PHOENIX CONTACT

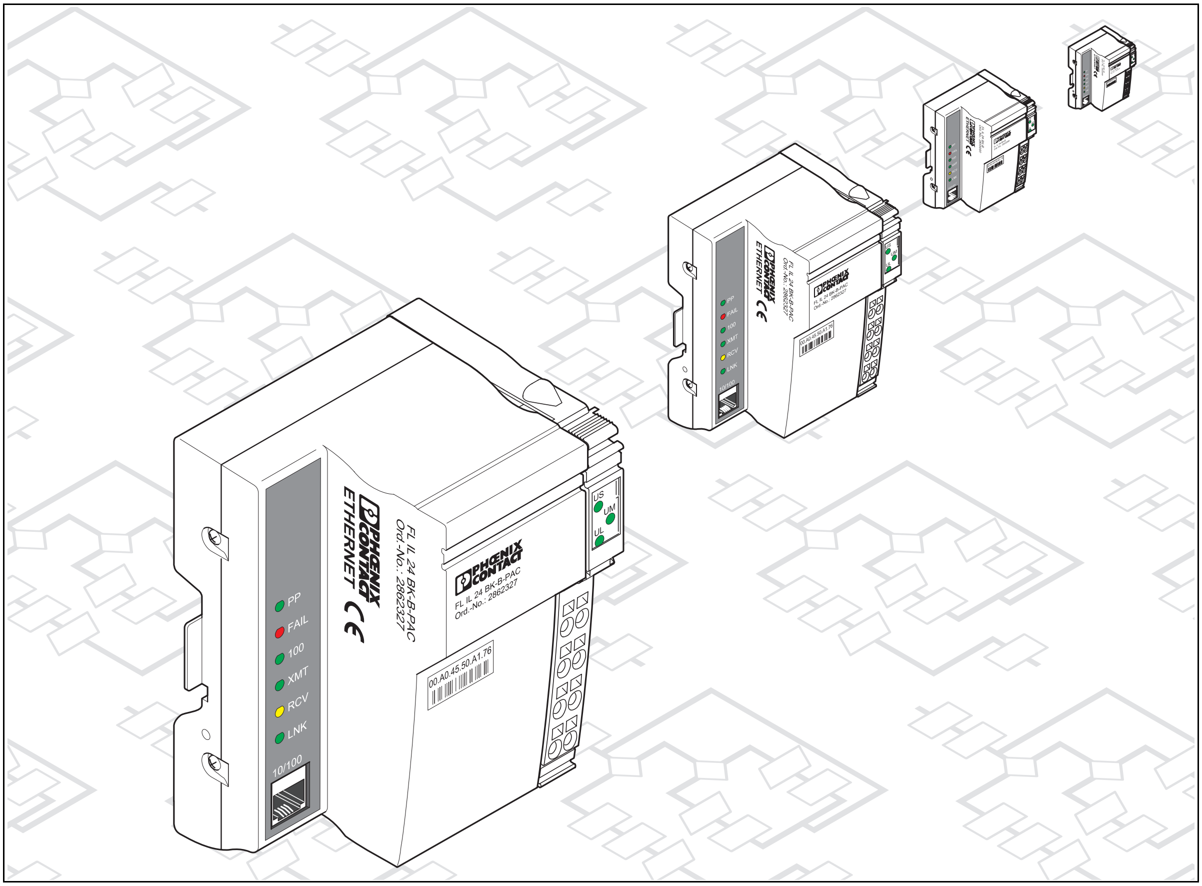
INNOVATION IN INTERFACE

User Manual

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

Designation: FL IL 24 BK-B UM E

Order No.: 26 98 76 6



Factory Line

User Manual

Hardware and Firmware Manual for the Factory Line Ethernet Bus Coupler FL IL 24 BK-B-PAC

Designation:	FL IL 24 BK-B UM E
Revision:	03
Order No.:	26 98 76 6

This user manual is valid for:
FL IL 24 BK-B-PAC with Firmware Version 1.10

654403

PHŒNIX CONTACT

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

☞ 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 *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.

We Are Interested in Your Opinion

We are constantly attempting to improve the quality of our manuals.

Should you have any suggestions or recommendations for improvement of the contents and layout of our manuals, we would appreciate it if you would send us your comments. Please use the universal fax form at the end of the manual for this.

Statement of Legal Authority

Warning
The FL IL 24 BK-B-PAC module is designed exclusively for SELV operation according to IEC 950/EN 60950/VDE 0805.

Shielding
The shielding ground of the connected twisted pair cables 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 *http://www.phoenixcontact.com or http://www.factoryline.de*. The Phoenix Contact homepage is 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 the 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 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. Until a new manual revision is published, any updates or changes will be documented on the Internet at http://www.phoenixcontact.com or *http://www.factoryline.de* .

Table of Contents

1 FL IL 24 BK-B-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-B-PAC Bus Coupler 1-5

1.3 Local Status and Diagnostic Indicators 1-6

1.4 Connecting the Supply Voltage 1-7

1.5 Connector Assignment 1-8

1.6 Supported Inline Modules 1-9

1.7 Basic Structure of Low-Level Signal Modules 1-14

1.7.1 Electronics Base 1-15

1.7.2 Connectors 1-16

1.8 Function Identification and Labeling 1-20

1.9 Dimensions of Low-Level Signal Modules 1-24

1.10 Electrical Potential and Data Routing 1-27

1.11 Circuits Within an Inline Station and Provision of the Supply Voltages 1-29

1.11.1 Supply of the Ethernet Bus Coupler 1-30

1.11.2 Logic Circuit U_L 1-30

1.11.3 Analog Circuit U_{ANA} 1-31

1.11.4 Main Circuit U_M 1-32

1.11.5 Segment Circuit 1-33

1.12 Voltage Concept 1-35

1.13 Diagnostic and Status Indicators 1-42

1.13.1 LEDs on the Ethernet Bus Coupler 1-42

1.13.2 Supply Terminal Indicators 1-44

1.13.3 Input/Output Module Indicators 1-46

1.13.4 Indicators on Other Inline Modules 1-47

1.14 Mounting/Removing Modules and Connecting Cables 1-48
1.14.1 Installation Instructions 1-48
1.14.2 Mounting and Removing Inline Modules 1-48
1.14.3 Mounting 1-49
1.14.4 Removal 1-51
1.14.5 Replacing a Fuse 1-53
1.15 Grounding an Inline-Station 1-55
1.15.1 Shielding an Inline Station 1-57
1.15.2 Shielding Analog Sensors and Actuators 1-57
1.16 Connecting Cables 1-61
1.16.1 Connecting Unshielded Cables 1-61
1.16.2 Connecting Shielded Cables Using the Shield Connector 1-63
1.17 Connecting the Power Supply 1-66
1.17.1 Power Terminal Supply 1-67
1.17.2 Provision of the Segment Voltage Supply at Power Terminals 1-68
1.17.3 Requirements Regarding the Voltage Supplies 1-68
1.18 Connecting Sensors and Actuators 1-68
1.18.1 Connection Methods for Sensors and Actuators 1-68
1.18.2 Examples of Connections for Digital I/O Modules 1-70

2 Startup/Operation 2-3
2.1 Firmware Startup 2-3
2.1.1 Sending BootP Requests 2-3
2.2 Assigning an IP Address Using the Factory Manager 2-3
2.2.1 BootP 2-4
2.3 Manual Addition of Devices Using The Factory Manager 2-4
2.4 Selecting IP Addresses 2-5
2.4.1 Possible Address Combinations 2-6
2.4.2 Subnet Masks 2-7
2.4.3 Structure of the Subnet Mask 2-8
2.5 Web-Based Management 2-9

2.5.1 Calling Web-Based Management (WBM) 2-9
2.5.2 Structure of the Web Pages 2-10
2.5.3 Layout of the Web Pages 2-11
2.5.4 Password Protection 2-11
2.5.5 Process Data Access via XML 2-12

2.6 Factory Line I/O Configurator..2-17
2.6.1 Factory Line I/O Browser 2-17
2.6.2 OPC Configurator 2-18

3 Driver Software 3-3

3.1 Documentation...3-3
3.1.1 Hardware and Software User Manual 3-3

3.2 The Software Structure 3-3
3.2.1 Ethernet / Inline Bus Terminal 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 3-7
3.4.1 Position of the Process Data (Example) 3-8

3.5 Startup Behavior of the Bus Terminal 3-9
3.5.1 Plug & Play Mode 3-9
3.5.2 Expert Mode 3-10
3.5.3 Possible Combination of Modes 3-10
3.5.4 Startup Diagrams of the Bus Coupler 3-11
3.5.5 Changing and Starting a Configuration in P&P Mode .3-13

3.6 Changing a Reference Configuration Using the Software 3-14
3.6.1 Effects of Expert Mode 3-14
3.6.2 Changing a Reference Configuration 3-14

3.7 Description of the Device Driver Interface (DDI).......................3-16
3.7.1 Introduction 3-16
3.7.2 Overview 3-17
3.7.3 Working Method of the Device Driver Interface 3-17
3.7.4 Description of the Functions of the
Device Driver Interface 3-20

3.8 Monitoring Function 3-35
3.8.1 Process Data Monitoring / Process Data Watchdog 3-37
3.8.2 Connection Monitoring (Host Checking) 3-39
3.8.3 Data Interface (DTI) Monitoring 3-42
3.8.4 I/O Fault Response Mode 3-46
3.8.5 Handling the NetFail Signal / Testing With ETH_SetNetFail 3-48
3.9 IN Process Data Monitoring 3-56
3.10 Notification Mode 3-60
3.11 Programming Support Macros 3-63
3.11.1 Introduction 3-63
3.12 Description of the Macros 3-65
3.12.1 Macros for Converting the Data Block of a Command 3-66
3.12.2 Macros for Converting the Data Block of a Message ..3-69
3.12.3 Macros for Converting Input Data 3-72
3.12.4 Macros for Converting Output Data 3-73
3.13 Diagnostic Options of the Driver Software 3-76
3.13.1 Introduction 3-76
3.14 Positive Messages 3-77
3.15 Error Messages 3-78
3.15.1 General Error Messages 3-78
3.15.2 Error Messages When Opening a Data Channel 3-80
3.15.3 Error Messages When Transmitting Messages/Commands 3-81
3.15.4 Error Messages When Transmitting Process Data 3-83
3.16 Example Program 3-86
3.16.1 Demo Structure Startup 3-86
3.16.2 Example Program Source Code 3-88
4 Firmware Services 4-3
4.1 Overview 4-3
4.1.1 Services That can be Used in

Every Operating Mode 4-3
4.1.2 Services That are Only Available in Expert Mode 4-4

4.2 Notes on Service Descriptions...4-4
4.2.1 Service "Name of the Service" 4-5

4.3 Services for Parameterizing the Controller Board.......................4-8
4.3.1 "Control_Parameterization" Service 4-8
4.3.2 "Set_Value" Service 4-10
4.3.3 "Read_Value" Service 4-12
4.3.4 "Initiate_Load_Configuration" Service 4-14
4.3.5 "Load_Configuration" Service 4-16
4.3.6 "Terminate_Load_Configuration" Service 4-20
4.3.7 "Read_Configuration" Service 4-22
4.3.8 "Complete_Read_Configuration" Service 4-29
4.3.9 "Delete_Configuration" Service 4-32
4.3.10 "Create_Configuration" Service 4-33
4.3.11 "Activate_Configuration" Service 4-36
4.3.12 "Control_Device_Function" Service 4-38
4.3.13 "Reset_Controller_Board" Service 4-40

4.4 Services for Direct INTERBUS Access....................................4-42
4.4.1 "Start_Data_Transfer" Service 4-42
4.4.2 "Alarm_Stop" Service 4-44

4.5 Diagnostic Services..4-45
4.5.1 "Get_Error_Info" Service 4-45
4.5.2 "Get_Version_Info" Service 4-52

4.6 Error Messages for Firmware Services:....................................4-56
4.6.1 Overview 4-56
4.6.2 Positive Messages 4-57
4.6.3 Error Messages 4-57

5 Modbus/TCP Protocol...5-3

5.1 Modbus Protocol 5-4
5.1.1 Modbus Connections 5-4
5.1.2 Modbus Interface 5-4

5.1.3 Modubus Conformity Classes 5-4
5.1.4 Modbus Message Format 5-5
5.1.5 Modbus Byte Sequence 5-5
5.1.6 Modbus Bit Sequence 5-6

5.2 Modbus Function Codes 5-6

5.3 Modbus Table 5-6
5.3.1 Example: Position of the Input / Output Data 5-8

5.4 Executable Functions..5-9

5.5 Supported Function Codes 5-10
5.5.1 Read Multiple Registers 5-10
5.5.2 Write Multiple Registers 5-12
5.5.3 Read Coils 5-13
5.5.4 Read Input Discretes 5-14
5.5.5 Read Input Registers 5-15
5.5.6 Write Coil 5-16
5.5.7 Write Single Register 5-17
5.5.8 Read Exception Status 5-19
5.5.9 Data Format of the Exception Status 5-19
5.5.10 Exception Responses 5-20
5.5.11 Write Multiple Coils 5-21
5.5.12 Read/Write Register 5-22

5.6 Reserved Registers for
Command and Status Words...5-24
5.6.1 Command Word 5-24
5.6.2 Status Word 5-25
5.6.3 Diagnostics Using the Analog Input Table 5-26
5.6.4 Error Table 5-26

5.7 Monitoring 5-28

5.8 Modbus Monitoring..5-29

5.9 I/O Fault Response Mode 5-30
5.9.1 The Power Up Table 5-31
5.9.2 The Connection Monitoring Table 5-33

6 Technical Data 6-3

6.1 Ordering Data 6-11

Section 1

This section informs you about

– the basic structure of low-level signal modules
– the arrangement of the diagnostic and status indicators
– the potential and data routing

FL IL 24 BK-B-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-B-PAC Bus Coupler 1-5

1.3 Local Status and Diagnostic Indicators 1-6

1.4 Connecting the Supply Voltage 1-7

1.5 Connector Assignment 1-8

1.6 Supported Inline Modules 1-9

1.7 Basic Structure of Low-Level Signal Modules 1-14

1.7.1 Electronics Base 1-15

1.7.2 Connectors 1-16

1.8 Function Identification and Labeling 1-20

1.9 Dimensions of Low-Level Signal Modules 1-24

1.10 Electrical Potential and Data Routing 1-27

1.11 Circuits Within an Inline Station and Provision of the Supply Voltages 1-29

1.11.1 Supply of the Ethernet Bus Coupler 1-30

1.11.2 Logic Circuit U_L 1-30

1.11.3 Analog Circuit U_{ANA} 1-31

1.11.4 Main Circuit U_M 1-32

1.11.5 Segment Circuit 1-33

1.12 Voltage Concept 1-35

1.13 Diagnostic and Status Indicators 1-42
1.13.1 LEDs on the Ethernet Bus Coupler 1-42
1.13.2 Supply Terminal Indicators 1-44
1.13.3 Input/Output Module Indicators 1-46
1.13.4 Indicators on Other Inline Modules 1-47
1.14 Mounting/Removing Modules and Connecting Cables 1-48
1.14.1 Installation Instructions 1-48
1.14.2 Mounting and Removing Inline Modules 1-48
1.14.3 Mounting 1-49
1.14.4 Removal 1-51
1.14.5 Replacing a Fuse 1-53
1.15 Grounding an Inline-Station 1-55
1.15.1 Shielding an Inline Station 1-57
1.15.2 Shielding Analog Sensors and Actuators 1-57
1.16 Connecting Cables 1-61
1.16.1 Connecting Unshielded Cables 1-61
1.16.2 Connecting Shielded Cables Using the Shield Connector 1-63
1.17 Connecting the Power Supply 1-66
1.17.1 Power Terminal Supply 1-67
1.17.2 Provision of the Segment Voltage Supply at Power Terminals 1-68
1.17.3 Requirements Regarding the Voltage Supplies 1-68
1.18 Connecting Sensors and Actuators 1-68
1.18.1 Connection Methods for Sensors and Actuators 1-68
1.18.2 Examples of Connections for Digital I/O Modules 1-70

1 FL IL 24 BK-B-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)
- Up to 63 other Inline modules can be connected (process data channel)
- Flexible installation system for Ethernet
- IP parameter setting via BootP
- DDI software interface (Device Driver Interface) and Modbus/TCP
- 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.

Software by Phoenix Contact Required for Process Data Operation

Table 1-1 Software for Process Data Operation

Operation	Software
DDI (Read and Write)	DDI driver
Modbus/TCP (Read and Write)	---
OPC (Read and Write)	OPC Server ≥ 2.01 FL IO Browser FL IO Configurator
XML (Read only)	---

Front View of the FL IL 24 BK-B-PAC

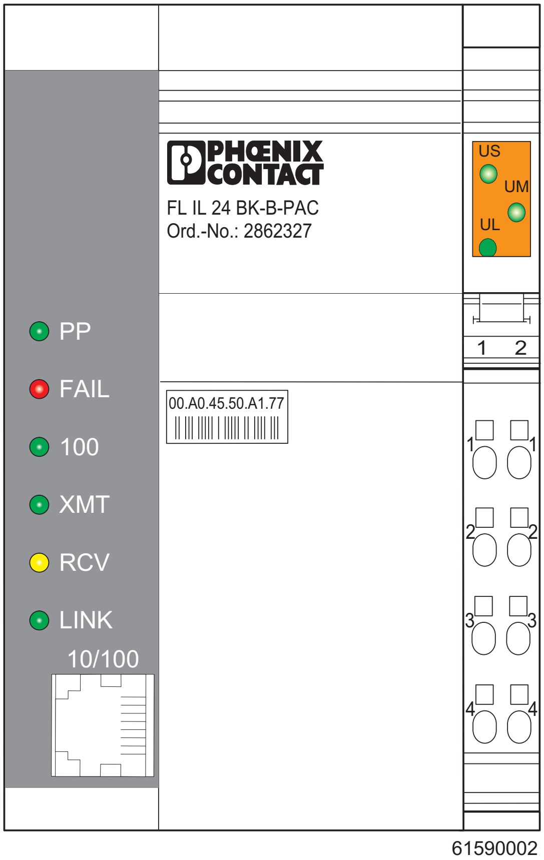


Figure 1-1 Front view of the FL IL 24 BK-B-PAC

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

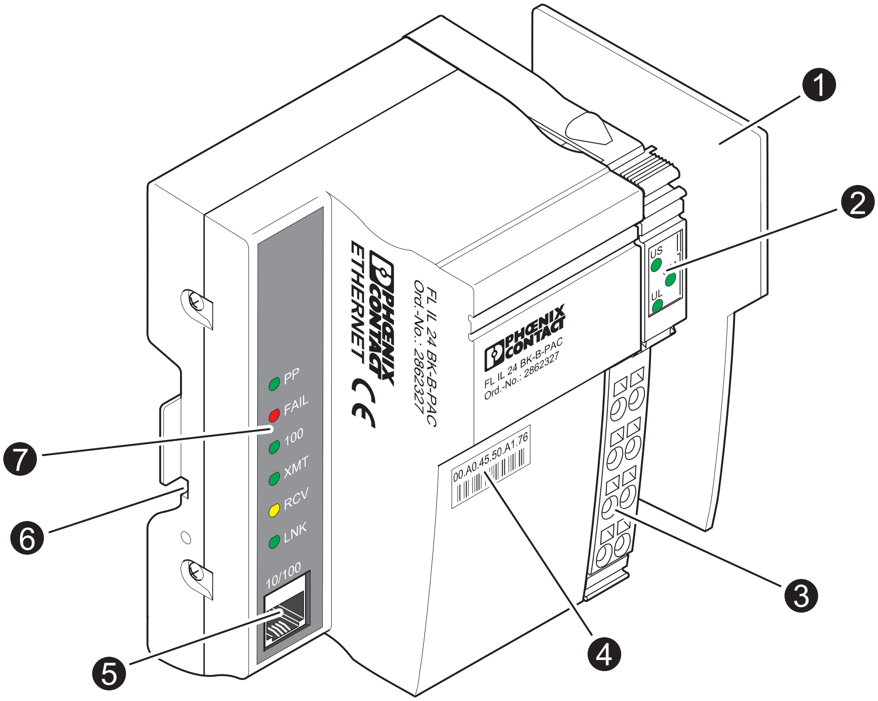


Figure 1-2 Structure of the FL IL 24 BK-B-PAC Bus Coupler

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

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 LED status and diagnostic indicators

1.3 Local Status and Diagnostic Indicators

Table 1-2 Local Status and Diagnostic Indicators

Des.	Color	Status	Meaning
Electronics module			
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 is present
		OFF	24 V segment supply is not present
Ethernet Port			
PP	Green	ON	Plug & play mode is activated
		OFF	Plug & play mode is not activated
FAIL	Red	ON	The firmware has detected an error
		OFF	The firmware has not detected any error
100	Green	ON	Operation at 100 Mbps (if LNK LED active)
		OFF	Operation at 10 Mbps (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

The bus coupler can be reset by switching the supply voltage off and on again.

1.4 Connecting the Supply Voltage

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

Typical Connection of the Supply Voltage

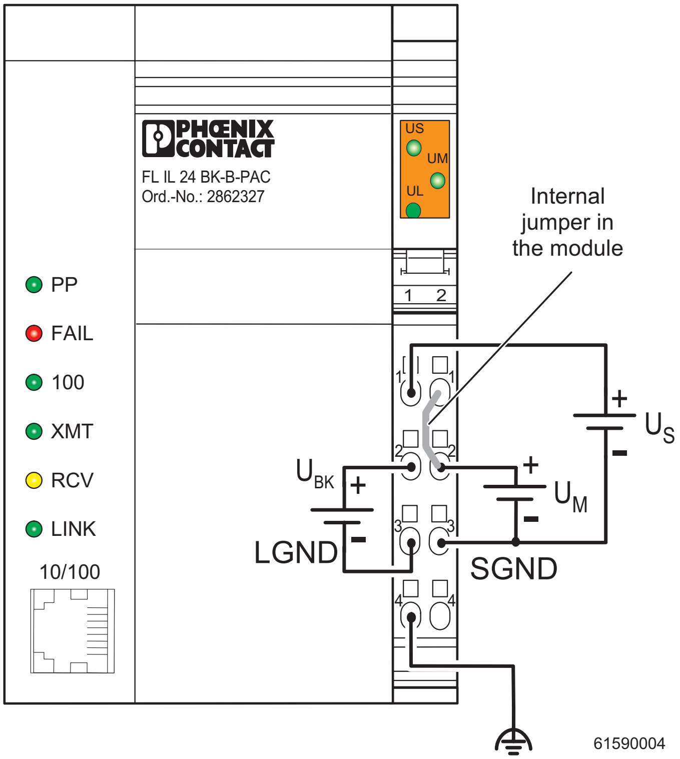


Figure 1-3 Typical connection of the supply voltage

1.5 Connector Assignment

Table 1-3 Connector assignment

Terminal Point	**Assignment**		**Wire Color/Remark**
Connector	**Power Connector**		
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 power	The main power 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.

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

1.6 Supported Inline Modules

Table 1-4 Digital I/O Modules

Designation	Features	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, acc. 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, acc. 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 connection, 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
IB IL 24 DO 2-PAC	2 outputs, 500 mA, 4-wire termination, 24 V DC	28 61 47 0

Designation (Contd.)	Features	Order No.
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, extensible diagnostics, parameterizable outputs	27 42 59 9
IB IL 24 EDO 2-PAC	2 outputs, 500 mA, 4-wire termination, 24 V DC, extensible diagnostics, parameterizable 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 mA, 3-wire termination	27 42 69 6
IB IL DO 4 AC-1A-PAC	1 output, 12 V - 253 V AC, 1 mA, 3-wire termination	28 61 65 8
IB IL 24/230 DOR 1/W	1 PDT relay contact, 5 V - 253 V AC, 3 A	28 36 43 4
IB IL 24/230 DOR 1/W-PAC	1 PDT relay contact, 5 V - 253 V AC, 3 A	28 61 88 1
IB IL 24/230 DOR 1/W-PC	1 PDT 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 PDT relay contact, 5 V - 253 V AC, 3 A for inductive and capacitive loads	28 62 17 8
IB IL 24/230 DOR 4/W	4 PDT relay contacts, 5 V - 253 V AC, 3 A	28 36 42 1
IB IL 24/230 DOR 4/W-PAC	4 PDT relay contacts, 5 V - 253 V AC, 3 A	28 61 87 8

Designation (Contd.)	Features	Order No.
IB IL 24/230 DOR 4/W-PC	4 PDT 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 PDT 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 relais PDT 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 relais PDT contacts, 5 V - 50 V AC, 5 V - 120 V DC, 2 A	28 63 11 9

Table 1-5 Analog I/O Modules

Designation	Features	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 contact, 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 contact, 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 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 UTH	8 inputs, 8 outputs, control function	28 19 31 2
IB IL TEMPCON UTH-PAC	8 inputs, 8 outputs, control function	28 61 80 7

Designation (Contd.)	Features	Order No.
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 1-6 Special Function Modules

Designation	Features	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
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 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 linear measuring scales with impulse interface	28 19 23 1
IB IL IMPULSE IN-PAC	1 input for magnetostrictive linear measuring scales with impulse interface	28 61 85 2

Table 1-7 Motor Terminals

Designation	Features	Order No.
IB IL 24 TC	Thermistor terminal	27 27 41 7
IB IL 24 TC-PAC	Thermistor terminal	28 61 36 0

Table 1-8 Power and Segment Terminals

Designation	Features	Order No.
IB IL 24 PRW IN	Power terminal, 24 V DC	27 26 31 1
IB IL 24 PWR 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 PWR 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 PWR 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	24 V DC segment terminal with electronic fuse	27 27 78 9
IB IL 24 SEG-ELF-PAC	24 V DC segment terminal 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 GND potential distribution	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

1.7 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).

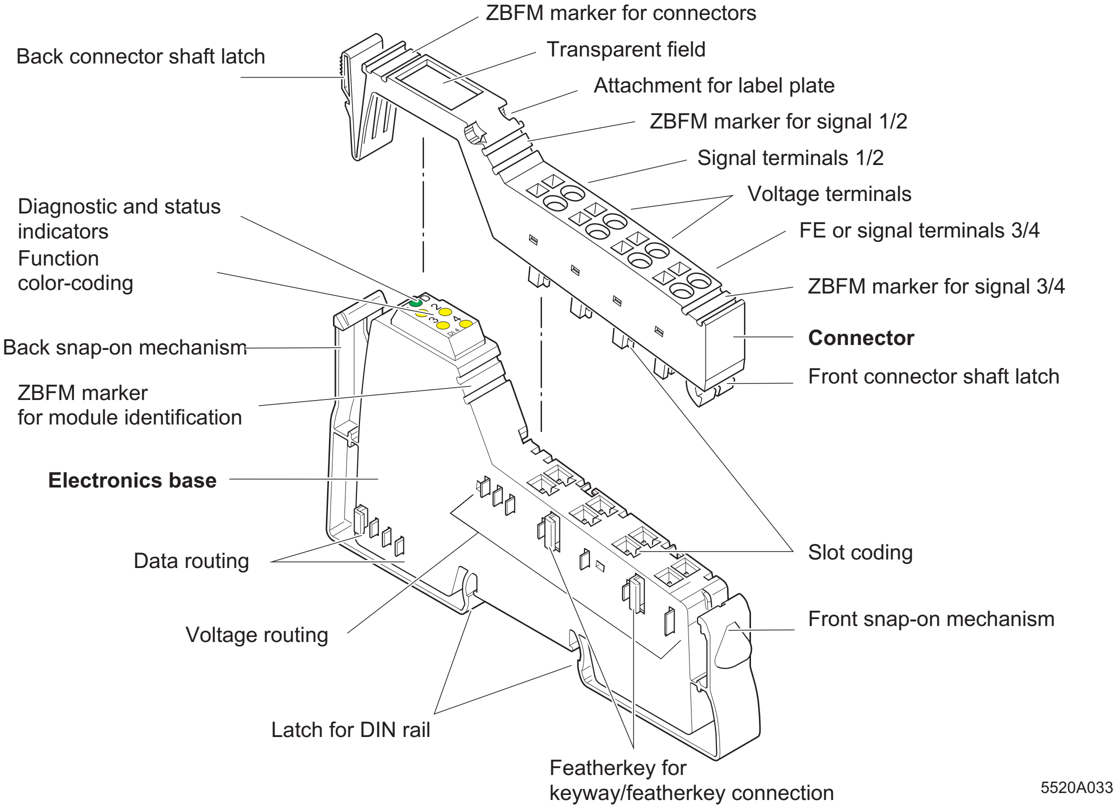

ZBFM marker for connectors
Back connector shaft latch
Transparent field
Attachment for label plate
ZBFM marker for signal 1/2
Signal terminals 1/2
Voltage terminals
Diagnostic and status indicators
Function color-coding
FE or signal terminals 3/4
ZBFM marker for signal 3/4
Back snap-on mechanism
Connector
Front connector shaft latch
ZBFM marker for module identification
Electronics base
Slot coding
Data routing
Front snap-on mechanism
Voltage routing
Latch for DIN rail
Featherkey for keyway/featherkey connection
5520A033

Figure 1-4 Basic structure of an Inline module

The most important of the components shown in Figure 1-4 are described in "Electronics Base" on page 1-15 and "Connectors" on page 1-16.

ZBFM: Zack marker strips, flat (see also Section "Function Identification and Labeling" on page 1-20)

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

1.7.1 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).

1.7.2 Connectors

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

Advantages

This snap-in-place 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:

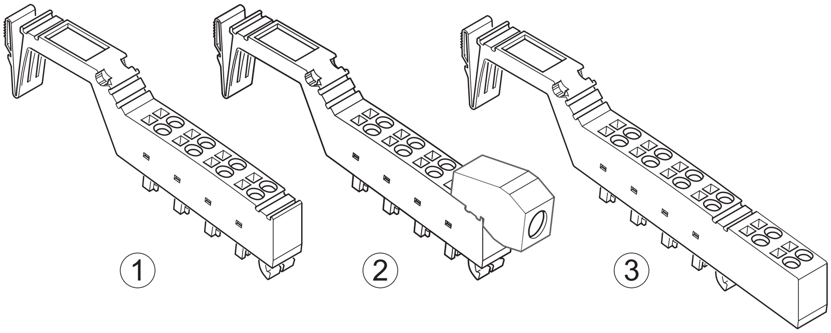


Figure 1-5 Connector types of Inline

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 1-18).

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 by a shield connection clamp rather than by 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 1-9 Terminal point color-coding

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

Internal structure of the connectors

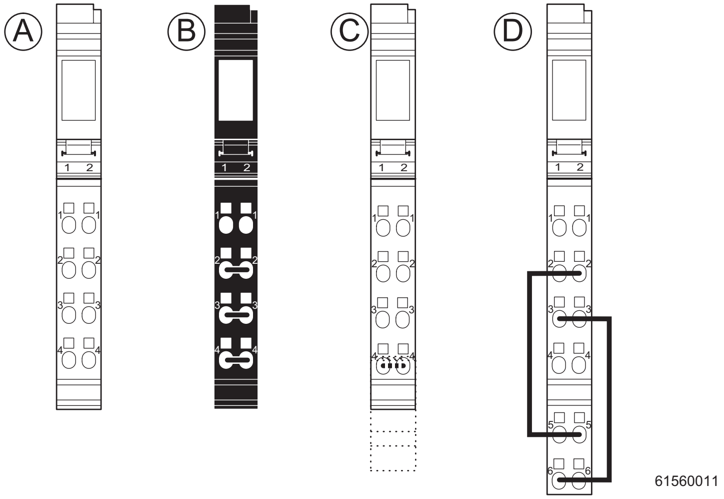


Figure 1-6 Internal structure of the connectors

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 integrated into 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 suitable connector onto a module. 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 place black connectors on supply terminals.
> When the terminal points are jumpered, power is carried through the jumpering in the connector and not through the printed circuit board of the module.

Connector coding

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

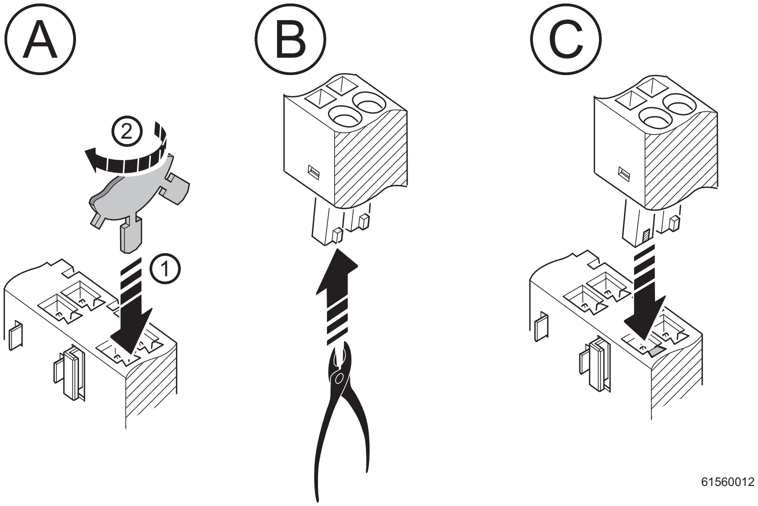


Figure 1-7 Connector keying

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

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

1.8 Function Identification and Labeling

Function identification

The modules are color-coded to enable visual identification of the functions (1 in Figure1-8).

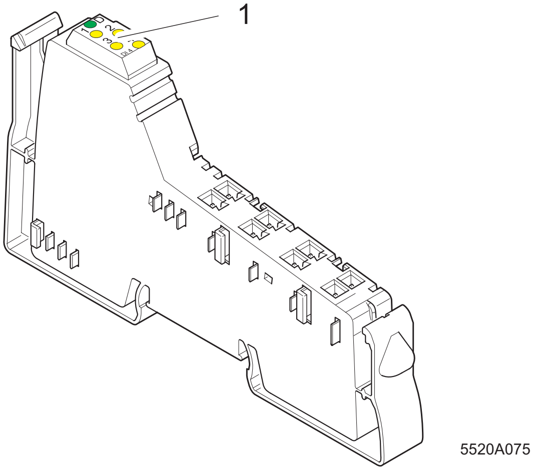


Figure1-8 Function identification

The following colors indicate the functions:

Table 1-10 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 1-17.

Labeling/ terminal point numbering

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

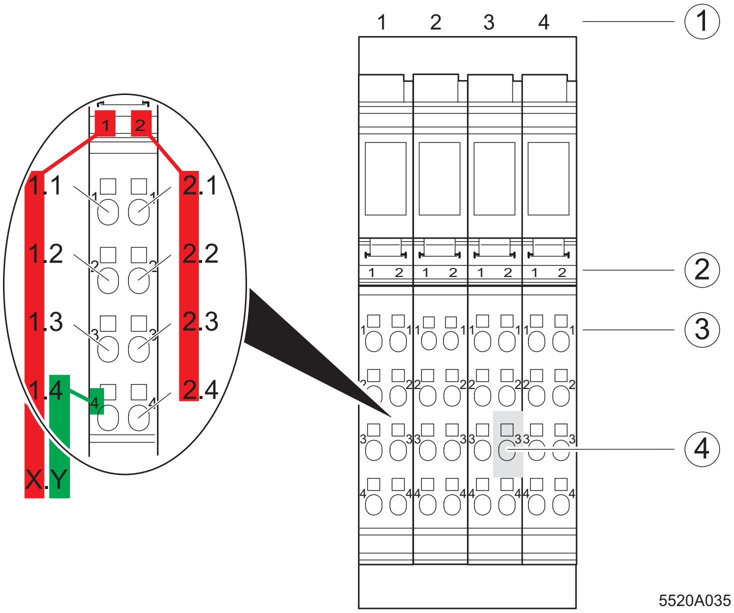


Figure1-9 Terminal point numbering

Slot/connector

The slots (connectors) on a base are numbered consecutively (1 in Figure1-9). 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 Figure1-9).

Y is the terminal point number in a row. It is directly indicated on the terminal point (3 in Figure1-9).

The precise designation for a point is thus specified by the slot and terminal point. The highlighted terminal point (4 in Figure1-9) 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.

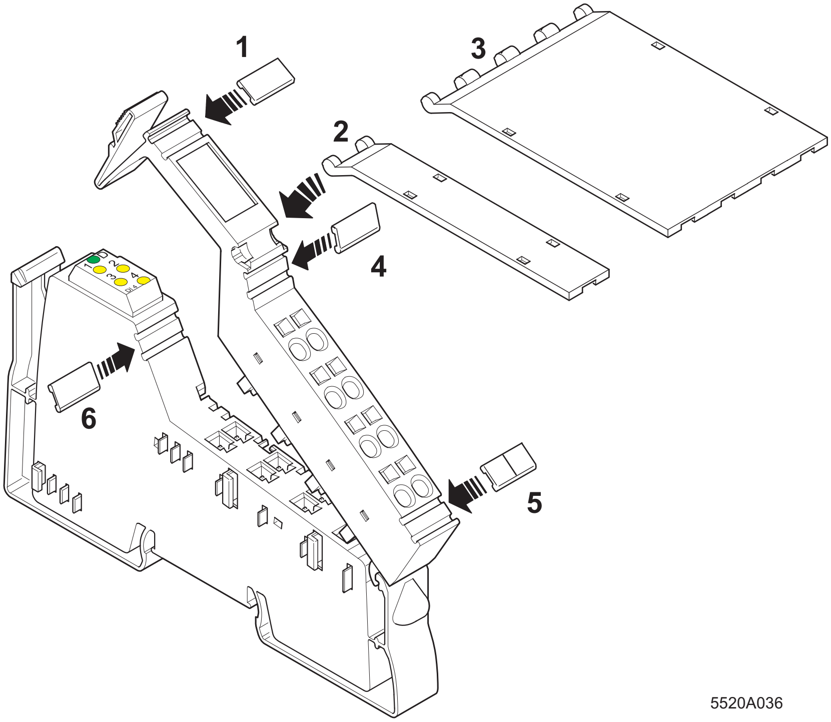


Figure1-10 Labeling of modules

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

1 Each connector can be labeled individually with Zack markers.

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. At each end 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.

1.9 Dimensions of Low-Level Signal Modules

Today, small I/O stations are frequently installed in 80 mm (3.150 in.) standard switch 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.

Electronics bases for low-level signal modules are available in three widths (12.2 mm, 24.4 mm and 48.8 mm [0.480 in., 0.961 in. and 1.921 in.]).

They take one (1), two (2) or four (4), 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 Figure1-14).

2-slot housing

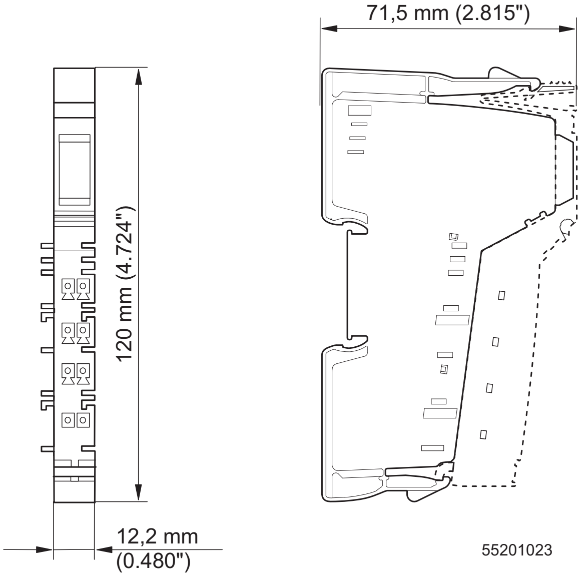


Figure1-11 Dimensions of the electronics bases (2-slot housing)

4-slot housing

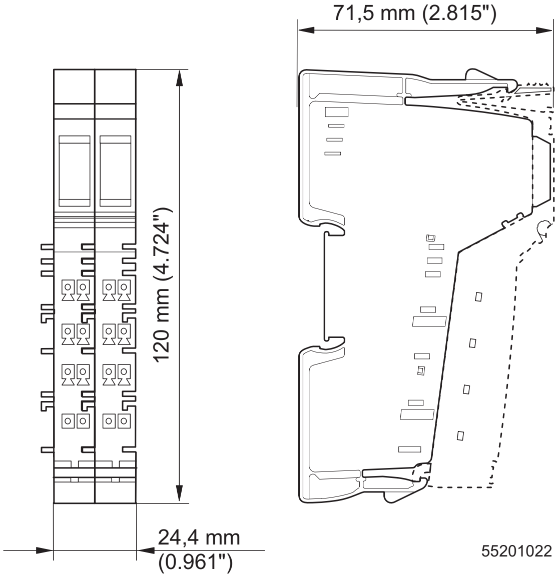


Figure1-12 Dimensions of the electronics bases (4-slot housing)

8-slot housing

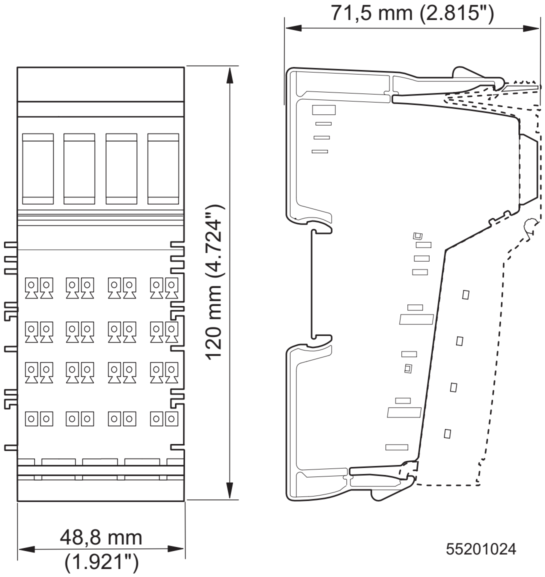


Figure1-13 Dimensions of the electronics bases (8-slot housing)

Connectors

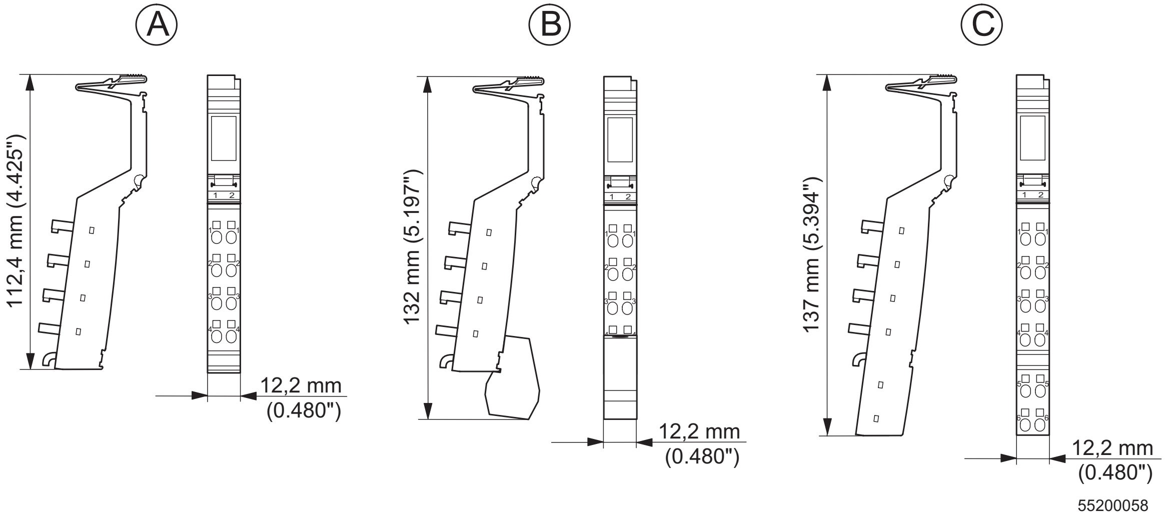


Figure1-14 Connector dimensions

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.

1.10 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. This 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 current circuits are realilzed within one station: the logic circuit and the I/O circuit.

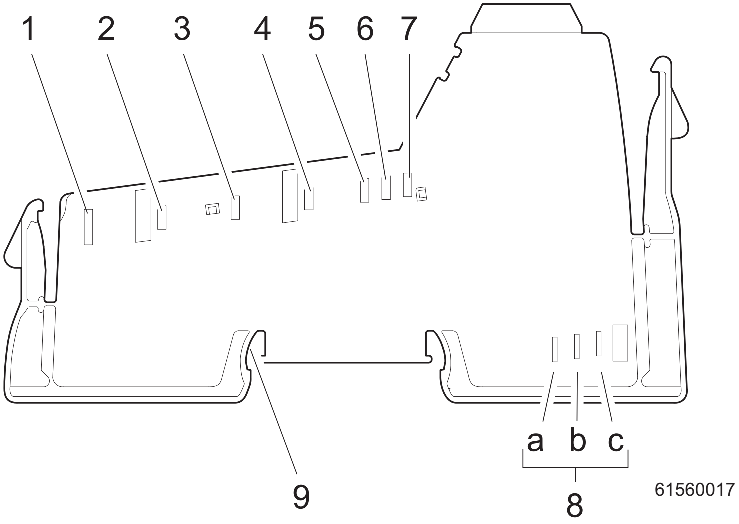


Figure 1-15 Potential and data routing

Table 1-11 Potential jumper (see Figure 1-15)

No.	Function		Meaning
1	FE	FE	Functional earth ground
2	SGND	SGND	Ground of segment and main supply
3	24 V	U_M	Supply for main circuit (if necessary with overload protection)
4	24 V	U_S	Supply for segment circuit (if necessary with overload protection) This jumper does not exist in power levels 120/230 V AC.
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 Figure1-9). 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 1-12 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

1.11 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.

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.

Load capacity of the jumper contacts

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 refer to the notes given in the module-specific data sheets.

1.11.1 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.

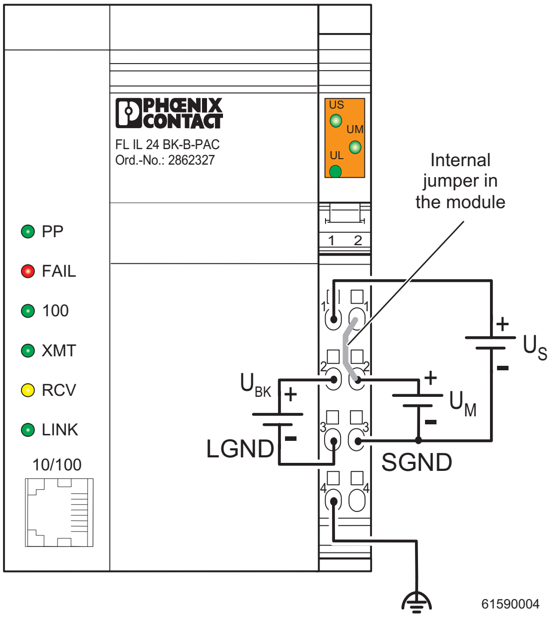


Figure 1-16 Typical connection of the supply voltage

1.11.2 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.

1.11.3 Analog Circuit U_{ANA}

The analog circuit with the supply for the analog modules (here also called 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.

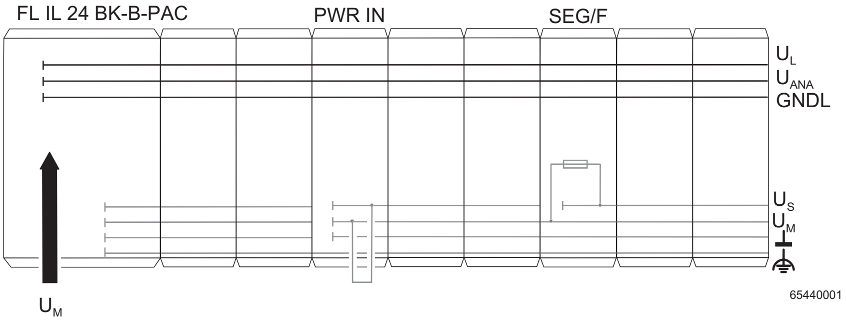


Figure 1-17 Logic and analog circuit

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

1.11.4 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.

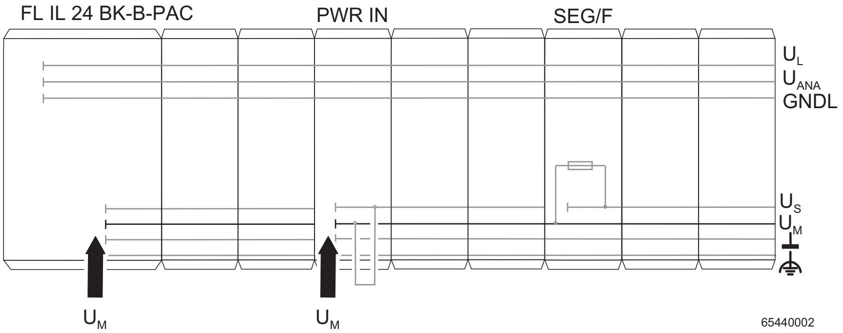


Figure 1-18 Main circuit

FL IL 24 BK-B	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 and in which case it is 24 V DC.

The power U_M can also be supplied via a power terminal. A power terminal **must** be used if one of the following occurs:

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.

1.11.5 Segment Circuit

The segment circuit or auxiliary circuit with 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 therefore 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 The segment voltage can be supplied at the Ethernet/Inline bus coupler or a power terminal.

2 The segment voltage can be tapped from the main voltage at the Ethernet/Inline bus coupler or a power terminal using a jumper or a switch.

3 A segment terminal can be used with a fuse. Within this terminal the segment voltage is automatically tapped from the main power.

4 A segment terminal can be used without a fuse and the segment voltage can be tapped 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.

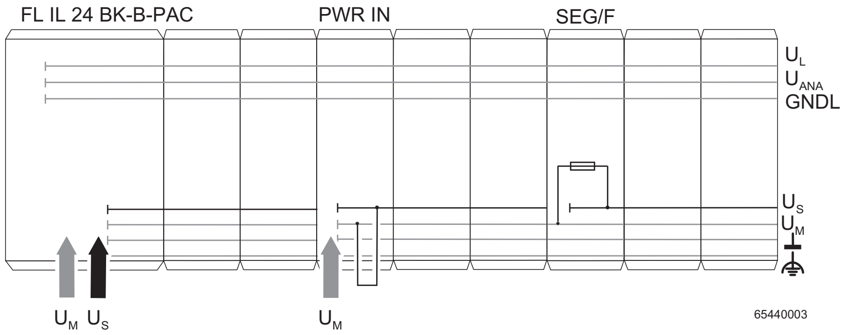


Figure 1-19 Segment circuit

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

1.12 Voltage Concept

The Ethernet bus coupler and the Inline local bus system have a defined potential 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 shielding is directly connected to functional earth ground. The device has two functional earth ground springs, which have contact with the DIN rail when they are snapped on. The springs are used to discharge interference, rather than serve as a 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 1-55). 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:

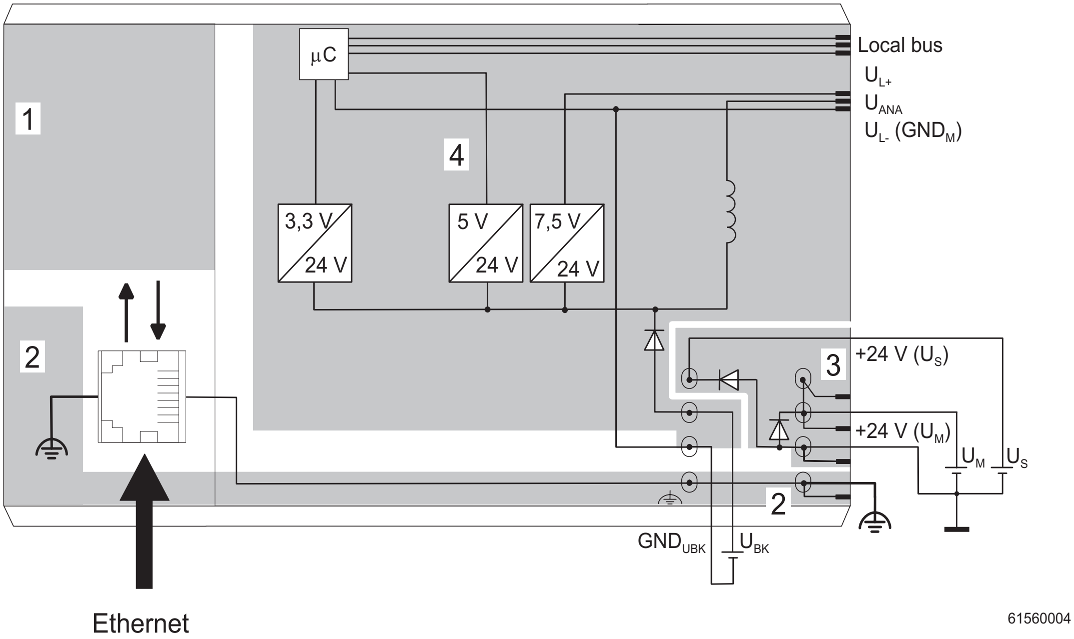


Figure 1-20 Potential areas in the bus coupler (two voltage supplies)

Voltage 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:

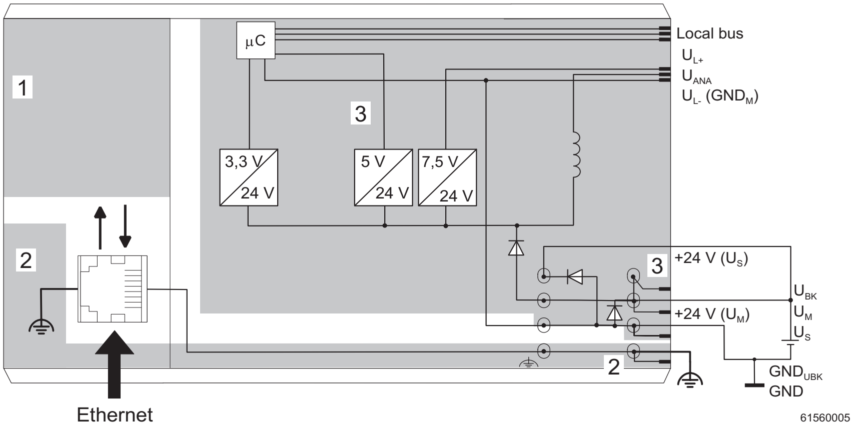


Figure 1-21 Bus coupler potentials (one voltage supply)

Voltage 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

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

Adjacent power connectors 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).

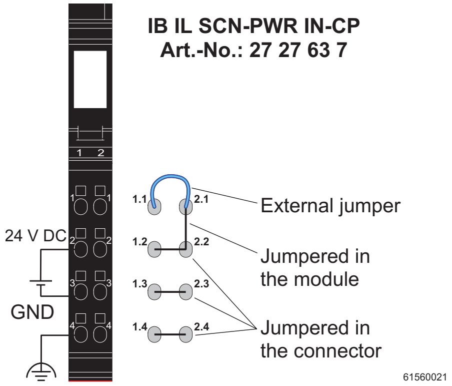
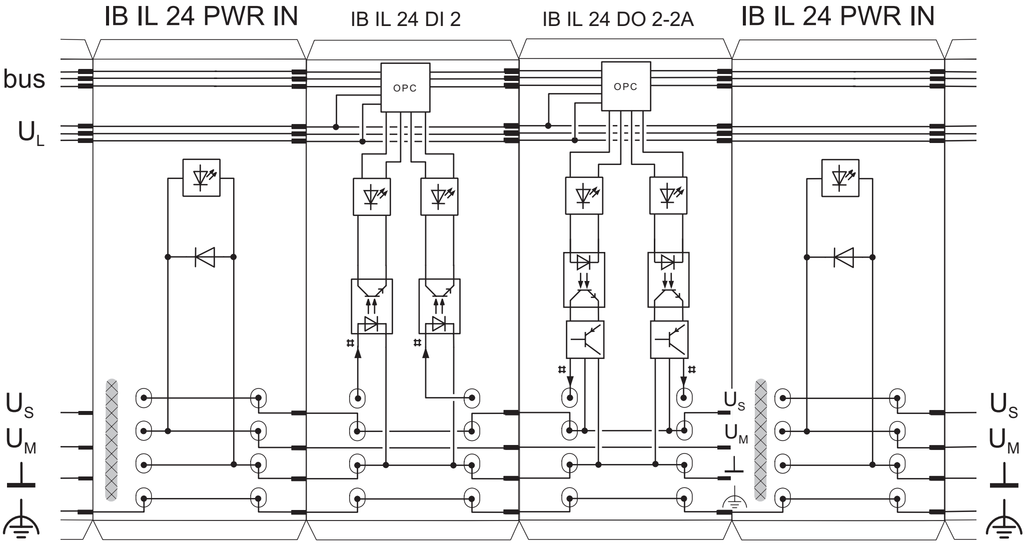


Figure 1-22 Power connector for supply from a single power supply unit

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 power 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 by means of the power terminal

The areas hatched in the figure XXXXX 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 so is also available to the next module.

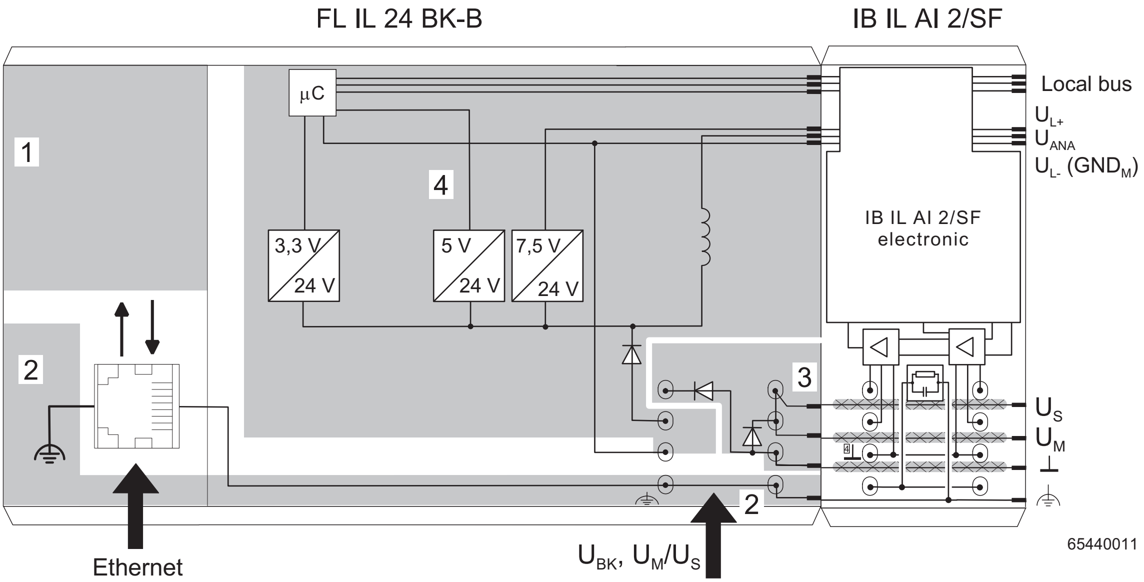


Figure 1-24 Electrical isolation between Ethernet bus coupler and analog module

The potential jumpers XXXXX hatched 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 by the analog circuit U_{ANA}.

Electrically isolated I/O supply

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, the 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.

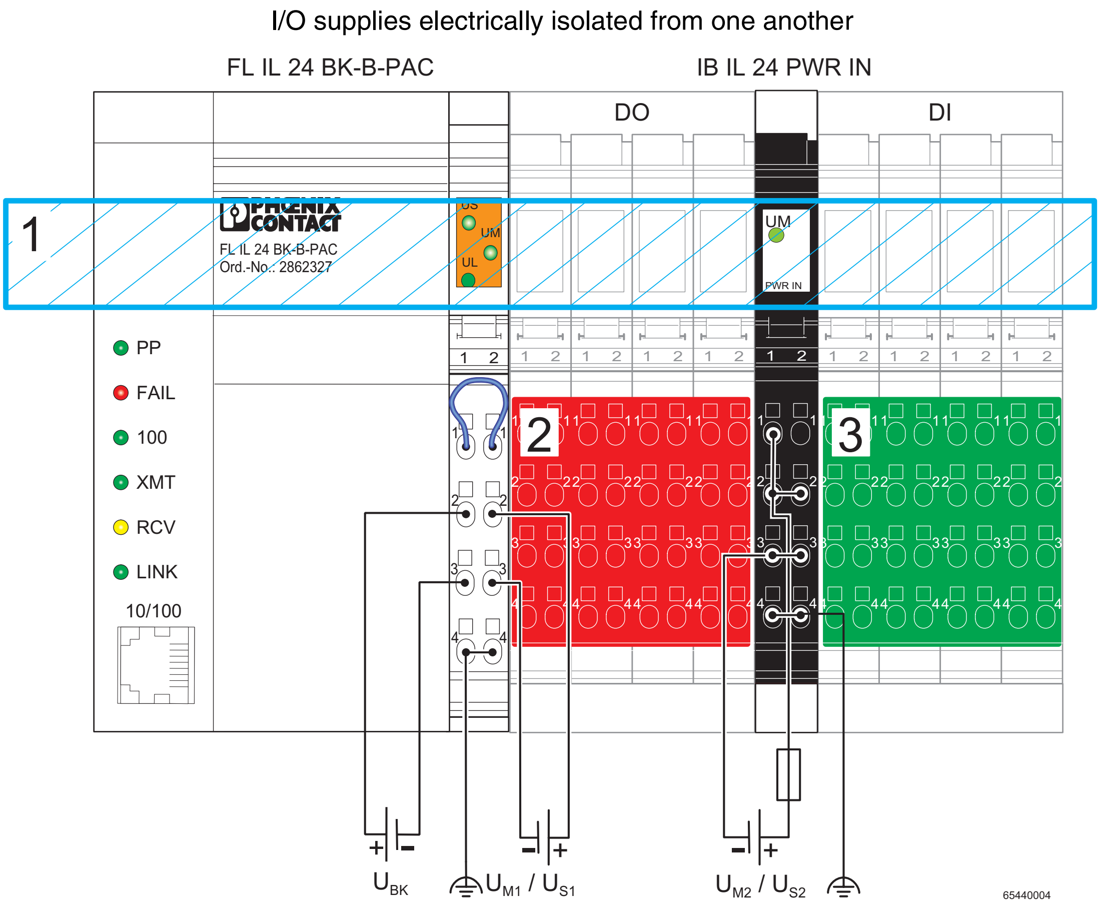


Figure 1-25 Structure of I/O supplies that are electrically isolated from one another

Potentials within the station:

1 Bus logic of the station

2 I/O (outputs)

3 I/O (inputs)

1.13 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.

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.

Refer to the module-specific data sheet for information about the diagnostic and status indicators on each module.

1.13.1 LEDs on the Ethernet Bus Coupler

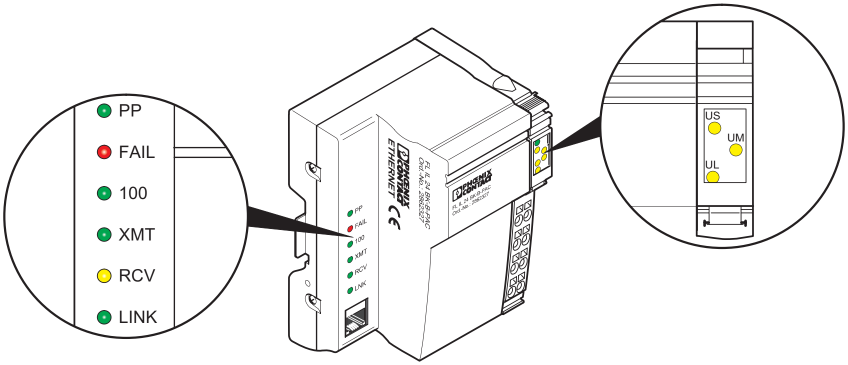


Figure 1-26 LEDs on the Ethernet bus coupler

Diagnostics

The following states can be read on the bus coupler:

Table 1-13 Diagnostic LEDs on the bus coupler

Des.	Color	Status	Meaning
Electronics Module			
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 is present
		OFF	24 V segment supply is not present
Ethernet Port			
PP	Green	ON	Plug & Play mode is activated
		OFF	Plug & Play mode is not activated
FAIL	Red	ON	The firmware has detected an error
		OFF	The firmware has not detected an error
100	Green	ON	Operation at 100 Mbps (if LNK LED active)
		OFF	Operation at 10 Mbps (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

1.13.2 Supply Terminal Indicators

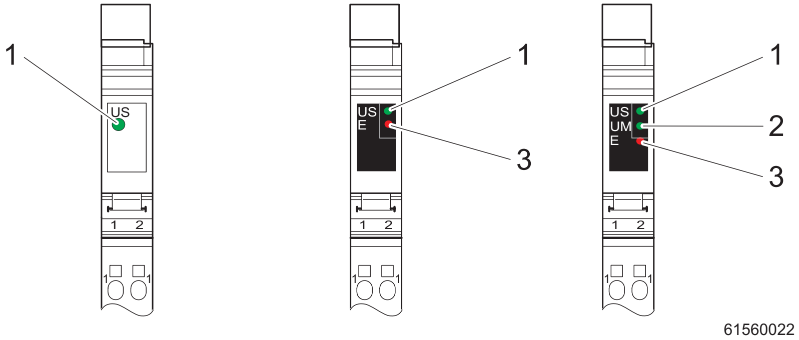


Figure 1-27 Possible indicators on supply terminals (segment terminal with and without fuse and power terminal)

Diagnostics

The following states can be read from the supply terminals:

Table 1-14 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 1-15 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 1-16 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.

1.13.3 Input/Output Module Indicators

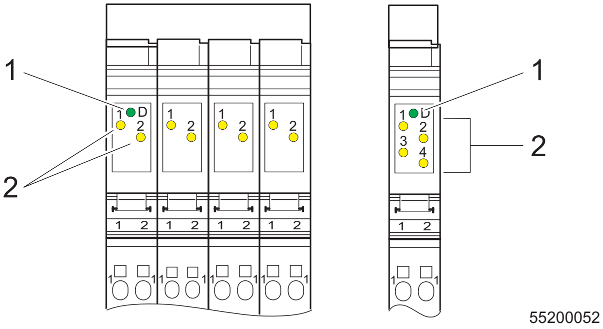


Figure 1-28 I/O module indicators

Diagnostics

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

Table 1-17 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 1-18 Status LEDs for 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 I/O

The assignment of a status LED and the corresponding I/O is given in the module-specific data sheet.

1.13.4 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.

1.14 Mounting/Removing Modules and Connecting Cables

1.14.1 Installation Instructions

To ensure installation is carried out correctly, please read "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.

1.14.2 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 clamp (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 interface module and needs not be ordered separately.

1.14.3 Mounting

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

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

☞ Ensure that **all** featherkeys and keyways on adjacent modules are interlocked (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 (C).

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

☞ The keyways of an electronics base do not continue on a connector. 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).

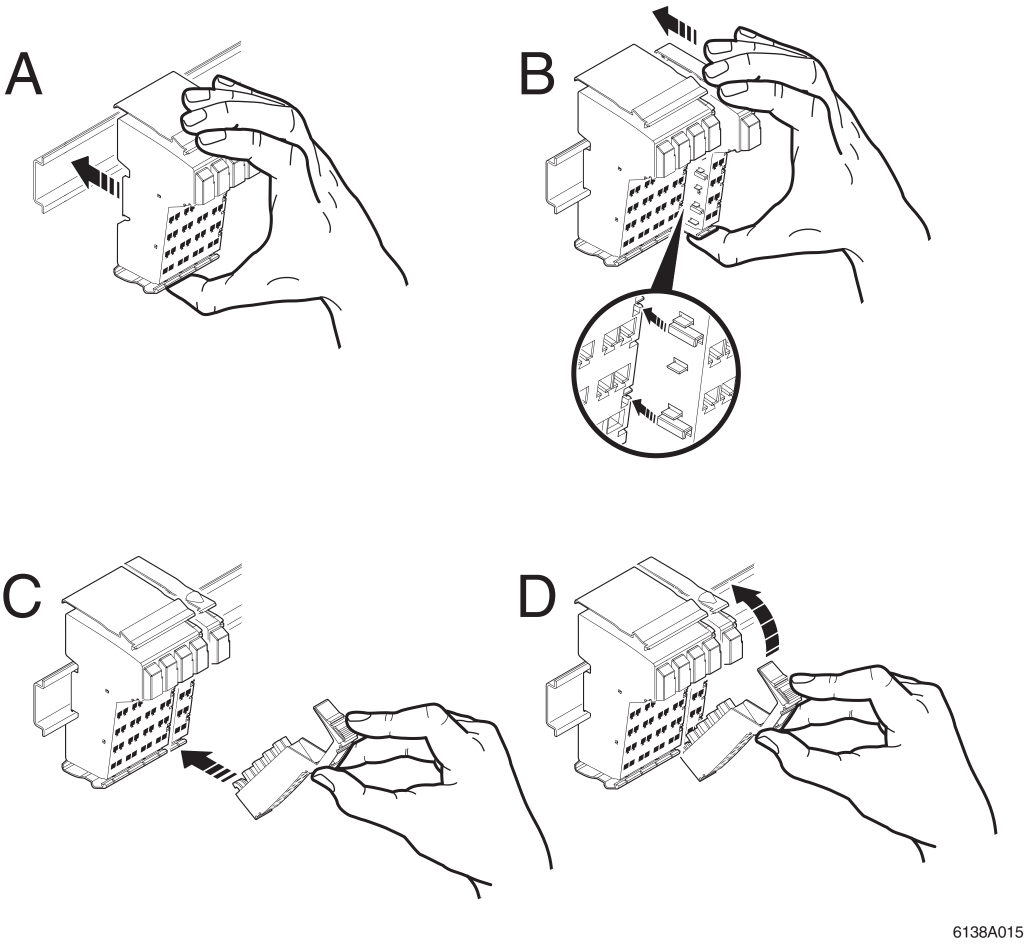


Figure 1-29 Snapping on a module

1.14.4 Removal

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

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

- Remove the connector (Fig. B).
- Remove the left-adjacent and right-adjacent connectors of the neighboring modules (C). This prevents the voltage routing featherkeys and the keyway/featherkey connection from being damaged. There also is more space available for accessing the module.
- Press the release mechanism, (D1 in Fig. D) and remove the electronics base from the DIN rail by pulling the base straight back (D2 in Fig. 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.

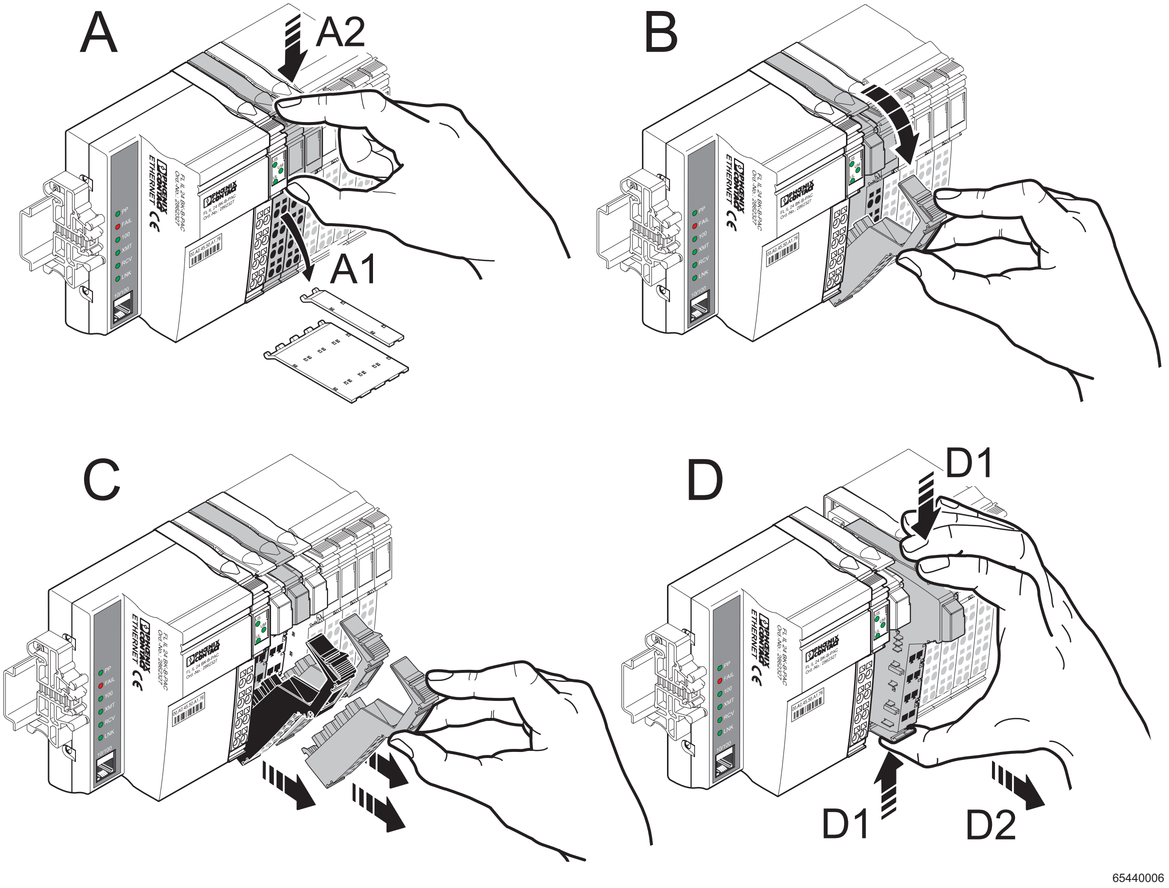


Figure 1-30 Removing a module

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).

1.14.5 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).

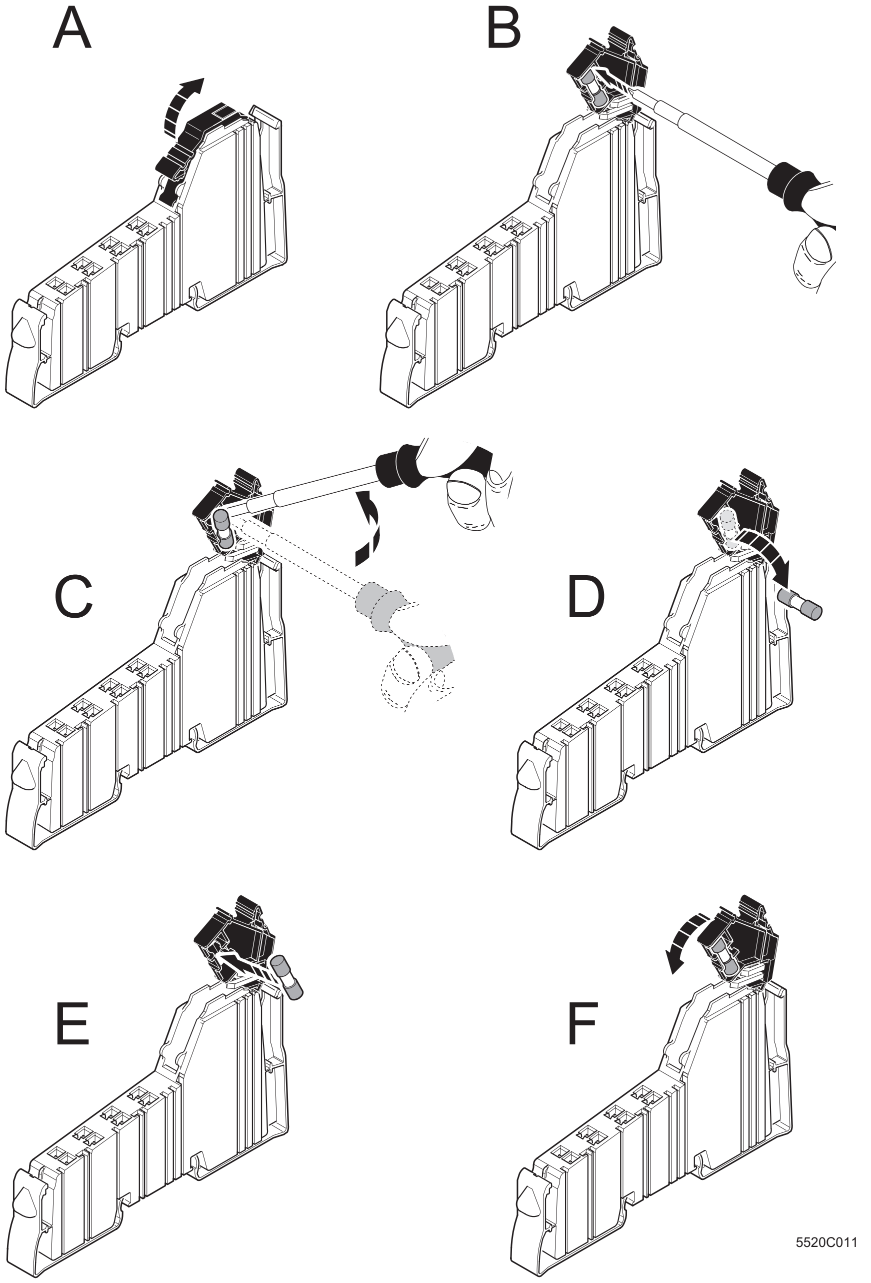


Figure 1-31 Replacing a fuse

1.15 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 mm^2 (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 underside 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).

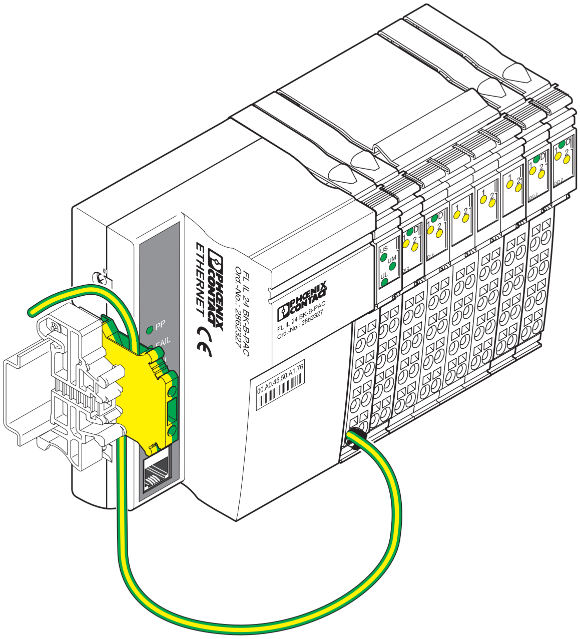


Figure 1-32 Additional grounding of the FL IL 24 BK-B-PAC

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.

1.15.1 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 connector and module.
- Do not damage or squeeze wires. Do not strip off the wires too far.
- Make a clean wire connection.

1.15.2 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 1.16.2, "Connecting Shielded Cables Using the Shield Connector".

Analog input and output modules require different shielding connections. The cable lengths must also be considered.

Table 1-19 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, grounding is connected to FE via an RC element.	< 10 m (32.81 ft.)	–
		> 10 m (98.43 ft.)	Connect the sensor directly to PE
Analog output module IB IL AO ...	Via shield connection clamp directly onto FE	< 10 m (32.81 ft.)	–
		> 10 m (32.81 ft.)	Isolate the actuator with an RC element and connect it to PE

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

- Connect the shielding to the shield connector (see Section 1.16.2, "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.

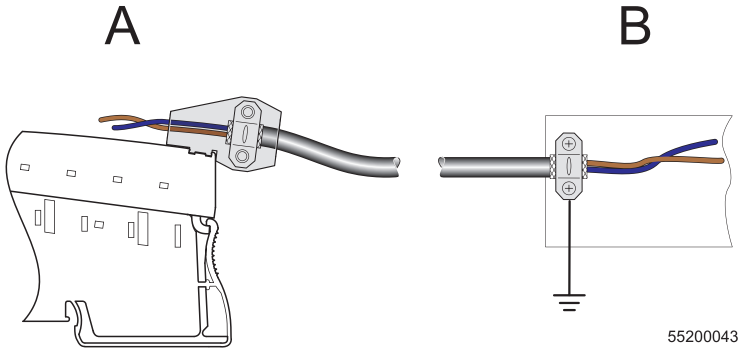


Figure 1-33 Connection of analog sensors, signal cables > 10 m (32.81 ft.)

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 as described above to the shield connector.

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.

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

- Connect the shielding via the shield connector (see Section 1.16.2, "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 with the 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.

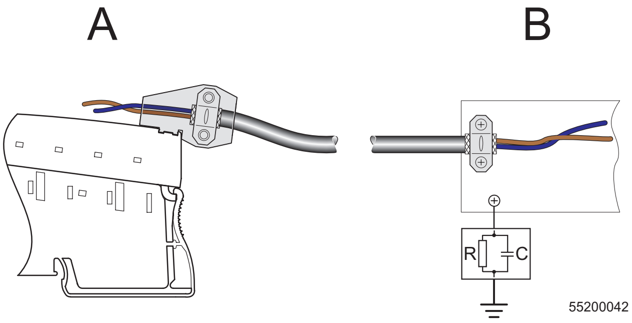


Figure 1-34 Connection of actuators, signal cables > 10 m (32.808 ft.)

A Module side

B Actuator side

1.16 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 mm^2 through 1.5 mm^2 (AWG 24 - 16) can be connected.

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

1.16.1 Connecting Unshielded Cables

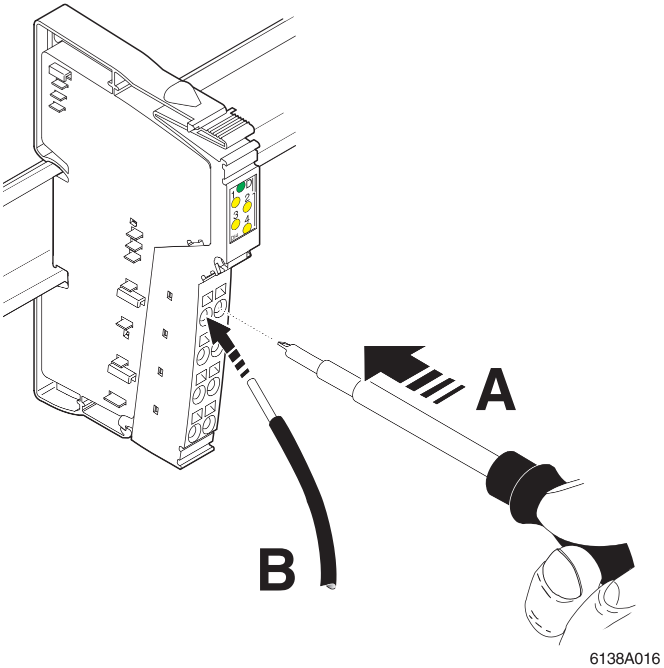


Figure 1-35 Connecting unshielded cables

Wire the connectors as required for your application.

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

When wiring, proceed as follows:

- Strip 8 mm (0.31 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, A), so that you can plug the wire into the spring opening. Phoenix Contact recommends using a SFZ 1 -0 x 3,5 screwdriver (Order No. 12 04 51 7; see "CLIPLINE" catalog from Phoenix Contact).
- Insert the wire (Figure 1-35, B). Remove the screwdriver from the opening. This clamps the wire.

After installation, the wires and the terminal points should be labeled.

1.16.2 Connecting Shielded Cables Using the Shield Connector

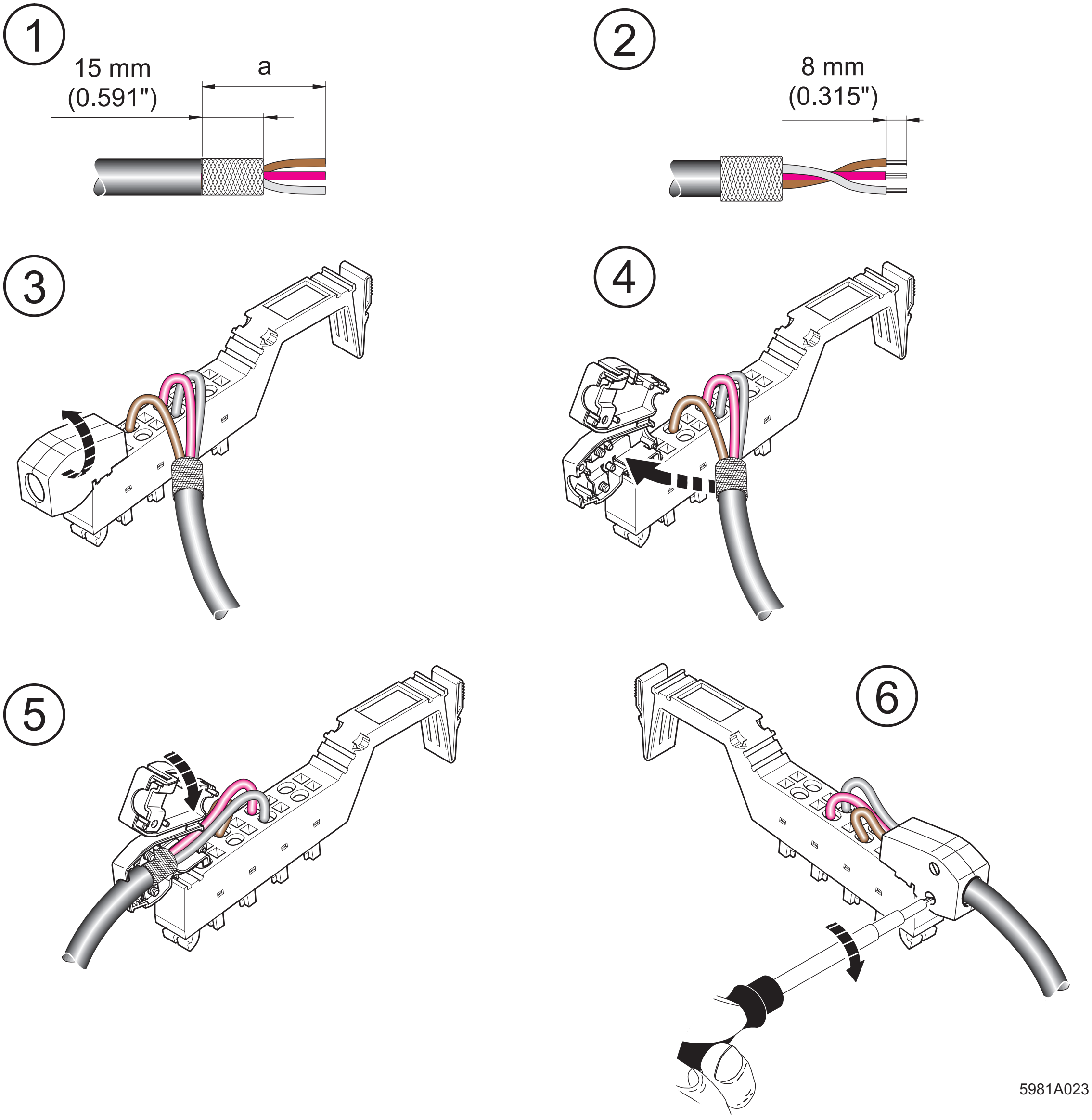


Figure 1-36 Connecting the shield to the shield connector

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 (0.59 in.). (1)
- Fold the braided shield back over the outer sheath. (2)
- Remove the protective foil.
- Strip 8 mm (0.32 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 for the appropriate connection (Figure 1-35 on page 1-61, 1), so that you can plug the wire into the spring opening.
   recommends using a SFZ 1 -0 x 3,5 screwdriver
  (Order No. 12 04 51 7; see "CLIPLINE" catalog from Phoenix Contact).
- Insert the wire (Figure 1-35 on page 1-61, detail 2). Remove the screwdriver from the opening. This clamps the wire.

For connector assignment, please consult 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 for the shield connector using a screwdriver. (6)

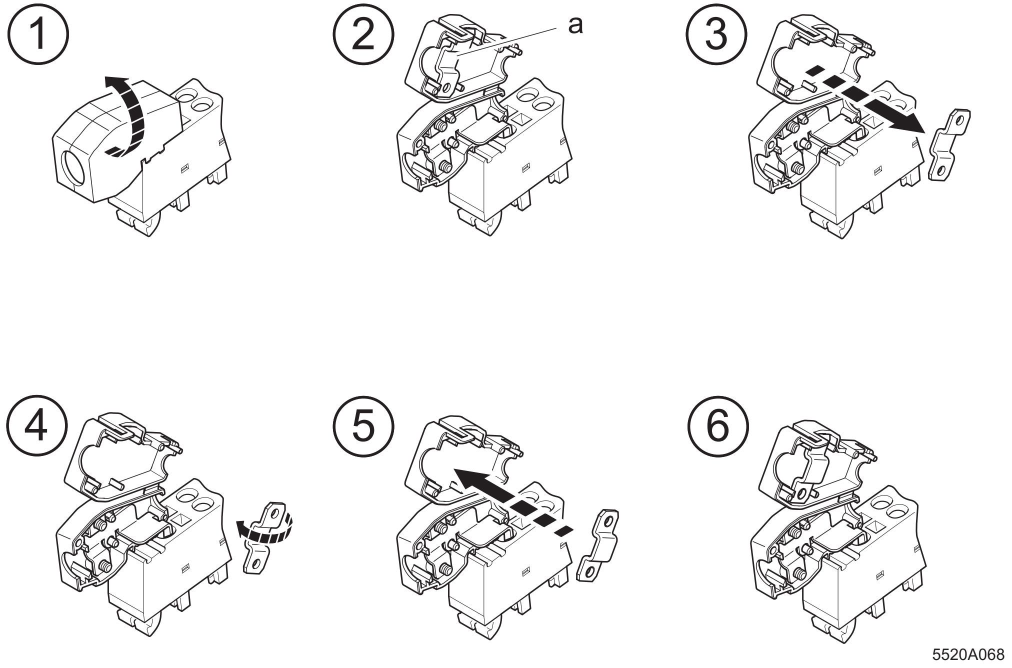


Figure 1-37 Shield connection clamp alignment

Shield connection clamp

The shield connection clamp (a in Figure 1-37, 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, 2). For thinner cables, the dip in the clamp is turned towards the cable (Figure 1-37, 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.

1.17 Connecting the Power 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 (Section 1.16.1).

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.

1.17.1 Power Terminal Supply

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

Voltage ranges

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

⚠

Use appropriate power terminals for different voltage ranges

To utilize different voltage ranges 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.

1.17.2 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.

1.17.3 Requirements Regarding the Voltage Supplies

⚠ **Use power supply units with safe isolation**

Only use power supplies 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.

1.18 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 1.16.1 on page 1-61 and the shielded cable as described in Section 1.16.2 on page 1-63.

1.18.1 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 shielding (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.

1.18.2 Examples of Connections 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 change accordingly. A connection example is given in each module-specific data sheet.

Table 1-20 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 (⊥)	–	X	X
Ground/FE shielding	FE (⏚)	–	–	X

X Used

– Not used

Table 1-21 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 (⊥)	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

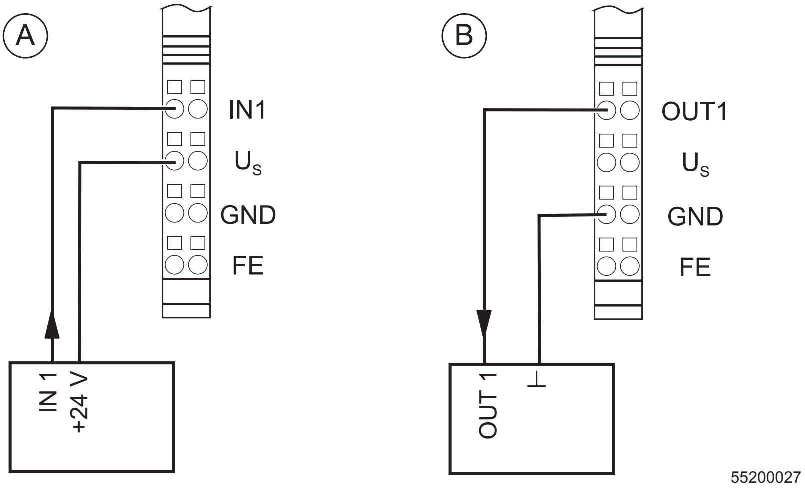


Figure 1-38 2-wire termination for digital devices

Sensor

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

Actuator

Figure 1-38, detail B, shows the connection of an actuator. The actuator power 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 (refer to the module-specific data sheet).

3-wire technology

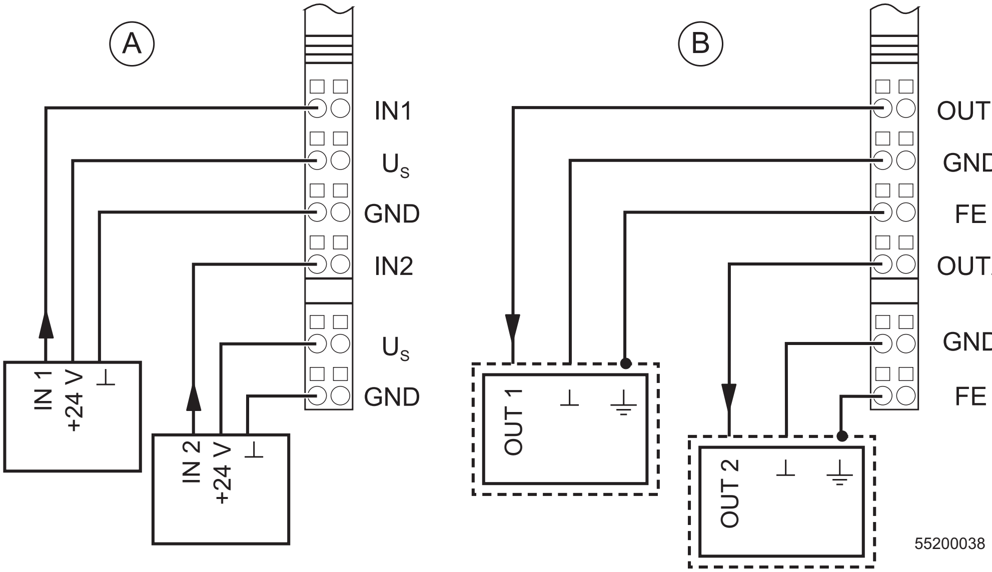


Figure 1-39 3-wire termination for digital devices

Sensor

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

Actuator

Figure 1-39, 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 (refer to the module-specific data sheet).

4-wire technology

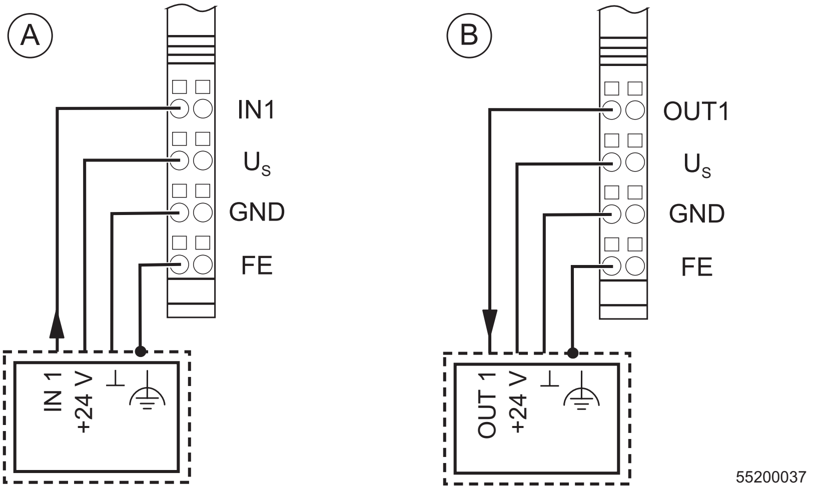


Figure 1-40 4-wire termination for digital devices

Sensor

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

Actuator

Figure 1-40, 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).

Section 2

This section informs you about

- the startup
- the IP paramter assignment
- the management information base (MIB)

Startup/Operation 2-3

2.1 Firmware Startup 2-3

2.1.1 Sending BootP Requests 2-3

2.2 Assigning an IP Address Using the Factory Manager 2-3

2.2.1 BootP 2-4

2.3 Manual Addition of Devices Using The Factory Manager 2-4

2.4 Selecting IP Addresses 2-5

2.4.1 Possible Address Combinations 2-6

2.4.2 Subnet Masks 2-7

2.4.3 Structure of the Subnet Mask 2-8

2.5 Web-Based Management 2-9

2.5.1 Calling Web-Based Management (WBM) 2-9

2.5.2 Structure of the Web Pages 2-10

2.5.3 Layout of the Web Pages 2-11

2.5.4 Password Protection 2-11

2.5.5 Process Data Access via XML 2-12

2.6 Factory Line I/O Configurator 2-17

2.6.1 Factory Line I/O Browser 2-17

2.6.2 OPC Configurator 2-18

2 Startup/Operation

2.1 Firmware Startup

After you power supplied your device, the firmware is started.

2.1.1 Sending BootP Requests

Initial Startup:

During initial startup, the device sends a BootP request 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.

Further Startups:

If the device already has valid configuration data, it only sends three more BootP requests on 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.

2.2 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.

2.2.1 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 of the device.
- Enter the relevant data in the Add New Ethernet Device dialog box (see Section 2.3).
- Save the configuration settings and restart the device (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 (power up). The device now sends another BootP request and receives the specified IP parameters from the BootP server.

2.3 Manual Addition of Devices Using The Factory Manager

- Click on the "Add device" command or use the key combination CTRL+A.
- 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 (power up).

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

2.4 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 are available, both for the network address (network ID) and the user address (host ID).

	Network ID	**Host ID**
Class A	7 bit	24 bit
Class B	14 bit	16 bit
Class C	21 bit	8 bit
Class D	28-bit multicast identifier	
Class E	27 bit (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 a 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.

2.4.1 Possible Address Combinations

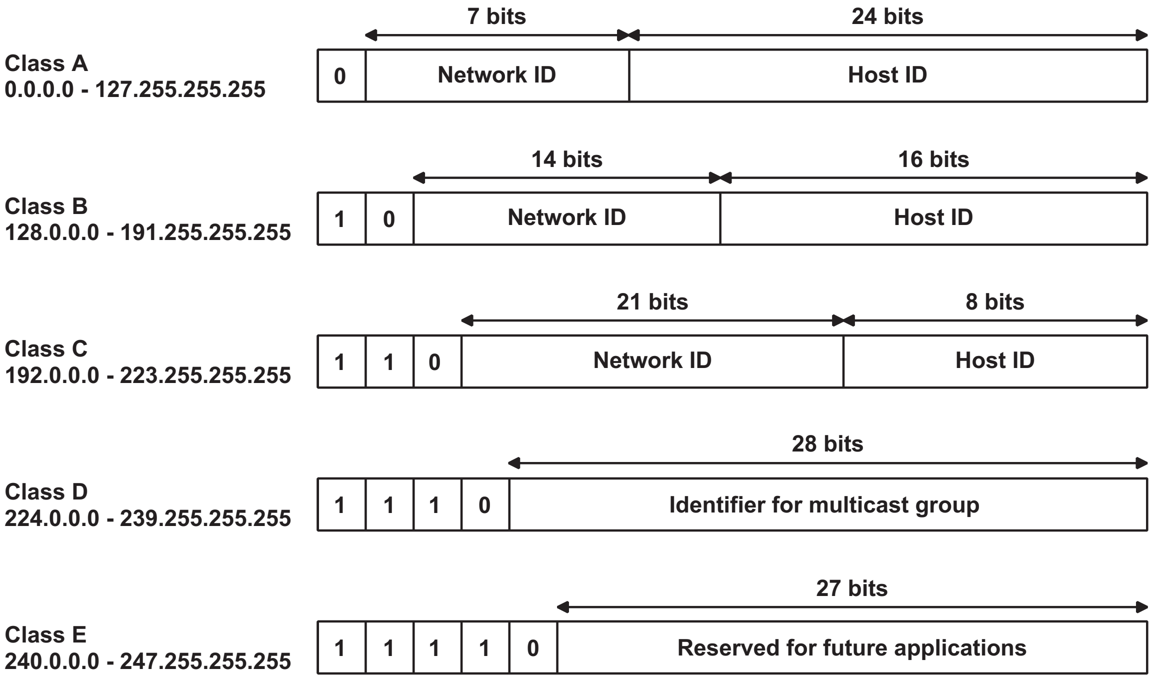


Figure 2-1 Structure of IP addresses

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 they should be tested 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.

2.4.2 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 PC, all the other devices display this extended IP address as a standard IP address.

2.4.3 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".

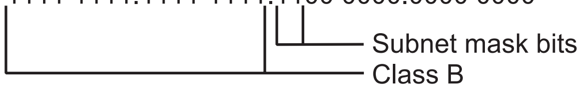
The remaining bits (three bytes) then contain the address of the subnetwork and the PC. The extended IP address is created when the bits of the IP address and the bits of the subnet mask are ANDed. Because the subnetwork is only recognized by local devices, the corresponding IP address appears as a "normal"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

Application

If the ANDing of the address bits gives the local network address and the local subnetwork address, the device is located in the local network. If the 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 with device 2 using the above subnet mask. Device 2 has IP address 59.EA.55.32.

IP address display 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.

2.5 Web-Based Management

The FL IL 24 BK-B-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).

2.5.1 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 terminal homepage is accessed by entering the URL "http://*ip-address*".

Example: http://192.168.2.81

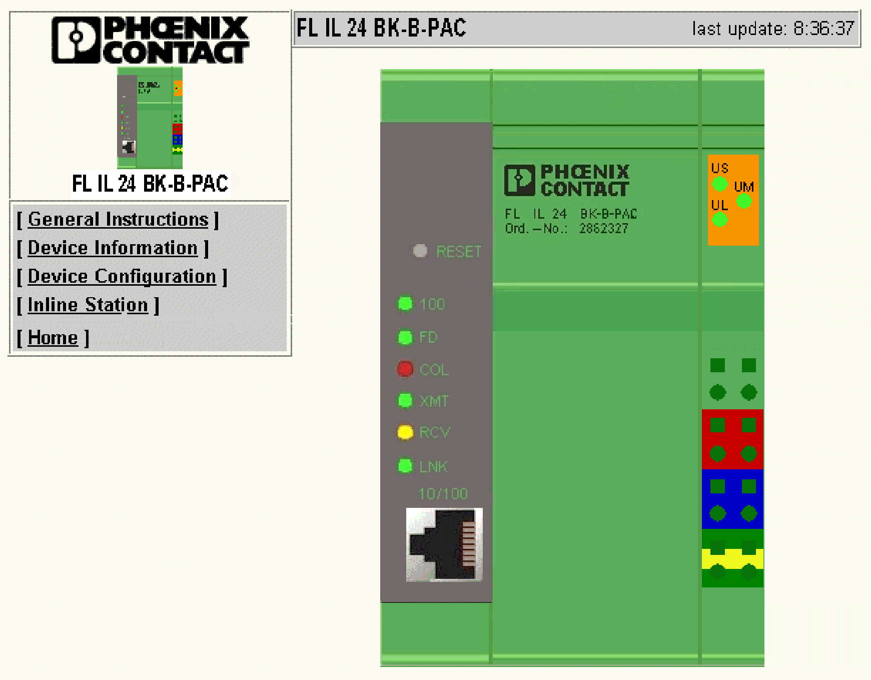
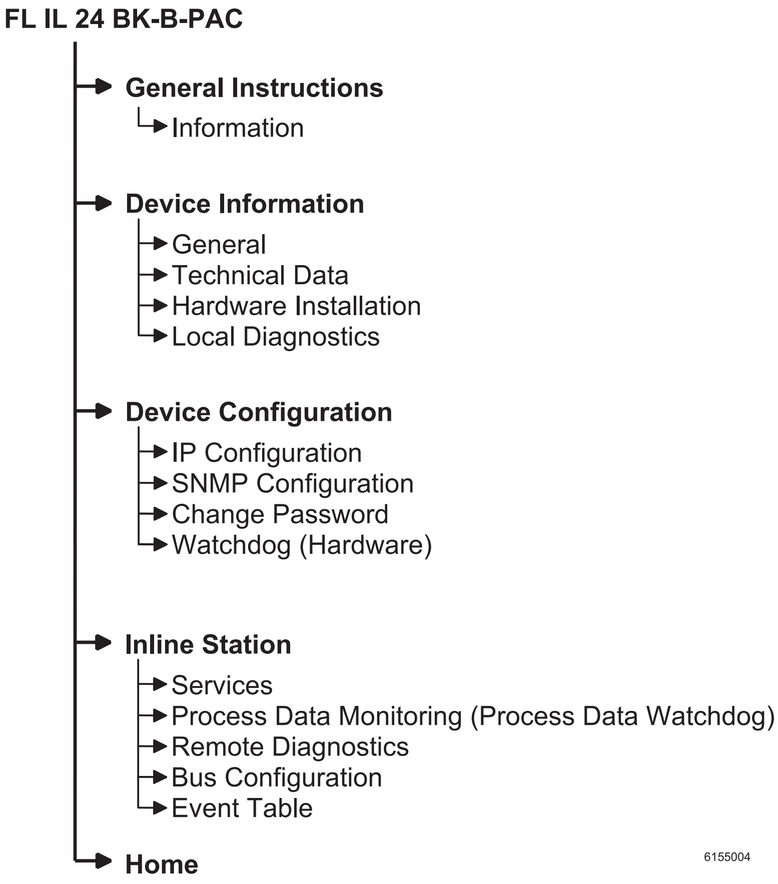


Figure 2-2 WBM homepage

2.5.2 Structure of the Web Pages

The Ethernet bus terminal pages are divided into two, 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 terminal can be found in the following menus.

2.5.3 Layout of the Web Pages



2.5.4 Password Protection

The bus terminal 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 terminal 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 contact the telephone number indicated on the last page.

2.5.5 Process Data Access via XML

The integrated web server of the FL IL 24 BK-B-PAC offers the possibility to access the process data of the connected Inline terminals via a website in XML format.

You can access the websites via a standard web browser. For calling the XML pages with the process data in the address line of the browser, enter the address in the following format: „http:// <IP-Adresse>/ processdata.xml“.

2.5.5.1 XML File Structure

The XML file contains different data areas:

IL_STATION Frames 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). Belonging to this data area: TERMINAL_TYPE, the module name NAME, the IP address IP_ADDRESS, the number of connected terminals MODULE_NUMBER and the INTERBUS diagnostic-register DIAGNOSTIC_STATUS_REGISTER and the INTERBUS status register DIAGNOSTIC_PARAMETER_REGISTER.

TERMINAL_TYPE This area contains the module, it is always FL IL 24 BK-B-PAC.

NAME Contains the user-specific station names. The station name can be modified via WBM.

IP_ADDRESS Contains the IP address of the station.

MODULE_NUMBER Contains the number of connected Inline terminals. In the case of a bus error, the number of the last known operable configuration is indicated.

DIAGNOSTIC_STATUS _REGISTER Contains the INTERBUS status, represented via all bits of the diagnostic status register. A detailed description can be found in the diagnostic parameter register. Whenever an error bit was set, the diagnostic parameter register was rewritten.

IL_BUS Frame for the connected Inline terminals.

IL_MODULE Frame for the data of an individual Inline terminal. The terminals are numbered from one up to 63.

MODULE_TYPE Contains the terminal type. Possible types are DI, DO, DIO, AI, AO, and AIO.

PD_CHANNELS Number of process data channels in an Inline terminal. For digital terminals the number of channels is equal to the number of supported bits. For other modules, the number of process data words is indicated. Example: An AO 2 has two process data channels and a DO 8 has 8 bits and 8 process data channels.

PD_WORDS Number of process data words in an Inline terminal. Observe that analog terminals always have the same number of output and input words. An AO 2 also has two input channels and an AI 2 also has tow output channels.

PD_IN This area is used by all terminals that occupy 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 and 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 and the 13th and the 14th input is 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 input word returns the value 12288 (2^{12} + 2^{13}).

d) Inline terminal with two analog inputs, only the first channel being active (14970).

```
<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 with the use of "PD_IN“.

2.5.5.2 Validity of Documentation

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

2.5.5.3 Error in an Inline Station

If the FL IL 24 BK-B-PAC does not configure the connected Inline terminals correctly, the process data are listed in the XML file as follows.

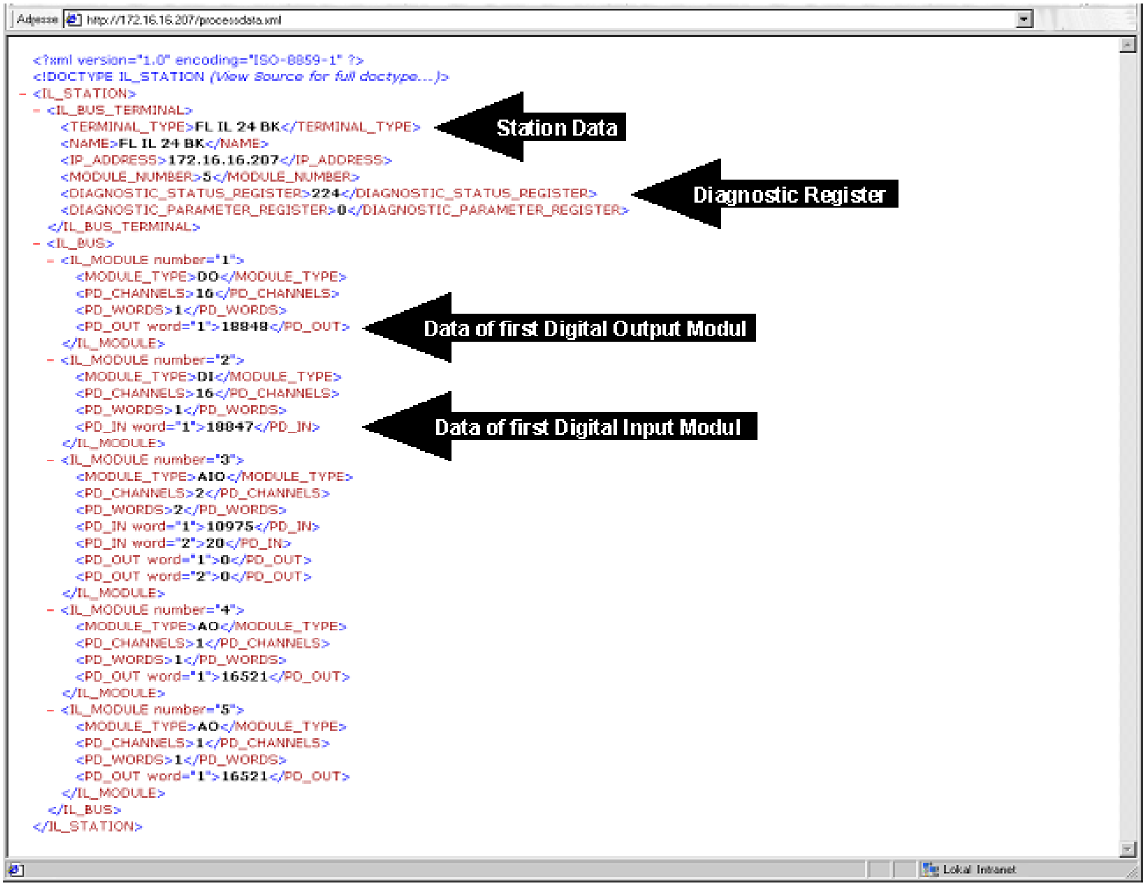
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE IL_STATION SYSTEM "processdata.dtd">
<IL_STATION>
        <IL_BUS_TERMINAL>
            <TERMINAL_TYPE>FL IL 24 BK-B-PAC</TERMINAL_TYPE>
            <NAME>FL IL 24 BK-B-PAC</NAME>
            <IP_ADDRESS>172.16.27.37</IP_ADDRESS>
            <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 and the diagnostic parameter register indicate the error cause. The number of connected terminals is "Zero", thus the area "IL_BUS" is empty.

In the event of a bus error, the process data are invalid because only internal values are indicated, however, not the values on the INTERBUS. The status is indicated in the diagnostic register.

In order to make sure that only valid data are displayed, the diagnostic register also must be requested. The same is valid in the event of a faulty configuration. In this case, the INTERBUS does not run and only internal values can be read in the XML file.

If an I/O error occurs, all data are valid, except for the data of the faulty terminal.



Adresse
http://172.16.16.207/processdata.xml
Station Data
Diagnostic Register
Data of first Digital Output Modul
Data of first Digital Input Modul
Lokal Intranet

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <!DOCTYPE IL_STATION (View Source for full doctype...)>
- <IL_STATION>
  - <IL_BUS_TERMINAL>
      <TERMINAL_TYPE>FL IL 24 BK</TERMINAL_TYPE>
      <NAME>FL IL 24 BK</NAME>
      <IP_ADDRESS>172.16.16.207</IP_ADDRESS>
      <MODULE_NUMBER>5</MODULE_NUMBER>
      <DIAGNOSTIC_STATUS_REGISTER>224</DIAGNOSTIC_STATUS_REGISTER>
      <DIAGNOSTIC_PARAMETER_REGISTER>0</DIAGNOSTIC_PARAMETER_REGISTER>
    </IL_BUS_TERMINAL>
  - <IL_BUS>
    - <IL_MODULE number="1">
        <MODULE_TYPE>DO</MODULE_TYPE>
        <PD_CHANNELS>16</PD_CHANNELS>
        <PD_WORDS>1</PD_WORDS>
        <PD_OUT word="1">18848</PD_OUT>
      </IL_MODULE>
    - <IL_MODULE number="2">
        <MODULE_TYPE>DI</MODULE_TYPE>
        <PD_CHANNELS>16</PD_CHANNELS>
        <PD_WORDS>1</PD_WORDS>
        <PD_IN word="1">10047</PD_IN>
      </IL_MODULE>
    - <IL_MODULE number="3">
        <MODULE_TYPE>AIO</MODULE_TYPE>
        <PD_CHANNELS>2</PD_CHANNELS>
        <PD_WORDS>2</PD_WORDS>
        <PD_IN word="1">10975</PD_IN>
        <PD_IN word="2">20</PD_IN>
        <PD_OUT word="1">0</PD_OUT>
        <PD_OUT word="2">0</PD_OUT>
      </IL_MODULE>
    - <IL_MODULE number="4">
        <MODULE_TYPE>AO</MODULE_TYPE>
        <PD_CHANNELS>1</PD_CHANNELS>
        <PD_WORDS>1</PD_WORDS>
        <PD_OUT word="1">16521</PD_OUT>
      </IL_MODULE>
    - <IL_MODULE number="5">
        <MODULE_TYPE>AO</MODULE_TYPE>
        <PD_CHANNELS>1</PD_CHANNELS>
        <PD_WORDS>1</PD_WORDS>
        <PD_OUT word="1">16521</PD_OUT>
      </IL_MODULE>
    </IL_BUS>
  </IL_STATION>
```

Figure 2-3 Screen for XML data

2.6 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 terminals. In particular, process data exchange is supported via **OPC** in an easy-to-use manner.

You will 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.

2.6.1 Factory Line I/O Browser

The bus structure is created using the I/O Browser. Out of all supported modules, select those that you want to use in your station (offline configuration) or that you are using at the moment (online configuration). With regard to the online configuration, you have the possibility to read in an existing bus structure and to test it.

Configuration

During system planning the I/O configurator offers an integrated online product catalog to help ensure optimal 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 with the help of the integrated INTERBUS technology.

Inline station structure with I/O configurator

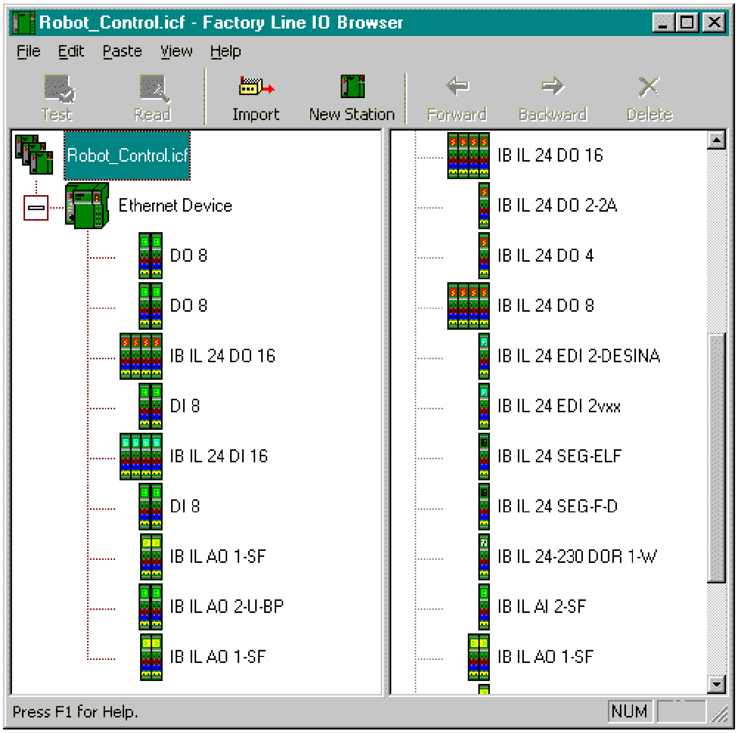


Figure 2-4 I/O browser screen

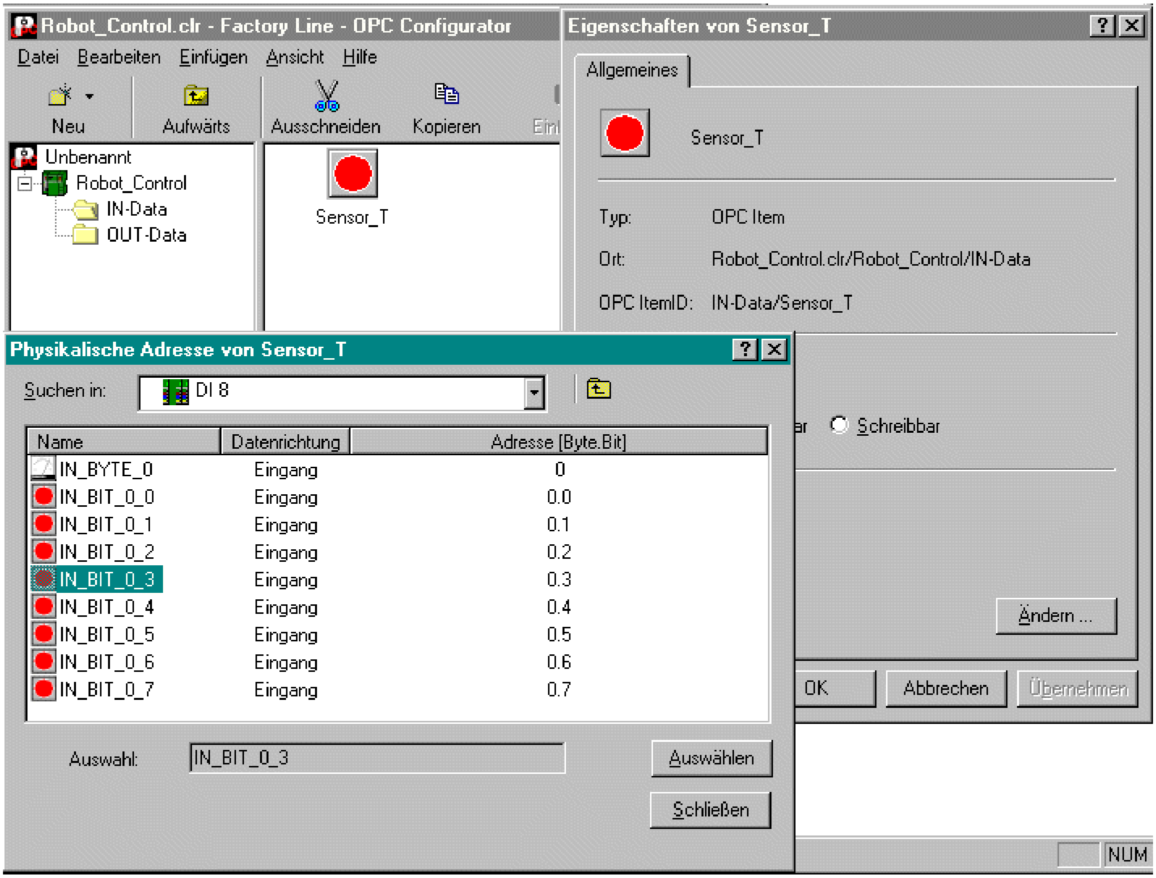
2.6.2 OPC Configurator

OPC Data Exchange

Process data exchange via OPC is supported in an very easy-to-use manner. Use the OPC Configurator to assign OPC items to the Inline station structure for the respective terminal points. With 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 terminal type. 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.


Robot_Control.clr - Factory Line - OPC Configurator
Datei Bearbeiten Einfügen Ansicht Hilfe
Neu
Aufwärts
Ausschneiden
Kopieren
Ein
Unbenannt
Robot_Control
IN-Data
OUT-Data
Sensor_T
Eigenschaften von Sensor_T
Allgemeines
Ändern ...
OK
Abbrechen
Übernehmen
Physikalische Adresse von Sensor_T
Auswählen
Schließen
NUM

Sensor_T

Typ: OPC Item

Ort: Robot_Control.clr/Robot_Control/IN-Data

OPC ItemID: IN-Data/Sensor_T

ar ○ Schreibbar

Suchen in: DI 8

Name	Datenrichtung	Adresse [Byte.Bit]
IN_BYTE_0	Eingang	0
IN_BIT_0_0	Eingang	0.0
IN_BIT_0_1	Eingang	0.1
IN_BIT_0_2	Eingang	0.2
IN_BIT_0_3	Eingang	0.3
IN_BIT_0_4	Eingang	0.4
IN_BIT_0_5	Eingang	0.5
IN_BIT_0_6	Eingang	0.6
IN_BIT_0_7	Eingang	0.7

Auswahl: IN_BIT_0_3

Figure 2-5 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 terminal 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.

Section 3

This section informs you about

- the driver software
- the example program

Driver Software 3-3

3.1 Documentation 3-3

3.1.1 Hardware and Software User Manual 3-3

3.2 The Software Structure 3-3

3.2.1 Ethernet / Inline Bus Terminal 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 3-7

3.4.1 Position of the Process Data (Example) 3-8

3.5 Startup Behavior of the Bus Terminal 3-9

3.5.1 Plug & Play Mode 3-9

3.5.2 Expert Mode...3-10

3.5.3 Possible Combination of Modes 3-10

3.5.4 Startup Diagrams of the Bus Coupler 3-11

3.5.5 Changing and Starting a Configuration in P&P Mode..3-13

3.6 Changing a Reference Configuration Using the Software 3-14

3.6.1 Effects of Expert Mode..3-14

3.6.2 Changing a Reference Configuration...........................3-14

3.7 Description of the Device Driver Interface (DDI) 3-16

3.7.1 Introduction 3-16

3.7.2 Overview 3-17

3.7.3 Working Method of the Device Driver Interface 3-17

3.7.4 Description of the Functions of the Device Driver Interface 3-20

3.8 Monitoring Function 3-37

3.8.1 Process Data Monitoring / Process Data Watchdog 3-39

3.8.2 Connection Monitoring (Host Checking) 3-41

3.8.3 Data Interface (DTI) Monitoring 3-44

3.8.4 I/O Fault Response Mode 3-48

3.8.5 Handling the NetFail Signal / Testing With ETH_SetNetFail 3-50

3.9 IN Process Data Monitoring 3-58

3.10 Notification Mode 3-62

3.11 Programming Support Macros 3-65

3.11.1 Introduction 3-65

3.12 Description of the Macros 3-67

3.12.1 Macros for Converting the Data Block of a Command. 3-68

3.12.2 Macros for Converting the Data Block of a Message... 3-71

3.12.3 Macros for Converting Input Data 3-74

3.12.4 Macros for Converting Output Data 3-75

3.13 Diagnostic Options of the Driver Software 3-78

3.13.1 Introduction 3-78

3.14 Positive Messages 3-79

3.15 Error Messages 3-80

3.15.1 General Error Messages 3-80

3.15.2 Error Messages When Opening a Data Channel........ 3-82

3.15.3 Error Messages When Transmitting Messages/Commands 3-83

3.15.4 Error Messages When Transmitting Process Data...... 3-85

3.16 Example Program 3-88

3.16.1 Demo Structure Startup 3-88

3.16.2 Example Program Source Code 3-90

3 Driver Software

3.1 Documentation

3.1.1 Hardware and Software User Manual

This *Hardware and Software User Manual for FL IL 24 BK-B-PAC* (Order No. 26 98 65 6) describes the hardware and software functions in association with an Ethernet network and the functions of the Device Driver Interface (DDI) software.

All figures, tables, and abbreviations are listed in the appendices. The index in the appendix makes it easier to search for specific key terms and descriptions.

3.2 The Software Structure

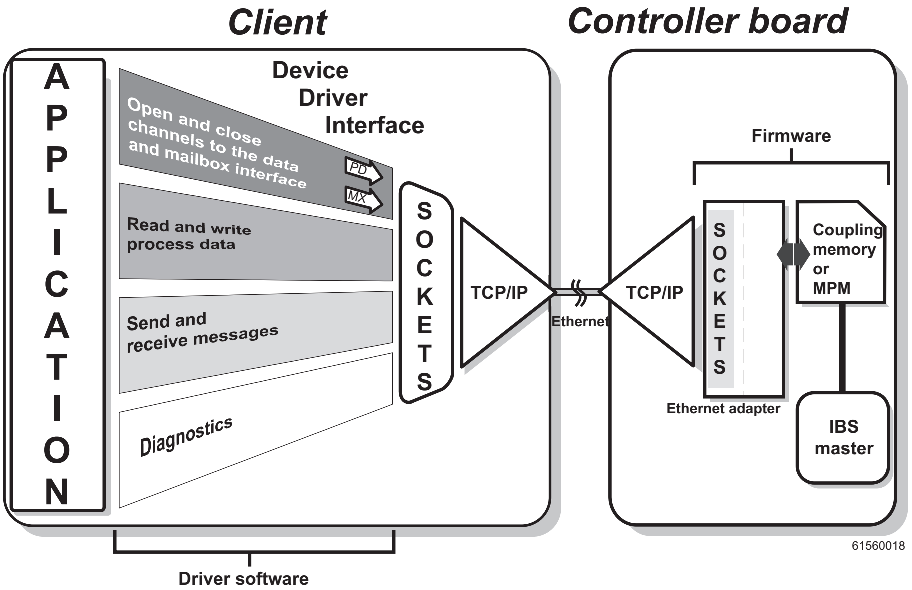


Figure 3-1 Software structure

3.2.1 Ethernet / Inline Bus Terminal Firmware

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

The bus terminal provides a basic interface for using services via the Ethernet network. The software primarily encodes and decodes the data telegrams for addressing the bus terminal services. The firmware also ensures the network-specific addressing of the bus terminal 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 the *IBS ETH DDI SWD E* (Order No. 27 51 13 7).

The driver software can be divided into three groups. The Device Driver Interface functions form the first group, which controls the bus terminal via the Ethernet network. Using these functions, firmware services can be called and started, and results can be requested on the bus terminal. The second group contains functions for monitoring the bus terminal 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.

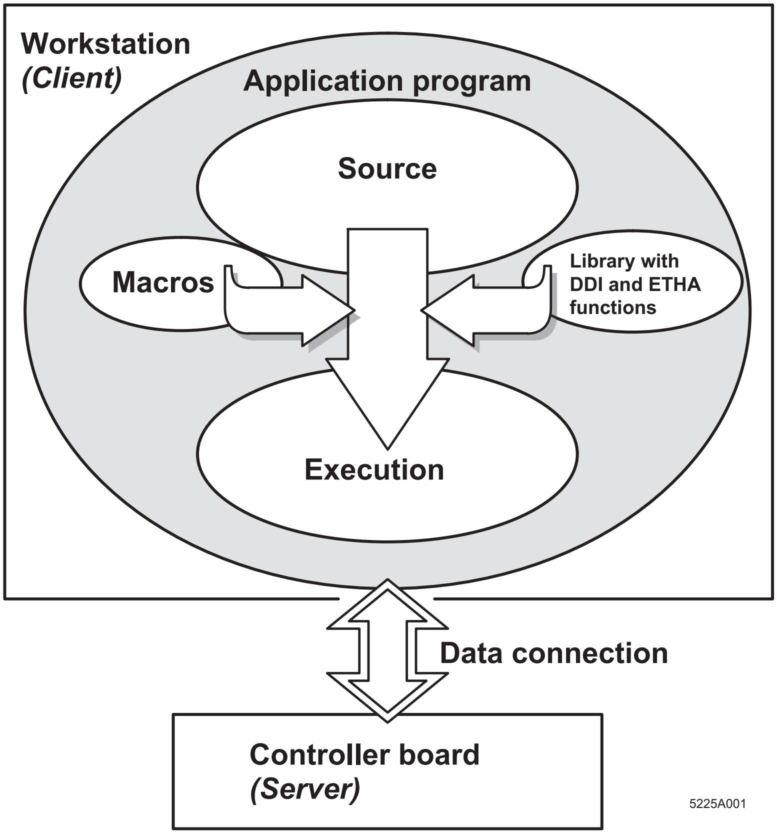


Figure 3-2 Using the driver software in the application program

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 http://www.phoenixcontact.com.

Training Courses

Our bus terminal 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:

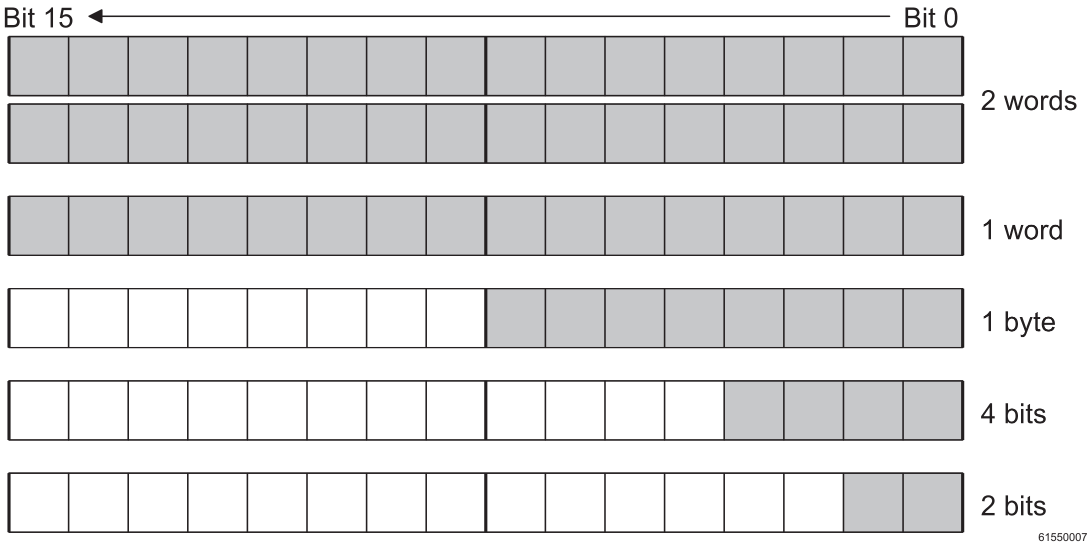


Figure 3-3 Position of the user data for individual devices in the word array

To achieve cycle consistency between input/output data and the station bus cycle, the bus terminal 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.

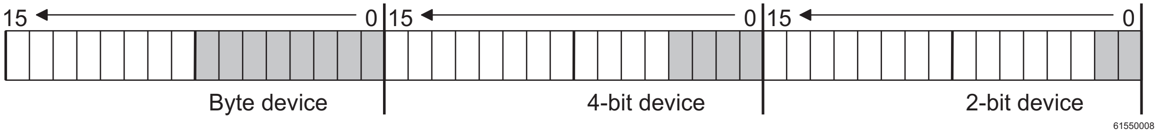


Figure 3-4 Position of the user data for several devices in the word array

3.4.1 Position of the Process Data (Example)

The physical assignment of the devices to the bus terminal determines the order of the process data in the memory. The following diagram illustrates an example bus configuration and the position of the relevant process data.

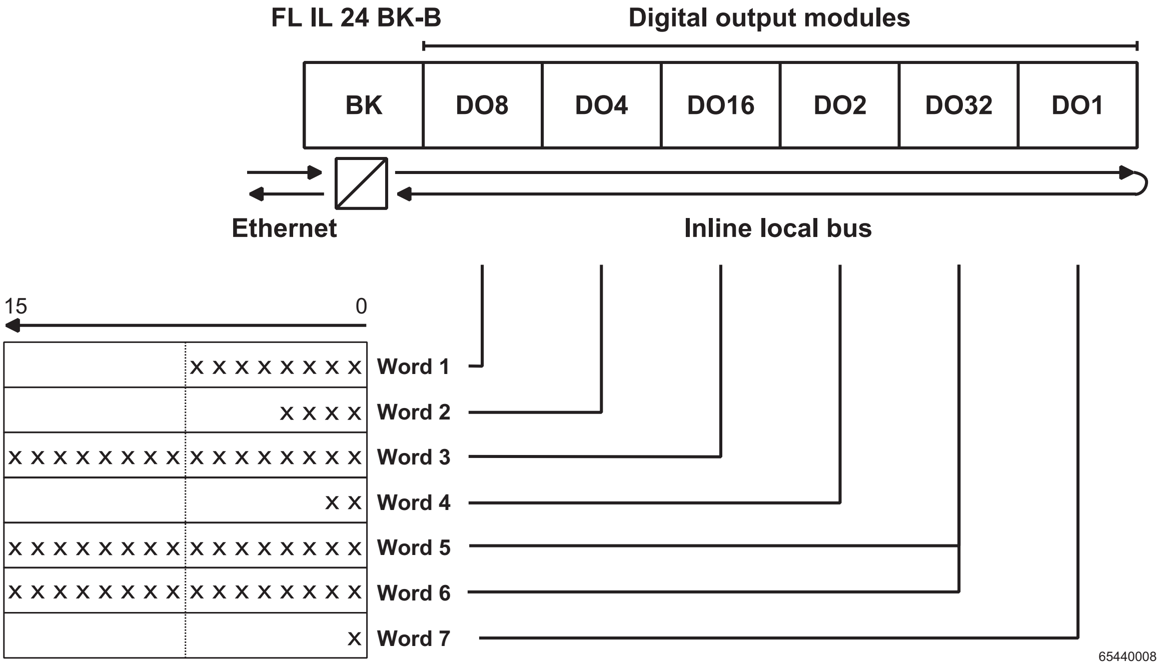


Figure 3-5 Position of the process data according to the physical bus configuration

3.5 Startup Behavior of the Bus Terminal

The startup behavior of the bus coupler is specified via two system parameters, the Plug & Play mode and the expert mode. In the delivery state, the P&P mode is activated and the export mode is deactivated.

3.5.1 Plug & Play Mode

Please observe that the following description is valid for the deactivated expert mode. Possible combinations of both modes and their behavior are described in on page 3-10.

P&P mode activated

The FL IL 24 BK-B-PAC supports the socalled Plug & Play mode (P&P). This mode enables Inline modules connected in the field to be started up using the FL IL 24 BK-B-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 detected and their function is checked. If this physical configuration is ready for operation it is stored retentively as reference configuration on the bus coupler.
The P&P mode needs to be deactivated again so that the reference configuration will not be overwritten once again during the next startup of the bus coupler. At the same time, the deactivation of the P&P mode also is the acknowledgment for the reference configuration and the release of the process data exchange.

P&P mode deactivated

When the P&P mode is deactivated, the reference configuration is compared to the physical configuration. If these two configurations are identical, the bus coupler can be set to the "RUN" state.

If the reference configuration and the physical configuration are not identical, the FAIL LED flashes and the process data exchange is not possible for safety reasons.

In order to operate the bus you have the following two options:

1. Restore the original configuration so that the reference configuration and the physical configuration are identical once more or
2. activate the P&P mode so that the current physical configuration can be accepted as reference configuration.

3.5.2 Expert Mode

☞ Please observe that the following description is valid for the deactivated P&P mode. Possible combinations of both modes and their behavior are described in on page 3-10.

Expert mode deactivated

If the expert mode is deactivated (default upon delivery) so an error-free configuration is automatically set to the "RUN" state. If the configuration has a technical defect or if it is not identical with the reference configuration, the FAIL LED flashes and the process data exchange is not possible.

Expert mode activated

If the expert mode is activated, the faulty configuration is set to the "READY" state but not automatically to the "RUN" state. The user must set the station to the "RUN" state using the appropriate firmware commands such as ACTIVATE_CONFIGURATION, 0x0711 or START_DATA_TRANSFER, 0x0701.

3.5.3 Possible Combination of Modes

Table 3-1 Possible combination of modes and their effects

P&P Mode	Expert Mode	Description/Effect	Diagram
Deactive	Deactive	Standard situation - The station sets valid configurations to the "RUN" state since the current configuration is identical with the memory. Process data exchange is possible.	Figure 3-6 on page 3-11
Deactive	Active	A valid configuration is set to the "READY" state. A process data exchage is only possible if the station has been set to the "RUN" state using the firmware command.	Figure 3-7 on page 3-11
Active	Deactive	The connected configuration is stored as reference configuration and the station is set to the "RUN" state. Process data exchange is not possible.	Figure 3-8 on page 3-12
Active	Active	A physical configuration is stored as reference configuration and is set to the "Ready" state. A process data exchage is only possible if the P&P mode is deactivated and the station has been set to the "RUN" state using firmware commands.	Figure 3-9 on page 3-12

3.5.4 Startup Diagrams of the Bus Coupler

"Standard" mode / P&P and expert mode deactivated

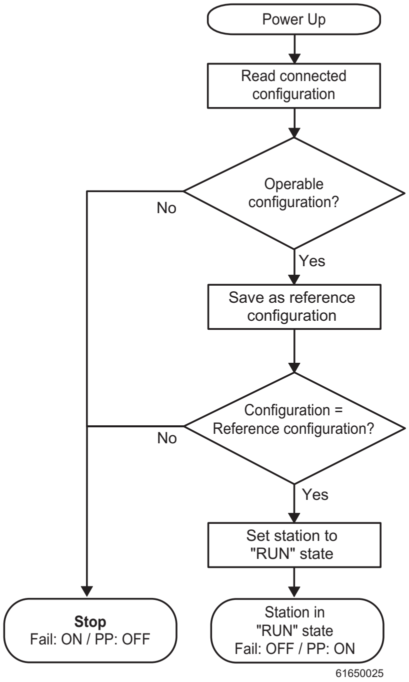

Power Up
Read connected configuration
Operable configuration?
No
Yes
Save as reference configuration
Configuration = Reference configuration?
No
Yes
Set station to "RUN" state
Stop
Fail: ON / PP: OFF
Station in "RUN" state
Fail: OFF / PP: ON
61650025

Figure 3-6 "Standard" mode / expert and P&P mode deactivatetd

P&P mode deactivated - expert mode activated

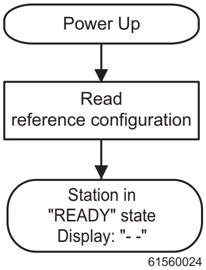


Figure 3-7 P&P mode deactivated - expert mode activated

P&P mode activated - expert mode activated

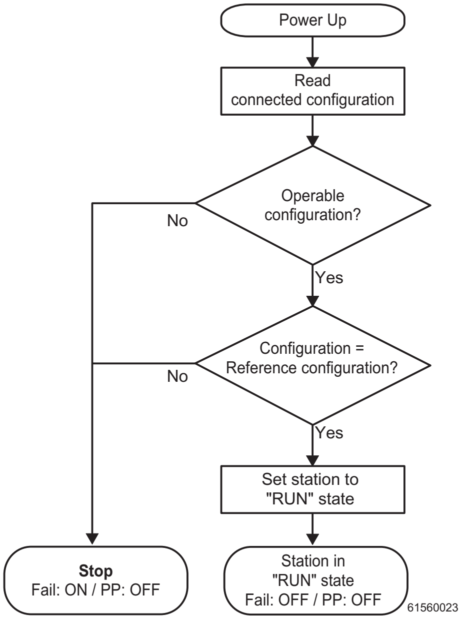


Figure 3-8 P&P mode activated - expert mode deactivated

P&P mode activated - expert mode deactivated

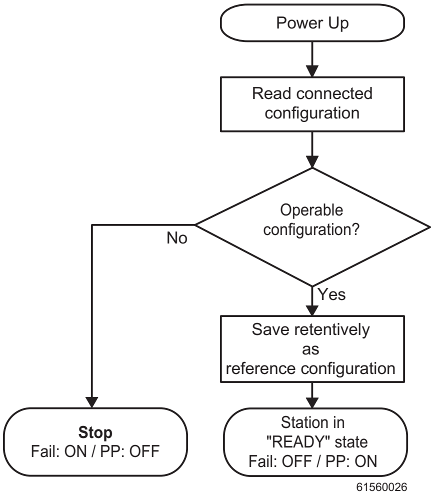


Figure 3-9 P&P mode activated - expert mode activated

3.5.5 Changing and Starting a Configuration in P&P Mode

Ensure that Plug & Play mode is activated and expert mode is deactivated.

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

- 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 up 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 active configuration is saved in the EEPROM as the reference configuration.
- All connected Inline devices are integrated in the active 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 either using the I/O Browser, or the "Set_Value" command via Ethernet. Once P&P mode has been switched off, the bus is only disconnected 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 terminal 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 activated, automatic startup of the connected local bus is prevented.

The user must manually place the bus in 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 terminal 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 in "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 it can be 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 via an error message, this means that no reference configuration is present.

System parameters for the "Set_Value" service. (750_{hex})

Variable ID	System parameters	Value/Comment
2216_{hex}	Up-to-date 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 modus deactivated (default) 1: Expert mode active
2277_{hex}	Fault response mode	1: Reset fault mode (default) 2: Hold Last State 0: Standard Fault Mode
2293_{hex}	Process data monitoring timeout	0: Process data watchdog deactivated 200 - 65000: Timeout value

☞ The P&P mode is only activated after the reboot.

3.7 Description of the Device Driver Interface (DDI)

3.7.1 Introduction

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

3.7.2 Overview

Table 3-2 Overview of the functions in the DDI

Functions	page
DDI_DevOpenNode	3-20
DDI_DevCloseNode	3-22
DDI_DTI_ReadData	3-24
DDI_DTI_WriteData	3-26
DDI_DTI_ReadWriteData	3-28
DDI_MXI_SndMessage	3-30
DDI_MXI_RcvMessage	3-32
GetIBSDiagnostic	3-34
ETH_SetHostChecking	3-40
ETH_ClearHostChecking	3-42
ETH_SetDTITimeoutCtrl	3-44
ETH_ClearDTITimeoutCtrl	3-45
ETH_SetNetFail	3-49
ETH_GetNetFailStatus	3-49
ETH_ClrNetFailStatus	3-52
DDI_SetMsgNotification	3-56
DDI_ClrMsgNotification	3-56
ETH_ActivatePDInMonitoring	3-57
ETH_DeactivatePDInMonitoring	3-60
ETH_SetNetFailMode	3-53
ETH_GetFailMode	3-55

3.7.3 Working Method of the Device Driver Interface

Remote procedure call

The entire Device Driver Interface (DDI) for the bus terminal 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 terminal 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 terminal) 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 terminal), 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.

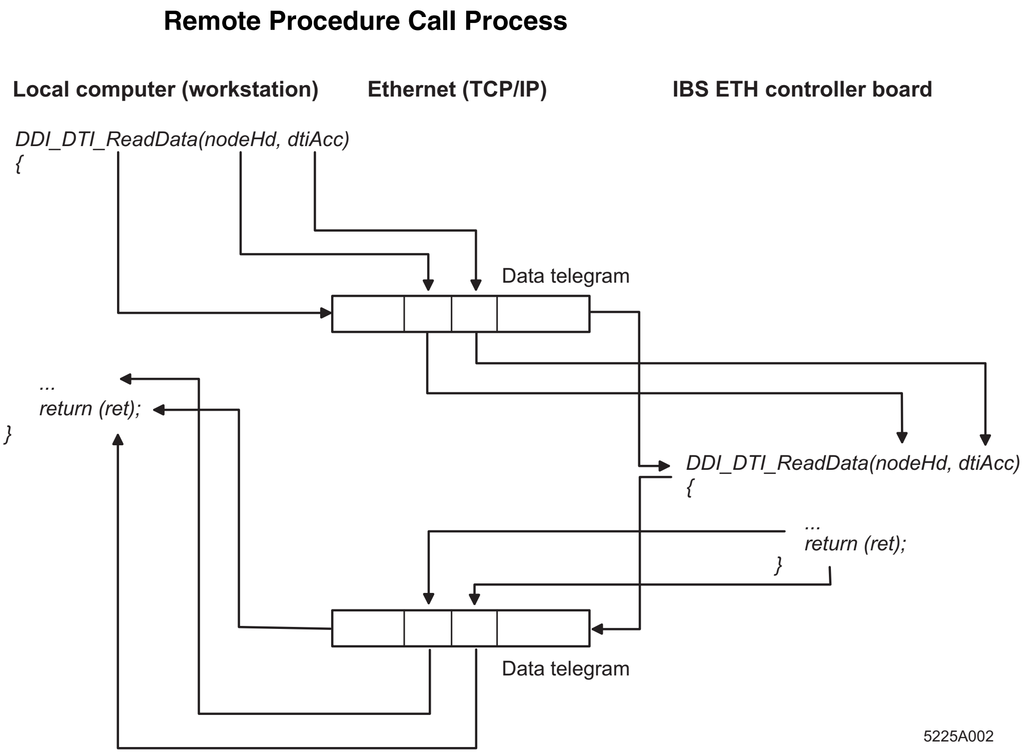

Remote Procedure Call Process
Local computer (workstation)
Ethernet (TCP/IP)
IBS ETH controller board
DDI_DTI_ReadData(nodeHd, dtiAcc)
{
Data telegram
...
return (ret);
}
DDI_DTI_ReadData(nodeHd, dtiAcc)
{
...
return (ret);
}
Data telegram
5225A002

Figure 3-10 Execution of a remote procedure call

3.7.4 Description of the 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 terminal 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 terminal.

☞ Another 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 terminal can be entered in "dotted notation": `192.168.5.76` or as server name: `etha2`, is of no importance. 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 terminal. The driver creates the entries for you. You will find the driver in the download area of www.phoenixcontact.com or on 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 terminal 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 accesses 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 terminal. This node handle has direct reference to an internal control structure, which contains all the corresponding data for addressing the relevant bus terminal.

The local node handle is used to obtain all the necessary parameters for addressing the bus terminal, such as the IP address, socket handle, node handle on the bus terminal, 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 8 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 *perm* 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 indicates 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);

Parameters:	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 terminal. 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 terminal 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 terminal 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 terminal.

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 terminal, 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 terminal.

> By default upon delivery, the watchdog is activated with 500 ms timeout. The first write process starts the process data watchdog; the next write process is expected during the next 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 terminal 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 terminal 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 terminal.

Syntax:

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

Parameters:

IBDDIHND nodeHd

Node handle (DTI) for the connection to which data is to be written. The node handle also determines the bus terminal, 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 of more than one byte.

Format of the T_DDI_DTI_ ACCESS ACCESS

```
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 starts the process data watchdog; the next write process is expected during the next 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 terminal 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 terminal.

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 terminal, 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 terminal. 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 terminal via the network without a plausibility check, which means that invalid parameters are first detected at the bus terminal and acknowledged with an error message. The *IBDDIHND nodeHd* parameter specifies the bus terminal 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 terminal.

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 terminal, 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 terminal. 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 terminals via the network without a plausibility check, which means that invalid parameters are first detected at the bus terminal and acknowledged with an error message.

The *nodeHd* parameter specifies the bus terminal 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 terminal.

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 terminal.

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 terminal, 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];
     unsignet 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 terminal from which the diagnostic bit register and diagnostic parameter register are to be read.

T_IBS_DIAG *infoPtrPointer 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)/* Depending on the operating
system */;
          ddiRet = GetIBSDiagnostic (ddiHnd, &infoPtr);
          stateAB = infoPtr.state;
          diagAB = infoPtr.diagPara;
     } while (...)
}
```

3.8 Monitoring Function

Monitoring functions with different features are available for monitoring the Ethernet communication or the connected devices:

- Process data watchdog (process data monitoring),
- Host checking and
- DTI monitoring.

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

Table 3-3 Monitoring functions

Monitoring Mechanism	Monitoring …			
	... the Client Application	... the Individual Channels	... the Ethernet Connection	... the 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 3-4 Available fault response modes

Fault Response Mode	Value	Function
Reset fault mode (default)	**1**	The digital 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.

Causes for Fault Response

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

Causes

The following may have been the cause:

```
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 */
```

Acknowledges the NetFail Signal

The Net Fail signal can be acknowledged using the web based management, or 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

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

3.8.1.1 Process Data Watchdog Function

A process data watchdog is integrated into the bus terminal to avoid uncontrolled setting/resetting of the Inline station outputs in the event of an error.
If outputs of the stations are set, ensure the controlling process access to the station. In an event of an error, e.g., network line interrupted or functional error in the controlling process, the bus terminal can respond appropriately. 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 exptected during timeout (default: 500 ms). During error-free operation, the write process is realized 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 occured. Two responses follow:

- The selected fault response mode is executed
- and the net fail signal is set.

The reason for setting the NetFail signal is listed in the reason code (see list on page 3-36).

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 can be acknowledged using the web-based management or using 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 to 65,000 ms. Timeout can be set to the object ID 0x2293 using the web-based management or 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 terminal is in "INIT" state. For switching off, the value of timeout is set to "Zero". The required fault response mode can also 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.

Status Diagram of the Process Data Watchdog

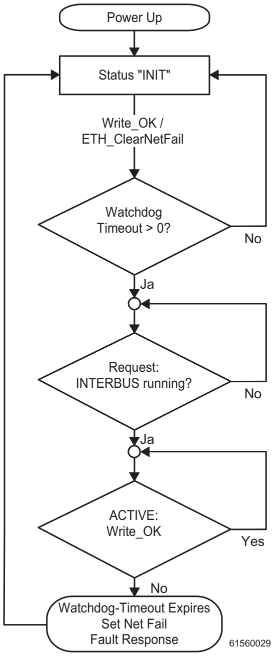


Figure 3-11 Status diagram of the process data watchdog

3.8.2 Connection Monitoring (Host Checking)

Application

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

- Cable broken, not connected or short-circuited.
- Transceiver faulty.
- Errors or faults in the Ethernet adapter of the bus terminal 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 so-called 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 terminal, 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 change according 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 terminal waits for a user-defined time for the reply to be received. If the reply is not received within this time, the bus terminal 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 terminal 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 (PC)

☞ On a PC with Windows as operating system, an echo server is running if TCP/IP has been installed. You will find these services under ...\system control\network\services. The user must ensure that the echo server answers within 500 ms in every operating state. The echo server implemented per default in Windows 2000 does not meet these requirements. For this reason, the user should use DTI monitoring for connection monitoring.

ETH_SetHostChecking

Task:

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

If the client does not respond within the predefined time (timeout time), 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 terminal.

Syntax:

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

Parameters:

IBDDIHND nodeHd	Node handle (MXI or DTI) for the bus terminal 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 for timeout time is 65,000 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)
          {
               /*hcSelected time is too short
               (330 ms, minimum)*/
               .
               .
               .
          }
     }
     UpdateData (FALSE)
}
```

ETH_ClearHostChecking

Task: The *ETH_ClearHostChecking* 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 *ETH_SetHostChecking*. 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);

Parameters:	IBDDIHND nodeHd	Node handle (MXI or DTI) for the bus terminal 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 terminal (application program) is still operating correctly. An extremely serious error occurs when the controlling process is no longer operating correctly, i.e., the bus terminal 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 terminal 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 not

therefore enable monitoring for another connection.
Once access has been enabled for the first time, all subsequent access must be enabled within the set timeout time, 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 terminal 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 terminal 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 terminal.

ETH_SetDTITimeoutCtrl

Task:	The *ETH_SetDTITimeoutCtrl* 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 (**time*) to a variable with the desired timeout time. After the function has been called, the timeout time calculated by the bus terminal can be found in the *USIGN16 *time* variable.	
Syntax:	IBDDIRET IBDDIFUNC ETH_SetDTITimeoutCtrl (IBDDIHND nodeHd, USIGN16 *time);	
Parameters:	IBDDIHND nodeHd	Node-Handle (DTI) der Busklemme, die überwacht werden soll.
	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 of 110 ms up 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 *ETH_ClearDTITimeoutCtrl* 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);

Parameters:

IBDDIHND nodeHd

Node handle (DTI) for the bus terminal 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 beforehand. Use the DDI command "Set_Value" on the object ID 2277_{hex} .The following table shows the three possible reactions:

Table 3-5 Available fault response modes

Fault Response Mode	Value	Function
Reset Fault Mode (Default)	**1**	The digital 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 FL IL 24 BK-B-PAC only has one internal volatile memory where the process data are stored during runtime. This memory image is displayed cyclically onto the appropriate Inline modules.

3.8.4.1 The Power Up Table

Table 3-6 Power up-sequence

Power Up Sequence				
Front ´View of the FL IL 24 BK	**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 onto an internal memory after power up.	"0” plus the new values	Internal memory	"0“ plus the new values	Internal memory
Operation	"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 3 I/O modules, an analog output module with a length of 16 bit (AO), a digital output module with a length of 16 bit (DO 16) and a digital output module with a length of 2 bit (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 onto the DO 16 module, we get the following output values:.

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

Then 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 onto the respective modules via several write accesses, we get the following output values:

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

Then this is the ""0" plus the sum of all new values" state.

3.8.4.2 The Connection Monitoring Table

This table shows the output values after the connection monitoring or the process data watchdog detected an error such as a disconnection or a communication error while the voltage supply remains the same.

Table 3-7 Connection monitoring table

Connection Monitoring Table after Connection Abort, a Cable Interrupt or a Communication Error.				
Configuration of the FL IL 24 BK	**Configuration): "Reset Fault Mode“**		**Configuration): "Last State Fault Mode“**	
	Internal memory	Actual output	Internal memory	Actual output
Cable or communication error removal after cable interrupt	Last values in the internal memory	All digital outputs are set to "0".	Last values in the internal memory	Values in the internal memory
First write access in the output table after restoring the connection	Last values in the internal memory plus the newly written values	Internal memory	Last values in the output table plus the newly written values	Internal memory
Operation	Last values in the internal memory plus all newly written values	Internal memory	Last values in the 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

If 0x00A1 is written into the internal memory of the DO 16 as first value after having restored the connection, we get the following actual output value:

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

This is the status "Last values in the internal memory plus the newly written values".

If values such as 0x0010 for AO, 0x0001 for DO 2 and 0xACDC for DO 16 have been written into the internal memory via several write accesses, we get the following output values:

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

This is the status "Last values in the internal memory plus the newly written values".

3.8.5 Handling the NetFail Signal / Testing With ETH_SetNetFail

The NetFail signal is set by writing a register in the coupling memory of the bus terminal. As soon as this signal has been detected by the bus terminal, all outputs of the local bus devices are set back.
Only after the NetFail signal has been set back to zero, the process data can be ouput again. The NetFail signal is always set if the connection to the client is interrupted, the bus terminal does not write data to the DTI within the specified time or a general malfunction has been detected on the bus terminal, which prevents safe operation.
The setting of the NetFail signal is indicated by setting the NetFail bit in the control word of each data telegram, which is sent by the bus terminal. The NetFail signal can be reset using the appropriate command or, if this is no longer possible, by executing a power up.

ETH_SetNetFail

Task:

The *ETH_SetNetFail* function sets the NetFail signal on the bus terminal and thus prevents the 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 terminal as a parameter.

Syntax:

IBDDIRET IBDDIFUNC ETH_SetNetFail (IBDDIHND nodeHd);

Parameters:	IBDDIHND nodeHd	Node handle (MXI or DTI) for the bus terminal 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 *ETH_GetNetFailStatus* function sends the NetFail status to the user, which is determined by the node handle of the bus terminal. The function is assigned a node handle for an open DTI or MXI data channel and a pointer to a *T_ETH_NET_FAIL* structure as parameters. After the function has been called successfully, the structure components contain the status (*status*) of the NetFail signal and an error code (*reason*) for triggering the Netfail signal if the NetFail signal has been set.

If the NetFail signal is not set, the *status* structure component has the value 0. Otherwise *status* has the value 0xFFFF. The *reason* structure component is only valid if the NetFail signal is set. The possible values for *reason* can be found in the IOCTRL.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 terminal 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; /* NetFailStatus */
     USIGN16 reason; /* Reason for the NetFail */
} T_ETH_NET_FAIL;
```

Possible values for the *status* structure component:

```
ETH_NET_FAIL_ACTIVE0xFFFF
/* NetFail signal triggered */
```

(See also "Causes for Fault Response" on page 3-36)

```
ETH_NET_FAIL_INACTIVE0x0000
/* 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.

Syntax: IBDDIRET IBDDIFUNC ETH_ClrNetFailStatus (IBDDIHND nodeHd);

Parameters: IBDDIHND nodeHd — Node handle (MXI or DTI) for the bus terminal 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.

> Observe the dependency of possible further active monitoring functions.

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 *netFailModeInfo);
```

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 *netFailModeInfo

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 data 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;
```

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 structure component *numOfBytes*, which contains the value zero in this case.

Syntax:

IBDDIRET IBDDIFUNC ETH_GetNetFailMode(IBDDIHND nodeHd,
T_ETH_NET_FAIL_MODE *netFailModeInfo)

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 *netFailModeInfo

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 IOCTRL.H file.

3.9 IN Process Data Monitoring

Functions that automatically monitor the process IN 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 process IN 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 process IN 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 process IN 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 the monitoring. A bit that is not set (0) means that a change in the associated bit in the process IN 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 PD IN 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 *ETH_DeactivatePDInMonitoring* function deactivates process IN data monitoring. The function is only assigned the node handle as a parameter, which is also used to activate monitoring with *ETH_ActivatePDInMonitoring*.

Syntax:

```
IBDDIRET IBDDIFUNC ETH_DeactivatePDInMonitoring (IBDDIHND
                      nodeHd);
```

Parameters:

IBDDIHND nodeHd	Node handle (DTI) for the controller board for which process data monitoring is to be deactivated.

Return value:

IBDDIRET	If the function is executed successfully, the value 0 (ERR_OK) 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 "process IN data monitoring" must always be deactivated before closing the connection (DDI_DevCloseNode).

3.10 Notification Mode

General:

The notification mode enables 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 the 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 FFFF_{hex} 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 the notification mode using this additional connection, the value FEDC_{hex} (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 be activated:

IBDDIRET IBDDIFUNC DDI_SetMsgNotification(IBDDIHND nodeHd, T_ETH_NOTIFY_INFO IBPTR *notifyInfoPtr)

Syntax to be deactivated:

IBDDIRET IBDDIFUNC DDI_ClrMsgNotification(IBDDIHND nodeHd, T_ETH_NOTIFY_INFO IBPTR *notifyInfoPtr)

UNIX

Parameters:

mode	Notification mode
processId	
threadId	
timeout	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 terminal 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).

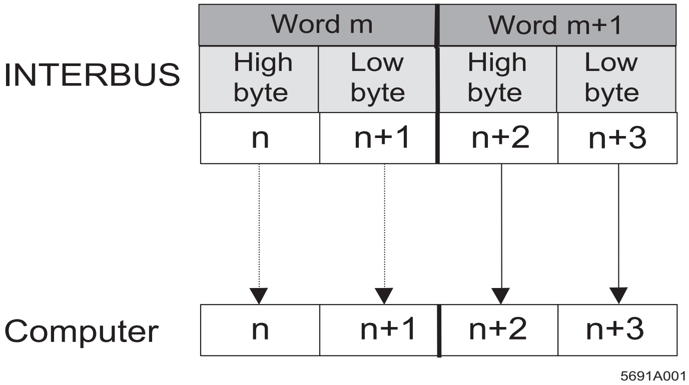


Figure 3-12 Assignment of the process data between the local bus and the computer systems

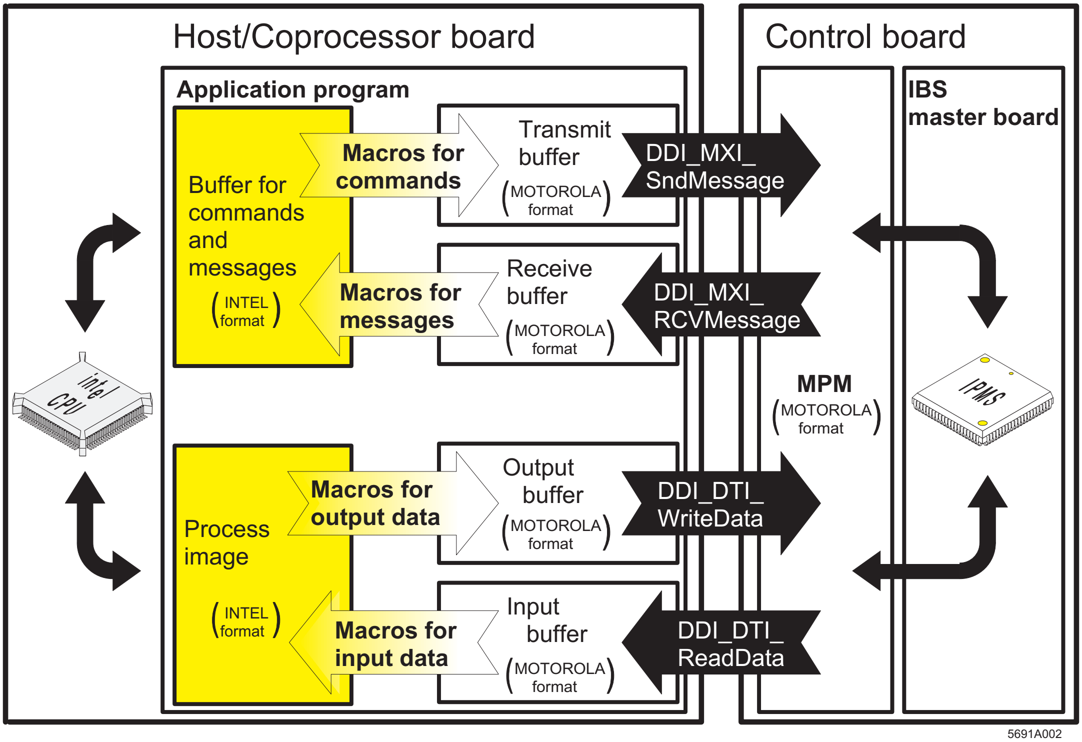


Figure 3-13 Using the macros for programming support

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

3.12 Description of the Macros

Table 3-8 Driver software macros

Macro	Task	Page
IB_SetCmdCode	Enters the command code (16-bit) in the specified transmit buffer	3-66
IB_SetParaCnt	Enters the parameter count (16-bit) in the specified transmit buffer	3-67
IB_SetParaN	Enters a parameter (16-bit) in the specified transmit buffer	3-67
IB_SetParaNHiByte	Enters the high-order byte (bit 8 to 15) of a parameter in the specified transmit buffer	3-67
IB_SetParaNLoByte	Enters the low-order byte (bit 0 to 7) of a parameter in the specified transmit buffer	3-67
IB_SetBytePtrHiByte	Returns the address of a parameter entry starting with the high-order byte (bit 8 to 15)	3-67
IB_SetBytePtrLoByte	Returns the address of a parameter entry starting with the low-order byte (bit 0 to 7)	3-69
IB_GetMsgCode	Reads a message code (16-bit) from the specified receive buffer	3-69
IB_GetParaCnt	Reads the parameter count (16-bit) from the specified receive buffer	3-69
IB_GetParaN	Reads a parameter (16-bit) from the specified receive buffer	3-69
IB_GetParaNHiByte	Reads the high-order byte (bit 8 to 15) of a parameter from the specified receive buffer	3-70
IB_GetParaNLoByte	Reads the low-order byte (bit 0 to 7) of a parameter from the specified receive buffer	3-70
IB_GetBytePtrHiByte	Returns the address of a parameter entry starting with the high-order byte (bit 8 to 15)	3-70
IB_GetBytePtrLoByte	Returns the address of a parameter entry starting with the low-order byte (bit 0 to 7)	3-71
IB_PD_GetLongDataN	Reads a double word (32-bit) from the specified position in the input buffer	3-72
IB_PD_GetDataN	Reads a word (16-bit) from the specified position in the input buffer	3-72

Table 3-8 Driver software macros

Macro	Task	Page
IB_PD_GetDataNHiByte	Reads the high-order byte (bit 8 to 15) of a word from the input buffer	3-72
IB_PD_GetDataNLoByte	Reads the low-order byte (bit 0 to 7) of a word from the input buffer	3-72
IB_PD_GetBytePtrHiByte	Returns the address of a word starting with the high-order byte (bit 8 to 15)	3-73
IB_PD_GetBytePtrLoByte	Returns the address of a word starting with the low-order byte (bit 0 to 7)	3-73
IB_PD_SetLongDataN	Writes a double word (32-bit) to the output buffer	3-73
IB_PD_SetDataN	Writes a word (16-bit) to the output buffer	3-74
IB_PD_GetDataNHiByte	Writes the high-order byte (bit 8 to 15) of a word to the output buffer	3-74
IB_PD_GetDataNLoByte	Writes the low-order byte (bit 0 to 7) of a word to the output buffer	3-74
IB_PD_GetBytePtrHiByte	Returns the address of a word starting with the high-order byte (bit 8 to 15)	3-74
IB_PD_GetBytePtrLoByte	Returns the address of a word starting with the low-order byte (bit 0 to 7)	3-74

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 No. (counting begins 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 No.
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 No.
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 No.
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 *USIGN8 ** data type.

Parameters:	n(USIGN8 *):	Pointer to the transmit buffer
	m(USIGN16):	Parameter No.
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.

Parameters:	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.

Parameters:	n(USIGN8 *):	Pointer to the receive buffer
Return value:	(USIGN16):	Parameter count

Remark: This macro only reads 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 No.

Return value:	(USIGN16):	Parameter value

Remark: This macro only reads the parameter value 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 No.
Return value:	(USIGN8):	Parameter value (byte)

Remark: This macro only reads the parameter value 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 No.
Return value:	(USIGN8):	Parameter value (byte)

Remark: This macro only reads the parameter value 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 No.
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 No.
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 a 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 a 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 of the 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 operating system related offset (ERR_BASE) is added to the the codes listed here. This offset has already been taken into consideration when using error message definitions.

Table 3-9 Driver software messages

Code	Error Message	Cause	Page
0000_{hex}	ERR_OK	The function was executed successfully	3-77
0085_{hex}	ERR_INVLD_NODE_HD	Invalid node handle specified	3-78
0086_{hex}	ERR_INVLD_NODE_STATE	Node handle of a data channel that is already closed specified	3-78
0087_{hex}	ERR_NODE_NOT_READY	Desired node not ready	3-78
0088_{hex}	ERR_WRONG_DEV_TYP	Incorrect node handle	3-78
0089_{hex}	ERR_DEV_NOT_READY	Local bus master not ready yet	3-79
$008A_{hex}$	ERR_INVLD_PERM	Access type not enabled for channel	3-79
$008C_{hex}$	ERR_INVLD_CMD	Utility function is not supported by driver Version 0.9	3-79
$008D_{hex}$	ERR_INVLD_PARAM	Command contains invalid parameter	3-79
0090_{hex}	ERR_NODE_NOT_PRES	Node not available	3-80
0091_{hex}	ERR_INVLD_DEV_NAME	Unknown device name used	3-80
0092_{hex}	ERR_NO_MORE_HNDL	Device driver resources used up	3-80
0096_{hex}	ERR_AREA_EXCDED	Access exceeds limit of selected data area	3-83
0097_{hex}	ERR_INVLD_DATA_CONS	Specified data consistency is not permitted	3-83
$009A_{hex}$	ERR_MSG_TO_LONG	Message or command contains too many parameters	3-81
$009B_{hex}$	ERR_NO_MSG	No message present	3-81
$009C_{hex}$	ERR_NO_MORE_MAILBOX	No further mailboxes of the required size free	3-81
$009D_{hex}$	ERR_SVR_IN_USE	Send vector register in use	3-82
$009E_{hex}$	ERR_SVR_TIMEOUT	Invalid node called	3-82

Table 3-9 Driver software messages

Code	Error Message	Cause	Page
009F$_{hex}$	ERR_AVR_TIMEOUT	Invalid node called	3-82
00A9$_{hex}$	ERR_PLUG_PLAY	Invalid write access to process data in P&P mode	3-83
0100$_{hex}$	ERR_STATE_CONFLICT	This service is not permitted in the selected operating mode of the controller	3-83
0101$_{hex}$	ERR_INVLD_CONN_TYPE	Service called via an invalid connection	3-84
0102$_{hex}$	ERR_ACTIVATE_PD_CHK	Process IN data monitoring could not be activated	3-84
0103$_{hex}$	ERR_DATA_SIZE	The data volume is too large	3-84
0200$_{hex}$	ERR_OPT_INVLD_CMD	Unknown command	3-84
0201$_{hex}$	ERR_OPT_INVLD_PARAM	Invalid parameter	3-84
1010$_{hex}$	ERR_IBSETH_OPEN	The IBSETHA file cannot be opened	3-85
1013$_{hex}$	ERR_IBSETH_READ	The IBSETHA file cannot be read	3-85
1014$_{hex}$	ERR_IBSETH_NAME	The device name cannot be found in the file	3-85
1016$_{hex}$	ERR_IBSETH_INTERNET	The system cannot read the computer name/ host address	3-85

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 terminal 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: Request a reset of 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 generated if you are working with older driver libraries or older DLLs.

Remedy: Use an up-to-date 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_PRES **0090_{hex}**

Cause: An attempt was made to open a data channel to a node, which is not present.

Remedy: Select the following node.

IBS ETH: Node 1 = Local bus master

ERR_INVLD_DEV_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_HNDL **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 messages are 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 space of time.

Remedy: Increase the time interval between individual mailbox requests and start the service: DDI_MXI_SndMessage once more.

Cause 2: No further mailboxes of the required size are available. Note 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 Kbyte in one call.

Remedy: Only read or write data records with a maximum size of 4 Kbyte.

Cause 2: The upper area limit (4 Kbyte 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 byte)
DTI_DATA_LWORD	: Double word data consistency (4 byte)
DTI_DATA_64BIT	: 64-bit data consistency (8 byte)

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: Process IN 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 value.

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-B-PAC. The inputs are individually jumpered to the outputs. The ground potential is created by the internal potential jumper.

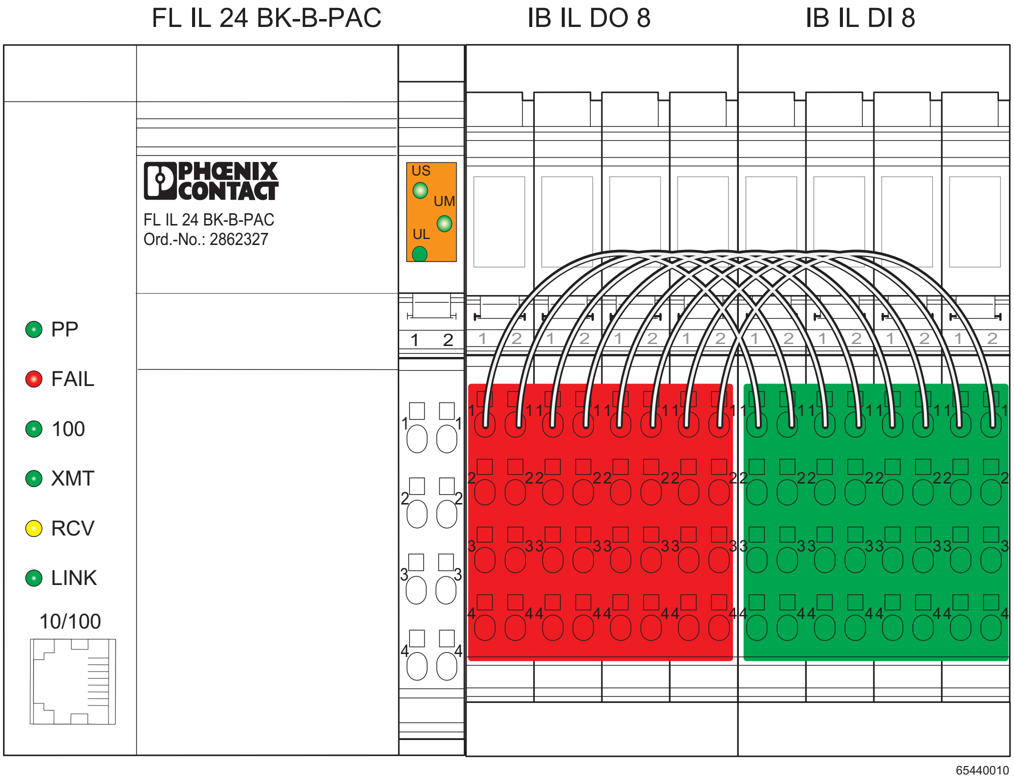


Figure 3-14 Structure of the station for the example program

3.16.1 Demo Structure Startup

The user is first prompted to specify the bus terminal 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.

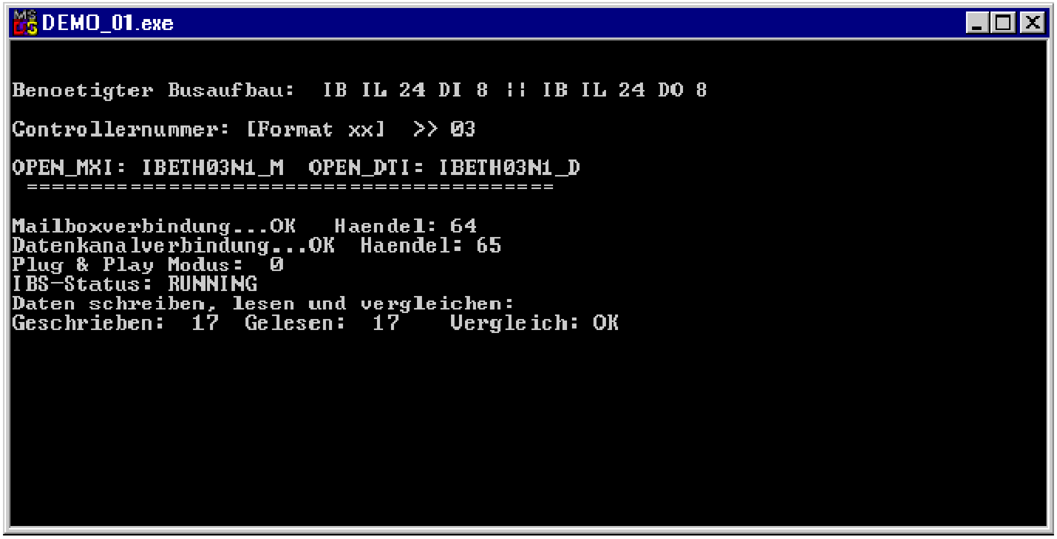
Function:

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. Jumpering 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.



```
Benoetigter Busaufbau:  IB IL 24 DI 8 !! IB IL 24 DO 8

Controllernummer: [Format xx]  >> 03

OPEN_MXI: IBETH03N1_M  OPEN_DTI: IBETH03N1_D
 =========================================

Mailboxverbindung...OK   Haendel: 64
Datenkanalverbindung...OK  Haendel: 65
Plug & Play Modus:  0
IBS-Status: RUNNING
Daten schreiben, lesen und vergleichen:
Geschrieben:  17  Gelesen:  17    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 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 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 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 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-B-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 reading the 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 resetting buffer. Error code: 0x%04X",locRet);
  }

  Sleep(100);

  //Loop for reading and writing 255 data items
  do
  {
  //Writing 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 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 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-B-PAC again
 locRet = DeleteConnection();

 printf("\nEND\n");

 Sleep(3000);

return 0;

}
```

Section 4

This section informs you about

– firmware functions

Firmware Services 4-3

4.1 Overview 4-3

4.1.1 Services That can be Used in Every Operating Mode 4-3

4.1.2 Services That are Only Available in Expert Mode..........4-4

4.2 Notes on Service Descriptions 4-4

4.2.1 Service "Name of the Service" 4-5

4.3 Services for Parameterizing the Controller Board 4-8

4.3.1 "Control_Parameterization" Service...............................4-8

4.3.2 "Set_Value" Service 4-10

4.3.3 "Read_Value" Service 4-12

4.3.4 "Initiate_Load_Configuration" Service 4-14

4.3.5 "Load_Configuration" Service 4-16

4.3.6 "Terminate_Load_Configuration" Service 4-20

4.3.7 "Read_Configuration" Service 4-22

4.3.8 "Complete_Read_Configuration" Service 4-29

4.3.9 "Delete_Configuration" Service.....................................4-32

4.3.10 "Create_Configuration" Service 4-33

4.3.11 "Activate_Configuration" Service 4-36

4.3.12 "Control_Device_Function" Service 4-38

4.3.13 "Reset_Controller_Board" Service 4-40

4.4 Services for Direct INTERBUS Access 4-42

4.4.1 "Start_Data_Transfer" Service......................................4-42

4.4.2 "Alarm_Stop" Service..4-44

4.5 Diagnostic Services..4-45

4.5.1 "Get_Error_Info" Service...4-45

4.5.2 "Get_Version_Info" Service 4-52

4.6 Error Messages for Firmware Services:.....................................4-56

4.6.1 Overview 4-56

4.6.2 Positive Messages 4-57
4.6.3 Error Messages.. 4-57

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. If the services are not used as specified in the table, this 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 That can be Used in Every Operating Mode

Table 4-1 Overview of the services that can be used in every operating mode

Code	Services	Page
0309$_{\text{hex}}$	Read_Configuration	4-22
030B$_{\text{hex}}$	Complete_Read_Configuration	4-29
0316$_{\text{hex}}$	Get_Error_Info	4-45
032A$_{\text{hex}}$	Get_Version_Info	4-52
0351$_{\text{hex}}$	Read_Value	4-12
0714$_{\text{hex}}$	Control_Device_Function	4-38
0750$_{\text{hex}}$	Set_Value	4-10
0956$_{\text{hex}}$	Reset_Controller_Board	4-40

4.1.2 Services That are Only Available in Expert Mode

Table 4-2 Services that are only available in expert mode

Code	Services	Page
0306_{hex}	Initiate_Load_Configuration	4-14
0307_{hex}	Load_Configuration	4-16
0308_{hex}	Terminate_Load_Configuration	4-20
$030C_{hex}$	Delete_Configuration	4-32
$030E_{hex}$	Control_Parameterization	4-8
0701_{hex}	Start_Data_Transfer	4-42
0710_{hex}	Create_Configuration	4-33
0711_{hex}	Activate_Configuration	4-36
1303_{hex}	Alarm_Stop	4-44

4.2 Notes on Service Descriptions

Use of 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 an 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 Service "Name of the 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 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	Code
Word 2	Parameter_Count
Word 3	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$_{hex}$	Initiate the parameterization phase
		0000$_{hex}$	Terminate 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 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

4.3.2 "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 parameters	Value/Comment
2216_{hex}	Up-to-date 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 deactivated 200 - 65000: Timeout time in ms

Table 4-4 Available fault response modes

Fault Response Mode	Valu e	Function
Reset Fault Mode (Default)	**1**	The digital 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.

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$_{\text{hex}}$ 1 parameter word

with a negative message:

0002$_{\text{hex}}$ 2 parameter words

Result: Result of the service processing

0000$_{\text{hex}}$ Indicates a positive message. The controller board executed the service successfully.

xxxx$_{\text{hex}}$ 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 4-10).

Syntax: **Read_Value_Request** **0351$_{\text{hex}}$**

Word 1	Code	
Word 2	Parameter_Count	
Word 3	Variable_Count	
Word 4	Variable_ID	1. Parameter
Bit	15 0	

Key:

Code: 0351$_{\text{hex}}$ 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 parameter
Word 6	Value	

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 *Result* 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 boards to transmit a configuration with either the "Load_Configuration“ (0307_{hex}) or the "Complete_Load_Configuration“ ($030A_{hex}$) service onto the INTERBUSmaster.

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 + (*Extension_Length* + 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 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

4.3.5 "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 the *Used_Atrributes* 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 4-17).

Settings for the*Used_Attributes* parameter:

Bit 0	Device number
Bit 1	Device code

Example:
If the entries only 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 4-17)

☞ According to the following syntax, 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 ranges:
00_{hex} ... $3F_{hex}$ (63_{dec}) for an Inline station
The *Bus_Segment_No* and *Position* parameters together form the device number.

Length_Code: Length code
The length code refers to the address space required by the device in the host.

ID_Code: 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 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) is 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	Code
Word 2	Parameter_Count
Word 3	Default_Parameter
Bit	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 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

0003_{hex} Automatic parameterization of the processd data 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 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

4.3.7 "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 *Frame_Reference* parameter.
0001_{hex}	$>0000_{hex}$	Entries of the configuration frame (CFG_Entry) selected with the *Frame_Reference* 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 in the physical bus structure

Only relevant if *Frame_Reference* > 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 *Used_Attributes* 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 *Frame_Reference* > 0000_{hex} and *Start_Entry_No* = 0000_{hex}:

$000D_{hex}$ 12 parameter words

with a positive message and if *Frame_Reference* > 0000_{hex} and *Start_Entry_No* > 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 *Result* parameter indicates why the service could not be executed (see also *Add_Error_Info*).

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 that was transferred with 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

Used_Attributes: Read attributes
The parameter contains the value that was transferred with 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 the service confirmation.
The *More_Follows* parameter indicates if there are further 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 *Used_Attributes* parameter (see the "Configuration_Entry" syntax on page 4-27).

☞ 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: Length code
The length code refers to the address space required by the INTERBUS device in the host.

ID_Code: ID code
The ID code describes the INTERBUS device function. It is printed as *Module Ident* 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$_{\text{hex}}$**

Word 1	Code
Word 2	Parameter_Count
Word 3	Used_Attributes
Bit	15 0

Key:

Code:	030B$_{\text{hex}}$	Command code of the service request
Parameter_Count:	Number of subsequent words	
	0001$_{\text{hex}}$	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 attribute that you want to 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 you want to 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 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

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
The parameter contains the value that was transferred with the service request.

Start_Entry_No: Number of the first entry.

0001hex 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 *Used_Attributes* parameter. For the description of the *Configuration_Entry* parameters see "Read_Configuration" service (0309_{hex}) on page 4-22.

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 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	

4.3.10 "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.

The new configuration frame and the active 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:

Device_Number:	According to the active configuration
Length_Code:	According to the active configuration
ID_Code:	According to the active configuration
Device_Level:	According to the active configuration
Group_Number:	For all INTERBUS devices $FFFF_{hex}$ (No group numbers are supported)
Device_State:	All INTERBUS devices are active

Syntax: **Create_Configuration_Request** **0710_{hex}**

Word 1	Code
Word 2	Parameter_Count
Word 3	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 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

4.3.11 "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

need to be checked.

If no errors are detected, the controller board activates this configuration frame 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: **Activate_Configuration_Request** **0711_{hex}**

Word 1	Code
Word 2	Parameter_Count
Word 3	Frame_Reference
Bit	15 0

Key:

Code:	0711_{hex}	Command code of the service request
Parameter_Count:	Number of subsequent words	
	0001_{hex}	1 parameter word
Frame_Reference:	0001_{hex}	

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 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

4.3.12 "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 an alarm output.

Syntax: **Control_Device_Function_Request** **0714$_{hex}$**

Word 1	Code	
Word 2	Parameter_Count (n)	
Word 3	Device_Function	
Word 4	Entry_Count	List of INTERBUS devices
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.
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 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

4.3.13 "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 always executes a cold restart.	

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 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

4.4 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.

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 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

4.4.2 "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.

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 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}**

Word 1	Code
Word 2	Parameter_Count
Bit	15 0

Key:

Code: 0316_{hex} Command code of the service request

Parameter_Count: Number of subsequent words

0000_{hex} No parameter word

Syntax: **Get_Error_Info_Confirmation** **8316$_{hex}$**

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 × *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 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.

Entry_Count: 0001_{hex}

Error_Code: Information on the error type

Add_Error_Info: with positive message:
Error location (*Bus segment* **.** *Position*), if it could be located.
with negative message:
Additional information on the error cause via error codes

Table 4-5 **Supported Error Codes**

Code	Error Type	Page
0x0A1C	E_SM_CFG_NUM_OF_DEV_TOO_BIG	4-48
0x0A2E	E_SM_CFG_IND_ADDR_LIST_TOO_BIG	4-48
0x0B02	E_PNM12_STATE_CONFLICT	4-49
0x0BB1	E_PNM12_DEVICE_STATE	4-49
0x0D10	E_PNM12_CONFIG_MISSING_DEVICE	4-49
0x0D20	E_PNM12_CONFIG_MAU_FAIL_DO	4-50
0x0D28	E_PNM12_CONFIG_MAU_FAIL DI	4-50
0x0D4C	E_PNM12_CONFIG_INVALID_ID	4-50
0x0D80	E_PNM12_CONFIG_MULTI_ERR_OUT	4-51
0x0D9C	E_PNM12_CONFIG_LB_TOO_LONG_OUT	4-51
0xFFFF	CONTROLLER_DEVICE_NUMBER	4-51

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 devices is 63.

Add_Error_Info: Number of specified or connected INTERBUS devices.

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:

1. Maybe
   - there is an empty configuration frame or
   - the first device behind the bus coupler is defect or is missing.

Remedy:

1. - Activate a correct configuration frame
   - Use the first device or aother functioning 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 0D10hex

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 0D20hex

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:

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 *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 kernel (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 version)

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 (e.g., FL IL 24 BK)	(20 byte)
Host_Version:	Version of the host-specific firmware interface (4 byte)	
Host_State:	Status of the host-specific firmware interface (6 byte)	
Host_Date:	Creation date of the host-specific firmware interface	(6 byte)
Host_Time:	Creation time of the host-specific firmware interface	(6 byte)
Start_FW_Version:	Version of the start firmware	(4 byte)
Start_FW_State:	Status of the start firmware	(6 byte)
Start_FW_Date:	Creation date of the start firmware	(6 byte)
Start_FW_Time:	Creation time of the start firmware	(6 byte)
HW_Art_No:	Order No. of the controller board	(8 byte)
HW_Art_Name:	Order designation of controller board	(30 byte)
HW_Motherboard_ID:	Identification of the motherboard (e.g., 32 43_{hex} for "2C"	(4 byte)
HW_Version:	Version of the hardware	(2 byte)
HW_Vendor_Name:	Manufacturer of the controller board	(20 byte)
HW_Serial_No:	Serial number of the controller board	(12 byte)
HW_Date:	Creation date of the controller board	(6 byte)

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	4-57
0907_{hex}	NO_OBJECT	4-57
0918_{hex}	UNKNOWN_CODE	4-57
0922_{hex}	ACTION_HANDLER_CONFLICT	4-57
$090A_{hex}$	INCORRECT_PARACOUNT	4-58
$091D_{hex}$	ACTION_HANDLER_OVERLAP	4-58
$0A02_{hex}$	INCORRECT_STATE	4-58
$0A18_{hex}$	INCORRECT_ATTRIB	4-58
$0A19_{hex}$	FRAME_NOT_SO_BIG	4-58
$0A22_{hex}$	INCORRECT_TN_NUMBER	4-58
$0A2F_{hex}$	DEVICE_ZERO	4-59
$0A51_{hex}$	INCORRECT_FRAME_REF	4-59
$0E22_{hex}$	INTERNAL_TIMEOUT	4-59
$0E23_{hex}$	FUNCTION_REG_NOT_FREE	4-59
$0E24_{hex}$	ACTION_ERROR	4-59

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 successfully.
Additional info 0005_{hex}:Bus data could not be detected.
Additional info $00A5_{hex}$: The configuration could not be activated.

Section 5

This section informs you about

– functions of the Modbus/TCP protocols

Modbus/TCP Protocol 5-3

5.1 Modbus Protocol 5-4

5.1.1 Modbus Connections 5-4

5.1.2 Modbus Interface 5-4

5.1.3 Modubus Conformity Classes 5-4

5.1.4 Modbus Message Format 5-5

5.1.5 Modbus Byte Sequence 5-5

5.1.6 Modbus Bit Sequence 5-6

5.2 Modbus Function Codes 5-6

5.3 Modbus Table 5-6

5.3.1 Example: Position of the Input / Output Data 5-8

5.4 Executable Functions 5-9

5.5 Supported Function Codes 5-10

5.5.1 Read Multiple Registers 5-10

5.5.2 Write Multiple Registers 5-12

5.5.3 Read Coils 5-13

5.5.4 Read Input Discretes 5-14

5.5.5 Read Input Registers 5-15

5.5.6 Write Coil 5-16

5.5.7 Write Single Register 5-17

5.5.8 Read Exception Status 5-19

5.5.9 Data Format of the Exception Status 5-19

5.5.10 Exception Responses 5-20

5.5.11 Write Multiple Coils 5-21

5.5.12 Read/Write Register 5-22

5.6 Reserved Registers for Command and Status Words 5-24

5.6.1 Command Word 5-24

5.6.2 Status Word 5-25

5.6.3 Diagnostics Using the Analog Input Table 5-26
5.6.4 Error Table 5-26
5.7 Monitoring 5-28
5.8 Modbus Monitoring...................5-29
5.9 I/O Fault Response Mode 5-30
5.9.1 The Power Up Table 5-31
5.9.2 The Connection Monitoring Table...................5-33

5 Modbus/TCP Protocol

This section describes the realization of the Modbus / TCP communication on the FL IL 24 BK-B-PAC.

Modbus Protocol

- Modbus connections
- Modbus interface
- Modubus conformity 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 Coil
- Write Single Register
- Read Exception Status
- Write Multiple Coils
- Read Write Registers

5.1 Modbus Protocol

The bus coupler supports a Modbus / TCP server with the following features:

5.1.1 Modbus Connections

The FL IL 24 BK-B-PAC supports up to 8 connections simultaneously. Thanks to this capacity, a connection can be restored quickly. This implies that the client can successfully restore an interrupted Modbus connection.

5.1.2 Modbus Interface

The Modbus communication via the FL IL 24 BK-B-PAC is supported via the Modbus interface in accordance with standard port 502.

5.1.3 Modubus Conformity Classes

The FL IL 24 BK-B-PAC supports the
Modbus conformity classes 0 and 1.

5.1.4 Modbus Message Format

The Modbus/TCP protocol has a special message format with the following structure:

Table 5-1 Modbus Message Format

Byte No.	Meaning
BYTE 0 – 1	Transaction identifier: unique ID, generated by the client
BYTE 2 – 3	Protocol identifer = 0
BYTE 4	Length field (upper byte) = 0 (all messages < 256)
BYTE 5	Length field (lower byte) = number of the following bytes
BYTE 6	Unit identifier
BYTE 7	Modbus function code
BYTE 8	In data if required

> The test fields "CRC 16" or "LRC" that usually are connected with Modbus are not required for Modbus/TCP because the test sum mechanism for TCP/IP and the safety layers are used to test the transmission of data packets.

5.1.5 Modbus Byte Sequence

Modbus uses the "Big Endian" format to display addresses and data elements. This means that the most significant byte is sent first if a numeric value (as individual or double word) that is larger than an individual byte is transmitted. Example:
The amount 0x1234 is transmitted in the following order: 0x12 0x34.
The amount 0x12345678 is transmitted in the following order: 0x12 0x34 0x56 0x78.

5.1.6 Modbus Bit Sequence

If a bit sequence is read into a register (for example %1 up to %I16), the bit with the highest number (%I16 in this example) is the least significant bit. The bit with the lowest number (%I1 in this example) is the most significant bit.

5.2 Modbus Function Codes

The following function codes are supported:

Table 5-2 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

5.3 Modbus Table

The definition of the reference tables for the Modbus protocol differs from the internal structure of the FL IL 24 BK-B-PAC tables. Modbus refers to a table of registers, input registers, discrete inputs as well as coils while the FL IL 24 BK-B-PAC refers to a table of digital inputs (%I), coils (%Q), analog inputs (%AI), analog outputs (%AQ) and special registers. The following table shows that every Modbus table is illustrated in the FL IL 24 BK-B-PAC tables. Please observe that all data in this table refer to the physical memory in the FL IL 24 BK. The FL IL 24 BK memory contains Modbus names. For example, if you output the "Read Input Discretes" to read the inputs in the table of the Modbus input discretes, the internal FL IL 24 BK table %I that is shown in the table of the Modbus input discretes will actually be read.

Table 5-3 Modbus Reference Tables

	Modbus Register Tables	**Register Tables of Modbus Inputs**	**Modbus Input Discretes Table**	**Modbus Output Tables**	**Internal FL IL 24 BK-B-PAC Tables**
Process Data	0 – 191 (16 bit words)	0 – 191 (16 bit words)	0 – 3071 (Bit)	---	%I1 – 3072 (Bit)
	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 (Bit)	%Q1 – 3072 (Bit)
	576 - 767 (16 bit words)	576 - 767 (16 bit words)	---	---	%AQ1 – 192 (16 bit words)
Special registers	1024 – 1087 (16 bit words)	1024 – 1087 (16 bit words)	---	---	Error table (32 errors x two 16-bit words per error)
	1280 (16 bit words)	1280 (16 bit words)	---	---	Timeout table, timeout value for connection monitoring
	2000 (16 bit words)	2000 (16 bit words)	---	---	Process data watchdog timeout
	2002 (16 bit words)	2002 (16 bit words)	---	---	Fault response mode
	2004 (16 bit words)	2004 (16 bit words)	---	---	NetFail rReason

5.3.1 Example: Position of the Input / Output Data

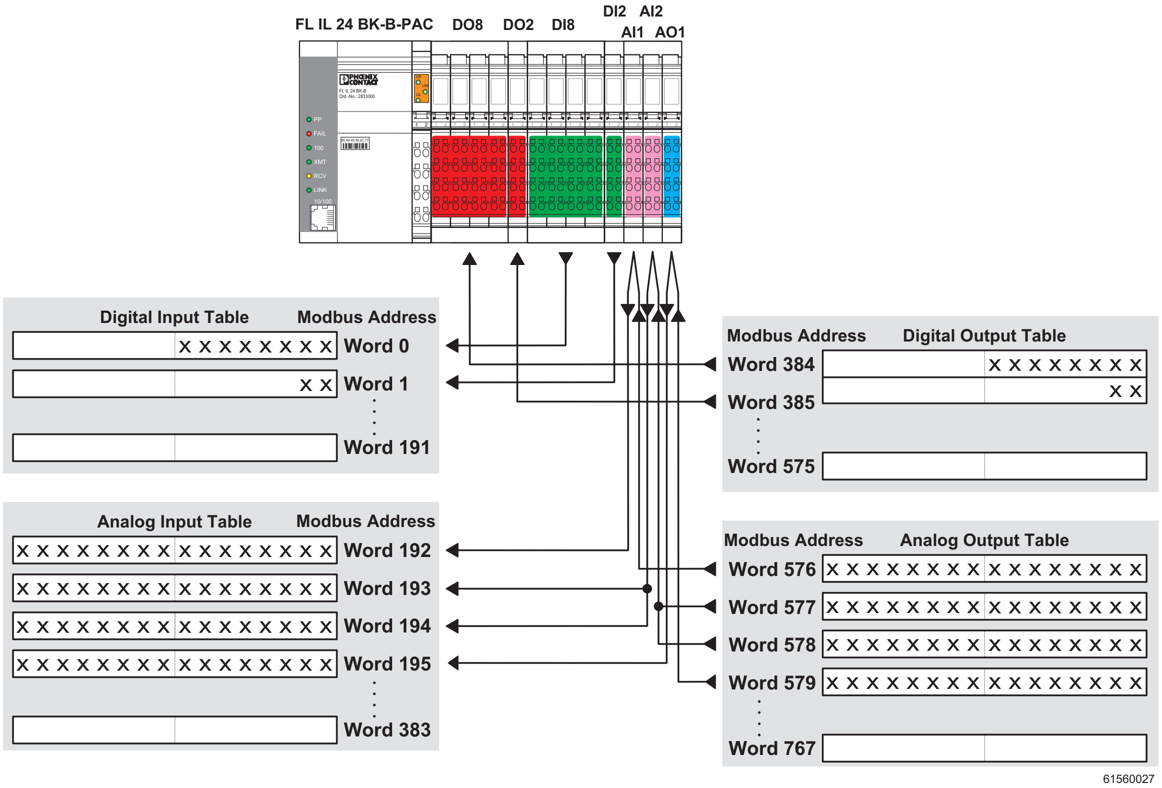


Figure 5-1 Position of the input / output data modules

Regarding data assignment, please observe that some Inline modules are configured via process data and thus occupy corresponding words in the Modbus tables.

5.4 Executable Functions

The FL IL 24 BK-B-PAC does not differentiate between Modbus register tables and Modbus input register tables. The Modbus register tables and the Modbus input register tables are displayed in all four FL IL 24 BK-B-E/A tables as well as in the error table.

Table 5-4 Executable Functions

Function	Function Code	READ/ WRITE	I_TAB.	AI_TAB.	Q_TAB.	AQ_TAB.	Special Register
Read Multiple Registers	3	READ	X	X	X	X	X
Read Input Registers	4	READ	X	X	X	X	X
Write Multiple Registers	16	WRITE	---	---	X	X	X
Read Coils	1	READ	---	---	X	---	---
Read Input Discretes	2	READ	X	---	---	---	---
Write Coil	5	WRITE	---	---	X	---	---
Write Single Register	6	WRITE	---	---	X	X	X
Read Exception Code	7	READ	---	---	---	---	---
Write Multiple Coils	15	WRITE	---	---	X	---	---
Read/Write Registers	23	READ/ WRITE	X	X	X	X	X

5.5 Supported Function Codes

The function codes are defined for Modbus memory mapping. For this reason, Table 5-3 is practical for the specification of the appropriate areas. This table shows the mapping of the designations in the Modbus tables via the appropriate designations in the FL IL 24 BK-B-PAC tables.

The FL IL 24 BK-B-PAC supports the following Modbus function codes:

- Read Multiple Registers (function code 3)
- Write Multiple Registers (function code 16)
- Read Coils (function code 1)
- Read Input Discretes (function code 2)
- Read Input Registers (function code 4)
- Write Coil (function code 5)
- Write Single Register (function code 6)
- Read Exception Status (function code 7)
- Write Multiple Coils (function code 15)
- Read/Write Registers (function code 23)

The following descriptions of the function commands and response messages start with the Modbus function codes (byte 0 is byte 7 of the Modbus message format). See "Modbus message format".

5.5.1 Read Multiple Registers

This command reads 16-bit words from 1 to 125 in the Modbus register table. Every part of the Modbus register table can be read using this function. When reading the error table, however, the entire table must be read. The Read Multiple Registers command has the following format:

Table 5-5 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 5-6 Answer to "Read Multiple Registers"

Byte No.	Meaning
BYTE 0	Function code = 3
BYTE 1	Byte Count of the 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 5-7 Answer to "Read Multiple Registers"

Byte No.	Meaning
BYTE 0	Function code = 0x83
BYTE 1	Exception response = 2

5.5.1.1 Example 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 error table.

Every combination of the register table offset and the Word Count that have access onto the offets > 767 und < 1024 results in an exception response. An exception response is also created when trying to read the erorr table and to enter a register table offset >1024 or a Word Count<> 64.

The special register 1280 - 2004 can only be read when the Word Count equals one.

5.5.2 Write Multiple Registers

This command reads 16-bit words from 1 to 100 in the Modbus Register table. Only that part of the Modbus Register table mapped to the %Q and %AQ I/O tables can be written using this function.
The Write Multiple Registers command has the following format:

Table 5-8 Write Multiple Register

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 the response (Byte Count =2x Word Count)
BYTE 6 – (B + 5)	Register values

The response to the Read Multiple Registers command has the following format:

Table 5-9 Answer to "Write Multiple Registers"

Byte No.	Meaning
BYTE 0	Function code = 0x10
BYTE 1 - 2	Register table offset (as in the command)
BYTE 3 - 4	Word Count (as in the command)

If the command accesses an invalid offset or receives an invalid length, an exception response with the following format is output:

Table 5-10 Exception response to "Write Multiple Registers"

Byte No.	Meaning
BYTE 0	Function code = 0x90
BYTE 1	Exception response = 2

5.5.2.1 Example 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.

Every combination of the register table offset and Word Count that either accesses offset < 384 or > 767 results in an exception response.

5.5.3 Read Coils

This command reads from 1 to 2000 bits from the Modbus register table. The Read Coils command has the following format:

Table 5-11 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 5-12 Answer to "Read Coils"

Byte No.	Meaning
BYTE 0	Function code = 1
BYTE 1	Byte Count of the response, Byte Count (B) = (Bit Count of the 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 receives an invalid length, an exception response with the following format is output:

Table 5-13 Exception response to "Read Coils"

Byte No.	Meaning
BYTE 0	Function code = 0x81
BYTE 1	Exception response = 2

5.5.3.1 Example 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.

Every combination of the Coil table offset and the Bit Count that accesses an offset > 3072 results in an exception response.

5.5.4 Read Input Discretes

This command reads from bit 1 to 2000 from the Modbus coil table.

The Read Input Discretes command has the following format:

Table 5-14 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 5-15 Answer to "Read Input Discretes"

Byte No.	Meaning
BYTE 0	Function code = 2
BYTE 1	Byte Count of the response, B = (Bit Count of the 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 receives an invalid length, an exception response with the following format is output:

Table 5-16 Exception response to "Read Input Discretes"

Byte No.	Meaning
BYTE 0	Function code = 0x82
BYTE 1	Exception code

5.5.4.1

Examples for Read Digital Coils:

Input Discrete table offset = 0 and Bit Count = 1 returns input discrete %I1.
Input Discrete Table offset = 0 and Bit Count = 2000 returns input discrete values %I1-2000.
Input Discrete Table offset = 4 and Bit Count = 13 returns input discrete values %Q5-17.
Every combination of the Input Discretes table offset with Bit Count that accesses offset > 3072 results in an exception response.

5.5.5 Read Input Registers

This command reads 16-bit words from 1 to 125 in 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 5-17 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 5-18 Answer to "Read Input Registers"

Byte No.	Meaning
BYTE 0	Function code = 4
BYTE 1	Byte Count of the response (B =2x 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 5-19 Exception response to "Read Digital Input Registers"

Byte No.	Meaning
BYTE 0	Function code = 0x84
BYTE 1	Exception response = 2

5.5.5.1 Example for the Read Input Registers command:

For examples refer to the "Examples for Read Multiple Registers" section.

5.5.6 Write Coil

With this command, 1 bit is written into the Modbus coil table. The Write Coil command has the following format:

Table 5-20 Write Coil

Byte No.	Meaning
BYTE 0	Function code = 5
BYTE 1 - 2	Coil table offset
BYTE 3	= 0xFF for setting the Coil to ON (ON), = 0 for setting the coil to OFF (OFF)
Byte 4	= 0

The response to the Write Coil command has the following format:

Table 5-21 Answer to "Write Coil"

Byte No.	Meaning
BYTE 0	Function code = 5
BYTE 1 -2	Coil table offset (as in the command)
BYTE 3	= 0xFF for setting the Coil to ON (ON), = 0 for setting the coil to OFF (OFF)
Byte 4	= 0

If the command accesses an invalid offset, the exception response has the following format:

Table 5-22 Exception response to "Write Coil"

Byte No.	Meaning

BYTE 0	Function code = 0x85
BYTE 1	Exception code = 2

5.5.6.1 Example for the Write Coil command:

With the Coil table offset = 0 and the value = 0xFF, the coil %Q1 is set to ON (ON). With the coil table offset = 0 and the value = 0, the coil %Q1 is set to OFF (OFF).
Each > 3072 coil table offset results in an exception response.

5.5.7 Write Single Register

With this command, a 16-bit word is written into the Modbus register table. Only that part of the Modbus register table mapped to the %Q and %AQ I/O tables as well as the first word of the error table can be written using this function.

The Write Single Register command has the following format:

Table 5-23 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 5-24 Response to "Write Single Register"

Byte No.	Meaning
BYTE 0	Function code = 6
BYTE 1 - 2	Register table offset (as in the command)
BYTE 3 - 4	Register value (as in the command)

If the command accesses an invalid offset, the exception response has the following format:

Table 5-25 Exception response to "Write Single Register"

Byte No.	Meaning
BYTE 0	Function code = 0x86
BYTE 1	Exception response = 2

5.5.7.1 Example for Write Single Register:

With the register table offset = 384, the register value is written in %Q1-16.

With the register table offset = 576, the register value is written in %AQ1.

Register table offset = 1024 and register value = 0 clears the Fault table.

With the register table offset = 1280 and a register value between 200 and 65,000, a new timeout value for the Modbus/TCP connection is written.

With the register table offset = 2,000 and a register value between 200 and 65,000, a new timeout value for the process data watchdog is written.

With the offset 2002, the fault response mode can be set.

1: Reset fault mode

0: Standard fault mode

2: Hold last state mode

Any Register Table offset < 384 or (> 576 and < 1024) or > 1024 produces an exception response.

5.5.8 Read Exception Status

This command reads a 8-bit status of the FL IL 24 BK-B-PAC.

The Read Exception Status command has the following format:

Table 5-26 Read Exception Status

Byte No.	Meaning
BYTE 0	Function code = 7

The response to the Read Exception Status command has the following format:

Table 5-27 Answer to "Read Exception Status"

Byte No.	Meaning
BYTE 0	Function code = 7
BYTE 1	Exception status

5.5.9 Data Format of the Exception Status

Table 5-28 Data Format Exception Status

Byte No.	Meaning
BYTE 0 - 5	Free
BYTE 6	Exception status
BYTE 7	Non-occupied error

5.5.10 Exception Responses

Table 5-29 Exception Responses

N o.	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 can generate the exception response 2.
3	ILLEGAL DATA VALUE	The value of this request is invalid for this device.
4	DEVICE FAILURE	- The Plug & Play mode still is active and thus prevents that data can be written. - A NetFail has occurred. - In addition, a DDI device could be connected that has exclusive write access. In this case, it is not possible to write data via Modbus/TCP.

5.5.11 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 5-30 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 5-31 Response to "Write Multiple Coils"

Byte No.	Meaning
BYTE 0	Function code = 0x0F
BYTE 1 - 2	Coil table offset (as in the command)
BYTE 3 - 4	Bit Count (as in the command)

If the command uses an invalid offset, the following exception response is generated:

Table 5-32 Exception response to "Write Multiple Coils"

Byte No.	Meaning
BYTE 0	Function code = 0x8F
BYTE 1	Exception response = 2

5.5.11.1 Example for the "Write Multiple Coils" command:

Coil table offset = 0 and Bit Count = 2 with a value of 3 sets coils %Q1 and %Q2.
Coil table offset = 0 and Bit Count = 2 with a value of 0 sets back coils %Q1 and %Q2.

5.5.12 Read/Write Register

This command reads 1 up to 125 words from a 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 5-33 Read/Write Register

Byte No.	Description
BYTE 0	Function code = 0x17
BYTE 1 - 2	Read register table offset
BYTE 3 - 4	Number of words to be read (1 to 125)
BYTE 5 - 6	Write register table offset
BYTE 7 - 8	Number of words to be written (1 - 100)
BYTE 9	Number of bytes to be written (B = 2 x number of words to be written)
BYTE 10 - (B +9)	Write register values

The response to the "Read/Write Register" command has the following format:

Table 5-34 Answer to Read/Write Register

Byte No.	Description
BYTE 0	Function code = 0x17
BYTE 1	Byte Count (B = 2 x number of words to be read)
BYTE 2 - (B + 1)	Read register values

If the command accesses an invalid offset, the exception response has the following format:

Table 5-35 Exception response to "Read/Write Register"

Byte No.	Description
BYTE 0	Function code = 0x97
BYTE 1	Exception code

5.5.12.1 Examples for the Read/Write Register command:

Register table offset = 0 and Word Count = 2 returns values of the input discretes %1-32.
Register table offset = 575 and Word Count = 2 returns values of the coils Q3057-3072 and the analog output %AQ1.
Register table offset = 1024 and Word Count = 64 returns an error table.
Every access onto a combination of register table offset and Word Count between >767 and <1024 generates an exception response.

The attempt to read the error 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 when the Word Count equals one.
Register table offset = 384 and Word Count = 2 writes the register values on the coils %Q1-32.

Register table offset = 575 and Word Count = 2 writes register values on the coils %Q3057-3072 and the analog output %AQ1.

Every access onto a combination of Register table offset and Word Count between >384 and <767 generates an exception response. The exception is writing one word into registers 2000 and 2002.

5.6 Reserved Registers for Command and Status Words

5.6.1 Command Word

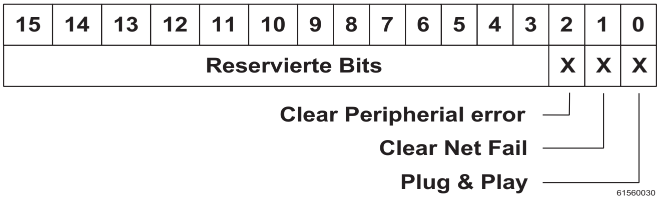
The last word of the table for analog outputs is automatically reserved as network interface command word via the bus terminal and starts using the Modbus address 40767. With this command word and via the Ethernet host controller, e.g. a PLC, the user can send commands with basic functions to the module. These commands enable startup without configuration software.

Table 5-36 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 Section Table 5-37. The remaining bits are reserved for later use. The activation/deactivation of the
Plug & Play mode is executed in the least significant bit of the command word. Bit 0 = "0“ -> PP deactivated; Bit 0 = "1“ -> PP activated.
A NetFail occurred, in this way the command word can be acknowledged by setting bit 1. If NetFail has been acknowledged successfully, bit 1 is reset to "zero".

Table 5-37 Network Interface Command Word



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reservierte Bits													X	X	X

5.6.2 Status Word

Table 5-38 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 are automatically reserved by the bus terminal as network interface status word. The user can extract up-to-date diagnostic information from this work using the Ethernet host controller, e.g. a PLC, without using a configuration software.
Only two of the least significant bits have a function. Bit 0 = "0" means that an error occured (e.g. a bus error). If bit 0 = "1", no error occured. Bit 1 indicates wether there is a NetFail (one) or not (zero).

Thus there are the following values for the status word:

0: An error occurred (e.g. bus error)
1: No error occurred.
2: A NetFail occurred.

Table 5-39 Status Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved bits														X	X

5.6.3 Diagnostics Using the Analog Input Table

Table 5-40 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 are entered into the analog input table. The diagnostic status register and the diagnostic parameter register occupy the last two words in the analog input table.

5.6.4 Error Table

Data Format of the Error Table

The Modbus client can access this internal error table that may contain 32 eror codes. This internal error table works accoring to the FIFO principle (**F**irst **I**n, **F**irst **O**ut). This means that the 33rd error entry deletes the oldest error entry.

An application can request all error entries or it can delete all entries via one command sent to the bus termimal. Every error entry is written in two words, beginning with the reference 1024 in the register table. All error entries serve as information and do not stop the bus terminal.

Reading the Error Table Data

The complete error table can be read out using the "Read Multiple Registers“command starting with the beginning of the error table (1024) with a length of 64 registers. It is impossible only to read parts of the error table. Empty entries contain the "0" value.

Please note that the entries are shifted downwards so that the latest error entry is located at position 1024.

Deleting the Error Table Data

If required, the application can write the value "0" into the first register (1024) of the error table using the "Write Single Register" command. You cannot write into any other register using the client.

Table 5-41 Registers

31..
.... 16
	15
	....
	 0

Diagnostic parameter register	Diagnostic status register

Entries in the Error Table

Every error entry is two words long and is positioned as follows:

If an error occurs, one or several bits are set within the diagnostic status register (PF, BUS or CTRL) and a new entry is added to the error table. The entry is displayed within the error table as shown below:

Table 5-42 Error Table

Error Table		
Error No.	Error 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

5.7 Monitoring

The three following monitoring mechanisms are available in the Modbus operating mode.

Table 5-43 Monitoring functions

Monitoring Mechanism	Monitoring …			
	... the Client Application	... the Individual Channels	... the Ethernet Connection	... the Process Data Exchange
Process data watchdog (process data monitoring),	X	- - -	X	X
Host checking	- - -	- - -	X	- - -
DTI / Modbus monitoring.	X	X	X	- - -

5.8 Modbus Monitoring

You can activate a monitoring mechanism for every Modbus/TCP connection so that the FL IL 24 BK-B-PAC can detect an error within a network (e.g. a defect cable) or a client (operating system crash or error in the TCP/IP protocol stack) and thus the module can respond accordingly. The monitoring mechanism is activated when reading or writing via the respective TCP connection for the first time.
In order to change the timeout value for the respective TCP connection, write the new timeout value into the timeout table to the special address 1280 either using the fc 6 or the fc 16 function. The value of this entry is the value of the timeout table. The time is indicated in milliseconds in the range of 200 ms up to 65,000 ms.
A timeout value of "0“ deactivates the monitoring function. Values between 1 and 199 as well as values larger than 65,000 ms generate the exception response 3 (ILLEGAL DATA VALUE).

☞ The connection monitoring is only activated using the new timeout values after the Modbus/TCP functions have been executed on the respective TCP connection.

After the first access via a Modbus/TCP function, all other accesses must be executed using the timeout value entered. Otherwise, the fault response mode is activated and the respective Modbus/TCP connection is closed.

5.9 I/O Fault Response Mode

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

Table 5-44 Available Fault Response modes

Fault Response Mode	Value	Function
Reset fault mode (Default)	1	The coils 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 as well as the actual output values for the first two options. One table regards the restart after power up and the other table regards the restart after an error occured. The output table is part of the internal memory of the bus terminal, current output values are the values of the output modules. The output table consists of two parts: digital and analog outputs.

5.9.1 The Power Up Table

The output table of the FL IL 24 BK-B-PAC is stored in a non-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 5-45 Power Up-Sequence

Power Up-Sequence				
Front View of the FL IL 24 BK	**Configuration): Reset Fault Mode**		**Configuration): Last State Fault Mode**	
	Output table	Actual output	Output table	Actual 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
Operation	“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, an analog output module with a length of 16 bit (AO), a coil module with a length of 16 bit (DO 16) and a coil module with a length of 2bit (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

Then 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, we get the following output values:

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

Then this is the ""0" plus the sum of all new values" state.

5.9.2 The Connection Monitoring Table

This table shows the output values after the connection monitoring or the process data watchdog detected an error such as a disconnection or a communication error while the voltage supply remains the same.

Table 5-46 Connection Monitoring Table

Connection Monitoring Table After Connection Abort, a Cable Interrupt or a Communication Error.				
Configuration of the FL IL 24 BK	**Configuration): "Reset Fault Mode"**		**Configuration): "Last State Fault Mode"**	
	Output table	Actual output	Output table	Actual output
Cable or communication error removal after cable interrupt	Last values in the output table	All coils 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
Operation	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

If 0x00A1 is written into the output table of the DO 16 as first value after having restored the connection, we get the following actual output value:

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

This is the status "Last values in the output table plus the newly written values".

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, we get the following output values:

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

This is the status "Last values in the output table plus the newly written values".

Section 6

This section informs you about

- technical data
- ordering data

Technical Data 6-3

6.1 Ordering Data 6-11

6 Technical Data

General Data	
Function	Ethernet / Inline bus coupler
Housing dimensions (width x height x depth)	90 mm x 72 mm x 116 mm (3.543 x 2.835 x 4.567 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, 2,000 m (6,561.66 ft.) above sea level
Air pressure (storage)	70 kPa to 108 kPa, 3,000 m (9,842.49 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	Via 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
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 1,500 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.

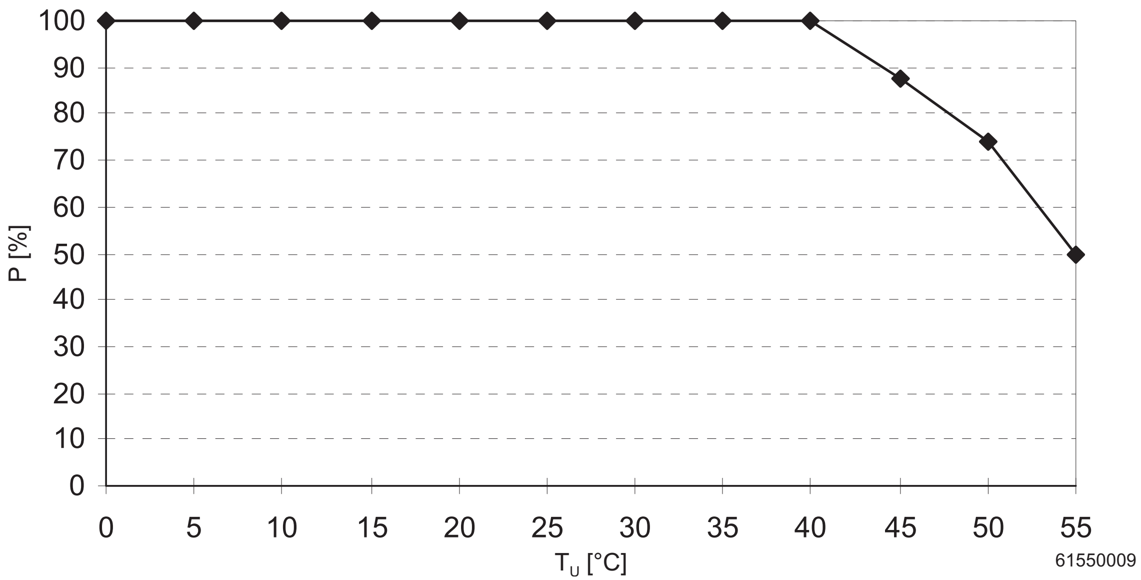
24 V Bus Coupler Supply	
Connection method	Spring-cage terminals
Recommended cable lengths	30 m (98.43 ft.), maximum; do not route cable through outdoor areas
Voltage continuation	Via potential routing U_L, U_{ANA}
Safety equipment	
Surge voltage	Input protective diodes (can be destroyed by permanent overload) Pulse loads up to 1,500 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
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 are 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 Routing)	
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
Derating of the Communications Power and the Analog Terminal Supply	



P [%] Loading capacity of the power supply unit for communications power and analog supply in %

T_A [°C] Ambient temperature in °C

Power Dissipation

Formula to Calculate the Power Dissipation of the Electronics

$$P_{EL} = P_{BUS} + P_{PERI}$$

$$P_{EL} = 2{,}6\ \mathrm{W} + (1{,}1 \frac{\mathrm{W}}{\mathrm{A}}\ \mathrm{x} \sum_{n=0}^{a} I_{Ln}) + (0{,}7 \frac{\mathrm{W}}{\mathrm{A}}\ \mathrm{x} \sum_{m=0}^{b} I_{Lm})$$

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)

$\sum_{m=0}^{b} I_{Ln}$ 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 W + 0.35 W = 2.55 W

2.55 W corresponds to 100% current carrying capacity of the power supply unit in the derating curves on page 6-6.

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 A, $I_{LAnalog}$ = 0.25 A

P_{PERI} = 1.1 W + 0.175 W

P_{PERI} = 1.275 W (corresponds to 50% of 2.55 W)

2. Possible logic current if the analog supply is not loaded:

P_{PERI} = 1.1 W/A x I_{LLogic} + 0 W

P_{PERI} / 1.1 W/A = I_{LLogic}

I_{LLogic} = 1.275 W / 1.1 W/A

I_{LLogic} = 1.159 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 1,500 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	
Limitation through software	63, maximum
Limitation through power supply unit	Maximum logic current consumption of the connected local bus modules: $I_{max} \leq 2$ 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 female connector 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 24 AWG)

Interfaces (Continued)	
Cable impedance	100 Ω
Transmission rate	10 / 100 Mbps
Maximum network segment expansion	100 m (328.08 ft.)

Protocols / MIBs	
Supported protocols	TCP/UDP BootP

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 ≥ 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.

6.1 Ordering Data

Description	Order Designation	Order No.
Ethernet / Inline bus coupler with connector and labeling field	FL IL 24 BK-B-PAC	28 62 32 7
Connector, with color print	IB IL SCN-8-CP	27 27 60 8
Labelling 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 45 55 4
RJ45 **gray** connector set for linear cable (2 pieces)	FL PLUG RJ45 GR/2	27 44 85 6
RJ45 connector set **green** 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

Description	Order Designation	Order No.
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
DIN EN 50022 DIN rail, 2 meters (6.56 ft.)	NS 35/7,5 PERFORATED NS 35/7,5 UNPERFORATED	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

We Are Interested in Your Opinion!

We would like to hear your comments and suggestions concerning this document.

We review and consider all comments for inclusion in future documentation.

Please fill out the form on the following page and fax it to us or send your comments, suggestions for improvement, etc. to the following address:

Phoenix Contact GmbH & Co. KG
Marketing Services
Dokumentation INTERBUS
32823 Blomberg
GERMANY

Phone	+49 - (0) 52 35 - 3-00
Telefax	+49 - (0) 52 35 - 3-4 18 08
E-Mail	tecdoc@phoenixcontact.com

FAX Reply

Phoenix Contact GmbH & Co. KG
Marketing Services
Dokumentation INTERBUS

Date: ____________________

Fax No: +49 - (0) 52 35 - 3-4 18 08

From:

Company:	____________________	Name:	____________________
	____________________	Department:	____________________
Address:	____________________	Job function:	____________________
City, ZIP code:	____________________	Phone:	____________________
Country:	____________________	Fax:	____________________

Document:

Designation: FL IL 24 BK-B UM E   Revision: 03   Order No.: 26 89 76 6

My Opinion on the Document

Form	**Yes**	**In part**	**No**
Is the table of contents clearly arranged?	□	□	□
Are the figures/diagrams easy to understand/helpful?	□	□	□
Are the written explanations of the figures adequate?	□	□	□
Does the quality of the figures meet your expectations/needs?	□	□	□
Does the layout of the document allow you to find information easily?	□	□	□
Contents	**Yes**	**In part**	**No**
Is the phraseology/terminology easy to understand?	□	□	□
Are the index entries easy to understand/helpful?	□	□	□
Are the examples practice-oriented?	□	□	□
Is the document easy to handle?	□	□	□
Is any important information missing? If yes, what?	□	□	□

____

Other Comments:

____

____