



## CAN-Functions on PCD3 & PCD2.M5 CPU Serie xx7

<b>0</b>	<b>Content</b>	
0.1	Document History .....	0-3
0.2	Brands and trademarks .....	0-3
<b>1</b>	<b>Introduction</b>	
1.1	CAN2.0 Specification (Data Link Layer) .....	1-1
1.2	Properties.....	1-1
1.3	Frame types .....	1-3
1.4	Message acceptance filtering .....	1-4
<b>2</b>	<b>Hardware</b>	
2.1	Communication interfaces .....	2-1
2.1.1	For PCD3.M6247 and PCD3.M6347 .....	2-1
2.1.2	CAN bus attachment, module PCD7.F7400 for the PCD2.M5x47.....	2-2
2.2	CAN Connection .....	2-3
2.2.1	Pin signal description .....	2-3
2.2.2	Connections and cabling.....	2-3
2.2.3	Maximum distance of the network .....	2-4
2.2.4	Shielding .....	2-4
2.3	Port ID Table for CAN Interfaces.....	2-4
<b>3</b>	<b>Function</b>	
3.1	General .....	3-1
3.2	CAN Direct Access.....	3-3
3.2.1	General .....	3-3
3.2.2	Polling vs interrupts.....	3-3
3.2.3	Remote transmission requests RTR .....	3-3
3.2.4	Initialisation sequence.....	3-3
3.2.5	Self-received frames .....	3-3
3.3	CAN Basic Services.....	3-4
3.3.1	General .....	3-4
3.3.2	Remote transmission requests RTR .....	3-4
3.3.3	Initialisation sequence.....	3-4
3.4	CAN Data Mapping .....	3-5
3.4.1	General .....	3-5
3.4.2	CAN process image .....	3-5
3.4.3	Configuration.....	3-5
3.4.4	Data transfer .....	3-6
3.4.5	Remote transmission requests RTR.....	3-6
3.4.6	Initialisation sequence.....	3-6
3.5	Simultaneous use of different operating modes.....	3-7
3.5.1	General .....	3-7
3.5.2	Priorities .....	3-7
3.5.3	Reception.....	3-7
3.5.4	Transmission.....	3-7

**4 Configuration and Programming**

4.1	System Functions .....	4-1
4.1.1	Polling vs Interrupts .....	4-1
4.1.2	Configuration.....	4-1
4.1.3	Initialisation sequence.....	4-2
4.2	User interface.....	4-3
4.2.1	General Functions.....	4-3
4.2.2	CAN Direct Access.....	4-6
4.2.3	CAN Basic Services.....	4-12
4.2.4	CAN Data Mapping.....	4-17
4.3	Programming model (CAN Direct Access).....	4-20
4.3.1	Transmission.....	4-20
4.3.2	Reception.....	4-21

**A Appendix**

A.1	Icons .....	A-1
A.2	Contact .....	A-2

**0.1 Document History**

Date	Version	Changes	Remarks
2005-12-15	EN01	-	Published Edition
2009-09-09	EN02	Ch02	Modifications for PCD2
2011-04-06	EN03	Ch02	Replaced wrong CPU type in 2.1.2
2013-10-07	EN04		New logo and new company name

**0.2 Brands and trademarks**

Saia PCD® and Saia PG5®  
are registered trademarks of Saia-Burgess Controls AG.

Technical modifications are based on the current state-of-the-art technology.

Saia-Burgess Controls AG, 2005. © All rights reserved.

Published in Switzerland

# 1 Introduction

1

This document gives an introduction and general description of the Saia PCD®s with CAN interface, as well as the specific functions of the xx7 user interface. The Hardware as well as the basic functionality is explained. Two systems are provided: a Classic version and a xx7 version.

The provided CAN functionality is a Layer-2 interface. The user is given a choice of several operating modes providing different levels of comfort and functionality. All functionality is accessed with System Functions by the user.

The CAN interface does not provide any CANopen functionality. However it is possible to access to simpler CANopen slaves through the given Layer-2 interface. This requires the user to implement all the relevant CANopen messages himself in the user program using the provided Layer-2 interface.

## 1.1 CAN2.0 Specification (Data Link Layer)

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed real-time control with a very high level of security. This chapter gives a short summary of the most relevant features defined in the CAN specification. For more details to any subject please refer to [www.can-cia.org](http://www.can-cia.org).

The CAN specification is divided in two parts: part A and part B where part B does also include and support all features of part A. The most important difference between the two parts concerns the format of the message identifier

<i>Specification</i>	<i>Message ID format</i>
CAN2.0 A	Standard 11 bits
CAN2.0 B	Extended 29 Bits

Messages in both formats can coexist on the same network.

CAN implementations that are designed according to part A, and CAN implementations that are designed according to part B of the CAN2.0 specification can communicate with each other as long as it made no use of the extended format.

## 1.2 Properties

### Message properties

Information on the bus is sent in fixed format messages of different length. CAN supports messages of up to 8 bytes data. Standard end extended message ID formats are possible.

### Bus properties

The CAN bus supports bit rates of up to 1 MBit/s. Different bit rates from 10 kBit/s to 1 MBit/s are recommended while the support for 20 kBit/s is mandatory.

The CAN specification prescribes several error detection mechanisms. Error counters and indicators allow a permanent control of the bus condition. Defect nodes are automatically switched off.

## Information routing

A CAN node does not make use of any information about the system configuration. For instance, no explicit station addresses are defined.

The identifier of a message does not indicate the destination of the message but describes the meaning of the data. Thus, every node can decide by message filtering whether the received data requires any action or not.

This mechanism allows multicast reception with time synchronization.

## Priorisation and arbitration

The priorities of messages are defined by their identifier. Messages with lower identifier have higher priority.

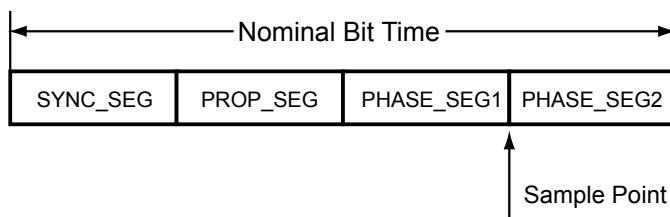
Any connected unit may start to transmit a message whenever the bus is idle. If 2 or more units start transmitting messages at the same time, the bus access conflict is resolved by bitwise arbitration using the message identifier. The mechanism of arbitration guarantees that neither information nor time is lost.

A data frame prevails over a remote frame (RTR). If a transmitted message is corrupted, an automatic retransmission will take place according to the message priority.

## Bit timing

The nominal bit time is divided into separate non-overlapping time segments. The length of each segment is described in so-called time quanta units. The chosen division of one bit time into time quanta is directly related to the generation of the bit rate. The ratio between the different time segments defines the sample point of one bit. The correct definition of the sample point depends on the network topology. The position of the sample point should be the same for all devices connected to the same network.

The different segments are defined as follows:



<b>Segment</b>	<b>Length</b>
SYNC_SEG	1 TIME QUANTUM
PROP_SEG	1,2,...,8 TIME QUANTA
PHASE_SEG1	1,2,...,8 TIME QUANTA
PHASE_SEG2	max (PHASE_SEG1; Information Processing Time)
INFORMATION PROCESSING TIME	<= 2 TIME QUANTA

### Time Segment 1

The sum of the propagation and phase segment 1 is commonly denoted as Time Segment 1 and is used in many CAN controllers for configuration of the bit timing. The propagation and phase segment 1 can then not be configured separately anymore. However, this is not required anyway since the sample point is located between the two phase segments.

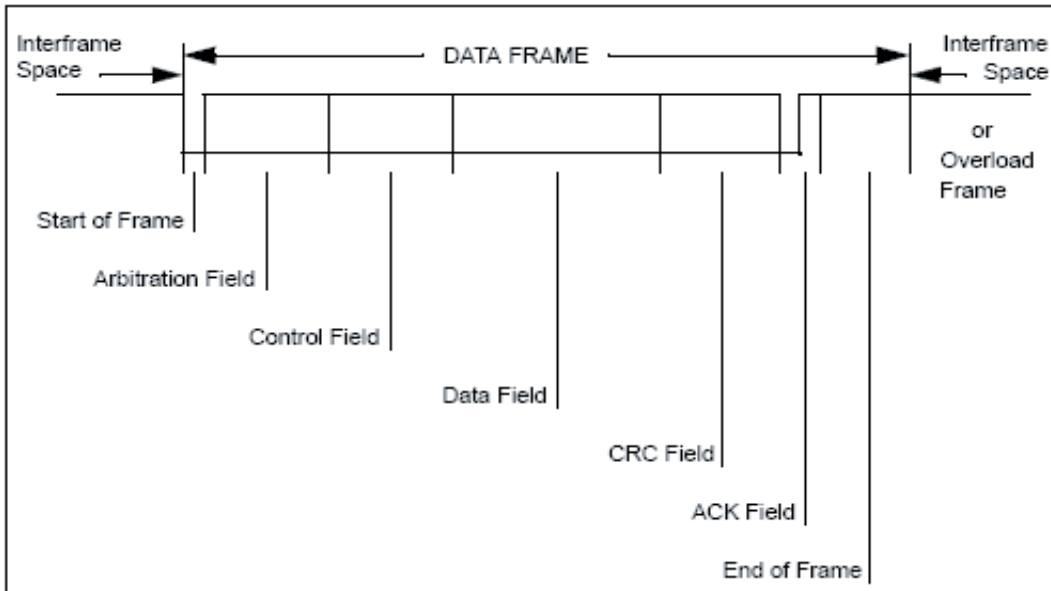
$$\text{TIME\_SEG1} = \text{PROP\_SEG} + \text{PHASE\_SEG1}$$

### 1.3 Frame types

#### Data frame

1

A data frame consists of several fixed format fields and up to 8 bytes data. The following image gives an overview of the data frame format.



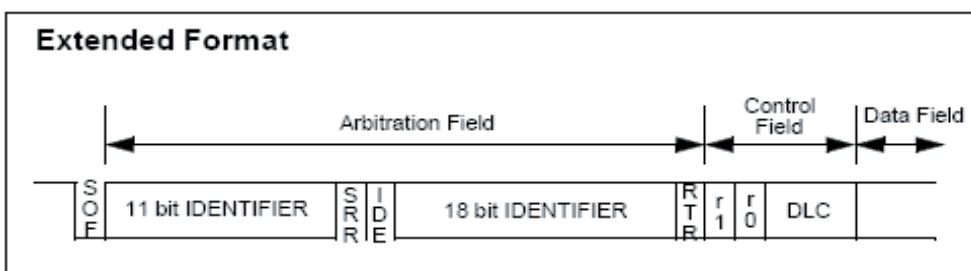
A data frame can be sent by any node at any time. The transmission is dependent on the arbitration process.

#### Remote frame

By sending a remote frame a node requiring data may request another node to send the corresponding data frame (Remote Transmission Request, RTR). The data frame and the corresponding remote frame are named by the same message identifier. The polarity of the RTR bit in the arbitration field indicates whether a transmitted frame is a data frame or a remote frame. A remote frame contains no data.

#### Identifier

The following image shows the different bits within the arbitration field (including the message identifier). It contains the IDE-bit indicating the format of the identifier and the RTR bit indicating the frame type.



**1.4 Message acceptance filtering**



Every CAN node implements a message acceptance filtering mechanism that allows receiving only a defined range of message identifiers while excluding the reception of any message identifiers outside of this range. Depending on the used CAN controller, several ranges may be defined.

The filtering is based on the use of a configured identifier and a filter mask. A message is accepted for reception into the buffer if the following condition is true:

$$(Received\ MsgID\ \mathbf{XOR}\ Configured\ MsgID)\ \mathbf{AND}\ Mask = 0$$

Two special cases can be derived from this condition (example for standard 11 bit identifiers):

<b>Mask</b>	<b>Condition</b>
0x000	Any message ID is accepted for reception
0x7FF	Only the configured message ID is accepted for reception

In the general case every bit of the mask that is set to 0 specifies the respective bit of the incoming message identifier to be “don’t care” for identifier matching. The following example illustrates the message filtering in the general case for standard 11 bit identifiers:

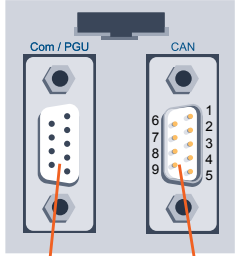
Hex	10	9	8	7	6	5	4	3	2	1	0	
<b>Acceptance ID</b>	0x137	0	0	1	0	0	1	1	0	1	1	1
<b>Acceptance Mask</b>	0x7EC	1	1	1	1	1	1	0	1	1	0	0
<b>Received Identifiers</b>		0	0	1	0	0	1	X	0	1	X	X
	0x124	0	0	1	0	0	1	0	0	1	0	0
	0x125	0	0	1	0	0	1	0	0	1	0	1
	0x126	0	0	1	0	0	1	0	0	1	1	0
	0x127	0	0	1	0	0	1	0	0	1	1	1
	0x134	0	0	1	0	0	1	1	0	1	0	0
	0x135	0	0	1	0	0	1	1	0	1	0	1
	0x136	0	0	1	0	0	1	1	0	1	1	0
	0x137	0	0	1	0	0	1	1	0	1	1	1




## 2 Hardware

### 2.1 Communication interfaces

#### 2.1.1 For PCD3.M6247 and PCD3.M6347

		RS 232/PGU		Controller Area Network (CAN)		
		D-Sub pin	Signal	CAN (9-pole D-Sub socket)		
				D-Sub pin	Signal	Explanation
		1	DCD	1	nc	not connected
		2	RXD	2	CAN_L*	Receive/transmit data negative
		3	TXD	3	GND*	0 V Data communication potent. CAN Ground
		4	DTR	4	nc	not connected
		5	GND	5	nc	not connected
		6	DSR	6	GND*	0 V Data communication potent. <b>Not connected on prototypes!</b>
		7	RTS	7	CAN_H*	Receive/transmit data positive
		8	CTS	8	nc	not connected
	PGU	CAN	9	nc	9	nc

\*) Galvanic isolated signals

For all types				Profibus signal	Profibus wiring		
Terminal block for supply, watchdog, interrupt inputs and Port 2							
	Pin	Signal	Explanation				
	1	D	Port 2; RS-485 up to 115 kbps usable as free user interface or Profi SBC S-Bus up to 187.5 kbps	RxD/TxD-N	A green		
	2	/D		RxD/TxD-P	B red		
	3	Int0	2 interrupt inputs or 1 fast counter				
	4	Int1					
	5	WD	Watchdog				
	6	WD					
	7	+24V	Power supply				
	8	GND					
RS-485 terminator switch							
Switch position	Designation	Explanation					
left	O	without termination resistors					
right	C	with termination resistors					

PCD3.M634x is a CPU that provides a galvanic isolated high-speed CAN interface.

The CPU has the same functions as a PCD3.M554x except that the SBC S-Net/MPI interface is replaced by CAN functionality. This document only specifies differences. Please refer to existing PCD3.M554x documentation for common features.

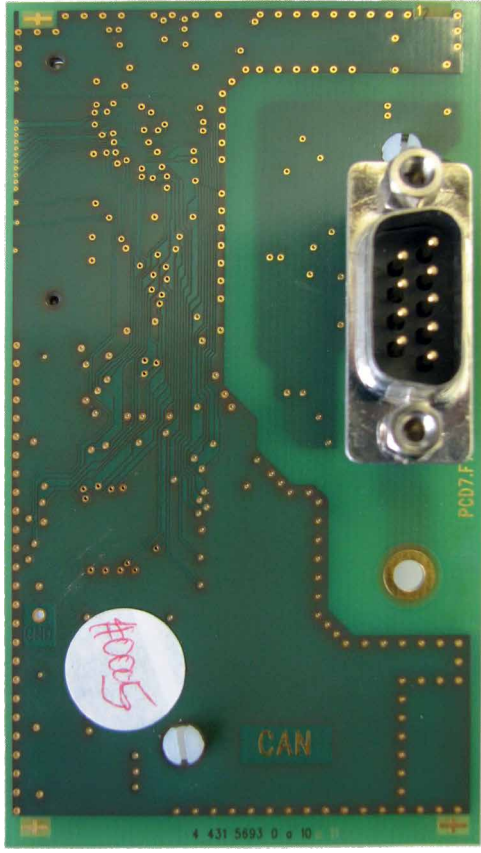
**2.1.2 CAN bus attachment, module PCD7.F7400 for the PCD2.M5x47**

The CAN Interface PCD7.F7400 provides a galvanic isolated high-speed CAN interface with up to 1 MBit/s. The CAN bus should be connected directly to the PCD7.F7400 module.

2

**PCD7.F7400**

to connect the CAN bus, 1 MBit/s  
 CAN Specifications: CAN 2.0B  
 Isolation: Galvanic isolation for CAN\_L and CAN\_H  
 Port: 8

PCD7.F7400		Controller Area Network (CAN)	
		CAN (9-pole D-Sub male) on slot C Port 8	
		D-Sub pin	Signal
1	nc	not connected	
2	CAN_L*	Receive/transmit data negative	
3	GND*	0 V Data communication potent. CAN Ground	
4	nc	not connected	
5	nc	not connected	
6	GND*	0 V Data communication potent.	
7	CAN_H*	Receive/transmit data positive	
8	nc	not connected	
9	nc	not connected	


<sup>\*)</sup> Galvanic isolated signals

**2.2 CAN Connection**

**2.2.1 Pin signal description**

CAN interface is the male D-Sub 9 on the right. Pin numbering is top view. The Pinout is compliant to CiA Draft Standard 102 Revision 2.0. This standard defines the CAN Physical Layer for Industrial Applications and is available for free on the Internet: [www.can-cia.org](http://www.can-cia.org)

2

CAN	Pin	CiA Standard 102	SBC wiring
	1	Reserved	not connected
	2	CAN_L bus line (dominant low)	CAN_L bus line (dominant low) Galvanic isolated
	3	CAN_GND CAN Ground	CAN_GND CAN Ground Galvanic isolated
	4	Reserved	not connected
	5	(CAN_SHLD) Optional CAN Shield	not connected
	6	(GND) Optional CAN Ground	CAN_GND CAN Ground Galvanic isolated <b>Not connected on prototype !</b>
	7	CAN_H bus line (dominant high)	CAN_H bus line (dominant high) Galvanic isolated
	8	Reserved (error line)	not connected
	9	(CAN_V+) Optional CAN external positive supply	not connected



D-Sub flange is connected to metallic PCD3 Backplate.

No bus termination resistor is provided. Nominal 120 Ω resistors must be placed at both extremities of the cable between CAN\_L and CAN\_H signals.

**2.2.2 Connections and cabling**

Commercial bus couplers having an integrated bus termination are available and allow to build a network that is not disrupted impedance match in case of unconnected cable.

Example of CAN bus coupler:  
ERNI ERbic CAN BUS



Other manufacturers like Delconec or FCT also provide CAN bus couplers. According to ISO 11898 Norm, Cable line impedance must be 120 Ω. For CAN cabling the use of twisted pair cable (2x2) having an impedance of 108...132 Ω is recommended.

Example of rigid cable: [www.intercond.com](http://www.intercond.com)

### 2.2.3 Maximum distance of the network

The maximum length of a CAN network is not only limited by signal levels but also by its delay. A bit-to-bit arbitration is done in the addressing phase and the issuer needs to listen to the answer on the cable to know if it grants the bus. This answer has to reach (delay) the issuer in one bit time.

At 1 MBit/s, it is theoretically possible to reach a length of 40 m.

However this 40 m distance is only possible with a non-isolated solution. With a galvanic isolated node, an additional delay occurs in the data path of the insulator (in our case 15 to 55 ns in every direction).

At 1 MBit/s, this reduces the maximum length to about 22 m. (Assumed each meter cable has a delay of about 5 ns).

Slower speeds also have a reduction of their maximum length but this is less relevant.

Possible baud rates and approximate cable length:

Baud rate in Bit	10 k	20 k	50 k	100 k	125 k	250 k	500 k	1 M
Total maximum length in m	5000	2500	1000	500	500	250	100	18
Maximum Stub length in m	Distances over 1000 m with repeater(s) same limits as 50 kbits		< 50	< 20	< 20	< 10	< 5	< 1
Sum length of all stubs			max. 250 m	max. 100 m	max. 100 m	max. 50 m	max. 25 m	max. 5 m



Never put a bus termination at the end of one stub.

**A CAN network requires a 120 Ω termination (between CAN\_L and CAN\_H) at both ends of the network even for short distances.**

### 2.2.4 Shielding

Shielded cable is necessary because High-speed CAN signal edges can not be limited to avoid electromagnetic pollution.

Shielding is also required for lower baud-rates.

Shielding can be connected at both ends or at just one end but at least at one place.

CAN galvanic isolation: minimum 500 V

Number of supported nodes: 64

## 2.3 Port ID Table for CAN Interfaces

PCD3.M6347	Port 0
PCD3.M6247	Port 0
PCD2.M5x47 with PCD7.F7400	Port 0

## 3 Function

### 3.1 General

#### CAN Layer-2

The CAN controller on the PLC extension implements a CAN Layer-2 access to the bus. All Layer-2 protocol handling is performed by the controller. The user can give a message to the controller and specify when it is sent. The user can also specify which messages are to be received by the controller and read the messages out of the controller memory after reception.

3

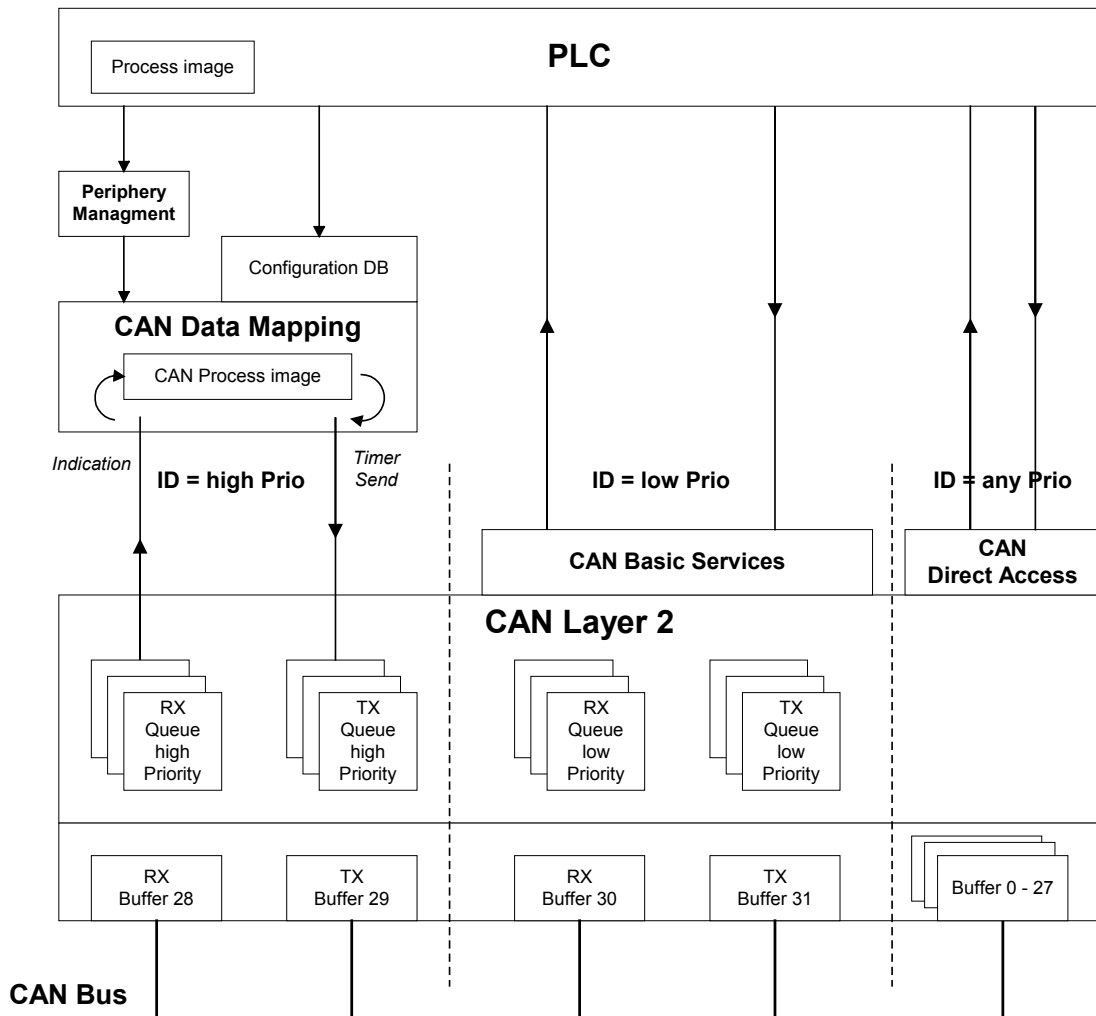
#### Operating modes

The CAN controller provides several buffers (message objects) which can be freely configured as transmission or reception buffers for any message ID. The user is provided with three different types of accesses to the controller functionality:

- **CAN Direct Access (FullCAN):** direct hardware access to all 32 buffers. The user can independently access all 32 buffers. An extensive interface provides all functionality integrated in the CAN controller. This mode is analogue to the FullCAN principle implemented in the controller itself.
- **CAN Basic Services (BasicCAN):** one Transmit and one Receive Queue allow a simpler handling of CAN communication in the user program. Several messages can be sent through the queue without waiting for the confirmation of each message. The receive queue ensures that no messages are lost on reception (if the queue depth is designed accordingly). The user interface is considerably simplified in comparison to the CAN Direct Access. This mode is analogue to the BasicCAN principle implemented in several simple CAN controllers with only one receive and one transmit path.
- **CAN Data Mapping:** the data mapping does simplify and automate the cyclic exchange of process data. This mapping is configured at start-up with the message IDs to be handled. The message data is directly mapped to process data of the PLC. All output messages are automatically sent by the data manager in a specified interval. The received messages are mapped to the process data by the manager.

The three modes can be used simultaneously with some limitations concerning the message ID ranges assigned to the different modes.

The following image gives an overview of the provided CAN architecture:



## 3.2 CAN Direct Access

### 3.2.1 General

The CAN controller provides 32 message objects. Each object represents one message buffer of up to 8 data bytes and is freely configurable as receive or transmit buffer and maskable.

The user interface provides access to the hardware controller as direct as possible. Therefore all accesses are oriented towards message objects rather than just simple messages. Each System Function call requires as parameter the Object ID referring to the respective message object in the CAN controller. This gives the application full access to the functionality of each CAN controller.

Of course not every CAN controller provides exactly the same functionality which is why a certain level of abstraction is required. Also some specialties of the used CAN controller must be known to the application programmer. These are explained in the detailed description hereafter.

### 3.2.2 Polling vs interrupts

Both System Functions for receiving and sending messages provide means to operate in poll and interrupt mode. Poll and interrupt modes can also be used concurrently for different message objects. The same object can also be polled to request its status while it is in interrupt mode.

The choice of poll or interrupt mode is entirely up to the application programmer and must be chosen according to the application's requirements.

Note however that it is not question of the hardware interrupts of the CAN controller here. The described interrupts are an abstraction to the application.

### 3.2.3 Remote transmission requests RTR

The driver supports handling of remote transmission requests for both receive and transmit objects.

A send object that is activated for RTR will send the frame only upon reception of a corresponding RTR from another station.

A receive object released for RTR will issue an RTR frame and store the response in the object's data buffer.

A message object is not constantly configured for RTR frames. For each job it can be used for RTR handling or not.

### 3.2.4 Initialisation sequence

Before any message transfers can occur the driver must be initialised and started by executing the following steps:

- 1) Configure the driver with the basic parameter set using the respective System Function
- 2) Configure each message object using the respective System Function
- 3) Release all reception buffers by calling the respective System Function for each receive object to be released

### 3.2.5 Self-received frames

The CAN controller does not receive self-transmitted frames even if there exists a matching receive message object.

### 3.3 CAN Basic Services

#### 3.3.1 General

The implementation of Queues provides the user with a concept that is analogue to the BasicCAN principle implemented by several CAN controllers. The user gets access to one receive and one send queue. In this mode the user does not care about or directly handle the message objects in the CAN controller. This task is performed by the Queue-Handler.

The queues use the two message objects with the highest number in the CAN controller. Access to these message objects is blocked for the CAN Direct Access System Functions as soon as the queues are configured.

It is possible to operate in CAN Direct Access and CAN Basic Services modes simultaneously. To ensure a correct handling of received messages, the ID ranges configured for either mode must be separated (i.e. not overlapping).

All CAN IDs can be received and sent over the same queue. The ID range to be handled by the queues is configurable. This leaves the possibility for the excluded ID range to be handled with the System Functions in the CAN Direct Access mode at the same time. The queue depth is configurable as well.

The send queue allows sending more than one telegram at a time without waiting for the successful transmission. There is no feedback possible for transmission of a single message. However when the queue becomes empty (i.e. the last message has successfully been sent) a corresponding interrupt call can be generated.

The receive queue avoids the loss of messages on reception. There is no indication of the reception of a single message. However when a message is received into an empty queue (i.e. the first message is received) an interrupt call can be generated.

#### 3.3.2 Remote transmission requests RTR

Remote requests RTR cannot be handled with-in the queues.

#### 3.3.3 Initialisation sequence

Before any message transfers can occur the driver must be initialised and started by executing the following steps:

- 1) Configure the driver with the basic parameter set using the respective System Function
- 2) Configure the receive and transmit queue using the respective System Function



## 3.4 CAN Data Mapping

### 3.4.1 General

The CAN Data Mapping provides an intermediary CAN application layer that automatically manages the copying, transmission and reception of PLC input and output media.

This principle shows certain similarities with the PDOs in CANopen. However it is clearly located somewhere between the CAN Layer 2 and the CANopen application layer and therefore it is SBC specific.

The CAN Data Mapping supports the following functionalities:

- A range of CAN message IDs can be assigned to the CAN Data Mapping such that the CAN data is directly mapped to the PLC input and output media
- Both, process image and direct periphery accesses are supported by the CAN Data Mapping (xx7)
- The configuration of the data mapping is defined in a DB/DBX
- The output media are automatically copied to the configured CAN telegrams at the end of the PLC cycle
- The transmission of all outputs is triggered by a configurable cyclic timer
- An option allows only sending telegrams whose data has changed since the last transmission
- The data of received CAN telegrams is automatically copied to the configured PLC input media at the beginning of the PLC cycle.

### 3.4.2 CAN process image

The CAN Data Mapping has its internal process image which represents the periphery to the PLC. The transfer of this image to and from other CAN nodes is performed by the CAN Data Mapping in an autonomous manner.

The following limitations are currently defined:

CAN process image size in bytes, total (Rx + Tx)	2048
Maximal number of messages, total (Rx + Tx)	256

8 bytes are allocated for each message ID even if not all bytes of that message are mapped to PLC media. The available memory space is evenly distributed between inputs (Rx) and outputs (Tx).

### 3.4.3 Configuration

CAN messages respectively IDs can be configured to exchange data with the CAN process image. All IDs must be in a high priority range which is definable with a single mask, typically from 0 to a threshold.

One CAN ID is associated to one single telegram per transfer direction. Hence, it is not possible to define several telegrams with the same ID for transmission (outputs) or reception (inputs).

The length of the **transmitted output telegrams** is defined by the highest byte in the message to which a PLC output is mapped. If bytes 0 and 1 are not mapped while bytes 2 and 3 are mapped for a specific message ID, then 4 bytes will be transmitted each cycle, where bytes 0 and 1 are always zero.

The length of the **received input telegrams** does not necessarily have to correspond to the number of mapped input bytes. Two cases are of importance:

- A received telegram contains more bytes than configured: All bytes configured in the DB are copied to the CAN process image. The remaining bytes are ignored resp. the user cannot access them
- A received telegram contains less bytes than configured: All received bytes are copied. The remaining bytes in the CAN process image are left unchanged.

#### 3.4.4 Data transfer

The data of all received and configured telegrams is automatically transferred to the internal CAN process image. The outputs are transferred from the CAN process image and are sent when the configured timer elapses. It is important to note that during transmission of the outputs the CAN bus is used to full 100% of its capacity.

The inputs are copied from the CAN process image to the PLC process image at the beginning of the PLC cycle. The outputs are transferred from the PLC process image to the CAN process image at the end of the PLC cycle. Furthermore, direct periphery accesses allow accessing the CAN process image at any time.

Since the received data is automatically copied from the CAN message to the process image it is not ensured that the user will get the data of every single received telegram. A following telegram will overwrite the data of the previous telegram with the same message ID. If the user has not read the data in-between, the data of the previous telegram will be "lost" to the user.

It is optionally possible to only send telegrams where at least one byte has changed since the last transmission. However this requires more computing power before each transmission.

Likewise it has to be kept in mind that after each reception the corresponding mapping entry must be searched. The time for this search increases with the number of input mapping entries.

As soon as the CAN Data Mapping is configured, buffers 28 and 29 of the CAN controller are reserved. Those are then not accessible anymore for CAN Direct Access or CAN Basic Services.

If the CAN Data Mapping receives a message whose ID is not configured, this message is ignored and data is lost.

#### 3.4.5 Remote transmission requests RTR

The use of Remote requests is not possible with-in the configuration of the CAN Data Mapping.

#### 3.4.6 Initialisation sequence

Before the CAN Data Mapping can be configured, the CAN driver must be initialised and started. The following steps must be executed:

- 1) Configure the CAN driver with the basic parameter set using the respective System Function.
- 2) Configure the CAN Data Mapping using the respective System Function

### 3.5 Simultaneous use of different operating modes

#### 3.5.1 General

Any combination of CAN operating modes can be used simultaneously. To ensure a correct handling of received messages, the ID ranges configured for either mode must be separated in the ideal case and the priorities of message identifiers and message buffers must be respected.

It is basically possible to send and receive telegrams defined with-in the ID range of the CAN Data Mapping or the CAN Basic Services through buffers 0-27 in the CAN Direct Access mode (i.e. overlapping ID ranges). This is because the buffers with lower numbers are processed with higher priority.

#### 3.5.2 Priorities

The priorities for processing of the messages associated with the different operating modes do not depend on the used message identifier but on the used message buffers. For both, reception and transmission, the buffers are processed in ascending order according to the buffer number.

The message buffer numbers for the different operating modes have been chosen such that a useful prioritization is possible. Consequently the corresponding priorities of the message identifiers should be chosen according to the recommendations below.

The principle is based on the assumption that the CAN Data Mapping is used for high priority data exchange (analogue to PDOs in CANopen) whereas the CAN Basic Services are used for configuration or sporadic low priority messages (analogue to SDOs in CANopen).

The following table gives an overview of operating mode and message prioritization:

Priority	Reception	Message ID	Transmission	Message ID
<b>1 (highest)</b>	CAN Direct Access	any Priority	CAN Direct Access	any Priority
<b>2</b>	CAN Data Mapping	high Priority	CAN Data Mapping	high Priority
<b>3 (lowest)</b>	CAN Basic Services	low Priority	CAN Basic Services	low Priority

The CAN Direct Access is not associated to a specific message priority and is placed at highest processing priority. See the following chapters for possible inconsistencies that can occur.

#### 3.5.3 Reception

Let's assume that CAN Direct Access and CAN Data Mapping are used simultaneously and the defined ID ranges are overlapping. A potential problem occurs when a receive buffer of the CAN Direct Access experiences an overrun. In this case the message is transferred to the CAN Data Mapping where the message is lost if no corresponding mapping entry exists. Ideally the configuration for the CAN Data Mapping should exclude the ID range to be handled simultaneously in any of the other modes.

#### 3.5.4 Transmission

It is important to note that an operating mode of higher transmission priority can potentially block any message transfer of an operating mode with lower transmission priority for an undefined time. This happens if too much message transfers are generated by the operating mode with higher priority.

## 4 Configuration and Programming

### 4.1 System Functions

The Step7 CAN interface provides several SFBs for configuration, data transfer and status.

#### General functions

**SFB403: CAN\_CFG**, Configure CAN-Driver

**SFB404: CAN\_STAT**, Request current status of CAN-Driver

These functions provide the basic configuration and status facilities required for any operating mode.

4

#### CAN Direct Access

**SFB402: CAN\_CFGO**, Configure Message objects

**SFB401: CAN\_TX**, Send message

**SFB400: CAN\_RX**, Receive message

These functions give direct and flexible access to the hardware CAN controller. This mode is analogue to the FullCAN principle implemented in the controller itself.

#### CAN Basic Services

**SFB412: CAN\_CFGQ**, Configure Queues

**SFB411: CAN\_TXQ**, Send message to Queue

**SFB410: CAN\_RXQ**, Receive message from Queue

These functions provide an additional hardware abstraction where queues handle the direct hardware access and the user does not have to be aware of the low-level CAN handling. This concept that is analogue to the BasicCAN principle implemented by several CAN controllers.

#### CAN Data Mapping

**SFB413: CAN\_DMAP**, Configure Data Mapping

#### 4.1.1 Polling vs Interrupts

All System Functions for receiving and sending messages provide means to operate in poll and interrupt mode. Poll and interrupt modes can also be used concurrently for different message objects. The same object can also be polled to request its status while it is in interrupt mode. Basic Services can also be used in poll and interrupt mode.



If polling and interrupt is used concurrently for the same System Function, it must be ensured that not the same Instance-DB is given as parameter in both cases.

The choice of poll or interrupt mode is entirely up to the application programmer and must be chosen according to the application's requirements.

#### 4.1.2 Configuration

The CAN configuration consists of different parts relating to the different operating modes.

The configuration is given by calling the respective SFBs. Some of them require an additional DB containing some configuration structures. These DBs must be edited manually.

The following System Functions are provided for the different configuration parts.

These can also be used to change the configuration previously applied at run-time

with the same System Functions.

**SFB403: CAN\_CFG**, Configure CAN-Driver

**SFB402: CAN\_CFGO**, Configure Message objects

**SFB412: CAN\_CFGQ**, Configure Queues

**SFB413: CAN\_DMAP**, Configure Data Mapping

### 4.1.3 Initialisation sequence

The following initialisation sequences must be followed for each operating mode. Of course if several operating modes are used, the call to SFB403 is required once only.

4

#### CAN Direct Access

- 1) Configure the driver with the basic parameter set using SFB403
- 2) Configure each message object using SFB402
- 3) Release all reception buffers by calling SFB400 for each receive object to be released

#### CAN Basic Services

- 1) Configure the driver with the basic parameter set using SFB403
- 2) Configure the receive and transmit queue using SFB412

#### CAN Data Mapping

- 1) Configure the CAN driver with the basic parameter set using SFB403
- 2) Configure the CAN Data Mapping using SFB413

**4.2 User interface**

**4.2.1 General Functions**

**SFB 403: CAN\_CFG, Config Driver**

SFB403 serves to set the basic parameters for the CAN driver. By calling this SFB all previous settings and object configurations performed with SFB402 and SFB412 are deleted and the driver and message objects are reset to their initial state. Typically this SFB will be only be called once in OB100.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port identifier
2.0	in	TimeSeg1	INT	0	Time segment 1: Propagation + phase seg1
4.0	in	Baudrate	DINT	L#0	Baudrate
8.0	in	RxIntOB	INT	0	Receive Interrupt OB Number: 40-47
10.0	in	RxIntPrm	WORD	W#16#0	Receive Interrupt parameter
12.0	in	TxIntOB	INT	0	Transmit Interrupt OB Number: 40-47
14.0	in	TxIntPrm	WORD	W#16#0	Transmit Interrupt parameter
16.0	out	RetVal	WORD	W#16#0	Return value

Two different global interrupt OBs can be defined, one for reception and one for transmission, each with a dedicated parameter. Of course receive and transmit interrupts can also be programmed to the same OB. The parameter can be used to distinguish the CAN interrupt from other interrupt sources using the same interrupt OB. The local data of the interrupt OB will also provide the Object ID of the object that caused the interrupt.

**PortID**

Reserved, set to 0

**TimeSeg1**

TimeSeg1 lets the user program the value of the time segment 1 with-in the bit timing. The time segment 1 can theoretically be programmed to values from 2 to 16 and is the sum of the propagation time and the phase segment 1 according to the CAN2.0 norm. However refer to the tables below for the accepted values for TimeSeg1 and the programmed values for all other segments in the different situations.



**OKI ML9620 Table 0**, TimeSegment1=000xH where x=value of Time Segment 1.

This table is not applicable for baudrate 1 MBit

	TimeSeg1 parameter value	Time Segment 1	Propagation Segment	Phase Segment 1	Phase Segment 2	SJW
	7	7	1	6	8	4
According to norm	8	8	1	7	7	4
	9	9	3	6	6	4
	10	10	5	5	5	4
	11	11	7	4	4	4
	12	12	8	4	3	3
	13	13	8	5	2	2
	14	14	8	6	1	1

**OKI ML9620 Table 1**, TimeSegment1=010xH where x=value of Time Segment 1 The table is applicable for all baudrates

	TimeSeg1 parameter value	Time Segment 1	Propagation Segment	Phase Segment 1	Phase Segment 2	SJW
	258	2	1	1	5	1
	259	3	1	2	4	2
Norm	260	4	1	3	3	3
	261	5	3	2	2	2
	262	6	5	1	1	1

#### **Baudrate**

Specifies the baudrate at which the CAN is to be operated. Supported baudrates are: 10'000 Bit/s, 20'000 Bit/s, 50'000 Bit/s, 100'000 Bit/s, 125'000 Bit/s, 250'000 Bit/s, 500'000 Bit/s, 1'000'000 Bit/s

#### **RxIntOB**

Specifies the OB that is called when a receive interrupt is requested. OB numbers 40-47 are supported.

#### **RxIntPrm**

Specifies the parameter that is given at offset 6 in the local data of the receive interrupt OB.

#### **TxIntOB**

Specifies the OB that is called when a transmit interrupt is requested. OB numbers 40-47 are supported.

#### **TxIntPrm**

Specifies the parameter that is given at offset 6 in the local data of the transmit interrupt OB.

#### **RetVal**

Possible return values of the SFB are:

Return value (Hex)	Meaning
0000	SFB successfully executed
80A0	Negative acknowledge from CAN controller
8187	Invalid Port number
8190	Invalid parameter: invalid TimeSeg1
8191	Invalid baudrate



**SFB 404: CAN\_STAT, Get Status**

The Status of the CAN-Driver is only provided in polling mode by calling the SFB404. The meaning and rules for the different status indicators are defined in the CAN2.0 norm.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port identifier
2.0	out	RxErrCtr	BYTE	B#16#0	Receive error counter
3.0	out	TxErxCtr	BYTE	B#16#0	Transmit error counter
4.0	out	BusOff	BOOL	FALSE	Bus Off
4.1	out	Error	BOOL	FALSE	Error: Tx bit, ack, CRC, format, stuff
4.2	out	Warning	BOOL	FALSE	RxErrCtr or TxErxCtr are >= 96
4.3	out	ErrorState	BOOL	FALSE	0: Error active, 1: Error Passive
6.0	out	RetVal	WORD	W#16#0	Return value of SFB

4

**PortID**

Reserved, set to 0

**RxErrCtr**

Returns the current value of the Receive Error Counter as defined by the CAN2.0 norm.

**TxErxCtr**

Returns the current value of the Transmit Error Counter as defined by the CAN2.0 norm.

**BusOff**

This bit is set to 1 when the CAN controller is in Bus Off state according to the CAN2.0 norm.

**Error**

This bit is set to 1 if any error is pending.

**Warning**

A warning is issued by setting this bit to 1 when any of the two error counters is equal or higher than 96.

**ErrorState**

This bit indicates the current error state of the CAN controller according to the CAN2.0 norm:

- 0: Error active
- 1: Error passive

**RetVal**

Possible return values of the SFB are:

Return value (Hex)	Meaning
0000	SFB successfully executed
8186	Invalid parameter
8187	Invalid Port number



## 4.2.2 CAN Direct Access

### SFB 402: CAN\_CFGO, Config Object

SFB402 serves for configuration of all available message objects.

A DB must be given containing the configuration for all message objects to be configured. The structure of the DB is described on the next page. The DB can contain the configuration for all message objects ever to be used. The SFB402 is then called at start-up once for all.

It is also possible to configure or reconfigure one or more objects at a later stage at run-time.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port ID
2.0	in	DBNr	INT	0	DB number containing the configuration
4.0	in	Offset	INT	0	Indexed offset in DB
6.0	in	NbObj	INT	0	Number of Object configurations in DB
8.0	out	RetVal	WORD	W#16#0	Return value

#### **PortID**

Reserved, set to 0

#### **DBNr**

Number of DB containing the configuration for the message objects.

#### **Offset**

Specifies the first object structure in the DB to be used for configuration. The offset must be given in 'number of object structures', rather than in number of bytes.

#### **NbObj**

Specifies the number of objects to be configured starting from the position given by the Offset.

#### **RetVal**

Possible return values of the SFB are:

Return value (Hex)	Meaning
0000	SFB successfully executed
80A0	Negative acknowledge from CAN controller
80C0	Access to message object denied
8186	Invalid ID
8187	Invalid Port number
8190	Invalid parameter: Offset and NbObj are inconsistent, or OB numbers configured with SFB403 are invalid
8F33	Error with DB number
8F3A	DB not loaded

### Object configuration structure

The object configuration DB contains a header of 4 WORDs followed by an array of object definitions:

Adresse	Name	Typ	Anfangswert
0.0		STRUCT	
+0.0	DBVersion	INT	0
+2.0	Reserved1	INT	0
+4.0	Reserved2	INT	0
+6.0	Reserved3	INT	0
+8.0	Obj	ARRAY[0..15]	
*16.0		"CanObjCfgDef"	
=264.0		END_STRUCT	

4

#### DBVersion

Version of the DB structure, set to 0

#### Reserved1, Reserved2, Reserved3

Reserved for future use, set to 0

A single object is defined as this:

Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	ObjID	INT	0	Number of Message buffer (Object)
+2.0	MsgID	DWORD	DW#16#0	Identifier of message
+6.0	Mask	DWORD	DW#16#FFFFFF	Identifier Mask (Rx only)
+10.0	IDE	BOOL	FALSE	0 -> Standard ID, 1 -> Extended ID
+10.1	Dir	BOOL	FALSE	Direction: 0 -> Rx, 1 -> Tx
+10.2	EnableInt	BOOL	FALSE	1: Enable interrupt OB
+12.0	Timeout	DINT	L#0	Timeout in ms
=16.0		END_STRUCT		

#### ObjID

The Object ID directly refers to the message object on the CAN controller. Supported values are:

OKI: 0-31

Since every object structure contains the object ID, the object configurations do not have to be placed in any specific order with-in the DB. However if two or more objects have the same Object ID it is always the latter that prevails over the previous.

#### MsgID

This specifies the Message ID to be handled by this message object. In case of a receive object only messages with this ID are received into this buffer. In case of a transmit object the configured Message ID can be overwritten by the SFB401 Send Message if it specifies a different Message ID.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and should be set to 0.

#### Mask

The Mask can be used to extend the matching of received messages to several message IDs rather than the single one defined by the object. Every bit of the mask that is set to 0 specifies the respective bit of the incoming message ID to be "don't care" for ID matching. If only one specific ID is to be received, i.e. no mask used, the mask should initialise to all 1.

A message is accepted for reception into the buffer if the following condition is true:  
 (Received MsgID **XOR** Configured MsgID) **AND** Mask = 0

The mask does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining upper bits are reserved and should be set to 0.

**IDE**

Specifies the ID format of the message:  
 0: Standard ID (11 bit, CAN2.0A)  
 1: Extended ID (29 bit, CAN2.0B)

**Dir**


Specifies the direction of the message object:  
 0: Receive object  
 1: Transmit object

**EnableInt**

Enables the call of the interrupt OB specified with SFB 403 after a successful transfer if set to 1.

**Timeout**

Specifies the Timeout in milliseconds after which a send job will be aborted. To deactivate the timeout functionality Timeout must be set to 0.

	No interrupt is generated if a timeout expires. This functionality only works in polling mode.
--	--

**SFB 401: CAN\_TX, Send Message**

SFB401 is used for sending messages on the CAN bus and also for requesting the status in polling mode.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	ObjID	INT	0	Number of Message buffer (Object)
2.0	in	PortID	INT	0	Port identifier
4.0	in	MsgID	DWORD	DW#16#0	Identifier of message
8.0	in	DLC	INT	0	Data Length Code
10.0	in	TxData	ANY		Pointer to data (up to 8 bytes)
20.0	in	MultipleTx	BOOL	FALSE	1: Multiple data transfer
20.1	in	RTR	BOOL	FALSE	1: Send data only as response RTR
20.2	in	Act	BOOL	FALSE	1: Send telegram, 0: Request status only
20.3	in	Abort	BOOL	FALSE	1: Abort send job
22.0	out	Done	BOOL	FALSE	Previous message has been transmitted
22.1	out	Error	BOOL	FALSE	An error occurred
24.0	out	Status	WORD	W#16#0	Return value of SFB

Only one send job can be processed by a message object at a time. The message priorities for transmission are defined by the Object IDs of the used objects. The object with the lowest ID has the highest priority and will be sent first if several messages are pending for transmission. The format of the ID (i.e. standard or extended) is defined by the configuration previously applied. A message object can only be used for transmission if it has been configured as such.

**ObjID**

Object ID of the object to be used for the transmission of the message.

**PortID**

Reserved, set to 0.

**MsgID**

ID of the message to be transmitted.

The format of the ID (i.e. standard or extended) is defined by the configuration previously applied.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and should be set to 0.

4

**DLC**

Data Length Code, number of bytes to be transmitted.

**TxData**

TxData points to the data bytes to be transmitted in the message.

**MultipleTx**

This bit is evaluated only if RTR is set. In this case it specifies whether the message is to be transmitted only once or multiple times as a response to a remote request.

**RTR**

If this bit is set to 1, this message will only be sent as a response to a remote request.

**Act**

Activates a send job if this flag changes from 0 to 1. If no positive edge of this flag is detected, all the input fields dedicated to the send job are ignored and only the Status and output bits are updated. This is to be used in polling mode.

A send job is marked as pending until it has successfully been transmitted. If in this case the programmer tries to activate a new job, the Status will indicate a pending job. An RTR frame to be transmitted multiple times will be marked as pending only until the first successful transmission.

**Abort**

When this bit is set to 1 the currently active send job is aborted. No other action is taken.

**Done**

The Done bit indicates if a send job has successfully been executed. This bit is to be used in polling mode and will only be set for one single call to the SFB for one specific send job.

A send job will never be completed with-in the same call to this SFB. If the Done bit is 1 when a new job has been initiated with Act = 1, then this refers to the previous send job. Therefore the programmer should always follow the rules in the programming model (chap. 4.3.1) in polling mode to assure a coherent interpretation of this bit.

**Error**

This bit is set to 1 if any error occurred during transmission or execution of the SFB.

**Status**

The Status of a send job can take the following values:

Done	Error	Status (Hex)	Meaning
1	0	0000	Job successfully terminated
0	0	0000	No job pending
0	0	8181	Job is accepted and being executed
0	0	8182	Job has been aborted
0	1	8185	Invalid length parameter DLC
0	1	8186	Invalid parameter ID
0	1	8187	Invalid Port number
0	1	8190	Invalid parameter: ANY pointer inconsistency
0	1	80A0	Negative acknowledge from CAN controller
0	1	80C0	Access to message object denied
0	1	80C2	Too many jobs pending
0	1	80C4	Timeout has expired

**SFB 400: CAN\_RX, Receive Message**

SFB400 is used for receiving messages on the CAN bus. The same SFB is intended for releasing a message buffer for reception and also for reading a received message out of a message buffer. In both polling and interrupt mode this SFB must be called to get the received message.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	ObjID	INT	0	Number of Message buffer (Object)
2.0	in	PortID	INT	0	Port identifier
4.0	in	RxData	ANY		Pointer to data (up to 8 bytes)
14.0	in	ReleaseRx	BOOL	FALSE	1: Release buffer for reception
14.1	in	RTR	BOOL	FALSE	1: Send a remote request
16.0	out	MDR	BOOL	FALSE	New Data Ready and copied to RxData
16.1	out	Error	BOOL	FALSE	An error occurred
18.0	out	MsgID	DWORD	DW#16#0	Identifier of received message
22.0	out	DLC	INT	0	Data Length Code
24.0	out	Status	WORD	W#16#0	Return value of SFB
	in_out				

The format of the ID (i.e. standard or extended) is defined by the configuration previously applied.

A message object can only be used for reception if it has been configured as such.

**ObjID**

Object ID of the object containing a message to be read.

**PortID**

Reserved, set to 0

**ReleaseRx**

By setting this bit to 1 the message object is directly released for reception of a next frame after the data has been copied to the RxData buffer. Additionally if a receive message object has not been released yet, a call to this SFB with this bit set to 1 is required to enable reception for this message object.

A message object cannot be deactivated by setting ReleaseRx to 0 after it has been released. It can only be not released after a successful reception of a message.

Likewise an already released message object cannot be released with different settings until a successful reception of a message. However an RTR receive buffer can be overwritten and released with new settings. Therefore ReleaseRx must always be

set to 0 while polling a RTR receive buffer, otherwise a remote request will be issued with each call to the SFB.

### **RTR**

If this bit is set to 1, a remote request will be issued by the CAN controller and the response frame will be received into this message object.

### **NDR**

New Data Ready indicates that the data of a newly received frame has been copied to the RxData buffer. This bit is to be used in polling mode and will only be set for one single call to the SFB for one specific receive frame.

### **Error**

This bit is set to 1 if any error occurred during reception or execution of the SFB.

### **MsgID**

This field contains the message ID of the received frame which is relevant if a mask has been used for message matching. If no mask is used the ID will be the same as the configured one of the object.

The format of the ID (i.e. standard or extended) is defined by the configuration previously applied.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and set to 0.

### **DLC**

Data Length Code, number of received bytes in RxData buffer.

### **RxData**

Pointer to the memory where the received data bytes shall be copied to if the NDR bit is set to 1.

### **Status**

The Status of a receive job can take the following values:

<b>NDR</b>	<b>Error</b>	<b>Status (Hex)</b>	<b>Meaning</b>
1	0	0000	New data copied
0	0	8180	No data has been received yet
1	1	80C2	Overrun, new data was received into a full buffer
0	1	8186	Invalid parameter ID
0	1	8187	Invalid Port number
0	1	8190	Invalid parameter: ANY pointer not enough memory
0	1	80A0	Negative acknowledge from CAN controller
0	1	80C0	Access to message object denied



### Interrupt OBs

Two global interrupt OBs can be defined in the CAN Direct Access mode using SFB403. The call to the respective interrupt OB can be enabled individually for each message object. The OB is then called if a message has successfully been transmitted or if new data has been received.

The local data of the interrupt OB contains the standard data as it is defined for any OB. Three parameters however are specific if the OB is used for CAN interrupts.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	temp	OB41_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), 1
1.0	temp	OB41_STRT_INF	BYTE		16#41/42/43/44 (Interrupt 1/2
2.0	temp	OB41_PRIORITY	BYTE		Priority of OB Execution
3.0	temp	OB41_OB_NUMBR	BYTE		41 (Organization block 41, OB
4.0	temp	OB41_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB41_IO_FLAG	BYTE		16#54 (input module), 16#55 (
6.0	temp	OB41_PARAM	WORD		Parameter of interrupt source
8.0	temp	OB41_OBJ_ID	INT		Object ID
10.0	temp	OB41_PORT_ID	INT		Port ID
12.0	temp	OB41_DATE_TIME	DATE_AND_TIME		Date and time OB41 started

#### **Param, Offset 6**

This field contains the configured parameter which can be used to distinguish the CAN from other interrupt sources if they use the same interrupt OB.

#### **ObjID, Offset 8**

Object ID of the message object that caused the interrupt.

#### **PortID, Offset 10**

Reserved, always set to 0

### 4.2.3 CAN Basic Services

#### **SFB 412: CAN\_CFGQ, Config Queues**

SFB412 serves to configure and activate the queues. Two queues are installed, one for reception and one for transmission. Typically this SFB is only called once at start-up in OB100.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port ID
2.0	in	MsgID	DWORD	DW#16#0	Message ID to be handled by Queue
6.0	in	Mask	DWORD	DW#16#0	Mask to define ID range to be handled
10.0	in	RxQueueDepth	INT	0	Receive Queue depth
12.0	in	RxIntOB	INT	0	Receive interrupt OB number: 40-47
14.0	in	RxIntPrm	WORD	W#16#0	Receive interrupt parameter
16.0	in	TxQueueDepth	INT	0	Transmit Queue depth
18.0	in	TxIntOB	INT	0	Transmit interrupt OB number: 40-47
20.0	in	TxIntPrm	WORD	W#16#0	Transmit interrupt parameter
22.0	in	IDE	BOOL	FALSE	0 = Standard, 1 = Extended ID
22.1	in	RxEnableInt	BOOL	FALSE	Enable Receive interrupt OB
22.2	in	TxEnableInt	BOOL	FALSE	Enable Transmit interrupt OB
24.0	out	RetVal	WORD	W#16#0	Return value

#### **PortID**

Reserved, set to 0

**MsgID**

This specifies the Message ID to be handled by the receive queue.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and should be set to 0.

**Mask**

The mask can be used to extend the matching of received messages to several message IDs rather than the single one defined by the MsgID. Every bit of the mask that is set to 0 specifies the respective bit of the incoming message ID to be “don’t care” for ID matching.

With the mask an ID range can be defined that must be handled by the queues. The queues can also handle the entire ID range in which case the mask must be set to 0. This is useful if only queues are used and the FullCAN mode is not considered.

A message is accepted for reception into the Queue if the following condition is true:

(Received MsgID **XOR** Configured MsgID) **AND** Mask=0

The mask does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining upper bits are reserved and should be set to 0.

**RxQueueDepth**

The depth of the receive queue specifies how many messages can be hold by the queue.

**RxIntOB**

Specifies the OB that is called when the first message is received into the queue. The OB is only called when a receive interrupt is requested. OB numbers 40-47 are supported.

**RxIntPrm**

Specifies the parameter that is given at offset 6 in the local data of the receive interrupt OB.

**TxQueueDepth**

The depth of the send queue specifies how many messages can be hold by the queue.

**TxIntOB**

Specifies the OB that is called when the last message from the queue has successfully been transmitted. The OB is only called when a transmit interrupt is requested. OB numbers 40-47 are supported.

**TxIntPrm**

Specifies the parameter that is given at offset 6 in the local data of the transmit interrupt OB.

**IDE**

Specifies the ID format of the messages:

0: Standard ID (11 bit, CAN2.0A)

1: Extended ID (29 bit, CAN2.0B)

**RxEnableInt**

Enables the call of the interrupt OB specified with RxIntOB if set to 1.

**TxEnableInt**

Enables the call of the interrupt OB specified with TxIntOB if set to 1.



**RetVal**

Possible return values of the SFB are:

Return value (Hex)	Meaning
0000	SFB successfully executed
80A0	Negative acknowledge from CAN controller
80C3	Not enough memory available
8187	Invalid Port number
8190	Invalid parameter (OB number)

**SFB 411: CAN\_TXQ, Send Queue**

SFB411 is used to insert a message into the send queue.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port identifier
2.0	in	TxData	ANY		Pointer to data (up to 8 bytes)
12.0	in	MsgID	DWORD	DW#16#0	Identifier of message to be transmitted
16.0	in	DLC	INT	0	Data Length Code (number of bytes in TxData)
18.0	in	Priority	BOOL	FALSE	1: Insert in queue according to priority of ID
20.0	out	Error	BOOL	FALSE	An error occurred
22.0	out	Status	WORD	W#16#0	Return value of SFB

The queue is implemented as a FIFO. However with the priority bit the insertion in the queue can be forced according to the priority of the message ID. If a message is inserted when the queue is full an error is displayed and the message is discarded.

**PortID**

Reserved, set to 0.

**TxData**

Pointer to the data bytes to be transmitted in the message.

**MsgID**

This field contains the ID of the message to be transmitted.

The format of the ID (i.e. standard or extended) is defined by the configuration previously applied with SFB412.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and should be set to 0.

**DLC**

Data Length Code, number of bytes to be transmitted

**Priority**

When this bit is set to 1 the message is inserted into the queue according to the priority of the message ID. The new message is inserted previous the first element with lower priority in the queue. Any message that already is in the send buffer of the CAN controller as well as the first element in the queue cannot be considered for priority insertion even if the message to be inserted has a higher priority. Hence the inserted element can at best be inserted in the second position of the queue

**Error**

This bit is set to 1 if any error occurred during transmission or execution of the SFB.

**Status**

The Status of a send queue can take the following values:

Error	Status (Hex)	Meaning
0	0000	Message successfully inserted in queue
1	8185	Invalid length parameter DLC
1	8187	Invalid Port number
1	8188	No queue configured
1	80A0	Negative acknowledge from CAN controller
1	80C2	Queue is full, message is ignored

**SFB 410: CAN\_RXQ, Receive Queue**

SFB410 is used to read a message from the receive queue.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port identifier
2.0	in	RxData	ANY		Pointer to data (up to 8 bytes)
12.0	out	MsgID	DWORD	DW#16#0	Identifier of received message
16.0	out	DLC	INT	0	Data Length Code (number of bytes in RxData)
18.0	out	NDR	BOOL	FALSE	New Data Ready and copied to RxData
18.1	out	Error	BOOL	FALSE	An error occurred
20.0	out	Status	WORD	W#16#0	Return value of SFB

The queue is implemented as a FIFO.

If a new message is received when the queue is full this message is discarded and lost. A one-time error is indicated with-in the next call to SFB 410.

**PortID**

Reserved, set to 0

**RxData**

Pointer to the memory where the received data bytes shall be copied to. The pointer must reference memory of at least 8 bytes, that is, an entire CAN message.

**MsgID**

This field contains the message ID of the received frame.

The format of the ID (i.e. standard or extended) is defined by the configuration previously applied with SFB412.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and are set to 0.

**DLC**

Data Length Code, number of received bytes in RxData buffer.

**NDR**

New Data Ready indicates that the data of one message has been copied from the queue to the RxData buffer.

**Error**

This bit is set to 1 if any error occurred during reception or execution of the SFB.

**Status**

The Status of a receive job can take the following values:

NDR	Error	Status (Hex)	Meaning
1	0	0000	New data copied
0	0	0000	Queue is empty
1	1	80C2	Queue full and an overrun has occurred
0	1	8186	Invalid parameter: invalid any pointer
0	1	8187	Invalid Port number
0	1	8188	No queue configured
0	1	80A0	Negative acknowledge from CAN controller
0	1	8330	Access to write-protected DB denied
0	1	8331	Access to write-protected DI denied

**Interrupt OBs**

For each queue an interrupt OB can be defined and enabled using SFB412. The receive interrupt OB is called when a message is received into an empty queue (i.e. the first message). The send interrupt OB is called when the queue becomes empty (i.e. the last message has been transmitted).

The local data of the interrupt OB contains the standard data as it is defined for any OB. Three parameters however are specific if the OB is used for CAN interrupts.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	temp	OB41_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), 1
1.0	temp	OB41_STRT_INF	BYTE		16#41/42/43/44 (Interrupt 1/2
2.0	temp	OB41_PRIORITY	BYTE		Priority of OB Execution
3.0	temp	OB41_OB_NUMBR	BYTE		41 (Organization block 41, OB
4.0	temp	OB41_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB41_IO_FLAG	BYTE		16#54 (input module), 16#55 (
6.0	temp	OB41_PARAM	WORD		Parameter of interrupt source
8.0	temp	OB41_OBJ_ID	INT		Object ID
10.0	temp	OB41_PORT_ID	INT		Port ID
12.0	temp	OB41_DATE_TIME	DATE_AND_TIME		Date and time OB41 started

**Param, Offset 6**

This field contains the configured parameter which can be used to distinguish the CAN from other interrupt sources if they use the same interrupt OB.

**ObjID, Offset 8**

Object ID of the message object that caused the interrupt.

**PortID, Offset 10**

Reserved, always set to 0

## 4.2.4 CAN Data Mapping

### Introduction

The CAN Data Mapping is configured only once, typically in OB100 at start-up, and then automatically performs the exchange of input and output data without any further user intervention.

When the PLC goes to STOP for whatever reason, the CAN Data Mapping is reset and data exchange is stopped. A new configuration using SFB413 is required to re-start the CAN Data Mapping, typically in the OB100.

### SFB 413: CFG\_IO, Config CAN Data Mapping

SFB413 serves to configure the mapping of Inputs/Outputs to CAN messages in the CAN Data Mapping and to define the ID range to be handled.

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	PortID	INT	0	Port ID
2.0	in	DBNr	INT	0	DB Number containing IO mapping
4.0	in	NbEntries	INT	0	Number of entries in DB
6.0	in	TxTime	INT	0	Time in ms when outputs are to be sent
8.0	in	MsgID	DWORD	DW#16#0	Message ID to be handled by IO-Manager
12.0	in	Mask	DWORD	DW#16#0	Mask to define ID range to be handled
16.0	in	IDE	BOOL	FALSE	0 = Standard, 1 = Extended ID
16.1	in	TxChange	BOOL	FALSE	1 = Send modified telegrams only
18.0	out	RetVal	WORD	W#16#0	Return value

#### **PortID**

Reserved, set to 0.

#### **DBNr**

Number of DB containing the configuration for the CAN Data Mapping.

#### **NbEntries**

Specifies the number of mapping entries given in the DB.

The CAN Data Mapping can be stopped by calling SFB413 with parameter NbEntries set to 0.

#### **TxTime**

The configured outputs are cyclically sent on the CAN bus in a defined interval. Tx-Time defines this interval in milliseconds.

#### **MsgID**

This specifies the Message ID to be handled by the reception buffer assigned to the CAN Data Mapping.

The Message ID does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining bits are reserved and should be set to 0.

#### **Mask**

The Mask can be used to extend the matching of received messages to several message IDs rather than the single one defined by the MsgID. Every bit of the mask that is set to 0 specifies the respective bit of the incoming message ID to be "don't care" for ID matching.

With the mask an ID range can be defined that must be handled by the CAN Data Mapping. The Mapping can also handle the entire ID range in which case the mask must be set to 0.

A message is accepted for reception into the CAN Data Mapping if the following condition is true:

(Received MsgID **XOR** Configured MsgID) **AND** Mask=0

The mask does always occupy the least significant bits for both standards (11 bits) and extended IDs (29 bits). The remaining upper bits are reserved and should be set to 0.

### **IDE**

Specifies the ID format of the messages:

0: Standard ID (11 bit, CAN2.0A)

1: Extended ID (29 bit, CAN2.0B)

### **TxChange**

If this bit is set to 1, only the telegrams that have been modified since the last transmission are transmitted.

### **RetVal**

Possible return values of the SFB are:

Return value (Hex)	Meaning
0000	SFB successfully executed
80A0	Negative acknowledge from CAN driver
80A1	Internal error
80C3	Not enough memory available (too many mapping entries)
8187	Invalid Port number
8189	Invalid configuration
8190	Invalid parameter (inconsistent DB size)
8192	Invalid virtual address (already occupied)
8F33	Error with DB number

## **Data mapping definition**

The SFB 413 must be given a DB containing the data mapping. The data mapping defines which PLC Process data are mapped to which CAN message ID and at what offset in the respective CAN message. Each BYTE, WORD or DWORD requires a separate entry in the DB. An entry is defined as this:

Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	MsgID	DWORD	DW#16#0	Identifier of message
+4.0	MsgOffset	INT	0	Offset in CAN message
+6.0	VA	INT	0	Virtual Address
+8.0	Size	INT	0	Byte = 1, Word = 2, DWord = 4
+10.0	IO	BOOL	FALSE	Input = 0, Output = 1
=12.0		END_STRUCT		

Up to 8 BYTES (resp. 4 WORDs or 2 DWORDS) can be assigned to a CAN message ID. If more than 8 bytes are mapped to the same ID a configuration error is indicated.

### **MsgID**

Identifier of CAN message to which PLC media shall be mapped.

### **MsgOffset**

Offset in the CAN message where the PLC media shall be mapped to.

### **VA**

Virtual Address of PLC input or output media.

**Size**

Size of mapped area, i.e. BYTE = 1, WORD = 2, DWORD = 4.  
Any number from 1 to 8 is possible.

**IO**

Flag to define if mapping of this entry is for input or output media, 0 = input, 1 = output.

**Configuration DB**

The configuration DB contains a header of 4 WORDs followed by an array of entries:

Adresse	Name	Typ	Anfangswert
0.0		STRUCT	
+0.0	DBVersion	INT	0
+2.0	Reserved1	INT	0
+4.0	Reserved2	INT	0
+6.0	Reserved2l	INT	0
+8.0	IOEntry	ARRAY[1..8]	
*12.0		"CanIOEntry"	
=104.0		END_STRUCT	

**DBVersion**

Version of the DB structure, set to 0.

**Reserved1, Reserved2, Reserved3**

Reserved for future use, set to 0.

**Example**

Adresse	Name	Typ	Anfangswert	Aktualwert	Kommentar
0.0	DBVersion	INT	0	0	
2.0	Reserved1	INT	0	0	
4.0	Reserved2	INT	0	0	
6.0	Reserved2l	INT	0	0	
8.0	IOEntry[1].MsgID	DWORD	DW#16#0	DW#16#318	Identifier of message
12.0	IOEntry[1].MsgOffset	INT	0	0	Offset in CAN message
14.0	IOEntry[1].VA	INT	0	10	Virtual Address
16.0	IOEntry[1].Size	INT	0	2	Byte = 1, Word = 2, DWord = 4
18.0	IOEntry[1].IO	BOOL	FALSE	FALSE	Input = 0, Output = 1
20.0	IOEntry[2].MsgID	DWORD	DW#16#0	DW#16#318	Identifier of message
24.0	IOEntry[2].MsgOffset	INT	0	2	Offset in CAN message
26.0	IOEntry[2].VA	INT	0	512	Virtual Address
28.0	IOEntry[2].Size	INT	0	2	Byte = 1, Word = 2, DWord = 4
30.0	IOEntry[2].IO	BOOL	FALSE	FALSE	Input = 0, Output = 1
32.0	IOEntry[3].MsgID	DWORD	DW#16#0	DW#16#318	Identifier of message
36.0	IOEntry[3].MsgOffset	INT	0	4	Offset in CAN message
38.0	IOEntry[3].VA	INT	0	35	Virtual Address
40.0	IOEntry[3].Size	INT	0	1	Byte = 1, Word = 2, DWord = 4
42.0	IOEntry[3].IO	BOOL	FALSE	FALSE	Input = 0, Output = 1

These three entries result in the following mapping of the CAN telegram with ID 0x318 to the PLC input media

CAN Message	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PLC Media	EB 10	EB11	EB 512	EB 513	EB 35	No Access		
	EW 10		EW 512					



### 4.3 Programming model (CAN Direct Access)

This chapter describes the programming model to be adapted by the application programmer for polling and interrupt operation with-in the CAN Direct Access mode.

#### 4.3.1 Transmission

##### Polling

- Call SFB401 with Act = 1 and evaluate Status for proper acceptance of the send job
- Call SFB401 with Act = 0 to request status until Done = 1
- The message object is now ready to accept the next job.

##### Interrupt

- Call SFB401 with Act = 1 and evaluate Status for proper acceptance of the send job
- Wait for call of the programmed OB
- Call SFB401 with Act = 0 and verify that Done = 1, which indicates that the message object is now ready to accept the next job.

##### RTR in polling mode

###### *Single transmission*

- Call SFB401 with Act = 1 and RTR = 1 and evaluate Status for proper acceptance of the send job
- Call SFB401 with Act = 0 to request status until Done = 1, in this case another station has requested the frame and it has successfully been sent
- The message object is now ready to accept the next job
- If the Done bit is never set a new job with a positive edge on Act can be initiated if desired.

###### *Multiple transmission*

- Call SFB401 with Act = 1, RTR = 1 and MultipleTx = 1 and evaluate Status for proper acceptance of the send job
- Call SFB401 with Act = 0 to request status. Each time that Done = 1, another station has requested the frame and it has successfully been sent
- A new job with a positive edge on Act can be initiated anytime.

##### RTR in interrupt mode

###### *Single transmission*

- Call SFB401 with Act = 1 and RTR = 1 and evaluate Status for proper acceptance of the send job
- Wait for call of the programmed OB which indicates that the frame has been requested by another station and has successfully been sent
- Call SFB401 with Act = 0 and verify that Done = 1, which indicates that the message object is now ready to accept the next job.

###### *Multiple transmission*

- Call SFB401 with Act = 1, RTR = 1 and MultipleTx = 1 and evaluate Status for proper acceptance of the send job
- Each time the programmed OB is called the frame has been requested by another station and has successfully been sent
- Optional: Call SFB401 with Act = 0 to clear the Done flag after each successful transmission
- A new job with a positive edge on Act can be initiated anytime.

### 4.3.2 Reception

#### Polling

- Call SFB400 with RxRelease = 1 and evaluate Error to ensure that buffer is released
- Call SFB400 until NDR = 1 and new data is copied to RxData
- If RxRelease = 1 in every call the buffer is directly released and ready to receive the next frame. If RxRelease = 0 the buffer will be locked until it is explicitly released again by calling SFB400 with RxRelease = 1
- Data in RxData buffer must now be processed or copied before next call to SFB400.

#### Interrupt

- Call SFB400 with RxRelease = 1 and evaluate Error to ensure that buffer is released
- Wait for call of the programmed OB which indicates that new data is received in the message object
- Call SFB400 to read data from message object
- If RxRelease = 1 the buffer is directly released and ready to receive the next frame. If RxRelease = 0 the buffer will be locked until it is explicitly released again by calling SFB400 with RxRelease = 1
- Data in RxData buffer must now be processed or copied before the next call to SFB400.

#### RTR in polling mode

- Call SFB400 with RxRelease = 1 and RTR = 1 and evaluate Error to ensure that buffer is released and a remote request is issued
- Call SFB400 with RxRelease = 0 until NDR = 1 and new data is copied to RxData
- Data in RxData buffer must now be processed or copied before the next call to SFB400
- A new job can now be initiated by calling SFB400 with RxRelease = 1.






#### RTR in interrupt mode

- Call SFB400 with RxRelease = 1 and RTR = 1 and evaluate Error to ensure that buffer is released and a remote request is issued
- Wait for call of the programmed OB which indicates that new data is received in the message object
- Call SFB400 to read data from message object
- If RxRelease = 1 the buffer is directly released and ready to receive the next frame. If RTR = 1 a new remote request is issued. If RxRelease = 0 the buffer will be locked until it is explicitly released again by calling SFB400 with RxRelease = 1
- Data in RxData buffer must now be processed or copied before the next call to SFB400.



## A Appendix

### A.1 Icons

	<p>In manuals, this symbol refers the reader to further information in this manual or other manuals or technical information documents.</p> <p>As a rule there is no direct link to such documents.</p>
	<p>This symbol warns the reader of the risk to components from electrostatic discharges caused by touch. This could happen, if cassettes have to be opened for changing jumpers like described for such cassettes in chapter 2.8 and 2.9.</p> <p><b>Recommendation:</b> at least touch the Minus of the system (cabinet of PGU connector) before coming in contact with the electronic parts. Better is to use a grounding wrist strap with its cable attached to the Minus of the system.</p>
	<p>This sign accompanies instructions that must always be followed.</p>
	<p>Explanations beside this sign are valid only for the Saia PCD® Classic serie.</p>
	<p>Explanations beside this sign are valid only for the Saia PCD® xx7 serie.</p>

**A.2 Contact**

**Saia-Burgess Controls AG**

Bahnhofstrasse 18  
3280 Murten  
Switzerland

Phone ..... +41 26 672 72 72

Fax..... +41 26 672 74 99

Email support: ..... [support@saia-pcd.com](mailto:support@saia-pcd.com)

Supportsite: ..... [www.sbc-support.com](http://www.sbc-support.com)

SBC site: ..... [www.saia-pcd.com](http://www.saia-pcd.com)

International Representatives &

SBC Sales Companies: ..... [www.saia-pcd.com/contact](http://www.saia-pcd.com/contact)

**Postal address for returns from customers of the Swiss Sales office**

**Saia-Burgess Controls AG**

Service Après-Vente  
Bahnhofstrasse 18  
3280 Murten  
Switzerland

