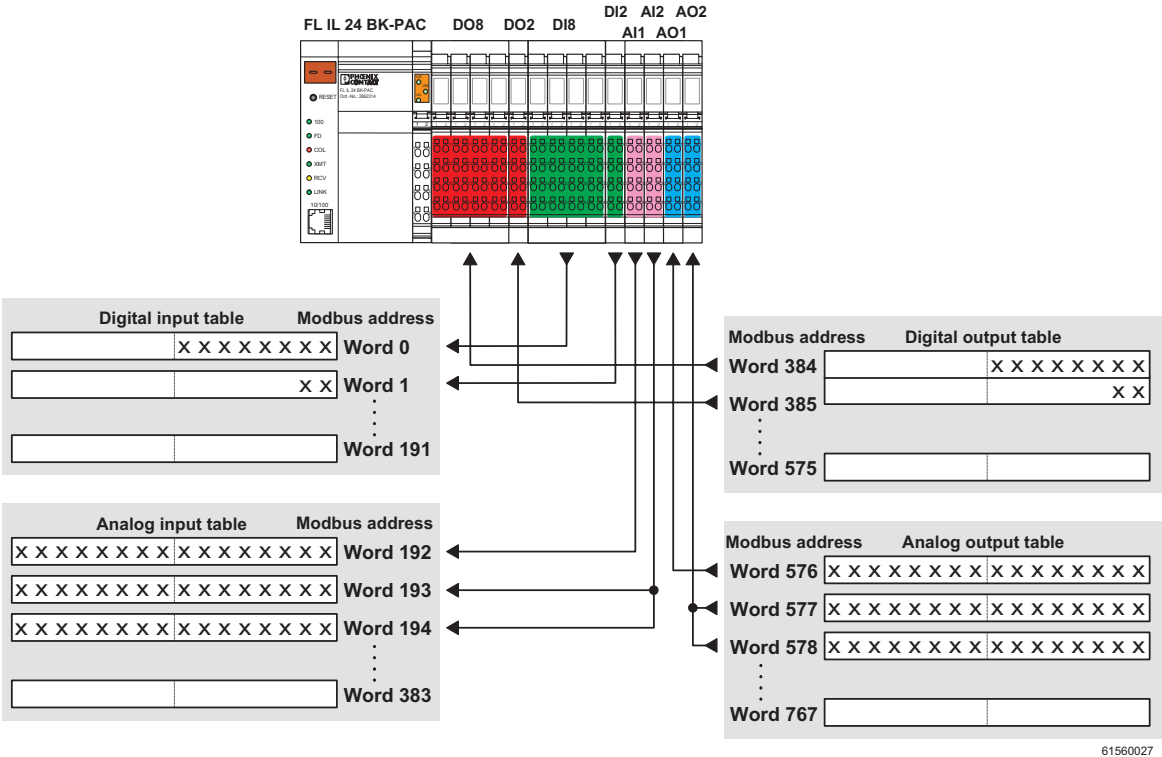
	Modbus Register Table	Modbus Input Register Table	Modbus Input Discrete Table	Modbus Coil Table	Internal FL IL 24 BK(-PAC) Tables
Process data	0 – 191 (16-bit words)	0 – 191 (16-bit words)	0 – 3071 (bits)	---	%I1 – 3072 (bits)
	192 - 383 (16-bit words)	192 - 383 (16-bit words)	---	---	%AI1 – 192 (16-bit words)
	384 - 575 (16-bit words)	384 - 575 (16-bit words)	---	0 – 3071 (bits)	%Q1 – 3072 (bits)
	576 - 767 (16-bit words)	576 - 767 (16-bit words)	---	---	%AQ1 – 192 (16-bit words)
Special register	1024 – 1087 (16-bit words)	1024 – 1087 (16-bit words)	---	---	Fault table (32 error x two 16-bit words per error)
	1280 (16-bit words)	1280 (16-bit words)	---	---	Timeout table, timeout value for connection monitoring
	1400 - 1463 (16-bit words)	1400 - 1463 (16-bit words)	---	---	To 1400: Number of INTERBUS devices. From 1401: ID code of the corresponding device
	2000 (16-bit word)	2000 (16-bit word)	---	---	Process data watchdog timeout
	2002 (16-bit word)	2002 (16-bit word)	---	---	Fault response mode
	2004 (16-bit word)	2004 (16-bit word)	---	---	NetFail reason
PCP	6020 – 6093 (16-bit words)	6020 – 6093 (16-bit words)	---	---	PCP

6.3.1 Dynamic Modbus/TCP Process Data Table

Table 6-4 Dynamic process data table

Modbus Register Table	Modbus Input Register Table	Modbus Input Discrete Table	Modbus Coil Table	Internal FL IL 24 BK(-PAC) Tables
8000 - 8192 (16-bit words)	8000 - 8192 (16-bit words)	---	---	Dynamic process data table

6.3.2 Example: Location of the Input/Output Data



61560027

Figure 6-1 Location of the data in input/output modules

6.3.3 Location of the Process Data in Dynamic Tables

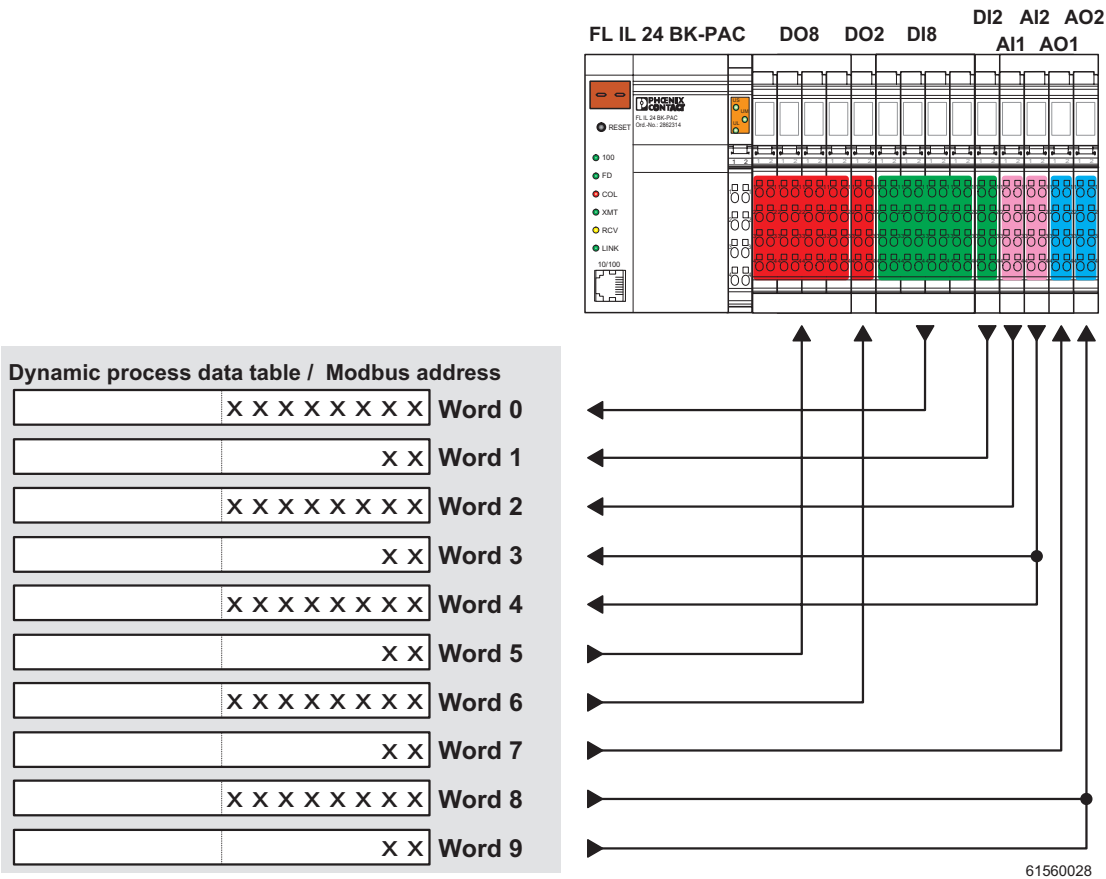


Figure 6-2 Location of the process data in dynamic tables

6.4 Applicable Functions

The FL IL 24 BK-PAC makes no distinction between Modbus register tables and Modbus input register tables, they are identically mapped to all four FL IL 24 BK-PAC I/O tables as well as to the fault table.

Table 6-5 Applicable functions

Function	Function Code	READ/ WRITE	I_TAB.	AI_TAB.	Q_TAB.	AQ_TAB.	Special Register	Dynam. Tables
Read coils	1	READ	---	---	X	---	---	---
Read input discretes	2	READ	X	---	---	---	---	---
Read multiple register	3	READ	X	X	X	X	X	X
Read input register	4	READ	X	X	X	X	X	X
Write coils	5	WRITE	---	---	X	---	---	---
Write single register	6	WRITE	---	---	X	X	X	---
Read exception code	7	READ	---	---	---	---	---	---
Write multiple coils	15	WRITE	---	---	X	---	---	---
Write multiple register	16	WRITE	---	---	X	X	X	X
Read/write registers	23	READ/ WRITE	X	X	X	X	---	X

6.5 Supported Function Codes

The function codes are defined for the Modbus memory mapping, so to determine what area of the memory is affected, refer to table Table 6-3, which maps the Modbus table names to their corresponding FL IL 24 BK-PAC table names.

The FL IL 24 BK -PAC supports the following Modbus function codes:

- Read multiple register (function code 3)
- Write multiple register (function code 16)
- Read coils (function code 1)
- Read input discretes (function code 2)
- Read input register (function code 4)
- Write coils (function code 5)

- Write single register (function code 6)
- Read exception status (function code 7)
- Write multiple coils (function code 15)
- Read/write register (function code 23)



The following function command and response message descriptions start with the Modbus function code (byte 0 is actually byte 7 of the Modbus message format). See : "Modbus message format".

6.5.1 Read Multiple Registers

This command reads from 1 to 125 16-bit words from the Modbus register table. Any part of the Modbus register table can be read using this function. When reading the fault table, however, the entire table must be read. The "read multiple registers" command has the following format:

Table 6-6 Read multiple registers

Byte No.	Meaning
BYTE 0	Function code = 3
BYTE 1 - 2	Register table offset
BYTE 3 - 4	Word count (1 - 125)

The response to the "read multiple registers" command has the following format:

Table 6-7 Response to "read multiple registers"

Byte No.	Meaning
BYTE 0	Function code = 3
BYTE 1	Byte count of response (byte count = 2 x word count in the command)
BYTE 2 – (B + 1)	Register values

If the command accesses an invalid offset or receives an invalid length, an exception response with the following format is output:

Table 6-8 Exception response to "read multiple registers"

Byte No.	Meaning
BYTE 0	Function code = 0x83
BYTE 1	Exception response = 2

6.5.1.1 Examples for "Read Multiple Registers":

Register table offset = 0 and word count = 2 returns %I1-32.

Register table offset = 575 and word count = 2 returns %Q3057-3072 and %AQ1.

Register table offset = 1024 and word count = 64 returns the fault table.

Any combination of the register table offset and the word count which accesses an offset > 767 and < 1024 produces an exception response.

An exception response is also generated when trying to read the fault table while giving a register table offset >1024 or a word count <> 64.

The special registers 1280 - 2004 can only be read when the word count equals one.

6.5.2 Write Multiple Registers

This command writes from 1 to 100 16-bit words in the Modbus register table. Only the part of the Modbus register table mapped to the %Q and %AQ I/O tables may be written using this function.

The "write multiple registers" command has the following format:

Table 6-9 Write multiple registers

Byte No.	Meaning
BYTE 0	Function code = 0x10
BYTE 1 - 2	Register table offset
BYTE 3 - 4	Word count (1 - 100)
BYTE 5	Byte count of response (byte count = 2 x word count)
BYTE 6 – (B + 5)	Register values

The response to the "write multiple registers" command has the following format:

Table 6-10 Response to "write multiple registers"

Byte No.	Meaning
BYTE 0	Function code = 0x10
BYTE 1 - 2	Register table offset (same as command)
BYTE 3 - 4	Word count (same as command)

If the command accesses an invalid offset or contains an invalid length, an exception response in the following format is output:

Table 6-11 Exception response to "write multiple registers"

Byte No.	Meaning
BYTE 0	Function code = 0x90
BYTE 1	Exception word = 2

6.5.2.1 Examples for "Write Multiple Registers":

Register table offset = 384 and word count = 2 writes the register values into %Q1-32.

Register table offset = 575 and word count = 2 writes the register values into %Q3057 - 3072 and %AQ1.

Any combination of the register table offset and the word count that accesses an offset < 384 or > 767 produces an exception response.

6.5.3 Read Coils

This command reads bits 1 to 2000 from the Modbus register table. The "read coils" command has the following format:

Table 6-12 Read coils

Byte No.	Meaning
BYTE 0	Function code = 1
BYTE 1 - 2	Coil table offset
BYTE 3 - 4	Bit count (1 - 2000)

The response to the "read coils" command has the following format:

Table 6-13 Response to "read coils"

Byte No.	Meaning
BYTE 0	Function code = 1
BYTE 1	Byte count of response, byte count (B) = (bit count of command + 7) / 8
BYTE 2 - (B+1)	Bit values (the least significant bit is the first coil)

If the command accesses an invalid offset or contains an invalid length, an exception response in the following format is output:

Table 6-14 Exception response to "read coils"

Byte No.	Meaning
BYTE 0	Function code = 0x81
BYTE 1	Exception word = 2

6.5.3.1 Examples for "Read Coils":

Coil table offset = 0 and bit count = 1 returns coil %Q1. Coil table offset= 0 and bit count = 2000 returns the coil values %Q1-2000.

Coil table offset = 4 and bit count = 13 returns the coil values %Q5-17.

Any combination of the coil table offset and the bit count that accesses an offset > 3072 produces an exception response.

6.5.4 Read Input Discretes

This command reads bits 1 to 2000 from the Modbus input discrete table.

The "read input discretes" command has the following format:

Table 6-15 Read input discretes

Byte No.	Meaning
BYTE 0	Function code = 2
BYTE 1 - 2	Input discretes table offset
BYTE 3 - 4	Bit count (1 - 2000)

The response to the "read input discretes" command has the following format:

Table 6-16 Response to "read input discretes"

Byte No.	Meaning
BYTE 0	Function code = 2
BYTE 1	Byte count of response, $B = (\text{bit count of command} + 7) / 8$.
BYTE 2 - (B + 1)	Bit values (the least significant bit is the first coil)

If the command accesses an invalid offset or contains an invalid length, an exception response in the following format is output:

Table 6-17 Exception response to "read input discretes"

Byte No.	Meaning
BYTE 0	Function code = 0x82
BYTE 1	Exception code

Examples for read digital inputs:

Input discrete table offset = 0 and bit count = 1 returns input discrete %I1.

Input discretes table offset = 0 and bit count = 2000 returns input discrete values %I1-2000.

Input discretes table offset = 4 and bit count = 13 returns input discrete values %Q5-17.

Any combination of the input discretes table offset with bit count that accesses offset > 3072 produces an exception response.

6.5.5 Read Input Registers

This command reads from 1 to 125 16-bit words from the Modbus register table. This command is used exactly like the "read multiple registers" command.

The "read input registers" command has the following format:

Table 6-18 Read input discretes

Byte No.	Meaning
BYTE 0	Function code = 4
BYTE 1 - 2	Register table offset
BYTE 3 - 4	Word count (1 - 125)

The response to the "read input registers" command has the following format:

Table 6-19 Response to "read input registers"

Byte No.	Meaning
BYTE 0	Function code = 4
BYTE 1	Byte count of response (B = 2 x word count in the command)
BYTE 2 - (B + 1)	Register values

If the command accesses an invalid offset or contains an invalid length, an exception response in the following format is output:

Table 6-20 Exception response to "read input registers"

Byte No.	Meaning
BYTE 0	Function code = 0x84
BYTE 1	Exception response = 2

6.5.5.1 Example for the "Read Input Registers":

For examples, please refer to the "Examples for 'Read Multiple Registers'" section.

6.5.6 Write Coils

With this command, 1 bit is written into the Modbus coil table. The "write coil" command has the following format:

Table 6-21 Write coils

Byte No.	Meaning
BYTE 0	Function code = 5
BYTE 1 -2	Coil table offset
BYTE 3	= 0xFF to turn coil ON, = 0 to turn coil OFF
Byte 4	= 0

The response to the "write coils" command has the following format:

Table 6-22 Response to "write coils"

Byte No.	Meaning
BYTE 0	Function code = 5
BYTE 1 - 2	Coil table offset (same as command)
BYTE 3	= 0xFF to turn coil ON, = 0 to turn coil OFF
Byte 4	= 0

If the command accesses an invalid offset, the exception response has the following format:

Table 6-23 Exception response to "write coils"

Byte No.	Meaning
BYTE 0	Function code = 0x85
BYTE 1	Exception code = 2

6.5.6.1 Examples for "Write Coils":

Coil table offset = 0 and the value = 0xFF turns coil %Q1 ON. Coil table offset = 0 and the value = 0 turns coil %Q1 OFF.

Each coil table offset > 3072 produces an exception response.

6.5.7 Write Single Register

This command writes a 16-bit word to the Modbus register table. Only the part of the Modbus register table mapped to the %Q and %AQ I/O tables as well as the first word of the fault table may be written using this function.

The "write single register" command has the following format:

Table 6-24 Write single register

Byte No.	Meaning
BYTE 0	Function code = 6
BYTE 1 - 2	Register table offset
BYTE 3 - 4	Register value

The response to the "write single register" command has the following format:

Table 6-25 Response to "write single register"

Byte No.	Meaning
BYTE 0	Function code = 6
BYTE 1 - 2	Register table offset (same as command)
BYTE 3 - 4	Register value (same as command)

If the command accesses an invalid offset, the exception response has the following format:

Table 6-26 Exception response to "write single register"

Byte No.	Meaning
BYTE 0	Function code = 0x86
BYTE 1	Exception response = 2

6.5.7.1 Examples for "Write Single Registers":

Register table offset = 384 writes the register value into %Q1-16.

Register table offset = 576 writes the register value into %AQ1.

Register table offset = 1024 and register value = 0 clears the fault table.

Register table offset = 1280 and a register value between 200 and 65000 sets a new timeout value for the Modbus/TCP connection.

Register table offset = 2000 and a register value between 200 and 65000 sets a new timeout value for the process data watchdog.

Offset 2002 can be used to set the fault response mode.

1: Reset fault mode

2: Standard fault mode

0: Hold last state mode

Any register table offset < 384 or (> 576 and < 1024) or > 1024 produces an exception response.

6.5.8 Read Exception Status

This command reads an 8-bit status of the FL IL 24 BK-B-PAC.

The "read exception status" command has the following format:

Table 6-27 Read exception status

Byte No.	Meaning
BYTE 0	Function code = 7

The response to the "read exception status" command has the following format:

Table 6-28 Answer to "read exception status"

Byte No.	Meaning
BYTE 0	Function code = 7
BYTE 1	Exception status

6.5.9 Exception Status Data Format

Table 6-29 Exception status data format

Byte No.	Meaning
BYTE 0 - 5	Not used
BYTE 6	Exception status
BYTE 7	Unused fault

6.5.10 Exception Responses

Table 6-30 Exception responses

No.	Designation	Meaning
1	ILLEGAL FUNCTION	The transmitted function code is not supported by this device version.
2	ILLEGAL DATA ADDRESS	The transmitted address is invalid for the device, the combination of reference number and transmission length is wrong. For a controller with 100 registers, an access with an offset of 96 and a length of 4 is successful, an access with an offset of 96 and a length of 5 will generate exception response 2.
3	ILLEGAL DATA VALUE	The value of this request is invalid for this device.
4	DEVICE FAILURE	- Plug & play mode is still active and thus prevents data from being written. - A NetFail occurred. - In addition, a DDI device could be closed that has exclusive write access. In this case it is not possible to write data via Modbus/TCP.
16	DOUBLE PCP CONNECTION	Another client already has a PCP connection to this bus coupler.

6.5.11 Write Multiple Coils

This command writes 1 up to 800 bits into the Modbus coil table.

The "write multiple coils" command has the following format:

Table 6-31 Write multiple coils

Byte No.	Meaning
BYTE 0	Function code = 0x0F
BYTE 1 - 2	Coil table offset
BYTE 3 - 4	Bit count
BYTE 5	Byte count
BYTE 6 – (B + 5)	Bit values (the least significant bit is the first coil)

The response to the "write multiple coils" command has the following format:

Table 6-32 Response to "write multiple coils"

Byte No.	Meaning
BYTE 0	Function code = 0x0F
BYTE 1 - 2	Coil table offset (same as command)
BYTE 3 - 4	Bit count (same as command)

If the command uses an invalid offset, the following exception response is generated:

Table 6-33 Exception response to "write multiple coils"

Byte No.	Meaning
BYTE 0	Function code = 0x8F
BYTE 1	Exception response = 2

6.5.11.1 Example for the "Write Multiple Coils" Command:

Coil table offset = 0 and bit count = 2 with the value 3 sets the coils %Q1 and %Q2.

Coil table offset = 0 and bit count = 2 with the value 0 resets the coils %Q1 and %Q2.

6.5.12 Read/Write Register

This command reads 1 up to 125 words from the Modbus register table and writes 1 up to 100 16-bit words into the Modbus register table. This command can only write in that part of the table that reflects the coils (%Q and %AQ).

The write/read command has the following format:

Table 6-34 Read/write register

Byte No.	Description
BYTE 0	Function code = 0x17
BYTE 1 - 2	Read register table offset
BYTE 3 - 4	Read word count (1 to 125)
BYTE 5 - 6	Write register table offset
BYTE 7 - 8	Write word count (1 - 100)
BYTE 9	Write byte count ($B = 2 \times \text{write word count}$)
BYTE 10 - (B + 9)	Write register values

The response to the "read/write register" command has the following format:

Table 6-35 Answer to "read/write register"

Byte No.	Description
BYTE 0	Function code = 0x17
BYTE 1	Byte count ($B = 2 \times \text{read word count}$)
BYTE 2 - (B + 1)	Read register values

If the command accesses an invalid offset, the exception response has the following format:

Table 6-36 Exception response to "read/write register"

Byte No.	Description
BYTE 0	Function code = 0x97
BYTE 1	Exception code

6.5.12.1 Examples for the "read/write register" command:

Register table offset = 0 and word count = 2 returns the input discrete values %1-32.

Register table offset = 575 and word count = 2 returns the value of the coils %Q3057-3072 and the analog output %AQ1.

Register table offset = 1024 and word count = 64 returns a fault table.

Any combination of the register table offset and a word count >767 and <1024 produces an exception response.

The attempt to read the fault table with a register table offset >1024 and a word count not equal to 64 also generates an exception response.

The special register 1280 - 2004 can only be read if the word count equals one.
Register table offset = 384 and word count = 2 writes the register values into coils %Q1-32.

Register table offset = 575 and word count = 2 writes the register values into coils %Q3057-3072 and the analog output %AQ1.

Any combination of register table offset and word count between >384 and <767 generates an exception response except writing one word into registers 2000 and 2002.

6.6 Reserved Registers for Command and Status Words

6.6.1 Command Word

The last word of the analog output table is automatically reserved by the bus coupler as network interface command word and starts at Modbus address 40767. This command word allows the Ethernet host controller, e.g., PLC, to send commands with basic functions to the module. These commands enable startup without configuration software.

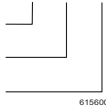
Table 6-37 Structure of the analog output table

Analog Output Table	Address
First output word	576
	577
	
Command word	767

The bits are defined as shown in Table 6-38. The remaining bits are reserved for future use. Activation/deactivation of plug & play mode is executed by means of the least significant bit of the command word. Bit 0 = "0" -> PP inactive; bit 0 = "1" -> PP active.

If a NetFail occurred it can be acknowledged by setting bit 1 in the command word. If NetFail has been acknowledged successfully, bit 1 is reset to "zero".

Table 6-38 Network interface command word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved bits													X	X	X
															
													Clear peripheral fault		
													Clear NetFail		
													Plug & play		

61560030

6.6.2 Status Word

Table 6-39 Structure of the input discretes table

Input Discretes Table	Address
The first 16 input bits	0
	1
	
Status word	191

The last word in the input discretes table is automatically reserved by the bus coupler as network interface status word. This word allows the Ethernet host controller, e.g., PLC, to receive up-to-date diagnostic information without using a configuration software.

Only the two least significant bits have a function. Bit 0 = "0" means that a fault occurred (e.g., bus error). Bit 0 = "1" means that no fault occurred. Bit 1 indicates whether there is a Net_Fail (one) or not (zero).

This results in the following values for the status word:

0: A fault occurred (e.g., bus fault)

1: No fault occurred.

2: A NetFail occurred.

Table 6-40 Status word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved bits														X	X

6.6.3 Diagnostics Using the Analog Input Table

Table 6-41 Structure of the analog input table

Analog Input Table	Address
First input word	192
	193
	
Diagnostic status register	382
Diagnostic parameter register	383

The diagnostic data is entered into the analog input table. The diagnostic status register and the diagnostic parameter register use the last two words in the analog input table.

6.6.4 Fault Table

Data Format of the Fault Table

The internal fault table, which may contain up to 35 fault codes, can be accessed by a Modbus client. This internal fault table works according to the FIFO principle (First In, First Out). This means that the 33rd fault entry deletes the oldest fault entry.

An application can request all fault entries or it can delete all entries via one command sent to the bus coupler. Every fault entry is written in two words, beginning with the reference 1024 in the register table. All fault entries serve as information and do not stop the bus coupler.

Reading the Fault Table Data

The entire fault table can be read using the "read multiple registers" command, starting at the beginning of the fault table (1024), with a length of 64 registers. It is impossible to only read parts of the fault table. Empty entries contain the value "0".



Please note that the entries are shifted downwards so that the latest fault entry is located at position 1024.

Deleting the Fault Table Data

If required, the application may write the value "0" into the first register (1024) of the fault table using the "write single register" command. A client cannot write to any other register.

Table 6-42 Registers

31..... 16	15 0
Diagnostic parameter register	Diagnostic status register

Fault Table Entries

Each fault entry is two words in length and is formatted as follows:

If a fault occurs, one or more bits are set in the diagnostic status register (PF, BUS or CTRL) and a new entry is added to the fault table. The representation of the fault table entry is shown in the following table:

Table 6-43 Fault table

Fault Table		
Fault No.	Fault Entry (2 Words)	
1	Diagnostic parameter register	Diagnostic status register
2	Diagnostic parameter register	Diagnostic status register
3	Diagnostic parameter register	Diagnostic status register
.....		
32	Diagnostic parameter register	Diagnostic status register

6.7 Monitoring

The three following monitoring mechanisms are available in the Modbus operating mode (see Section 3.8 on page 3-33).

Table 6-44 Monitoring functions

Monitoring Mechanism	Monitoring ...			
	... the client application	... the individual channels	... the Ethernet connection	... process data exchange
Process data watchdog (process data monitoring)	X	---	X	X
Host checking	---	---	X	---
DTI/Modbus monitoring	X	X	X	---

6.8 Modbus Monitoring

A monitoring mechanism can be activated for every Modbus/TCP connection in order for the FL IL 24 BK-PAC to detect a fault in the network (e.g., defective cable) or in the client (operating system crash or error in the TCP/IP protocol stack) and react correspondingly. The monitoring mechanism is activated via the relevant TCP connection upon the first read or write procedure.

To change the timeout value for the relevant TCP connection, write the new timeout value to the timeout table to the special address 1280 using the functions fc 6 or fc 16. The value of this entry is the value of the timeout table. The time is indicated in milliseconds and ranges from 200 ms to 65000 ms.

A timeout value of "0" deactivates the monitoring function. Values between 1 and 199 as well as values bigger than 65000 ms generate exception response 3 (ILLEGAL DATA VALUE).



Connection monitoring with the new timeout values is only activated after a Modbus/TCP function has been executed on the relevant TCP connection.

After the first access by a Modbus/TCP function, all other access must be carried out using the entered timeout value. Otherwise, fault response mode is activated and the Modbus/TCP connection is disabled.

6.9 I/O Fault Response Mode

In case the communication connection is disrupted, the user can select the reaction of the FL IL 24 BK-PAC beforehand. Use the DDI "Set_Value" command on the object ID 2277_{hex}. The following table shows the three possible reactions:

Table 6-45 Available fault response modes

Fault Response Mode	Value	Function
Reset fault mode (default)	1	The discrete outputs are set to "0" and the analog outputs are set to the value configured by the user (default = "0")
Standard fault mode	0	All outputs are set to "0".
Hold last state mode	2	All outputs retain their last value.

The following tables show the output tables and real output values for the first two configuration choices. One table is for restart after power up and the other table for restart after a fault occurred. The output table is part of the bus coupler's internal memory; real output values are the values of the output modules. The output table consists of two parts: discrete and analog outputs.

6.9.1 Power Up Table

The FL IL 24 BK(-PAC) output table of the is stored in a volatile memory. For this reason, all values of the output table are set to "0" after a power up. Configuration settings are stored in a non-volatile EEPROM.

Table 6-46 Power up sequence

Power Up Sequence				
FL IL 24 BK-PAC Status	Configuration: Reset Fault Mode		Configuration: Last State Fault Mode	
	Output table	Real output	Output table	Real output
Power up	"0"	"0"	"0"	"0"
First read access in output table after power up	"0" plus the new values	Output table	"0" plus the new values	Output table
Operating	"0" plus the sum of all new values	Output table	"0" plus the sum of all new values	Output table

Example: A station consists of 3 I/O modules, a 16-bit analog output module (AO), a 16-bit discrete output module (DO 16) and 2-bit discrete output module (DO 2). After a power up, all outputs are set to "0":

Module	AO	DO 16	DO 2
Value	0x0000	0x0000	0x0000

If 0x0200 as first value after the power up is written into the output table of the DO16 module, we get the following output values.

Module	AO	DO 16	DO 2
Value	0x0000	0x0200	0x0000

This is the "'0" plus the new values" state.

If values such as 0x0010 for AO, 0x0001 for DO 2 and 0xACDC for DO 16 have been written into the output table via several write accesses, the following values are output:

Module	AO	DO 16	DO 2
Value	0x0010	0xACDC	0x0001

This is the "'0" plus the sum of all new values" state.

6.9.2 Connection Monitoring Table

This table shows the output values after the connection monitoring or the process data watchdog detected a fault such as a disconnection or a communication error while the voltage supply remains the same.

Table 6-47 Connection monitoring table

Connection monitoring table after connection abort, a cable interrupt or a communication error.				
Configuration of the FL IL 24 BK-PAC	Configuration: "Reset Fault Mode"		Configuration: "Last State Fault Mode"	
	Output table	Real output	Output table	Real output
Cable removal or communication error after cable interrupt	Last values in the output table	All discrete outputs are set to "0".	Last values in the output table	Values of the output table
First write access in the output table after restoring the connection	Last values in the output table plus the newly written values	Output table	Last values in the output table plus the newly written values	Output table
Operating	Last values in the output table plus all newly written values	Output table	Last values in the output table plus all newly written values	Output table

Example: The last entries in the output table have the following values:

Module	AO	DO 16	DO 2
Value	0x0123	0x4321	0x0002

Writing 0x00A1 into the output table of the DO 16 as the first value after having restored the connection gives the following actual output value:

Module	AO	DO 16	DO 2
Value	0x0123	0x00A1	0x0002

This is the "Last values in the output table plus the newly written values" state.

If values such as 0x0010 for AO, 0x0001 for DO 2 and 0xACDC for DO 16 have been written into the output table via several write accesses, the following values are output:

Module	AO	DO 16	DO 2
Value	0x0010	0xACDC	0x0001

This is the "Last values in the output table plus the newly written values" state.

6.10 Modbus/TCP PCP Registers

There are two classes of PCP registers:

- Communication register for data exchange with the desired PCP device
- Configuration register for selecting the Invoke ID, index, and subindex of a PCP device

The FL IL 24 BK(-PAC) supports eight PCP devices, which means that eight communication registers and 24 configuration registers are supported.

Table 6-48 PCP registers

PCP Communication Reference	BK Communication Register	Configuration Register	Remark
CR 2	6020		
		6021	Index
		6022	Subindex
		6023	Invoke ID
		6024 - 6029	Reserved
CR 3	6030		
		6031	Index
		6032	Subindex
		6033	Invoke ID
		6034 - 6039	Reserved
CR 4	6040		
		6041	Index
		6042	Subindex
		6043	Invoke ID
		6044 - 6049	Reserved
CR 5	6050		
		6051	Index
		6052	Subindex
		6053	Invoke ID
		6054 - 6059	Reserved
CR 6	6060		
		6061	Index
		6062	Subindex
		6063	Invoke ID
		6064 - 6069	Reserved
CR 7	6070		
		6071	Index
		6072	Subindex
		6073	Invoke ID
		60724 - 6079	Reserved

Table 6-48 PCP registers

CR 8	6080		
		6081	Index
		6082	Subindex
		6083	Invoke ID
CR 9	6090	6084 - 6089	Reserved
		6091	Index
		6092	Subindex
		6093	Invoke ID
		6094 - 6099	Reserved

Example: To read object 0x5FE0 of an IB IL RS232 using communication reference 4, the configuration registers (6041 - 6043) must be set to the desired values (e.g., 6041 index: 0x5FE0, 6042 subindex: 0x0, 6043 Invoke ID: 0x0) first using the FC 16 command. Then use the FC 3 command to read 29 words via the communication register 6040.

A Modbus function can only be used to read or write in one single PCP index. You cannot use the FC 3 command to read 20 words from the registers 6020 to 6039. The communication register contains a different value range which depends on the selected register values and the module used. The IB IL RS 232 module, for example, has three different PCP objects. Two of them consist of one word and the third of 29 words. The three configuration registers can be read/written using a single Modbus command. Access to a reserved register produces an exception response.

Section 7

This section provides information about

- technical data
- ordering data

Technical Data.....	7-3
7.1 Ordering Data.....	7-11

7 Technical Data

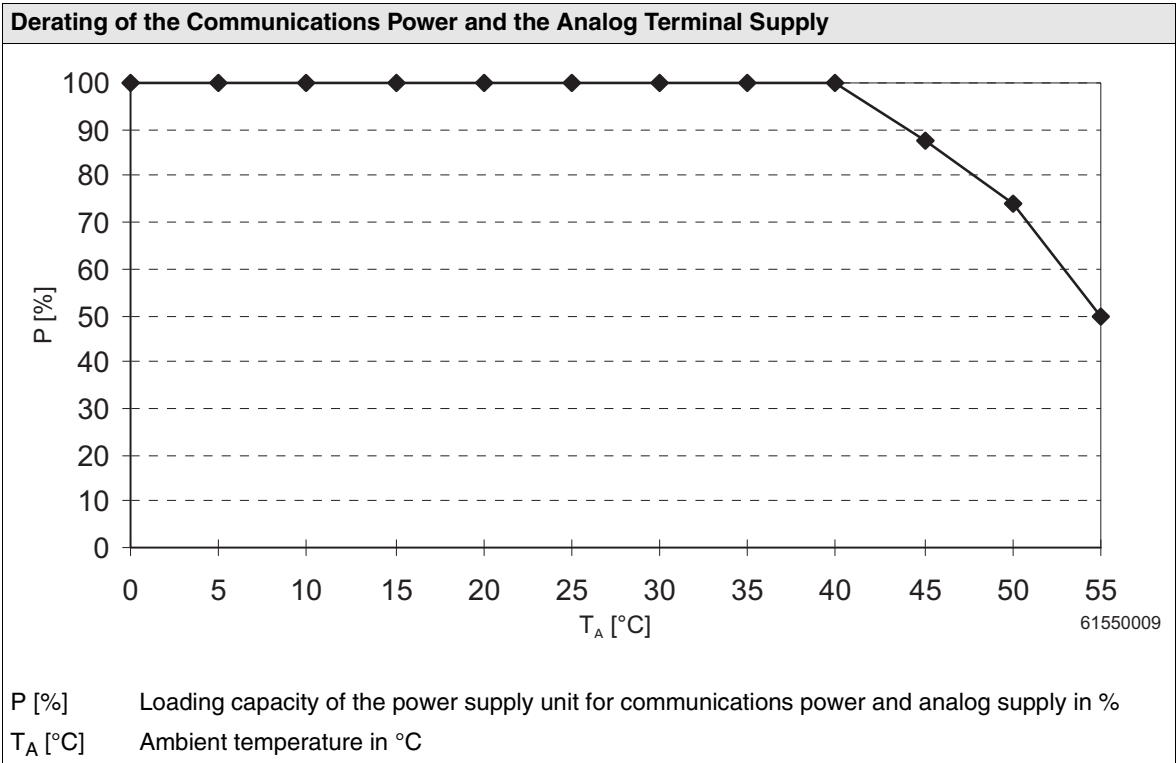
General Data	
Function	Ethernet/Inline bus coupler
Housing dimensions (width x height x depth)	90 mm x 116 mm x 72 mm (3.543 x 4.567 x 2.835 in.)
Permissible operating temperature (EN 60204-1)	0°C to 55°C (+32°F to +131°F)
Permissible storage temperature (EN 60204-1)	-25°C to 85°C (-13°F to +185°F)
Degree of protection	IP20, DIN 40050, IEC 60529
Class of protection	Class 3 VDE 0106; IEC 60536
Humidity (operation) (EN 60204-1)	5% to 90%, no condensation
Humidity (storage) (EN 60204-1)	5% to 95%, no condensation
Air pressure (operation)	80 kPa to 108 kPa, 2000 m (6562 ft.) above sea level
Air pressure (storage)	70 kPa to 108 kPa, 3000 m (9843 ft.) above sea level
Preferred mounting position	Perpendicular to a standard DIN rail
Connection to protective earth ground	The functional earth ground must be connected to the 24 V DC supply/functional earth ground connection. The contacts are directly connected to the potential jumper and FE springs on the bottom of the housing. The terminal is grounded when it is snapped onto a grounded DIN rail. Functional earth ground is only used to discharge interference.
Environmental compatibility	Free from substances which would hinder coating with paint or varnish (according to VW specification)
Resistance to solvents	Standard solvents
Weight	270 g, typical
24 V Main Supply/24 V Segment Supply	
Connection method	Spring-cage terminals
Recommended cable lengths	30 m (98.43 ft.), maximum; do not route cable through outdoor areas
Voltage continuation	Through potential routing
Special demands on the voltage supply	The supplies U_M/U_S and the bus coupler supply U_{BK} do not have the same ground potential because they are supplied by two separate power supply units.
Behavior in the event of voltage fluctuations	Voltages (main and segment supply) that are transferred from the bus coupler to the potential jumpers follow the supply voltages without delay.
Nominal value	24 V DC

24 V Main Supply/24 V Segment Supply	
Tolerance	-15%/+20% (according to EN 61131-2)
Ripple	±5%
Permissible range	19.2 V to 30 V
Current carrying capacity	8 A, maximum (total current of U_S and U_M)
Safety equipment	
Surge voltage	Input protective diodes (can be destroyed by permanent overload) Pulse loads up to 1500 V are short circuited by the input protective diode.
Polarity reversal	Parallel diodes against polarity reversal; in the event of an error the high current through the diodes causes the preconnected fuse to blow.
	This 24 V area must be fused externally. The power supply unit must be able to supply 4 times (400%) the nominal current of the external fuse, to ensure that the fuse blows safely in the event of an error.

24 V Bus Coupler Supply	
Connection method	Spring-cage terminals
Recommended cable lengths	30 m (98.43 ft.), maximum; do not route cable through outdoor areas
Voltage continuation	Via potential routing U_L, U_{ANA}
Safety equipment	
Surge voltage	Input protective diodes (can be destroyed by permanent overload) Pulse loads up to 1500 V are short circuited by the input protective diode.
Polarity reversal	Serial diode in the lead path of the power supply unit; in the event of an error only a low current flows. In the event of an error the fuse in the external power supply unit does not trip. Ensure protection of 2 A by fuses through the external power supply unit.
	Observe the current consumption of the modules Observe the logic current consumption of each device when configuring an Inline station. This information is given in every module-specific data sheet. The current consumption can differ depending on the individual module. The permissible number of devices that can be connected therefore depends on the specific station structure.
Nominal value	24 V DC

24 V Bus Coupler Supply	
Tolerance	-15%/+20% (according to EN 61131-2)
Ripple	±5%
Permissible range	19.2 V to 30 V
Minimum current consumption at nominal voltage	92 mA (At no-load operation, i.e., Ethernet connected, no local bus devices connected, bus inactive)
Maximum current consumption at nominal voltage	1.5 A (Loading the 7.5 V communications power with 2 A, the 24 V analog voltage with 0.5 A)

24 V Module Supply	
- Communications Power (Potential Jumper)	
Nominal value	7.5 V DC
Tolerance	±5%
Ripple	±1.5%
Maximum output current	2 A DC (observe derating)
Safety equipment	Electronic short-circuit protection
- Analog Supply (Potential Jumper)	
Nominal value	24 V DC
Tolerance	-15%/+20%
Ripple	±5%
Maximum output current	0.5 A DC (observe derating)
Safety equipment	Electronic short-circuit protection



Power Dissipation	
Formula to Calculate the Power Dissipation of the Electronics	
$P_{EL} = P_{BUS} + P_{PERI}$	
$P_{EL} = 2.6 \text{ W} + \left(1.1 \frac{\text{W}}{\text{A}} \times \sum_{n=0}^a I_{Ln}\right) + \left(0.7 \frac{\text{W}}{\text{A}} \times \sum_{m=0}^b I_{Lm}\right)$	
Where	
P_{EL}	Total power dissipation in the terminal
P_{BUS}	Power dissipation for bus operation without I/O load (permanent)
P_{PERI}	Power dissipation with I/O connected
I_{Ln}	Current consumption of the device n from the communications power
n	Index of the number of connected devices ($n = 1$ to a)
a	Number of connected devices (with communications power supply)
$\sum_{n=0}^a I_{Ln}$	Total current consumption of the devices from the 7.5 V communications power (2 A, maximum)
I_{Lm}	Current consumption of the device m from the analog supply
m	Index of the number of connected analog devices ($m = 1$ to b)
b	Number of connected analog devices (supplied with analog voltage)
$\sum_{m=0}^b I_{Lm}$	Total current consumption of the devices from the 24 V analog supply (0.5 A, maximum)

Power Dissipation/Derating

Using the maximum currents 2 A (logic current) and 0.5 A (current for analog terminals) in the formula to calculate the power dissipation when the I/O is connected gives the following result:

$$P_{PERI} = 2.2 \text{ W} + 0.35 \text{ W} = 2.55 \text{ W}$$

2.55 W corresponds to 100% current carrying capacity of the power supply unit in the derating curves on page 7-6.

Make sure that the indicated nominal current carrying capacity in the derating curves is not exceeded when the ambient temperature is above 40°C (104°F). Corresponding with the formula, the total current carrying capacity of the connected I/O is relevant (P_{PERI}). If, for example, no current is drawn from the analog supply, the percentage of current coming from the communications power can be increased.

Example:

Ambient temperature: 55°C (131°F)

1. Nominal current carrying capacity of the communications power and analog supply: 50% according to the diagram

$$I_{LLogic} = 1 \text{ A}, I_{LAnalog} = 0.25 \text{ A}$$

$$P_{PERI} = 1.1 \text{ W} + 0.175 \text{ W}$$

$$P_{PERI} = 1.275 \text{ W} \text{ (corresponds to 50\% of 2.55 W)}$$

2. Possible logic current if the analog supply is not loaded:

$$P_{PERI} = 1.1 \text{ W/A} \times I_{LLogic} + 0 \text{ W}$$

$$P_{PERI} / 1.1 \text{ W/A} = I_{LLogic}$$

$$I_{LLogic} = 1.275 \text{ W} / 1.1 \text{ W/A}$$

$$I_{LLogic} = 1.159 \text{ A}$$

Safety Equipment

Surge voltage (segment supply/main supply/bus coupler supply)	Input protective diodes (can be destroyed by permanent overload) Pulse loads up to 1500 V are short circuited by the input protective diode.
Polarity reversal (segment supply/main supply)	Parallel diodes against polarity reversal; in the event of an error the high current through the diodes causes the preconnected fuse to blow.
Polarity reversal (bus coupler supply)	Serial diode in the lead path of the power supply unit; in the event of an error only a low current flows. In the event of an error the fuse in the external power supply unit does not trip. Ensure protection of 2 A by fuses through the external power supply unit.

Bus Interface of the Lower-Level System Bus	
Interface	Inline local bus
Electrical isolation	No
Number of Inline terminals that can be connected Limited by software Limited by power supply unit	63, maximum Maximum logic current consumption of the connected local bus modules: $I_{\max} \leq 2 \text{ A DC}$
 Observe the current consumption of the modules Observe the logic current consumption of each device when configuring an Inline station. This information is given in every module-specific data sheet. The current consumption can differ depending on the individual module. The permissible number of devices that can be connected therefore depends on the specific station structure.	

Interfaces	
Ethernet Interface	
Number	One
Connection format	8-pos. RJ45 socket on the bus coupler
Connection medium	Twisted pair cable with a conductor cross section of 0.14 mm^2 to 0.22 mm^2 (26 AWG to 25 AWG)
Cable impedance	100 Ω
Transmission speed	10/100 Mbps
Maximum network segment expansion	100 m (328 ft.)

Protocols/MIBs	
Supported protocols	TCP / UDP SNMP BootP TFTP HTTP PCP Modbus/TCP
Supported standard MIB	RFC 1213 (MIB II)
Supported private MIBs	PhoenixContact MIB FL MIB FL Device MIB

Mechanical Tests	
Shock test according to IEC 60068-2-27	Operation: 25g, 11 ms period, half-sine shock pulse Storage/transport: 50g, 11 ms period, half-sine shock pulse
Vibration resistance according to IEC 60068-2-6	Operation/storage/transport: 5g, 150 Hz, Criterion A
Free fall according to IEC 60068-2-32	1 m (3.28 ft.)

Conformance With EMC Directives	
Developed according to IEC 61000-6.2	
IEC 61000-4-2 (ESD)	Criterion B 6 kV contact discharge 6 kV air discharge (without labeling field) 8 kV air discharge (with labeling field in place)
IEC 61000-4-3 (radiated noise immunity)	Criterion A
IEC 61000-4-4 (burst)	Criterion B
IEC 61000-4-5 (surge)	Criterion B
IEC 61000-4-6 (conducted noise immunity)	Criterion A
IEC 61000-4-8 (noise immunity against magnetic fields)	Criterion A
EN 55011 (noise emission)	Class A
 Warning: Portable radiotelephone equipment ($P \geq 2$ W) must not be operated any closer than 2 m (6.56 ft). There should be no strong radio transmitters or ISM (industrial scientific and medical) devices in the vicinity.	

Approvals	
Approvals	cUL 508, cUL 2279, cUL 1604 Class 1 Div 2

7.1 Ordering Data

Description	Order Designation	Order No.
Ethernet/Inline bus coupler with connector and labeling field	FL IL 24 BK-PAC	28 62 31 4
Ethernet/Inline bus coupler	FL IL 24 BK	28 31 05 7
Connector, with color print	IB IL SCN-8-CP	27 27 60 8
Labeling field	IB IL FIELD 8	27 27 50 1
End clamp	E/UK	12 01 44 2
Zack "Quick" marker strip	ZBFM 6-... (see CLIPLINE)	
Factory Manager, network management software	FL SWT	28 31 04 4
FL SNMP OPC gateway, software for information exchange between SNMP and OPC	FL SNMP OPC SERVER	28 32 16 6
	FL OPC SNMP AGENT	28 32 17 9
OPC server	IBS OPC SERVER	27 29 12 7
CD-ROM with user documentation in pdf format, driver software, example program, and OPC configurator	CD FL IL 24 BK	28 32 06 9
"Configuring and Installing the INTERBUS Inline Product Range" user manual	IB IL SYS PRO UM E	27 43 04 8
RJ45 gray connector set for linear cable (2 pieces)	FL PLUG RJ45 GR/2	27 44 85 6
RJ45 green connector set for crossed cable (2 pieces)	FL PLUG RJ45 GN/2	27 44 57 1
Double sheathed Ethernet cable	FL CAT5 HEAVY	27 44 81 4
Flexible Ethernet cable	FL CAT5 FLEX	27 44 83 0
Assembly tool for RJ45 connector	FL CRIMPTOOL	27 44 86 9
Media converter 660 nm	FL MC 10BASE-T/FO POF	27 44 51 3
Voltage supplies	QUINT-PS ... see "INTERFACE" catalog	
Keying profile (100 pcs./package)	CP-MSTB see "COMBICON" catalog	17 34 63 4
Zack markers for labeling terminals	ZB 6 ... see "CLIPLINE" catalog	
Labeling field covering one connector	IB IL FIELD 2	27 27 50 1
Labeling field covering four connectors	IB IL FIELD 8	27 27 51 5
Insert strips for IB IL FIELD 2, perforated, can be labeled using a laser printer, marker pen or CMS system (72 strips, 1 pcs./package)	ESL 62X10	08 09 49 2
Insert strips for IB IL FIELD 8, perforated, can be labeled using a laser printer, marker pen or CMS system (15 strips, 5 pcs./package)	ESL 62X46	08 09 50 2

Description	Order Designation	Order No.
DIN EN 50022 DIN rail, 2 meters (6.56 ft.)	NS 35/7,5 gelocht NS 35/7,5 ungelocht	08 01 73 3 08 01 68 1
End clamp snapped on without tools (50 pcs./package)	CLIPFIX 35	30 22 21 8
End clamp fixed using screws (50 pcs./package)	E/UK	12 01 44 2
Screwdriver according to DIN 5264, blade width 3.5 mm (0.138 in.)	SZF 1 - 0,6 x 3,5	12 04 51 7

Phoenix Contact GmbH & Co. KG
Flachsmarktstr. 8
32825 Blomberg
Germany



+ 49 - (0) 52 35 - 3-00



+ 49 - (0) 52 35 - 3-4 12 00



www.phoenixcontact.com



Worldwide Locations:
www.phoenixcontact.com/salesnetwork

HOTLINE:

If problems occur which cannot be solved with the help of this documentation,
please contact our hotline:



+ 49 - (0) 52 35 - 3-4 18 88



factoryline-service@phoenixcontact.com

A Reference Data

A 1 Table of Figures

Section 1

Figure 1-1:	Front view of the FL IL 24 BK-PAC	1-4
Figure 1-2:	Structure of the FL IL 24 BK-PAC bus coupler	1-5
Figure 1-3:	Typical connection of the supply voltage	1-7
Figure 1-4:	Basic structure of an Inline module	1-16
Figure 1-5:	Inline connector types	1-17
Figure 1-6:	Internal structure of the connectors	1-19
Figure 1-7:	Connector keying	1-20
Figure 1-8:	Function identification	1-21
Figure 1-9:	Terminal point numbering	1-22
Figure 1-10:	Labeling of modules	1-23
Figure 1-11:	Dimensions of the electronics bases (2-slot housing)	1-24
Figure 1-12:	Dimensions of the electronics bases (4-slot housing)	1-25
Figure 1-13:	Dimensions of the electronics bases (8-slot housing)	1-25
Figure 1-14:	Connector dimensions	1-26
Figure 1-15:	Potential and data routing	1-27
Figure 1-16:	Typical connection of the supply voltage	1-30
Figure 1-17:	Logic and analog circuit	1-31
Figure 1-18:	Main circuit	1-32
Figure 1-19:	Segment circuit	1-34
Figure 1-20:	Potential areas in the bus coupler (two voltage supplies)	1-36
Figure 1-21:	Bus coupler potentials (one voltage supply)	1-37
Figure 1-22:	Power connector for supply from a single power supply unit	1-37
Figure 1-23:	Example: Interruption/creation of the potential jumpers using the power terminal	1-38
Figure 1-24:	Electrical isolation between Ethernet bus coupler and analog module	1-39

Figure 1-25:	Structure of I/O supplies that are electrically isolated from one another	1-41
Figure 1-26:	LEDs on the Ethernet bus coupler	1-42
Figure 1-27:	Possible indicators on supply terminals (segment terminal with and without fuse and power terminal)	1-44
Figure 1-28:	I/O module indicators	1-45
Figure 1-29:	Snapping on a module	1-49
Figure 1-30:	Removing a module	1-51
Figure 1-31:	Replacing a fuse	1-53
Figure 1-32:	Additional grounding of the FL IL 24 BK(-PAC)	1-54
Figure 1-33:	Connection of analog sensors, signal cables > 10 m (32.81 ft.)	1-57
Figure 1-34:	Connection of actuators, signal cables > 10 m (32.81 ft.) ...	1-58
Figure 1-35:	Connecting unshielded cables	1-59
Figure 1-36:	Connecting the shield to the shield connector	1-61
Figure 1-37:	Shield connection clamp alignment	1-63
Figure 1-38:	2-wire termination for digital devices	1-68
Figure 1-39:	3-wire termination for digital devices	1-69
Figure 1-40:	4-wire termination for digital devices	1-70

Section 2

Figure 2-1:	Structure of IP addresses	2-7
Figure 2-2:	WBM homepage	2-10
Figure 2-3:	Screenshot of the XML data	2-16
Figure 2-4:	WBM firmware update	2-18
Figure 2-5:	Trap representation in the Factory Manager using a few example traps	2-19
Figure 2-6:	Defining the trap manager	2-20
Figure 2-7:	I/O browser screen	2-22
Figure 2-8:	Linking items and terminal points	2-23

Section 3

Figure 3-1:	Software structure	3-3
Figure 3-2:	Using the driver software in the application program	3-5
Figure 3-3:	Position of the user data for individual devices in the word array	3-6
Figure 3-4:	Position of the user data for several devices in the word array	3-6
Figure 3-5:	Position of the process data according to the physical bus configuration	3-7
Figure 3-6:	"Normal" mode / expert mode and P&P mode inactive	3-10
Figure 3-7:	P&P mode inactive - expert mode active	3-10
Figure 3-8:	P&P mode active - expert mode inactive	3-11
Figure 3-9:	P&P mode active and expert mode active	3-11
Figure 3-10:	Execution of a remote procedure call	3-17
Figure 3-11:	Status diagram of the process data watchdog	3-36
Figure 3-12:	Assignment of the process data between the local bus and the computer systems	3-58
Figure 3-13:	Using the macros for programming support	3-59
Figure 3-14:	Structure of the station for the example program	3-78
Figure 3-15:	Screenshot of the example program	3-79

Section 6

Figure 6-1:	Location of the data in input/output modules	6-7
Figure 6-2:	Location of the process data in dynamic tables	6-8

A 2 List of Tables

Section 1

Table 1-1:	Local status and diagnostic indicators	1-6
Table 1-2:	Connector assignment.....	1-8
Table 1-3:	Digital Input/Output Modules	1-9
Table 1-4:	Analog input/output modules	1-11
Table 1-5:	Special function modules.....	1-12
Table 1-6:	Motor terminal blocks.....	1-13
Table 1-7:	Power and segment terminals	1-14
Table 1-8:	Safety.....	1-15
Table 1-9:	Controller / CPU.....	1-15
Table 1-10:	Terminal point color-coding	1-18
Table 1-11:	Module color-coding	1-21
Table 1-12:	Potential jumper (see Figure 1-15)	1-28
Table 1-13:	Data jumper (see Figure 1-15).....	1-28
Table 1-14:	Diagnostic LEDs of the bus coupler.....	1-43
Table 1-15:	Diagnostic LED on the power terminal	1-44
Table 1-16:	Diagnostic LED on the segment terminal	1-44
Table 1-17:	Additional LED on supply terminals with fuse.....	1-44
Table 1-18:	Diagnostic LED of the I/O modules.....	1-45
Table 1-19:	Status LEDs of the I/O terminals	1-46
Table 1-20:	Overview: shield connection of analog sensors/actuators	1-56
Table 1-21:	Overview of the connections used for digital input modules	1-67
Table 1-22:	Overview of the connections used for digital output modules	1-67

Section 2

Table 2-1:	Sequence displayed after the device is switched on	2-3
Table 2-2:	During startup/operation:	2-44
Table 2-3:	Additional information:	2-44
Table 2-4:	During firmware update:	2-44
Table 2-5:	Boot loader error messages:	2-45
Table 2-6:	Firmware error messages:.....	2-45
Table 2-7:	Priority of the error messages.....	2-47
Table 2-8:	Differences on the display in DDI mode and Modbus/TCP mode	2-47

Section 3

Table 3-1:	Possible combination of the modes and their effect	3-9
Table 3-2:	System parameters for the "Set_Value" service (750 _{hex})	3-13
Table 3-3:	Overview of the functions in the DDI	3-15
Table 3-4:	Monitoring functions.....	3-33
Table 3-5:	Available fault response modes.....	3-33
Table 3-6:	Available fault response modes.....	3-44
Table 3-7:	Power-up sequence	3-44
Table 3-8:	Connection monitoring table	3-45
Table 3-9:	Driver software macros	3-60
Table 3-10:	Driver software messages	3-68

Section 4

Table 4-1:	Services available in both operating modes	4-3
Table 4-2:	Services available only in expert mode	4-4
Table 4-3:	System parameters.....	4-9
Table 4-4:	Available fault response modes.....	4-9
Table 4-5:	Supported error codes	4-43
Table 4-6:	Overview of error messages (according to error codes).....	4-50

Section 5

Table 5-1:	Supported PCP commands	5-5
------------	------------------------------	-----

Section 6

Table 6-1:	Modbus message format	6-4
Table 6-2:	Supported function codes	6-5
Table 6-3:	Modbus reference tables	6-6
Table 6-4:	Dynamic process data table	6-6
Table 6-5:	Applicable functions.....	6-9
Table 6-6:	Read multiple registers	6-10
Table 6-7:	Response to "read multiple registers"	6-10
Table 6-8:	Exception response to "read multiple registers"	6-10
Table 6-9:	Write multiple registers	6-11
Table 6-10:	Response to "write multiple registers"	6-11
Table 6-11:	Exception response to "write multiple registers"	6-12
Table 6-12:	Read coils	6-12
Table 6-13:	Response to "read coils"	6-12
Table 6-14:	Exception response to "read coils"	6-13
Table 6-15:	Read input discretes	6-13
Table 6-16:	Response to "read input discretes"	6-13
Table 6-17:	Exception response to "read input discretes"	6-14
Table 6-18:	Read input discretes	6-14
Table 6-19:	Response to "read input registers"	6-14
Table 6-20:	Exception response to "read input registers"	6-15
Table 6-21:	Write coils	6-15
Table 6-22:	Response to "write coils"	6-15
Table 6-23:	Exception response to "write coils"	6-15
Table 6-24:	Write single register	6-16
Table 6-25:	Response to "write single register"	6-16
Table 6-26:	Exception response to "write single register"	6-16
Table 6-27:	Read exception status	6-17
Table 6-28:	Answer to "read exception status"	6-17

Table 6-29:	Exception status data format	6-17
Table 6-30:	Exception responses	6-18
Table 6-31:	Write multiple coils.....	6-19
Table 6-32:	Response to "write multiple coils"	6-19
Table 6-33:	Exception response to "write multiple coils"	6-19
Table 6-34:	Read/write register.....	6-20
Table 6-35:	Answer to "read/write register"	6-20
Table 6-36:	Exception response to "read/write register"	6-20
Table 6-37:	Structure of the analog output table.....	6-22
Table 6-38:	Network interface command word	6-22
Table 6-39:	Structure of the input discretes table	6-23
Table 6-40:	Status word.....	6-23
Table 6-41:	Structure of the analog input table	6-23
Table 6-42:	Registers.....	6-24
Table 6-43:	Fault table	6-25
Table 6-44:	Monitoring functions.....	6-25
Table 6-45:	Available fault response modes.....	6-26
Table 6-46:	Power up sequence	6-27
Table 6-47:	Connection monitoring table	6-28
Table 6-48:	PCP registers.....	6-30