CoDeSys FBE - Library

Reference Guide for use with

EASY215, EASY217, EASY219, EASY235, EASY237, EASY238, EASY2502, EASY2504, EASY2506

For EASY242, 2606 and hipecs PLC check special description!

1. Content

2. Introduction

In order to support the powerful EASY PLC core modules there is a library extension for the CoDeSys development environment. Any libraries are internal, that means they are implemented in the PLC runtime system. Others are external IEC-Code libraries.

There are the following libraries:

Library name	available	Description	Type
FBE_FastTask.lib	now	Implementation of Interrupt service routines	RT
FBE_DataBus.lib	now	Access to the microcontroller data bus from IEC61131 applications	RT
FBE_Encoder.lib	now	Support of incremental encoders	RT
FBE_CIA405.lib	now	Support of CANopen functions according to DS405 standard	RT
FBE_Util.lib	now	Several Utilities	RT
FBE_Com.lib	now	Support of serial interfaces	RT
FBE_BasicCan.lib	now	Support of standard CAN message communication	RT
FBE_FastAdc.lib	now	Support of analog to digital conversion	RT
FBE_Fifo.lib	now	Support of a fifo-buffer for user defined data type elements	RT
FBE_Keyboard.lib	now	Support of Keyboard	RT
FBE_SmPos.lib	now	Support of Stepper Motor	RT
FBE_SpiEeprom	1.42	Support of external SPI-EEPOM	RT
FBE_Pulse.lib	now	Support of PWM and frequency oscillator	IEC
FBE_Pulse215.lib	now	Support of PWM only EASY215 / EASY217	IEC
FBE_LCD.lib	now	Support of Character-Displays	IEC

RT:	included in PLC Runtime system
IEC:	external IEC-Code library

3. Version History

Firmware Version	Type	Description
1.100	new	Support for CANopen Slave functionality The EASY21x and EASY235 system support CANopen slave functionality according to DS401 (I/O-module)
	new	Implementation of Status Registers and the functions for Status Register Access
1.101	new	Support of event counter The Encoder channels support event counting capabilities.
1.420	new	Support of external SPI-EEPOM
1.421	new	Support of function FBE_UTIL_OUT_SETMASK

4. Hardware to Library Cross-Reference

Not all libraries work with each EASY Module because they have different hardware units for application. This reference shows the libraries, which can be used on an EASY.

Library name	EASY 215	EASY 217	EASY 219	EASY 235	EASY 237	EASY 238	EASY 2502	EASY 2504	EASY 2506
FBE_FastTask.lib	X	X	X	X	X	X		X	X
FBE_DataBus.lib				X	X	X		X	
FBE_Encoder.lib	X	X	X	X	X	X	X	X	X
FBE_CIA405.lib	X	X	X	X	X	X	X	X	X
FBE_Util.lib	X	X	X	X	X	X	X	X	X
FBE_Com.lib				X	X	X		X	X
FBE_BasicCan.lib	X	X	X	X	X	X	X	X	X
FBE_FastAdc.lib				X	X	X		X	
FBE_Fifo.lib				X	X	X	X	X	X
FBE_Keyboard.lib								X	
FBE_SmPos.lib	X	X	X	X	X	X	X	X	
FBE_Pulse.lib				X	X	X	X	X	X
FBE_Pulse215.lib	X	X							
FBE_LCD.lib				X	X	X		X	X
FBE_SpiEeprom	X	X	X	X	X	X			

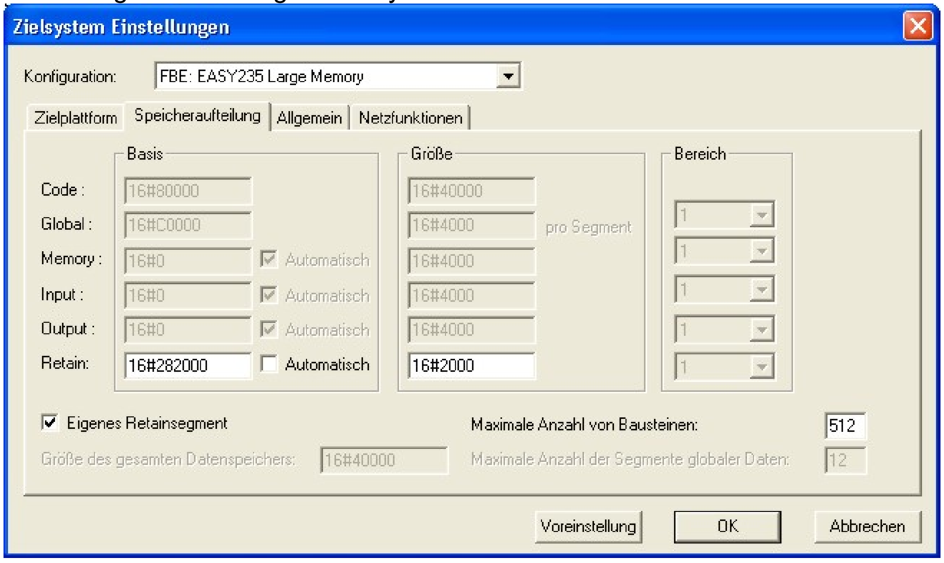
5. Non Volatile Memory for Retain Segment

The EASY235 gives the possibility to add a NVRAM to the data bus in order to support non volatile Retain segments. The NVRAM must be connected to the data bus of the EASY235 system using 8 data bits. Chip-Select CS4# must be used to enable the RAM. It is recommended to use a time keeper with an access time of 120 nsec maximum for NVRAM.
The retain segment must be configured in the target memory layout dialog of the CoDeSys programming environment.

The use of the following chips (or compatible) is recommended:

chip type	Vendor	Size in kBytes	Start address in CoDeSys
M48T08	ST Microelectronics	8	16#282000
M48T35	ST Microelectronics	32	16#280000
M48T128	ST Microelectronics	128	16#280000
DS1646	Dallas Semiconductor	128	16#280000

The configuration dialog for 8 kByte must be set as follows



Zielsystem Einstellungen

Konfiguration: FBE: EASY235 Large Memory

Zielplattform | Speicheraufteilung | Allgemein | Netzfunktionen

	Basis		Größe		Bereich
Code :	16#80000		16#40000		
Global :	16#C0000		16#4000	pro Segment	1
Memory :	16#0	☑ Automatisch	16#4000		1
Input :	16#0	☑ Automatisch	16#4000		1
Output :	16#0	☑ Automatisch	16#4000		1
Retain:	16#282000	☐ Automatisch	16#2000		1

☑ Eigenes Retainsegment

Maximale Anzahl von Bausteinen: 512

Größe des gesamten Datenspeichers: 16#40000

Maximale Anzahl der Segmente globaler Daten: 12

Voreinstellung | OK | Abbrechen

EASY 235, 237, 238, 2504, 2505, 2506

Size in kBytes	Start address in CoDeSys	Größe in CoDeSys
8	16#282000	16#2000
32	16#280000	16#8000
128	16#280000	16#20000

EASY 2606

Size in kBytes	Start address in CoDeSys	Größe in CoDeSys
8	16#142000	16#2000
32	16#140000	16#8000
128	16#140000	16#20000

6. Utilities

The Library FBE_Util.lib is a Library extension for the CoDeSys PLC runtime system and supporting several utilities for IEC61131 applications running on systems from frenzel + berg elektronik. It is an internal library; all functions are included in the runtime system.

The following functions are implemented:

Function name	Description
FBE_UTIL_CYCLETIME	Returns the cycle time for the PLC application
FBE_UTIL_GETSV	Reads the software version of the run time system
FBE_UTIL_GETPLCID	Read the PLC type product code or target ID
FBE_UTIL_LED	Sets the user LEDs (not on EASY21x)
FBE_UTIL_REGLOOP	Registers a IEC61131 pou that is implemented as program for endless loop processing.
FBE_UTIL_REGTC	Registers a IEC61131 pou that is implemented as program for time cycle processing. The time cycle is started every 2 milli seconds and is running completely independent from the PLC cycle.
FBE_UTIL_GETSTATUSREG	Read status register
FBE_UTIL_MASKSTATUSREG	Reset several bits of the status register
FBE_UTIL_OUT_SETMASK	Set an output mask that masks single bits that will not longer be modified by the PLC.

6.1. FBE_UTIL_CYCLETIME

Description:

Read actual cycle time in milliseconds for one PLC cycle. The actual cycle time is added to an offset given as parameter. This enables long time measurement.

Declaration:

```
FUNCTION FBE_UTIL_CYCLETIME : UINT
VAR_INPUT
        OffsetVal: UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
OffsetVal	UINT	Offset to add to the actual cycle time

Return Value (Data type UINT)

The function returns the sum: “actual cycle time” + OffsetVal.

6.2. FBE_UTIL_GETSV

Description:
Read the software version of the firmware.

Declaration:

```
FUNCTION FBE_UTIL_GETSV : UINT
VAR_INPUT
        Dummy: UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	UINT	Must be held at zero

Return Value (Data type UINT)
The function returns the software version

6.3. FBE_UTIL_GETPLCID

Description:
Read the hardware identification out of the PLC system. The function can return either the CodeSys specific target identification or the frenzel + berg elektronik product code.

Declaration:

```
FUNCTION FBE_UTIL_GETPLCID: UDINT
VAR_INPUT
        CodeType        : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
CodeType	UINT	0 The system will return the Codesys target identification 1 The system will return the frenzel + berg elektronik product code

Return Value (Data type UDINT)
The function returns the PLC Id

Hardware	Codesys Target (dez)	FBE-Code (hex)	Hardware	Codesys Target (dez)	FBE-Code (hex)
EASY21x	14900	0202 1x01	EASY2502	14905	0425 0201
EASY235-CR24-L4	14902	0202 3502	EASY2504-20L	14903	04250401
EASY236-40-L4	14906	0202 3601			

6.4. FBE_UTIL_LED

Description:
Set the User-LEDs.

Declaration:

```
FUNCTION FBE_UTIL_LED : BOOL
VAR_INPUT
        LED_NR: BYTE;
        LIGHT  : INT;
END_VAR
```

Parameters:

Name	Data-Type	Description
LED_NR	BYTE	Number of the LED to program (0 for the first LED)
LIGHT	INT	LED Behavior. 0 LED is off 1..100 Duty cycle of LED-blinking (100% means on) 10% means short on time and long off time 90% means long on time and short off time -1 .. -9 LED flicker times. The LED will be switched on for –LIGHT times and then will be switched off for approx 1 second.

Return Value (Data type BOOL)
The function returns true, if the LED was programmed successfully.

6.5. FBE_UTIL_REGLOOP

Description:

Registers a function for the use as endless loop task. Registering of program modules is done with the individual Id of this module. The Id can be checked with the function "INDEXOF" of the runtime system. With registration of the function, the endless loop task will always be executed if there is a spare of processor power. The endless loop is completely independent from the PLC cycle.

Declaration:

```
FUNCTION FBE_UTIL_REGLOOP : BOOL
VAR_INPUT
        NPOU_ID: INT;
END_VAR
```

Parameters:

Name	Data-Type	Description
NPOU_ID	INT	Number of the program module that must be registered

Return Value (Data type BOOL)

The function returns TRUE, if the function was successfully registered to the PLC runtime system. Otherwise FALSE is returned.

6.6. FBE_UTIL_REGTC

Description:

Registers a function for the use as time cycle task. Registering of program modules is done with the individual Id of this module. The Id can be checked with the function "INDEXOF" of the runtime system. With registration of the function, the time cycle task will be called every 2 milli seconds. The time cycle task enables parts of a program which must guarantee a specific calling frequency. It is completely independent from the PLC cycle.

Declaration:

```
FUNCTION FBE_UTIL_REGTC : BOOL
VAR_INPUT
        NPOU_ID: INT;
END_VAR
```

Parameters:

Name	Data-Type	Description
NPOU_ID	INT	Number of the program module that must be registered.

Return Value (Data type BOOL)

The function returns TRUE, if the function was successfully registered to the PLC runtime system. Otherwise FALSE is returned.

6.7. FBE_UTIL_GETSTATUSREG

Description:

Reads the actual value from the status register of the PLC.

Register Name	Reg. Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PLC-Status	0	-	-	-	-	-	-	-	ilop	-	-	-	-	-	-	-	-

Ilop	Processor error occurred. Word access to odd address. This is a critical processor error that is caused by accessing odd addresses with word or long pointer access.

Register Name	Reg. Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CAN-Status	1	mm	slm	coe	-	-	-	-	-	NMT-State							

mm	CANopen Master Mode:	1 CAN-Master enabled, 0 CAN-Master disabled
slm	CANopen Slave Mode:	1 CAN-Slave enabled, 0 CAN-Slave disabled
NMT-State	CANopen NMT-State	

Declaration:

```
FUNCTION FBE_UTIL_GETSTATUSREG : UINT
VAR_INPUT
        RegNr  : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
REG_NR	UINT	Number (Address) of the register to read.

Return Value (Data type UINT)

The function returns the register value.

6.8. FBE_UTIL_MASKSTATUSREG

Description:
Resets bits of the status register of the PLC.

Declaration:
```
FUNCTION FBE_UTIL_MASKSTATUSREG : BOOL
VAR_INPUT
        RegNr  : UINT;
        Mask   : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
REG_NR	UINT	Number (Address) of the register
MASK	UINT	Each Bit that is set in the parameter MASK resets the corresponding bit of the status register.

Return Value (Data type BOOL)
The function returns TRUE, if the masking function was successfully. Otherwise FALSE is returned.

6.9. FBE_UTIL_OUT_SETMASK

Description:
Sets an output mask register for the corresponding output byte. Each masked bit will not be modified from the PLC. This function is implemented to support setting of hardware output bits directly from within IEC1131.
Example: FBE_UTIL_OUT_SETMASK(0, 16#03); will disable the automatic update ouf OUT0.0 and OUT0.1

Declaration:
```
FUNCTION FBE_UTIL_OUT_SETMASK : BOOL
VAR_INPUT
        ByteNr : UINT;
        Mask   : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
BYTE_NR	UINT	Number of Output Byte
MASK	UINT	Each Bit that is set in the parameter MASK resets PLC control for the corresponding output bit.

Return Value (Data type BOOL)
The function returns TRUE, if the masking function was successfully. Otherwise FALSE is returned.

7. *FastTask / Interrupt*

The Library FBE_FastTask.lib is a Library extension for the CoDeSys PLC runtime system and enables implementation of Interrupt services for IEC61131 applications. It is an internal library; all functions are included in the runtime system.

Each Interrupt channel is assigned to a dedicated interrupt input pin. The interrupts are edge sensitive and may be configured to positive, negative or both transitions at the corresponding interrupt input pin. The interrupt priority may be selected from three levels.

An interrupt may not only be activated from hardware signal transitions, but also by IEC61131 application software. This feature enables implementation of program units at different CPU priorities.

The following functions are implemented:

Function name	Description
FBE_IRQ_REGISTER	Registers a specific program module for the use with interrupt control. Sets the requested interrupt priority, and interrupt edge
FBE_IRQ_ENABLE	Enables a dedicated interrupt channel
FBE_IRQ_DISABLE	Disables a dedicated interrupt channel
FBE_IRQ_SETIRQ	Set Interrupt request from IEC61131 application
FBE_IRQ_CLRIRQ	Clear Interrupt request from IEC61131 application

The program module to call from interrupt should be implemented as "Program" in the CoDeSys Development environment. There must be no parameters into this module.

Calling of the library functions is according to the IEC61131 standard. See Library Manager of the CoDeSys programming tool for detailed parameter information of the library functions.

7.1. FBE_IRQ_REGISTER

Description:

Registers a function for the use with the interrupt control system. Registering of program modules as an interrupt task is done with the individual Id of this module. The Id can be checked with the function "INDEXOF" of the runtime system.
With registration of the interrupt function, the interrupt keeps still disabled. In order to use this interrupt channel, it must be enabled with function "FBE_IRQ_ENABLE".
See example for more information.

Declaration:

```
FUNCTION FBE_IRQ_REGISTER : BOOL
VAR_INPUT
        IrqNr           : UINT;
        nPOU_ID         : INT;
        IrqPriority     : UINT;
        Edge            : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
IRQNR	UINT	Number of the Interrupt Channel, which must be used for this Interrupt function.
NPOU_ID	INT	Number of the program module that must be registered for this interrupt.
IRQPRIORITY	UINT	Priority level of the Interrupt channel 0 : Lowest Priority Lower than System Interrupt, Serial Programming Interface Lower than PLC cycle 1 : Middle Priority Lower than system IRQ and System Sio Higher than PLC cycle 2 : High Priority Higher than system IRQ and System Sio Higher than PLC cycle 3 : Highest Priority (only supported for IRQ channels 0..3) Use with care !!! Higher than all other interrupt sources. Only for very short interrupt program
EDGE	UINT	Enables the active edge for interrupt activation (Both edges may be enabled at the same time) Bit0 Enables/Disables interrupt enable on rising edge of input signal at dedicated interrupt pin Bit1 Enables/Disables interrupt enable on falling edge of input signal at dedicated interrupt pin Setting of the bits is interpreted as follows Bitx = 0 Edge disabled, Interrupt is not activated at this transition Bitx = 1 Edge enabled, Interrupt is activated at this transition

Return Value (Data type BOOL)

The function returns TRUE, if the interrupt function was successfully registered to the PLC runtime system. Otherwise FALSE is returned.

Example

A program module named "CallMeFromIrq" is to call with rising edge at interrupt input pin IRQ2. The requested Priority for this interrupt is 2.
The registration of the interrupt is done with:

```
…
VAR
        …
        Success: BOOL;                    (* Use this var for Success of Boolean functions *)
        …
END_VAR
…
…
Success:= FBE_IRQ_REGISTER(2, INDEXOF(CallMeFromIrq), 2, 1);
…
```

Note:
Using of interrupt priorities higher than level 1 are only possible for very short interrupt programs. Too long interrupt program execution time might cause data losses in any interface (CAN, RS232 …)

7.2. FBE_IRQ_ENABLE

Description:

Enables an interrupt channel for reception of interrupt requests. Previously set request bits will be cleared.
With registration of the interrupt function, the interrupt keeps still disabled. In order to use this interrupt channel, it must be enabled with this function.
The user must take care of the correct handling of registering and enabling of interrupts. This function does not check, whether there is a interrupt task registered to the channel, that should be enabled.

Declaration:

```
FUNCTION FBE_IRQ_ENABLE : BOOL
VAR_INPUT
        IrqNr: UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
IRQNR	UINT	Number of the Interrupt Channel, which must be enabled.

Return Value (Data type BOOL)

The function returns TRUE, if the interrupt function was successfully enabled. Otherwise FALSE is returned.

Example

A program module named "CallMeFromIrq" is to call with both edges at interrupt input pin IRQ0. The requested Priority for this interrupt is 1. Afterwards the interrupt must be enabled.

```
…
VAR
        …
        Success: BOOL;                (* Use this var for Success of Boolean functions *)
        …
END_VAR
…
…
Success:= FBE_IRQ_REGISTER(0, INDEXOF(CallMeFromIrq), 1, 3);
…
Success:= FBE_IRQ_ENABLE(0);
…
```

7.3. FBE_IRQ_DISABLE

Description:
Disables an interrupt channel for reception of interrupt requests.

Declaration:
```
FUNCTION FBE_IRQ_DISABLE : BOOL
VAR_INPUT
        IrqNr: UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
IRQNR	UINT	Number of the Interrupt Channel, which must be disabled.

Return Value (Data type BOOL)
The function returns TRUE, if the interrupt function was successfully disabled. Otherwise FALSE is returned.

7.4. FBE_IRQ_SETIRQ

Description:
Sets the interrupt request flag of a dedicated interrupt channel. If this channel was previously enabled, the interrupt will be called. If the Priority is 1 or 2 (higher than PLC cycle) the interrupt will be called immediately, otherwise it will be called, after the PLC cycle has finished.

Declaration:
```
FUNCTION FBE_IRQ_SETIRQ : BOOL
VAR_INPUT
        IrqNr: UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
IRQNR	UINT	Number of the Interrupt Request Channel, which must be set.

Return Value (Data type BOOL)
The function returns TRUE, if the interrupt request was successfully set. Otherwise FALSE is returned.

Example
The interrupt request for Interrupt channel 3 must be set.

```
VAR
        Success: BOOL;                  (* Use this var for Success of Boolean functions *)
END_VAR
Success:= FBE_IRQ_SETIRQ(0);
```

7.5. FBE_IRQ_CLRIRQ

Description:
Clears the interrupt request flag of a dedicated interrupt channel.

Declaration:

```
FUNCTION FBE_IRQ_CLRIRQ : BOOL
VAR_INPUT
        IrqNr: UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
IRQNR	UINT	Number of the Interrupt Request Channel, which must be cleared.

Return Value (Data type BOOL)
The function returns TRUE, if the interrupt request was successfully cleared. Otherwise FALSE is returned.

Example
The interrupt request for Interrupt channel 0 must be cleared.

```
…
VAR
        …
        Success: BOOL;                (* Use this var for Success of Boolean functions *)
        …
END_VAR

…
Success:= FBE_IRQ_CLRIRQ(0);
…
```

8. Data Bus

The Library FBE_DataBus.lib is a Library extension for the CoDeSys PLC runtime system and enables access to the microcontroller data bus for IEC61131 applications. It is an internal library; all functions are included in the runtime system.

There are two different access types:

Access type	Description
BUS	Access to the BUS means that all data bytes are read from (written to) ascending addresses. The lowest byte is read from (written to) the lowest address. Access type BUS is implemented for supporting peripherals like memories, communication controllers etc. For 16 and 32 bit data types there must be an even start address for the bus access. The library functions will force Bit0 of the address to 0 in this case.
PORT	Access to a PORT means that all data bytes are read from (written to) the same addresses. The lowest byte is read (written) first. Access type PORT is implemented for supporting peripherals like FIFOs etc.

Example

Address	Memory data
….	…
16#100103	16#04
16#100102	16#03
16#100101	16#02
16#100100	16#01
….	…

Read from data BUS:
Result:= FBE_BUS_RD32(16#100100) Result will be 16#04030201

Read from data Port
Result:= FBE_PORT_RD32(16#100100) Result will be 16#01010101

The following functions are implemented:

Function name	Description
FBE_BUS_RD8	Read a byte value from the data bus.
FBE_BUS_RD16	Read an unsigned integer value from the data bus using address auto increment
FBE_BUS_RD32	Read an unsigned long value from the data bus using address auto increment
FBE_BUS_WR8	Write a byte value to the data bus.
FBE_BUS_WR16	Write an unsigned integer value to the data bus using address auto increment
FBE_BUS_WR32	Write an unsigned long value to the data bus using address auto increment
FBE_PORT_RD8	Read a byte value from a data port.
FBE_PORT_RD16	Read an unsigned integer value from a data port (all data bytes from the same address)
FBE_PORT_RD32	Read an unsigned long value from a data port (all data bytes from the same address)
FBE_PORT_WR8	Write a byte value to a data port.
FBE_PORT_WR16	Write an unsigned integer value to a data port (all data bytes to the same address)
FBE_PORT_WR32	Write an unsigned long value to a data port (all data bytes to the same address)

8.1. FBE_BUS_RDx

Description:

Reads a byte, unsigned integer or unsigned long value from the data bus.

Declaration:

```
FUNCTION FBE_BUS_RD8 : BYTE
VAR_INPUT
        Address : UDINT;
END_VAR

FUNCTION FBE_BUS_RD16 : UINT
…
FUNCTION FBE_BUS_RD32 : UINT
```

Parameters:

Name	Data-Type	Description
AddressToReadFrom	UDINT	Address for the data bus access In case of multiple byte access: • AddressToReadFrom gives the lowest address for the least significant byte. • AddressToReadFrom must be an even address

Return Value (Data type BYTE / UINT / UDINT)

The function returns the value read from address "AddressToReadFrom"

8.2. FBE_BUS_WRx

Description:
Writes a byte, unsigned integer or unsigned long value to the data bus.

Declaration:
```
FUNCTION FBE_BUS_WR8 : BOOL
VAR_INPUT
        Address         : UDINT;
        Data    : BYTE;
END_VAR

FUNCTION FBE_BUS_WR16 : BOOL
…

FUNCTION FBE_BUS_WR32 : BOOL
…
```

Parameters:

Name	Data-Type	Description
AddressToWrite	UDINT	Address for the data bus access In case of multiple byte access: • AddressToWrite gives the lowest address for the least significant byte. • AddressToWrite must be an even address
DataToWrite	BYTE UINT UDINT	Data to write to the bus.

Return Value (Data type BOOL)
The function returns always the value "TRUE"

8.3. FBE_PORT_RDx

Description:

Reads a byte, unsigned integer or unsigned long value from a data PORT. All bytes are read from the same address.

Declaration:

```
FUNCTION FBE_PORT_RD8 : BYTE
VAR_INPUT
        Address         : UDINT;
END_VAR

FUNCTION FBE_PORT_RD16 : BYTE
…

FUNCTION FBE_PORT_RD32 : BYTE
…
```

Parameters:

Name	Data-Type	Description
AddressToReadFrom	UDINT	Address for the data bus access In case of multiple byte access: All Bytes are read from the same address, lowest Byte is read first.

Return Value (Data type BYTE / UINT / UDINT)

The function returns the value read from address "AddressToReadFrom"

8.4. FBE_PORT_WRx

Description:

Writes a byte, unsigned integer or unsigned long value to a data PORT. All bytes are written to the same address.

Declaration:

```
FUNCTION FBE_PORT_WR8 : BOOL
VAR_INPUT
        Address         : UDINT;
        Data    : BYTE;
END_VAR

FUNCTION FBE_PORT_WR16 : BOOL
…

FUNCTION FBE_PORT_WR32 : BOOL
…
```

Parameters:

Name	Data-Type	Description
AddressToWrite	UDINT	Address for the data bus access In case of multiple byte access: All Bytes are written to the same address, lowest Byte is written first.
DataToWrite	BYTE UINT UDINT	Data to write to the port.

Return Value (Data type BOOL)

The function returns always the value "TRUE"

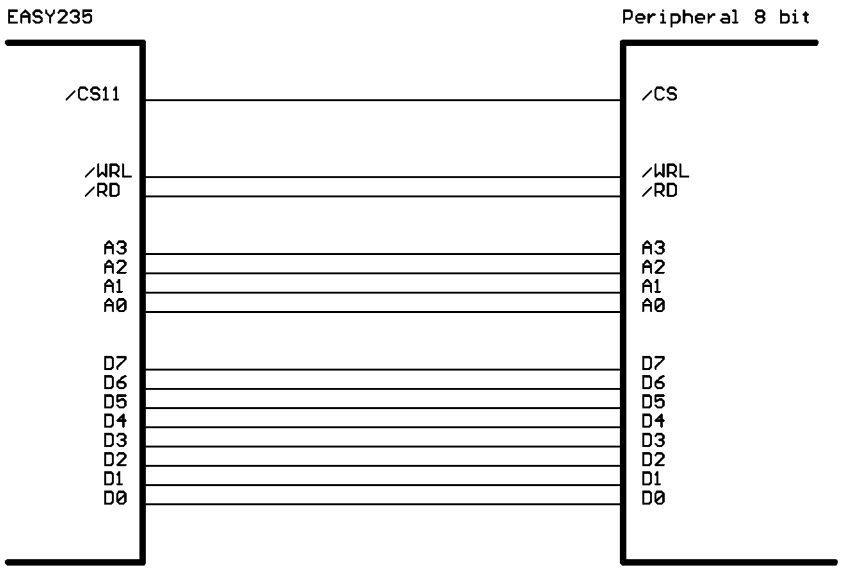
8.5. Bus Connection

Writes a byte, unsigned integer or unsigned long value to a data PORT. All bytes are written to the same address.

In order to meet correct bus interface timings, the following conditions are required.
Peripherals must use the Chip Select Lines /CS2, /CS4, /CS9, /CS10, /CS11 or /CS12.
The addressing schema from the CoDeSys application sight of few is as follows:

Chip Select	Bus Width	Address range	Remarks
/CS2	Programmable 8 or 16 bit	0 .. 16#0FFFFF	-
/CS4	8 bit	16#200000 .. 16#21FFFF	Reserved for Time Keeper
/CS9	8 bit	16#140000 .. 16#14FFFF	-
/CS10	8 bit	16#150000 .. 16#15FFFF	-
/CS11	8 bit	16#160000 .. 16#16FFFF	-
/CS12	8 bit	16#170000 .. 16#17FFFF	-

Example



9. Encoder / Event Counter

The Library FBE_Encoder.lib is a Library extension for the CoDeSys PLC runtime system and enables access to incremental encoders with 2 tracks for IEC61131 applications. The encoder counters are 32 bit counters.
The number of supported encoder channels depends on the target system.
It is an internal library; all functions are included in the runtime system.

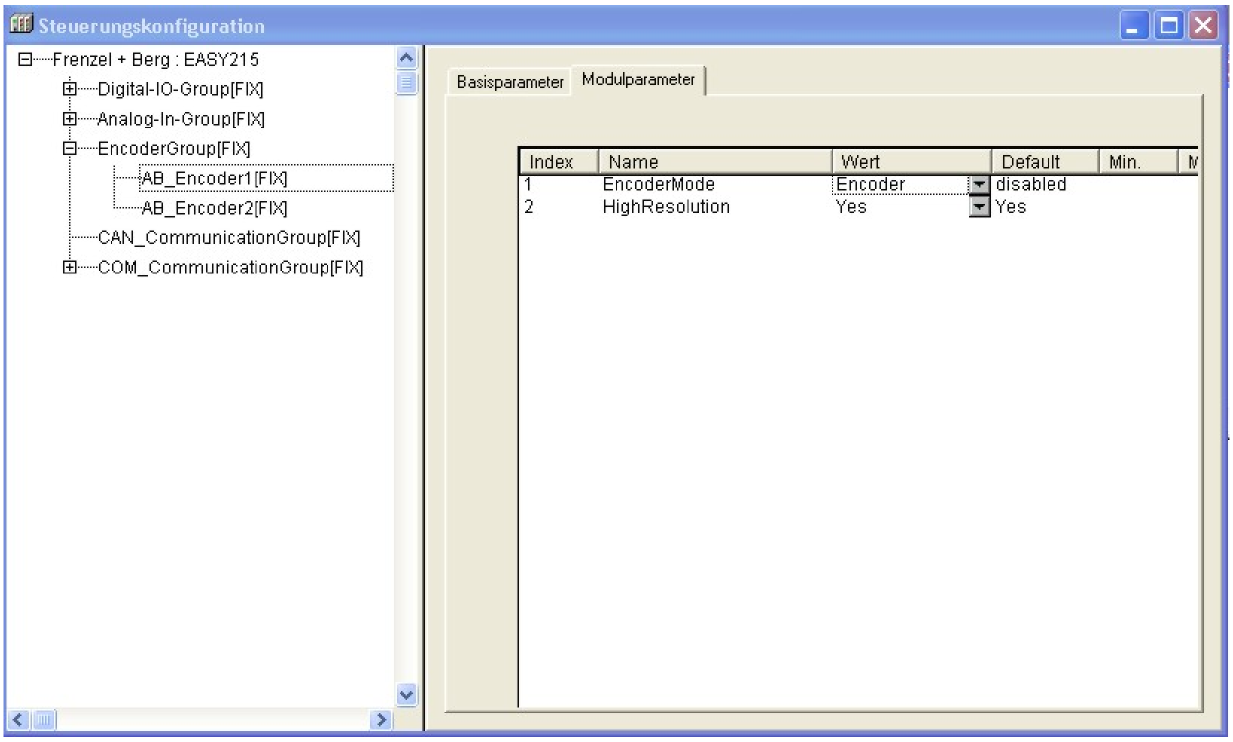
Configuration up to Firmware Version 1.100 :

The configuration of the encoder channels is done from the CoDeSys development environment. There are two parameters in the corresponding dialog of the PLC configuration window.

Access type	Description
Enable	Switches an encoder channel on or off
HighResolution	If this parameter is set to YES, the input frequency of the encoder channel is doubled.

Configuration for Firmware Version 1.101 and newer :

The configuration of the encoder channels is done from the CoDeSys development environment. The encoder channels may be also configured for event counter mode. There are two parameters in the corresponding dialog of the PLC configuration window.



Index	Name	Wert	Default	Min.
1	EncoderMode	Encoder	disabled	
2	HighResolution	Yes	Yes	

The new configuration dialog is compatible with old firmware versions, with exception that the event counter mode is not a valid selection.

Access type	Description
Encoder Mode	Selects the mode for each encoder channel. In event counter mode only pin ENCxA is used as count input pin.
HighResolution	If this parameter is set to YES, the input frequency of the encoder channel is doubled. In event counter mode: HighResolution = NO: Count on rising edge at input pin ENCxA HighResolution = YES: Count on rising and falling edge at input pin ENCxA

The following functions are implemented:

Function name	Description
FBE_ENC_CTL	Encoder Controller function This function is used to: Start/stop/Clear/Preset the encoder channel and read the count value.
FBE_ENC_SETUP	Setup or reconfiguration of an encoder channel The setup of the encoder channels is done with application startup. So the use of this function is optional and only required, if a channel must be reconfigured during run time.
FBE_ENC_READ	Read the value from the encoder as long value (32 bit position)
FBE_ENC_READ16	Read the value from the encoder as integer value (16 bit position)

9.1. Channel to Hardware Cross Reference

This reference shows the connections, which can be used on an EASY.

Typ / ENC Channel	EASY215 EASY217	EASY235		EASY2504
ENC0-A	**-**	**ENC0-A**		**INB2.6**
ENC0-B	**-**	**ENC0-B**		**INB2.4**
ENC1-A	**ENC1-A**	**ENC1-A**		**INB2.5**
ENC1-B	**ENC1-B**	**ENC1-B**		**INB1.6**
ENC2-A	**ENC2-A**	**ENC2-A**		**INB2.7**
ENC2-B	**ENC2-B**	**ENC2-B**		**INB1.7**
ENC3-A	**-**	**ENC3-A**		**INB1.5**
ENC3-B	**-**	**ENC3-B**		**-**
ENC4-A	**-**	**ENC4-A**		**INB1.4**
ENC4-B	**-**	**ENC4-B**		**-**

9.2. FBE_ENC_SETUP

Description:

Performs a setup or reconfiguration for the selected encoder channel. The setup of the encoder channels is done with application startup. So the use of this function is optional and only required, if a channel must be reconfigured during run time.

Declaration:

```
FUNCTION FBE_ENC_SETUP : BOOL
VAR_INPUT
        EncNr   : BYTE;
        DoSetup: BOOL;
        HighRes         : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
ENCNR	BYTE	Number of Encoder Channel for this function call
DOSETUP	BOOL	If set to TRUE, the setup for the selected encoder channel is performed
HIGHRES	BOOL	If set to TRUE, the input frequency of the encoder channel is doubled.

Return Value (Data type BOOL)

The function returns TRUE if the setup for the selected encoder channel was performed, otherwise false.

9.3. FBE_ENC_CTL

Description:

Controller cycle for complete access to the encoder counter and control flags.

Declaration:

```
FUNCTION FBE_ENC_CTL : DINT
VAR_INPUT
        EncNr   : BYTE;
        Enable  : BOOL;
        Clear   : BOOL;
        Preset  : BOOL;
        LoadVal         : DINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
ENCNR	BYTE	Number of Encoder Channel for this function call
ENABLE	BOOL	Enable of Encoder Channel If Channel is enabled counting of encoder pulses is enabled. Otherwise counting is stopped.
CLEAR	BOOL	If set to TRUE, the count value of the selected encoder channel is set to zero
PRESET	BOOL	If set to TRUE, the count value of the selected encoder channel is set to the value given with parameter LoadVal. Note: If parameters PRESET and CLEAR are set to TRUE at the same time CLEAR has priority and will be performed.
LOADVAL	DINT	If parameter PRESET is set to TRUE, the count value of the selected encoder channel is set to the value given with parameter LoadVal.

Return Value (Data type DINT)

The function returns the count value of the selected encoder channel.

9.4. FBE_ENC_READ

Description:
Reads the complete count value from the encoder channel. The position is returned as DINT (32 bit value)

Declaration:
```
FUNCTION FBE_ENC_READ : DINT
VAR_INPUT
        EncNr: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
ENCNR	BYTE	Number of Encoder Channel for this function call

Return Value (Data type DINT)
Position as DINT (32 bit value)

9.5. FBE_ENC_READ16

Description:
Reads only the lower word of count value from the encoder channel. The position is returned as INT (16 bit value) This function is implemented to reduce the time for encoder access for applications that request only 16 bit position range.

Declaration:
```
FUNCTION FBE_ENC_READ16 : INT
VAR_INPUT
        EncNr: BYTE;
END_VAR
```

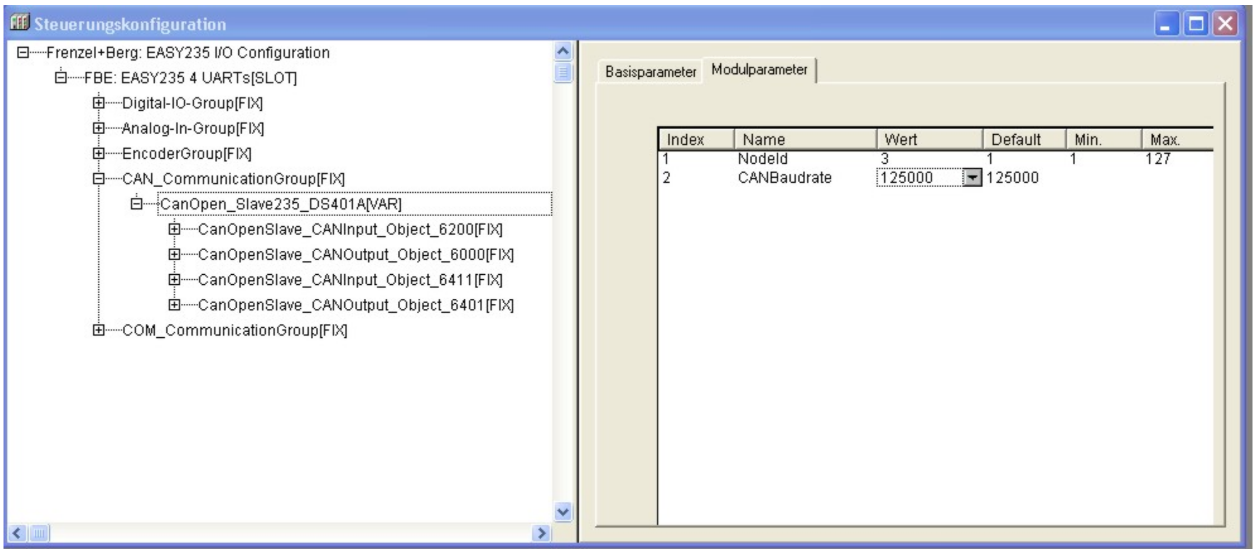
Parameters:

Name	Data-Type	Description
ENCNR	BYTE	Number of Encoder Channel for this function call

Return Value (Data type INT)
Position as INT (16 bit value)

10. CANopen Slave

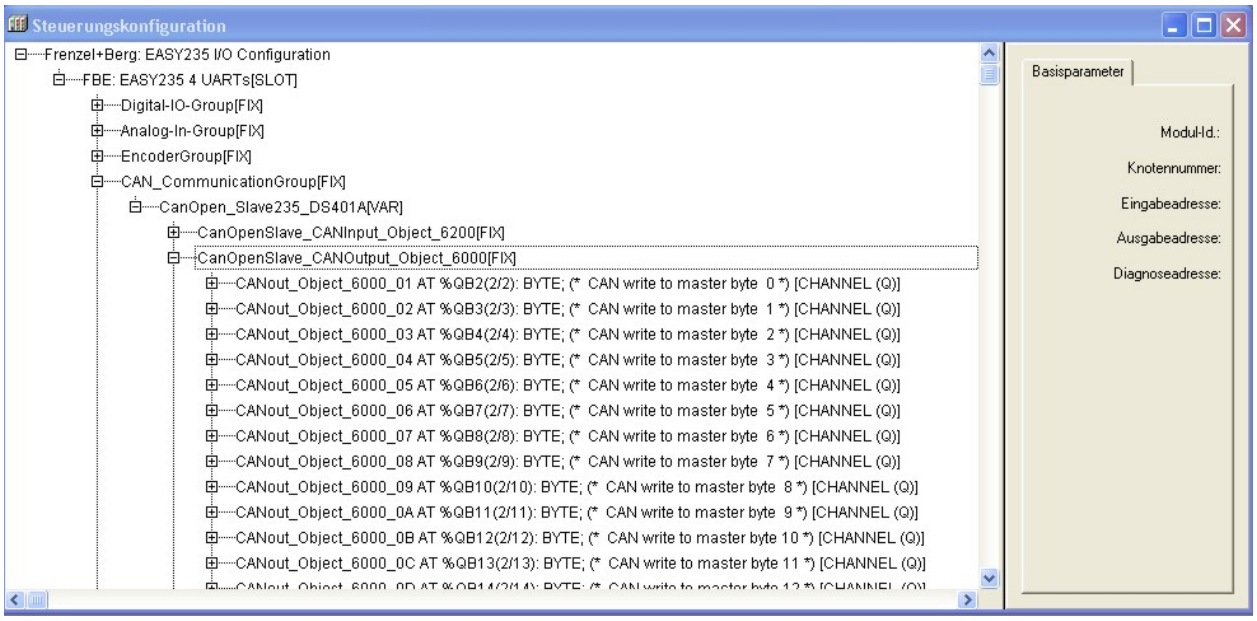
The run time system includes a CANopen slave module. If selected, the EASYxxx system acts as a CANopen slave according to DS401 (analog and digital I/O module) in a CANopen network.
The Node-ID and baud rate must be selected from the configuration dialog window.



```
⊟ Frenzel+Berg: EASY235 I/O Configuration
   ⊟ FBE: EASY235 4 UARTs[SLOT]
      ⊞ Digital-IO-Group[FIX]
      ⊞ Analog-In-Group[FIX]
      ⊞ EncoderGroup[FIX]
      ⊟ CAN_CommunicationGroup[FIX]
         ⊟ CanOpen_Slave235_DS401A[VAR]
            ⊞ CanOpenSlave_CANInput_Object_6200[FIX]
            ⊞ CanOpenSlave_CANOutput_Object_6000[FIX]
            ⊞ CanOpenSlave_CANInput_Object_6411[FIX]
            ⊞ CanOpenSlave_CANOutput_Object_6401[FIX]
      ⊞ COM_CommunicationGroup[FIX]
```

Index	Name	Wert	Default	Min.	Max.
1	NodeId	3	1	1	127
2	CANBaudrate	125000	125000		

The target system offers a number of digital I/O bytes and analog I/O integers for data transfer purposes. This data transfer memory is implemented as CANopen Object Dictionary area covering the objects 6000, 6200, 6401 and 6411. In order to transmit data from the slave to the master, it is only necessary to write to the variables named as "CANout_Object..". It is the masters job to map the related data to a PDO and set the transfer mode needed.



```
⊟ Frenzel+Berg: EASY235 I/O Configuration
   ⊟ FBE: EASY235 4 UARTs[SLOT]
      ⊞ Digital-IO-Group[FIX]
      ⊞ Analog-In-Group[FIX]
      ⊞ EncoderGroup[FIX]
      ⊟ CAN_CommunicationGroup[FIX]
         ⊟ CanOpen_Slave235_DS401A[VAR]
            ⊞ CanOpenSlave_CANInput_Object_6200[FIX]
            ⊟ CanOpenSlave_CANOutput_Object_6000[FIX]
               ⊞ CANout_Object_6000_01 AT %QB2(2/2): BYTE; (* CAN write to master byte  0 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_02 AT %QB3(2/3): BYTE; (* CAN write to master byte  1 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_03 AT %QB4(2/4): BYTE; (* CAN write to master byte  2 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_04 AT %QB5(2/5): BYTE; (* CAN write to master byte  3 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_05 AT %QB6(2/6): BYTE; (* CAN write to master byte  4 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_06 AT %QB7(2/7): BYTE; (* CAN write to master byte  5 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_07 AT %QB8(2/8): BYTE; (* CAN write to master byte  6 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_08 AT %QB9(2/9): BYTE; (* CAN write to master byte  7 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_09 AT %QB10(2/10): BYTE; (* CAN write to master byte  8 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_0A AT %QB11(2/11): BYTE; (* CAN write to master byte  9 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_0B AT %QB12(2/12): BYTE; (* CAN write to master byte 10 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_0C AT %QB13(2/13): BYTE; (* CAN write to master byte 11 *) [CHANNEL (Q)]
               ⊞ CANout_Object_6000_0D AT %QB14(2/14): BYTE; (* CAN write to master byte 12 *) [CHANNEL (Q)]
```

11. CANopen Master

The run time system includes a very powerful CANopen master. The CANopen interface is based on a two level software structure. Adding a CANopen master to the PLC configuration activates the lower level according to DS302 level. This layer handles the complete network boot up and PDO transfer automatically.
The second level according to DS405 establishes an interface for the IEC61131 application to the CANopen layer. This layer is implemented in the " FBE_CIA405.lib" library file.

Configuration:

The configuration of this master is done with the CoDeSys PLC configuration dialog. The master is enabled if there is CANopen master added to the system configuration.
The functionality and the maximum number of slaves depends on the target system.

The following functions are implemented:

For the DS302 implementation there are no special functions required. The complete network startup and PDO data transmission is handled automatically. So if the application does not need any manual SDO transfers or NMT messages, the CANopen master can run completely without the use of the CIA405 library.
There are only some functions to support the CANopen layer according to the DS405 standard.

Function name	Description
CIA405_GET_LOCAL_NODE_ID	Read the node ID of the CANopen master
CIA405_NMT	Send a NMT command to the network
CIA405_GET_KERNEL_STATE	Returns the state of the CANopen driver
CIA405_GET_STATE	Returns the CANopen node transition state of a node
CIA405_IS_ANY_EMY	Checks whether there are any emergencies saved in the masters emergency FIFO memory
CIA405_RECV_EMY	Reads one emergency from the masters emergency FIFO memory
CIA405_SDO_READ4	Read data from a slave node using SDO transfer The maximum data length is 4 bytes
CIA405_SDO_WRITE4	Write data to a slave node using SDO transfer The maximum data length is 4 bytes
CIA405_SDO_WRITE4Li	Write data to a slave node using SDO transfer, additionally this function includes information of SDO length
*init	Several Init function. This init functions are called from the operation system and should not be called by the user. Some functions are implemented as function blocks. In this case the compiler generates the complete parameter data structure for each instance of the function block. The *init functions are called automatically from the kernel driver to initialize all instance data structures.

Each function is controlled by an ENABLE input parameter, but there are two different implementation methods. The difference is static check of ENABLE parameter or transition dependent execution of the library function. The user must take care to use the correct ENABLE programming in the application software.
In case of transition dependent execution, the ENABLE and CONFIRM parameters give a complete software handshake between the IEC-61131 application and the CANopen kernel software.
See chapter: " ENABLE / CONFIRM handshake" for further details.

11.1. Data Types

In order to implement the CANopen library functions there are several new data types declared in the CANopen library.

CIA405_CANOPEN_KERNEL_ERROR

```
TYPE
        CIA405_CANOPEN_KERNEL_ERROR : WORD;
END_TYPE
```

Data type for coding of internal errors of the CANopen kernel. The CANopen kernel error is forced to zero.

CIA405_DEVICE

```
TYPE
        CIA405_DEVICE: USINT (0..127);
END_TYPE
```

Data type for coding the node ID of CANopen node

CIA405_EMY_ERROR

```
TYPE CIA405_EMY_ERROR :
STRUCT
        EMY_ERROR_CODE    : WORD;
        ERROR_REGISTER    : BYTE;
        ERROR_FIELD       : ARRAY[1..5] OF BYTE;
END_STRUCT
END_TYPE
```

Data type for coding the emergency message of a CANopen node. The CIA405_EMY_ERROR structure represents the content of the error frame. The element ERROR_REGISTER keeps the Error Register (Object 1000h) of the CAN node.

CANopen emergency frame							
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Emergency Error Code		Error Reg Object 1000h	Manufacturer specific error field				
Encoding in CIA405_EMY_ERROR data type							
EMY_ERROR_CODE		ERROR_REGISTER	ERROR_FIELD[]				
			[1]	[2]	[3]	[4]	[5]

CIA405_SDO_ERROR

```
TYPE
	CIA405_SDO_ERROR : UDINT
END_TYPE
```

Abort Code sent from a CANopen slave in case of SDO abort by the slave. Otherwise 16#05040000 for indicating a SDO time out, if the master terminates the SDO transfer.

CIA405_STATUS

```
TYPE CIA405_STATUS : (
	INIT,
	RESET_COMM,
	RESET_APP,
	PRE_OPERATIONAL,
	STOPPED,
	OPERATIONAL,
	UNKNOWN,
	NOT_AVAIL
);
END_TYPE
```

Data type for describing the NMT sate of a CANopen node.

CIA405_TRANSITION_STATE

```
TYPE CIA405_TRANSITION_STATE : (
	START_REMOTE_NODE:= 16#01,
	STOP_REMOTE_NODE:= 16#02,
	ENTER_PRE_OPERATIONAL:= 16#80,
	RESET_NODE:= 16#81,
	RESET_COMMUNICATION:= 16#82
);
END_TYPE
```

Data type for activation of NMT commands using NMT messages.

_11.2. CIA405_GET_LOCAL_NODE_ID_

Description:

Reads the node ID of the IEC-61131 PLC system. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.

Declaration:

```
FUNCTION_BLOCK CIA405_GET_LOCAL_NODE_ID
VAR_INPUT
        ENABLE          : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM         : BOOL:= FALSE;
        DEVICE                  : CIA405_DEVICE;
END_VAR
```

Parameters:

Parameter Name	Data-Type	Description
ENABLE	BOOL	TRUE will enable this function
CONFIRM	BOOL	TRUE, if the function was successfully completed. FALSE otherwise.
DEVICE	CIA405_DEVICE	Node ID of the device.

11.3. CIA405_NMT

Description:

Sends a NMT command to the CANopen network. This command may be used, if slaves transition states must be changed while the network is still running.

Declaration:

```
FUNCTION_BLOCK CIA405_NMT
VAR_INPUT
        DEVICE                  : CIA405_DEVICE;
        STATE           : CIA405_TRANSITION_STATE;
        ENABLE          : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM         : BOOL;
        ERROR                   : CIA405_CANOPEN_KERNEL_ERROR;
END_VAR
VAR
END_VAR
```

Parameters:

Parameter Name	Data-Type	Description
DEVICE	CIA405_DEVICE	ID of the device for sending the NMT command
TRANSITION_STATE	CIA405_TRANSITION _STATE	CANopen NMT state. The slave “DEVICE” See “Data Types” for further details
ENABLE	BOOL	A FALSE to TRUE transition will enable this function
CONFIRM	BOOL	TRUE, if the function was successfully completed. FALSE otherwise.
ERROR	CIA405_CANOPEN _KERNEL_ERROR	Don’t care

11.4. CIA405_GET_KERNEL_STATE

Description:

Reads the error state of the CANopen kernel driver. For the current version always ZERO is returned. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.

Declaration:

```
FUNCTION_BLOCK CIA405_GET_KERNEL_STATE
VAR_INPUT
        ENABLE          : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM         : BOOL:= FALSE;
        STATE           : CIA405_CANOPEN_KERNEL_ERROR;
END_VAR
VAR
END_VAR
```

Parameters:

Parameter Name	Data-Type	Description
ENABLE	BOOL	TRUE will enable this function
CONFIRM	BOOL	TRUE, if the function was successfully completed. FALSE otherwise.
STATE	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.

11.5. CIA405_GET_STATE

Description:

Reads the NMT state of a selected CANopen node. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.

Declaration:

```
FUNCTION_BLOCK CIA405_GET_STATE
VAR_INPUT
        DEVICE                  : CIA405_DEVICE;
        ENABLE          : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM         : BOOL:= FALSE;
        STATUS          : CIA405_STATUS;
END_VAR
```

Parameters:

Parameter Name	Data-Type	Description
DEVICE	CIA405_DEVICE	Node ID from which the NMT state is requested
ENABLE	BOOL	TRUE will enable this function
CONFIRM	BOOL	TRUE, if the function was successfully completed. FALSE otherwise.
STATUS	CIA405_STATUS	NMT state of the selected node. If ENABLE = FALSE the status will return CO302_NMTSTATE_UNKNOWN

11.6. CIA405_IS_ANY_EMY

Description:

Checks whether there are any emergencies stored in the emergency FIFO memory.

Declaration:

```
FUNCTION CIA405_IS_ANY_EMY : BOOL
VAR_INPUT
        Enable: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
ENABLE	BOOL	TRUE will enable this function

Return Value (Data type BOOL)

The function returns TRUE if there are any emergency messages stored in the emergency FIFO memory, otherwise false.

11.7. CIA405_RECV_EMY

Description:

Reads the oldest emergency message stored in the emergency FIFO memory. Reading of an emergency will also delete this message from the FIFO memory, so each message can only be read once. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.

Declaration:

```
FUNCTION_BLOCK CIA405_RECV_EMY
VAR_INPUT
        ENABLE          : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM         : BOOL;
        DEVICE                  : CIA405_DEVICE;
        ERROR                   : CIA405_CANOPEN_KERNEL_ERROR;
        EMY_ERROR       : CIA405_EMY_ERROR;
END_VAR
```

Parameters:

Name	Data-Type	Description
ENABLE	BOOL	A FALSE to TRUE transition will enable this function
CONFIRM	BOOL	TRUE, if the function was successfully completed and output data is valid. FALSE otherwise.
DEVICE	CIA405_DEVICE	Node ID of the CANopen node that produced this emergencies 0 if the emergency was created from the CANopen master
ERROR	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.
EMY_ERROR	CIA405_EMY _ERROR	Emergency message of the CANopen node.

See below for details

The Function may either return an emergency sent by a slave (external emergency) or an emergency created from the CANopen Master itself (internal emergency)

External Emergency:

Sent by an external CANopen slave. In this case the Variable EMY_ERROR exactly represents the values transmitted by the connected Slave.

Name	Data-Type	Description
DEVICE	CIA405_DEVICE	Node ID of the CANopen node that produced this emergencies
ERROR	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.
EMY_ERROR.	CIA405_EMY _ERROR	Emergency message of the CANopen node.
Emy_Error_Code		Error Code sent from the slave within the emergency message
Error_Register		Error Register sent from the slave within the emergency message
Error_Field[1]..[5]		Error Field sent from the slave within the emergency message

Internal Emergency:

Created from the CANopen master. In this case the Variable EMY_ERROR shows the slave number that caused the Emergency of the master.

Name	Data-Type	Description
DEVICE	CIA405_DEVICE	0: Always zero to indicate, that the emergency was created from the master firmware and not transmitted over the CAN network
ERROR	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.
EMY_ERROR.	CIA405_EMY _ERROR	Emergency message of the CANopen master.
Emy_Error_Code		Error Code created from the master firmware
Error_Register		1: Error is set 0: Information for no Error (or automatically fixed error)
Error_Field[1]		Slave-ID that caused the emergency For example if the node guarding of a connected slave fails, the node ID of this slave is reported in Error_Field[1].
Error_Field[2]..[5]		0x00000000 The CANopen master was not able to check the exact reason for the emergency. For example bus errors or distortions may cause such entries in the emergency FIFO. 0x00000001 There was a guarding error detected at this slave (node guarding or heartbeat will not be distinguished) 0x00000002 The slave answered a SDO transfer with an Abort SDO message other codes reserved for future use

11.8. CIA405_SDO_READ4

Description:

Read data from a slave node using SDO transfer. Maximum size of data is 4 bytes. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.

Declaration:

```
FUNCTION_BLOCK CIA405_SDO_READ4
VAR_INPUT
        DEVICE          : CIA405_DEVICE;
        INDEX           : WORD;
        SUBINDEX        : BYTE;
        ENABLE          : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM         : BOOL:= FALSE;
        ERROR                   : CIA405_CANOPEN_KERNEL_ERROR;
        ERRORINFO       : CIA405_SDO_ERROR;
        DATA            : ARRAY[1..4] OF BYTE;
        DATALENGTH : USINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
DEVICE	CIA405_DEVICE	Node ID from which the must be read.
INDEX	WORD	Index of the nodes object dictionary for data access
SUBINDEX	BYTE	Sub-Index of the nodes object dictionary for data access
ENABLE	BOOL	A FALSE to TRUE transition will start the SDO transmission
CONFIRM	BOOL	TRUE, if the SDO transfer was completed. FALSE otherwise. The result of the SDO transfer must be checked by reading the ERRORINFO parameter
ERROR	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.
ERRORINFO	CIA405_SDO _ERROR	Abort Code in case of the SDO transfer fails. Zero if the SDO transfer was successfully completed and the data is valid
DATA	ARRAY[1..4] OF BYTE	Data field representing the result of the SDO transmission. The data field must be converted to the requested data type by calling the corresponding function.
DATALENGTH	USINT	Length of the valid data field

11.9. CIA405_SDO_WRITE4

Description:

Write data to a slave node using SDO transfer. Maximum size of data is 4 bytes. The implementation is done as function block. This will cause the automatic implementation of a data structure according to the parameters.

Declaration:

```
FUNCTION_BLOCK CIA405_SDO_WRITE4
VAR_INPUT
        DEVICE       : CIA405_DEVICE;
        INDEX        : WORD;
        SUBINDEX     : BYTE;
        DATA         : ARRAY[1..4] OF BYTE;
        DATALENGTH   : USINT;
        ENABLE       : BOOL;
END_VAR
VAR_OUTPUT
        CONFIRM      : BOOL:= FALSE;
        ERROR                : CIA405_CANOPEN_KERNEL_ERROR;
        ERRORINFO    : CIA405_SDO_ERROR;
END_VAR
```

Parameters:

Name	Data-Type	Description
DEVICE	CIA405_DEVICE	Node ID to which the data must be written.
INDEX	WORD	Index of the nodes object dictionary for data access
SUBINDEX	BYTE	Sub-Index of the nodes object dictionary for data access
DATA	ARRAY[1..4] OF BYTE	Data field representing the source data for the SDO transmission. The data field must be converted from the requested data type by calling the corresponding function.
DATALENGTH	USINT	Length of the valid data field
ENABLE	BOOL	A FALSE to TRUE transition will start the SDO transmission
CONFIRM	BOOL	TRUE, if the SDO transfer was completed. FALSE otherwise. The result of the SDO transfer must be checked by reading the ERRORINFO parameter
ERROR	CIA405_CANOPEN _KERNEL_ERROR	State of the CANopen kernel software.
ERRORINFO	CIA405_SDO _ERROR	Abort Code in case of the SDO transfer fails. Zero if the SDO transfer was successfully completed and the data is valid

11.10. CIA405_SDO_WRITE4Li

Description:
See Description 10.9. CIA405_SDO_WRITE, additionally this function includes information of SDO length.

11.11. ENABLE / CONFIRM handshake

The enable parameter takes control of all library functions. There are two different implementation methods. The difference is static check of ENABLE parameter or transition depended execution of the library function. The user must take care to use the correct ENABLE programming in the application software.

Static check of ENABLE

The functions will work always, if the ENABLE input parameter is scanned as TRUE.
Example: CIA405_GET_LOCAL_NODE_ID

Transition dependent execution

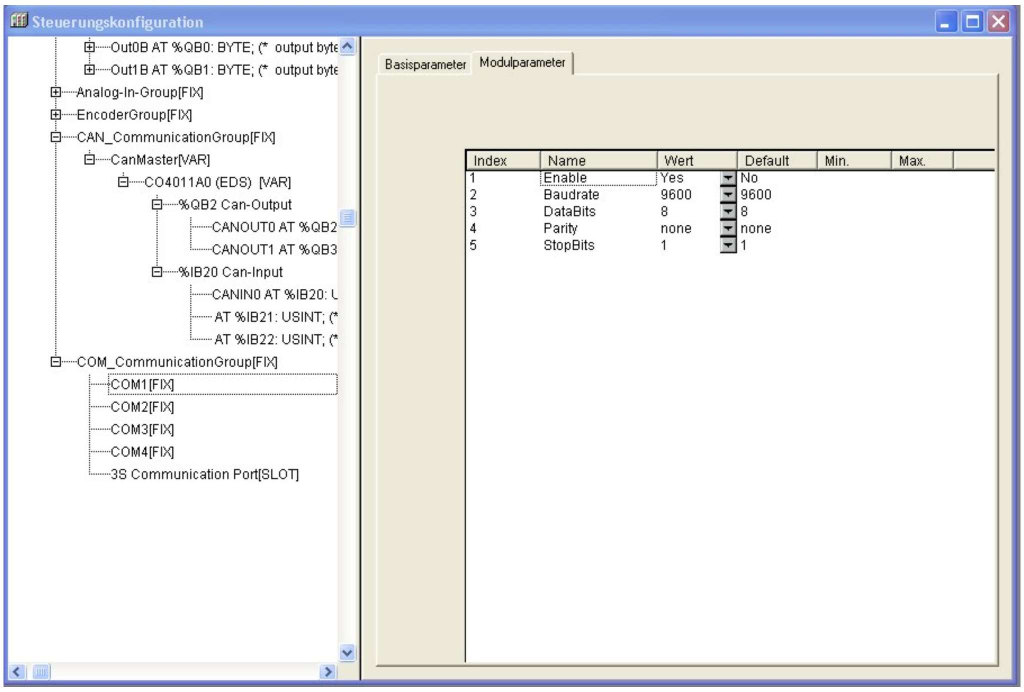
The function will be started with a FALSE to TRUE transition of the ENABLE input parameter. The transition starts the function for example the SDO transfer to a slave node. During the time the function is working, the ENABLE input parameter must be held on TRUE, otherwise the function will be terminated. For example the SDO transfer to a slave node will be aborted by sending a SDO abort message to the slave node.
If the ENABLE input parameter is held at TRUE the function will set the CONFIRM output parameter as soon as the function was finished successfully. For example the master has received the SDO answer message. After CONFIRM is set TRUE the application software may read the result, for example the SDO data, keeping the ENABLE input parameter still on TRUE. After processing of the functions output data is finished, the ENABLE input parameter can be set FALSE. This will cause the end of the function. The output parameter CONFIRM is set to FALSE by the operation system.
The ENABLE and CONFIRM parameters give a complete software handshake between the IEC-61131 application and the CANopen kernel software.
Example: CIA405_SDO_WRITE4

12. Serial Interface / COM

The Library FBE_COM.lib is a Library extension for the CoDeSys PLC runtime system to support serial COM interfaces from within IEC61131-3 applications. The COM interfaces are handled by the operation system. For each COM interface there is a transmit and a receive FIFO buffer.

Target Harware	Receive FIFO size	Transmit FIFO size
EASY235-CR24-L4	512 Bytes	512 Bytes
EASY2504	512 Bytes	512 Bytes

Configuration of the serial interfaces can be done either by using the CoDeSys System Configuration Dialog, or by using the libraries init function.



```
Out0B AT %QB0: BYTE; (*  output byte
Out1B AT %QB1: BYTE; (*  output byte
Analog-In-Group[FIX]
EncoderGroup[FIX]
CAN_CommunicationGroup[FIX]
  CanMaster[VAR]
    CO4011A0 (EDS)  [VAR]
      %QB2 Can-Output
        CANOUT0 AT %QB2
        CANOUT1 AT %QB3
      %IB20 Can-Input
        CANIN0 AT %IB20: U
        AT %IB21: USINT; (*
        AT %IB22: USINT; (*
COM_CommunicationGroup[FIX]
  COM1[FIX]
  COM2[FIX]
  COM3[FIX]
  COM4[FIX]
  3S Communication Port[SLOT]
```

Index	Name	Wert	Default	Min.	Max.
1	Enable	Yes	No		
2	Baudrate	9600	9600		
3	DataBits	8	8		
4	Parity	none	none		
5	StopBits	1	1		

The following functions are implemented:

Function name	Description
FBE_COM_INIT	Initialize COM interface
FBE_COM_RXBUFNUM	Get number of received characters
FBE_COM_READ	Read single character from the serial interface
FBE_COM_RDSTR	Read string from serial channel
FBE_COM_STATUS	Read status of serial channel
FBE_COM_WRITE	Write single character to the serial channel
FBE_COM_WRSTR	Write string to the serial channel
FBE_COM_CLEAR	Clears the receiver register and receiver buffer
FBE_COM_CLOSE	Closes a COM interface. Receiver and transmitter will be disabled.
FBE_COM_REOPEN	Opens a COM interface again with last parameters. Receiver and transmitter register and buffer will cleared.
FBE_COM_RXREADY	Checks for characters in the receiver buffer. Returns TRUE if minimum one character is in the receiver buffer.

12.1. Data Types

In order to implement the serial COM library functions there are several new data types declared in the library FBE_COM.LIB. It is strongly recommended to use this data types with the library functions, even if the replaced constant would be also processed by the CoDeSys compiler without any problems.

Type_COM_NR

Data type to select the requested serial channel.

```
TYPE type_COM_NR : (
        COM1:= 0,
        COM2:= 1,
        COM3:= 2,
        COM4:= 3
);
END_TYPE
```

TYPE_BAUD

Data type to set the requested baud rate for the serial channel

```
TYPE type_BAUD : (
        BAUD_1200:= 12,
        BAUD_2400:= 24,
        BAUD_4800:= 48,
        BAUD_9600:= 96,
        BAUD_19200:= 192,
        BAUD_38400:= 384,
        BAUD_57600:= 576
);
END_TYPE
```

type_PARITY

Data type for setting parity selection for a serial channel

```
TYPE type_PARITY : (
	PARITY_EVEN:= 69,
	PARITY_ODD:= 79,
	PARITY_NONE:= 78
);
END_TYPE
```

type_DATA_BITS, type_STOP_BITS

This data types are direct replacements of data type INT

```
TYPE type_DATA_BITS : INT;
END_TYPE
TYPE type_STOP_BITS :  INT;
END_TYPE
```

12.2. FBE_COM_INIT

Description:

Initializes a COM interface and opens it for data transfer operations. If the user configures the serial channel within the CoDeSys system configuration dialog, there is no need to call the FBE_COM_INIT function.

Declaration:

```
FUNCTION FBE_COM_INIT : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
        Baud            : type_BAUD;
        DataBits                : type_DATA_BITS;
        Parity          : type_PARITY;
        StopBits                : type_STOP_BITS;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface
Baud	type_BAUD	Baudrate for serial data transmission
Data Bits	type_DATA_BITS	Number of data bits for serial data transmission
Parity	type_PARITY	Parity information for serial data transmission
StopBits	type_STOP_BITS	Number of Stop Bits for serial data transmission

Return Value (Data type BOOL)

TRUE	If initialization of the serial channel was successful
FALSE	If initialization of the serial channel failed

12.3. FBE_COM_STATUS

Description:

Returns the status of a serial COM interface.

Declaration:

```
FUNCTION FBE_COM_STATUS : BYTE
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR

END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr		Number of the serial interface

Return Value (Data type BYTE)

The function FBE_COM_STATUS returns the status of a serial COM interface in a Byte.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TxOVL	RxOVL	-	-	-	TxRDY	TxEM	RxRDY

RxRDY	Receiver Ready 1: The COM interface has received one or more characters. 0: There are no received characters stored to the receiver FIFO buffer
TxEM	Transmitter empty 1: There are no more characters in the transmission FIFO buffer. 0: There are characters in the transmitter FIFO
TxRDY	Transmitter Ready 1: The transmitter FIFO buffer is ready for storing additional characters. 0: The transmitter FIFO buffer is full. Do not start any further transmissions.
RxOVL	Receiver Overflow 1: There was an overflow of the receiver FIFO buffer. There are some lost characters. 0: No overflow occurred. The Overflow flag is reset after reading the status byte using function FBE_COM_STATUS. So this overflow can only be read for one time.
TxOVL	Transmitter Overflow 1: There was an overflow of the transmitter FIFO buffer. There are some lost characters. 0: No overflow occurred. The Overflow flag is reset after reading the status byte using function FBE_COM_STATUS. So this overflow can only be read for one time.

12.4. FBE_COM_RXBUFNUM

Description:
Returns the number of characters stored in the receiver FIFO buffer of a serial COM interface.

Declaration:
```
FUNCTION FBE_COM_RXBUFNUM : UINT
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface

Return Value (Data type UINT)
Number of characters stored in the receiver FIFO buffer.

12.5. FBE_COM_READ

Description:
Read one character from the receiver FIFO buffer.

Declaration:
```
FUNCTION FBE_COM_READ : BYTE
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface

Return Value (Data type BYTE)
One character from the receiver FIFO buffer. If there is no received character in the buffer, the function returns “0”.

12.6. FBE_COM_RDSTR

Description:

Read a complete String from the receiver FIFO buffer. The string is either terminated with character ZERO or if the maximum string length is exceeded.

Declaration:

```
FUNCTION FBE_COM_RDSTR : UINT
VAR_INPUT
        ComNr           : type_COM_NR;
        StringData      : STRING;
        MaxLen                  : UINT;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface
StringData	STRING	Destination where the function has to copy the string to.
MaxLen	UINT	Maximum valid length of this string.

Return Value (Data type UINT)

The function returns the length of the received string in bytes

12.7. FBE_COM_WRITE

Description:

Writes a single character to the transmitter FIFO buffer.

Declaration:

```
FUNCTION FBE_COM_WRITE : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
        Data            : BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface
Data	BYTE	Data Byte to transmit

Return Value (Data type BOOL)

TRUE	If transmission of the data byte was successful
FALSE	If transmission of the data byte failed

12.8. FBE_COM_WRSTR

Description:

Writes a complete string to the transmitter FIFO buffer. The string must be terminated be a character ZERO.

Declaration:

```
FUNCTION FBE_COM_WRSTR : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
        StringData      : STRING;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface
StringData	STRING	String Data to transmit

Return Value (Data type BOOL)

TRUE	If transmission of the string was successful
FALSE	If transmission of the string failed

12.9. FBE_COM_CLEAR

Description:

Clears the receiver register and receiver buffer.

Declaration:

```
FUNCTION FBE_COM_CLEAR : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface

Return Value (Data type BOOL)

TRUE If function was successful
FALSE If execution of this function failed

12.10. FBE_COM_CLOSE

Description:

Closes a COM interface. Receiver and transmitter will be disabled.

Declaration:

```
FUNCTION FBE_COM_CLOSE : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface

Return Value (Data type BOOL)

TRUE If function was successful
FALSE If execution of this function failed

12.11. FBE_COM_REOPEN

Description:

Opens a COM interface again with last parameters. Receiver and transmitter register and buffer will cleared. The functions FBE_COM_CLOSE and FBE_COM_REOPEN may be used to block serial reception for some time.

Declaration:

```
FUNCTION FBE_COM_REOPEN : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface

Return Value (Data type BOOL)

TRUE	If function was successful
FALSE	If execution of this function failed

12.12. FBE_COM_RXREADY

Description:

Check for characters in the receiver buffer. Returns TRUE if minimum one character is in the receiver buffer.

Declaration:

```
FUNCTION FBE_COM_RXREADY : BOOL
VAR_INPUT
        ComNr           : type_COM_NR;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
ComNr	type_COM_NR	Number of the serial interface

Return Value (Data type BOOL)

TRUE	If one or more character is/are in receiver buffer.
FALSE	If receiver buffer is empty.

13. BasicCAN

The Library FBE_BasicCan.lib is a Library extension for the CoDeSys PLC runtime system and provides sending and receiving of CAN frames without the help of the integrated CANopen interface. Nevertheless it is necessary to add either the CANopen master or the CANopen slave to the system configuration. In any case all CAN-Identifiers that belong to the CANopen interface will be processed by the CANopen interface and will not be available for the basic CAN library.

The following functions are implemented:

Function name	Description
BASICCAN_RESET	Resets the Basic CAN library
BASICCAN_RXMSG	Receives a CAN message
BASICCAN_RXREADY	Checks whether there are received CAN frames
BASICCAN_TXMSG	Transmits a CAN message

13.1. BASICCAN_RESET

Description:

If Enable = TRUE Resets the Basic CAN library. The receiver buffer will be cleared.

Declaration:

```
FUNCTION BASICCAN_RESET : BOOL
VAR_INPUT
        Enable          : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Enable	BOOL	TRUE will enable this function

Return Value (Data type BOOL)

The function returns the value of Input parameter “Enable”.

13.2. BASICCAN_RXMSG

Description:

Reads the next CAN message from receiver buffer.

Declaration:

```
FUNCTION_BLOCK BASICCAN_RXMSG
VAR_INPUT
END_VAR
VAR_OUTPUT
        Id      : UDINT;
        Data    : ARRAY[0..7] OF BYTE;
        Length  : BYTE;
        Valid   : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Id	UDINT	Message Identifier
Data	ARRAY[0..7 OF BYTE]	Data bytes of the CAN message
Length	BYTE	Length of CAN message / received number of data bytes
Valid	BOOL	TRUE if the received CAN message was detected as valid CAN frame

13.3. BASICCAN_RXREADY

Description:

Check for received massages. If one or more received messages are in the receiver buffer, this function returns TRUE. If buffer is empty (no message has received) FALSE will be returned.

Declaration:

```
FUNCTION BASICCAN_RXREADY : BOOL
VAR_INPUT
        Enable          : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Enable	BOOL	TRUE will enable this function

Return Value (Data type BOOL)

TRUE	If one or more CAN messages is/are in the receiver buffer.
FALSE	If receiver buffer is empty.

13.4. BASICCAN_TXMSG

Description:
Stores a CAN message into the transmitter buffer.

Declaration:

```
FUNCTION_BLOCK BASICCAN_TXMSG
VAR_INPUT
        Id              : UDINT;
        Data            : ARRAY[0..7] OF BYTE;
        Length          : BYTE;
END_VAR
VAR_OUTPUT
        Success                 : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Id	UDINT	Message Identifier
Data	ARRAY[0..7] OF BYTE	Data bytes of the CAN message
Length	BYTE	Length of CAN message / number of data bytes to transmit
Success	BOOL	True if transmission of the CAN message was successful

14. FASTADC / Analog to Digital

The Library FBE_FastADC.lib is a Library extension for the CoDeSys PLC runtime system and supports direct access to the A/D converter in order to support high speed A/D conversion within interrupt service routines or fast tasks. A conversion process always processes all channels.

The following functions are implemented:

Function name	Description
FastADC_IsConvActive	Check for currently active conversion process
FastADC_Read	Read A/D converter value
FastADC_StartConv	Start next conversion

14.1. FASTADC_ISCONVACTIVE

Description:

Checks for currently active conversion process. A currently active conversion must not be disturbed by additional conversion start commands. If a conversion is still active the conversion result is undefined.

Declaration:

```
FUNCTION FastADC_IsConvActive : BOOL
VAR_INPUT
        Dummy           : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Unused only for compatibility reasons (use TRUE)

Return Value (Data type BOOL)

TRUE if conversion is active
FALSE if no conversion is active

14.2. FASTADC_READ

Description:
Reads the conversion values for all channels to the array addressed with the pointer given with the parameter.

Declaration:

```
FUNCTION FastADC_Read : BOOL
VAR_INPUT
        AnalogMem     : POINTER TO ARRAY[0..7] OF INT;
END_VAR
```

Parameters:

Name	Data-Type	Description
AnalogMem	POINTER TO ARRAY[0..7] OF INT	

Return Value (Data type BOOL)
TRUE if reading was successful
FALSE if reading failed.

14.3. FASTADC_STARTCONV

Description:
Starts a conversion cycle for all analog channels.

Declaration:

```
FUNCTION FastADC_StartConv : BOOL
VAR_INPUT
        Dummy         : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	

Return Value (Data type BOOL)
TRUE if successful
FALSE if starting failed.

15. FIFO

The Library FBE_Fifo.lib is a Library extension for the CoDeSys PLC runtime system and may be used for buffering data with user-defined data size. The Buffer works “first in – first out”.

The following functions are implemented:

Function name	Description
Fifo_Create	Creates a fifo-buffer for user defined size and data-types
Fifo_GetDepth	Returns the bnumber of entries pushed to the FiFo
Fifo_IsEmpty	Returns TRUE if fifo is empty
Fifo_IsFull	Returns TRUE if fifo is full
Fifo_Pop	Takes next element/data out of the fifo
Fifo_Push	Stores one element/data into the fifo buffer
Fifo_Reset	Clears the fifo-buffer

15.1. Data Types

In order to implement the CANopen library functions is a new data types declared in the CANopen library.

TFifo

```
TYPE TFifo : POINTER TO BYTE;
END_TYPE
```

15.2. FIFO_CREATE

Description:

Creates a fifo-buffer with defined size and data-types. The Fifo will be created for the maximum quatity of data elements that fits in the given FiFo memory size.

Declaration:

```
FUNCTION Fifo_Create : TFifo
VAR_INPUT
          Memory                 : TFifo;
          FifoByteSize    : UINT;
          DataSize        : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Memory	TFifo	Pointer to the memory location where the FiFo will be created.
FifoByteSize	UINT	Size of fifo-buffer in bytes
DataSize	UINT	Size of one element in byte

Return Value (Data type TFifo)

Pointer to the FiFo that was created. Use this pointer to access this FiFo with the library functions.

15.3. FIFO_GETDEPTH

Description:

Returns the depth of the buffer in elements/entries that can store additional.

Declaration:

```
FUNCTION Fifo_GetDepth : UINT
VAR_INPUT
          Fifo: TFifo;
END_VAR
```

Parameters:

Name	Data-Type	Description
Fifo	TFifo	Pointer for fifo selection for this function call

Return Value (Data type UINT)

Number of elements that are currently saved to the FiFo memory.

15.4. FIFO_ISEMPTY

Description:
Checks the state of the fifo-buffer. Returns TRUE if fifo is empty

Declaration:
```
FUNCTION Fifo_IsEmpty : BOOL
VAR_INPUT
        Fifo: TFifo;
END_VAR
```

Parameters:

Name	Data-Type	Description
Fifo	TFifo	Pointer for fifo selection for this function call

Return Value (Data type BOOL)

TRUE	If fifo is empty.
FALSE	IF one or more elements are in the fifo.

15.5. FIFO_ISFULL

Description:
Checks the state of the fifo-buffer. Returns TRUE if fifo is full.

Declaration:
```
FUNCTION Fifo_IsFull : BOOL
VAR_INPUT
        Fifo: TFifo;
END_VAR
```

Parameters:

Name	Data-Type	Description
Fifo	BOOL	Pointer for fifo selection for this function call

Return Value (Data type BOOL)

TRUE	If fifo is empty.
FALSE	If one or more elements are in the fifo.

15.6. FIFO_POP

Description:

Takes next element out of the fifo and stores it to destination.

Declaration:

```
FUNCTION Fifo_Pop : BOOL
VAR_INPUT
        Fifo            : TFifo;
        PtrDestination  : POINTER TO BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Fifo	Tfifo	Pointer for fifo selection for this function call
PtrDestination	POINTER TO BYTE	Pointer to destination where data is stored to

Return Value (Data type BOOL)

TRUE if there was one element popped from the FiFo memory.
FALSE if there was nothing to pop

15.7. FIFO_PUSH

Description:

Puts an element into the fifo, taken from given source.

Declaration:

```
FUNCTION Fifo_Push : BOOL
VAR_INPUT
        Fifo            : TFifo;
        PtrSource       : POINTER TO BYTE;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
Fifo	Tfifo	Pointer for fifo selection for this function call
PtrSource	POINTER TO BYTE	Pointer to source data-element which must be stored to buffer

Return Value (Data type BOOL)

TRUE if pushing this element to the FiFo was successful.
FALSE if the FiFo was already full or other reasons caused the function to fail.

15.8. FIFO_RESET

Description:

Clears the fifo-buffer. After running this function the buffer is empty.

Declaration:

```
FUNCTION Fifo_Reset : BOOL
VAR_INPUT
        Fifo: TFifo;
END_VAR
VAR
END_VAR
```

Parameters:

Name	Data-Type	Description
Fifo	TFifo	Pointer for fifo selection for this function call

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

16. KEYBOARD

The Library FBE_Keyboard.lib is a Library extension for the CoDeSys PLC runtime system. Depending on hardware the library runs with matrix-keyboard or standard PC-keyboard or both. All key-hits were translated in one byte values and stored in key-fifo-buffer. The translation-table of the PC-keyboard is fix implemented. For the matrix-keyboard a default translation-table is used after initialization. Assigning a user-defined table is possible. For default key-codes see tables below.

Key-fifo-buffer size:	32 bytes
Keyboard matrix dimensions:	8 x 6 (rows x lines)

Default key-codes of matrix-keyboard: (key-code = (line * 8) + row) + 16#30

row line	0	1	2	3	4	5	6	7
0	48	49	50	51	52	53	54	55
1	56	57	58	59	60	61	62	63
2	64	65	66	67	68	69	70	71
3	72	73	74	75	76	77	78	79
4	80	81	82	83	84	85	86	87
5	88	89	90	91	92	93	94	95

(For a user defined table create a variable as an ARRAY[0..7, 0..5] and fill in values for possible row/line-points and use FBE_KBD_NEWCODEMATRIX for assigning)

Key-codes of PC-keyboard:

All ASCII-keys (A .. Z, a..z, 0 .. 9, %, §, RETURN, SPACE, and so on) shall translated to their associated ASCII-code-value (A = 65, 1=49, RETURN=13, ESC=27 and so on).
All function-key (F1, F2, …, F12) shall translated to hex values (16#F1, 16#F2, …, 16#FC).

The other key-codes shall translated as shown in this table:

Key	Code	Key	Code	Key	Code	Key	Code
Arrow-Right	16#1C	Arrow-Left	16#1D	Arrow-Up	16#1E	Arrow-Down	16#1F
Insert	16#E0	Delete	16#E1				

The following functions are implemented:

Function name	Description
FBE_KBD_INIT	Initialize KBD interface
FBE_KBD_ISKEY	Examines whether any key is in key-buffer
FBE_KBD_NEWCODEMATRIX	Assigns a new key table to the matrix keyboard
FBE_KBD_READKEY	Takes naxt key value from the key buffer
FBE_KBD_RESET	Clears the keyboard buffer
FBE_KBD_WAITKEY	Pushes a key value from the keyboard buffer by waiting for next key entry if buffer is empty.
FBE_KBD_WRITEBACKKEY	Puts a value into the key-fifo-buffers front position

16.1. FBE_KBD_INIT

Description:
Initializes a matrix- and/or a PC-keyboard interface and installs a key-fifo-buffer.

Declaration:
```
FUNCTION FBE_KBD_INIT : BOOL
VAR_INPUT
        Dummy: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BYTE	Value don’t care (for future use)

Return Value (Data type BOOL)
Returns TRUE if initialization was successful.

16.2. FBE_KBD_ISKEY

Description:

This function is looking for key-entries in the key-fifo-buffer. It returns TRUE, if key-fifo-buffer is not empty.

Declaration:

```
FUNCTION FBE_KBD_ISKEY : BOOL
VAR_INPUT
        Dummy: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BYTE	Value don't care (for future use)

Return Value (Data type BOOL)

Returns TRUE, if key-fifo-buffer is not empty, else FALSE.

16.3. FBE_KBD_ NEWCODEMATRIX

Description:

This function is used to change the key-code-translation-table for the keyboard (matrix-keyboard only). This allows the user to get defined values for key-hits. Independent of the connected matrix-keyboard a key-code-translation-table must have 48 entries (8 rows x 6 lines). The table can be defined as an ARRAY[0..7, 0..5]. Smaller matrix-keyboards uses then a smaller 2 dimension area inside that table. Normally (after init) a default table is used for key-code-translation. For more details, see top of this chapter “KEYBOARD“,

Declaration:

```
FUNCTION FBE_KBD_NEWCODEMATRIX : BOOL
VAR_INPUT
        MATRIXADDRESS: UDINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
MATRIXADDRESS	UDINT	Address of the key-code-translation-table that should be used

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

16.4. FBE_KBD_READKEY

Description:
This function returns next key-code taken from fifo-buffer or zero if the buffer is empty.

Declaration:
```
FUNCTION FBE_KBD_READKEY : BYTE
VAR_INPUT
        Dummy: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BYTE	Value don't care (for future use)

Return Value (Data type BYTE)
Returns key-code if buffer is not empty. Else zero will be returned.

16.5. FBE_KBD_RESET

Description:
Clears the key-fifo-buffer.

Declaration:
```
FUNCTION FBE_KBD_RESET : BOOL
VAR_INPUT
        Dummy: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BYTE	Value don't care (for future use)

Return Value (Data type BOOL)
TRUE if function has finished successfully
FALSE if function failed

16.6. FBE_KBD_WAITKEY

Description:

This function returns the next key-code taken from fifo-buffer. If buffer is empty the function is still waiting until next key-hit enters a new key code. (Note: endless loop! use this function only for multi-tasking applications)

Declaration:

```
FUNCTION FBE_KBD_WAITKEY : BYTE
VAR_INPUT
        Dummy: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BYTE	Value don't care (for future use)

Return Value (Data type BYTE)

Returns the next key-code taken from fifo-buffer (Note: endless loop possible!)

16.7. FBE_KBD_WRITEBACKKEY

Description:

Stores a key-code into the fifo-buffer. This key-code gets the previous buffer position. So the next read returns this code. With this function the key-codes may be stored 'last in – first out'.

Declaration:

```
FUNCTION FBE_KBD_WRITEBACKKEY : BYTE
VAR_INPUT
        WRBACKVALUE         : BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
WRBACKVALUE	BYTE	Value to write back to the keyboard buffer

Return Value (Data type BYTE)

KeyCode of the key that was written back to the keyboard buffer

17. SMPOS / Stepper Motor

The Library FBE_SmPos.lib is a Library extension for the CoDeSys PLC runtime system to support absolute and relative positioning using stepper motors. The system outputs a clock and a direction control signal for the stepper motor. The library supports exponential ramps for acceleration.

The following functions are implemented:

Function name	Description
SmPos_ChangeRampPara	Change ramp parameters for acceleration and deceleration
SmPos_DefHome	Define actual position as home position (zero pos) The demand position keeps unchanged
SmPos_DisableDriver	Disables the stepper motor driver
SmPos_FixPosToHome	Define actual position as home position (zero pos) The demand position will be reset to 0
SmPos_GetActualPos	Read the actual position
SmPos_GetDemandPos	Read a previously set demand position
SmPos_GetDemandVelocity	Read a previously set velocity
SmPos_InitAxis	Initialize one axis
SmPos_InitDriver	Initialize the step motor driver library
SmPos_IsAxisActive	Check whether axis is active
SmPos_IsAxisMoving	Check whether axis is moving
SmPos_IsPositionReached	Check whether axis has reached the demand position
SmPos_SetActualPos	Set (modify) the actual position
SmPos_SetDemandPos	Set demand position
SmPos_SetOffset	Add a offset to the actual position and use this value as new demand position
SmPos_SetVelocity	Set new velocity
SmPos_StartMotion	Start a new motion
SmPos_StopMotion	Stop actual motion
SmPos_SyncAxis	Synchronize one axis to a master axis
SmPos_UnSyncAxis	Release synchronization of axis

17.1. Data Types

In order to implement the CANopen library functions there are several new data types declared in the CANopen library.

TSmPos_Axis

```
TYPE TSmPos_Axis : (
        SMPOS_AXIS0:= 0,
        SMPOS_AXIS1:= 1,
        SMPOS_AXIS2:= 2,
        SMPOS_AXIS3:= 3,
        SMPOS_AXIS4:= 4,
        SMPOS_AXIS5:= 5,
        SMPOS_AXIS6:= 6,
        SMPOS_AXIS7:= 7,
        SMPOS_NOAXIS:= -1
);
END_TYPE
```

Data type for Axis declaration

TSmPos_Direction

```
TYPE TSmPos_Direction : (
        STOP        := 0,
        FORWARD     := 1,
        REVERSE     := -1
);
END_TYPE
```

Data type for Direction of position movement

TSmPos_Position

```
TYPE TSmPos_Position : INT;
END_TYPE
```

Data type for describing absolute positions or offsets

TSmPos_RampMode

```
TYPE TSmPos_RampMode : (
          SMPOS_RAMP_EXP3:= 0
);
END_TYPE
```

Data type for defining the ramp shape for acceleration and deceleration.
SMPOS_RAMP_EXP3 Exponential ramp up to 3 TAU.

17.2. SMPOS_CHANGERAMPPARA

Description:

Change ramp parameters for one axis. The axis will take the parameter frequencies to calculate the ramp shape. The speed can be reduced using the SMPOS_SetVelocity function.

Declaration:

```
FUNCTION SmPos_ChangeRampPara : BOOL
VAR_INPUT
          Axis             : TSmPos_Axis;
          RampMode         : TSmPos_RampMode;
          RampLength       : UINT;
          FrequencyStart : UINT;
          FrequencyMax   : UINT;
          FrequencyStop  : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification
RampMode	TSmPos_RampMode	Shape of acceleration deceleration ramp
RampLength	UINT	Number of steps to use on one ramp It is recommended to use a minimum of 50 steps
FrequencyStart	UINT	Ramp start frequency for acceleration
FrequencyMax	UINT	Frequency at full speed
FrequencyStop	UINT	Frequency after slow down

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.3. SMPOS_DEFHOME

Description:

Defines tha actual position as home position. The movement will not be modified. The demand position will not be changed. Ths function may be used to set a new zero point while a motion is still in progress.

Declaration:

```
FUNCTION SmPos_DefHome : BOOL
VAR_INPUT
          Axis     : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.4. SMPOS_DISABLEDRIVER

Description:

Disables the step motor library

Declaration:

```
FUNCTION SmPos_DisableDriver : BOOL
VAR_INPUT
          Mode     : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Mode	UINT	

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.5. SMPOS_ENCASSIGN

Description:

Assigns an encoder channel to a step motor axis. This enables to driving of this axis under direct control of an encoder. The actual position for this axis will not be a result of counting the steps. In this case the encoder will be read to get the actual position.

Declaration:

```
FUNCTION SmPos_EncAssign : BOOL
VAR_INPUT
        Axis            : TSmPos_Axis;
        EncoderNr       : UINT;
        EncToStepShift  : UINT;
        Config          : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification
EncoderNr	UINT	Number of the encoder channel to assign with this axis
EncToStepShift	UINT	Nr of SHR to calculate Steps from EncoderPulses The resolution of the encoder may be a multiple of the stepper resolution, but the operation to rescale the encoder value to a step count value must be a “shift right” procedure.
Config	UINT	= 0 : Reserved for future use

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.6. SMPOS_ENCRELEASE

Description:

A previously assigned encoder channel for position control of this axis will be released. The stepper axis will the use pulse counting of the stepper steps to calculate the actual position.

Declaration:

```
FUNCTION SmPos_EncRelease : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.7. SMPOS_ENCSTOPUSE

Description:

The Axis will not release a previously assigned encoder for position control, but will do the rest of this motion without encoder control. It is recommended to use this function for the last few steps in order to avoid a jitter after motion if the axis has reached its demand position.

Declaration:

```
FUNCTION SmPos_EncStopUse : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.8. SMPOS_ENCUSE

Description:
Reenables the use of e previously assigned encoder for this motion.

Declaration:
```
FUNCTION SmPos_EncUse : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type BOOL)
TRUE if function has finished successfully
FALSE if function failed

17.9. SMPOS_FIXPOSTOHOME

Description:
Sets the actual position as home position (zero position) and also resets the demand position to this value. This function may be used to set the actual position to reference position while an axis is moving. In this case the axis will decelerate and will drive the axis to this position. This is the recommended way to realize a reference homing mode.

Declaration:
```
FUNCTION SmPos_FixPosToHome : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type BOOL)
TRUE if function has finished successfully
FALSE if function failed

17.10. SMPOS_GETACTUALPOS

Description:

Reads the actual position of the addressed axis

Declaration:

```
FUNCTION SmPos_GetActualPos : TSmPos_Position
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type TSmPos_Position)

Actual position

17.11. SMPOS_GETDEMANDPOS

Description:

Reads back a previously set demend position

Declaration:

```
FUNCTION SmPos_GetDemandPos : TSmPos_Position
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type TSmPos_Position)

Demand position

17.12. SMPOS_GETDEMANDVELOCITY

Description:
Read back a previously set demand velocity

Declaration:
```
FUNCTION SmPos_GetDemandVelocity : UINT
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type UINT)
Velocity. The velocity is a percentage of the maximum possible speed given by the ramp configuration values.

17.13. SMPOS_INITAXIS

Description:

Initialize one axis. The axis will take the parameter frequencies to calculate the ramp shape. The speed can be reduced using the SMPOS_SetVelocity function. This velocity will be set to the default value of 100 (percent)

Declaration:

```
FUNCTION SmPos_InitAxis : BOOL
VAR_INPUT
        Axis                    : TSmPos_Axis;
        RampMode                : TSmPos_RampMode;
        RampLength              : UINT;
        FrequencyStart          : UINT;
        FrequencyMax            : UINT;
        FrequencyStop           : UINT;
        SoftStopPulses          : UINT;
        Tolerance               : UINT;
        InvertOutputLevel       : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification
RampMode	TSmPos_RampMode	Shape of acceleration deceleration ramp
RampLength	UINT	Number of steps to use on one ramp It is recommended to use a minimum of 50 steps
FrequencyStart	UINT	Ramp start frequency for acceleration
FrequencyMax	UINT	Frequency at full speed
FrequencyStop	UINT	Frequency after slow down
SoftStopPulses	UINT	Number of pulses to drive the stepper after slowing down to the FrequencyStop in order to give the mechanical system better behaviour.
Tolerance	UINT	Position tolerance gives the number of pulses, a position might be different from the exact position.
InvertOutputLevel	UINT	Option to invert the direction output signal in order to adapt positive position values to a desired direction.

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.14. SMPOS_INITDRIVER

Description:

Initializes the library. This function must be executed once before any other library function may be used.

Declaration:

```
FUNCTION SmPos_InitDriver : BOOL
VAR_INPUT
        Mode    : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Mode	UINT	

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.15. SMPOS_ISAXISACTIVE

Description:

Checks whether the addressed axis is currently active. Note an axis might be active without performing any movement. This might occur if the axis is synchronized to another axis, and this master axis is stopped.
See also: SmPos_IsAxisMoving

Declaration:

```
FUNCTION SmPos_IsAxisActive : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type BOOL)

TRUE if the axis is active. FALSE otherwise.

17.16. SMPOS_ISAXISMOVING

Description:
Checks whether the axis is performing any movement.

Declaration:
```
FUNCTION SmPos_IsAxisMoving : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type BOOL)
TRUE if there is a positioning motion in progress.
FALSE if axis is halted

17.17. SMPOS_ISPOSITIONREACHED

Description:
Checks whether the addressed axis has reached its demand position.

Declaration:
```
FUNCTION SmPos_IsPositionReached : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type BOOL)
TRUE demand position is reached
FALSE demand position is not (yet) reached

17.18. SMPOS_SETACTUALPOS

Description:

Sets a new position as actual position for this axis.

Declaration:

```
FUNCTION SmPos_SetActualPos : BOOL
VAR_INPUT
        Axis            : TSmPos_Axis;
        NewActualPos    : TSmPos_Position;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification
NewActualPos	TSmPos_Position	New position that must be set as actual position

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.19. SMPOS_SETDEMANDPOS

Description :

Set a new demand position. The axis will not try to reach a previously given demand pos, but will try to reach the new demand position as soon as possible. Nevertheless if the position is in the opposite direction to the actual axis movement the axis will slow down using the ramp parameters, then the axis will change direction and will then drive the remaining distance as fast as possible using the ramp parameters.

Declaration:

```
FUNCTION SmPos_SetDemandPos : BOOL
VAR_INPUT
        Axis            : TSmPos_Axis;
        DemandPos       : TSmPos_Position;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification
DemandPos	TSmPos_Position	New position that must be set as demand position

Return Value (Data type BOOL)

TRUE if function has finished successfully FALSE if function failed

17.20. SMPOS_SETOFFSET

Description:

Adds an offset to the actual position and sets this position as new demand position

Declaration:

```
FUNCTION SmPos_SetOffset : BOOL
VAR_INPUT
        Axis     : TSmPos_Axis;
        Offset   : TSmPos_Position;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification
Offset	TSmPos_Position	Offset to add

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.21. SMPOS_SETVELOCITY

Description:

Sets a new velocity. This velocity is a percentage of the maximum possible speed values given by the frequencies that are set with the function SmPos_InitAxis() or SmPos_ChangeRampPara()

Declaration:

```
FUNCTION SmPos_SetVelocity : BOOL
VAR_INPUT
        Axis            : TSmPos_Axis;
        Velocity        : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification
Velocity	UINT	Percentage of the max. possible speed that must be used

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.22. SMPOS_STARTMOTION

Description:
Starts a motion with the previously set values and positions.

Declaration:
```
FUNCTION SmPos_StartMotion : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axis for identification

Return Value (Data type BOOL)
TRUE if function has finished successfully
FALSE if function failed

17.23. SMPOS_STOPMOTION

Description:
Stops a motion

Declaration:
```
FUNCTION SmPos_StopMotion : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type BOOL)
TRUE if function has finished successfully
FALSE if function failed

17.24. SMPOS_SYNCAXIS

Description:

Synchronize an axis to another one. The axis will then perform exactly the same movement as the master axis. The synchronization procedure will be done using the ramp steps as a reference. Therefore axis that must be synchronized have the same ramp parameters. Different speeds can only achieved using different velocities for this axes.

Declaration:

```
FUNCTION SmPos_SyncAxis : BOOL
VAR_INPUT
        Axis            : TSmPos_Axis;
        MasterAxis      : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Axis that must be synchronized
MasterAxis	TSmPos_Axis	Master Axis

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

17.25. SMPOS_UNSYNCAXIS

Description:

The synchronization of the addressed axis to another one will be released. The axis will continue an active motion with its own position values.

Declaration:

```
FUNCTION SmPos_UnSyncAxis : BOOL
VAR_INPUT
        Axis    : TSmPos_Axis;
END_VAR
```

Parameters:

Name	Data-Type	Description
Axis	TSmPos_Axis	Number of axe for identification

Return Value (Data type BOOL)

TRUE if function has finished successfully
FALSE if function failed

18. PULSE

The Library FBE_Pulse.lib is an IEC-Code external library and may be used for generating an Oscillator with constant frequency or a Pulse Width Modulation (PWM). (Use FBE_Pulse215.lib for EASY215 and EASY217 module / see below)

18.1. Hardware Cross Reference

Not all functions work with each EASY Module because they have different I/O's for application. This reference shows the connections, which can be used on an EASY.

Typ / Pulse Channel	EASY 215/217	EASY 235/237/238	EASY 2504	
0	**IO 4**	**OUT1.0 / PWM0**	**Output Byte 1.0**	
1	**IO 5**	**OUT1.1 / PWM1**	**Output Byte 1.1**	
2	**-**	**OUT1.2 / PWM2**	**Output Byte 1.2**	
3	**-**	**OUT1.3 / PWM3**	**Output Byte 1.3**	
Range	**Max. CPU-Clock / 8**	**Max. CPU-Clock / 2**	**Max. 100kHz**	

Note:
If one of the four possible channels (depending on used hardware) is initialized as an Oscillator or a PWM-Channel, this channel can't be used as digital output.

If a output is used as a pulse-channel and will be direct written to "0" or "1", the pulse signal will be inverted.

The following functions are implemented:

Function name	Description
FBE_PULSE_InitOsc	Inizialize the request Oscillator channel
FBE_PULSE_InitPwm	Inizialize the request Pulse Width Modulation channel
FBE_PULSE_SetFHzOsc	Set the frequency of the Oscillator channel
FBE_PULSE_SetPwmDiv	Changes the clock pre-divider
FBE_PULSE_SetPwmSteps	Set the Steps of the Pulse Width Modulation channel
FBE_PULSE_Stop	Stops the request channel
FBE_PULSE_UninstallOsc	Uninstall the request Oscillator channel
FBE_PULSE_UninstallPwm	Uninstall the request Pulse Width Modulation channel

Attention:
The FBE_PULSE Library for EASY215 and EASY217 is called FBE_PULSE215.LIB!
Please check page 90!

18.2. FBE_PULSE_InitOsc

Description:

Inizialize the request Oscillator channel.

Declaration:

```
FUNCTION FBE_PULSE_InitOsc : BOOL
VAR_INPUT
            PLSCpuClockMHz      : UINT;
            PLSChannel          : UINT;
            PLSPara             : UINT;
            PLSDiv2             : BOOL;
            PLSDiv64            : BOOL;
 END_VAR
```

Parameters:

Name	Data-Type	Description
PLSCpuClockMHz	UINT	CPU Clock in MHz of the System (default 20 MHz)
PLSChannel	UINT	Number of channel for Oscillator-Signal-Output (Range 0..3)
PLSPara	UINT	Pulse length (y=PLSPara/PLSCpuClockMHz)
PLSDiv2	BOOL	To select Edge Alligned (FALSE) or Center Alligned (TRUE) PULSE
PLSDiv64	BOOL	To select Oscclock resulution=CPUClock (FALSE) or OscillatorClock resulution=CPUClock /64 (TRUE)

Return Value (Data type BOOL)

TRUE	If initialization of the oscillator channel was successful
FALSE	If initialization of the oscillator channel failed

18.3. FBE_PULSE_InitPwm

Description:

Inizialize the request Pulse Width Modulation channel.

Declaration:

```
FUNCTION FBE_PULSE_InitPwm : BOOL
VAR_INPUT
        PLSCpuClockMHz      : UINT;
        PLSChannel          : UINT;
        PLSSteps            : UINT;
        PLSDiv2             : BOOL;
        PLSDiv64            : BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
PLSCpuClockMHz	UINT	CPU Clock in MHz of the System (default 20 MHz)
PLSChannel	UINT	Number of channel for PWM-Signal-Output (Range 0..3)
PLSSteps	UINT	Maximum of steps for resolution Note: This Parameter is on EASY215 and EASY217 for both channels equal. The last initialized channel set se resolution.
PLSDiv2	BOOL	To select Edge Alligned (FALSE) or Center Alligned (TRUE) PULSE Note: This parameter is not supported by EASY215, EASY217 TRUE or FALSE has no function
PLSDiv64	BOOL	To select Oscclock resulution=CPUClock (FALSE) or OscillatorClock resulution=CPUClock /64 (TRUE) Note This parameter is not supported by EASY215, EASY217 TRUE or FALSE has no function

Return Value (Data type BOOL)

TRUE	If initialization of the PWM channel was successful
FALSE	If initialization of the PWM channel failed

18.4. FBE_PULSE_SetFHzOsc

Description:

Set the frequency of the Oscillator channel. (value → Hz)

Declaration:

```
FUNCTION FBE_PULSE_SetFHzOsc : BOOL
VAR_INPUT
            PLSChannel              : UINT;
            PLSFRQ                  : UINT;
 END_VAR
```

Parameters:

Name	Data-Type	Description
PLSChannel	UINT	Number of channel for Oscillator-Signal-Output (Range 0..3)
PLSFRQ	UINT	Frequency of the Oscillator-Signal at Hz

Return Value (Data type BOOL)

TRUE if frequency is set, else FALSE.

18.5. FBE_PULSE_SetPwmSteps

Description:

Set the Steps of the Pulse Width Modulation channel.

Declaration:

```
FUNCTION FBE_PULSE_SetPwmSteps : BOOL
VAR_INPUT
            PLSChannel              : UINT;
            PLSValue                : WORD ;
END_VAR
```

Parameters:

Name	Data-Type	Description
PLSChannel	UINT	Number of channel for PWM-Signal-Output (Range 0..3)
PLSValue	WORD	Number of steps to go (PLSValue <PLSSteps)

Return Value (Data type BOOL)

TRUE if steps are set, else FALSE.

18.6. FBE_PULSE_SetPwmDiv

Description:

Set the Steps of the Pulse Width Modulation channel. (THIS FUNCTION WORKS ONLY WITH EASY215 and EASY217 / see chapter FBE_Pulse215.lib).

Declaration:

```
FUNCTION FBE_PULSE_SetPwmDiv : BOOL
VAR_INPUT
        ClkDivider      :WORD;
END_VAR
```

Parameters:

Name	Data-Type	Description
WORD	UINT	Clock divider identity: 0 = 1/8, 1 = 1/16, 2 = 1/32, 3 = 1/64, 4 = 1/128, 5 = 1/256, 6 = 1/512, 7 = 1/1024

Return Value (Data type BOOL)

TRUE if divider was changed, else FALSE.

18.7. FBE_PULSE_Stop

Description:

Stops the signal of the request channel.

Declaration:

```
FUNCTION FBE_PULSE_Stop : BOOL
VAR_INPUT
        PLSChannel          : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
PLSChannel	UINT	Number of channel which has to stop Signal-Output (Range 0..3)

Return Value (Data type BOOL)

TRUE if stop is set, else FALSE.

18.8. FBE_PULSE_UninstallOsc

Description:
Uninstalls the request Oscillator-Channel to use it as a digital Port.

Declaration:
```
FUNCTION FBE_PULSE_UninstallOsc : BOOL
VAR_INPUT
        PLSChannel          : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
PLSChannel	UINT	Number of Oscillator-Channel which shoud be uninstalled (Range 0..3)

Return Value (Data type BOOL)
TRUE if channel is successfully uninstalled, else FALSE.

18.9. FBE_PULSE_UninstallPwm

Description:
Uninstalls the request PWM-Channel to use it as a digital Port.

Declaration:
```
FUNCTION FBE_PULSE_UninstallPwm : BOOL
VAR_INPUT
        PLSChannel          : UINT;
END_VAR
```

Parameters:

Name	Data-Type	Description
PLSChannel	UINT	Number of PWM-Channel which shoud be uninstalled (Range 0..3)

Return Value (Data type BOOL)
TRUE if channel is successfully uninstalled, else FALSE.

19. PULSE215

The Library FBE_Pulse215.lib is an IEC-Code library and may be used for generating a Pulse Width Modulation (PWM) with the EASY215 / EASY217 only. This library has any identical functions as the library FBE_Pulse.lib (see previous chapter)

19.1. Hardware Cross Reference

This reference shows the connections, which can be used on an EASY215 / EASY217.

Typ / Pulse Channel	EASY 215 EASY 217			
0	**IO4**			
1	**IO5**			
Range	**Max. CPU-Clock / 8**			

The following functions are implemented:

Function name	Description
FBE_PULSE_InitPwm	Initialize the request Pulse Width Modulation channel
FBE_PULSE_SetPwmDiv	Changes the clock pre-divider
FBE_PULSE_SetPwmSteps	Set the Steps of the Pulse Width Modulation channel
FBE_PULSE_Stop	Stops the request channel
FBE_PULSE_UninstallPwm	Uninstall the request Pulse Width Modulation channel

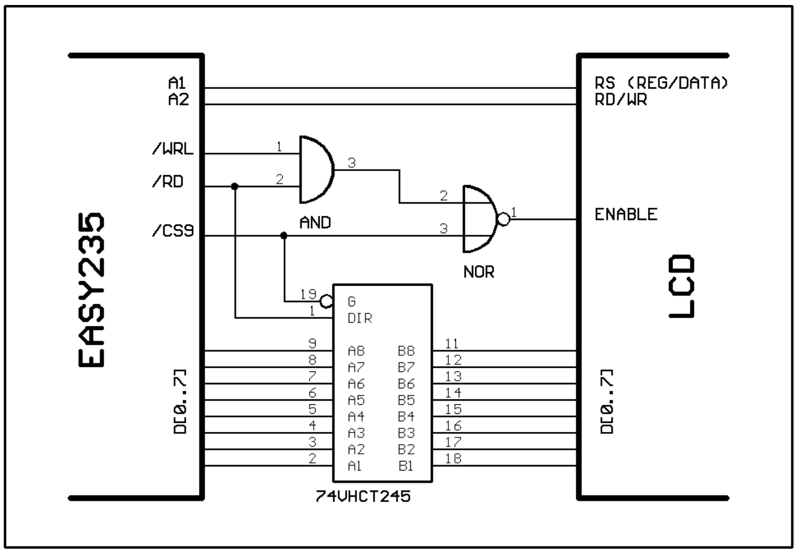
All functions in this library are identical functions as in the library FBE_Pulse.lib. See previous chapter (FBE_Pulse.lib) for description of these functions.

20. LCD

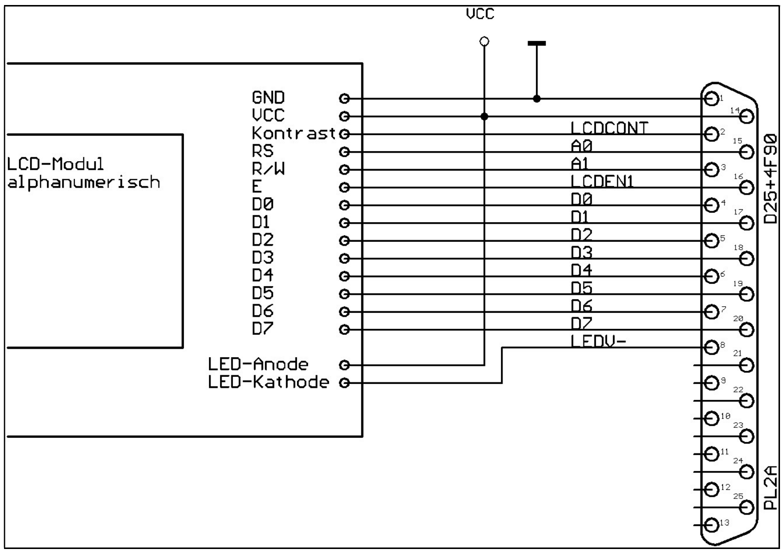
The Library FBE_LCD.lib is an IEC-Code external and may be used for controlling alphanumeric character displays. The maximum supported display size is 20 characters x 4 lines.

20.1. Schematics for display connection

EASY235:



EASY2504:



The following functions are implemented:

Function name	Description
FBE_LCD_CURSORBLINK	Activates blink mode of the cursor (Wait/Break)
FBE_LCD_ CURSORMOVETO	Moves the cursor to a new position (Wait/Break)
FBE_LCD_ CURSOROFF	Sets the cursor invisibly (Wait/Break)
FBE_LCD_ CURSORON	Sets the cursor visibly (Wait/Break)
FBE_LCD_ DELAY	Runs a time delay
FBE_LCD_ INIT	Initialize the LCD library parameters and display interface
FBE_LCD_ WAITUNTILREADY	Waits until display is not busy
FBE_LCD_ WHEREX	Returns the X (row) value of the actual cursor position
FBE_LCD_ WHEREY	Returns the Y (line) value of the actual cursor position
FBE_LCD_ WRCHAR	Writes a character at the actual cursor position on display
FBE_LCD_ WRSTRING	Writes a zero terminated String at the actual cursor position on display

Note:
Most of these functions are waiting until the display is ready for next command. For the reason, that the display will not go ready, a time overrun breaks those functions after 5000 CPU-Cycles. All functions using this mechanism are signed with (Wait/Break) in the upper table.
If waiting functions are not wanted in the program, the function FBE_LCD_READY can be used to check the actual state before calling a (Wait/Break) function. If FBE_LCD_READY returns true no wait is done by the following (Wait/Break) function.

20.2. Data Types

In order to implement the LCD library functions there was a new data types declared in the LCD library.

TSmPos_Axis

```
TYPE type_FBE_LCD_BASE :
        STRUCT
                XColumns        : BYTE;
                YLines          : BYTE;
                LineBase        : ARRAY[0..3] OF BYTE;
                pWrReg          : POINTER TO BYTE;
                pRdReg          : POINTER TO BYTE;
                pWrData         : POINTER TO BYTE;
                pRdData         : POINTER TO BYTE;
        END_STRUCT
END_TYPE
```

20.3. FBE_LCD_CURSORBLINK

Description:

Activates blink mode of the cursor.

Declaration:

```
FUNCTION FBE_LCD_CURSORBLINK : BOOL
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don’t care (for future use)

Return Value (Data type BOOL)

TRUE   If access was successfully.
FALSE If LCD busy was continuously set until the access-time-overrun occurs.

20.4. FBE_LCD_CURSORMOVETO

Description:

Moves the cursor to the Xpos (row) / Ypos (line) position. Valid values for row are (1..20) and (1..4) for line.

Declaration:

```
FUNCTION FBE_LCD_CURSORMOVETO : BOOL
VAR_INPUT
        XPos    :BYTE;
        YPos    :BYTE;
 END_VAR
```

Parameters:

Name	Data-Type	Description
XPos	BYTE	Row number of cursor destination. Valid range 1 .. 20
YPos	BYTE	Line number of cursor destination. Valid range 1 .. 4

Return Value (Data type BOOL)

TRUE   If access was successfully.
FALSE If LCD busy was continuously set until the access-time-overrun occurs or the X/YPos parameters are out of range.

20.5. FBE_LCD_ CURSOROFF

Description:

Sets the cursor invisibly.

Declaration:

```
FUNCTION FBE_LCD_CURSOROFF : BOOL
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don't care (for future use)

Return Value (Data type BOOL)

TRUE If access was successfully.
FALSE If LCD busy was continuously set until the access-time-overrun occurs.

20.6. FBE_LCD_ CURSOROFF

Description:

Sets the cursor invisibly.

Declaration:

```
FUNCTION FBE_LCD_CURSOROFF : BOOL
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don't care (for future use)

Return Value (Data type BOOL)

TRUE If access was successfully.
FALSE If LCD busy was continuously set until the access-time-overrun occurs.

20.7. FBE_LCD_ CURSORON

Description:
Sets the cursor visibly.

Declaration:

```
FUNCTION FBE_LCD _CURSORON: BOOL
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don’t care (for future use)

Return Value (Data type BOOL)

TRUE	If access was successfully.
FALSE	If LCD busy was continuously set until the access-time-overrun occurs.

20.8. FBE_LCD_ DELAY

Description:
Runs a time delay.

Declaration:

```
FUNCTION FBE_LCD_DELAY : BOOL
VAR_INPUT
        WaitTime: TIME;
END_VAR
```

Parameters:

Name	Data-Type	Description
WaitTime	Time	Wait time

Return Value (Data type BOOL)

TRUE	If access was successfully.
FALSE	If LCD busy was continuously set until the access-time-overrun occurs.

20.9. FBE_LCD_ INIT

Description:

Initiates the LCD library parameters and display interface. The maximum supported display size is 20x4 (rows x lines).
This function must be called once before using other LCD-functions.
Note: This function needs more then 40ms because there are any waits in the display initialization sequence. So the best way to use this function is to call it once before the PLC program starts.

Declaration:

```
FUNCTION FBE_LCD_INIT : BOOL
VAR_INPUT

        XColumns        : BYTE;
        YLines          : BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
XColumns	BYTE	Size X of the display / valid values [1 .. 20]
YLines	BYTE	Size X of the display / valid values [1 .. 4]

Return Value (Data type BOOL)

TRUE	If access was successfully.
FALSE	If error occurs and initiation was not done.

20.10. FBE_LCD_ WAITUNTILREADY

Description:

This function is waiting until the display is ready for next command. For the reason, that the display will not go ready, a time overrun breaks that function after 5000 CPU Cycles.

Declaration:

```
FUNCTION FBE_LCD_WAITUNTILREADY : BOOL
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don't care (for future use)

Return Value (Data type BOOL)

TRUE	If display is ready for next data or command.
FALSE	If LCD busy was continuously set until the access-time-overrun occurs.

20.11. FBE_LCD_ WHEREX

Description:

Returns the X (row) value of the actual cursor position.

Declaration:

```
FUNCTION FBE_LCD_WHEREX : BYTE
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don't care (for future use)

Return Value (Data type BYTE)

X (row) value of actual cursor position.

20.12. FBE_LCD_ WHEREY

Description:

Returns the Y (line) value of the actual cursor position.

Declaration:

```
FUNCTION FBE_LCD_WHEREY : BYTE
VAR_INPUT
        Dummy: BOOL;
END_VAR
```

Parameters:

Name	Data-Type	Description
Dummy	BOOL	Value don't care (for future use)

Return Value (Data type BYTE)

Y (line) value of actual cursor position.

20.13. FBE_LCD_ WRCHAR

Description:

Writes a character at the actual cursor position on display.

Declaration:

```
FUNCTION FBE_LCD_WRCHAR : BOOL
VAR_INPUT
        Character: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Character	BYTE	Character to write

Return Value (Data type BOOL)

TRUE If access was successfully.
FALSE If LCD busy was continuously set until the access-time-overrun occurs.

20.14. FBE_LCD_ WRSTRING

Description:

Writes a zero terminated String at the actual cursor position on display.

Declaration:

```
FUNCTION FBE_LCD_WRSTRING : BOOL
VAR_INPUT
        pStr    : POINTER TO BYTE;          (* Pointer to Zero terminated string *)
END_VAR
```

Parameters:

Name	Data-Type	Description
pStr	POINTER TO BYTE	Pointer to first byte of a zero terminated string

Return Value (Data type BOOL)

TRUE If access was successfully.
FALSE If LCD busy was continuously set until the access-time-overrun occurs.

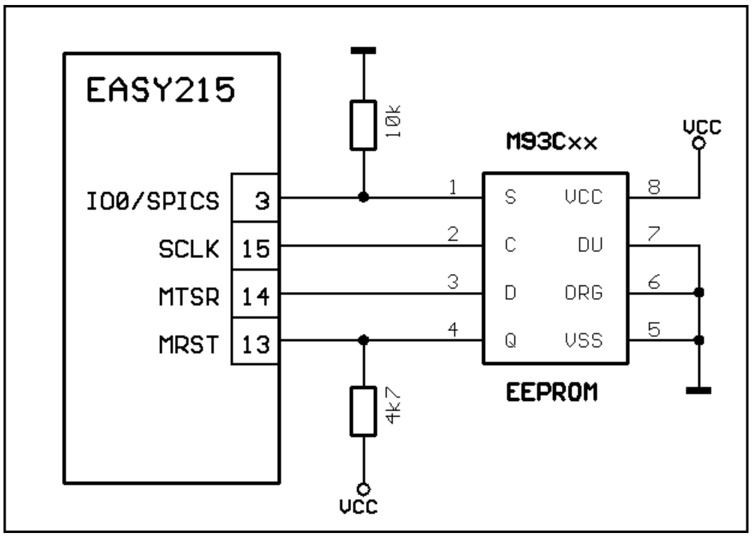
21. SPI-EEPROM

The Library FBE_ SpiEeprom.lib is a Library extension for the CoDeSys PLC runtime system to write and read data to and from an external SPI-Bus EEPROM. The following types are supported for using in BYTE (x8) configuration:

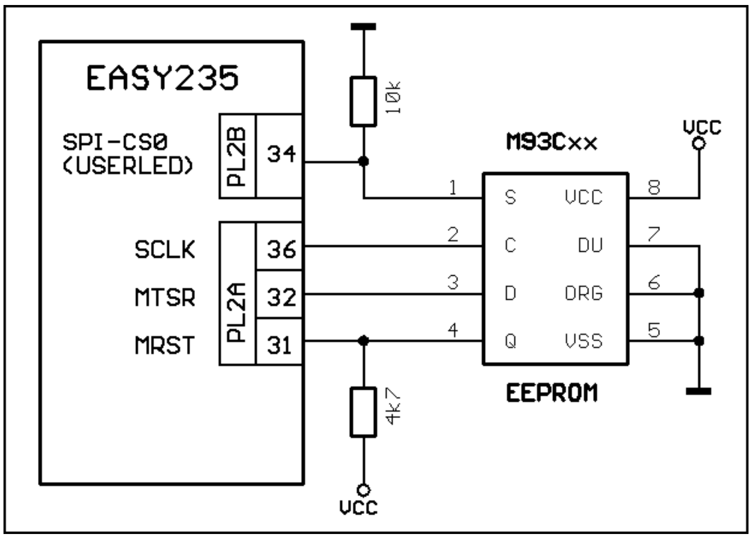
Serial Microwire Bus EEPROM M93C46, M93C56, M93C66, M93C76, M93C86 and compatible.

21.1. Schematics for SPI-EEPROM connection

EASY215 / EASY217:



EASY235:



21.2. Hardware Cross Reference

This reference shows the connections, which can be used on an EASY.

Typ / Pulse Channel	EASY215 EASY217	EASY235		
0	**IO0 SPI-CS0**	**(USERLED) SPI-CS0**		

The following functions are implemented:

Function name	Description
FBE_SPI_EEP_ERASE	Erases the whole memory. (All memory bits are set to 1)
FBE_SPI_EEP_INIT	Initialize the SPI library parameters and SSC interface
FBE_SPI_EEP_READ	Reads data from SPI EEPROM
FBE_SPI_EEP_WRITE	Writes data to SPI EEPROM

Note: The function Spi-Eep-Init must running at least once before the others.
EASY215/EASY217: The function Spi-Eep-Erase is not available.
EASY215/EASY217: The output of the status LED is also used for SPI-Clock. So if the FBE_SpiEeprom library is used, the status LED is flickering undefined
while a SPI transfer runs.

EASY235: The output of the USERLED is also used for SPI-CS0. So if the FBE_SpiEeprom library is used, the USERLED is flickering undefined while a SPI transfer runs.

21.3. FBE_SPI_EEP_ERASE

Description:

Erases the whole memory. (All memory bits are set to 1)

Declaration:

```
FUNCTION_BLOCK FBE_SPI_EEP_ERASE
VAR_INPUT
        Start:          BOOL;
END_VAR
VAR_OUTPUT
        Ready:          BOOL;
        Error:          BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Start	BOOL	Starts function block with rising edge. (When function is active, a low level breaks running)
Ready	BOOL	Returns TRUE when Erase function is ready
Error	BYTE	Error status: 0: No error, 1: reserved 2: No Device (→ run FBE_SPI_EEP_INIT once before)

21.4. FBE_SPI_EEP_INIT

Description:

Initialize the SPI library parameters and 'Synchronous-Serial-Communication' interface. The function must run at least once before the others

Declaration:

```
FUNCTION_BLOCK FBE_SPI_EEP_INIT
VAR_INPUT
        Channel:        BYTE;
        Baudrate:       UDINT;
        Start:          BOOL;
END_VAR
VAR_OUTPUT
        Ready:          BOOL;
        Error:          BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Start	BOOL	Starts function block with rising edge. (When function is active, a low level breaks running)
Channel	BYTE	Selects the Channel witch is used for the external SPI device. 0: SPI-EEPROM, 1..255 reserved for future use
Baudrate	UDINT	Selects the Baudrate in bits per second (SPI-Clock) Possible Values: 1000 up to 500000 (1k..500k Baud)
Ready	BOOL	Returns TRUE when function is ready
Error	BYTE	Error status: 0: No error 1: Device not found >1: reserved

21.5. FBE_SPI_EEP_READ

Description:

Copies a data block with given number of bytes from the SPI-EEPROM into a selectable variable or memory area.

Declaration:

```
FUNCTION_BLOCK FBE_SPI_EEP_READ
VAR_INPUT
        AdrSource:      UINT;
        AdrDestination: UDINT;
        Size:           UINT;
        Start:          BOOL;
END_VAR
VAR_OUTPUT
        Ready:          BOOL;
        Error:          BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Start	BOOL	Starts function block with rising edge. (When function is active, a low level breaks running)
AdrSource	UINT	Address of the first byte in the EEPROM.
AdrDestination	UDINT	Address of the first byte of the destination variable or memory.
Size	UINT	Number of bytes to copy from source to destination
Ready	BOOL	Returns TRUE function is ready
Error	BYTE	Error status: 0: No error 1: reserved 2: No Device (→ run FBE_SPI_EEP_INIT once before)

Note: Make sure that the size of destination value (memory area) is large enough. If not, the function writes data's over the end and other variables will be overwritten.

21.6. FBE_SPI_EEP_WRITE

Description:

Copies a data block with given number of bytes from a selectable variable or memory area into the SPI-EEPROM.

Declaration:

```
FUNCTION_BLOCK FBE_SPI_EEP_WRITE
VAR_INPUT
        AdrDestination: UINT;
        AdrSource:      UDINT;
        Size:           UINT;
        Start:          BOOL;
END_VAR
VAR_OUTPUT
        Ready: BOOL;
        Error: BYTE;
END_VAR
```

Parameters:

Name	Data-Type	Description
Start	BOOL	Starts function block with rising edge. (When function is active, a low level breaks running)
AdrDestination	UDINT	Address of the first byte of the source variable or memory.
AdrSource	UINT	Address for the first byte in the EEPROM.
Size	UINT	Number of bytes to copy from source to destination
Ready	BOOL	Returns TRUE function is ready
Error	BYTE	Error status: 0: No error 1: reserved 2: No device (→ run FBE_SPI_EEP_INIT once before)

Note: Storing a byte into a serial EEPROM needs time. This time depends on the used EEPROM chip. (M93Cxx = 10ms / byte max.)