BAUMÜLLER

Option Board CAN-Master for Ωmega Drive-Line II CAN-M-01

Technical Description
and Operating Instructions

Edition: July 2001

E	5.00031.02

BAUMÜLLER

ΩMEGA CAN-MASTER OPTION BOARD FOR DRIVE-LINE II

Technical Description and Operating Instructions

Edition: July, 1st 2001

Document no. 5.00031.02

This operation manual is intended as a complement to the technical description and the operation manual of the apparatus.

BEFORE CARRYING OUT COMMISSIONING, CAREFULLY READ AND OBSERVE THE OPERATING INSTRUCTIONS AND SAFETY INFORMATION

This document contains all the information necessary to correctly use the products it describes. It is intended for specially trained, technically qualified personnel who are well-versed in all warnings and commissioning activities.

The equipment is manufactured using state-of-the-art technology and is safe in operation. It can safely be installed and commissioned and functions without problems if the safety information is followed.

You may not carry out commissioning until it has been established that the machine into which this component is to be installed complies with the specifications of the EC machine guidelines.

This technical description/these operating instructions invalidate all previous descriptions of the corresponding product. Within the scope of further development of our products, Baumüller GmbH reserves the right to change their technical data and handling.

Manufacturer and Supplier's Address:
Baumüller Nürnberg GmbH
Ostendstr. 80
D-90482 Nürnberg
Tel. ++49 (0)9 11/54 32 - 0
Fax ++49 (0)9 11/54 32 - 1 30

Country of Origin: Germany

Date of Manufacture: Determined from the serial number on the equipment

TABLE OF CONTENTS

1 SAFETY INFORMATION

General Information

These operating instructions contain all the information necessary for correct operation of the products described. The document is intended for specially trained, technically qualified personnel who are well-versed in all warnings and commissioning activities.

The equipment is manufactured using state-of-the-art technology and is safe in operation. It can safely be installed and commissioned and functions without problems if the safety information in these operating instructions is followed.

Danger Information

On the one hand, the information below is for your own personal safety and on the other to prevent damage to the described products or to other connected equipment.

In the context of the operating instructions and the information on the products themselves, the terms used have the following meanings:

DANGER

This means that **death, severe personal injury**, or **damage to property will** occur unless appropriate safety measures are taken.

WARNING

This means that **death, severe personal injury**, or damage to property **may** occur unless appropriate safety measures are taken.

NOTE

This draws your attention to **important information** about the product, handling of the product or to a particular section of the documentation.

Qualified Personnel

In the context of the safety-specific information in this document or on the products themselves, qualified personnel are considered to be persons who are familiar with setting up, assembling, commissioning and operating the product and who have qualifications appropriate to their activities:

- Trained or instructed or authorized to commission, ground and mark circuits and equipment in acordance with recognized safety standards.
- Trained or instructed in accordance with recognized safety standards in the care and use of appropriate safety equipment.

Appropriate Use

WARNING

You may only use the equipment/system for the purposes specified in the operating instructions and in conjunction with the third-party equipment and components recommended or authorized by BAUMÜLLER NÜRNBERG GmbH.

For safety reasons, you must not change or add components on/to the equipment/system.

The machine minder must report immediately any changes that occur which adversely affect the safety of the equipment/system.

2 TECHNICAL DATA

2.1 General

A CANsync-Master node is integrated in an Ωmega Drive-Line II. To use the CAN, you must provide the additional CAN-M-01 CAN-Master module.

The CAN-M-01 option board makes possible

- communication with up to 32 CAN nodes
  - of which a maximum of 16 I/O modules at 125 kbps
  - of which a maximum of 8 I/O modules at 250 kbps
  - of which a maximum of 16 controllers with a CAN interface
  - of which a maximum of 16 Ωmegas with a CAN interface
  - of which a maximum of 16 absolute value encoders

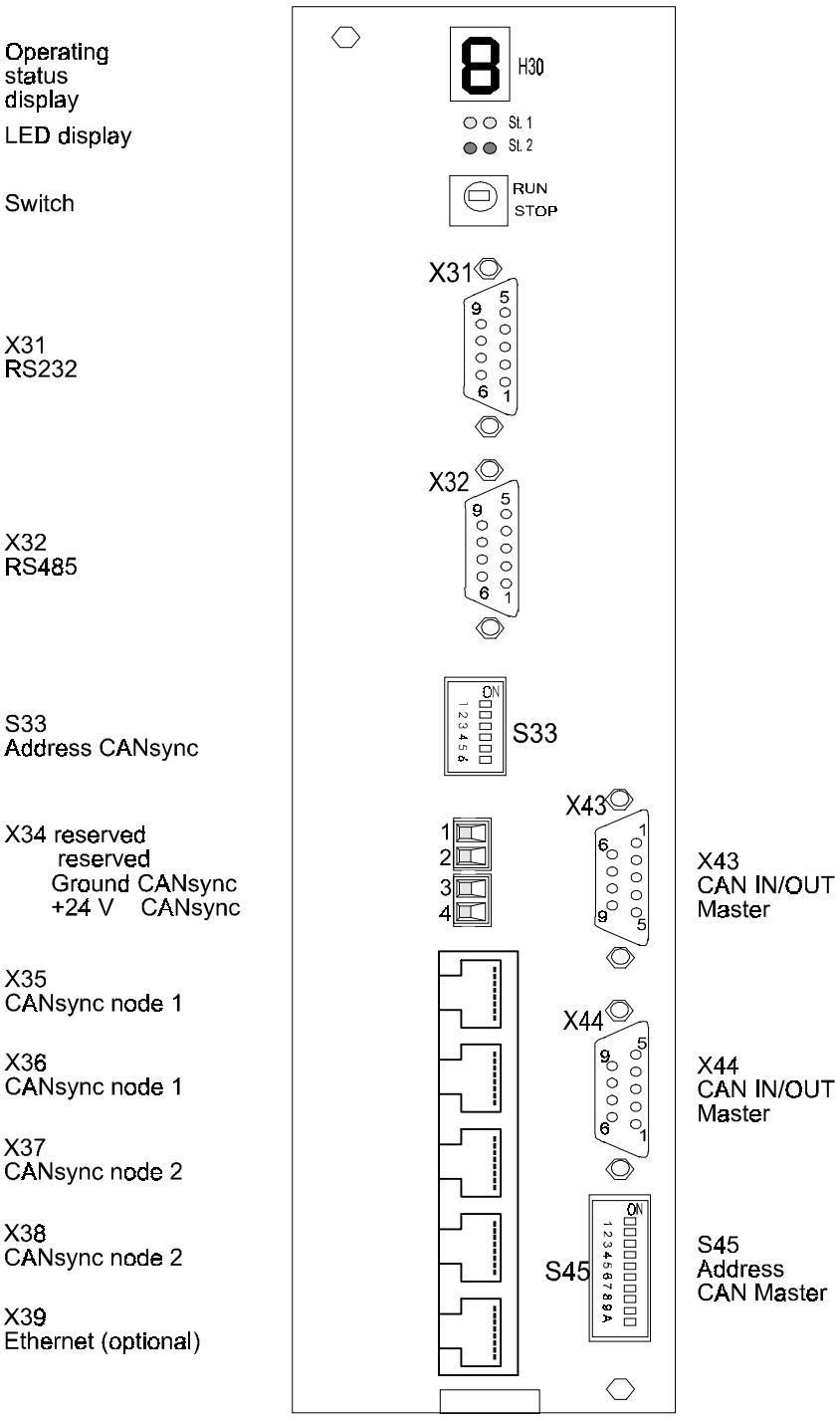
2.2 Technical Data of CAN-Master

CPU	16-MHz 8-bit CPU
Operating voltage	+5 V
Current consumption	Max. 1A
Ambient temperature	0 ... 55° C
Storage temperature	-15 ... 70°C
Rel. humidity	Max. 90%
Memory	32 kB RAM, 64 kB EPROM
Link to Ωmega Drive-Line II	Dual port RAM 2k x 16
CAN controller	SJA1000T
Physical layer	ISO 11898
Baud rate	Max. 1 Mbps
Potential separation	Optocoupler, DC/DC converter
Plug connection to the CAN bus	9-pin SUB-D male connector and female connector

3 INSTALLATION

3.1 Checks Before Switching On

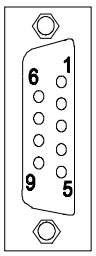
- Connect the connectors



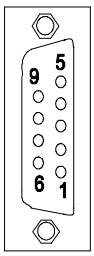
3.2 Pin assignments

- CAN-IN / -OUT

X 43 SUB-D plug connector
X 44 SUB-D female connector



X 43



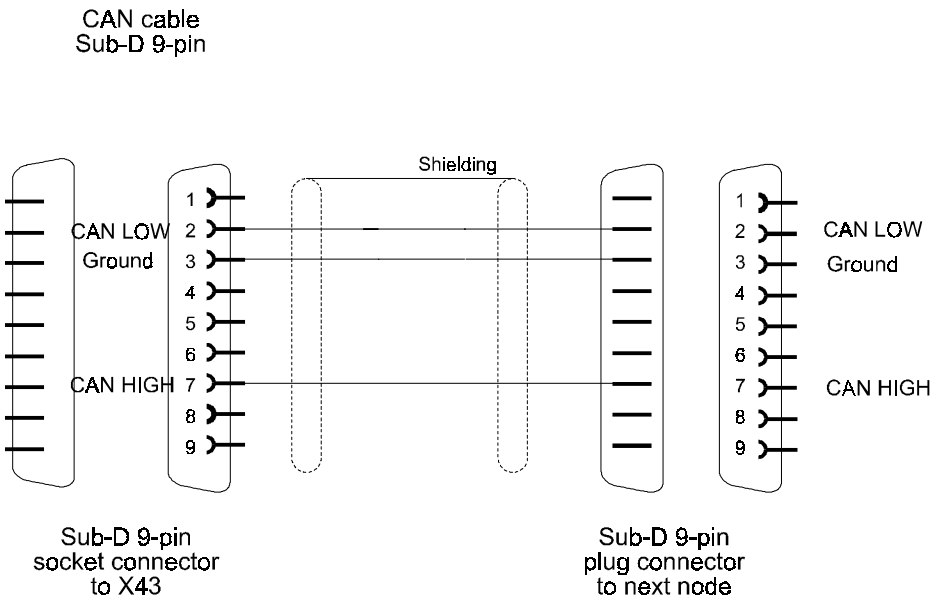
X 44

Pin No.	Assignment
1	Reserved
2	CAN LOW bus line (dominant low)
3	GND Ground
4	Reserved
5	Reserved
6	Reserved
7	CAN HIGH bus line (dominant high)
8	Reserved
9	Reserved

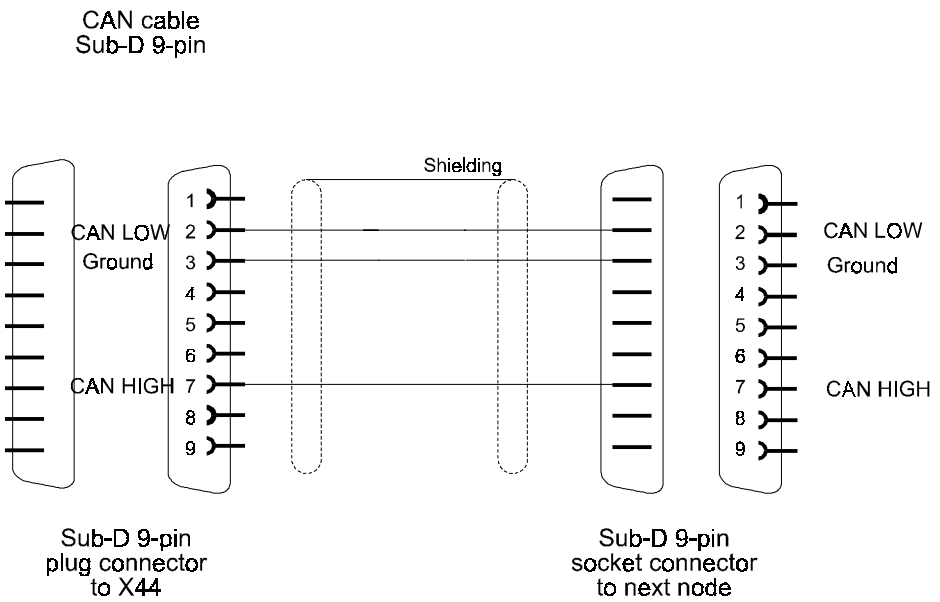
3.3 Connecting Cables

Connecting cables for further CAN nodes

- 9-pin connection for X43



- 9-pin connection for X44

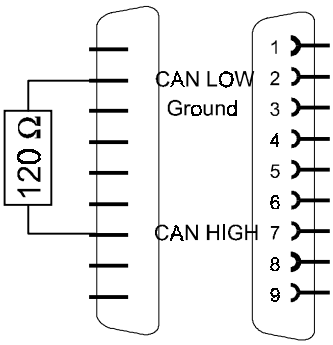


3.4 Accessories

120 Ω terminating resistor connector

- 9-pin connection

CAN terminating resistor connector
Sub-D 9-pin



Sub-D 9-pin
plug connector
to X44

4 CAN NODES

The Basic CAN Controller SJA1000T is used as the controller. The integrated firmware makes it possible to emulate a Full CAN Controller with a considerably increased number of objects. The bus connection is made in accordance with ISO 11898 (CAN High-Speed).

The bus connection is galvanically separated via rapid optocouplers and is supplied via a local DC/DC converter.

Features

– up to 16 drives (with Baumüller CAN interface module)
– up to 16 absolute value encoders
– up to 16 distributed I/O modules
– up to 15 further Ωmega CAN interface modules

that are operator-accessible via integrated firmware.

CAN properties

- serial asynchronous bus system
- realtime-capable (maximum of 1 Mbps with a 40-meter bus)
- broadcast/multicast and point-to-point communication
- powerful error detection and handling
- high reliability (Hamming distance = 6)
- Multimaster
- priority-driven bus assignment
- guaranteed maximum latency for high-prioritized messages
- open system
- complies with international standards

4.1 Information on Programming

For programming the CAN, you have available in a PROPROG wt II project under IEC 61131-3 the function blocks from library CAN_DLII_20bd00 (or above) (see “Function Blocks for CAN” on page 55).

Base address

Base address of CAN-M-01 CAN-Master option board	%MB3.3000000

Initialization

You only need to initialize the CAN interface module after a system start (reset or power-up). The following FBs must be called after a reset or power-up:

FB CAN_DIG_INPUT_INIT
FB CAN_DIG_OUTPUT_INIT
FB CAN_DRIVE_INIT
FB CAN_INIT

FB CAN_INIT carries out initialization. FBs CAN_DIG_INPUT_INIT, CAN_DIG_OUTPUT_INIT and CAN_DRIVE_INIT are for simple interconnection of several of FB CAN_INIT's inputs.

Process data communication

FB CAN_PD_COMM in a cyclical task is used for process data communication.

Requirements data communication

The following FBs are available for requirements data communication:

FB CAN_OBJ_READ
FB CAN_OBJ_WRITE
FB CAN_PA_LIST_READ
FB CAN_PA_LIST_WRITE
FB CAN_PAR_READ
FB CAN_PAR_WRITE
FB CAN_PE_LIST_READ
FB CAN_PE_LIST_WRITE
FB CAN_SD_CONTROL

The assignment of communications RAM in the Ωmega for the CAN-M-01 interface module is explained below. Structures have been created for accessing individual registers in communication RAM that make possible user-friendly access to communication RAM.

At initialization of the CAN-M-01 interface module, the registers in communication RAM have a different meaning than with cyclical operation.

This means that, for initialization there is the

CAN_INIT_BMSTRUCT structure

and for cyclical operation, the

CAN_CTRL_BMSTRUCT structure

These structures contain

- 8-bit elements,
- 16-bit elements,
- 32-bit elements,
- structures from the elements mentioned above
- Fields (ARRAY) from the elements and structures mentioned above

Short designations have been prepended to the data types (8-, 16-, 32-bit elements, structures and fields) that are used in a structure. This is for the sake of clarity when using the structures in programming.

Data type	Short designation	Number of bits
BYTE	b	8
WORD	w	16
DWORD (double word)	d	32
SINT (short integer)	si	8
INT (integer)	i	16
DINT (double integer)	di	32
USINT (unsigned short integer)	us	8
UINT (unsigned integer)	u	16
UDINT (unsigned double integer)	ud	32
STRUCT	_ (underline)	-
ARRAY	a	-

Other data types that are not used in the structures include:

Data type	Short designation	Number of bits
BOOL (bit)	x	1
TIME	t	-

Explanation of declaring the global variables

The structures are assigned to the base address of the respective communication RAM for a CAN-Master interface module at programming in PROPROG wt II.

For initialization, you create a global variable of type CAN_INIT_BMSTRUCT that is fixed to the base address of the CAN-M-01 interface module.

e.g. `_CAN_INIT_OPT AT %MB3.3000000 : CAN_INIT_BMSTRUCT;`

for the CAN node on the CAN-M-01 option board on the Ωmega Drive-Line II,

Where, for example,

`_CAN_INIT_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_INIT_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN-M-01 interface module on the Ωmega Drive-Line II.

For cyclical operation, you create a global variable of type CAN_CTRL_BMSTRUCT that is fixed to the base address of the CAN-Master interface module.

e.g. `_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;`

for the CAN node on the CAN-M-01 option board on the Ωmega Drive-Line II,

Where, for example,

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN-M-01 interface module on the Ωmega Drive-Line II.

In the following tables, the variable name is replaced by an asterisk (*).

This means that you access register `*.w_CPU_CONTROL` via

`_CAN_INIT_OPT.w_CPU_CONTROL`

you access `*.w_OPTION_STATUS` via

`_CAN_INIT_OPT.w_OPTION_STATUS.`

Where, for example,

`_CAN_INIT_OPT`	is the variable name with the data type short designation "_" for STRUCT
`w_CPU_CONTROL`	is the control register with the data type short designation "w" for WORD

Registers `*.w_CPU_CONTROL` and `*.w_OPTION_STATUS` can also be triggered via the structure for cyclical operation. This makes possible access via

`_CAN_CTRL_OPT.w_CPU_CONTROL` and

`_CAN_CTRL_OPT.w_OPTION_STATUS.`

Where, for example,

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`w_OPTION_STATUS`	is the status register with the data type short designation "w" for WORD

Example of accessing an element of a field that is used in the structure:

According to table: `*.a_VALUE_DI_0_15[3]`
Access: `_CAN_CTRL_OPT.a_VALUE_DI_0_15[3]`

Where:

`_CAN_INIT_OPT`	is the variable name with the data type short designation "_" for STRUCT
`a_VALUE_DI_0_15[3]`	is the register for the input image of the CAN-DI-16-01 digital input module with CAN node ID 3 (-> [3]) and with the data type short designation "a" for ARRAY. The data type of the elements of the field (of the input image) is taken from the corresponding table and the description.

Example of accessing an element a (sub) structure, which is itself a field that is used in the structure:

According to table: `*.a_FIFO_0_31[31].w_ID`
Access: `_CAN_CTRL_OPT.a_FIFO_0_31[31].w_ID`

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`a_FIFO_0_31[31]`	is the field containing the reception data of the FIFO buffer for the message frame entered at entry 31 with the data type short designation "a" for ARRAY
`w_ID`	is the register for the identifier of the message frame that is entered at entry 31 with the data type short designation "w" for WORD

General registers of the CAN interface module

Register	Contents
*.w_CAN_STATUS	CAN status
*.w_OMEGA_NR	The Ωmega number set using a DIP switch
*.i_SW1_NR	Card software number
*.i_SW1_RELEASE	Software revision incompatible and compatible

CAN status

Each time the CAN processor cycle is run through, the system outputs the CAN status to *.w_CAN_STATUS.

Meaning

Bit No.	Meaning (bit = TRUE)
0	Reserved
1	Overrun: A CAN message could not be received
2	CAN send buffer is free
3	CAN send job executed successfully
4	CAN message currently being received
5	CAN message currently being sent
6	Error present (warning)
7	CAN node is deactivated (bus off)
8-15	Reserved

Ωmega number

In register *.w_OMEGA_NR, the system displays the Ωmega number that was set using the DIP switch (S45). In the case of a CAN interface module, this number is the Ωmega number.

Software number and software version

In register *.i_SW1_NR, the system displays the number of the CAN software on the Ωmega Drive-Line II.

In register *.i_SW1_RELEASE, the system displays the compatible and the incompatible revision of the CAN software on the Ωmega Drive-Line II.

The CAN interface module is controlled via register *.w_CPU_CONTROL. The system displays the current status in register *.w_OPTION_STATUS.

Register	**Contents**
*.w_CPU_CONTROL	Control register of CAN interface module
*.w_OPTION_STATUS	Status register of CAN interface module

(* at initialization, corresponds to _CAN_INIT_OPT, for example; after initialization, corresponds to _CAN_CTRL_OPT) for example

Control register of CAN interface module	Meaning
16#0000	Cold restart
16#0001	Test handshake
16#0002	Take over initialization data
16#0080	(Bit 7 = TRUE) reset CAN controller
16#0100	Start operation

Status register of CAN interface module	Meaning
16#0001	Start up
16#0002	Take over waiting for initialization data
16#0003	Waiting for start
16#0100	Operation has started

4.2 Configuring

4.2.1 Bus Timing

By connecting inputs b_BIT_TIMING0 and b_BIT_TIMING1 on FB CAN_INIT, you can individually initialize the bus timing of the SJA1000T CAN controller.

Bus timing register b_BIT_TIMING0							
7	6	5	4	3	2	1	0
SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

Bus timing register b_BIT_TIMING1							
7	6	5	4	3	2	1	0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0

In bus timing register 0, you specify the values for baud rate prescaler *(BRP)* and Synchronization Jump Width *(SJW).*

From this, you can determine the period duration of the system cycle:

$t_{SCL} = 2t_{CLK}$ (32BRP.5 + 16BRP.4 + 8BRP.3 + 4BRP.2 + 2BRP.1 + BRP.0 + 1)

t_{CLK} = 62.5ns

You can determine the Synchronization Jump Width *(SJW)* as follows:

$t_{SJW} = t_{SCL}$ (2SJW.1 + SJW.0 + 1)

In bus timing register 1, you specify the length of time segments 1 *(TSEG1)* and 2 *(TSEG2),* as well as the number of samples per bit *(SAM).*

$t_{SEG1} = t_{SCL}$ (8TSEG1.3 + 4TSEG1.2 + 2TSEG1.1 + TSEG1.0 + 1)

$t_{SEG2} = t_{SCL}$ (4TSEG2.2 + 2TSEG2.1 + TSEG2.0 + 1)

Number of samples:

Bit	Value	Meaning
SAM	1	3 samples
	0	1 sample (preferred init.)

Bus timing (definitions)

– ISO

Sync_Seg	Prop_Seg	Phase_Seg1	Phase_Seg2
1	1-8	1-8	1-8
8-25 * t_{SCL}			

– SJA1000T

Sync_Seg	TSEG1	TSEG2
1	2-16	2-8
5-25 * t_{SCL}		

- Sync_SEG synchronization segment
- TSEG1 corresponds to Prop_Seg + Phase_Seg1;
  defines the position of the sampling instant
- TSEG2 corresponds to Phase_Seg2;
  Synchronization buffer;
- N_{SJW} Synchronization Jump Width;
  Number of bit time units (1-4) by which the bit time is to be extended or reduced in the case of postsynchronization.

4.2.2 Acceptance Filter

You set the SJA1000T CAN controller's acceptance filter via inputs b_ACCEPT_MASK (AM) and b_ACCEPT_CODE (AC) on FB CAN_INIT.

Acceptance Code Register ACR							
7	6	5	4	3	2	1	0
AC.7	AC.6	AC.5	AC.4	AC.3	AC.2	AC.1	AC.0

Acceptance Mask Register AMR							
7	6	5	4	3	2	1	0
AM.7	AM.6	AM.5	AM.4	AM.3	AM.2	AM.1	AM.0

Bits AC.7 - AC.0 in the Acceptance Code Register and the eight most significant identifier bits Id10 - Id3 of an object must be the same at the bit positions that are marked as relevant in the Acceptance Mask Register.

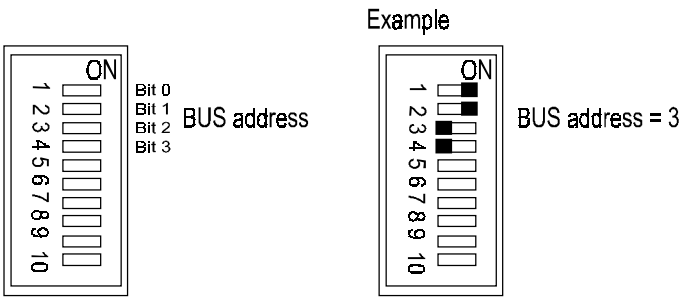
Acceptance Mask Bit	Value	Meaning
AM.7 - AM.0	1	This bit position is don't care for the acceptance of an object
	0	This bit position is relevant to the acceptance of an object

Since this acceptance filtering only takes into account identifier bits Id10 - Id3, at least eight objects always pass through the filter.

4.2.3 Setting the Bus Address

(From software version 3.01 onwards)

Using DIP switches 1 - 4 (S45 on the CAN-M-01), you set the bus address binary-coded with which the system can trigger the Ωmega. You must set for each Ωmega on a CAN bus an address that is different from the others (Ωmega number and CAN node ID).



4.3 Operating Distributed I/O Modules

4.3.1 Addressing

Assignment of register range for node1 (extract):

Address/Register	Meaning
*.w_DIG_IN_LIST	Dig_In_List
*.w_DIG_OUT_LIST	Dig_Out_List
*.a_CTRL_IO_0_15[0]	Control register of IO module 0
*.a_CTRL_IO_0_15[1]	Control register of IO module 1
*.a_CTRL_IO_0_15[2]	Control register of IO module 2
•	•
•	•
•	•
*.a_CTRL_IO_0_15[14]	Control register of IO module 14
*.a_CTRL_IO_0_15[15]	Control register of IO module 15
*.a_VALUE_DI_0_15[0]	Input image of digital input module 0
*.a_VALUE_DI_0_15[1]	Input image of digital input module 1
*.a_VALUE_DI_0_15[2}	Input image of digital input module 2
•	•
•	•
•	•
*.a_VALUE_DI_0_15[14]	Input image of digital input module 14
*.a_VALUE_DI_0_15[15]	Input image of digital input module 15
*.a_VALUE_DO_0_15[0]	Output of digital output module 0
*.a_VALUE_DO_0_15[1]	Output of digital output module 1
*.a_VALUE_DO_0_15[2]	Output of digital output module 2
•	•
•	•
•	•
*.a_VALUE_DO_0_15[14]	Output of digital output module 14
*.a_VALUE_DO_0_15[15]	Output of digital output module 15

* Corresponds, for example, to _CAN_CTRL_OPT.

4.3.2 Communications

*.w_DIG_IN_LIST: **Dig_In_List**

A status register that indicates the node number at which a digital input module is located. You can use it to check initialization. In this connection, bit 0 corresponds to module 0 and bit 15 to module 15. If the corresponding bit is "TRUE", the module in question with the corresponding node number is a CAN-DI-16 module.

*.w_DIG_OUT_LIST: **Dig_Out_List**

A status register that indicates the node number at which a digital output module is located. You can use it to check initialization. In this connection, bit 0 corresponds to module 0 and bit 15 to module 15. If the corresponding bit is "TRUE", the module in question with the corresponding node number is a CAN-DO-16 module.

*.a_CTRL_IO_0_15[0] to *.a_CTRL_IO_0_15[15]: **Control register for distributed I/O modules 0 - 15 (see below)**

After the system has gone through initialization, it displays whether the module has signalled correctly. The entries are of data type BYTE.

16#20: No module (module is missing, there is no power supply, the CAN bus is not plugged-in)

16#00: A module with the corresponding address is present.

*.a_VALUE_DI_0_15[0] to *.a_VALUE_DI_0_15[15]: **Data area of digital input modules**

The area in which the system maps the input status conditions of the digital input modules. At addresses at which no module is located or there is one of a different type, the contents are 0. The entries are of data type WORD.

*.a_VALUE_DO_0_15[0] to *.a_VALUE_DO_0_15[15]: **Data area of digital/relay output modules**

The area to which the system writes the output words of the digital or relay output modules. At addresses at which no module is located or there is one of a different type, the system ignores any data that may have been entered. The entries are of data type WORD.

Digital input (CAN-DI 16)

FB CAN_DIG_INPUT_INIT allows you to set various operating modes.

- Operating mode 1
  Not implemented.

- Operating mode 2
  Requesting input data via the control register.
  By writing 16#01 to the control register that is associated with the respective input module, the system requests the input word via a Remote Frame.
  If the desired data is located at the corresponding address in the input data area, the system displays this by 16#02 in the control register.

- Operating mode 3:
  Cyclical requesting of input data. The cycle time must be specified at initialization. The desired data is located at the corresponding address in the input data area.
  Note that in this operating mode too, the system indicates that the input word has been received by displaying 16#02 in the control register of the respective module.

NOTE

Regardless of the set operating mode, you can use registers input w_DI_POS_EDGE and w_DI_NEG_EDGE respectively to make the module signal automatically a change in the input status (see technical description of CAN-DI-16).

Digital/relay output (CAN-DO 16/CAN-DO 8R)

FB CAN_DIG_OUTPUT_INIT allows you to set various operating modes.

- Operating mode 1
  Not implemented.

- Operating mode 2
  Sending output data via the control register.
  By writing 16#03 to the control register that is associated with the respective output module, the system writes the output word to the module. The desired data must be available at the corresponding address in the output data area.
  The system indicates that the output word has been sent successfully by displaying 16#04 in the control register.

- Operating mode 3:
  Cyclical sending of output data. The cycle time must be specified at initialization. The desired data must be available at the corresponding address in the output data area.
  Note that in this operating mode too, the system indicates that the output word has been sent by displaying 16#04 in the control register of the respective module.

4.4 Operating a Maximum of 16 Controllers

4.4.1 Addressing

A register area of 32 words in communication RAM is reserved for each of the maximum of 16 controllers with CAN interface modules (which are referred to from now on as controllers or drives).

Assignment of register area:

Address/Register	Meaning
*.a_DRIVE_0_15[0].w_D_CONTROLWORD	Drive 0 control word
*.a_DRIVE_0_15[0].w_D_STATUSWORT	Drive 0 status word
*.a_DRIVE_0_15[0].w_D_DUMMY_0	Reserved for Drive 0
*.a_DRIVE_0_15[0].w_D_WRVALUE_0	Drive 0 reference value 0
*.a_DRIVE_0_15[0].d_D_WRVALUE_1	Drive 0 reference value 1
*.a_DRIVE_0_15[0].d_D_WRVALUE_2	Drive 0 reference value 2
*.a_DRIVE_0_15[0].d_D_WRVALUE_3	Drive 0 reference value 3
*.a_DRIVE_0_15[0].w_D_DUMMY_1	Reserved for Drive 0
*.a_DRIVE_0_15[0].w_D_RDVALUE_0	Drive 0 actual value 0
*.a_DRIVE_0_15[0].d_D_RDVALUE_1	Drive 0 actual value 1
*.a_DRIVE_0_15[0].d_D_RDVALUE_2	Drive 0 actual value 2
*.a_DRIVE_0_15[0].d_D_RDVALUE_3	Drive 0 actual value 3
*.a_DRIVE_0_15[0].w_D_SD_0	Drive 0 requirements data 0
*.a_DRIVE_0_15[0].w_D_SD_1	Drive 0 requirements data 1
*.a_DRIVE_0_15[0].w_D_SD_2	Drive 0 requirements data 2
*.a_DRIVE_0_15[0].w_D_SD_3	Drive 0 requirements data 3
*.a_DRIVE_0_15[0].b_D_CTRLREG_STATUSWORD	Drive 0 control register status word
*.a_DRIVE_0_15[0].w_D_CTRLREG_CONTROLWORD	Drive 0 control register control word
*.a_DRIVE_0_15[0].w_D_DUMMY_2	Reserved for Drive 0
*.a_DRIVE_0_15[0].b_D_CTRLREG_WRVALUE_0	Drive 0 control register reference value 0
*.a_DRIVE_0_15[0].b_D_CTRLREG_WRVALUE_1	Drive 0 control register reference value 1
*.a_DRIVE_0_15[0].b_D_CTRLREG_WRVALUE_2	Drive 0 control register reference value 2
*.a_DRIVE_0_15[0].b_D_CTRLREG_WRVALUE_3	Drive 0 control register reference value 3
*.a_DRIVE_0_15[0].b_D_CTRLREG_RDVALUE_0	Drive 0 control register actual value 0
*.a_DRIVE_0_15[0].b_D_CTRLREG_RDVALUE_1	Drive 0 control register actual value 1
*.a_DRIVE_0_15[0].b_D_CTRLREG_RDVALUE_2	Drive 0 control register actual value 2
*.a_DRIVE_0_15[0].b_D_CTRLREG_RDVALUE_3	Drive 0 control register actual value 3
*.a_DRIVE_0_15[0].b_D_CTRLREG_SD	Drive 0 control register requirements data
*.a_DRIVE_0_15[0].b_D_CC_REG	Drive 0 communications control requirements data
*.a_DRIVE_0_15[0].w_D_ERROR_NR	Drive 0 error number requirements data
*.a_DRIVE_0_15[0].w_D_PARAMETER_NR	Drive 0 parameter number requirements data
*.a_DRIVE_0_15[0].b_D_FORMAT	Drive 0 format requirements data
*.a_DRIVE_0_15[0].b_D_ELEMENT	Drive 0 element requirements data
*.a_DRIVE_0_15[1].w_D_CONTROLWORD	Drive 1 control word
*.a_DRIVE_0_15[1].w_D_STATUSWORT	Drive 1 status word
*.a_DRIVE_0_15[1].w_D_DUMMY_0	Reserved for Drive 1

Address/Register	Meaning
*.a_DRIVE_0_15[1].w_D_WRVALUE_0	Drive 1 reference value 1
*.a_DRIVE_0_15[1].d_D_WRVALUE_1	Drive 1 reference value 1
*.a_DRIVE_0_15[1].d_D_WRVALUE_2	Drive 1 reference value 2
*.a_DRIVE_0_15[1].d_D_WRVALUE_3	Drive 1 reference value 3
*.a_DRIVE_0_15[1].w_D-DUMMY_1	Reserved for Drive 1
*.a_DRIVE_0_15[1].w_D_RDVALUE_0	Drive 1 actual value 0
*.a_DRIVE_0_15[1].d_D_RDVALUE_1	Drive 1 actual value 1
*.a_DRIVE_0_15[1].d_D_RDVALUE_2	Drive 1 actual value 2
*.a_DRIVE_0_15[1].d_D_RDVALUE_3	Drive 1 actual value 3
*.a_DRIVE_0_15[1].w_D_SD_0	Drive 1 requirements data 0
*.a_DRIVE_0_15[1].w_D_SD_1	Drive 1 requirements data 1
*.a_DRIVE_0_15[1].w_D_SD_2	Drive 1 requirements data 2
*.a_DRIVE_0_15[1].w_D_SD_3	Drive 1 requirements data 3
*.a_DRIVE_0_15[1].b_D_CTRLREG_STATUSWORD	Drive 1 control register status word
*.a_DRIVE_0_15[1].b_D_CTRLREG_CONTROLWORD	Drive 1 control register control word
*.a_DRIVE_0_15[1].w_D_DUMMY_2	Reserved for Drive 1
*.a_DRIVE_0_15[1].b_D_CTRLREG_WRVALUE_0	Drive 1 control register reference value 0
*.a_DRIVE_0_15[1].b_D_CTRLREG_WRVALUE_1	Drive 1 control register reference value 1
*.a_DRIVE_0_15[1].b_D_CTRLREG_WRVALUE_2	Drive 1 control register reference value 2
*.a_DRIVE_0_15[1].b_D_CTRLREG_WRVALUE_3	Drive 1 control register reference value 3
*.a_DRIVE_0_15[1].b_D_CTRLREG_RDVALUE_0	Drive 1 control register actual value 0
*.a_DRIVE_0_15[1].b_D_CTRLREG_RDVALUE_1	Drive 1 control register actual value 1
*.a_DRIVE_0_15[1].b_D_CTRLREG_RDVALUE_2	Drive 1 control register actual value 2
*.a_DRIVE_0_15[1].b_D_CTRLREG_RDVALUE_3	Drive 1 control register actual value 3
*.a_DRIVE_0_15[1].b_D_CTRLREG_SD	Drive 1 control register requirements data
*.a_DRIVE_0_15[1].b_D_CC_REG	Drive 1 communications control requirements data
*.a_DRIVE_0_15[1].w_D_ERROR_NR	Drive 1 error number requirements data
*.a_DRIVE_0_15[1].w_D_PARAMETER_NR	Drive 1 parameter number requirements data
*.a_DRIVE_0_15[1].b_D_FORMAT	Drive 1 format requirements data
*.a_DRIVE_0_15[1].b_D_ELEMENT	Drive 1 element requirements data
•	•
•	•
•	•
*.a_DRIVE_0_15[15].w_D_CONTROLWORD	Drive 15 control word
*.a_DRIVE_0_15[15].w_D_STATUSWORT	Drive 15 status word
*.a_DRIVE_0_15[15].w_D_DUMMY_0	Reserved for Drive 15
*.a_DRIVE_0_15[15].w_D_WRVALUE_0	Drive 15 reference value 0
*.a_DRIVE_0_15[15].d_D_WRVALUE_1	Drive 15 reference value 1
*.a_DRIVE_0_15[15].d_D_WRVALUE_2	Drive 15 reference value 2
*.a_DRIVE_0_15[15].d_D_WRVALUE_3	Drive 15 reference value 3
*.a_DRIVE_0_15[15].w_D-DUMMY_1	Reserved for Drive 15
*.a_DRIVE_0_15[15].w_D_RDVALUE_0	Drive 15 actual value 0
*.a_DRIVE_0_15[15].d_D_RDVALUE_1	Drive 15 actual value 1
*.a_DRIVE_0_15[15].d_D_RDVALUE_2	Drive 15 actual value 2
*.a_DRIVE_0_15[15].d_D_RDVALUE_3	Drive 15 actual value 3

Address/Register	Meaning
*.a_DRIVE_0_15[15].w_D_SD_0	Drive 15 requirements data 0
*.a_DRIVE_0_15[15].w_D_SD_1	Drive 15 requirements data 1
*.a_DRIVE_0_15[15].w_D_SD_2	Drive 15 requirements data 2
*.a_DRIVE_0_15[15].w_D_SD_3	Drive 15 requirements data 3
*.a_DRIVE_0_15[15].b_D_CTRLREG_STATUSWORD	Drive 15 control register status word
*.a_DRIVE_0_15[15].b_D_CTRLREG_CONTROLWORD	Drive 15 control register control word
*.a_DRIVE_0_15[15].w_D_DUMMY_2	Reserved for Drive 15
*.a_DRIVE_0_15[15].b_D_CTRLREG_WRVALUE_0	Drive 15 control register reference value 0
*.a_DRIVE_0_15[15].b_D_CTRLREG_WRVALUE_1	Drive 15 control register reference value 1
*.a_DRIVE_0_15[15].b_D_CTRLREG_WRVALUE_2	Drive 15 control register reference value 2
*.a_DRIVE_0_15[15].b_D_CTRLREG_WRVALUE_3	Drive 15 control register reference value 3
*.a_DRIVE_0_15[15].b_D_CTRLREG_RDVALUE_0	Drive 15 control register actual value 0
*.a_DRIVE_0_15[15].b_D_CTRLREG_RDVALUE_1	Drive 15 control register actual value 1
*.a_DRIVE_0_15[15].b_D_CTRLREG_RDVALUE_2	Drive 15 control register actual value 2
*.a_DRIVE_0_15[15].b_D_CTRLREG_RDVALUE_3	Drive 15 control register actual value 3
*.a_DRIVE_0_15[15].b_D_CTRLREG_SD	Drive 15 control register requirements data
*.a_DRIVE_0_15[15].b_D_CC_REG	Drive 15 communications control requirements data
*.a_DRIVE_0_15[15].w_D_ERROR_NR	Drive 15 error number requirements data
*.a_DRIVE_0_15[15].w_D_PARAMETER_NR	Drive 15 parameter number requirements data
*.a_DRIVE_0_15[15].b_D_FORMAT	Drive 15 format requirements data
*.a_DRIVE_0_15[15].b_D_ELEMENT	Drive 15 element requirements data

* Corresponds, for example, to _CAN_CTRL_OPT.

4.4.2 Communications

In the following descriptions, we will use the following abbreviations:

16#	hexadecimal
2#	binary
ID	identifier
IDB1	identifier bits 10-3
IDB2	identifier bits 2-0
XXXX	operated axle number (0-15)
RTR	remote bit
DLC	number of data bytes
DB0-DB7	data bytes 0 to 7

The stated addresses apply to the controller with CAN node ID 0 (in the case of controllers with different CAN node IDs, you must use a_DRIVE_0_15[CAN node ID)]. (See above)

For the meanings of the parameters, refer to the respective controller description.

Process data

You can use FB CAN_PD_COMM for process data communication. The process data in question is high-priority messages.

- control word
- status word
- reference values 0-3
- actual values 0-3

Control word (→ FB CAN_PD_COMM)

Parameter number in the controller: 120

There are two options for making available a control word on the bus:

- Writing 16#03 to *control register control word* (*.a_DRIVE_0_15[0].b_D_CTRLREG_CONTROLWORD)
  The contents of *.a_DRIVE_0_15[0].w_D_CONTROLWORD *(control word)* are made available on the bus.
  The system acknowledges this by 16#04 in the *control register control word.*

- Changing the control word
  If the control word is changed, the new contents are automatically made available on the bus.
  The system acknowledges that sending was successful by 16#04 in the corresponding control register.

Communications object:

ID	RTR	DLC	DB0	DB1
2#0000XXXX001	0	2	Control word	

Status word (→ FB CAN_PD_COMM)

Parameter number in the controller: 121

To request the status word, the system must write *control register status word* (*.a_DRIVE_0_15[0].b_D_CTRLREG_STATUSWORD) with 16#01.

This results in the following request:

Status word request

ID	RTR	DLC
2#0001XXXX001	1	0

The system expects the following message as the response:

This object is received even without a request, i. e. the system does not have to explicitly request the status word, since the drive node sends it automatically when there is a change of status.

Communications object:

ID	RTR	DLC	DB0	DB1
2#0001XXXX001	0	2	Status word	

The system acknowledges reception with 16#02 in *control register status word*. After this, the new data is available in register *.a_DRIVE_0_15[0].w_D_STATUSWORT *(status word).*

Reference values (→ FB CAN_PD_COMM)

A maximum of four reference values are available, which you can send cyclically, if desired.

You can set as the reference value any writable parameter that the controller makes available. You set the desired parameter numbers using FB CAN_DRIVE_INIT. You can also set the desired parameter numbers using "PA-Daten schreiben" [Write PA data], which will be dealt with separately. Note in this connection that the transferred reference value length matches the format of the parameter in the controller. The following data types are supported:

Reference value 0:	- word	DLC=2;
Reference value 1-3	- word	DLC=2;
	- Doubleword	DLC=4;

The respective format of reference value **x** is specified in *control register reference value* (*.a_DRIVE_0_15[0].b_D_CTRLREG_WRVALUE_**x**).

2#xxxx 0001:	word
2#xxxx 0010:	doubleword

You must enter the reference values in the corresponding register *reference value 0-3* (*.a_DRIVE_0_15[0].w_D_WRVALUE_0 to *.a_DRIVE_0_15[0].w_D_WRVALUE_3).

If you have not chosen cyclical transfer, the system controls it via *control register reference value.* Writing a reference value to the bus is started by 2#0011xxxx in *control register reference value.* The system confirms that sending was successful with 2#0100xxxx.

Communications object:

IDB0	IDB1	RTR	DLC	DB0	DB1	DB2	DB3
2#0010XXXX	SV No.	0	1-4	Reference value data			

SW_Nr: reference value number 0-3

Example: Transferring reference value 1 to the controller with CAN node ID 0 (no cycle time set)

- Set the desired parameter numbers using FB CAN_DRIVE_INIT, FB CAN_INIT (initialization)
- Write the data to the intended address:
  Data in *.a_DRIVE_0_15[0].d_D_WRVALUE_1
- Send the message via *control register reference value 1*
  (*.a_DRIVE_0_15[0].b_D_CTRLREG_WRVALUE_1).

  - Word: 16#31

  - Doubleword: 16#32

  If the reference value has been sent, the system displays this in *control register reference value 1.*

  - Word: 16#41

  - Doubleword: 16#42

Reference values (→ FB CAN_PD_COMM)

A maximum of four actual values are available, which you can request cyclically, if desired. You can set as the actual value any parameter that the controller makes available. You set the desired parameter numbers using FB CAN_DRIVE_INIT. You can also set the desired parameter numbers using "PA-Daten schreiben" [Write PE data], which will be dealt with separately. The following data types are supported:

Actual value 0: -	Word	DLC=2;
Actual value 1-3-	Word	DLC=2;
-	Doubleword	DLC=4;

To request an actual value from the controller, the system must write *control register actual value* **x** (*.a_DRIVE_0_15[0].b_D_CTRLREG_RDVALUE_**x**) with 16#01. This results in the following request:

IDB0	IDB1	RTR	DLC
2#0011XXXX	AV No.	1	0

IW_Nr: actual value number 0-3

The system expects the following message as the response.

Communications object:

IDB0	IDB1	RTR	DLC	DB0	DB1	DB2	DB3
2#0011XXXX	AV No.	0	1-4	Actual value data			

Example: Requesting actual value 0 from the controller with CAN node ID 0 (not a cyclical request)

- Set the desired parameter numbers using FB CAN_DRIVE_INIT, FB CAN_INIT (initialization)
- Enter 16#01 in *control register actual value 0*
  (*.a_DRIVE_0_15[0].b_D_CTRLREG_RDVALUE_0)
- The system indicates that the desired actual value has been received by displaying 16#02 in *control register actual value 0.*
- The received actual value is available in *.a_DRIVE_0_15[0].w_D_RDVALUE_0.

Requirements data

For requirements data communication, you can use FBs CAN_PA_LIST_READ, CAN_PA_LIST_WRITE, CAN_PE_LIST_READ, CAN_PE_LIST_WRITE, CAN_PAR_READ, CAN_PAR_WRITE, CAN_OBJ_READ, CAN_OBJ_WRITE and CAN_SD_CONTROL. Requirements data transfer is for directly accessing all the controller's parameters or the objects that are in a controller's object list. By contrast with process data transfer, the system transfers at the same time the parameter numbers (object numbers) with every message. The CAN interface module confirms all the read or write accesses on the respective controller.

PA message

PA messages are for reading or writing the parameter numbers of the reference values that are to be sent using reference value transfer (process data). On the controller, there is a list in which a default setting is saved. If this list is overwritten, it is possible to take over the new setting using global parameter storage (parameter 190).

Word number	PA parameter number list
0	Parameter number of reference value 0
1	Parameter number of reference value 1
2	Parameter number of reference value 2
3	Parameter number of reference value 3

When specifying reference value numbers, you must ensure that reference value 0 has a maximum length of 2 bytes. You must enter 16#0000 as the parameter number of reference values that are not needed.

Write PA data (→ FB CAN_PA_LIST_WRITE)

Using the write PA data function, you set a new PA parameter number list in the controller.

– Enter the four parameter numbers in *requirements data 0-3*
(*.a_DRIVE_0_15[0].w_D_SD_0 to *.a_DRIVE_0_15[0].w_D_SD_3)
– Enter 16#0B in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
2#1000XXXX100	0	8	Reference value no. 0		Reference value no. 1		Reference value no. 2		Reference value no. 3	

Note: The reference values are released using object 16#6002

The system expects the following response to confirm that the write access was successful:

ID	RTR	DLC	DB0
2#1000XXXX101	0	1	16#00

The system indicates that the response was received by displaying 16#0C *in control register requirements data.*

If the write access cannot be carried out, the system expects the following error message.

ID	RTR	DLC	DB0	DB1	DB2
2#1000XXXX101	0	3	16#80	16#00	16#41

The system indicates the error by displaying 16#2A in *control register requirements data.*

Read PA data (→ FB CAN_PA_LIST_READ)

Using the read PA data function, you can read out the current PA parameter number list in the controller.

– Enter 16#09 in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Request:

ID	RTR	DLC
2#1000XXXX100	1	0

The system expects the requested data in the following form:

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
2#1000XXXX101	0	8	Reference value no. 0		Reference value no. 1		Reference value no. 2		Reference value no. 3	

The system indicates that the response was received by displaying 16#0A in *control register requirements data.*

The four received parameter numbers are located in registers *requirements data 0-3* (*.a_DRIVE_0_15[0].w_D_SD_0 to *.a_DRIVE_0_15[0].w_D_SD_3)

If the read access cannot be carried out, the system expects the following error message.

ID	RTR	DLC	DB0	DB1	DB2
2#1000XXXX011	0	3	16#20	Error no.	

The system indicates the error by displaying 16#2A in *control register requirements data.*

PE message

PE messages are for reading or writing the parameter numbers of the actual values that the controller's CAN interface module is to send using actual value transmission (process data). On the controller, there is a list in which a default setting is saved. If this list is overwritten, it is possible to take over the new setting using global parameter storage (parameter 190).

Word number	PE parameter number list
0	Parameter number of actual value 0
1	Parameter number of actual value 1
2	Parameter number of actual value 2
3	Parameter number of actual value 3

When specifying actual value numbers, you must ensure that actual value 0 has a maximum length of 2 bytes. You must enter 16#0000 as the parameter number of actual values that are not needed.

Write PE data (→ FB CAN_PE_LIST_WRITE)

Using the write PE data function, you set a new PE parameter number list in the controller.

– Enter the four parameter numbers in *requirements data 1-4*
(*.a_DRIVE_0_15[0].w_D_SD_0 to *.a_DRIVE_0_15[0].w_D_SD_3)

– Enter 16#0F in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
2#1000XXXX110	0	8	Actual value no. 0		Actual value no. 1		Actual value no. 2		Actual value no. 3	

The system expects the following response to confirm that the write access was successful:

ID	RTR	DLC	DB0
2#1000XXXX111	0	1	16#00

The system indicates that the response was received by displaying 16#10 *in control register requirements data.*

If the read access cannot be carried out, the system expects the following error message.

ID	RTR	DLC	DB0	DB1	DB2
2#1000XXXX111	0	3	16#80	16#00	16#41

The system indicates the error by displaying 16#2E in *control register requirements data.*

Read PE data (→ FB CAN_PE_LIST_READ)

Using the read PE data function, you can read out the current PE parameter number list in the controller.

– Enter 16#0D in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Request:

ID	RTR	DLC
2#1000XXXX110	1	0

The system expects the requested data in the following form:

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
2#1000XXXX111	0	8	Actual value no. 0		Actual value no. 1		Actual value no. 2		Actual value no. 3	

The system indicates that the response was received by displaying 16#0E in *control register requirements data.*

The four received parameter numbers are located in registers *requirements data 0-3* (*.a_DRIVE_0_15[0].w_D_SD_0 to *.a_DRIVE_0_15[0].w_D_SD_3)

If the read access cannot be carried out, the system expects the following error message.

ID	RTR	DLC	DB0	DB1	DB2
2#1000XXXX011	0	3	16#20	Error no.	

The system indicates the error by displaying 16#2E in *control register requirements data.*

Write parameter (→ FB CAN_PAR_WRITE)

Using the write parameter function, you can access all the controller's parameters. It is possible to transfer data in word and doubleword formats. To do this, proceed as follows:

- Enter the controller parameter number (*.a_DRIVE_0_15[0].w_D_PARAMETER_NR)
- Enter the data format to be transferred (*.a_DRIVE_0_15[0].b_D_FORMAT)

  16#01: word

  16#02: doubleword
- Enter the desired element of the parameter (*.a_DRIVE_0_15[0].b_D_ELEMENT)

  e.g. 16#07: parameter data
- Enter the data to be transferred

  Word: *.a_DRIVE_0_15[0].w_D_SD_0

  Doubleword (low word): *.a_DRIVE_0_15[0].w_D_SD_0

  Doubleword (high word): *.a_DRIVE_0_15[0].w_D_SD_1
- Enter 16#07 in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
2#1000XXXX100	0	5-8	Format		PA No.		Data 1-4 bytes			

The system expects the following response to confirm that the write access was successful:

ID	RTR	DLC	DB0
2#1000XXXX011	0	1	16#08

The system indicates that the response was received by displaying 16#08 *in control register requirements data.*

If the system cannot write the parameter, it expects the following error message.

ID	RTR	DLC	DB0	DB1	DB2
2#1000XXXX011	0	3	16#28	Error no.	

The system indicates the error by displaying 16#28 in *control register requirements data.* The associated error number is available in *.a_DRIVE_0_15[0].w_D_ERROR_NR.

Value	Meaning
16#0000	No error occurred
16#FFFF	Error occurred
16#FFFE	Value less than minimum value
16#FFFD	Value greater than maximum value
16#FFFC	Element cannot be changed
16#FFFB	Element not present
16#FFFA	Data not available (e.g. being processed)
16#FFF9	Error in data format

Read parameter (→ FB CAN_PAR_READ)

Using the read parameter function, you can access all the controller's parameters. It is possible to read data in word and doubleword formats. To do this, proceed as follows:

– Enter the controller parameter number (*.a_DRIVE_0_15[0].w_D_PARAMETER_NR)
– Enter the data format to be transferred (*.a_DRIVE_0_15[0].b_D_FORMAT)

  16#01: word

  16#02: doubleword
– Enter the desired element of the parameter (*.a_DRIVE_0_15[0].b_D_ELEMENT)

  e.g. 16#07: parameter data
– Enter 16#05 in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Request:

ID	RTR	DLC	DB0	DB1	DB2	DB3
2#1000XXXX010	0	4	For-mat	Ele-ment	PA No.	

The system expects the requested parameter in the following form:

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4
2#1000XXXX011	0	2-5	For-mat	Data 1-4 bytes			

The system indicates that the response was received by displaying 16#06 *in control register requirements data.*

The received data is located in registers requirements data 0 (and 1 if the parameter is a doubleword)

Word: *.a_DRIVE_0_15[0].w_D_SD_0

Doubleword (low word): *.a_DRIVE_0_15[0].w_D_SD_0

Doubleword (high word): *.a_DRIVE_0_15[0].w_D_SD_1

If the system cannot read the parameter, it expects the following error message.

ID	RTR	DLC	DB0	DB1	DB2
2#1000XXXX011	0	3	16#20	Error no.	

The system indicates the error by displaying 16#26 in *control register requirements data.* The associated error number is available in *.a_DRIVE_0_15[0].w_D_ERROR_NR.

Value	Meaning
16#0000	No error occurred
16#FFFF	Error occurred
16#FFFE	Value less than minimum value
16#FFFD	Value greater than maximum value
16#FFFC	Element cannot be changed
16#FFFB	Element not present
16#FFFA	Data not available (e. g. being processed)
16#FFF9	Error in data format

Write object (→ FB CAN_OBJ_WRITE)

Using the write object function, you can access all the objects in the object list. It is possible to transfer data in byte, word and doubleword formats. To do this, proceed as follows:

– Enter the object number (*.a_DRIVE_0_15[0].w_D_PARAMETER_NR)

– Enter the data format to be transferred (*.a_DRIVE_0_15[0].b_D_FORMAT)

16#00: byte

16#01: word

16#02: doubleword

– Enter the desired subindex of the object (*.a_DRIVE_0_15[0].b_D_ELEMENT)
  e.g. 16#00: to write the entire object
– Enter the data to be transferred

  Word: *.a_DRIVE_0_15[0].w_D_SD_0

  Doubleword (low word): *.a_DRIVE_0_15[0].w_D_SD_0

  Doubleword (high word): *.a_DRIVE_0_15[0].w_D_SD_1
– Enter 16#03 in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
2#1000XXXX000	0	5-8	For-mat	Sub-index	Object no.		Data 1-4 bytes			

The system expects the following response to confirm that the write access was successful:

ID	RTR	DLC	DB0
2#1000XXXX001	0	1	16#00

The system indicates that the response was received by displaying 16#04 *in control register requirements data.*

If the system cannot write the object, it expects the following error message.

ID	RTR	DLC	DB0
2#1000XXXX001	0	1	16#50

The system indicates the error by displaying 16#22 in *control register requirements data.*

Read object (→ FB CAN_OBJ_READ)

Using the read object function, you can access all the objects in the object list. It is possible to read data in byte, word and doubleword formats. To do this, proceed as follows:

– Enter the object number (*.a_DRIVE_0_15[0].w_D_PARAMETER_NR)
– Enter the data format to be transferred (*.a_DRIVE_0_15[0].b_D_FORMAT)

  16#01: word

  16#02: doubleword
– Enter the desired subindex of the object (*.a_DRIVE_0_15[0].b_D_ELEMENT)
  e.g. 16#00: to read the entire object
– Enter 16#01 in *control register requirements data* (*.a_DRIVE_0_15[0].b_D_CTRLREG_SD)

Request:

ID	RTR	DLC	DB0	DB1	DB2	DB3
2#1000XXXX000	0	4	For-mat	Ele-ment	Object no.	

The system expects the requested parameter in the following form:

Communications object:

ID	RTR	DLC	DB0	DB1	DB2	DB3	DB4
2#1000XXXX001	0	2-5	For-mat	Data 1-4 bytes			

The system indicates that the response was received by displaying 16#02 *in control register requirements data.*

The received data is located in registers requirements data 0 (and 1 if the parameter is a doubleword)

Word:		*.a_DRIVE_0_15[0].w_D_SD_0
Doubleword	(low word):	*.a_DRIVE_0_15[0].w_D_SD_0
Doubleword	(high word):	*.a_DRIVE_0_15[0].w_D_SD_1

If the system cannot read the object, it expects the following error message.

ID	RTR	DLC	DB0
2#1000XXXX001	0	1	16#50

The system indicates the error by displaying 16#22 in *control register requirements data.*

4.5 Operating a Maximum of 16 Encoders

4.5.1 Addressing

Note: It is intended to use type HE-65-M-CAN absolute value encoders

Resolution per revolution:	4096 steps
Measuring range:	4096 revolutions
Encoder capacity:	24-bit
Operating voltage:	11-27V_{DC}
Output code:	binary
Baud rate:	250/500 kbps
Interface:	galvanically separated

Address/Register	**Meaning**
*.w_BRC_ENCODER	Control register broadcast
*.CTRL_ENCODER_0_15[0]	Control register encoder 0
*.CTRL_ENCODER_0_15[1]	Control register encoder 1
*.CTRL_ENCODER_0_15[2]	Control register encoder 2
•	•
•	•
•	•
*.CTRL_ENCODER_0_15[14]	Control register encoder 14
*.CTRL_ENCODER_0_15[15]	Control register encoder 15
*.a_ABS_POS_0_15[0]	Absolute position encoder 0
*.a_ABS_POS_0_15[1]	Absolute position encoder 1
*.a_ABS_POS_0_15[2]	Absolute position encoder 2
•	•
•	•
•	•
*.a_ABS_POS_0_15[14]	Absolute position encoder 14
*.a_ABS_POS_0_15[15]	Absolute position encoder 15

* Corresponds, for example, to _CAN_CTRL_OPT.

4.5.2 Communications

The system polls the absolute encoder positions of the connected absolute value encoders by writing 16#01 in *control register broadcast* (*.w_BRC_ENCODER) (WORD format). This sends a data message frame with the identifier “0” and Data Length Code “0”. After this message frame has been sent, the system sets *control register broadcast* back to 16#00.

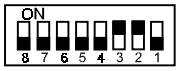
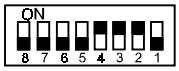
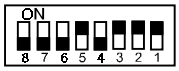
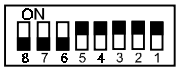
The system confirms that the individual encoder positions have been received by a 16#02 in *control registers encoder 0-15 (* *.CTRL_ENCODER_0_15[0] to *.CTRL_ENCODER_0_15[15] *)* (BYTE format).

The position is available in the following format (using encoder1 as an example):
(DWORD format)

	*.a_ABS_POS_0_15[1]			
BYTE	High word High byte	High word low byte	Low word high byte	Low word low byte
Encoder pos.	2^{23}... 2^{16}	2^{15}... 2^{8}	2^{7}... 2^{0}	0

The system outputs the position in the respective registers *absolute position encoder 0-15* (*.a_ABS_POS_0_15[0] to *.a_ABS_POS_0_15[15].)

Setting the encoder number using the DIP switches on the encoder:

Encoder number	DIP switch ID-bit
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

4.6 Sending Any Message Frame You Like

This function allows you to send any message frames you like that have a maximum of 8 bytes of useful data. The following structure is specified for this:

Address/Register	Meaning							
*.b_STEUREG_TELE	Control register message frame							
*.b_TELE_DUMMY	Reserved							
*.b_TELE_ADR_LOW	Id2	Id1	Id0	RTR	DLC3	DLC2	DLC1	DLC0
*.b_TELE_ADR_HIGH	Id10	Id9	Id8	Id7	Id6	Id5	Id4	Id3
*.b_TELE_DB_0	Data byte 0							
*.b_TELE_DB_1	Data byte 1							
*.b_TELE_DB_2	Data byte 2							
*.b_TELE_DB_3	Data byte 3							
*.b_TELE_DB_4	Data byte 4							
*.b_TELE_DB_5	Data byte 5							
*.b_TELE_DB_6	Data byte 6							
*.b_TELE_DB_7	Data byte 7							

* Corresponds, for example, to _CAN_CTRL_OPT.

After entering the desired data frame, the system makes it available on the bus by writing 16#01 to *control register message frame.* The system indicates that the data frame has been sent successfully by displaying 16#02 in the control register.

(When sending individual message frames, the system must output a value greater than 0 but less than 17 on FB CAN_INIT at input us_MAX_DRIVE_NR_COMM).

4.7 Operating a Maximum of 16 Ωmegas with CAN Interface Modules

4.7.1 Addressing

A 32-byte register area is reserved in the CAN interface module's communication RAM for each Ωmega for carrying out communication with a maximum of 16 Ωmegas. This register area has the following meanings:

Address/Register	Meaning
*.a_OMEGA_0_15[0].b_O_STEUREG_SEND	Ωmega 0 control register send
*.a_OMEGA_0_15[0].b_O_STEUREG_EMPF	Ωmega 0 control register receive
*.a_OMEGA_0_15[0].b_O_STATREG_SEND	Ωmega 0 status register send
*.a_OMEGA_0_15[0].b_O_STATREG_SEND	Ωmega 0 status register receive
*.a_OMEGA_0_15[0].b_O_LENG_SEND	Ωmega 0 length register of send message
*.a_OMEGA_0_15[0].b_O_DUMMY0	Reserved for Ωmega 0
*.a_OMEGA_0_15[0].b_O_LENG_EMPF	Ωmega 0 length register of receive message
*.a_OMEGA_0_15[0].b_O_ABS_BRC	Ωmega 0 sender register with broadcast message
*.a_OMEGA_0_15[0].b_O_SEND_DB_0	Ωmega 0 send register data byte 0
*.a_OMEGA_0_15[0].b_O_SEND_DB_1	Ωmega 0 send register data byte 1
*.a_OMEGA_0_15[0].b_O_SEND_DB_2	Ωmega 0 send register data byte 2
*.a_OMEGA_0_15[0].b_O_SEND_DB_3	Ωmega 0 send register data byte 3
*.a_OMEGA_0_15[0].b_O_SEND_DB_4	Ωmega 0 send register data byte 4
*.a_OMEGA_0_15[0].b_O_SEND_DB_5	Ωmega 0 send register data byte 5
*.a_OMEGA_0_15[0].b_O_SEND_DB_6	Ωmega 0 send register data byte 6
*.a_OMEGA_0_15[0].b_O_SEND_DB_7	Ωmega 0 send register data byte 7
*.a_OMEGA_0_15[0].b_O_EMPF_DB_0	Ωmega 0 receive message data byte 0
*.a_OMEGA_0_15[0].b_O_EMPF_DB_1	Ωmega 0 receive message data byte 1
*.a_OMEGA_0_15[0].b_O_EMPF_DB_2	Ωmega 0 receive message data byte 2
*.a_OMEGA_0_15[0].b_O_EMPF_DB_3	Ωmega 0 receive message data byte 3
*.a_OMEGA_0_15[0].b_O_EMPF_DB_4	Ωmega 0 receive message data byte 4
*.a_OMEGA_0_15[0].b_O_EMPF_DB_5	Ωmega 0 receive message data byte 5
*.a_OMEGA_0_15[0].b_O_EMPF_DB_6	Ωmega 0 receive message data byte 6
*.a_OMEGA_0_15[0].b_O_EMPF_DB_7	Ωmega 0 receive message data byte 7
*.a_OMEGA_0_15[0].d_O_DUMMY1	Reserved for Ωmega 0 (doubleword)
*.a_OMEGA_0_15[0].d_O_DUMMY2	Reserved for Ωmega 0 (doubleword)
*.a_OMEGA_0_15[1].b_O_STEUREG_SEND	Ωmega 1 control register send
*.a_OMEGA_0_15[1].b_O_STEUREG_EMPF	Ωmega 1 control register receive
*.a_OMEGA_0_15[1].b_O_STATREG_SEND	Ωmega 1 status register send
*.a_OMEGA_0_15[1].b_O_STATREG_SEND	Ωmega 1 status register receive
*.a_OMEGA_0_15[1].b_O_LENG_SEND	Ωmega 1 length register of send message
*.a_OMEGA_0_15[1].b_O_DUMMY0	Reserved for Ωmega 1
*.a_OMEGA_0_15[1].b_O_LENG_EMPF	Ωmega 1 length register of receive message
*.a_OMEGA_0_15[1].b_O_ABS_BRC	Ωmega 1 sender register with broadcast message
*.a_OMEGA_0_15[1].b_O_SEND_DB_0	Ωmega 1 send register data byte 0
*.a_OMEGA_0_15[1].b_O_SEND_DB_1	Ωmega 1 send register data byte 1
*.a_OMEGA_0_15[1].b_O_SEND_DB_2	Ωmega 1 send register data byte 2
*.a_OMEGA_0_15[1].b_O_SEND_DB_3	Ωmega 1 send register data byte 3
*.a_OMEGA_0_15[1].b_O_SEND_DB_4	Ωmega 1 send register data byte 4
*.a_OMEGA_0_15[1].b_O_SEND_DB_5	Ωmega 1 send register data byte 5
*.a_OMEGA_0_15[1].b_O_SEND_DB_6	Ωmega 1 send register data byte 6
*.a_OMEGA_0_15[1].b_O_SEND_DB_7	Ωmega 1 send register data byte 7

Address/Register	Meaning
*.a_OMEGA_0_15[1].b_O_EMPF_DB_0	Ωmega 1 receive message data byte 0
*.a_OMEGA_0_15[1].b_O_EMPF_DB_1	Ωmega 1 receive message data byte 1
*.a_OMEGA_0_15[1].b_O_EMPF_DB_2	Ωmega 1 receive message data byte 2
*.a_OMEGA_0_15[1].b_O_EMPF_DB_3	Ωmega 1 receive message data byte 3
*.a_OMEGA_0_15[1].b_O_EMPF_DB_4	Ωmega 1 receive message data byte 4
*.a_OMEGA_0_15[1].b_O_EMPF_DB_5	Ωmega 5 receive message data byte 1
*.a_OMEGA_0_15[1].b_O_EMPF_DB_6	Ωmega 1 receive message data byte 6
*.a_OMEGA_0_15[1].b_O_EMPF_DB_7	Ωmega 1 receive message data byte 7
*.a_OMEGA_0_15[1].d_O_DUMMY1	Reserved for Ωmega 1 (doubleword)
*.a_OMEGA_0_15[1].d_O_DUMMY2	Reserved for Ωmega 1 (doubleword)
•	•
•	•
•	•
*.a_OMEGA_0_15[15].b_O_STEUREG_SEND	Ωmega 15 control register send
*.a_OMEGA_0_15[15].b_O_STEUREG_EMPF	Ωmega 15 control register receive
*.a_OMEGA_0_15[15].b_O_STATREG_SEND	Ωmega 15 status register send
*.a_OMEGA_0_15[15].b_O_STATREG_SEND	Ωmega 15 status register receive
*.a_OMEGA_0_15[15].b_O_LENG_SEND	Ωmega 15 length register of send message
*.a_OMEGA_0_15[15].b_O_DUMMY0	Reserved for Ωmega 15
*.a_OMEGA_0_15[15].b_O_LENG_EMPF	Ωmega 15 length register of receive message
*.a_OMEGA_0_15[15].b_O_ABS_BRC	Ωmega 15 sender register with broadcast message
*.a_OMEGA_0_15[15].b_O_SEND_DB_0	Ωmega 15 send register data byte 0
*.a_OMEGA_0_15[15].b_O_SEND_DB_1	Ωmega 15 send register data byte 1
*.a_OMEGA_0_15[15].b_O_SEND_DB_2	Ωmega 15 send register data byte 2
*.a_OMEGA_0_15[15].b_O_SEND_DB_3	Ωmega 15 send register data byte 3
*.a_OMEGA_0_15[15].b_O_SEND_DB_4	Ωmega 15 send register data byte 4
*.a_OMEGA_0_15[15].b_O_SEND_DB_5	Ωmega 15 send register data byte 5
*.a_OMEGA_0_15[15].b_O_SEND_DB_6	Ωmega 15 send register data byte 6
*.a_OMEGA_0_15[15].b_O_SEND_DB_7	Ωmega 15 send register data byte 7
*.a_OMEGA_0_15[15].b_O_EMPF_DB_0	Ωmega 15 receive message data byte 0
*.a_OMEGA_0_15[15].b_O_EMPF_DB_1	Ωmega 15 receive message data byte 1
*.a_OMEGA_0_15[15].b_O_EMPF_DB_2	Ωmega 15 receive message data byte 2
*.a_OMEGA_0_15[15].b_O_EMPF_DB_3	Ωmega 15 receive message data byte 3
*.a_OMEGA_0_15[15].b_O_EMPF_DB_4	Ωmega 15 receive message data byte 4
*.a_OMEGA_0_15[15].b_O_EMPF_DB_5	Ωmega 15 receive message data byte 5
*.a_OMEGA_0_15[15].b_O_EMPF_DB_6	Ωmega 15 receive message data byte 6
*.a_OMEGA_0_15[15].b_O_EMPF_DB_7	Ωmega 15 receive message data byte 7
*.a_OMEGA_0_15[15].d_O_DUMMY1	Reserved for Ωmega 15 (doubleword)
*.a_OMEGA_0_15[15].d_O_DUMMY2	Reserved for Ωmega 15 (doubleword)

* Corresponds, for example, to _CAN_CTRL_OPT.

4.7.2 Communications

Sending single messages from Ωmega to Ωmega

You can send between Ωmegas messages that are between 0 and 8 bytes long. To do this, you enter the data to be sent in the send data area of the Ωmega to which the message is to be sent. You enter the number of data bytes contained in the message in the send length register. The system sends the message by entering 16#05 in the send control register. Once the message has been sent on the CAN, the CAN processor enters 16#04 as the acknowledgement in the send control register and the send status register. For the Ωmega and the CAN processor to be able to access the data area in-parallel without conflicts, the Ωmega must first enter 16#05 in the send status register and then in the send control register. Similarly, at querying of whether sending has been carried out and whether the next message can be sent, the send control register (which must be queried first) and the send status register must get the value 16#04.

Example: Ωmega 3 is to send Ωmega 5 the value 16#7E8C.

To do this, Ωmega 3 enters in Ωmega 5's send data area the value 16#8C to

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_SEND_DB_0

and the value 16#7E to

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_SEND_DB_1

In send length register

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_LENG_SEND

the system enters length 16#02.

Entering 16#05 in send status register

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_STATREG_SEND

and in send control register

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_STEUREG_SEND

sends the message (see below) to the CAN.

The system indicates that sending has been carried out by displaying in send control register

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_STEUREG_SEND

and in send status register

_CAN_CTRL_OPT.a_OMEGA_0_15[5].b_O_STATREG_SEND

16#04.

ID 1	ID 2	RTR	DLC	DB 0	DB 1
2#01000101	011	0	2	16#8C	16#7E

In general, the CAN message with a sender of Ωmega 0 - 7 look like this:

ID 1	ID 2	RTR	DLC	DB 0 - DB 7
2#0100receiver no.	Sender no.	0	Length in bytes	Data bytes 0 to 7

In general, the CAN message with a sender of Ωmega 8 - 15 look like this:

ID 1	ID 2	RTR	DLC	DB 0 - DB 7
2#0110receiver no.	Sender no - 8	0	Length in bytes	Data bytes 0 to 7

Sending broadcast messages from Ωmega to Ωmega

It is possible to send 0- to 8-byte-long messages from every Ωmega that all the other Ωmegas on the CAN bus can receive. To do this, you enter the data to be sent in the send data area of the Ωmega that sends the message. You enter the number of data bytes contained in the message in the send length register. The system sends the message by entering 16#09 in the send control register. Once the message has been sent on the CAN, the CAN processor enters 16#08 as the acknowledgement in the send control register and the send status register. For the Ωmega and the CAN processor to be able to access the data area in-parallel without conflicts, the Ωmega must first enter 16#09 in the send status register and then in the send control register. Similarly, at querying of whether sending has been carried out and whether the next message can be sent, the send control register (which must be queried first) and the send status register must get the value 16#08.

Example: Ωmega 3 is to send the value 16#A4 as a broadcast message.

Ωmega 3 enters in Ωmega 3's send data area the value 16#A4 to

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_SEND_DB_0

length 16#01 is entered in the send length register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_LENG_SEND

Entering 16#09 in send status register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STATREG_SEND

and in send control register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STEUREG_SEND

sends the message (see below) to the CAN.

The system indicates that sending has been carried out by displaying in send control register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STEUREG_SEND

and in send status register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STATREG_SEND

16#08.

ID 1	ID 2	RTR	DLC	DB 0
2#00010011	100	0	1	16#A4

In general the broadcast message looks like this:

ID 1	ID 2	RTR	DLC	DB 0 - DB 7
2#0001sender no.	100	0	Length in bytes	Data bytes 0 to 7

Receiving single messages from Ωmega to Ωmega

To be able to receive a single message from another Ωmega, you must enable reception by entering 16#03 in the data area of the Ωmega from which you want to receive the message in the receive control register. As soon as a message from this Ωmega is received on the CAN, the CAN processor enters the received data bytes in the receive data area, the length in the receive length area and 16#02 as the acknowledgement in the receive control register and in the receive status register. For the Ωmega and the CAN processor to be able to access the data area in-parallel without conflicts, the Ωmega must start the reception procedure by first entering 16#03 in the receive status register and then in the receive control register. Similarly, at querying of whether a message has been received, the receive control register (which must be queried first) and the receive status register must get the value 16#02.

If a message was received at the CAN and the receive control register is not at 16#03, the system ignores this message and does not write it in the receive data memory.

Example: Ωmega 9 is to receive a message from Ωmega 0. This message contains data byte 16#75.

To do this, Ωmega 9 enters in the data area of Ωmega 0 in receive status register

_CAN_CTRL_OPT.a_OMEGA_0_15[0].b_O_STATREG_EMPF

and in receive control register

_CAN_CTRL_OPT.a_OMEGA_0_15[0].b_O_STEUREG_EMPF

the value 16#03.

After the CAN has received the message, receive length register

_CAN_CTRL_OPT.a_OMEGA_0_15[0].b_O_LENG_EMPF

contains the length 16#01 and receive data area

_CAN_CTRL_OPT.a_OMEGA_0_15[0].b_O_EMPF_DB_0

contains data byte 16#75.

The system indicates that reception has taken place by displaying 16#02 in receive control register

_CAN_CTRL_OPT.a_OMEGA_0_15[0].b_O_STEUREG_EMPF

and receive status register

_CAN_CTRL_OPT.a_OMEGA_0_15[0].b_O_STATREG_EMPF

Receiving broadcast messages from Ωmega to Ωmega

To be able to receive a broadcast message from another Ωmega, you must enable reception by entering 16#07 in the data area of the Ωmega that wants to receive the message in the receive control register. As soon as a broadcast message is received on the CAN, the CAN processor enters the received data bytes in the receive data area, the length in the receive length area and 16#06 as the acknowledgement in the receive control register and in the receive status register. You can read the sender number of the broadcast message in the sender register. For the Ωmega and the CAN processor to be able to access the data area in-parallel without conflicts, the Ωmega must start the reception procedure by first entering 16#07 in the receive status register and then in the receive control register. Similarly, at querying of whether a message has been received, the receive control register (which must be queried first) and the receive status register must get the value 16#06.

If a broadcast message was received at the CAN and the receive control register is not at 16#07, the system ignores this message and does not write it in the receive data memory.

Example: Ωmega 3 is to receive a broadcast message. The next broadcast message comes from Ωmega 7 with the value 16#79E47F.

To do this, Ωmega 3 enters in the data area of Ωmega 3 in receive status register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STATREG_EMPF

and in receive control register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STEUREG_EMPF

the value 16#07.

After the CAN has received the message, receive length register
_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_LENG_EMPF

contains length 16#03, sender register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_ABS_BRC

contains sender number 16#07 and in receive data area in

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_EMPF_DB_0 : 16#7F

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_EMPF_DB_1 : 16#E4

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_EMPF_DB_2 : 16#79.

The system indicates that reception has taken place by displaying 16#06 in receive control register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STEUREG_EMPF

and receive status register

_CAN_CTRL_OPT.a_OMEGA_0_15[3].b_O_STATREG_EMPF

4.8 Operating the FIFO Buffer

4.8.1 Addressing

A 512-byte register area is reserved in the CAN interface module's communication RAM for using the FIFO functionality. This register area has the following meanings:

Address/Register	Meaning
*.a_FIFO_0_31[0].b_BYTE0	Read index
*.a_FIFO_0_31[0].b_BYTE1	Write index
*.a_FIFO_0_31[0].w_ID	Identifier and length information of message 0
*.a_FIFO_0_31[0].b_DATABYTE0	Data byte 0 message 0
*.a_FIFO_0_31[0].b_DATABYTE1	Data byte 1 message 0
*.a_FIFO_0_31[0].b_DATABYTE2	Data byte 2 message 0
*.a_FIFO_0_31[0].b_DATABYTE3	Data byte 3 message 0
*.a_FIFO_0_31[0].b_DATABYTE4	Data byte 4 message 0
*.a_FIFO_0_31[0].b_DATABYTE5	Data byte 5 message 0
*.a_FIFO_0_31[0].b_DATABYTE6	Data byte 6 message 0
*.a_FIFO_0_31[0].b_DATABYTE7	Data byte 7 message 0
*.a_FIFO_0_31[0].d_DWORD	Reserved for message 0
*.a_FIFO_0_31[1].b_BYTE0	Reserved for message 1
*.a_FIFO_0_31[1].b_BYTE1	Reserved for message 1
*.a_FIFO_0_31[1].w_ID	Identifier and length information of message 1
*.a_FIFO_0_31[1].b_DATABYTE0	Data byte 0 message 1
*.a_FIFO_0_31[1].b_DATABYTE1	Data byte 1 message 1
*.a_FIFO_0_31[1].b_DATABYTE2	Data byte 2 message 1
*.a_FIFO_0_31[1].b_DATABYTE3	Data byte 3 message 1
*.a_FIFO_0_31[1].b_DATABYTE4	Data byte 4 message 1
*.a_FIFO_0_31[1].b_DATABYTE5	Data byte 5 message 1
*.a_FIFO_0_31[1].b_DATABYTE6	Data byte 6 message 1
*.a_FIFO_0_31[1].b_DATABYTE7	Data byte 7 message 1
*.a_FIFO_0_31[1].d_DWORD	Reserved for message 1
•	•
•	•
•	•
*.a_FIFO_0_31[31].b_BYTE0	Reserved for message 31
*.a_FIFO_0_31[31].b_BYTE1	Reserved for message 31
*.a_FIFO_0_31[31].w_ID	Identifier and length information of message 31
*.a_FIFO_0_31[31].b_DATABYTE0	Data byte 0 message 31
*.a_FIFO_0_31[31].b_DATABYTE1	Data byte 1 message 31
*.a_FIFO_0_31[31].b_DATABYTE2	Data byte 2 message 31
*.a_FIFO_0_31[31].b_DATABYTE3	Data byte 3 message 31
*.a_FIFO_0_31[31].b_DATABYTE4	Data byte 4 message 31
*.a_FIFO_0_31[31].b_DATABYTE5	Data byte 5 message 31
*.a_FIFO_0_31[31].b_DATABYTE6	Data byte 6 message 31
*.a_FIFO_0_31[31].b_DATABYTE7	Data byte 7 message 31
*.a_FIFO_0_31[31].d_DWORD	Reserved for message 31

* Corresponds, for example, to _CAN_CTRL_OPT.

Structure of Identifier and length information:

*.a_FIFO_0_31[0].w_ID															
IDENT_1			RB	LENGTH				IDENT_0_H				IDENT_0_L			
Id2	Id1	Id0	RB	L3	L2	L1	L0	Id10	Id9	Id8	Id7	Id6	Id5	Id4	Id3

The LENGTH entry contains the number of data bytes of the CAN message.

4.8.2 Operation

CAN messages that are entered in the FIFO buffer have identifiers that the CAN interface module does not know individually.

The system enters these messages in sequence in the FIFO buffer. The system enters message 0 from *.a_FIFO_0_31[0].w_ID onwards, message 1 from *.a_FIFO_0_31[1].w_ID onwards, etc.

After each message has been entered, the system increments by 1 write index *.a_FIFO_0_31[0].b_BYTE1.

If a message is read from the FIFO buffer, the **application** must increment by 1 read index *.a_FIFO_0_31[0].b_BYTE0.

If the write index is 1 less than the read index, the system does not enter any more messages. (Note that considering 5 bits, 31 + 1 = 0).

If a message is now read, the **application** must increment the read index by 1 and enter the next received message in the FIFO buffer (as well as increment the write index by 1).

Example:

The read index is at 31, the write index is at 29.

The system now receives the next message to be entered.

The system enters this message from

_CAN_CTRL_OPT.a_FIFO_0_31[29].w_ID

onwards and increments write index

_CAN_CTRL_OPT.a_FIFO_0_31[0].b_BYTE1

from 29 (16#1D) to 30 (16#1E).

The system now receives another message to be entered.

Write index

_CAN_CTRL_OPT.a_FIFO_0_31[0].b_BYTE1

is now 1 less than read index

_CAN_CTRL_OPT.a_FIFO_0_31[0].b_BYTE0

and the system does not enter the message.

Now the application reads a message. In this connection, the **application** must increment the **read index by 1;** in this case, read index

_CAN_CTRL_OPT.a_FIFO_0_31[0].b_BYTE0

must become 0 (16#00) again.

Example: from a binary point of view, (31+1) AND 0x1F yields

	2#0001 1111	31	16#1F
+	2#0000 0001	01	16#01
	2#0010 0000	32	16#20
AND	2#0001 1111	31	16#1F
=	2#0000 0000	00	16#00

The system now receives another message to be entered. Write index

_CAN_CTRL_OPT.a_FIFO_0_31[0].b_BYTE1

is still at 30 (16#1E).

The system enters this message from

_CAN_CTRL_OPT.a_FIFO_0_31[30].w_ID

onwards and increments write index

_CAN_CTRL_OPT.a_FIFO_0_31[0].b_BYTE1

to 31 (16#1F).

5 FUNCTION BLOCKS FOR CAN

5.1 Overview

In addition to the standard functions, you can use manufacturer-defined functions if you have logged on libraries in a project.

NOTE

Logging on of libraries is described in the general help.

The following function blocks are available for CAN:

Function	Brief description
CAN_BKR_INIT	CAN BKR initialization
CAN_DIG_INPUT_INIT	Initializes the data of a CAN-DI-16-01/2 module
CAN_DIG_OUTPUT_INIT	Initializes the data of a CAN-DO-16-01/2 module
CAN_DRIVE_INIT	Initializes the data of a controller
CAN_INIT	Initializes the distributed I/O modules and controller
CAN_INIT_FB1	Is used by FB CAN_INIT
CAN_INIT_FB2	Is used by FB CAN_INIT
CAN_OBJ_READ	Reads an object of the controller
CAN_OBJ_WRITE	Writes an object of the controller
CAN_PA_LIST_READ	Reads the current reference value parameter number list for cyclical communication in the controller
CAN_PA_LIST_WRITE	Writes a new reference value parameter number list for cyclical communication in the controller
CAN_PAR_READ	Reads a parameter of the controller
CAN_PAR_WRITE	Writes a parameter of the controller
CAN_PD_COMM	Process data communication via CAN between Ωmega and controller
CAN_PE_LIST_READ	Writes the current actual value parameter number list for cyclical communication in the controller
CAN_PE_LIST_WRITE	Writes a new actual value parameter number list for cyclical communication in the controller
CAN_SD_CONTROL	Monitors requirements data communication of a drive

5.2 CAN_BKR_INIT

Description

You can use this function block for CAN to condition the data of a BKR or DSM (disk-type motor) for CAN initialization.

NOTE

This FB is used together with FB CAN_INIT.

Parameter input	Data type	Description
us_BKR_NR	USINT 0 to 15	Device number of the CAN interface module of the BKR
t_WR_TIME0	TIME 0 to 255 ms	Reference value cycle time
t_RD_TIME0	TIME 0 to 255 ms	Actual value cycle time

Parameter output	Data type	Description
_CAN_DRIVE_DATA	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the BKR with device number us_BKR_NR
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

FB CAN_BKR_INIT conditions the data that a BKR needs for CAN initialization and makes the data available at output _CAN_DRIVE_DATA (for FB CAN_INIT). Initialization of the CAN interface module with the data for the BKR is carried out by FB CAN_INIT.

Input us_BKR_NR:

At input us_BKR_NR, you state the device number of the BKR (address on the CAN bus that is set using parameter 41 in the BKR).

Input t_WR_TIME0:

At input t_WR_TIME0, you enter the cycle time in ms for reference value 0 (e. g. 20 ms → t_WR_TIME0 = 20ms).

Input t_RD_TIME0:

Input t_RD_TIME0 is currently not used.

You set the actual value cycle time (for actual value 0) using parameter 43 and 44 in the BKR.

Output _CAN_DRIVE_DATA:

At output _CAN_DRIVE_DATA, the system outputs the (conditioned) data for CAN initialization.

A variable of data type CAN_INIT_DRIVE_BMSTRUCT is connected at output _CAN_DRIVE_DATA.

NOTE

Depending on the us_BKR_NR, this variable is connected to one of inputs _CAN_DRIVE0 to _CAN_DRIVE15 of the FB CAN_INIT.

Example:

The data of the BKR with device number 12 (us_BKR_NR = 12) is transferred to FB CAN_INIT.

→ FB CAN_BKR_INIT, output _CAN_DRIVE_DATA is connected to FB CAN_INIT, input _CAN_DRIVE12.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_BKR_NR not 0 to 15)
1	Reference value cycle time (t_WR_TIME0) greater than 255 ms
2 - 7	Reserved

5.3 CAN_DIG_INPUT_INIT

Description

You can use this function block for CAN to condition the data of a CAN-DI-16-01 or CAN-DI-16-02 module for CAN initialization.

NOTE

This FB is used together with FB CAN_INIT.

Parameter input	Data type	Description
us_DI_NR	USINT 0 to 15	Device number of the CAN-DI-16-xx module
us_DI_MODE	USINT 2, 3	Operating mode of the CAN-DI-16-xx module
t_DI_TIME	TIME 0 to 255 ms [a)]	½ the cycle time of the CAN-DI-16-xx module (where us_DI_MODE = 3)
w_DI_NEG_EDGE	WORD	Send with negative edge
w_DI_POS_EDGE	WORD	Send with positive edge

[a)] See description of input t_DI_TIME

Parameter output	Data type	Description
_CAN_DI_DATA	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN-DI-16-xx module with device number us_DI_NR
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

FB CAN_DIG_INPUT_INIT conditions the data that a CAN-DI-16-xx module needs for CAN initialization and makes the data available at output _CAN_DI_DATA (for FB CAN_INIT).

Input us_DI_NR:

At input us_DI_NR you state the device number of the CAN-DI-16-xx module (address on the CAN bus that you set using the DIP switches on the CAN-DI-16-xx module).

Input us_DI_MODE:

At input us_DI_MODE, you set the operating mode of the CAN-DI-16-xx module.

us_DI_MODE	Meaning
0, 1	Reserved
2	Polling operation (acyclical requesting of the input word by the user)
3	Cyclical requesting of the input word by the CAN interface module
4 -255	Reserved

Input t_DI_TIME:

You specify the cycle time for cyclical requesting of the input word by the CAN interface module (us_DI_MODE = 3) at input t_DI_TIME. You must set a value between 1 and 255.

The cycle time is calculated as follows:

Cycle time = t_DI_TIME * 2 ms

Example:

t_DI_TIME = 10 ms $\rightarrow$ cycle time = 20 ms

NOTE

The system does not check the values at input t_DI_TIME.

In polling operation (acyclical requesting of the input word by the user), t_DI_TIME must be = 0.

Input w_DI_NEG_EDGE:

At input w_DI_NEG_EDGE, you can specify the inputs of the CAN-DI-16-xx module at which a negative edge triggers sending of the input word. The default setting is 16#0000, i.e. the input word is not sent with negative edges. [1)]

Input w_DI_POS_EDGE:

At input w_DI_POS_EDGE, you can specify the inputs of the CAN-DI-16-xx module at which a positive edge triggers sending of the input word. The default setting is 16#0000, i.e. the input word is not sent with positive edges. [1)]

Output _CAN_DI_DATA:

At output _CAN_DI_DATA, the system outputs the (conditioned) data for CAN initialization.

A variable of data type CAN_INIT_IO_BMSTRUCT is connected at output _CAN_DI_DATA.

---

1) With _MODE = 3, this function causes additional CAN bus loading.

NOTE

Depending on the us_DI_NR, this variable is connected to one of inputs _CAN_DIG_INOUT0 to _CAN_DIG_INOUT15.

Example:

The data of CAN-DI-16-xx module 12 (us_DI_NR = 12) is transferred to FB CAN_INIT.

→ FB CAN_DIG_INPUT_INIT, output _CAN_DI_DATA is connected to FB CAN_INIT, input _CAN_DIG_INOUT12.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DI_NR not 0 to 15)
1	Invalid operating mode (not 2, 3)
2	No cycle time where us_DI_MODE = 3
3	You stated a cycle time where us_DI_MODE ≠ 3
4 - 7	Reserved

5.4 CAN_DIG_OUTPUT_INIT

Description

You can use this function block for CAN to condition the data of a CAN-DI-16-01 or CAN-DI-16-02 module for CAN initialization.

NOTE

This FB is used together with FB CAN_INIT.

Parameter input	Data type	Description
us_DO_NR	USINT 0 to 15	Device number of the CAN-DO-16-xx module
us_DO_MODE	USINT 2, 3	Operating mode of the CAN-DO-16-xx module
t_DO_TIME	TIME0 to 255 ms [a)]	½ the cycle time of the CAN-DO-16-xx module (where us_DO_MODE = 3)
w_DO_OUT_EN	WORD	Enables the outputs of the CAN-DO-16-xx module

a) See description of input t_DO_TIME

Parameter output	Data type	Description
_CAN_DO_DATA	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN-DO-16-xx module with device number us_DO_NR
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

FB CAN_DO_INPUT_INIT conditions the data that a CAN-DO-16-xx module needs for CAN initialization and makes the data available at output _CAN_DO_DATA (for FB CAN_INIT).

Input us_DO_NR:

At input us_Do_NR you state the device number of the CAN-DO-16-xx module (address on the CAN bus that you set using the DIP switches on the CAN-DO-16-xx module).

Input us_DO_MODE:

At input us_DO_MODE, you set the operating mode of the CAN-DO-16-xx module.

us_DO_MODE	Meaning
0, 1	Reserved
2	Polling operation (acyclical sending of the output word by the user)
3	Cyclical sending of the output word by the CAN interface module
4 - 255	Reserved

Input t_DO_TIME:

You specify the cycle time for cyclical sending of the output word by the CAN interface module (us_DO_MODE = 3) at input t_DO_TIME. You must set a value between 1 and 255.

The cycle time is calculated as follows:

Cycle time = t_DO_TIME * 2 ms

Example:

t_DO_TIME = 10 ms $\rightarrow$ cycle time = 20 ms

NOTE

The system does not check the values at input t_DI_TIME.

In polling operation (acyclical sending of the output word by the user), t_DO_TIME must be = 0.

Input w_DO_OUT_EN:

At input w_DO_OUT_EN, you can state which of the CAN-DO-16-xx module's outputs are released for switching. The default setting is 16#FFFF, i. e. all the outputs are released for switching.

Output _CAN_DO_DATA:

At output _CAN_DO_DATA, the system outputs the (conditioned) data for CAN initialization.

A variable of data type CAN_INIT_IO_BMSTRUCT is connected at output _CAN_DO_DATA.

NOTE

Depending on the us_DO_NR, this variable is connected to one of inputs _CAN_DIG_INOUT0 to _CAN_DIG_INOUT15.

Example:

The data of CAN-DO-16-xx module 7 (us_DI_NR = 7) is transferred to FB CAN_INIT.

→ FB CAN_DIG_OUTPUT_INIT, output _CAN_DO_DATA is connected to FB CAN_INIT, input _CAN_DIG_INOUT7.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Device number (us_DO_NR not 0 to 15)
1	Invalid operating mode (not 2, 3)
2	No cycle time where us_DO_MODE = 3
3	You stated a cycle time where us_DO_MODE ≠ 3
4 - 7	Reserved

5.5 CAN_DRIVE_INIT

Description

You can use this function block for CAN to condition the data of a controller (Drive Controller) for CAN initialization.

NOTE

This FB is used together with FB CAN_INIT.

Parameter input	Data type	Description
us_DRIVE_NR	USINT 0 to 15	Device number of the CAN interface module of the controller
u_WR_PAR_NR0	UINT	Reference value parameter number 0
us_WR_FORMAT0	USINT 0, 1	Reference value format 0
t_WR_TIME0	TIME 0 to 255 ms	Reference value cycle time 0
u_WR_PAR_NR1	UINT	Reference value parameter number 1
us_WR_FORMAT1	USINT 1, 2	Reference value format 1
t_WR_TIME1	TIME 0 to 255 ms	Reference value cycle time 1
u_WR_PAR_NR2	UINT	Reference value parameter number 2
us_WR_FORMAT2	USINT 1, 2	Reference value format 2
t_WR_TIME2	TIME 0 to 255 ms	Reference value cycle time 2
u_WR_PAR_NR3	UINT	Reference value parameter number 3
us_WR_FORMAT3	USINT 1, 2	Reference value format 3
t_WR_TIME3	TIME 0 to 255 ms	Reference value cycle time 3
u_RD_PAR_NR0	UINT	Actual value parameter number 0
us_RD_FORMAT0	USINT 0, 1	Actual value format 0
t_RD_TIME0	TIME 0 to 255 ms	Actual value cycle time 0
u_RD_PAR_NR1	UINT	Actual value parameter number 1
us_RD_FORMAT1	USINT 1, 2	Actual value format 1
t_RD_TIME1	TIME 0 to 255 ms	Actual value cycle time 1
u_RD_PAR_NR2	UINT	Actual value parameter number 2
us_RD_FORMAT2	USINT 1, 2	Actual value format 2

Parameter input	Data type	Description
t_RD_TIME2	TIME 0 to 255 ms	Actual value cycle time 2
u_RD_PAR_NR3	UINT	Actual value parameter number 3
us_RD_FORMAT3	USINT 1, 2	Actual value format 3
t_RD_TIME3	TIME 0 to 255 ms	Actual value cycle time 3

Parameter output	Data type	Description
_CAN_DRIVE_DATA	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number us_DRIVE_NR
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

FB CAN_DRIVE_INIT conditions the data that a controller needs for CAN initialization and makes the data available at output _CAN_DRIVE_DATA (for FB CAN_INIT).

Input us_DRIVE_NR:

At input us_DRIVE_NR you state the device number of the controller (address on the CAN bus that you set using the DIP switches on the controller's CAN interface module).

Inputs u_WR_PAR_NR0, us_WR_FORMAT0, t_WR_TIME0:

At input u_WR_PAR_NR0, you state the parameter number of reference value 0.

At input us_WR_FORMAT0, you state the format of reference value 0.

NOTE

Reference value 0 may have the format byte or word.
Reference values 1, 2 and 3 may have the format word or doubleword.

Format specification: us_WR_FORMAT0 = 0: format byte (reference value 0)

us_WR_FORMAT0 = 1: format word (reference values 0, 1, 2 and 3)

us_WR_FORMAT0 = 2: format doubleword(reference values 1, 2 and 3)

At input t_WR_TIME0, you state the cycle time for reference value 0 in ms (e.g. 20 ms → t_WR_TIME0 = 20ms).

Inputs u_WR_PAR_NR1, us_WR_FORMAT1, t_WR_TIME1:

At input u_WR_PAR_NR1, you state the parameter number of reference value 1.

At input us_WR_FORMAT1, you state the format of reference value 1.

At input t_WR_TIME1, you enter the cycle time in ms of reference value 1.

Inputs u_WR_PAR_NR2, us_WR_FORMAT2, t_WR_TIME2:

At input u_WR_PAR_NR2, you state the parameter number of reference value 2.

At input us_WR_FORMAT2, you state the format of reference value 2.

At input t_WR_TIME2, you enter the cycle time in ms of reference value 2.

Inputs u_WR_PAR_NR3, us_WR_FORMAT3, t_WR_TIME3:

At input u_WR_PAR_NR3, you state the parameter number of reference value 3.

At input us_WR_FORMAT3, you state the format of reference value 3.

At input t_WR_TIME3, you enter the cycle time in ms of reference value 3.

Inputs u_RD_PAR_NR0, us_RD_FORMAT0, t_RD_TIME0:

At input u_WR_RD_NR0, you state the parameter number of actual value 0.

At input us_RD_FORMAT0, you state the format of actual value 0.

NOTE

Actual value 0 may have the format byte or word.
Actual values 1, 2 and 3 may have the format word or doubleword.

Format specification: us_RD_FORMAT0 = 0: format byte (actual value 0)

us_RD_FORMAT0 = 1: format word (actual values 0, 1, 2 and 3)

us_RD_FORMAT0 = 2: format doubleword(actual values 1, 2 and 3)

At input t_RD_TIME0, you state the cycle time for actual value 0 in ms (e.g. 20 ms → t_WR_TIME0 = 20ms).

Inputs u_RD_PAR_NR1, us_RD_FORMAT1, t_RD_TIME1:

At input u_RD_PAR_NR1, you state the parameter number of actual value 1.

At input us_RD_FORMAT1, you state the format of actual value 1.

At input t_RD_TIME1, you enter the cycle time in ms of actual value 1.

Inputs u_RD_PAR_NR2, us_RD_FORMAT2, t_RD_TIME2:

At input u_RD_PAR_NR2, you state the parameter number of actual value 2.

At input us_RD_FORMAT2, you state the format of actual value 2.

At input t_RD_TIME2, you enter the cycle time in ms of actual value 2.

Inputs u_RD_PAR_NR3, us_RD_FORMAT3, t_RD_TIME3:

At input u_RD_PAR_NR3, you state the parameter number of actual value 3.

At input us_RD_FORMAT3, you state the format of actual value 3.

At input t_RD_TIME3, you enter the cycle time in ms of actual value 3.

Output _CAN_DRIVE_DATA:

At output _CAN_DRIVE_DATA, the system outputs the (conditioned) data for CAN initialization.

A variable of data type CAN_INIT_DRIVE_BMSTRUCT is connected at output _CAN_DRIVE_DATA.

NOTE

Depending on the us_DRIVE_NR, this variable is connected to one of inputs _CAN_DRIVE0 to _CAN_DRIVE15 of the FB CAN_INIT.

Example:

The data of the controller with CAN interface module and device number 7 (us_DRIVE_NR = 7) is transferred to FB CAN_INIT.

→ FB CAN_DRIVE_INIT, output _CAN_DRIVE_DATA is connected to FB CAN_INIT, input _CAN_DRIVE7.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Invalid format of reference value 0 (us_WR_FORMAT0 not 0 or 1)
2	Invalid format of reference value 1, 2 or 3 (us_WR_FORMAT1, us_WR_FORMAT2 and/or us_WR_FORMAT3 not 1 or 2)
3	Invalid format of actual value 0 (us_RD_FORMAT0 not 0 or 1)
4	Invalid format of reference value 1, 2 or 3 (us_RD_FORMAT1, us_RD_FORMAT2 and/or us_RD_FORMAT3 not 1 or 2)
5	At least one reference value or actual value cycle time is greater than 255 ms (t_WR_TIMEx, t_RD_TIMEx > 255 ms; x = 0, 1, 2, 3)
6 - 7	Reserved

5.6 CAN_INIT

Description

You can use this function block for CAN to initialize a CAN interface module on the Ωmega Drive-Line II.

In this connection, you can set up communication to CAN-DI-16-xx modules and CAN-DO-16-xx modules (a maximum of 16 in the following CAN IO module) and to controllers with a CAN interface module (a maximum of 16 in the following controller) on the CAN bus.
(CAN-DI-16-xx: digital input module, CAN-DO-16-xx: digital output module)

NOTE

FB CAN_INIT uses library BM_TYPES_20bd00 or above and firmware library Bit_UTIL.

Parameter input	Data type	Description
_BASE_INIT	CAN_INIT_BMSTRUCT	Initialization data for the CAN interface module
_BASE_CTRL	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_NR_OF_IO_TO_INIT	USINT 0, 1 to 16	Number of CAN IO modules on the CAN bus
us_MAX_IO_NR_COMM	USINT 0, 1 to 16	(Maximum device number + 1) of the CAN IO modules on the CAN bus
us_MAX_DRIVE_NR_COMM	USINT 0, 1 to 16	(Maximum device number + 1) of the controllers with a CAN interface module on the CAN bus
us_MAX_OMEGA_NR_COMM	USINT0, 1 to 16	(Maximum device number + 1) of the Ωmegas with a CAN interface module on the CAN bus
us_NR_OF_OMEGAS	USINT 0, 1 to 16	Number of Ωmegas with a CAN interface module on the CAN bus
x_RECAL_SL	BOOL	Recalibration slave
b_ACCEPT_MASK	BYTE	Acceptance mask
b_ACCEPT_CODE	BYTE	Acceptance code
b_BIT_TIMING0	BYTE	Bus timing
b_BIT_TIMING1	BYTE	Bus timing
us_BAUDRATE	USINT 0, 1 to 6	Baud Rate
_CAN_DIG_INOUT0	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 0
_CAN_DIG_INOUT1	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 1
_CAN_DIG_INOUT2	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 2
_CAN_DIG_INOUT3	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 3

Parameter input	Data type	Description
_CAN_DIG_INOUT4	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 4
_CAN_DIG_INOUT5	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 5
_CAN_DIG_INOUT6	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 6
_CAN_DIG_INOUT7	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 7
_CAN_DIG_INOUT8	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 8
_CAN_DIG_INOUT9	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 9
_CAN_DIG_INOUT10	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 10
_CAN_DIG_INOUT11	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 11
_CAN_DIG_INOUT12	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 12
_CAN_DIG_INOUT13	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 13
_CAN_DIG_INOUT14	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 14
_CAN_DIG_INOUT15	CAN_INIT_IO_BMSTRUCT	Initialization data for the CAN IO module with device number 15
_CAN_DRIVE0	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 0
_CAN_DRIVE1	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 1
_CAN_DRIVE2	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 2
_CAN_DRIVE3	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 3
_CAN_DRIVE4	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 4
_CAN_DRIVE5	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 5
_CAN_DRIVE6	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 6
_CAN_DRIVE7	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 7
_CAN_DRIVE8	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 8
_CAN_DRIVE9	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 9
_CAN_DRIVE10	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 10
_CAN_DRIVE11	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 11
_CAN_DRIVE12	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 12
_CAN_DRIVE13	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 13

Parameter input	Data type	Description
_CAN_DRIVE14	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 14
_CAN_DRIVE15	CAN_INIT_DRIVE_BMSTRUCT	Initialization data for the controller with device number 15
t_TIME	TIME	Monitoring time

Parameter output	Data type	Description
_BASE_INIT	CAN_INIT_BMSTRUCT	Initialization data for the CAN interface module
_BASE_CTRL	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
w_ERR_IO	WORD	Error word of CAN IO modules
w_ERR1_DRIVES	WORD	Error word 1 controller
w_ERR2_DRIVES	WORD	Error word 2 controller
w_ERR	WORD	Error word general
x_ERR	BOOL	Error bit
w_OK_IO	WORD	OK word of CAN IO modules
w_OK_DRIVES	WORD	OK word of controller
x_OK	BOOL	OK bit

FB CAN_INIT carries out initialization of a CAN interface module on the Ωmega Drive-Line II and communication to CAN IO modules, Ωmegas with a CAN interface module and controllers with a CAN interface module. While doing this, the system processes the following steps:

- initialization of the CAN interface module on the Ωmega Drive-Line II
- initialization of the CAN-DI-16-xx CAN IO modules. The initialization data is stated at FBs CAN_DIG_INPUT_INIT.
- initialization of the CAN-DO-16-xx CAN IO modules. The initialization data is stated at FBs CAN_DIG_OUTPUT_INIT.
- initialization of the controllers with a CAN interface module. The initialization data is stated at FBs CAN_DRIVE_INIT.
- checking communication to the controllers with a CAN interface module.
- checking communication to the CAN IO modules.

Initializing the CAN interface module on the Ωmega Drive-Line II

Input/output _BASE_INIT:

At _BASE_INIT, you must connect a global variable of data type CAN_INIT_BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_INIT_OPT AT %MB3.3000000 : CAN_INIT_BMSTRUCT;
```

Where:

`_CAN_INIT_OPT`	is the variable name with the data type short designation "_" for STRUCT

`CAN_INIT_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input/output _BASE_CTRL:

At _BASE_CTRL, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_NR_OF_IO_TO_INIT:

At input us_NR_OF_IO_TO_INIT, you state the number of CAN IO modules to be initialized. If the input is not assigned, this yields the presetting us_NR_OF_IO_TO_INIT = 0.

If the number of CAN IO modules is not in the range 0 (none), 1 to 16, the system sets bit 1 in error word w_ERR.

Input us_MAX_IO_NR_COMM:

This input tells the CAN interface module up to which CAN IO module it processes data.

At input us_MAX_IO_NR_COMM, you specify the highest device number of a CAN IO module on the CAN bus + 1.

If the input is not assigned, this yields the presetting us_MAX_IO_NR_COMM = 0, i.e. no CAN IO modules process data.

If the highest device number of a CAN IO module on the CAN bus + 1 is not in the range 0 (none), 1 to 16, the system sets bit 2 in error word w_ERR.

Input us_MAX_DRIVE_NR_COMM:

This input tells the CAN interface module up to which controller it processes data.

At input us_MAX_DRIVE_NR_COMM, you specify the highest device number of a controller on the CAN bus + 1.

If the input is not assigned, this yields the presetting us_MAX_DRIVE_NR_COMM = 0, i.e. no controllers process data.

If the highest device number of a controller on the CAN bus+1 is not in the range 0 (none), 1 to 16, the system sets bit 3 in error word w_ERR.

Input us_MAX_OMEGA_NR_COMM:

This input tells the CAN interface module up to which Ωmega module it processes data.

At input us_MAX_IO_OMEGA_COMM, you specify the highest device number of an Ωmega on the CAN bus + 1.

If the input is not assigned, this yields the presetting us_MAX_OMEGA_NR_COMM = 0, i.e. no Ωmegas process data.

If the highest device number of an Ωmega on the CAN bus + 1 is not in the range 0 (none), 1 to 16, the system sets bit 4 in error word w_ERR.

Input us_NR_OF_OMEGAS:

At input us_NR_OF_OMEGAS, you state how many Ωmegas (includung the master) are connected on the CAN bus. This is for checking the number of nodes on the CAN bus. If the input is not assigned, this yields the presetting us_NR_OF_OMEGAS = 0. If the number of Ωmegas is greater than 16, the system sets bit 5 in error word w_ERR.

Input x_RECAL_SL:

If there are several Ωmegas on the CAN bus, you must specify which ones are recalibration slave which one is the Ωmega recalibration master, i. e. which Ωmega cyclically sends the recalibration message frame.

At input x_RECAL_SL, you enter FALSE for the recalibration master and x_RECAL_SL = TRUE for all the recalibration slaves.

The default setting is x_RECAL_SL = FALSE, i. e. the CAN interface module on the Ωmega that is to be initialized is the recalibration master.

Inputs b_ACCEPT_MASK, b_ACCEPT_CODE:

At inputs b_ACCEPT_MASK and b_ACCEPT_CODE, you can set the acceptance filter of the CAN interface module. For more information on this topic, refer to the CAN-M-01 technical description and user guide. If the inputs are not assigned, this yields presettings b_ACCEPT_MASK = 16#FF and b_ACCEPT_CODE = 16#FF, i. e. the system takes into account all the objects.

Inputs b_BIT_TIMING0, b_BIT_TIMING1, us_BAUDRATE:

At input us_BAUDRATE, you set the baud rate for the CAN bus. You must set the maximum baud rate that all the nodes on the CAN bus can "understand".

The bus timing is calculated for six different baud rates and it is transferred to the CAN interface module by FB CAN_INIT.

The default setting is us_BAUDRATE = 3, i. e. a baud rate of 125 kbps is set when us_BAUDRATE is not assigned.

Baud Rate	us_BAUDRATE	b_BIT_TIMING0	b_BIT_TIMING1
	0	Individual	Individual
20 kbps	1	16#53	16#2F
50 kbps	2	16#47	16#2F
125 kbps	3	16#03	16#1C
250 kbps	4	16#01	16#1C
500 kbps	5	16#00	16#1C
1000 kbps	6	16#00	16#14

If you state at initialization that CAN IO modules are located on the CAN bus and the baud rate is set to more than 250 kbps, the system sets bit 10 in error word w_ERR.

This means that if input us_NR_OF_IO_TO_INIT ≠ 0 **or** us_MAX_IO_NR_COMM ≠ 0 **and** us_BAUDRATE > 4, the system sets bit 10 in error word w_ERR.

If value us_BAUDRATE > 6, the system sets bit 6 in error word w_ERR.

At inputs b_BIT_TIMING0 and b_BIT_TIMING1, you can set individually the bus timing of the CAN interface module. For more information on this topic, refer to the CAN-M-01 technical description and user guide.

The system applies the settings of these inputs when input us_BAUDRATE = 0.

The system ignores the settings of these inputs when input us_BAUDRATE is assigned with a value from 1 to 6.

NOTE

The system **does not** apply the values at inputs b_BIT_TIMING0 and b_BIT_TIMING1 when input us_BAUDRATE ≠ 0.

The system **only** applies the values at inputs b_BIT_TIMING0 and b_BIT_TIMING1 when input us_BAUDRATE = 0.

You can connect a maximum of 32 nodes to the CAN bus. One Ωmega must be the recalibration master. You can connect 31 nodes to the CAN bus (in addition to the recalibration master). The system does not check the number of nodes on the CAN bus during initialization.

If the number of CAN IO modules, controllers and Ωmegas (includung the recalibration master) is outside the range 0 (none), 1 to 32, the system sets bit 7 in error word w_ERR.

If the system enters an error in error word w_ERR due to the input assignment, checking of communication to the CAN IO modules and the controllers is not carried out, output x_OK stays FALSE, the system sets bit 11 in error word w_ERR and error bit x_ERR. Initialization of the CAN interface module and communication to the CAN IO modules and the controllers was not carried out correctly. On the basis of the information in error word w_ERR (see below), you must look for and eliminate the error.

Initialization of CAN-DI-16-xx and CAN-DO-16-xx CAN IO modules:

Inputs _CAN_DIG_INOUT0 to _CAN_DIG_INOUT15:

At inputs _CAN_DIG_INOUT0 to _CAN_DIG_INOUT15, you connect the initialization data of the maximum of 16 CAN IO modules. The initialization data is of data type CAN_INIT_IO_BMSTRUCT and is available in the respective FB CAN_DIG_INPUT_INIT at output _CAN_DI_DATA or at the respective FB CAN_DIG_OUTPUT_INIT at output _CAN_DO_DATA.

Initialization of the controllers with a CAN interface module:

Inputs _CAN_DRIVE0 to _CAN_DRIVE15:

At inputs _CAN_DIG_DRIVE0 to _CAN_DIG_DRIVE15, you connect the initialization data of the maximum of 16 controllers. The initialization data is of data type CAN_INIT_DRIVE_BMSTRUCT and is available at the respective FB CAN_DRIVE_INIT (or CAN_BKR_INIT) at output _CAN_DRIVE_DATA.

Input t_TIME:

At input t_TIME, you state the monitoring time. The default setting is t_TIME = 10s. If initialization of the CAN interface module and of communication to the CAN IO modules, Ωmegas with a CAN interface module and the controllers with a CAN interface module has not been completed successfully during this time, the system sets bit 8 in error word w_ERR.

A global variable of data type CAN_INIT_DATA_BMSTRUCT is connected at output _CAN_DATA.

Example:

```
_CAN_INIT_DATA_OPT : CAN_INIT_DATA_BMSTRUCT;
```

Where:

`_CAN_INIT_DATA_OPT`	is variable name with the data type short designation "_" for STRUCT
`CAN_INIT_DATA_BMSTRUCT`	is the data type

This variable contains the initialization data of the controllers with a CAN interface module, the Ωmegas with a CAN interface module and the CAN IO modules of this CAN interface module, and it is needed by some FBs for communication.

Checking communication to the controllers with a CAN interface module:

Outputs w_OK_DRIVES, w_ERR1_DRIVES, w_ERR2_DRIVES:

The system checks communication to the controllers by requesting the status word. If the status word has been received, the system transfers if necessary the parameter numbers of the specified and actual values and the actual value cycle times to the controller.

If all the transfers were successful, the system sets the bit in OK word w_OK_DRIVES that corresponds to the controller's device number on the CAN bus. For the controller with device number 0, the system sets bit 0 in w_OK_DRIVES; for the controller with device number 1, it sets bit 1 in w_OK_DRIVES, etc.

If there was an error, the system sets the bit in error word w_ERR1_DRIVES or w_ERR2_DRIVES that corresponds to the controller's device number on the CAN bus.

For the controller with device number 0, the system sets bit 0 in w_ERR1_DRIVES or w_ERR2_DRIVES; for the controller with device number 1, it sets bit 1 in w_ERR1_DRIVES or w_ERR2_DRIVES, etc.

w_ERR1_DRIVES:	Timeout when requesting the status word, transferring the parameter lists, the actual value cycle time or at reference value enabling.
w_ERR2_DRIVES:	Communications error when transferring the parameter lists, the actual value cycle time or at reference value enabling

Checking communication to the CAN IO modules:

Outputs w_OK_IO, w_ERR_IO:

When checking communication of the CAN IO modules, the system requests the input word from the initialized CAN-DI-16-xx modules. If the input word has been received, the system sets the bit in OK word w_OK_IO that corresponds to the CAN-DI-16-xx module's device number on the CAN bus.

For the CAN-DI-16-xx module with device number 0, the system sets bit 0 in w_OK_IO; for the CAN-DI-16-xx with device number 1, it sets bit 1 in w_OK_IO, etc.

If there was an error, the system sets the bit in error word w_ERR_IO that corresponds to the CAN-DI-16-xx module's device number on the CAN bus.

For the CAN-DI-16-xx module with device number 0, the system sets bit 0 in w_ERR_IO; for the CAN-DI-16-xx with device number 1, it sets bit 1 in w_ERR_IO, etc.

The system sends an output word to the initialized CAN-DO-16-xx modules. Provision of the output word on the CAN bus is acknowledged. Then, the system sets the bit in OK word w_OK_IO that corresponds to the CAN-DO-16-xx module's device number on the CAN bus.

For the CAN-DO-16-xx module with device number 0, the system sets bit 0 in w_OK_IO; for the CAN-DO-16-xx with device number 1, it sets bit 1 in w_OK_IO, etc.

If there was an error, the system sets the bit in error word w_ERR_IO that corresponds to the CAN-DO-16-xx module's device number on the CAN bus.

For the CAN-DO-16-xx module with device number 0, the system sets bit 0 in w_ERR_IO; for the CAN-DO-16-xx with device number 1, it sets bit 1 in w_ERR_IO, etc.

Output x_OK:

The system indicates with x_OK = TRUE that initialization of the CAN interface module on the Ωmega Drive-Line II and communication to the CAN IO modules and controllers was completed successfully.

Outputs x_ERR, w_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs the corresponding error words. In this case output x_OK stays FALSE.

Error word w_ERR:

Bit No.	Error
0	Timeout handshaking with the CAN interface module
1	us_NR_OF_IO_TO_INIT greater than 16
2	us_MAX_IO_NR_COMM greater than 16
3	us_MAX_DRIVE_NR_COMM greater than 16
4	us_MAX_OMEGA_NR_COMM greater than 16
5	us_NR_OF_OMEGAS greater than 16
6	us_BAUDRATE greater than 6
7	Number of nodes on CAN bus greater than 32
8	Timeout (initialization of CAN interface module and communication to CAN IO modules and controllers)
9	Error with initialization data of at least one CAN IO module
10	us_BAUDRATE > 4 if us_NR_OF_IO_TO_INIT ≠ 0 or us_MAX_IO_NR_COMM ≠ 0
11	Initialization of CAN interface module not completed
12 - 15	Reserved

Error word w_ERR_IO:

Bit No.	Meaning
0	CAN IO module with device number 0: error
1	CAN IO module with device number 1: error
2	CAN IO module with device number 2: error
3	CAN IO module with device number 3: error
4	CAN IO module with device number 4: error
5	CAN IO module with device number 5: error
6	CAN IO module with device number 6: error
7	CAN IO module with device number 7: error
8	CAN IO module with device number 8: error
9	CAN IO module with device number 9: error
10	CAN IO module with device number 10: error
11	CAN IO module with device number 11: error
12	CAN IO module with device number 12: error
13	CAN IO module with device number 13: error
14	CAN IO module with device number 14: error
15	CAN IO module with device number 15: error

Error words w_ERR1_DRIVES and w_ERR2_DRIVES:

Bit No.	Meaning
0	Controller with device number 0: error
1	Controller with device number 1: error
2	Controller with device number 2: error
3	Controller with device number 3: error
4	Controller with device number 4: error
5	Controller with device number 5: error
6	Controller with device number 6: error
7	Controller with device number 7: error
8	Controller with device number 8: error
9	Controller with device number 9: error
10	Controller with device number 10: error
11	Controller with device number 11: error
12	Controller with device number 12: error
13	Controller with device number 13: error
14	Controller with device number 14: error
15	Controller with device number 15: error

OK word w_OK_IO:

Bit No.	Meaning
0	CAN IO module with device number 0: OK
1	CAN IO module with device number 1: OK
2	CAN IO module with device number 2: OK
3	CAN IO module with device number 3: OK
4	CAN IO module with device number 4: OK
5	CAN IO module with device number 5: OK
6	CAN IO module with device number 6: OK

Bit No.	Meaning
7	CAN IO module with device number 7: OK
8	CAN IO module with device number 8: OK
9	CAN IO module with device number 9: OK
10	CAN IO module with device number 10: OK
11	CAN IO module with device number 11: OK
12	CAN IO module with device number 12: OK
13	CAN IO module with device number 13: OK
14	CAN IO module with device number 14: OK
15	CAN IO module with device number 15: OK

OK word w_OK_DRIVES:

Bit No.	Meaning
0	Controller with device number 0: OK
1	Controller with device number 1: OK
2	Controller with device number 2: OK
3	Controller with device number 3: OK
4	Controller with device number 4: OK
5	Controller with device number 5: OK
6	Controller with device number 6: OK
7	Controller with device number 7: OK
8	Controller with device number 8: OK
9	Controller with device number 9: OK
10	Controller with device number 10: OK
11	Controller with device number 11: OK
12	Controller with device number 12: OK
13	Controller with device number 13: OK
14	Controller with device number 14: OK
15	Controller with device number 15: OK

5.7 CAN_INIT_FB1

Description

This function block for CAN is used by FB CAN_INIT.

5.8 CAN_INIT_FB2

Description

This function block for CAN is used by FB CAN_INIT.

5.9 CAN_OBJ_READ

Description

You can use this function block for CAN to read an object of a controller with a CAN interface module.

NOTE

FB CAN_OBJ_READ uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
w_OBJ_NR	WORD	Object number
us_OBJ_FORMAT	USINT 1, 2	Object format
us_OBJ_SUBINDEX	USINT	Object subindex
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
ud_OBJ_VALUE	UDINT	Read object value
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_OBJ_READ transfers with the object number (w_OBJ_NR), the object format (us_OBJ_FORMAT) and the object subindex (us_OBJ_SUBINDEX) a read object job to the controller containing the device number us_DRIVE_NR. The controller processes the read object job and returns the result of communication. The contents of the object are output at output ud_OBJ_VALUE.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_DRIVE_NR:

At input us_DRIVE_NR you state the device number of the controller on the CAN bus to which the read object job is to be transferred.

Input u_OBJ_NR:

You state the object number for the read object job at input u_OBJ_NR.

Input us_OBJ_FORMAT:

At input us_OBJ_FORMAT, you state the format of the object value to be read. us_OBJ_FORMAT = 1 means word format, us_OBJ_FORMAT = 2 means doubleword format. The default setting is us_OBJ_FORMAT = 1, i. e. word format.

Input us_OBJ_SUBINDEX:

You set the object subindex at input us_OBJ_SUBINDEX. The default setting is us_OBJ_SUBINDEX = 0, i. e. reading the entire object.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY = FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_OBJ_READ by means of x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Output ud_OBJ_VALUE:

The read object value is output at output ud_OBJ_VALUE.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (read object)
2	Invalid format (us_OBJ_FORMAT greater than 2)
3 - 7	Reserved

5.10 CAN_OBJ_WRITE

Description

You can use this function block for CAN to write to an object of the controller.

NOTE

FB CAN_OBJ_WRITE uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
w_OBJ_NR	WORD	Object number
us_OBJ_FORMAT	USINT 1, 2	Object format
us_OBJ_SUBINDEX	USINT	Object subindex
ud_OBJ_VALUE	UDINT	Object value to be sent
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_OBJ_WRITE transfers with the object number (w_OBJ_NR), the object format (us_OBJ_FORMAT), the object subindex (us_OBJ_SUBINDEX) and the object value (ud_OBJ_VALUE) a write object job to the controller containing the device number us_DRIVE_NR. The controller processes the write object job and returns the result of communication.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_DRIVE_NR:

At input us_DRIVE_NR you state the device number of the controller on the CAN bus to which the write object job is to be transferred.

Input w_OBJ_NR:

You state the object number for the write object job at input w_OBJ_NR.

Input us_OBJ_FORMAT:

At input us_OBJ_FORMAT, you state the format of the object value to be sent. us_OBJ_FORMAT = 1 means word format, us_OBJ_FORMAT = 2 means doubleword format. The default setting is us_OBJ_FORMAT = 1, i. e. word format.

Input us_OBJ_SUBINDEX:

You set the object subindex at input us_OBJ_SUBINDEX. The default setting is us_OBJ_SUBINDEX = 0, i. e. writing the entire object.

Input ud_OBJ_VALUE:

The object value to be written is connected at input ud_OBJ_VALUE.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY=FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_OBJ_WRITE by means of x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (write object)
2	Invalid format (> 2)
3 - 7	Reserved

5.11 CAN_PA_LIST_READ

Description

You can use this function block for CAN to read the current reference value parameter number list (PA data list) for cyclical communication in the controller.

In general, the reference value parameter numbers are stated at FB CAN_DRIVE_INIT and sent to the controller via FB CAN_INIT.

NOTE

FB CAN_PA_LIST_READ uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
u_WR_PAR_NR0	UINT	Reference value parameter number 0
u_WR_PAR_NR1	UINT	Reference value parameter number 1
u_WR_PAR_NR2	UINT	Reference value parameter number 2
u_WR_PAR_NR3	UINT	Reference value parameter number 3
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PA_LIST_READ transfers a read PA data job to the controller with device number "us_DRIVE_NR". The controller processes the read PA data job and returns the result of communication. The system outputs the contents of the PA data list at outputs u_WR_PAR_NR0 to u_WR_PAR_NR3.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_DRIVE_NR:

At input us_DRIVE_NR you state the device number of the controller on the CAN bus to which the read PA data job is to be transferred.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY = FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_OBJ_READ by means of x_RESET = TRUE.

Input x_RESET:

You can reset the FB using input x_RESET = TRUE.

Outputs u_WR_PAR_NR0 to u_WR_PAR_NR3:

The system outputs the contents of the controller's PA data list, i.e. reference value parameter numbers 0 to 3, at outputs u_WR_PAR_NR0 to u_WR_PAR_NR3.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (read PA data)
2 - 7	Reserved

5.12 CAN_PA_LIST_WRITE

Description

You can use this function block for CAN to set a new reference value parameter number list (PA data list) for cyclical communication in the controller.

In general, the reference value parameter numbers, reference value formats and the reference value cycle times are stated at FB CAN_DRIVE_INIT and sent to the controller via FB CAN_INIT.

FB CAN_PA_LIST_WRITE makes it possible to change at a later date the reference value parameter numbers and formats in the controller while retaining the reference value cycle times in the Ωmega Drive-Line II.

NOTE

To enable the new reference value parameter number list, the system must send object 16#6002. For more information on this topic, see the "CAN Interface Option Board" description.

FB CAN_PA_LIST_WRITE uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
u_WR_PAR_NR0	UINT	Reference value parameter number 0
us_WR_FORMAT0	USINT 0, 1	Reference value format 0
u_WR_PAR_NR1	UINT	Reference value parameter number 1
us_WR_FORMAT1	USINT 1, 2	Reference value format 1
u_WR_PAR_NR2	UINT	Reference value parameter number 2
us_WR_FORMAT2	USINT 1, 2	Reference value format 2
u_WR_PAR_NR3	UINT	Reference value parameter number 3
us_WR_FORMAT3	USINT 1, 2	Reference value format 3
x_EN	BOOL	Release
x_RESET	BOOL	Reset
x_BUSY	BOOL	Communication is active

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PA_LIST_WRITE transfers with the values of inputs u_WR_PAR_NR0 to u_WR_PAR_NR3 and us_WR_FORMAT0 to us_WR_FORMAT3 a write PA data job to the controller containing the device number us_DRIVE_NR. The controller processes the write PA data job and returns the result of communication.

The system enters the changed reference value parameter numbers and reference value formats in the variable that contains the initialization data of the controllers and the CAN IO modules (see input/output _CAN_DATA). FB CAN_PD_COMM needs this variable for process data communication, and FB CAN_SD_CONTROL needs it for requirements data communication monitoring.

To enable the new reference value parameter number list, FB CAN_OBJ_WRITE must then send object 16#6002. See the "CAN Interface Option Board" description.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.
You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input/output _CAN_DATA:

A global variable of data type CAN_INIT_DATA_BMSTRUCT is connected at output _CAN_DATA.

In general, this variable was already connected at initialization of the CAN interface module on FB CAN_INIT, output _CAN_DATA.

Example:

```
_CAN_INIT_DATA_OPT: CAN_INIT_DATA_BMSTRUCT;
```

Where:

`_CAN_INIT_DATA_OPT`	is variable name with the data type short designation "_" for STRUCT
`CAN_INIT_DATA_BMSTRUCT`	is the data type

This variable contains the initialization data of the controllers and the CAN IO modules of this CAN interface module, and it is needed by some FBs for communication.

Input us_DRIVE_NR:

At input us_DRIVE_NR you state the device number of the controller on the CAN bus to which the write PA data job is to be transferred.

Inputs u_WR_PAR_NR0 to u_WR_PAR_NR3:

You state the reference value parameter numbers at inputs u_WR_PAR_NR0 to u_WR_PAR_NR3. If no inputs are assigned, the system sends parameter number 0 to the controller as the reference value parameter number.

Inputs us_WR_FORMAT0 to us_WR_FORMAT3:

You state the formats of the reference values at inputs us_WR_FORMAT0 to us_WR_FORMAT3.

NOTE

Reference value 0 may have the format byte or word. Reference values 1, 2 and 3 may have the format word or doubleword.

Format specification:	us_WR_FORMATx = 0:	format byte (reference value 0)
	us_WR_FORMATx = 1:	format word (reference values 0, 1, 2 and 3)
	us_WR_FORMATx = 2:	format doubleword(reference values 1, 2 and 3)
	x = 0, 1, 2, 3	

The system retains the reference value cycle times that were stated at initialization at FB CAN_DRIVE_INIT.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY = FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_PA_LIST_WRITE by means of x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (write PA data)
2 - 7	Reserved

5.13 CAN_PAR_READ

Description

You can use this function block for CAN to read a parameter of a controller with a CAN interface module.

NOTE

FB CAN_PAR_READ uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
u_PAR_NR	UINT	Parameter number
us_PAR_FORMAT	USINT 1, 2	Parameter format
us_PAR_ELEMENT	USINT 7	Parameter element
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
ud_PAR_VALUE	UDINT	Read parameter value
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PAR_READ transfers with the values of inputs u_PAR_NR, us_PAR_FORMAT and us_PAR_ELEMENT a read parameter job to the controller with device number "us_DRIVE_NR". The controller processes the read parameter job and returns the result of communication. The system outputs the contents of the parameter element at output ud_PAR_VALUE.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_DRIVE_NR:

At input us_DRIVE_NR, you specify the device number of the controller on the CAN bus.

Input u_PAR_NR:

You state the parameter number for the read parameter job at input u_PAR_NR.

Input us_PAR_FORMAT:

At input us_PAR_FORMAT, you state the format of the parameter value to be read. us_PAR_FORMAT = 1 means word format, us_PAR_FORMAT = 2 means doubleword format. The default setting is us_PAR_FORMAT = 1, i. e. word format.

Input us_PAR_ELEMENT:

You set the parameter element at input us_PAR_ELEMENT. The default setting is us_PAR_ELEMENT = 7, i. e. read the parameter value.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY = FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_OBJ_READ by means of x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Output ud_PAR_VALUE:

The system makes available the read parameter element at output ud_PAR_VALUE.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (read parameter data)
2	Invalid format (> 2)
3 - 7	Reserved

5.14 CAN_PAR_WRITE

Description

You can use this function block for CAN to write a parameter of the controller.

NOTE

FB CAN_PAR_WRITE uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
u_PAR_NR	UINT	Parameter number
us_PAR_FORMAT	USINT 1, 2	Parameter format
us_PAR_ELEMENT	USINT 7	Parameter element
ud_PAR_VALUE	UDINT	Parameter value to be sent
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PAR_WRITE transfers with the values of inputs u_PAR_NR, us_PAR_FORMAT, us_PAR_ELEMENT and ud_PAR_VALUE a write parameter job to the controller with device number "us_DRIVE_NR". The controller processes the write parameter job and returns the result of communication.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_DRIVE_NR:

At input us_DRIVE_NR, you specify the device number of the controller on the CAN bus.

Input u_PAR_NR:

You state the parameter number for the write parameter job at input u_PAR_NR.

Input us_PAR_FORMAT:

At input us_PAR_FORMAT, you state the format of the parameter value to be sent. us_PAR_FORMAT = 1 means word format, us_PAR_FORMAT = 2 means doubleword format. The default setting is us_PAR_FORMAT = 1, i. e. word format.

Input us_PAR_ELEMENT:

You set the parameter element (e.g. us_PAR_ELEMENT = 7 - parameter data) at input us_PAR_ELEMENT. The default setting is us_PAR_ELEMENT = 7, i. e. parameter data.

Input ud_PAR_VALUE:

The object value to be transferred/written is connected at input ud_PAR_VALUE.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY=FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB PAR_OBJ_WRITE by means of x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Output x_BUSY:

Output x_BUSY indicates by TRUE the FB is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (write parameter)
2	Invalid format (> 2)
3 - 7	Reserved

NOTE

The system sets bit 1 if an error in communication occurred. This happens in the case of a value range violation, for example. Some word parameters, for example, may not be written with values greater than 16#3FFF while others may not be 0. For more information on this topic, refer to the controller or parameter descriptions.

5.15 CAN_PD_COMM

Description

You can use this function block for CAN to carry out process data communication via CAN between Ωmegas and controllers. The control word and the reference values are transmitted to the controller, the controller receives the status word and the actual values and outputs them.

The specified and actual value parameter numbers are set at initialization at FB CAN_DRIVE_INIT and sent to the controller by means of FB CAN_INIT.

NOTE

FB CAN_PD_COMM uses library BM_TYPES_20bd00 or above.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
w_CONTROLWORD	WORD	Control word
x_ACYCLIC	BOOL	Cyclical/acyclical sending and receiving
ud_WR_VALUE0	UDINT	Reference value 0
ud_WR_VALUE1	UDINT	Reference value 1
ud_WR_VALUE2	UDINT	Reference value 2
ud_WR_VALUE3	UDINT	Reference value 3
t_TIME	TIME > 0, to 30 s	Monitoring time
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
w_STATUSWORD	WORD	Status word
ud_RD_VALUE0	UDINT	Actual value 0
ud_RD_VALUE1	UDINT	Actual value 1
ud_RD_VALUE2	UDINT	Actual value 2
ud_RD_VALUE3	UDINT	Actual value 3
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PD_COMM sends the control word and the reference values to the controller and receives the status word and the actual values from the controller.

You state the parameter numbers, the formats and, if necessary, the cycle times of the reference values and actual values at initialization of the CAN interface module and communication with the controllers (for more information on this topic, refer to the descriptions of FBs CAN_DRIVE_INIT and CAN_INIT.)

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input/output _CAN_DATA:

A global variable of data type CAN_INIT_DATA_BMSTRUCT is connected at output _CAN_DATA.

In general, this variable was already connected at initialization of the CAN interface module on FB CAN_INIT, output _CAN_DATA.

Example:

```
_CAN_INIT_DATA_OPT : CAN_INIT_DATA_BMSTRUCT;
```

Where:

`_CAN_INIT_DATA_OPT`	is variable name with the data type short designation "_" for STRUCT
`CAN_INIT_DATA_BMSTRUCT`	is the data type

This variable contains the initialization data of the controllers and the CAN IO modules of this CAN interface module, and it is needed by some FBs for communication.

Input us_DRIVE_NR:

At input us_DRIVE_NR, you specify the device number of the controller on the CAN bus.

Input w_CONTROLWORD:

You connect the control word to be sent to the controller at input w_CONTROLWORD.

Input x_ACYCLIC:

Using input x_ACYCLIC, you specify whether the specified and actual values are to be transferred cyclically or acyclically. If x_ACYCLIC = FALSE, the system transfers the specified and actual values cyclically with the cycle times that are specified in FB CAN_DRIVE_INIT. If x_ACYCLIC = TRUE, the system transfers the specified and actual values acyclically. For this, the system enters in the respective control register of the specified and actual values a send request (for the reference values) or a reception request (for the actual values). The default setting is x_ACYCLIC = FALSE (the specified and actual values are transferred cyclically with the cycle times that are specified in FB CAN_DRIVE_INIT.

Inputs ud_WR_VALUE0 to ud_WR_VALUE3:

The reference values are connected to inputs ud_WR_VALUE0 to ud_WR_VALUE3. The sequence of reference values corresponds to the sequence of reference value parameter numbers stated at FB CAN_DRIVE_INIT.

NOTE

If cycle times for the specified and actual values are stated in FB CAN_DRIVE_INIT (or actual value cycle times are stored in the controller) and acyclical transfer is selected in FB CAN_PD_COMM, then bus loading is increased, since the system must send and receive additional messages on the CAN bus.

Input x_EN:

You use input x_EN to enable FB CAN_PD_COMM for process data communication. To do this, x_EN is set to TRUE. If x_EN is set to FALSE before x_OK = TRUE, it is assumed that communication was cancelled deliberately. You should then reset the FB using input x_RESET = TRUE.

Input t_TIME:

At input t_TIME, you state the monitoring time. If input t_TIME is not assigned, this yields a presetting of 3 s.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Output w_STATUSWORD:

At output w_STATUSWORD, the system outputs the status word that was received from the controller.

Outputs ud_RD_VALUE0 to ud_RD_VALUE3:

The received actual values are available at outputs ud_RD_VALUE0 to ud_RD_VALUE3. The sequence of reference values corresponds to the sequence of actual value parameter numbers stated at FB CAN_DRIVE_INIT.

Output x_OK:

Output x_OK indicates with TRUE that the status word was received.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	No controller with this us_DRIVE_NR is initialized
2	Timeout
3 - 7	Reserved

5.16 CAN_PE_LIST_READ

Description

You can use this function block for CAN to read the current actual value parameter number list (PE data list) for cyclical communication in the controller.

In general, the actual value parameter numbers are set at FB CAN_DRIVE_INIT and sent to the controller via FB CAN_INIT.

NOTE

FB CAN_PE_LIST_READ uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
u_RD_PAR_NR0	UINT	Actual value parameter number 0
u_RD_PAR_NR1	UINT	Actual value parameter number 1
u_RD_PAR_NR2	UINT	Actual value parameter number 2
u_RD_PAR_NR3	UINT	Actual value parameter number 3
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PE_LIST_READ transfers a read PE data job to the controller with device number us_DRIVE_NR. The controller processes the read PE data job and returns the result of communication. The system outputs the contents of the PE data list at outputs u_RDR_PAR_NR0 to u_RD_PAR_NR3.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input us_DRIVE_NR:

At input us_DRIVE_NR, you specify the device number of the controller on the CAN bus.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY = FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_PE_LIST_READ by means of x_RESET = TRUE.

Input x_RESET:

You can use x_RESET = TRUE to reset the FB.

Outputs u_RD_PAR_NR0 to u_RD_PAR_NR3:

The system outputs the contents of the controller's PE data list, i.e. actual value parameter numbers 0 to 3, at outputs u_RD_PAR_NR0 to u_RD_PAR_NR3.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (read PE data)
2 - 7	Reserved

5.17 CAN_PE_LIST_WRITE

Description

You can use this function block for CAN to set a new actual value parameter number list (PE data list) for cyclical communication in the controller.

In general, the actual value parameter numbers, actual value formats and the actual value cycle times are set at FB CAN_DRIVE_INIT and sent to the controller via FB CAN_INIT.

FB CAN_PE_LIST_WRITE makes it possible to change at a later date the actual value parameter numbers and formats in the controller while retaining the actual value cycle times in the controller.

NOTE

FB CAN_PE_LIST_WRITE uses library BM_TYPES_20bd00 or above.

This FB is used together with FB CAN_SD_CONTROL.

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
u_RD_PAR_NR0	UINT	Actual value parameter number 0
us_RD_FORMAT0	USINT 0, 1	Actual value format 0
u_RD_PAR_NR1	UINT	Actual value parameter number 1
us_RD_FORMAT1	USINT 1, 2	Actual value format 1
u_RD_PAR_NR2	UINT	Actual value parameter number 2
us_RD_FORMAT2	USINT 1, 2	Actual value format 2
u_RD_PAR_NR3	UINT	Actual value parameter number 3
us_RD_FORMAT3	USINT 1, 2	Actual value format 3
x_EN	BOOL	Release
x_RESET	BOOL	Reset

Parameter output	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
x_BUSY	BOOL	Communication is active
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit
x_OK	BOOL	OK bit

FB CAN_PE_LIST_WRITE transfers with the values of inputs u_RD_PAR_NR0 / us_RD_FORMAT0 to inputs u_RD_PAR_NR3 / us_RD_FORMAT3 a write PE data job to the controller with device number "us_DRIVE_NR". The controller processes the write PE data job and returns the result of communication.

The system enters the changed actual value parameter numbers and actual value formats in the variable that contains the initialization data of the controllers and the CAN IO modules (see input/output _CAN_DATA). Several FBs need this variable for communication.

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input/output _CAN_DATA:

A global variable of data type CAN_INIT_DATA_BMSTRUCT is connected at output _CAN_DATA.

In general, this variable was already connected at initialization of the CAN interface module on FB CAN_INIT, output _CAN_DATA.

Example:

```
_CAN_INIT_DATA_OPT : CAN_INIT_DATA_BMSTRUCT;
```

Where:

`_CAN_INIT_DATA_OPT`	is variable name with the data type short designation "_" for STRUCT
`CAN_INIT_DATA_BMSTRUCT`	is the data type

This variable contains the initialization data of the controllers and the CAN IO modules of this CAN interface module, and it is needed by some FBs for communication.

Input us_DRIVE_NR:

At input us_DRIVE_NR, you specify the device number of the controller on the CAN bus.

Inputs u_RD_PAR_NR0 to u_RD_PAR_NR3:

You state the actual value parameter numbers at inputs u_RD_PAR_NR0 to u_RD_PAR_NR3. If no inputs are assigned, the system sends parameter number 0 to the controller as the actual value parameter number.

Inputs us_RD_FORMAT0 to us_RD_FORMAT3:

You state the formats of the actual values at inputs us_RD_FORMAT0 to us_RD_FORMAT3.

NOTE

Actual value 0 may have the format byte or word. Actual values 1, 2 and 3 may have the format word or doubleword.

Format specification: us_RD_FORMATx = 0: format byte (actual value 0)

us_RD_FORMATx = 1: format word (actual values 0, 1, 2 and 3)

us_RD_FORMATx = 2: format doubleword(actual values 1, 2 and 3)

x = 0, 1, 2, 3

The system retains the actual value cycle times that were stated at initialization at FB CAN_DRIVE_INIT.

Input x_EN:

Communication is started by means of x_EN = TRUE. If x_EN is set to FALSE before x_BUSY = FALSE, it is assumed that communication was cancelled deliberately. You must then reset FB CAN_PE_LIST_WRITE by means of x_RESET = TRUE.

Input x_RESET:

You can reset the FB using input x_RESET = TRUE.

Output x_BUSY:

Output x_BUSY indicates by TRUE that communication is active.

Output x_OK:

The system sets output x_OK to TRUE when communication has been completed successfully.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	Invalid device number (us_DRIVE_NR greater than 15)
1	Error in communication (write PE data)
2 - 7	Reserved

5.18 CAN_SD_CONTROL

Description

You use this function block for CAN to monitor the requirements data communication of a controller.

NOTE

FB CAN_SD_CONTROL uses library BM_TYPES_20bd00 or above.

You must use FB CAN_SD_CONTROL if you are using the following FBs of requirements data communication:

CAN_PAR_READ, CAN_PAR_WRITE,
CAN_OBJ_READ, CAN_OBJ_WRITE,
CAN_PA_LIST_READ, CAN_PA_LIST_WRITE,
CAN_PE_LIST_READ and/or CAN_PE_LIST_WRITE

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
us_DRIVE_NR	USINT 0 to 15	Device number of the controller on the CAN bus
us_MAX_NR_OF_COMM	USINT	Maximum number of communications attempts
t_TIME	TIME > 0 s	Monitoring time
x_RESET	BOOL	Reset

Parameter input	Data type	Description
_BASE	CAN_CTRL_BMSTRUCT	Operating data for the CAN interface module
_CAN_DATA	CAN_INIT_DATA_BMSTRUCT	Initialization data of the CAN
b_ERR	BYTE	Error byte
x_ERR	BOOL	Error bit

Input/output _BASE:

At _BASE, you must connect a global variable of data type CAN_CTRL _BMSTRUCT.

You must assign this variable via declaration of global variables to the base address of the CAN interface module; in general, this is already connected at initialization of the CAN interface module on FB CAN_INIT, input _BASE_CTRL.

Example:

CAN-M-01 option board for Ωmega Drive-Line II

```
_CAN_CTRL_OPT AT %MB3.3000000 : CAN_CTRL_BMSTRUCT;
```

Where:

`_CAN_CTRL_OPT`	is the variable name with the data type short designation "_" for STRUCT
`CAN_CTRL_BMSTRUCT`	is the data type
`%MB3.3000000`	is the base address of the CAN interface module on the CAN-M-01 option board

Input/output _CAN_DATA:

A global variable of data type CAN_INIT_DATA_BMSTRUCT is connected at output _CAN_DATA.

In general, this variable was already connected at initialization of the CAN interface module on FB CAN_INIT, output _CAN_DATA.

Example:

```
_CAN_INIT_DATA_OPT: CAN_INIT_DATA_BMSTRUCT;
```

Where:

`_CAN_INIT_DATA_OPT`	is variable name with the data type short designation "_" for STRUCT
`CAN_INIT_DATA_BMSTRUCT`	is the data type

This variable contains the initialization data of the controllers and the CAN IO modules of this CAN interface module, and it is needed by some FBs for communication.

Input us_DRIVE_NR:

At input us_DRIVE_NR, you specify the device number of the controller on the CAN bus.

Input us_MAX_NR_OF_COMM:

You set the maximum number of communications attempts at input us_MAX_NR_OF_COMM. If input us_MAX_NR_OF_COMM is not assigned, this yields a presetting of us_MAX_NR_OF_COMM = 1.

You set the monitoring time in seconds at input t_TIME. If input t_TIME is not assigned, this yields a presetting of 3 s.

Input x_RESET:

You can reset the FB using input x_RESET = TRUE.

Outputs x_ERR, b_ERR:

If an error occurs, the system sets error bit x_ERR to TRUE and outputs error byte b_ERR.

Error byte b_ERR:

Bit No.	Error
0	General timeout occurred
1	General error
2	Controller not initialized
3	No communication attempts are taking place
4 - 7	Reserved

NOTE

If no requirements data communication is taking place, the system starts a timer and sets the error byte to 16#00. If the timer expires, the system sets bit 3 of the error byte. The next time that FB CAN_SD_CONTROL is called, the system starts the timer again and the error byte becomes 16#00 again. This means that bit 3 of the error byte is only set until the next time that the FB is called.

6 INDEX

T

V