AXL F IF CAN 1H

**Axioline F, interface module,
CAN, transparent protocol,
max. transmission speed of 1 Mbps**

Data sheet
107308_en_04

1 Description

The module is designed for use within an Axioline F station. This module can be used to integrate a lower-level CAN bus system into the Axioline F station and therefore the bus system used.
This Axioline F module acts as the interface for the transparent reading and writing of CAN messages. With appropriate programming on the higher-level controller, it is suitable for CANopen®, J1939, NMEA 2000, and proprietary CAN protocols.

Features

- Transparent reading and writing of CAN messages
- Integrated buffer memory for 256 CAN messages in the receive direction and 64 CAN messages in the transmit direction
- Parameterizable filter function for 60 filters (11-bit CAN identifier) and 30 filters (29-bit CAN identifier)
- Device rating plate stored
- Diagnostic and status indicators

This data sheet is only valid in association with the UM EN AXL F SYS INST user manual.

Make sure you always use the latest documentation.
It can be downloaded at: phoenixcontact.net/product/2702668

2 Table of contents

3 Ordering data

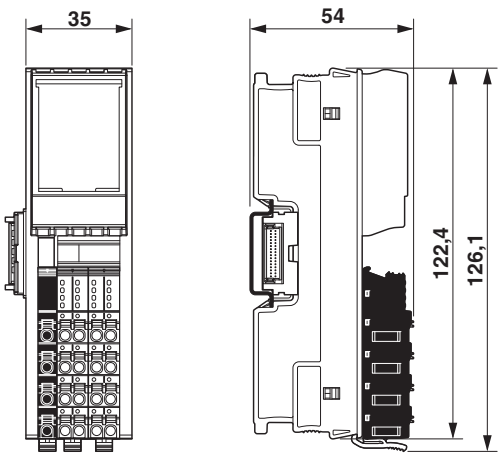
Description	Type	Order No.	Pcs./Pkt.
Axioline F, interface module, CAN, transparent protocol, max. transmission speed of 1 Mbps, IP20 protection, including bus base module and Axioline F connectors	AXL F IF CAN 1H	2702668	1

Accessories	Type	Order No.	Pcs./Pkt.
Axioline F bus base module for housing type H (Replacement item)	AXL F BS H	2700992	5
Zack marker strip for Axioline F (device labeling), in 2 x 20.3 mm pitch, unprinted, 25-section, for individual labeling with B-STIFT 0.8, X-PEN, or CMS-P1-PLOTTER (Marking)	ZB 20,3 AXL UNPRINTED	0829579	25
Zack Marker strip, flat, Strip, white, unlabeled, can be labeled with: PLOTMARK, CMS-P1-PLOTTER, mounting type: snap into flat marker groove, for terminal block width: 10.15 mm, lettering field size: 4 of 10.15 x 5 mm and 1 of 5.8 x 5 mm, Number of individual labels: 50 (Marking)	ZBF 10/5,8 AXL UNPRINTED	0829580	50
Axioline shield connection set (contains 2 shield bus holders and 2 SK 5 shield connection clamps)	AXL SHIELD SET	2700518	1
Insert label, Roll, white, unlabeled, can be labeled with: THERMOMARK ROLLMASTER 300/600, THERMOMARK X1.2, THERMOMARK ROLL X1, THERMOMARK ROLL 2.0, THERMOMARK ROLL, mounting type: snapped into marker carrier, lettering field size: 35 x 28 mm, Number of individual labels: 500 (Marking)	EMT (35X28)R	0801602	1

Documentation	Type	Order No.	Pcs./Pkt.
User manual, English, Axioline F: System and installation	UM EN AXL F SYS INST	-	-
User manual, English, Axioline F: Diagnostic registers, and error messages	UM EN AXL F SYS DIAG	-	-

4 Technical data

Dimensions (nominal sizes in mm)	



Width	35 mm
Height	126.1 mm
Depth	54 mm
Note on dimensions	The depth is valid when a TH 35-7,5 DIN rail is used (according to EN 60715).

General data	
Color	traffic grey A RAL 7042
Weight	190 g (with connectors and bus base module)
Mounting type	DIN rail
Ambient temperature (operation)	-25 °C ... 60 °C
Ambient temperature (storage/transport)	-40 °C ... 85 °C
Permissible humidity (operation)	5 % ... 95 % (non-condensing)
Permissible humidity (storage/transport)	5 % ... 95 % (non-condensing)
Air pressure (operation)	70 kPa ... 106 kPa (up to 3000 m above sea level)
Air pressure (storage/transport)	70 kPa ... 106 kPa (up to 3000 m above sea level)
Degree of protection	IP20
Protection class	III (IEC 61140, EN 61140, VDE 0140-1)
Overvoltage category	II (IEC 60664-1, EN 60664-1)
Degree of pollution	2 (IEC 60664-1, EN 60664-1)
Mounting position	any (no temperature derating)

Connection data: Axioline F connector	
Connection method	Push-in connection
Conductor cross section, rigid	0.2 mm² ... 1.5 mm²
Conductor cross section, flexible	0.2 mm² ... 1.5 mm²
Conductor cross section [AWG]	24 ... 16
Stripping length	8 mm

Please observe the information provided on conductor cross sections in the “Axioline F: system and installation” user manual.

Interface: Axioline F local bus	
Number	2
Connection method	Bus base module
Transmission speed	100 Mbps

Interface: CAN bus	
No. of channels	1
Connection method	Push-in connection
Transmission speed	10 kbps ... 1 Mbps (Default: 20 kbps)
Transmission physics	CAN bus according to standard ISO 11898-2 (high-speed CAN)

Axioline F local bus supply (U_{Bus})	
Supply voltage	5 V DC (via bus base module)
Current consumption	max. 150 mA
Power consumption	max. 0.75 W

Feed-in of supply voltage (U_l)	
Supply voltage	24 V DC
Supply voltage range	19.2 V DC ... 30 V DC including all tolerances, including ripple
Current consumption	max. 25 mA
Power consumption	max. 0.6 W

Input and output address area	
Input address area	64 Byte
Output address area	64 Byte

Configuration and parameter data in a PROFIBUS system	
Required parameter data	1 Byte
Required configuration data	7 Byte

Electrical isolation/isolation of the voltage areas	
Test section	**Test voltage**
5 V supply of the local bus (U_{Bus}) / 24 V supply (I/Os)	500 V AC, 50 Hz, 1 min.
5 V supply of the local bus (U_{Bus}) / CAN I/Os	500 V AC, 50 Hz, 1 min.
24 V supply (I/O) / CAN I/O	500 V AC, 50 Hz, 1 min.
5 V supply of the local bus (U_{Bus}) / functional ground	500 V AC, 50 Hz, 1 min.
24 V supply (I/O) / functional ground	500 V AC, 50 Hz, 1 min.

Mechanical tests	
Vibration resistance in acc. with EN 60068-2-6/ IEC 60068-2-6	5g
Shock in acc. with EN 60068-2-27/IEC 60068-2-27	30g
Continuous shock according to EN 60068-2-27/ IEC 60068-2-27	10g

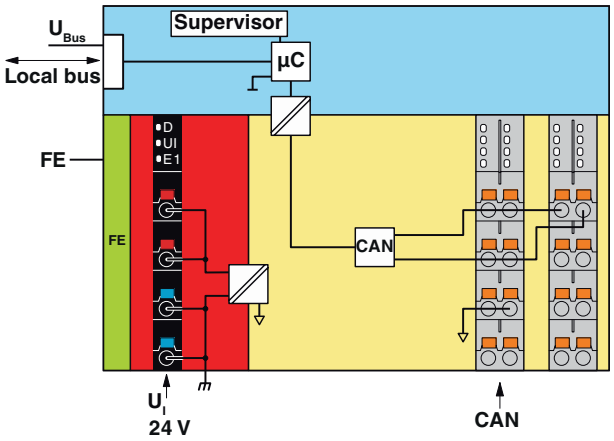
Conformance with EMC Directive 2014/30/EU	
Noise immunity test in accordance with EN 61000-6-2	
Electrostatic discharge (ESD) EN 61000-4-2/ IEC 61000-4-2	Criterion B, 6 kV contact discharge, 8 kV air discharge
Electromagnetic fields EN 61000-4-3/IEC 61000-4-3	Criterion A, Field intensity: 10 V/m
Fast transients (burst) EN 61000-4-4/IEC 61000-4-4	Criterion B, 2 kV
Transient overvoltage (surge) EN 61000-4-5/ IEC 61000-4-5	Criterion B, DC supply lines: ±0.5 kV/±0.5 kV (symmetrical/ asymmetrical)
Conducted interference EN 61000-4-6/IEC 61000-4-6	Criterion A, Test voltage 10 V
Noise emission test according to EN 61000-6-3	Class B

Approvals

For the latest approvals, please visit phoenixcontact.net/products.

5 Internal circuit diagram

Bild 1 Internal wiring of the terminal points



Key:

Local bus	Axioline F local bus (hereinafter referred to as local bus)
FE	Functional ground
µC	Microcontroller
	Electrical isolation for data or power supply
CAN	CAN bus logic
Supervisor	Hardware monitoring
	Logic reference ground
	Noiseless ground
	Reference ground of supply voltage U_I
	Electrically isolated areas

6 For your safety

6.1 Intended use

Use the Axioline F modules exclusively in accordance with the specifications in the accompanying data sheet and the “Axioline F: System and Installation” user manual.

6.2 Qualification of users

The use of products described in this data sheet is oriented exclusively to electrically skilled persons or persons instructed by them. The users must be familiar with the relevant safety concepts of automation technology as well as applicable standards and other regulations.

6.3 Electrical safety

WARNING: loss of electrical safety

If used incorrectly, device safety may be impaired.

The instructions given in this data sheet as well as the UM EN AXL F SYS INST user manual must be observed during installation, startup, and operation.

6.4 Installation

Only install the Axioline F modules in a control cabinet or junction box.

The enclosure must meet the requirements regarding the protection against spread of fire according to the following standards:

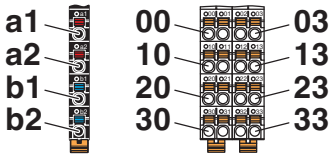
– EN 61010-1/IEC 61010-1
– UL 61010-1 (for applications with UL approval)

7 UL note

– Use copper conductor only.

8 Terminal point assignment

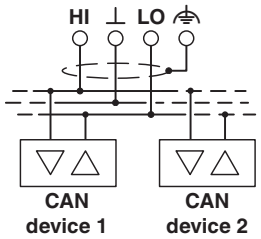
Bild 2 Terminal point assignment



Terminal point	Color	Assignment	
Supply voltage input			
a1, a2	Red	24 V DC (U_l)	Supply voltage feed-in (bridged internally)
b1, b2	Blue	GND	Reference potential of the supply voltage (bridged internally)
CAN			
00, 01	Orange	CAN High	Bridged internally
10, 11	Orange	CAN Low	Bridged internally
20, 21	Orange	CAN_GND	Bridged internally
30, 31	Orange	CAN_SHLD	Bridged internally
Pluggable termination resistor			
02	Orange	R+	Connection of termination resistor
03	Orange	R-	Connection of termination resistor
12, 13	Orange	Not used	
22, 23	Orange	Not used	
32, 33	Orange	Not used	

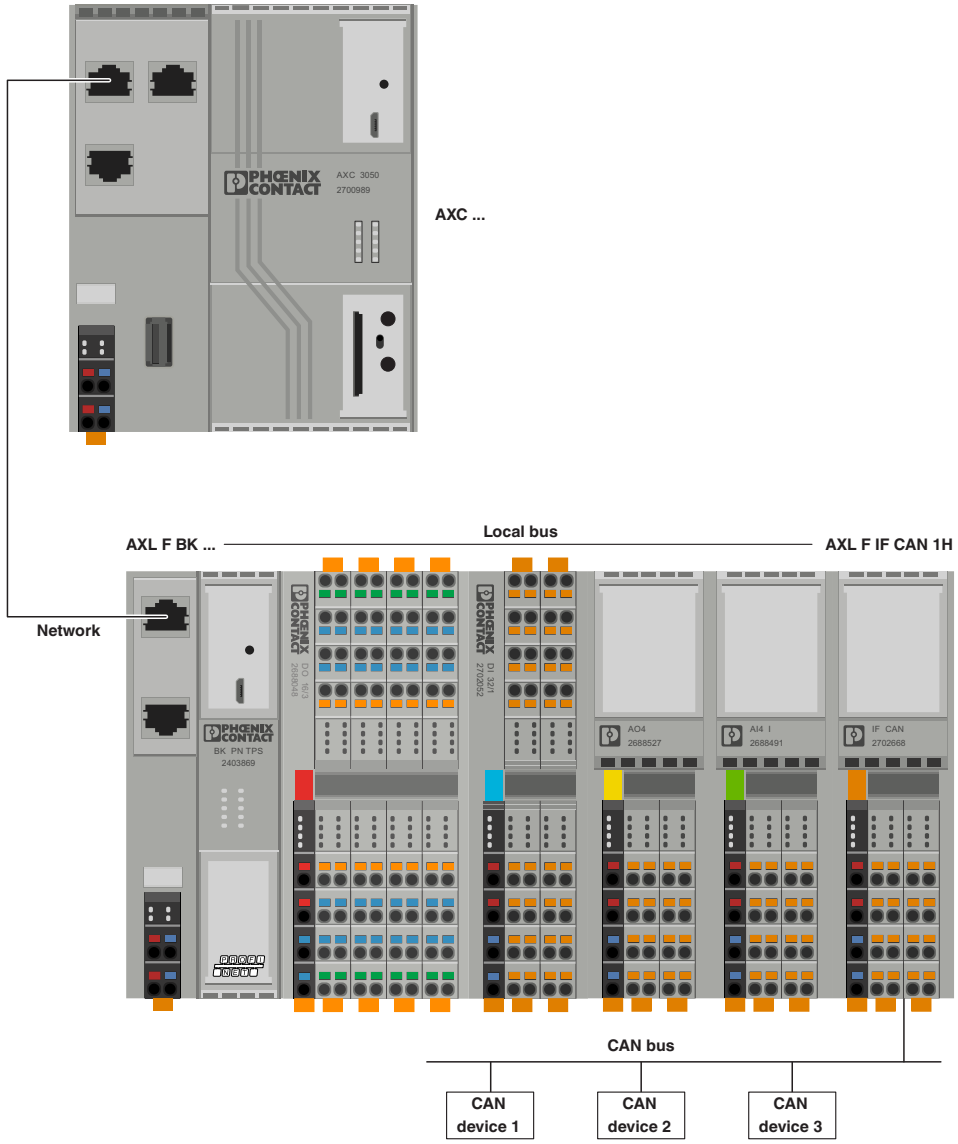
9 Connection example

Bild 3 Connection example



10 Application example

Bild 4 Example application



AXC ...	Controller
AXL F BK ...	Axioline F bus coupler
Local bus	Axioline F local bus with I/O modules according to your application
CAN bus	Lower-level CAN bus

11 Restrictions and special features

11.1 Use of the Axioline F CAN interface module

You can operate the Axioline F CAN interface module in Axioline F stations with the following station heads:

Head of the Axioline F station	Type	Note
Controller	AXC ...	
Bus coupler	AXL F BK PN (XC)	Restrictions with regard to startup parameterization
	AXL F BK PN TPS (XC)	

AXL F BK PN ... bus couplers can be used under any Phoenix Contact controller that supports PROFINET.

For PC Worx and PLCnext Engineer, the CANbus_Vx... function block library is available for the Axioline F CAN interface module. This library contains function blocks for communication and parameterization.

The AXL_CAN_COMM function block is used for communication between the AXL F IF CAN 1H and the higher-level controller.
This function block handles the sequential transfer of CAN_IN_Boxes and CAN_OUT_Boxes between the Axioline F CAN interface module and the higher-level controller.

AXL_CAN_Para... function blocks are used to parameterize the AXL F IF CAN 1H.

11.2 Startup parameterization when using a bus coupler

11.2.1 Use on the AXL F BK PN

You can parameterize the module via the startup parameterization. The only exception is the “Restrict reading of CAN messages to those with specific identifiers” function (filter).

In the device catalog in PC Worx or PLCnext Engineer, select the “AXL F IF CAN 1H - AXL F BK PN” entry.

The filters can be set with Startup+, in PC Worx, or in PLCnext Engineer. In PC Worx and PLCnext Engineer, you can use the AXL_CAN_Para function block or the FDT interface for this. The filters are then stored retentively in the module.

11.2.2 Use on another PROFINET bus coupler

You can parameterize the module completely via the startup parameterization.

In the device catalog in PC Worx or PLCnext Engineer, select the “AXL F IF CAN 1H” entry.

12 Connection notes

CAN bus

Observe the DR303-1 CANopen® specification when installing the CAN bus.

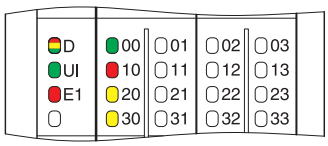
Termination resistor

Observe the following when connecting the termination resistor:

- Use a resistance of 120 Ω.
- Insulate the connecting wires of the resistor.
- If the cable is not sufficiently stable to open the Push-in connection, open the terminal point spring by pressing on the spring lever with a screwdriver.

13 Local diagnostic and status indicators

Bild 5 Local diagnostic and status indicators



Designation	Color	Meaning	State	Description
D	Red/yellow/green	Diagnostics of local bus communication		
		Run	Green on	The device is ready for operation, communication within the station is OK. All data is valid. An error has not occurred.
		Active	Green flashing	The device is ready to operate, communication within the station is OK. The data is **not** valid. The controller or superordinate network is not delivering valid data. There is no error on the module.
		Device application not active	Green/yellow flashing	The device is ready for operation, communication within the station is OK. Output data **cannot** be outputted and/or input data **cannot** be read. There is a fault on the periphery side of the module..
		Ready	Yellow on	The device is ready for operation but did not detect a valid cycle after power-up.
		Connected	Yellow flashing	The device is not (yet) part of the active configuration.
		Reset	Red on	The device is ready for operation but has lost the connection to the bus head.
		Not connected	Red flashing	The device is ready for operation but there is no connection to the previously existing device.
		Power down	Off	Device is in (power) reset.
UI	Green	U_I	On	I/O supply is present.
			Off	I/O supply is not present.
E1	Red	Device error	On	I/O supply is faulty.
				Parameter memory is faulty.
			Off	No error
00	Green	Protocol transmission status	On	Normal operation
			Flashing	200 ms on, 1000 ms off: protocol transmission fault.
			Off	Access to the parameter memory (flash mode).
10	Red	Error	Off	No error
			Flashing	200 ms on, 200 ms off: invalid configuration
				200 ms on, 1000 ms off: warning limit reached
			On	CAN controller is not connected to the bus (bus off).

Designation	Color	Meaning	State	Description
20	Yellow	Transmit error status (Tx)	Off	No transmission and no transmission errors detected on the CAN bus.
			Flashing	200 ms on, 200 ms off: transmission of CAN messages
			On	TX error counter was > 12 and is still > 0.
30	Yellow	Receive error status (Rx)	Off	No CAN messages received and no receive errors detected on the CAN bus.
			Flashing	200 ms on, 200 ms off: receipt of CAN messages
			On	RX error counter was > 12 and is still > 0.

14 Principle function

You can use this module to exchange messages between an Axioline F station and a CAN bus system. The messages are transmitted transparently from CAN to the fieldbus and from there to the higher-level controller, and in the opposite direction too.
Transparent transmission allows direct access to Layer 2 of the CAN bus (according to the ISO/OSI reference model). The higher layers of the CAN protocol must be processed by the controller.

In addition, the module provides buffer memory (message buffers) and a filter function.
The buffer function ensures that no messages are lost due to asynchrony when transporting messages from the CAN bus to the controller and reactions to them.
To this end, the integrated buffer memory in the module is able to buffer CAN messages:

– Up to 256 CAN messages in the receive direction
– Up to 64 CAN messages in the transmit direction

On the connected CAN, more identifiers may be transferred than the PLC application requires.
Using a filter function, you can set parameters to ensure that only CAN messages with the desired identifiers are read, forwarded to the controller, and processed by the controller.
You can define 60 filters when using 11-bit identifiers.
You can define 30 filters when using 29-bit identifiers.
You can parameterize the filter areas using PDI objects that the controller accesses with asynchronous services.

The individual CAN messages are transmitted via the process data channel. The handshake mechanism controls transmission between the module and PLC. This ensures that no messages are lost or transmitted multiple times when message packets are forwarded as process data. The handshake mechanism is implemented by bits and counters in the 6 bytes of the CAN_IN_HEADER and the 4 bytes of the CAN_OUT_HEADER.

To simplify the transfer of messages between a controller and the AXL F IF CAN 1H, and the handling of the handshake mechanism, function blocks are provided for Phoenix Contact controllers.

Protocol functions in the process data channel:

– Initializing the CAN controller
– Starting and stopping the CAN controller
– Transmitting a CAN message (with handshake mechanism)
– Reporting the status of the CAN controller (e.g., Bus Off)

The function blocks for PC Worx and PLCnext Engineer also support these functions.

Message transfer between controller and AXL F IF CAN using handshake mechanism

(function of the bits and counters in the CAN_IN_HEADER)

The description below is further illustrated by the associated figure. Therefore please refer to the figure when reading the description.

CAN messages are transmitted transparently between a higher-level controller and the CAN bus.

Looking at this in detail, message transfer in the AXL F IF CAN 1H module is split into two transmission areas:

– Between the CAN bus and the buffer memory, the module transmits the CAN messages automatically according to the FIFO principle.
– The controller is responsible for transmitting the CAN messages between the buffer memory and the higher-level controller (via the higher-level bus).

Between the buffer memory and the higher-level controller, the messages are transported by means of process data. To ensure loss-free and error-free transport, a handshake mechanism is required. This handshake mechanism defines that the 64-byte process data in each transmission direction consists of a header (the first 4 to 6 bytes) and a CAN box (the following 58 to 60 bytes).
One or more CAN messages are transmitted in a CAN box. The number of CAN messages depends on the demand or data volume and the maximum capacity of the CAN box. The header contains the command and confirmation bits as well as the signaling and acknowledgment counters. The handshake mechanism uses these bits and counters to ensure the transfer of a CAN box between the controller and CAN interface module (and back).

In the **receive direction**, the Axioline F CAN interface module reads incoming CAN messages from the connected CAN bus and places them in the receive buffer memory. The module then packs as many CAN messages as possible from the buffer (as separate CAN_IN_MESSAGE_PARCELs) without gaps into the CAN_IN_BOX and deletes the messages from the buffer.
The number of CAN_IN_MESSAGE_PARCELs depends on the number of CAN messages in the buffer, their length, and the size of the CAN_IN_BOX.
The module then makes this CAN_IN_BOX available to the controller so it can be read and signals this in the header by incrementing the value in CAN_IN_BOX_ID. Along with this, the module displays the number of CAN_IN_MESSAGE_PARCELs in the header (in CAN_IN_PARCEL_CNT).
The controller can now read the CAN_IN_BOX, and therefore the CAN messages packed inside it. The module keeps the contents of the CAN_IN_BOX stable until the controller indicates that they have been successfully read. The controller signals that it has finished reading by copying the value from CAN_IN_BOX_ID to CAN_IN_BOX_ACK.

The transfer of CAN messages between the buffer memory of the module and the higher-level controller is controlled according to the handshake mechanism using counters in the header. In the CAN_IN_BOX, between one and 19 CAN messages (CAN_IN_MESSAGE_PARCELs) can be transmitted simultaneously. The maximum possible number depends on the length of the individual CAN messages.

In the **transmit direction**, the controller writes the CAN messages which are tightly packed (as individual CAN_OUT_MESSAGE_PARCELs) to the CAN_OUT_BOX. The number of CAN_OUT_MESSAGE_PARCELs depends on the number of CAN messages in the controller waiting to be sent, their length, and the size of the CAN_OUT_BOX. When the CAN_OUT_BOX is completely written, the controller signals this to the module in the header by incrementing the value in CAN_OUT_BOX_ID. Along with this, the controller displays the number of CAN_OUT_MESSAGE_PARCELs in the header (in CAN_OUT_PARCEL_CNT).

The controller must keep the contents of the CAN_OUT_BOX stable until the module indicates that it has successfully transmitted the CAN messages to the transmit buffer memory. The module signals that it has finished transmitting by copying the value from CAN_OUT_BOX_ID to CAN_OUT_BOX_ACK.
The Axioline F CAN interface module retrieves the CAN messages to be sent from the buffer memory, writes them to the connected CAN bus, and deletes them from the buffer.
The transfer of CAN messages between the higher-level controller and the buffer memory of the module is controlled according to the handshake mechanism using counters in the header. In the CAN_OUT_BOX, between one and 20 CAN messages (CAN_OUT_MESSAGE_PARCELs) can be transmitted simultaneously. The maximum possible number depends on the length of the individual CAN messages.

In addition, the CAN interface module signals how full the buffer memories currently are as a percentage (IN_BUF_State and OUT_BUF_State) and how many CAN messages the buffer memories contain (CAN_IN_PARCEL_BUFF_CNT and CAN_OUT_PARCEL_BUFF_CNT).

A function block (AXL_CAN_COMM) is available for Phoenix Contact controllers. It handles the transfer of CAN_IN_BOX and CAN_OUT_BOX between the CAN interface module and the higher-level controller, see “Function block” section.
The actual CAN protocol for the higher protocol layers must be processed in the application program of the controller.

Enabling automatic message transfer between the buffer memory and CAN bus

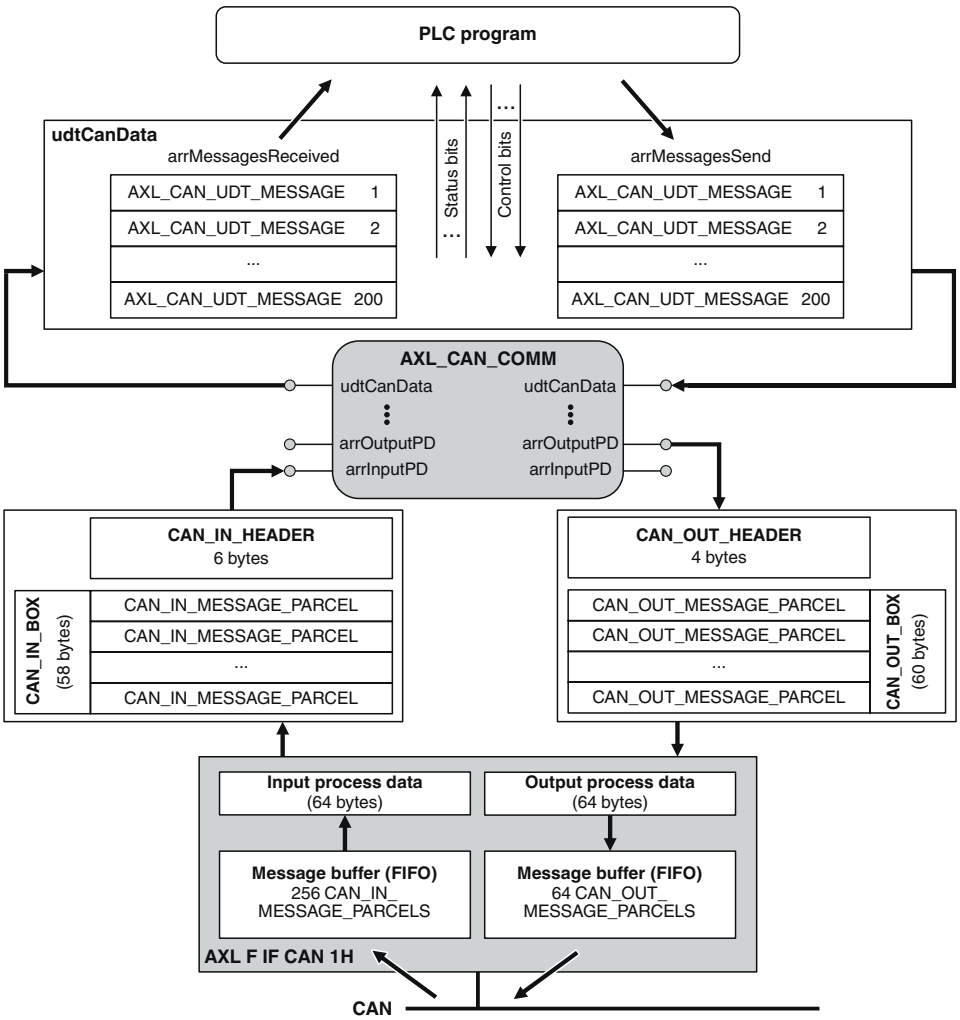
The module automatically transfers messages between the buffer memories and CAN bus. It fills the receive buffer and empties the transmit buffer according to the FIFO principle. The higher-level controller can control the function of the buffer memories and how they exchange data with CAN. To ensure that communication is started unambiguously, stop communication and empty the buffer memory. You can then definitively start communication.

The CAN transmit buffer can be emptied using the Clear_Tx-Buf control bit. While it is active, no more CAN messages are sent. New CAN messages are not entered in the transmit buffer, instead they are discarded.

The CAN receive buffer can be emptied using the Clear_Rx-Buf control bit. While it is active, no more CAN messages are received. The module does not enter any new CAN messages in the receive buffer.

You can use the PI_Ex_Stop control bit to stop data exchange between the buffer memory and Axioline F.
You can stop automatic message transfer between the buffer memory and CAN bus separately using the CAN_Stop control bit. This method of stopping is similar to interrupting the connection to the CAN bus cable.
The module confirms the receipt of a signal change at the PI_Ex_Stop and CAN_Stop control bits with the PI_Ex_Stop and CAN_Stop bits in the input process data.

Bild 6 Overview of data flow between higher-level controller and the lower-level CAN bus



PLC program	Program in the higher-level controller
udtCanData	Data structure for exchanging CAN messages between the PLC program and the function block
AXL_CAN_COMM	Function block for handling the Axioline F CAN interface module
CAN_IN_HEADER	Header for the IN process data
CAN_IN_BOX	Packed CAN messages that have been received.
CAN_OUT_HEADER	Header for the OUT process data
CAN_OUT_BOX	Packed CAN messages that are to be sent.
AXL F IF CAN 1H	Axioline F CAN interface module
CAN	Lower-level CAN bus

15 Process data

15.1 OUT process data

The output process data is shown in the tables below. The controller sends this to the module. The output process data contains the header and the packed CAN messages.
The handshake mechanism and transmission to CAN are controlled with the header. The CAN_OUT_HEADER occupies byte 0 to byte 3.
The bytes of the CAN_OUT_BOX, which can contain one to 60 CAN messages (CAN_OUT_MESSAGE_PARCELs), follow the header.

Structure of the output process data

CAN_OUT_HEADER	4 bytes
CAN_OUT_MESSAGE_PARCEL	CAN_OUT_BOX Byte 0 ... Byte 59
CAN_OUT_MESSAGE_PARCEL	
...	
CAN_OUT_MESSAGE_PARCEL	

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	CAN_OUT_HEADER byte 0							
...	...							
Byte	3							
Contents	CAN_OUT_HEADER byte 3							
Byte	4							
Contents	CAN_OUT_BOX Byte 0							
...	...							
Byte	63							
Contents	CAN_OUT_BOX Byte 59							

Designation	Meaning
CAN_OUT_HEADER Byte 0 ... 3	Header for the OUT process data
CAN_OUT_BOX Byte 0 ... Byte 59	Packed CAN messages that are to be sent.

15.1.1 Header for the OUT process data (CAN_OUT_HEADER)

The output process data header is shown in the table below. It occupies byte 0 to byte 3.

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	0	0	Clear_Rx-Buf	Clear_Tx-Buf	0	0	CAN_Stop	PI_Ex_Stop
Byte	1							
Contents	0	0	0	0	0	0	0	0
Byte	2							
Contents	CAN_OUT_PARCEL_CNT							
Byte	3							
Contents	CAN_IN_BOX_ACK				CAN_OUT_BOX_ID			

Designation	Meaning	
PI_Ex_Stop	0	Send and receive CAN messages.
	1	Stop data exchange between the CAN buffer memory and Axioline F. Discard received CAN messages.
CAN_Stop	0	Start CAN operation, normal operation.
		Automatic transfer between buffer memory and CAN bus.
	1	Stop CAN operation.
Clear_Rx-Buf	0	Normal operation
	1	Empty CAN receive buffer. Discard new received CAN messages. Do not enter CAN messages in the input buffer.
Clear_Tx-Buf	0	Normal operation
	1	Empty CAN transmit buffer. Discard CAN messages still to be sent. Do not enter CAN messages in the transmit buffer.
CAN_OUT_PARCEL_CNT	Number of CAN messages in the CAN_OUT_BOX.	
CAN_OUT_BOX_ID	ID code of the CAN messages in the CAN_OUT_BOX.	
	The code is part of the handshake mechanism between the higher-level controller and the Axioline F CAN interface module.	
	The code indicates whether the data in the CAN_OUT_BOX is new data or whether old data has been transmitted again. If the code remains unchanged compared to the previous data packet, the data has been resent. If the code differs in any way, this means that the data is new data. The sender increments this code for new data.	
	If the value is equal to 0, no data is present. Observe the following in the event of an overrun of F_{hex} to 0 or 1: if new valid data is to be displayed, the value 0 must be skipped.	
CAN_IN_BOX_ACK	Confirmation code (acknowledgment, mirroring) of the higher-level controller.	
	The code indicates that the controller has successfully read the CAN messages with this identification code from the Axioline F CAN interface module.	
	The code is part of the handshake mechanism between the higher-level controller and the Axioline F CAN interface module.	
	When the CAN_IN_BOX_ACK acknowledgment code matches the CAN_IN_BOX_ID ID code, the Axioline F CAN interface module can send new CAN messages to the higher-level controller.	

15.1.2 CAN_OUT_BOX

The bytes of the CAN_OUT_BOX follow the header. They occupy byte 4 to byte 63, maximum in the output process data. The higher-level controller transmits the CAN data packets to be sent (CAN_OUT_MESSAGE_PARCELs) to the Axioline F CAN interface module in the max. 60-byte CAN_OUT_BOX.
The CAN_OUT_PARCEL_CNT byte in the header displays the number of CAN messages in the CAN_OUT_BOX.
The data is tightly packed in order to transmit as many CAN messages as possible.

CAN messages with 11-bit CAN identifier use two bytes fewer than CAN messages with 29-bit CAN identifier.

Structure of a data packet (CAN_OUT_MESSAGE_PARCEL) when using an 11-bit CAN identifier

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	NUM_BYTES				0	0	0	Ext = 0
Byte	1							
Contents	CAN_ID11_BITS_7_0							
Byte	2							
Contents	RTR	0	0	0	0	CAN_ID11_BITS_10_8		
Byte	3							
Contents	CAN_BYTE_0							
...	...							
Byte	3 + n							
Contents	CAN_BYTE_n							
...	...							
Byte	10							
Contents	CAN_BYTE_7 (max.)							

Structure of a data packet (CAN_OUT_MESSAGE_PARCEL) when using a 29-bit CAN identifier

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	NUM_BYTES				0	0	0	Ext = 1
Byte	1							
Contents	CAN_ID29_BITS_7_0							
Byte	2							
Contents	CAN_ID29_BITS_15_8							
Byte	3							
Contents	CAN_ID29_BITS_23_16							
Byte	4							
Contents	RTR	0	0	CAN_ID29_BITS_28_24				
Byte	5							
Contents	CAN_BYTE_0							
...	...							
Byte	5 + n							
Contents	CAN_BYTE_n							
...	...							
Byte	12							
Contents	CAN_BYTE_7 (max.)							

Designation	Meaning	
Ext	0	11-bit CAN identifier
	1	29-bit CAN identifier
NUM_BYTES	Number of bytes that follow	
	The value in NUM_BYTES is based on the sum of the bytes for the identifier + 1 and the number of bytes for the CAN message.	
CAN_ID11_BITS_7_0	Bits 0 ... 7 of the CAN identifier	
CAN_ID11_BITS_10_8	Bits 8 ... 10 of the CAN identifier	
CAN_ID29_BITS_7_0	Bits 0 ... 7 of the CAN identifier	
CAN_ID29_BITS_15_8	Bits 8 ... 15 of the CAN identifier	
CAN_ID29_BITS_23_16	Bits 16 ... 23 of the CAN identifier	
CAN_ID29_BITS_28_24	Bits 24 ... 28 of the CAN identifier	
RTR	Remote request frame	
	0	CAN data frame
	1	CAN remote frame
CAN_Byte_0 ... CAN_BYTE_n (n ≤ 7)	Contents (data) of the CAN message	

15.2 IN process data

The input process data is shown in the tables below. The controller retrieves this from the module. The input process data contains the header and the packed CAN messages.
The handshake mechanism and transmission from CAN are controlled with the header. The CAN_IN_HEADER occupies byte 0 to byte 5.
The bytes of the CAN_IN_BOX, which can contain one to 60 CAN messages (CAN_IN_MESSAGE_PARCELs), follow the header.

Structure of the input process data

CAN_IN_HEADER	6 bytes
CAN_IN_MESSAGE_PARCEL	CAN_IN_BOX byte 0 ... byte 57
CAN_IN_MESSAGE_PARCEL	
...	
CAN_IN_MESSAGE_PARCEL	

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	CAN_IN_HEADER byte 0							
...	...							
Byte	5							
Contents	CAN_IN_HEADER byte 5							
Byte	6							
Contents	CAN_IN_BOX byte 0							
...	...							
Byte	63							
Contents	CAN_IN_BOX byte 57							

Designation	Meaning
CAN_IN_HEADER byte 0 ... byte 5	Header for the IN process data
CAN_IN_BOX byte 0 ... byte 57	Packed CAN messages that have been received.

15.2.1 Header for the IN process data (CAN_IN_HEADER)

The input process data contains the header with the counters for the handshake mechanism and the status bits.

The input process data header is shown in the table below. It occupies byte 0 to byte 5.

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	0	0	0	0	0	0	CAN_Stop	PI_Ex_Stop
Byte	1							
Contents	IN_BUF_State		OUT_BUF_State		CC_Bus_off	CC_Warn	CC_OVR	CAIN_OVR
Byte	2							
Contents	CAN_IN_PARCEL_CNT							
Byte	3							
Contents	CAN_IN_BOX_ID				CAN_OUT_BOX_ACK			
Byte	4							
Contents	CAN_IN_PARCEL_BUFF_CNT							
Byte	5							
Contents	CAN_OUT_PARCEL_BUFF_CNT							

Designation	Meaning	
PI_Ex_Stop	0	CAN messages are sent and received
	1	Data exchange between CAN and Axioline F has been stopped. Received CAN messages in the module buffer have been discarded.
CAN_Stop	0	Normal operation
	1	CAN controller has been stopped.
CAIN_OVR	0	No CAIN overrun. CAN messages are received and forwarded via the receive buffer.
	1	CAIN Overrun
		Overrun of the receive buffer. Messages that were received after the overrun have been discarded.
CC_OVR	0	No CAN controller overrun
	1	CAN controller overrun
		A receive CAN message in the CAN controller has been lost.
CC_Warn	0	The CAN controller is not in the error warning state.
	1	The CAN controller is in the error warning state.
CC_Bus_off	0	The CAN controller is not in the bus off state.
	1	The CAN controller is in the bus off state.
CC_Warn, CC_Bus_off		See section “CC_Warn and CC_Bus_off”.
OUT_BUF_State	Fill level indicator for the transmit buffer	
	00	Empty
	01	Not empty ... < 33% full
	10	33% ... < 67% full
	11	67% ... 100% full
IN_BUF_State	Fill level indicator for the receive buffer	
	00	Empty
	01	Not empty ... < 33% full
	10	33% ... < 67% full
	11	67% ... 100% full

Designation	Meaning
CAN_IN_PARCEL_CNT	Number of CAN messages transmitted within the current 64 bytes of process data.
CAN_IN_BOX_ID	ID code of the CAN messages in the CAN_IN_BOX.
	The code is part of the handshake mechanism between the higher-level controller and the Axioline F CAN interface module.
	The code indicates whether the data in the CAN_IN_BOX is new data or whether old data has been transmitted again. If the code remains unchanged compared to the previous data packet, the data has been resent. If the code differs in any way, this means that the data is new data. The sender increments this code for new data.
	If the value is equal to 0, no data is present. Observe the following in the event of an overrun of F_{hex} to 0 or 1: if new valid data is to be displayed, the Axioline F CAN interface module skips the value 0.
CAN_OUT_BOX_ACK	Acknowledgment code (acknowledge, mirroring) from the Axioline F CAN interface module that the CAN messages with this ID code have been received by higher-level controller.
	The code indicates that the controller has successfully received the CAN messages with this identification code.
	The code is part of the handshake mechanism between the higher-level controller and the Axioline F CAN interface module.
	When the CAN_OUT_BOX_ACK acknowledgment code matches the CAN_OUT_BOX_ID ID code, the higher-level controller can send new CAN messages to the Axioline F CAN interface module.
CAN_IN_PARCEL_BUFF_CNT	Number of CAN messages currently contained in the receive buffer inside the Axioline F CAN interface module.
CAN_OUT_PARCEL_BUFF_CNT	Number of CAN messages currently contained in the transmit buffer inside the Axioline F CAN interface module.

15.2.2 CC_Warn and CC_Bus_off

The bits CC_Warn and CC_Bus_off are delivered directly from the CAN chip in this module.

The CAN chip has an error counter. This error counter is incremented if an error is detected when sending or receiving CAN messages, i.e., if problems arose on the CAN bus. If the module later receives or sends messages again without error, the counter drops back again.

When the counter reaches an initial threshold value, only the CC_Warn bit is set.
If the counter then falls back below the threshold value, because messages have been sent or received without error in the meantime, the CC_Warn bit is automatically reset.
If the error counter reaches the second threshold value, the CAN chip switches to bus off mode and the module stops communication with the CAN bus. This is indicated by the CC_Bus_off bit in the input process data.

To restart the module, set the bit CAN_Stop to 1 in the process output data and then back to 0. This causes the application to acknowledge the bus-off mode and start the module.

15.2.3 CAN_IN_BOX

The bytes of the CAN_IN_BOX follow the header. They occupy byte 6 to byte 63, maximum in the input process data.
The Axioline F CAN interface module transmits the received CAN data packets (CAN_IN_MESSAGE_PARCELs) to the higher-level controller in the max. 58-byte CAN_IN_BOX.
The CAN_IN_PARCEL_CNT byte in the header displays the number of CAN messages in the CAN_IN_BOX.
The data is tightly packed in order to transmit as many CAN messages as possible.

CAN messages with 11-bit CAN identifier use two bytes fewer than CAN messages with 29-bit CAN identifier.

Structure of a data packet (CAN_IN_MESSAGE_PARCEL) when using an 11-bit CAN identifier

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	NUM_BYTES				0	0	0	Ext = 0
Byte	1							
Contents	CAN_ID11_BITS_7_0							
Byte	2							
Contents	RTR	0	0	0	0	CAN_ID11_BITS_10_8		
Byte	3							
Contents	CAN_BYTE_0							
...	...							
Byte	3 + n							
Contents	CAN_BYTE_n							
...	...							
Byte	10							
Contents	CAN_BYTE_7 (max.)							

Structure of a data packet (CAN_IN_MESSAGE_PARCEL) when using a 29-bit CAN identifier

Bit	7	6	5	4	3	2	1	0
Byte	0							
Contents	NUM_BYTES				0	0	0	Ext = 1
Byte	1							
Contents	CAN_ID29_BITS_7_0							
Byte	2							
Contents	CAN_ID29_BITS_15_8							
Byte	3							
Contents	CAN_ID29_BITS_23_16							
Byte	4							
Contents	RTR	0	0	CAN_ID29_BITS_28_24				
Byte	5							
Contents	CAN_BYTE_0							
...	...							
Byte	5 + n							
Contents	CAN_BYTE_n							
...	...							
Byte	12							
Contents	CAN_BYTE_7 (max.)							

Designation	Meaning	
Ext	0	11-bit CAN identifier
	1	29-bit CAN identifier
NUM_BYTES	Number of bytes that follow	
	The value in NUM_BYTES is based on the sum of the bytes for the identifier + 1 and the number of bytes for the CAN message.	
CAN_ID11_BITS_7_0	Bits 0 ... 7 of the CAN identifier	
CAN_ID11_BITS_10_8	Bits 8 ... 10 of the CAN identifier	
CAN_ID29_BITS_7_0	Bits 0 ... 7 of the CAN identifier	
CAN_ID29_BITS_15_8	Bits 8 ... 15 of the CAN identifier	
CAN_ID29_BITS_23_16	Bits 16 ... 23 of the CAN identifier	
CAN_ID29_BITS_28_24	Bits 24 ... 28 of the CAN identifier	
RTR	Remote request frame	
	0	CAN data frame
	1	CAN remote frame
CAN_Byte_0 ... CAN_BYTE_n (n ≤ 7)	Contents (data) of the CAN message	

16 Parameter, diagnostics and information (PDI)

Parameter and diagnostic data as well as other information is transmitted as objects via the PDI channel of the Axioline F station.

The standard and application objects stored in the module are described in the following section.

The following applies to all tables below:

Please refer to the UM EN AXL F SYS INST for an explanation of the data types.

Abbreviation	Meaning
R	Read
W	Write

Each visible string is terminated with a null terminator (00_{hex}). The length of a visible-string-type element is therefore at least one byte larger than the number of user data items.
If the number of user data items plus null terminator is smaller than the specified length of the element, the visible string will be populated with a null character (00_{hex}).

For detailed information on PDI objects, please refer to the UM EN AXL F SYS INST user manual.

Address all objects using subindex 0. The module does not support object addressing using subindices > 0.

17 Standard objects

17.1 Objects for identification (device rating plate)

Index (hex)	Object name	Data type	A	L	Rights	Meaning	Contents
Manufacturer							
0001	VendorName	Visible String	1	32	R	Vendor name	Phoenix Contact
0002	VendorID	Visible String	1	7	R	Vendor ID	00A045
0003	VendorText	Visible String	1	58	R	Vendor text	Components and systems for industrial automation
0012	VendorURL	Visible String	1	58	R	Vendor URL	http://www.phoenixcontact.com
Module - general							
0004	DeviceFamily	Visible String	1	20	R	Device family	I/O function module
0006	ProductFamily	Visible String	1	32	R	Product family	AXL F
000E	CommProfile	Visible String	1	4	R	Communication profile	633
000F	DeviceProfile	Visible String	1	5	R	Device profile	0010
0011	ProfileVersion	Record of Visible Strings	2	11; 21	R	Profile version	2011-12-07; Basic - Profile V2.0
0017	Language	Record of Visible Strings	2	6; 8	R	Language	en-us; English
003A	VersionCount	Array of UINT16	4	4 * 2	R	Version counter	e.g., 0007 0001 0001 0001_{hex}
Module - special							
0005	Capabilities	Visible String	1	8	R	Capabilities	Energ_0
0007	ProductName	Visible String	1	16	R	Product name	AXL F IF CAN 1H
0008	SerialNo	Visible String	1	11	R	Serial number	e. g., 1234512345
0009	ProductText	Visible String	1	24	R	Product text	1 communication channel
000A	OrderNumber	Visible String	1	8	R	Order No.	2702668
000B	HardwareVersion	Record of Visible Strings	2	11; 3	R	Hardware version	e. g., 2011-02-04; 00
000C	FirmwareVersion	Record of Visible Strings	2	11; 5	R	Firmware version	e.g., 2017-12-31; 1.00
000D	PChVersion	Record of Visible Strings	2	11; 6	R	PDI version	2010-01-08; V1.00
0032	FieldBus_ID	Record	2	3	R	Fieldbus ID	00_{hex}; 00 02_{hex}
0037	DeviceType	Octet string	1	8	R	Device type	00 02 00 40 00 00 00 $B1_{hex}$
Use of the device							
0014	Location	Visible String	1	59	R/W	Location	Can be completed by the user.
0015	EquipmentIdent	Visible String	1	59	R/W	Equipment identifier	Can be completed by the user.
0016	ApplDeviceAddr	UINT16	1	2	R/W	Application-specific device address	Can be completed by the user.

17.2 Miscellaneous standard objects

Index (hex)	Object name	Data type	A	L	Rights	Meaning/contents	
Object descriptions							
0038	ObjDescrReq	Record	2	3	R/W	Object description request	
0039	ObjDescr	Record	16	Max. 58	R	Object description	
Diagnostics objects							
0018	DiagState	Record	6	58	R	Diagnostic state	*
0019	ResetDiag	UINT8	1	1	W	Acknowledge diagnostic messages	*
Objects for process data management							
0025	PDIN	Octet string	1	64	R	Input process data	*
0026	PDOUT	Octet string	1	64	R	Output process data	*
003B	PDIN_Descr	Array of Records	1	8; 2; 2	R	Description of the IN process data	
003C	PDOUT_Descr	Array of Records	1	8; 2; 2	R	Description of the output process data	
Objects for device management							
002D	ResetParam	UINT8	1	1	R/W	Reset parameterization	*
002E	CheckSum	UINT32	1	4	R	Checksum	*
0040	ListOfObjToRestore	Array of Records	2	8	R	List of all the startup parameters of the device	*

The objects identified with * in the last column are described in more detail in the following sections.

The description of the other objects is to be found in the user manual UM EN AXL F SYS INST.

The objects 0038_{hex}, 0039_{hex}, $003B_{hex}$ and $003C_{hex}$ are only applicable to tools.

17.3 Diagnostics state (0018_{hex}: DiagState)

This object is used for a structured message of an error.

0018_{hex}: Diagnostics state (read)					
Subindex	**Data type**	**Length in bytes**	**Meaning**	**Contents**	
0	Record	Max. 58	Diagnostic state	Complete diagnostics information	
1	UINT16	2	Error number	0 ... 65535_{dec}	
2	UINT8	1	Priority	00_{hex}	No error
				01_{hex}	Error
				02_{hex}	Warning
				81_{hex}	Error removed
				82_{hex}	Warning eliminated
3	UINT8	1	Channel/group/module	00_{hex}	No error (if subindex 5 = 00_{hex})
				FF_{hex}	Entire device
4	UINT16	2	Error code	See table below	
5	UINT8	1	More follows	00_{hex}	
6	Visible String	max. 51	Text	00_{hex}	Text with end ID 00_{hex}

The message with priority 81_{hex} or 82_{hex} is a one-off, internal message to the bus coupler. The bus coupler transfers this error message to the error mechanisms of the higher-level system.

After all errors have been eliminated, it is automatically reset.

Error and status of the local diagnostics and status indicators

Subindex	2	3	4	6								
Error	Priority	Channel/ group/ module	Error code	Text	Process data		LED					
					Err	ErrState						
	hex	hex	hex		bin	bin	D	UI	E1	RUN	ERR	TX, RX
No error	00	00	0000	Status OK	0	00	X	●	○	X	X	X
Supply voltage faulty (I/O supply)	01	FF	5160	Supply fail	1	10	☼	○	●	○	○	○
Parameter memory faulty	01	FF	6320	Invalid para	0	00	☼	X	●	○	○	○

Key:

○	Off	☼	Green/yellow flashing
●	On	X	The LED is not affected by this error.

17.4 Acknowledge diagnostic messages (0019_{hex}: ResetDiag)

You can delete the diagnostics memory and acknowledge the diagnostic messages with this object.

0019_{hex}: Acknowledge diagnostic messages (read, write)				
Subindex	Data type	Length in bytes	Contents	
			Code (hex)	Meaning
0	UINT8	1	00	Permit all diagnostic messages
			02	Delete and acknowledge all diagnostic messages that are still pending
			03	Delete and acknowledge all diagnostic messages and reset the error counter
			06	Delete and acknowledge all diagnostic messages and do not permit new diagnostic messages
			Other	Reserved

17.5 Input process data (0025_{hex}: PDIN)

You can read the IN process data of the module with this object.

The structure corresponds to the representation in the "Process data" section.

0025_{hex}: IN process data (read)			
Subindex	**Data type**	**Length in bytes**	**Meaning/contents**
0	Octet string	64	Input process data

17.6 Output process data (0026_{hex}: PDOUT)

You can read the OUT process data of the module with this object.

The structure corresponds to the representation in the "Process data" section.

0026_{hex}: OUT process data (read)			
Subindex	**Data type**	**Length in bytes**	**Meaning/contents**
0	Octet string	64	Output process data

17.6.1 Reset parameterization ($002D_{hex}$: ResetParam)

This object is used to reset the startup parameters of the module to the default settings. The startup parameters are listed in object 0040_{hex}.

To reset the parameters, value 01_{hex} must be transferred during write access. All other values are invalid and will be acknowledged with an error.

Then the default settings of the module are loaded and all the user-defined parameters are reset.

17.6.2 Checksum ($002E_{hex}$: CheckSum)

The data of the startup objects is verified with this checksum. The checksum only changes if an object relevant for startup has been changed. The checksum is therefore suitable for comparing the parameterization.

17.6.3 List of all the startup parameters of the device (0040_{hex}:ListOfObjToRestore)

The object contains a list of all the startup parameters of the device. This enables parameters to be uploaded from the device (e.g., to clone a station).

18 Application objects

To parameterize the module, describe the application objects via the PDI channel.

Index (hex)	Object name	Data type	A	L	Rights	Meaning/contents
0560	PI_Exch_Stop	UINT8	1	1	R/W	PI-Exchange-Stop PD-Command
0561	CAN_Stop	UINT8	1	1	R/W	CAN-Stop PD-Command
0562	CAN_Bit_Rate	UINT32	1	4	R/W	CAN bus bit rate
056A	11_Bit_Filter_Func	UINT8	1	1	R/W	11-bit filter function
056B	Filter11BitRanges	Record of UINT32	60	60 * 4	R/W	Filter masks for 11-bit CAN identifier
056C	29_Bit_Filter_Func	UINT8	1	1	R/W	29-bit filter function
056D	Filter29BitRanges	Record of UINT32	30	30 * 8	R/W	Filter masks for 29-bit CAN identifier

18.1 PI exchange stop PD command (0560_{hex}: PI_Exch_Stop)

Use this object to specify whether or not the PI_Ex_Stop control bit is active in the OUT process data.

0560_{hex}: PI exchange stop PD command (read/write)				
Subindex	Data type	Length in bytes	Contents	
			Code	Meaning
0	UINT8	1	0	Flag inactive: PI_Ex_Stop control bit is not active (bit 0 in byte 0 of CAN_OUT_HEADER)
			1	Flag active: PI_Ex_Stop control bit is active (default)

18.2 CAN stop PD command (0561_{hex}: CAN_Stop)

Use this object to specify whether or not the CAN_Stop control bit is active in the OUT process data.

0561_{hex}: CAN stop PD command (read/write)				
Subindex	Data type	Length in bytes	Contents	
			Code	Meaning
0	UINT8	1	0	Flag inactive: CAN_Stop control bit is not active (bit 1 in byte 0 of CAN_OUT_HEADER)
			1	Flag active: CAN_Stop control bit is active (default)

18.3 CAN bus bit rate (0562_{hex}: CAN_Bit_Rate)

Use this object to set the transmission speed on the CAN bus to a value between 10 kbps and 1000 kbps.

0562_{hex}: CAN-Bus Bit-Rate (read/write)				
Subindex	**Data type**	**Length in bytes**	**Contents**	
			Value	**Baud rate in the CAN bus**
0	UINT32	4	10000	10 kBit/s
			20000	20 kbps
			50000	50 kbps
			100000	100 kbps
			125000	125 kbps
			250000	250 kbps (default)
			500000	500 kbps
			800000	800 kbps
			1000000	1000 kbps

18.4 Filter ($056A_{hex}$ ... $056D_{hex}$)

In order to reduce data flow in the receive direction, you can filter the received CAN messages (filtering via the identifier). Use objects $056A_{hex}$ to $056D_{hex}$ to parameterize the filter function.

The filters can be set independently of one another. Both have the same function.
Parameterize the filter for 11-bit CAN identifier using objects $056A_{hex}$ and $056B_{hex}$.
Parameterize the filter for 29-bit CAN identifier using objects $056C_{hex}$ and $056D_{hex}$.

Use the objects to parameterize the filter function, the lower filter limit (Min ID), and the upper filter limit (Max ID).

Each pair of values for the lower and upper filter limit defines a CAN identifier range that is to be filtered. Pairs of values that contain $EEEE_{hex}$ or $EEEEEEEE_{hex}$ are evaluated as invalid value pairs. After the first invalid value pair is detected, the other value pairs are no longer checked.

18.4.1 11-bit filter ($056A_{hex}$, $056B_{hex}$)

11-bit filter function ($056A_{hex}$: 11_Bit_Filter_Func)

$056A_{hex}$: 11-Bit-Filterfunktion (read/write)					
Subindex	**Data type**	**Length in bytes**	**Contents**		
			Code (hex)	**Meaning**	
0	UINT8	1	00	receive_all	Filter open. All CAN messages are received.
			01	block_all	Filter closed. No CAN messages are received.
			02	receive_ Filter11Bit Ranges	Filter active. CAN messages that are in at least one of the defined CAN identifier ranges are received.
					The ranges are defined in object $056B_{hex}$.
			03	block_ Filter11Bit Ranges	Filter active. CAN messages that are in at least one of the defined CAN identifier ranges are discarded.
					The ranges are defined in object $056B_{hex}$.

Enter all 60 filter pairs in object $056B_{hex}$. Use $EEEE_{hex}$ to identify unused filter pairs.

Filter masks for 11-bit CAN identifier ($056B_{hex}$: Filter11BitRanges)

$056B_{hex}$: Filtermasken für 11-Bit-CAN-Identifier (read/write)				
Subindex	**Data type**	**Length in bytes**	**Contents**	
			Min ID	**Max ID**
0	Record of UINT32	60 * 4	Min_ID_1	Max_ID_1
			Min_ID_2	Max_ID_2
			...	...
			Min_ID_60	Max_ID_60

Value range:

0000_{hex} ... $07FF_{hex}$
EEEEhex Unused

Make sure that the following requirements are met:

Min_ID_x	≤	Max_ID_x	x = 1 ... 60
Max_ID_x	≤	Max_ID_(x+1)	x = 1 ... 59

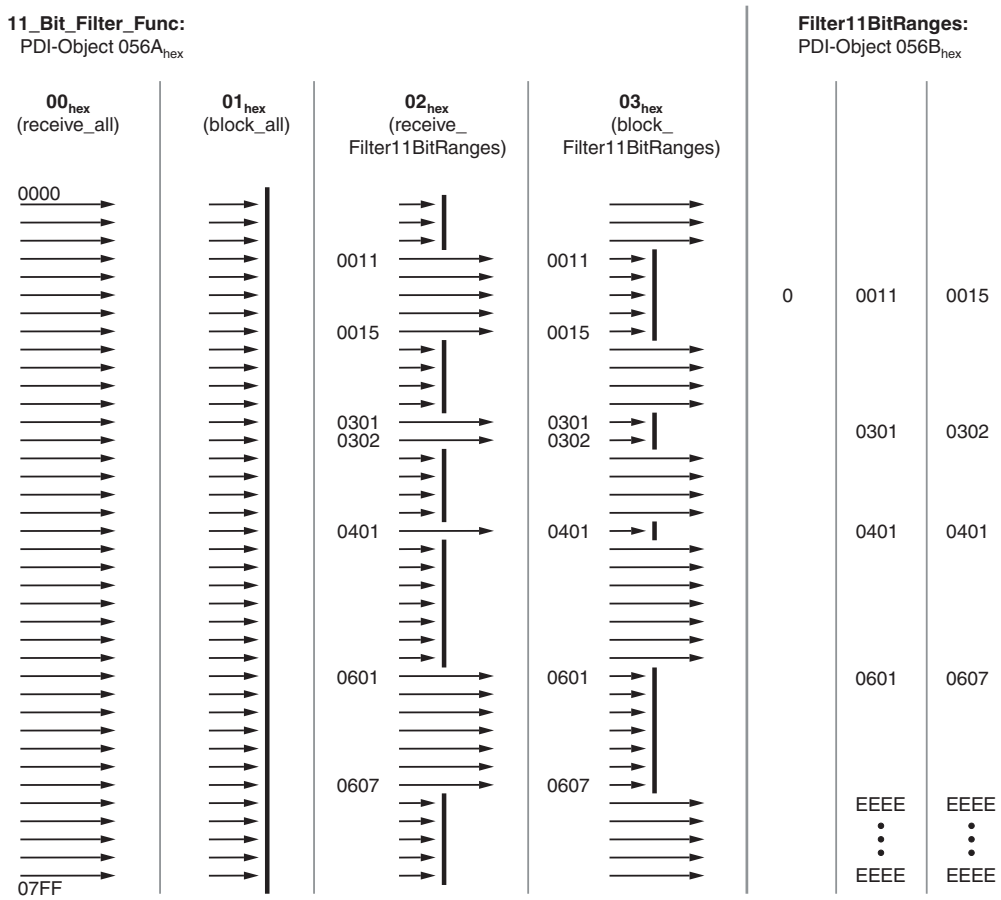
Setting for filters in the delivery state

$056B_{hex}$: Filtermasken für 11-Bit-CAN-Identifier (read/write)		
Subindex	**Min ID**	**Max ID**
	hex	**hex**
0	0000	07FF
	EEEE	EEEE
	...	...
	EEEE	EEEE

Example: setting for four filters

$056B_{hex}$: Filtermasken für 11-Bit-CAN-Identifier (read/write)		
Subindex	**Min ID**	**Max ID**
	hex	**hex**
0	0011	0015
	0301	0302
	0401	0401
	0601	0607
	EEEE	EEEE
	...	...
	EEEE	EEEE

Bild 7 Example: behavior of the filter function for CAN identifier depending on the settings in object $056A_{hex}$



The filter function that is parameterized using object $056C_{hex}$ is functionally identical. However, in this case it applies for 29-bit CAN identifier with a value range of the appropriate size.

18.4.2 29-bit filter ($056C_{hex}$, $056D_{hex}$)

29-bit filter function ($056C_{hex}$: 29_Bit_Filter_Func)

$056C_{hex}$: 29-Bit-Filterfunktion (read/write)					
Subindex	**Data type**	**Length in bytes**	**Contents**		
			Code (hex)	**Name**	**Meaning**
0	UINT8	1	00	receive_all	Filter open. All CAN messages are received.
			01	block_all	Filter closed. No CAN messages are received.
			02	receive_ Filter29Bit Ranges	Filter active. CAN messages that are in at least one of the defined CAN identifier ranges are received.
					The ranges are defined in object $056D_{hex}$.
			03	block_ Filter29Bit Ranges	Filter active. CAN messages that are in at least one of the defined CAN identifier ranges are discarded.
					The ranges are defined in object $056D_{hex}$.

Enter all 30 filter pairs in object $056D_{hex}$. Use $EEEE\ EEEE_{hex}$ to identify unused filter pairs.

Filter masks for 29-bit CAN identifier ($056D_{hex}$: Filter29BitRanges)

$056D_{hex}$: filter masks for 29-bit CAN identifier (read/write)				
Subindex	**Data type**	**Length in bytes**	**Contents**	
			Min ID	**Max ID**
0	UINT64	30 * 8	Min_ID_1	Max_ID_1
			Min_ID_2	Max_ID_2
			...	...
			Min_ID_30	Max_ID_30

Value range:

$0000\ 0000_{hex}$ ... $1FFF\ FFFF_{hex}$
$EEEE\ EEEE_{hex}$ Unused

Make sure that the following requirements are met:

Min_ID_x	≤	Max_ID_x	x = 1 ... 30
Max_ID_x	≤	Max_ID_(x+1)	x = 1 ... 29

Setting for filters in the delivery state

$056D_{hex}$: filter masks for 29-bit CAN identifier (read/write)		
Subindex	**Min ID**	**Max ID**
	hex	**hex**
0	0000 0000	1FFF FFFF
	EEEE EEEE	EEEE EEEE
	...	...
	EEEE EEEE	EEEE EEEE

19 Function block

The engineering software for Phoenix Contact controllers is PC Worx and PLCnext Engineer.

For PC Worx and PLCnext Engineer, the CANbus_Vx... function block library is available for the Axioline F CAN interface module. This library contains function blocks for communication and parameterization.

The AXL_CAN_COMM function block is used for communication between the AXL F IF CAN 1H and the higher-level controller.
This function block handles the sequential transfer of CAN_IN_Boxes and CAN_OUT_Boxes between the Axioline F CAN interface module and the higher-level controller.

AXL_CAN_Para... function blocks are used to parameterize the AXL F IF CAN 1H.

The function block library can be downloaded at phoenixcontact.net/products.