Manual

SAIA BURGESS CONTROLS

CAN-Functions on PCD2.M5_ & PCD3 CPU Serie Classic

Document 26-839 │ Edition ENG05 │ 2019-05-27

0 Content

0.1 Document History

Date	Version	Changes / Remarks
2005-12-15	EN01	Published Edition
2009-09-09	EN02	Modifications for PCD2.M5_
2013-09-30	EN03	New Logo and new company name
2014-06-20	EN04	3 types of PCD3 with CAN
2019-05-27	ENG05	New phone number: (2015-02-15)

0.2 About this manual

See the section in the appendix in relation to some of the terms, abbreviations and the references used in this manual.

0.3 Brands and trademarks

1 Introduction

This document gives an introduction and general description of the Saia PCD® with CAN interface, as well as the specific functions of the Classic 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.

There are three types of PCD3 with CAN-bus. The two types PCD3.M6240 and PCD3.M6340 are no longer fabricated. They were replaced with the PCD3.M6360, which is faster and has more storage capacity.

For the PCD2.M5x40 we offer the card PCD7.F7400.

These three PCD3s with CAN-fonction as well as the card PCD7.F2400 are OEM-products and appear neither in the system catalogue nor in the price list.

Overview of the PCD3-types with CAN-fonction:

		PCD3.M6240	PCD3.M6340	PCD3.M6360
Technical data CPUs		CAN		
Number of inputs/outputs or I/O module slots		1023 64		
Expansion connection		yes		
Processing time [µs]	■ bit operation ■ word operation	0.3…1.5 µs 0.9 µs		0.1…0.8 µs 0.3 µs
Integrated Web server + USB + Date-time (RTC)		yes		
On-board memory				
User memory (RAM)		1 MByte		2 MB prog. + 1 MB text/ DB (flash memory)
Backup memory flash		1 MByte (on board)		
Flash file system		-		16 MByte
Data backup		1…3 years with Lithium battery		
Optional memory		up to 4 GByte		
On-board data interfaces		4…5		
RS-485 on terminal block (Profibus-DP slave, Profi-S-Net (S-IO, S-Bus))		up to 115.2 kBit/s or Profi-S-Net up to 187.5 kBit/s		
USB 1.1		no	yes	
Ethernet-TCP/IP 10/100 MBit/s		yes		
RS-232 up to 115.2 kBit/s		yes (on D-Sub)		
RS-422/RS-485 on Port #3		no		
Profibus-DP slave, Profi-S-Net (S-IO, S-Bus), up to 1.5MBit/s		no		
Controller Area Network (CAN 2.0B)		yes (on D-Sub)		
Profibus-DP Master up to 12 MBit/s		no		
Optional data interfaces		Up to 8		
Optional PCD3.F1xxmodules for RS-232, RS-485, RS-422, TTY/20 mA and Belimo MP-Bus		only Slot #0		
Optional PCD3.F2xxmodules for RS-232, RS-485, RS-422, TTY/20 mA and Belimo MP-Bus		Slot #0…3 up to 8 ports		
General				
Supply voltage (according to EN/IEC61131-2)		24 VDC / -20/+25 % max.incl. 5 % ripple		
Capacity 5 V/24 V intern		max. 600 mA/100 mA		
Programmable		from PG5 Version $1.3.100		from PG5 Version $2.0.136

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.
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

Specification	***Message ID format***
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.

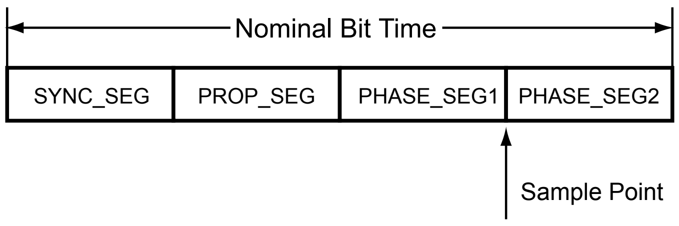
Priorisation and arbitration

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:



Segment	***Length***
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

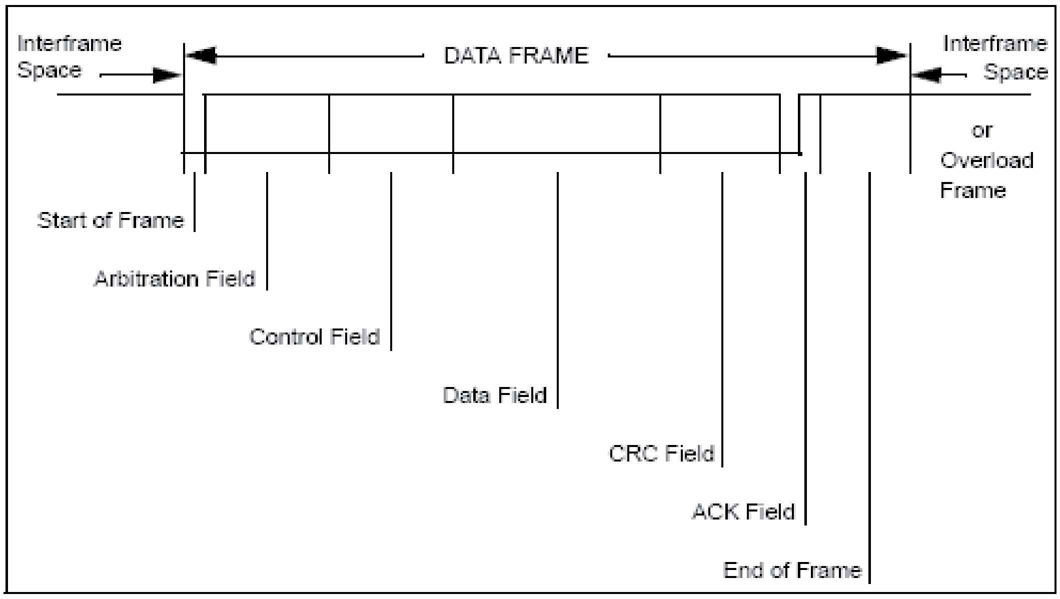
The sum of the propagation and phase segment 1 is commonly denoted as Time Segment 1 and is used in many CAN controllers for configuration of the bit timing. The propagation and phase segment 1 can then not be configured separately anymore. However, this is not required anyway since the sample point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

TIME_SEG1 = PROP_SEG + PHASE_SEG1

1.3 Frame types

Data frame

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.



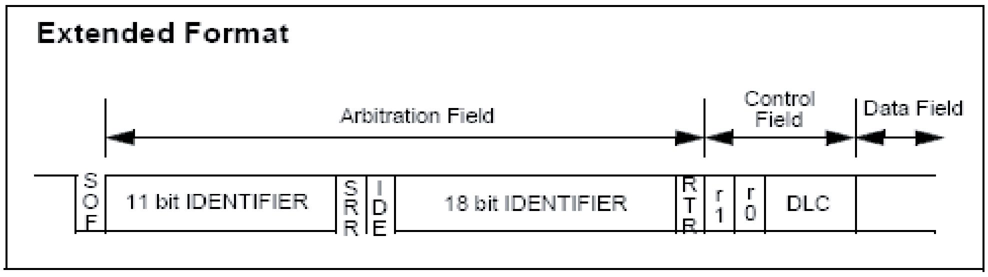
A data frame can be sent by any node at any time. The transmission is dependent on the arbitration process.

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

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

Mask	***Condition***
0x000	Any message ID is accepted for reception
0x7FF	Only the configured message ID is accepted for reception

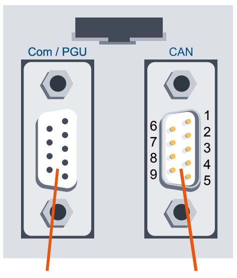
In the general case every bit of the mask that is set to 0 specifies the respective bit of the incoming message identifier to be «don't care» for identifier matching. The following example illustrates the message filtering in the general case for standard 11 bit identifiers:

	Hex	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Acceptance ID	0x137	0	0	1	0	0	1	1	0	1	1	1
Acceptance Mask	0x7EC	1	1	1	1	1	1	0	1	1	0	0
Received Identifiers		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 the PCD3.M6240, PCD3.M6340 and PCD3.M6360

		RS 232/PGU Port 0		Controller Area Network (CAN) CAN (9-pole D-Sub socket) Port 10		
		D-Sub pin	Signal	D-Sub pin	Signal	Explanation
 Com / PGU CAN 1 2 3 4 5 6 7 8 9		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. **Not connected on prototypes!**
		7	RTS	7	CAN_H*	Receive/transmit data positive
		8	CTS	8	nc	not connected
Port 0	Port 10	9	nc	9	nc	not connected

*) Galvanic isolated signals

For all types					
Terminal block for supply, watchdog, interrupt inputs and Port 2				Profibus signal	Profibus wiring
	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		
	4	Int1	1 fast counter		
	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.M6xxx is a CPU that provides a galvanic isolated high-speed CAN interface.

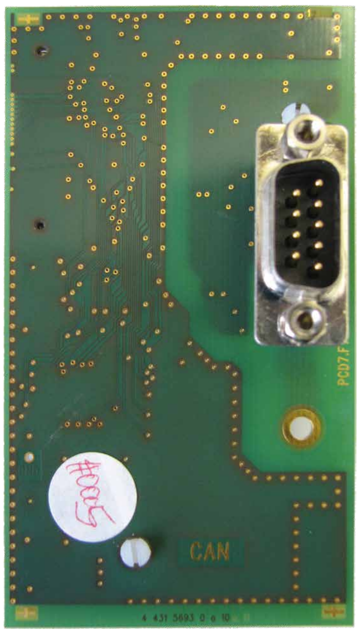
The CPU has the same functions as a PCD3.M554x except that the SBC S-Net/MPI interface is replaced by CAN functionality. This document only specifies differences. Please refer to existing PCD3.M554x documentation for common features.

2.1.2 CAN bus attachment, module PCD7.F7400 for the PCD2.M5x40

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.

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

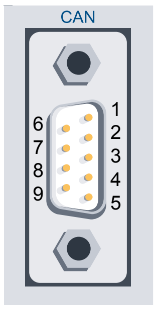
PCD7.F7400	**Controller Area Network (CAN)**		
	CAN (9-pole D-Sub male) on slot C Port 8		
	D-Sub pin	**Signal**	**Explanation**
	1	nc	not connected
	2	CAN_L*	Receive/transmit data negative
	3	GND*	0 V Data communication potent. CAN Ground
	4	nc	not connected
	5	nc	not connected
	6	GND*	0 V Data communication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

*) 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

CAN (Port 10)	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 **Not connected on prototype!**
	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 place 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

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 o ccurs 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.M6360	Port 10
PCD3.M6340	Port 10
PCD3.M6240	Port 10
PCD2.M5x40 with PCD7.F7400	Port 8

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.

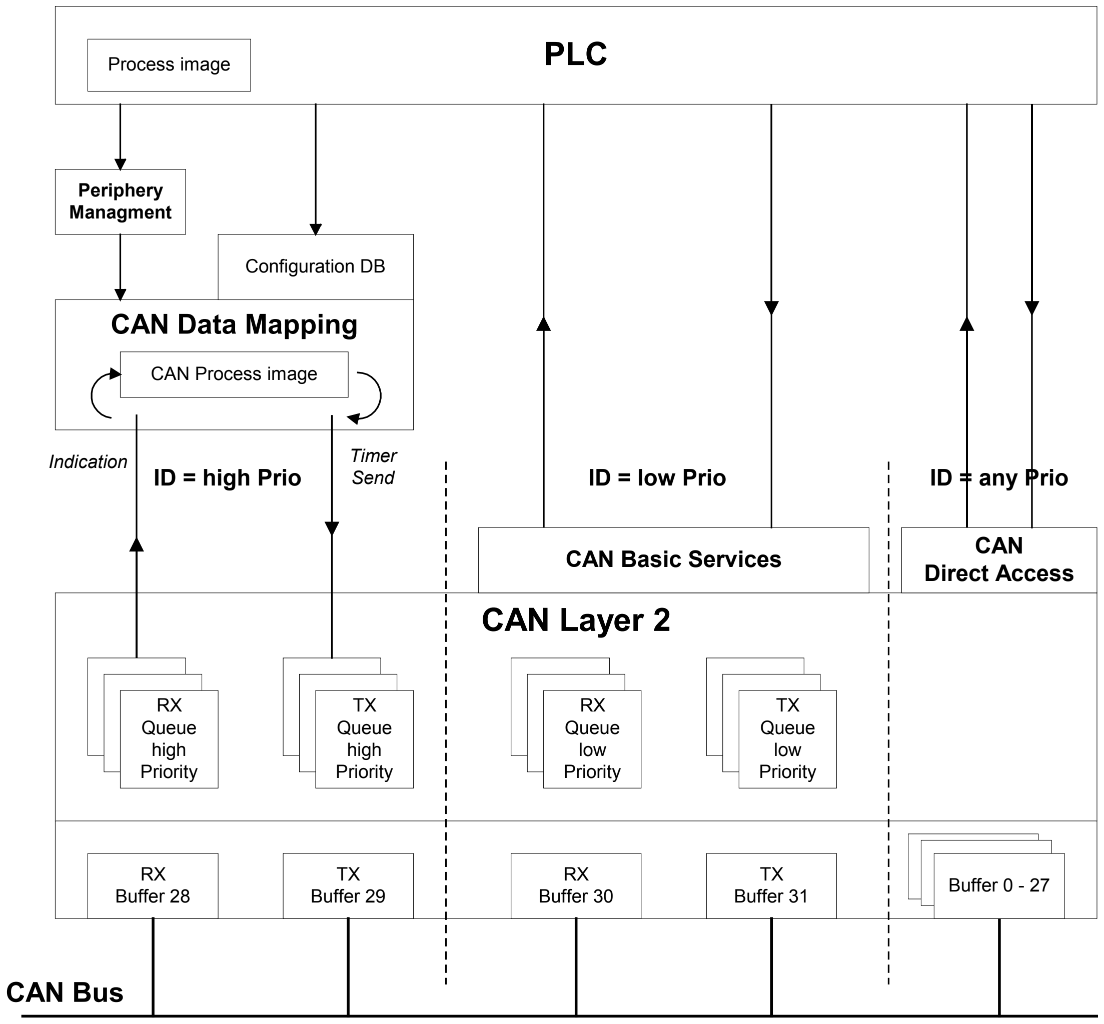
Operating modes

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 cont rollers 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 mes sage 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 igno red 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 priorization 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 priorization:

Priority	Reception	Message ID	Transmission	Message ID
1 (highest)	CAN Direct Access	any Priority	CAN Direct Access	any Priority
2	CAN Data Mapping	high Priority	CAN Data Mapping	high Priority
3 (lowest)	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 Classic CAN user interface provides several System Functions for data transfer and status. While some System Functions for configuration do also exist, the configuration is basically given in a DBX at startup (see next page).
All definitions required for programming the System Function calls are included in a Library .inc file. The file CANLib.inc must be included in every source file where the CAN CSF are used.

General functions

S.CAN.Config, Configure CAN-Driver
S.CAN.Status, Request current status of CAN-Driver

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

CAN Direct Access (FullCAN)

S.CAN.CfgObj, Configure a single Message object
S.CAN.Tx, Send message
S.CAN.Rx, Receive message

These functions give direct and flexible access to the hardware CAN controller.

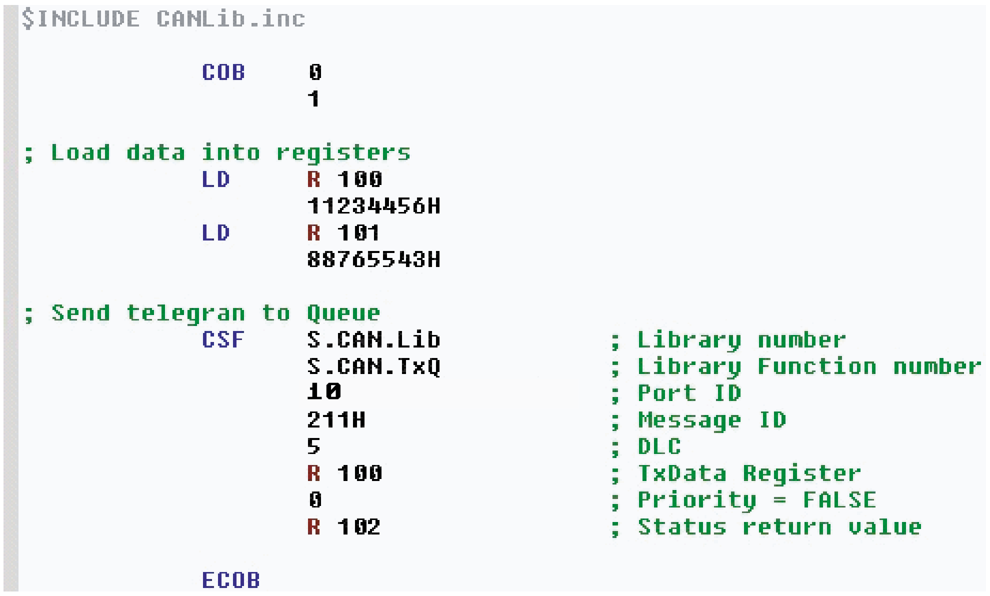
CAN Basic Services (BasicCAN)

S.CAN.CfgQ, Configure Queues
S.CAN.TxQ, Send message to Queue
S.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.

Most of the parameters for the System Functions can be Registers or Constants. Keep in mind that **Constants are only 14 bits wide**. Hence for any values higher than 16383 the use of Registers is mandatory, since Constants cannot hold those values. If the ‘K’ is not used in calls to System Functions, Constants can also be 16 bits wide. In this case values up to 65535 are possible.

The following image shows an example of a call to a System Function:



```
$INCLUDE CANLib.inc

            COB     0
                    1

; Load data into registers
            LD      R 100
                    11234456H
            LD      R 101
                    88765543H

; Send telegran to Queue
            CSF     S.CAN.Lib               ; Library number
                    S.CAN.TxQ               ; Library Function number
                    10                      ; Port ID
                    211H                    ; Message ID
                    5                       ; DLC
                    R 100                   ; TxData Register
                    0                       ; Priority = FALSE
                    R 102                   ; Status return value

            ECOB
```

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 Media are given as parameters 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 in a DBX at start-up of the system. This DBX can be edited manually by the user or it can be generated using the CAN Configuration Utility.
Additionally the following System Functions are provided for some selected configuration parts. These can be used instead of the DBX or at run-time to change the configuration previously applied with the DBX or with the same System Functions.

S.CAN.Config, Configure CAN-Driver
S.CAN.CfgObj, Configure a single Message object
S.CAN.CfgQ, Configure Queues

4.1.3 Initialisation sequence

If a DBX is used, no initialisation sequence must be respected since this is all integrated in the firmware. However if the System Functions are used for configuration the following initialisation sequences must be followed for each operating mode.

CAN Direct Access
1) Configure the driver with the basic parameter set using ***S.CAN.Config***
2) Configure each message object using ***S.CAN.CfgObj***
3) Release all reception buffers by calling ***S.CAN.Rx*** for each receive object to be released

CAN Basic Services
1) Configure the driver with the basic parameter set using ***S.CAN.Config***
2) Configure the receive and transmit queue using ***S.CAN.CfgQ***

CAN Data Mapping
The CAN Data Mapping cannot be configured by using a System Function

4.1.4 Overview operating options

The following table gives an overview of the different usage options for configuration and operation of the different operating modes.

		Configuration		Operation		
Mode	**Option**	**CSF**	**Cfg Util**	**CSF**	**Saia PG5® FBox**	**Automatic**
Direct Access	1	X		X		
	2		X	X		
Basic Services	1	X		X		
	2		X	X		
	3	X			X	
	4		X		X	
Data Mapping	1		X			X

4.1.5 PLC States

RUN
Any configured CAN operating mode is running and enabled in RUN state

STOP
Any configured CAN operating mode will continue running in STOP state as in RUN state.

HALT
Every CAN operating mode is reset and disabled in HALT state.

4.2 User interface

4.2.1 General

S.CAN.Config

S.CAN.Config	
Function	Serves to set the basic parameters for the CAN driver. By calling this System Function all previous settings and object configurations are deleted and the driver and message objects are reset to their initial state. Typically this CSF is only called once in XOB 16.
Number of parameters	8

Parameters

No	Name	Direction	Media	Comment
1	PortID	in	R, K	Set to (see chapt. 2.3)
2	TimeSeg1	in	R, K	Refer to tables below
3	Baudrate	in	R, K	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
4	RxXOB	in	R, K	Number 32 – 63, Default 32
5	RxIntPrm	in	R, K	Reserved
6	TxXOB	in	R, K	Number 32 – 63, Default 33
7	TxIntPrm	in	R, K	Reserved
8	RetVal	out	R	Return value of system function call

Return values			**Error Flag**
	0	System function call successfully executed	false
	-1	Negative acknowledge from CAN controller	true
	-4	Invalid Port number	true
	-8	Invalid parameter: invalid TimeSeg1 or OB number	true
	-9	Invalid baudrate	true

Two different global interrupt XOBs are defined, one for reception and one for transmission.

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	Propaga-tion Seg-ment	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	Propaga-tion Seg-ment	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

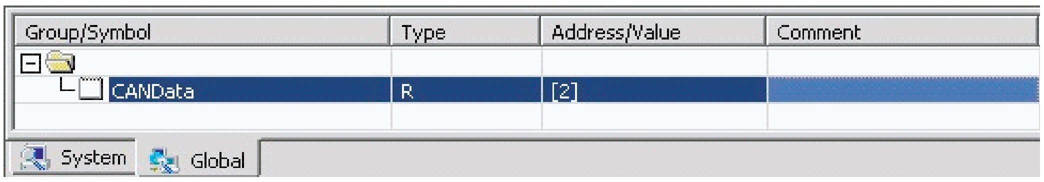
S.CAN.Status

S.CAN.Status					
Function			The meaning and rules for the different status indicators are defined in the CAN2.0 norm.		
Number of parameters			5		
Parameters					
No	**Name**	**Direction**	**Media**	**Comment**	
1	PortID	in	R, K	Set to (see chapt. 2.3)	
2	RxErrCtr	out	R	Receive error counter	
3	TxErrCtr	out	R	Transmit error counter	
4	Status	out	R	Bit 0: BusOff Bit 1: Error, any error pending Bit 2: Warning, any counter >= 96 Bit 3: ErrorState, 0: Error active, 1: Error passive	
5	RetVal	out	R	Return value of system function call	
Return values					**Error Flag**
	0	System Function successfully executed			false
	-4	Invalid Port number			true
	-8	Invalid parameter			true

4.2.2 Data Registers

For the Receive and Transmit CSF in both, Direct Access and Basic Services operating modes, two consecutive registers are defined to accommodate the CAN data. It is always the first of the two registers that is given as parameter to the CSF.

In order to ensure that the second register is not used for any other purpose, the CAN data registers should always be defined as an array of 2 when using symbols as shown in the image below.



Group/Symbol	Type	Address/Value	Comment
CANData	R	[2]	

The following example shows how the bytes are ordered in the two registers for every possible number of bytes. Byte 0 is the first and byte 7 is the last one in the CAN telegram on the bus (this is the byte order on the bus and has no relation with MSB/LSB of WORD or DWORD values).

Number of bytes	**Byte order**							
	Reg n+0				Reg n+1			
1				0				
2			0	1				
3		0	1	2				
4	0	1	2	3				
5	0	1	2	3				4
6	0	1	2	3			4	5
7	0	1	2	3		4	5	6
8	0	1	2	3	4	5	6	7

4.2.3 CAN Direct Access

S.CAN.CfgObj, Config Object

S.CAN.CfgObj				
Function			Serves for configuration of a single message object. Typically this CSF is only called once for each message object in XOB 16. It can also be used to configure or reconfigure any object at a later stage at run-time	
Number of parameters			7	
Parameters				
No	**Name**	**Direction**	**Media**	**Comment**
1	PortID	in	R, K	Set to (see chapt. 2.3)
2	ObjID	in	R, K	Number of message buffer (object): 0-31
3	MsgID	in	R, K	This specifies the Message ID to be handled by this message object. 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	Mask	in	R, K	Mask can be used to extend the matching of received messages to several message IDs. 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.
5	Flags	in	R, K	Bit 0: IDE Bit 1: Dir Bit 2: Enable XOB
6	Timeout	in	R, K	Specifies the Timeout in milliseconds after which a send job will be aborted. To deactivate the timeout functionality Timeout must be set to 0. ***Important:*** No interrupt is generated if a timeout expires. This functionality only works in polling mode.
7	RetVal	out	R	Return value of system function call
Return values				**Error Flag**
	0	System Function successfully executed		false
	-1	Negative acknowledge from CAN controller		true
	-2	Access to message object denied		true
	-4	Invalid Port number		true
	-10	Invalid ID		true

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 System Function S.CAN.Tx Send Message if it specifies a different Message ID.

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

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

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

Flags:Enable XOB
Enables the call of the respective interrupt XOB after a successful transfer if set to 1.

S.CAN.Tx, Send Message

S.CAN.Tx				
Function			used for sending messages on the CAN bus and also for requesting the status in polling mode	
Number of parameters			7	
Parameters				
No	**Name**	**Direction**	**Media**	**Comment**
1	PortID	in	R, K	Set to (see chapt. 2.3)
2	ObjID	in	R, K	Number of message buffer (object)
3	MsgID	in	R, K	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	in	R, K	Data Length Code (number of bytes)
5	TxData	in	R	First of two consecutive registers containing the CAN telegram data, refer chap. 4.2.2
6	Cmd	in	R, K	0: Request status 4: Send telegram 6: Send telegram as response to RTR (once) 7: Send telegram as response to RTR (multiple) 8: Abort send job
7	Status	out	R	Return value of system function call

Status values			**Error Flag**
	0	Job successfully terminated	false
	1	No job pending	false
	2	Job is accepted and being executed	false
	3	Job has been aborted	false
	-1	Negative acknowledge from CAN controller	true
	-2	Access to message object denied	true
	-4	Invalid Port number	true
	-10	Invalid parameter ID	true
	-11	Invalid length parameter DLC	true
	-16	Too many jobs pending	false
	-17	Timeout has expired	false

Command symbols (Parameter No. 6)		
0	Request status	S.CAN.TxCmd.TX_STAT
4	Send telegram	S.CAN.TxCmd.TX
6	Send telegram as response to RTR (once)	S.CAN.TxCmd.TX_RTR
7	Send telegram as response to RTR (multiple)	S.CAN.TxCmd.TX_RTR_MULT
8	Abort send job	S.CAN.TxCmd.TX_ABORT

Only one send job can be processed by a message object at a time. 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.
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.
A message object can only be used for transmission if it has been configured as such.

S.CAN.Rx, Receive Message

S.CAN.Rx				
Function	Used for receiving messages on the CAN bus. The same System Function 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 CSF must be called to get the received message.			
Number of parameters	7			
Parameters				
No	**Name**	**Direction**	**Media**	**Comment**
1	PortID	in	R, K	Set to (see chapt. 2.3)
2	ObjID	in	R, K	Number of message buffer (object)
3	MsgID	out	R	ID of the received message 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 should be set to 0.
4	DLC	out	R	Data Length Code (number of bytes in RxData)
5	RxData	out	R	First of two consecutive registers to receive the CAN telegram data Always two registers must be reserved even if only 4 or less bytes are expected Bytes that do not contain valid CAN data according to DLC are set to 0 by the CSF
6	Cmd	in	R, K	0: Request status and data if ready 1: Release message buffer for reception 3: Issue a remote request
7	Status	out	R	Return value of system function call
Status values				**Error Flag**
	0	New data copied to RxData, this value will only be returned for 1 single call to the System Function for 1 specific receive frame		false
	1	No data has been received yet		false
	4	Overrun, new data was received into a full buffer, new data has been copied to registers		false
	-1	Negative acknowledge from CAN controller		true
	-2	Access to message object denied		true
	-4	Invalid Port number		true
	-10	Invalid parameter ID		true

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

Cmd = 0, S.CAN.RxCmd.RX_STAT
With Cmd = 0 only the status of the message object in question is returned. The message object is not released for reception (unless it is already released in which case it will remain released and receive the next matching message)

Cmd = 1, S.CAN.RxCmd.RX_RLS
By setting Cmd to 1 the message object is directly released for reception of a next frame after the data has been copied to the RxData registers. Additionally if a receive

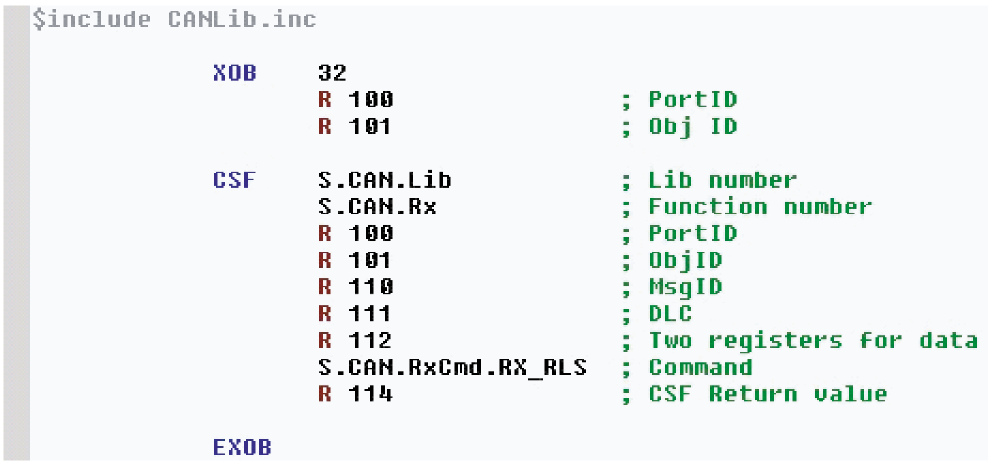
message object has not been released yet, a call to this System Function with Cmd = 1 is required to enable reception for this message object.
A receive message object cannot be deactivated by setting Cmd = 0 after it has been released. It can only be not released after a successful reception of a message. Likewise an already released message object cannot be released with different settings until a successful reception of a message.

***Cmd* = 3,** S.CAN.RxCmd.RX_RTR
If Cmd is set to 3, a remote request will be issued by the CAN controller and the response frame will be received into this message object.
An RTR receive buffer released with Cmd = 3 can be overwritten and released with new settings. Therefore Cmd must always be set to 0 while polling an RTR receive buffer (released with Cmd = 3), otherwise a remote request will be issued with each call to the System Function.

Interrupt XOBs
Two global interrupt XOB are defined for Direct Access operating mode. The call to the respective XOB can be enabled individually for each message object. The XOB is then called if a message has successfully been transmitted or if new data has been received.
The following XOBs are defined:

Default number	Usage
XOB 32	CAN Direct Access Reception Interrupt
XOB 33	CAN Direct Access Transmission Interrupt

Two parameters are required in the XOB in order to know which object on which port has caused the interrupt.
The XOB statement allows copying parameters to PLC Media. The parameters are copied to the given media before execution of the XOB.
The statement looks like this for the CAN Receive XOB 32:



```
$include CANLib.inc

            XOB     32
                    R 100                   ; PortID
                    R 101                   ; Obj ID

            CSF     S.CAN.Lib               ; Lib number
                    S.CAN.Rx                ; Function number
                    R 100                   ; PortID
                    R 101                   ; ObjID
                    R 110                   ; MsgID
                    R 111                   ; DLC
                    R 112                   ; Two registers for data
                    S.CAN.RxCmd.RX_RLS      ; Command
                    R 114                   ; CSF Return value

            EXOB
```

4.2.4 CAN Basic Services

S.CAN.CfgQ, Config Queues

S.CAN.CfgQ				
Function			Serves to configure and activate the queues. Two queues are installed, one for reception and one for transmission. Typically this CSF is only called once at start-up in XOB 16.	
Number of parameters			11	
Parameters				
No	**Name**	**Direction**	**Media**	**Comment**
1	PortID	in	R, K	Set to (see chapt. 2.3)
2	MsgID	in	R, K	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.
3	Mask	in	R, K	Mask can be used to extend the matching of received messages to several message IDs. 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.
4	RxQDepth	in	R, K	The depth of the receive queue specifies how many messages can be hold by the queue.
5	RxXOB	in	R, K	Number 32 – 63, Default 34
6	RxIntPrm	in	R, K	Reserved
7	TxQDepth	in	R, K	The depth of the send queue specifies how many messages can be hold by the queue.
8	TxXOB	in	R, K	Number 32 – 63, Default 35
9	TxIntPrm	in	R, K	Reserved
10	Flags	in	R, K	Bit 0: IDE, 0=Standard (11 Bit), 1=Extended (29 Bit) Bit 1: Enable Rx XOB Bit 2: Enable Tx XOB
11	RetVal	out	R	Return value of system function call

Return values			**Error Flag**
	0	System Function successfully executed	false
	-1	Negative acknowledge from CAN controller	true
	-4	Invalid Port number	true
	-8	Invalid paramtere	true
	-18	Not enough memory available	true

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

S.CAN.TxQ, Send Queue

S.CAN.TxQ					
Function			used to insert a message into the send queue		
Number of parameters			6		
Parameters					
No	**Name**	**Direction**	**Media**	**Comment**	
1	PortID	in	R, K	Set to (see chapt. 2.3)	
2	MsgID	in	R, K	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.	
3	DLC	in	R, K	Data Length Code (number of bytes)	
4	TxData	in	R	First of two consecutive registers containing the CAN telegram data	
5	Priority	in	F, K	insert into queue according to the priority of ID	
6	Status	out	R	Return value of system function call	
Status values					**Error Flag**
	0	Message successfully inserted in queue			false
	-1	Negative acknowledge from CAN controller			true
	-4	Invalid Port number			true
	-11	Invalid length parameter DLC			true
	-12	No queue configured			true
	-16	Queue is full, message is ignored			false

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.

Priority
When this flag 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 to 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 the higher priority. Hence the inserted element can at best be inserted in the second position of the queue

S.CAN.RxQ, Receive Queue

S.CAN.RxQ	
Function	used to read a message from the receive queue
Number of parameters	5

Parameters				
No	**Name**	**Direction**	**Media**	**Comment**
1	PortID	in	R, K	Set to (see chapt. 2.3)
2	MsgID	out	R	ID of the received message 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.
3	DLC	out	R	Data Length Code (number of bytes in RxData)
4	RxData	out	R	First of two consecutive registers to receive the CAN telegram data, refer to chap. 4.2.2. Always two registers must be reserved even if only 4 or less bytes are expected Bytes that do not contain valid CAN data according to DLC are set to 0 by the CSF
5	Status	out	R	Return value of system function call

Status values			**Error Flag**
	0	New data copied to RxData	false
	1	Queue is empty	false
	4	Queue full and an overrun has occurred, new data has been read	false
	-1	Negative acknowledge from CAN controller	true
	-4	Invalid Port number	true
	-12	No queue configured	true

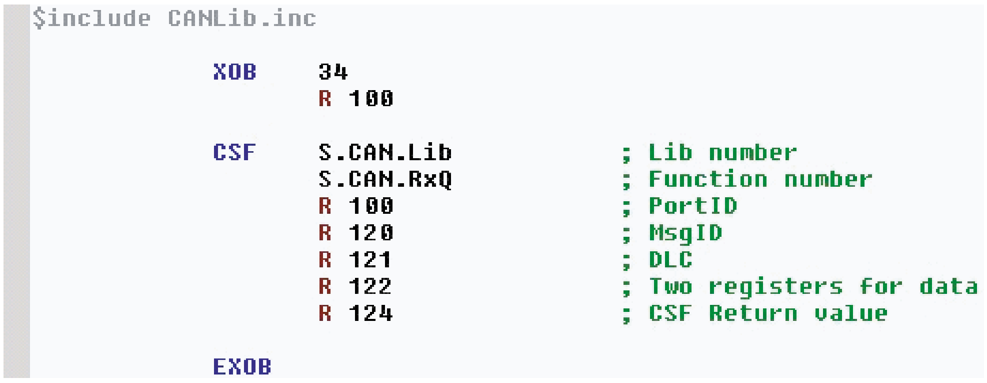
The queue is implemented as a FIFO.
If a new message is received when the queue is full this message is discarded and lost. A one-time error is indicated with-in the next call to the System Function.

Interrupt XOBs

Two interrupt XOBs are defined for Basic Services operating mode. Both of them can be enabled individually. The receive XOB is called when a message is received into an empty queue (i.e. the first message). The send XOB is called when the queue becomes empty (i.e. the last message has been transmitted).
The following XOBs are defined:

Default number	**Usage**
XOB 34	CAN Basic Services Reception Interrupt
XOB 35	CAN Basic Services Transmission Interrupt

The Queue XOB works the same way as for Direct Access. Simply there is no Object ID required. The following image shows an example:



```
$include CANLib.inc

            XOB     34
                    R 100

            CSF     S.CAN.Lib           ; Lib number
                    S.CAN.RxQ           ; Function number
                    R 100               ; PortID
                    R 120               ; MsgID
                    R 121               ; DLC
                    R 122               ; Two registers for data
                    R 124               ; CSF Return value

            EXOB
```

4.2.5 CAN Data Mapping

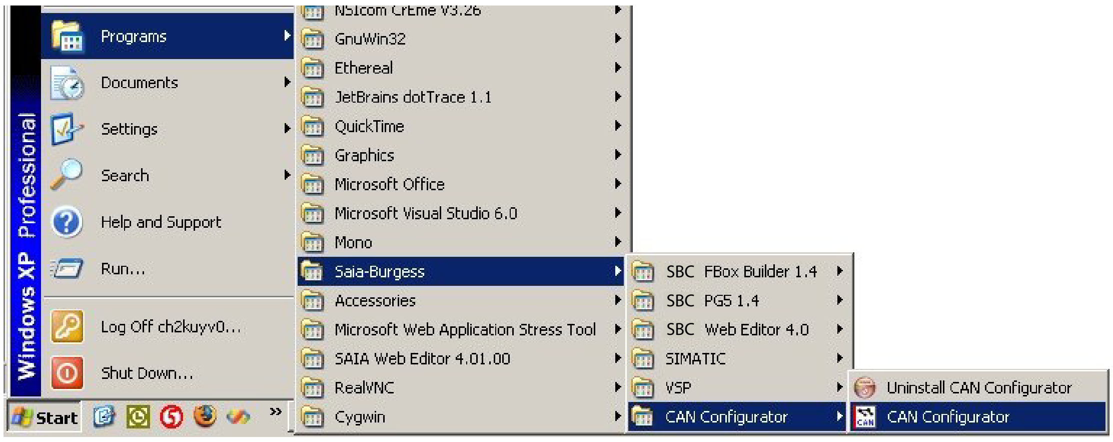
The CAN Mapping does not require any further user intervention after configuration.

4.3 Configuration

4.3.1 Starting the Configurator

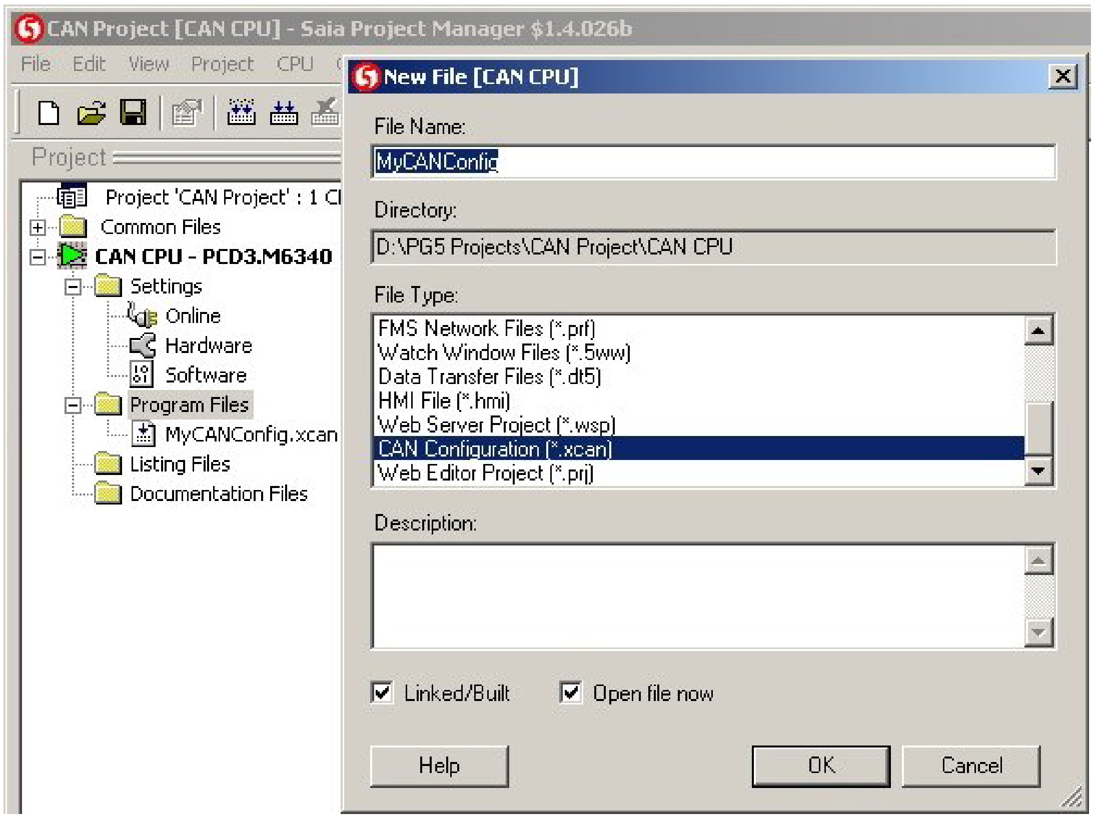
Standalone version

Starting the standalone version of the CAN Configurator is straightforward. If it is installed into the default start menu location then it will be found it in the start menu as depicted in the screenshot below.



Saia PG5® add-on tool version

This step requires that the CAN Configurator will be installed as a PG5 add-on tool. To use the CAN Configurator a Saia PG5® project has to be created that contains a CAN enabled CPU. A new file of the type CAN Configuration has to be added to it. This will open the CAN Configurator automatically. To modify the configuration on a later point in time simply open the *.xcan file.



File Name: MyCANConfig

Directory: D:\PG5 Projects\CAN Project\CAN CPU

File Type:
- FMS Network Files (*.prf)
- Watch Window Files (*.5ww)
- Data Transfer Files (*.dt5)
- HMI File (*.hmi)
- Web Server Project (*.wsp)
- CAN Configuration (*.xcan)
- Web Editor Project (*.prj)

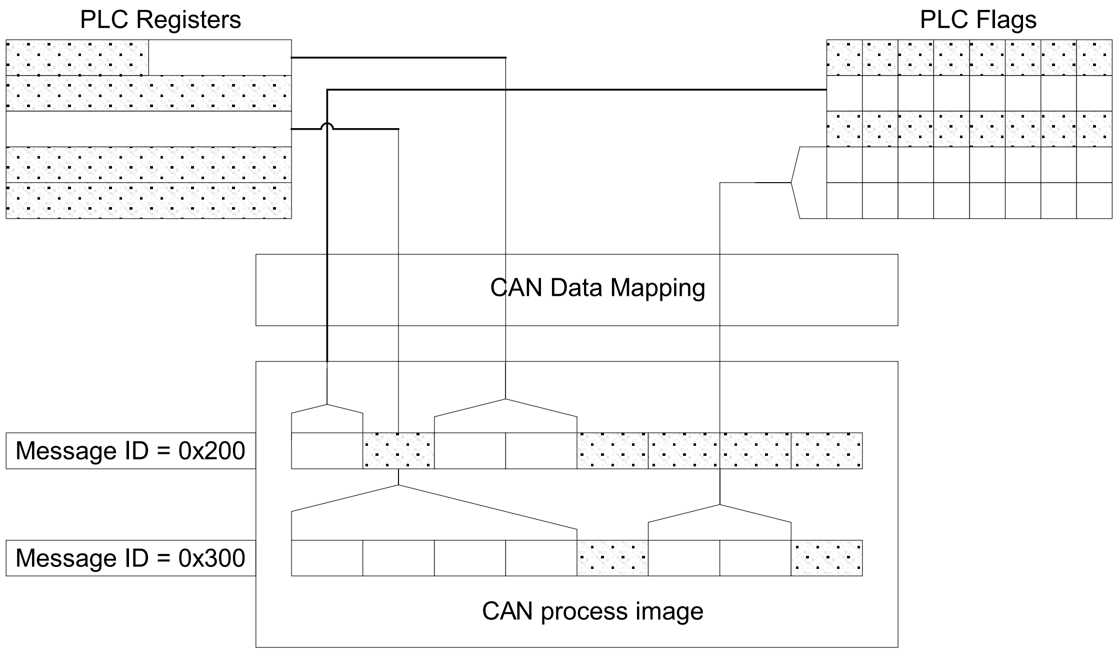
Description:

[✓] Linked/Built [✓] Open file now

Each CPU can have only one CAN configuration file. If the chosen CPU type in the hardware settings does not provide CAN support, it isn‘t possible to build the project if it contains a CAN configuration file.

4.3.2 Data Mapping

A single CAN message consists of up to 8 bytes data. The CAN Data Mapping has a reserved memory space called the CAN process image containing 8-byte-data CAN messages. Even if not all bytes of a CAN message are mapped to PLC media, the reserved space for this message is 8 bytes in the CAN process image anyway.
Any byte or word in the CAN message can be mapped to PLC Registers or Flags for both inputs and outputs by defining the respective transfer functions.
A CAN message is identified by its message ID. One CAN message ID is associated to one single telegram per transfer direction. Any 29-bit ID is possible.
The following picture shows an example of a mapping. The picture applies to inputs and outputs; it is just the transfer direction of the data that is reversed.
Each of the four transfer definitions shown below will result in a separate mapping entry.



4.4 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.

4.4.1 Transmission

Polling

- Call S.CAN.Tx with Cmd = 4 and evaluate Status for proper acceptance of the send job
- Call S.CAN.Tx with Cmd = 0 to request status until Status = 0
- The message object is now ready to accept the next job

Interrupt

- Call S.CAN.Tx with Cmd = 4 and evaluate Status for proper acceptance of the send job
- Wait for call of the programmed XOB
- Optional: Call S.CAN.Tx with Cmd = 0 and verify that Status = 0, which indicates that the message object is now ready to accept the next job.

RTR in polling mode

Single transmission

- Call S.CAN.Tx with Cmd = 6 and evaluate Status for proper acceptance of the send job
- Call S.CAN.Tx with Cmd = 0 to request status until Status = 0, 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 Status is never set to 0, a new job Cmd != 0 can be initiated if desired.

Multiple transmission

- Call S.CAN.Tx with Cmd = 7 and evaluate Status for proper acceptance of the send job
- Call S.CAN.Tx with Cmd = 0 to request status. Each time that Status = 0, an other station has requested the frame and it has successfully been sent
- A new job with Cmd != 0 can be initiated anytime

RTR in interrupt mode

Single transmission

- Call S.CAN.Tx with Cmd = 6 and evaluate Status for proper acceptance of the send job
- Wait for call of the programmed XOB which indicates that the frame has been requested by another station and has successfully been sent
- Optional: Call S.CAN.Tx with Cmd = 0 and verify that Status = 0, which indicates that the message object is now ready to accept the next job.

Multiple transmission

- Call S.CAN.Tx with Cmd = 7 and evaluate Status for proper acceptance of the send job
- Each time the programmed XOB is called the frame has been requested by another station and has successfully been sent
- Optional: Call S.CAN.Tx with Cmd = 0 to clear the Done Status after each successful transmission
- A new job with Cmd != 0 can be initiated anytime

4.4.2 Reception

Polling

- Call S.CAN.Rx with Cmd = 1 and evaluate Status to ensure that buffer is released
- Call S.CAN.Rx until Status = 0 and new data is copied to RxData
- If Cmd = 1 in every call the buffer is directly released and ready to receive the next frame. If Cmd = 0 the buffer will be locked until it is explicitly released again by calling S.CAN.Rx with Cmd = 1
- Data in RxData registers must now be processed or copied before next call to S.CAN.Rx with the same registers as parameters.

Interrupt

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

RTR in polling mode

- Call S.CAN.Rx with Cmd = 3 and evaluate Status to ensure that buffer is released and a remote request is issued
- Call S.CAN.Rx with Cmd = 0 until Status = 0 and new data is copied to RxData
- Data in RxData registers must now be processed or copied before next call to S.CAN.Rx
- A new job can now be initiated by calling S.CAN.Rx with Cmd = 3

RTR in interrupt mode

- Call S.CAN.Rx with Cmd = 3 and evaluate Status to ensure that buffer is released and a remote request is issued
- Wait for call of the programmed XOB which indicates that new data is received in the message object
- Call S.CAN.Rx to read data from message object
- If Cmd = 1 the buffer is directly released and ready to receive the next frame. If Cmd = 3 a new remote request is issued. If Cmd = 0 the buffer will be locked until it is explicitly released again by calling S.CAN.Rx with Cmd = 1
- Data in RxData registers must now be processed or copied before next call to S.CAN.Rx.

A Appendix

A.1 Icons

ℹ	In manuals, this symbol refers the reader to further information in this manual or other manuals or technical information documents. As a rule there is no direct link to such documents.
⚡	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. **Recommendation:** 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.
!	This sign accompanies instructions that must always be followed.
Classic	Explanations beside this sign are valid only for the Saia PCD® Classic serie.
xx7	Explanations beside this sign are valid only for the Saia PCD® xx7 serie.

A.2 Contact

Saia-Burgess Controls AG
Bahnhofstrasse 18
3280 Murten
Switzerland

Phone	+41 26 580 30 00
Fax..	+41 26 580 34 99

Email support:	support@saia-pcd.com
Supportsite:	www.sbc-support.com
SBC site:	www.saia-pcd.com
International Represetatives & SBC Sales Companies:	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