



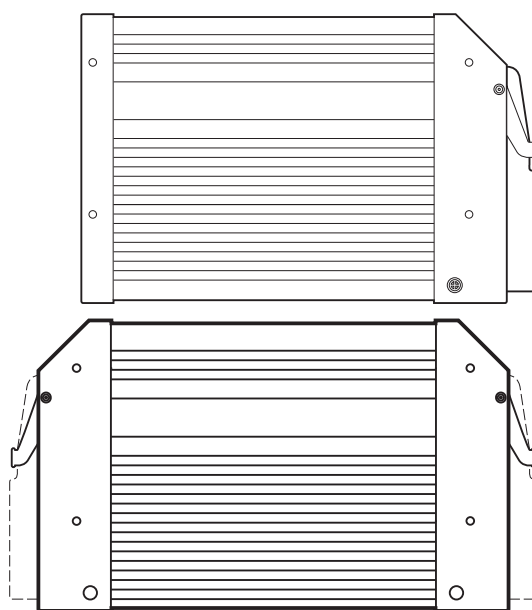
System Manual
SafetyController
ExtendedSafetyController

ecomat100®
CR7021
CR7201
CR7506

for ISO 13849 up to PL d
for IEC 62061 up to SIL CL 2

CoDeSys® V2.3
Target V06

English



Contents

1	About this manual	7
1.1	What do the symbols and formats mean?	7
1.2	How is this manual structured?	8
1.3	Changes of the manual (S16)	8
2	Safety instructions	9
2.1	Important!	9
2.2	What previous knowledge is required?	10
3	Notes on safety-related applications	11
3.1	General information	11
3.1.1	What does machine safety mean?	11
3.1.2	Risk assessment of a machine	11
3.1.3	Archiving of documentation	12
3.2	Standard ISO 13849	13
3.2.1	Risk assessment	13
3.2.2	Performance level PL	14
3.2.3	Categories to ISO 13849	16
3.2.4	Risk graph to ISO 13849	18
3.2.5	Technology of the safety-related control functions for PL or SIL	19
3.2.6	Safety for bus systems	20
3.3	Safety-related programming with CoDeSys to ISO 13849	21
3.3.1	Safety-related applications software (SRASW)	22
3.3.2	Rules on the specification of the SRASW	23
3.3.3	Rules for selecting the tools, libraries, languages	23
3.3.4	Rules on the program structure	24
3.3.5	Rules on SRASW and non safety-related software in one component	25
3.3.6	Rules on the software implementation / coding	25
3.3.7	Rules on the safety-related function blocks	25
3.3.8	Rules on the use of variables	26
3.3.9	Rules on the use of data types	27
3.3.10	Rules for testing safety-related software	27
3.3.11	Rules for documenting safety-related software	28
3.3.12	Rules on the verification of safety-related software	28
3.3.13	Rules for subsequent program modifications	28
3.4	SafetyController	29
3.4.1	ExtendedSafetyController CR7200/CR7201	29
3.4.2	Safe state	29
3.4.3	Safety-related inputs and outputs	29
3.4.4	Fail-safe sensors and safety signal transmitters	30
3.4.5	Test input	30
3.4.6	Use in applications up to CAT 3 / PL d	31
3.5	Safety functions	33
3.5.1	SAFE_ANALOG_OK (FB)	34
3.5.2	SAFE_FREQUENCY_OK (FB)	36
3.5.3	SAFE_INPUTS_OK (FB)	38
3.5.4	SAFETY_SWITCH (FB)	41

Contents

4	System description	44
4.1	Information concerning the device	44
4.1.1	Test basis for certification	44
4.1.2	Functions and features	44
4.2	Information concerning the software	46
4.3	PLC configuration	47
4.4	Monitoring concept	48
4.4.1	Hardware structure	48
4.4.2	Operating principle of the delayed switch-off	49
4.4.3	Operating principle of the monitoring concept	50
4.4.4	Feedback in case of externally supplied outputs	54
4.4.5	Safety concept	55
5	Operating states and operating system	57
5.1	Operating states	57
5.1.1	INIT state (Reset)	57
5.1.2	STOP state	57
5.1.3	Fatal error	57
5.1.4	RUN state	58
5.1.5	No operating system	58
5.2	Status LED	58
5.3	Load the operating system	59
5.4	Operating modes	60
5.4.1	TEST mode	60
5.4.2	SERIAL_MODE	60
5.4.3	DEBUG mode	60
6	Configurations	61
6.1	Set up programming system	61
6.1.1	Set up programming system manually	61
6.1.2	Set up programming system via templates	65
6.1.3	ifm demo programs	75
6.2	Function configuration of the inputs and outputs	78
6.2.1	Configure inputs	79
6.2.2	Configure outputs	92
6.3	Hints to wiring diagrams	101
6.4	Operating modes of the ExtendedSafetyController	103
6.4.1	Operating mode master/master	104
6.4.2	Operating mode master/slave	105
7	Limitations and programming notes	107
7.1	Limits of the device	107
7.1.1	CPU frequency	107
7.1.2	Above-average stress	108
7.1.3	Limits of the SafetyController	108
7.1.4	Watchdog behaviour	109
7.1.5	Available memory (CR7nnn)	109
7.2	Programming notes for CoDeSys projects	110
7.2.1	FB, FUN, PRG in CoDeSys	110
7.2.2	Note the cycle time!	110
7.2.3	Creating application program	111
7.2.4	Save	112

Contents

7.2.5	Using ifm downloader	112
7.2.6	Certification and distribution of the safety-related software	112
7.2.7	Changing the safety-relevant software after certification	113
8	Error messages	114
8.1	Slight errors	114
8.2	Serious errors	115
8.3	CAN error	116
8.4	Fatal errors	116
8.5	Response to the system error	117
8.5.1	Notes on devices with monitoring relay	117
8.5.2	Example process for response to a system error	118
9	Using CAN	119
9.1	General about CAN	119
9.1.1	Topology	119
9.1.2	CAN interfaces	120
9.1.3	System configuration	120
9.2	Physical connection of CAN	121
9.2.1	Network structure	121
9.2.2	CAN bus level	122
9.2.3	CAN bus level according to ISO 11992-1	122
9.2.4	Bus cable length	123
9.2.5	Wire cross-sections	124
9.3	Exchange of CAN data	125
9.3.1	Hints	126
9.3.2	Data reception	128
9.3.3	Data transmission	128
9.4	Description of the CAN standard program units	129
9.4.1	CAN1_BAUDRATE (FB)	130
9.4.2	CAN1_DOWNLOADID (FB)	132
9.4.3	CAN1_EXT (FB)	134
9.4.4	CAN1_EXT_TRANSMIT (FB)	136
9.4.5	CAN1_EXT_RECEIVE (FB)	138
9.4.6	CAN1_EXT_ERRORHANDLER (FB)	140
9.4.7	CAN2 (FB)	141
9.4.8	CANx_TRANSMIT (FB)	143
9.4.9	CANx_RECEIVE (FB)	145
9.4.10	CANx_RECEIVE_RANGE (FB)	147
9.4.11	CANx_EXT_RECEIVE_ALL (FB)	150
9.4.12	CANx_ERRORHANDLER (FB)	152
9.5	CAN units acc. to SAE J1939	154
9.5.1	CAN for the drive engineering	154
9.5.2	Units for SAE J1939	158
9.6	ifm CANopen library	170
9.6.1	Technical about CANopen	171
9.6.2	Start-up of the network without [Automatic startup]	186
9.6.3	Units for CANopen	204
9.7	CANopen Safety in safety-related applications	231
9.7.1	General notes and explanations on CANopen Safety	231
9.7.2	CANopen for safety-related communication	232
9.7.3	Functions for CANopen Safety	236
9.8	CAN errors and error handling	242
9.8.1	CAN errors	242
9.8.2	Structure of an EMCY message	245
9.8.3	Overview CANopen error codes	247

Contents

10	In-/output functions	250
10.1	Processing analogue input values	250
10.1.1	INPUT_ANALOG (FB)	251
10.1.2	INPUT_VOLTAGE (FB)	253
10.1.3	INPUT_CURRENT (FB)	254
10.2	Adapting analogue values	255
10.2.1	NORM (FB)	256
10.3	Counter functions for frequency and period measurement	258
10.3.1	Applications	258
10.3.2	Use as digital inputs	258
10.4	PWM functions	272
10.4.1	Availability of PWM	272
10.4.2	PWM signal processing	273
10.4.3	Current control with PWM	285
10.4.4	Hydraulic control in PWMi	292
10.5	Controller functions	313
10.5.1	General	313
10.5.2	Setting rule for a controller	315
10.5.3	Functions for controllers	316
11	Communication via interfaces	327
11.1	Use of the serial interface	327
11.1.1	SERIAL_SETUP (FB)	328
11.1.2	SERIAL_TX (FB)	330
11.1.3	SERIAL_RX (FB)	331
11.1.4	SERIAL_PENDING (FB)	333
11.2	Communication via the internal SSC interface	334
11.2.1	SSC_RECEIVE (FB)	335
11.2.2	SSC_TRANSMIT (FB)	337
12	Managing the data	338
12.1	Software reset	338
12.1.1	SOFTRESET (FB)	339
12.2	Reading / writing the system time	340
12.2.1	TIMER_READ (FB)	341
12.2.2	TIMER_READ_US (FB)	342
12.3	Saving, reading and converting data in the memory	343
12.3.1	Automatic data backup	343
12.3.2	Manual data storage	344
12.4	Data access and data check	351
12.4.1	SET_DEBUG (FB)	352
12.4.2	SET_IDENTITY (FB)	353
12.4.3	GET_IDENTITY (FB)	355
12.4.4	SET_PASSWORD (FB)	357
12.4.5	CHECK_DATA (FB)	359
13	Optimising the PLC cycle	361
13.1	Processing interrupts	361
13.1.1	SET_INTERRUPT_XMS (FB)	362
13.1.2	SET_INTERRUPT_I (FB)	365

Contents

14	Annex	368
14.1	Address assignment and I/O operating modes	368
14.1.1	Addresses / I/O variables	368
14.1.2	Possible operating modes inputs / outputs.....	370
14.1.3	Address assignment inputs / outputs	372
14.2	System flags	374
14.3	Overview of the files and libraries used.....	376
14.3.1	General overview.....	376
14.3.2	What are the individual files and libraries used for?.....	378
14.4	Troubleshooting	382
15	Glossary of Terms	383
16	Index	398

1 About this manual

Contents

What do the symbols and formats mean?	7
How is this manual structured?	8
Changes of the manual (S16)	8

202

In the additional "Programming Manual for CoDeSys V2.3" you will obtain more details about the use of the programming system "CoDeSys for Automation Alliance". This manual can be downloaded free of charge from **ifm's** website:

- a) → www.ifm.com > select your country > [Service] > [Download] > [Control systems]
- b) → **ifm**-CD "Software, tools and documentation"

Nobody is perfect. Send us your suggestions for improvements to this manual and you will receive a little gift from us to thank you.

© All rights reserved by **ifm electronic gmbh**. No part of this manual may be reproduced and used without the consent of **ifm electronic gmbh**.

All product names, pictures, companies or other brands used on our pages are the property of the respective rights owners.

1.1 What do the symbols and formats mean?

203

The following symbols or pictograms depict different kinds of remarks in our manuals:

 DANGER

Death or serious irreversible injuries are to be expected.

 WARNING

Death or serious irreversible injuries are possible.

 CAUTION


Slight reversible injuries are possible.

NOTICE

Property damage is to be expected or possible.

 NOTE

Important notes on faults and errors.

 Info

Further hints.

► ...	Required action
> ...	Response, effect
→ ...	"see"
abc	Cross references (links)
[...]	Designations of keys, buttons or display

1.2 How is this manual structured?

204

This documentation is a combination of different types of manuals. It is for beginners and also a reference for advanced users.

How to use this documentation:

- Refer to the table of contents to select a specific subject.
- The print version of the manual contains a search index in the annex.
- At the beginning of a chapter we will give you a brief overview of its contents.
- Abbreviations and technical terms are listed in the glossary.

In case of malfunctions or uncertainties please contact the manufacturer at:

→ www.ifm.com > select your country > [Contact].

We want to become even better! Each separate section has an identification number in the top right corner. If you want to inform us about any inconsistencies, please indicate this number with the title and the language of this documentation. Thank you for your support.

We reserve the right to make alterations which can result in a change of contents of the documentation. You can find the current version on **ifm's** website at:

→ www.ifm.com > select your country > [Service] > [Download] > [Control systems]

1.3 Changes of the manual (S16)

9191

What has been changed in this manual? An Overview:

Date	Theme	Change
2010-09-09	PID2 (FB)	parameters of the inputs corrected
2010-10-20	extended safety outputs for CR7200, CR7201	UK only: correction in topic 3816
2010-11-10	terminating resistors	correction in topic 1244

2 Safety instructions

Contents

Important!	9
What previous knowledge is required?	10

213

2.1 Important!

214

No characteristics are warranted with the information, notes and examples provided in this manual. The drawings, representations and examples imply no responsibility for the system and no application-specific particularities.

The manufacturer of the machine/equipment is responsible for the safety of the machine/equipment.

WARNING

Property damage or bodily injury possible when the notes in this manual are not adhered to!
ifm electronic gmbh does not assume any liability in this regard.

- ▶ The acting person must have read and understood the safety instructions and the corresponding chapters of this manual before performing any work on or with this device.
- ▶ The acting person must be authorised to work on the machine/equipment.
- ▶ Adhere to the technical data of the devices!
You can find the current data sheet on **ifm's** homepage at:
→ www.ifm.com > select your country > [Data sheet search] > (Article no.) > [Technical data in PDF format]
- ▶ Note the installation and wiring information as well as the functions and features of the devices!
→ supplied installation instructions or on **ifm's** homepage:
→ www.ifm.com > select your country > [Data sheet search] > (Article no.) > [Operating instructions]

ATTENTION

The driver module of the serial interface can be damaged!

Disconnecting the serial interface while live can cause undefined states which damage the driver module.

- ▶ Do not disconnect the serial interface while live.

Start-up behaviour of the controller

The manufacturer of the machine/equipment must ensure with his application program that when the controller starts or restarts no dangerous movements can be triggered.

A restart can, for example, be caused by:

- voltage restoration after power failure
- reset after watchdog response because of too long a cycle time

2.2 What previous knowledge is required?

215

This document is intended for people with knowledge of control technology and PLC programming with IEC 61131-3.

If this device contains a PLC, in addition these persons should know the CoDeSys® software.

The document is intended for specialists. These specialists are people who are qualified by their training and their experience to see risks and to avoid possible hazards that may be caused during operation or maintenance of a product. The document contains information about the correct handling of the product.

Read this document before use to familiarise yourself with operating conditions, installation and operation. Keep the document during the entire duration of use of the device.

Adhere to the safety instructions.

3 Notes on safety-related applications

Contents

General information.....	11
Standard ISO 13849	13
Safety-related programming with CoDeSys to ISO 13849	21
SafetyController	29
Safety functions.....	33

3750

3.1 General information

Contents

What does machine safety mean?.....	11
Risk assessment of a machine	11
Archiving of documentation.....	12

3836

3.1.1 What does machine safety mean?

3751

The European directive and the standards define the safety of a machine as its ability to execute their function without causing any injury. By means of the design of the machine it must be ensured that operation, equipment and maintenance in case of proper usage or foreseeable misuse can be carried out without causing hazard to persons.

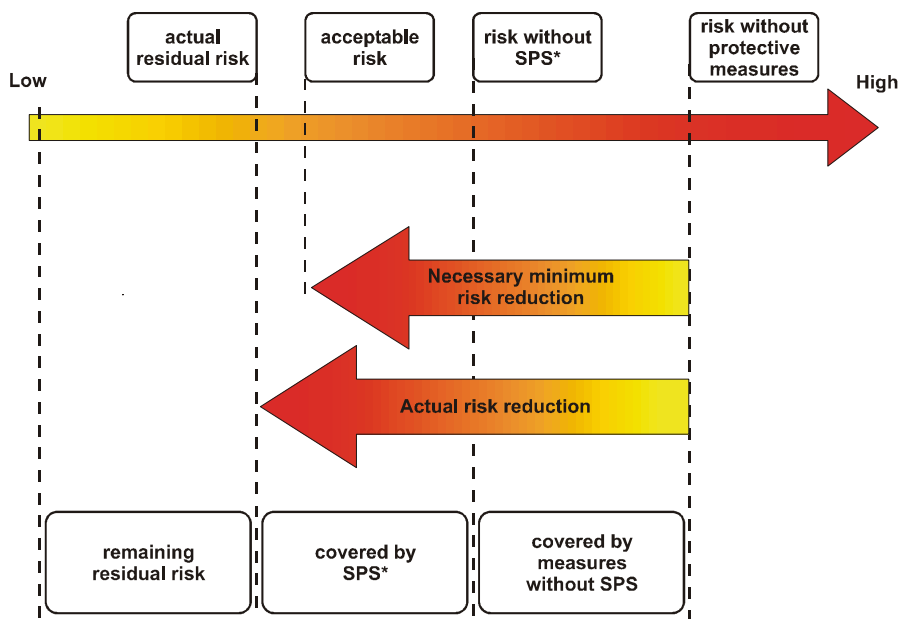
3.1.2 Risk assessment of a machine

3752

The machinery directive requires to exclude accident risks during the life cycle of the machine. Principally, there is no zero risk for technical equipment, i.e. residual risks must be reduced to an acceptable level. A risk of a machine can be considered acceptable if it is borne by the operators.

The standard defines the following risk elements:

- risk without protective measure,
- risk without safety-related protective measure (STS),
- acceptable or justifiable residual risk,
- actual residual risk.



*SPS = safety-related parts of control systems

Figure: Residual risk after risk reduction

The standard defines for the risk probability:

- frequency and duration of the hazard,
- possibility of a hazardous event,
- possibility to avoid or limit damage.

Besides the risk probability, also the possible extent of damage must be taken into account, → Risk assessment (→ page 13).

Thus, the risk depends on the severity of the harm AND the probability of occurrence of that harm.

3.1.3 Archiving of documentation

3840

Documentation must be archived as follows:

- printed source code of the application software,
- application software as two copies, write-protected (e.g. disk, CD),

Documentation must clearly show, among others, the following:

- the version of the operating system used,
- the programming software and
- the hardware used.

If requested, the version of the application software and the operating system can be compared with the archived software via the download software. When doing so, also the checksum CRC is compared.

3.2 Standard ISO 13849

Contents

Risk assessment	13
Performance level PL	14
Categories to ISO 13849	16
Risk graph to ISO 13849	18
Technology of the safety-related control functions for PL or SIL	19
Safety for bus systems	20

4007

The standard ISO 13849 belongs to the type B1 standards
→ chapter Safety standard types (→ page [395](#)).

3.2.1 Risk assessment

3756

The standard ISO 13849 is based on the principles of risk assessment. The standard is independent of the application and the technology used (controller, hydraulics, etc.). Depending on function and operating mode different safety requirements can exist in a machine.

Risk assessment is important for obtaining a safe machine. Risk potentials of any kind are to be detected before the first development step is taken or the first line of the application software is programmed. Measures to minimise the risk must be implemented by means of design and technical protective equipment.

When assigning safety tasks to the machine control system, ISO 13849 must be taken into account. One part of ISO 13849 is a risk assessment. Moreover, the 5 levels (PL a...PL e) provide information about the machine control system's resistance to the loss of the safety function.

For every single safety function the machine manufacturer must carry out the following process:

Step	Activity
1.	► Determine the required performance level PL_r (→ PL_r). → ISO 13849, annex A
2.	► Design and technically implement safety functions, identify safety-related parts which execute the safety function. → SRP/CS (→ page 396)
3.	► Determine the performance level PL of the above-mentioned safety-related parts. ► To be taken into account: - category (CAT), - mean life $MTTF_d$ (→ $MTTF_d$), - diagnostic coverage DC, - possible common cause failure CCF.
4.	► Verification of the PL for the safety function. If $PL \geq PL_r$, → continue with step 5. If $PL < PL_r$, → go back to step 2.
5.	► The validation must show that the combination for each safety function of the SRP/CS meets the corresponding requirements of ISO 13849. If all requirements are met, → continue with step 6. If not all requirements are met, → go back to step 2.
6.	When all safety functions have been analysed, → ready! Otherwise: → Analyse the next safety function

Table: Iterative process for designing the safety-related parts of controllers SRP/CS

3.2.2 Performance level PL

4011

Parts of machine control systems which are assigned to provide safety functions are called "safety-related parts of control systems" (SRP/CS). These parts can consist of hardware and / or software and can either be separate from the control machine or an integral part of it. In addition to providing safety functions an SRP/CS can also provide operating functions, e.g. a two-hand control for starting a process.

The ability of safety-related parts of controllers to perform a safety function at foreseeable conditions is assigned to one of 5 levels of the so-called performance level PL (PL a...PL e). This performance level is defined as the probability of a dangerous failure per hour.

Performance level	Probability of a dangerous failure [1/h]	Probable operating time without a dangerous failure [h]
PL a	$\geq 0.000\ 01 \dots < 0.000\ 1$	$> 10\ 000 \dots \leq 100\ 000$
PL b	$\geq 0.000\ 003 \dots < 0.000\ 01$	$> 100\ 000 \dots \leq 333\ 333$
PL c	$\geq 0.000\ 001 \dots < 0.000\ 003$	$> 333\ 333 \dots \leq 1\ 000\ 000$
PL d	$\geq 0.000\ 000\ 1 \dots < 0.000\ 001$	$> 1\ 000\ 000 \dots \leq 10\ 000\ 000$
PL e	$\geq 0.000\ 000\ 01 \dots < 0.000\ 000\ 1$	$> 10\ 000\ 000 \dots \leq 100\ 000\ 000$

The probability of a dangerous failure of the safety function depends on several factors, e.g.:

- the hardware and software structure,
- the scope of the fault detection mechanisms (= diagnosis coverage degree DC),
- the reliability of components (= mean time to failure $MTTF_d$),
- the failures with a common cause CCF,
- the design process,
- operating stress,
- environmental conditions and
- the operating conditions.

NOTE: For achieving the PL, further measures are necessary in addition to the maximum permissible probability of a dangerous failure per hour.

The PL of the SRP/CS must be determined by assessing the following aspects:

- $MTTF_d$ value of individual components (\rightarrow ISO 13849, annexes C+D),
- the diagnosis coverage degree DC,
- the possible failure of several components due to the same cause CCF,
- the structure,
- the behaviour of the safety function under fault conditions,
- safety-relevant software,
- systematic failures,
- the ability to perform a safety function under foreseeable conditions.

If required, further aspects are important for the PL:

- operational aspects
- demand rate rd ,
- test rate rt .

In order to simplify the determination of the achieved PL, ISO 13849 provides a method based on the categorisation of structures according to specific design criteria and specified behaviour under fault conditions. These categories are allocated one of 5 levels, the categories CAT B, CAT 1...CAT 4:

Category	Requirement	System behaviour
CAT B	The system must withstand the influences to be expected.	A fault may lead to a loss of the safety function.
CAT 1	The requirement of CAT B must be met. Use of tried-and-tested components and principles.	System behaviour of CAT B, but with a lower fault probability.
CAT 2	The requirement of CAT 1 must be met. The safety function must be checked by the machine control system at defined intervals.	System behaviour of CAT B, but a fault is identified at the following check.
CAT 3	The requirement of CAT 1 must be met. A single fault must be identified and must not lead to a loss of the safety function.	Some but not all faults are detected. Accumulation of undetected faults can lead to a loss of the safety function.
CAT 4	The requirement of CAT 3 must be met. Single faults are detected in each safety-related part. Accumulation of faults must not lead to a loss of the safety function.	Faults are detected in due time.

Category	CAT B	CAT 1	CAT 2	CAT 2	CAT 3	CAT 3	CAT 4
DC	none	none	low	medium	low	medium	high
low MTTFd	PL a	not covered	PL a	PL b	PL b	PL c	not covered
medium MTTFd	PL b	not covered	PL b	PL c	PL c	PL d	not covered
high MTTFd	not covered	PL c	PL c	PL d	PL d	PL d	PL e

DC = diagnosis coverage degree

MTTF_d = mean life

❗ NOTE

Part of the inputs and outputs of the SafetyController is approved for applications ...

- up to PL d to ISO 13849,
- up to SIL CL 2 to IEC 62061.

A prerequisite for this is that the inputs and outputs of the SafetyController are wired and evaluated by the application program (as described in the chapter Configurations (→ page [61](#))).

3.2.3 Categories to ISO 13849

4013

The standard ISO 13849 differentiates between the following categories:

Characteristic	Category				
	B	1	2	3	4
Draft according to the corresponding standards. Must withstand to the expected influences.	X	X	X	X	X
Fundamental safety principles	X	X	X	X	X
Tried-and-tested safety principles		X	X	X	X
Use of tried-and-tested components.		X	X	X	X
Mean time to dangerous failure MTTFd	low to medium	high	low to high	low to high	high
Fault detection (tests)			X	X	X
Single fault tolerance				X	X
Consideration of fault accumulation					X
Diagnostic coverage DC	none	none	medium to low	medium to low	high
Measures against CCF			X	X	X
Mainly characterised by	component selection		structure		

X = characteristic must be met

Especially **for category 3 / performance level d** this means in detail:

According to ISO 13849 the SRP/CS shall, as a minimum, be designed, constructed, selected, assembled and combined in accordance with the relevant standards, and meet basic safety principles for the specific application to withstand the following:

- the expected operating requirements e.g. the reliability regarding switching capacity and switching frequency,
- the influence of the material to be processed, e.g. the cleaning agent,
- other relevant external influences, e.g. mechanical vibrations, electromagnetic interference, interruptions or disturbances of the power supply.

SRP/CS shall be designed and constructed using tried-and-tested components and safety principles. This means:

- A tried-and-tested component for a safety-related application has been widely used with successful results in similar applications in the past.
- Or it has been produced and verified by using principles which demonstrate its suitability and reliability for safety-related applications.
- Newly developed components and safety principles can be considered equivalent if they meet the foregoing condition.

The MTTF_d (mean time to failure) of each redundant channel (depending on the PL_r (required performance level)) must be low to high.

In SRP/CS the machine control system must check its functions at suitable intervals.

- The check must be carried out...
 - when starting the machine
 - prior to initiation of hazardous situations,
 - periodically during operation if the risk assessment and operating mode indicate this to be necessary.
- The test must allow the operation if no fault was detected or generate an output which initiates suitable control measures if a fault is detected.
- The test itself must not lead to a hazardous situation.

The diagnosis coverage DC of the entire SRP/CS including fault detection is to be as high as possible if the $MTTF_d$ is low.

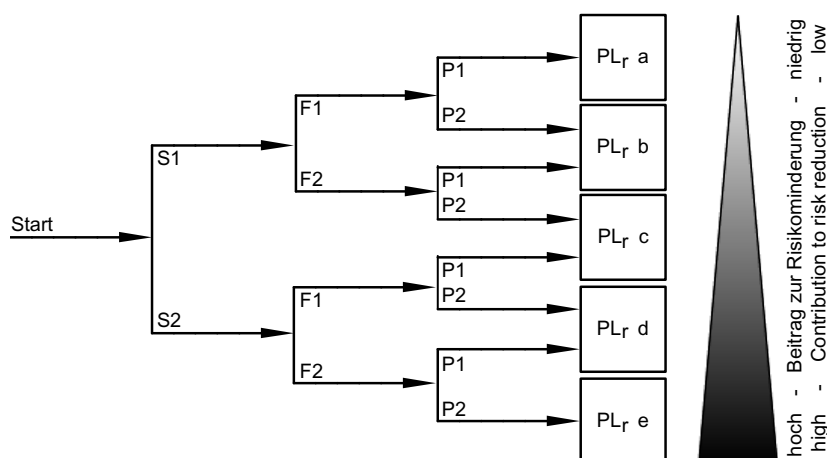
Measures against CCF (common cause failure) must be taken.

A single fault in one of the components of an SRP/CS must be detected and must not lead to a loss of the safety function.

- If feasibly in an appropriate manner, the fault must be detected at or before the next time when the safety function is required, e.g. by positively guided relay contacts and monitoring of redundant outputs.
- Some but not all faults are detected.
- Accumulation of undetected faults can lead to a loss of the safety function.

3.2.4 Risk graph to ISO 13849

3757



Graphics: Risk graph to ISO 13849

Legend:

S = How severe is the possible injury?

S1 = slight, reversible injury

S2 = severe, irreversible injury of one or several persons or death of a person

F = How often occurs the hazard and how long is the exposure to the hazard? ¹⁾

F1 = seldom to less often and / or exposure time is short

F2 = frequently to continuously and / or exposure time is long

P = Is it possible to avoid that the person is exposed to the hazard? ²⁾

P1 = possible under certain conditions

P2 = scarcely possible

PLr = required performance level

a = low contribution to risk reduction

...

e = high contribution to risk reduction

¹⁾ For the frequency it is not important whether always the same person is exposed to the hazard or whether several persons are exposed to the hazard one after the other.

²⁾ Can the corresponding person identify or avoid the hazard in due time or can the impact of an accident significantly be reduced? This depends, among others, on the following aspects:

- operation with or without supervision,
- operation by qualified or unqualified staff,
- the speed with which the hazard arises
- good or bad possibility to evade hazard by escaping,
- practical experiences with the safety of such a process.

The graphic representation shown should be taken into account for each safety function. The risk assessment method is based on ISO 14121 and should be carried out according to ISO 12100-1.

3.2.5 Technology of the safety-related control functions for PL or SIL

4012

IEC 62061 and ISO 13849-1 specify requirements for the design and implementation of safety-related control systems of machines. The user of one of the two standards can assume that he meets the required safety requirements if he works in compliance with the indicated areas of application. The following table summarizes the application areas of IEC 62061 and ISO 13849-1.

Type	Technology of the safety-related control functions	Performance level PL to ISO 13849	Safety integrity level SIL to IEC 62061
A	not electrical, e.g. hydraulic	limited to the specified architectures	- not contained -
B	electromechanical, e.g. relay and / or non complex electronics	limited to the specified architectures ¹⁾ up to PL e	all architectures up to SIL CL 3
C	complex electronics ²⁾ , e.g. programmable	limited to the specified architectures ¹⁾ up to PL d	all architectures up to SIL CL 3
D	A combined with B	limited to the specified architectures ¹⁾ up to PL e	all architectures ³⁾
E	C combined with B	limited to the specified architectures ¹⁾ up to PL d	all architectures up to SIL CL 3
F	C combined with A or C combined with A and B	limited to the specified architectures ¹⁾ up to PL d	all architectures ³⁾

¹⁾ specified architectures → ISO 13849-1, → chapter Categories to ISO 13849 (→ page [16](#)).

²⁾ for complex electronics: use of the specified architectures to ISO 13849-1 up to PL d or any architecture to IEC 62061.

³⁾ for non electrical technology: use of the components in accordance with ISO 13849-1 as subsystems.

For information purposes we show here how the results of the standard ISO 13849 (Performance Level PL) and the standard IEC 62061 (Safety Integrity Level SIL) can be compared:

Performance level PL to ISO 13849		Safety integrity level SIL to IEC 62061
PL a		(no equivalence)
PL b	↔	SIL CL 1
PL c	↔	SIL CL 1
PL d	↔	SIL CL 2
PL e	↔	SIL CL 3
(no equivalence)		SIL CL 4

3.2.6 Safety for bus systems

3761

In addition to the above-mentioned measures for the set-up of safe machine control systems, the faults during the transmission of "safe data" via a bus system must be detected and controlled.

For the development of the protocol supplement CANopen Safety, the measures below were taken into account and integrated into the protocol.

Info

In conjunction with the SafetyController, CANopen Safety can be used in applications up to PL d.

Fault	useful measure							
	1	2	3	4	5	6	7	8
Repetition of old messages which are no longer valid.	X	X						
Loss of messages	X							
Insertion of wrong messages	X		X	X	X			
Wrong order of messages	X	X						
Data corruption							X	X
Delay in message transmission		X	X					
Mixing of safety-related and non safety-related messages				X		X		X

- 1 = assign and check a serial number
- 2 = timestamp (to be sent to specified times)
- 3 = expected time (time-out to be monitored)
- 4 = echo (receiver repeats the command)
- 5 = identification of transmitter and receiver
- 6 = message identification
- 7 = data storage
- 8 = encryption methods

3.3 Safety-related programming with CoDeSys to ISO 13849

Contents

Safety-related applications software (SRASW)	22
Rules on the specification of the SRASW	23
Rules for selecting the tools, libraries, languages.....	23
Rules on the program structure.....	24
Rules on SRASW and non safety-related software in one component	25
Rules on the software implementation / coding	25
Rules on the safety-related function blocks	25
Rules on the use of variables.....	26
Rules on the use of data types.....	27
Rules for testing safety-related software.....	27
Rules for documenting safety-related software.....	28
Rules on the verification of safety-related software	28
Rules for subsequent program modifications.....	28

4120

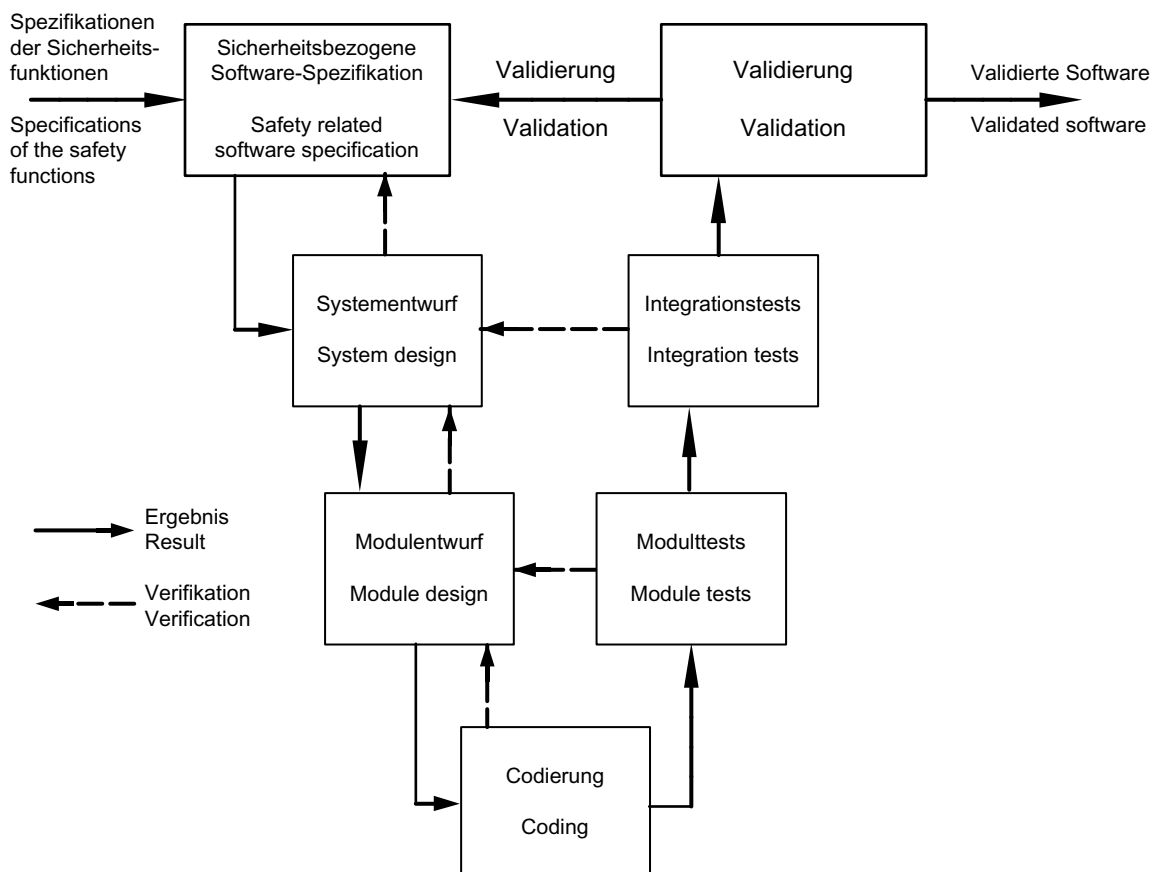
For SRASW in components with PL_r c to PL_r e (**ecomatmobile** controller only to PL_r d possible) the following measures are additionally required or recommended:

3.3.1 Safety-related applications software (SRASW)

3834

- Principally all information and notes in this manual must be taken into account for creation of the application software.
- All activities in the life cycle of safety-related embedded software or application software must mainly target to avoid faults caused during the software life cycle (→ following graphics).

The main goal of the following requirements is to achieve readable, understandable and serviceable software.



Graphics: Simplified V model of the software life cycle to ISO 13849

3.3.2 Rules on the specification of the SRASW

4122

- ▶ Check the specification of the SRASW.
- ▶ Make the specification available to all persons who are involved in the life cycle of the software.

The specification must contain the following descriptions:

- safety functions with required PL_r and corresponding operating modes,
- performance criteria, e.g. response times,
- hardware architecture with external signal interfaces,
- methods for detecting and controlling external failures.

3.3.3 Rules for selecting the tools, libraries, languages

4123

- ▶ Use appropriate tools proven in use.
- ▶ Tools should enforce language subsets and programming guidelines, or at least guide the developer. This is largely supported by CoDeSys.
- ▶ During the developing phase check as often as possible the application software by means of compilation in CoDeSys. In this way systematic errors are detected and eliminated at an early stage.
- ▶ Use validated function block libraries (FB) in compliance with the applicable standards:
 - safety-related FB libraries delivered by **ifm** or
 - validated application-specific FB libraries.

Principally, all programming languages are allowed, however:

Only the following programming languages shall be used for safety-related applications:

- Limited variability language (LVL) that provides the capability of combining predefined, application-specific library functions.
In CoDeSys these are LD (ladder diagram) and FBD (function block diagram).
- Full variability language (FVL) provides the capability of implementing a wide variety of functions. These include e.g. C, C++, Assembler. In CoDeSys it is ST (structured text).
- ▶ Structured text is recommended exclusively in separate, certified functions, usually in embedded software.
- ▶ In the "normal" application program only LD and FBD should be used. The following minimum requirements shall be met.

In general the following minimum requirements are made on the safety-related application software (SRASW):

- ▶ Modular and clear structure of the program. Consequence: simple testability.
- ▶ Functions are represented in a comprehensible manner:
 - for the operator on the screen (navigation)
 - readability of a subsequent print of the document.
- ▶ Use symbolic variables (no IEC addresses).
- ▶ Use meaningful variable names and comments.
- ▶ Use easy functions (no indirect addressing, no variable fields).
- ▶ Defensive programming.
- ▶ Easy extension or adaptation of the program possible.

We highly recommend to establish procedures and data backup to identify and archive all documents, software modules, results of verification and validation and tool configuration related to a specific SRASW.

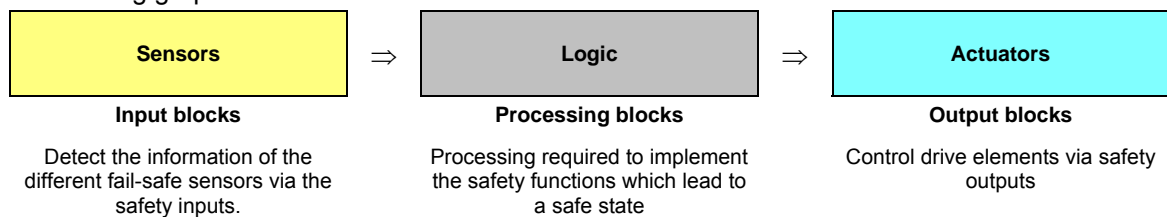
3.3.4 Rules on the program structure

4124

- ▶ Design and implement the interface between user and SRP/CS such that no person is endangered during the intended use (= functions and features) or reasonably foreseeable incorrect use of the machine.
- ▶ Use ergonomic principles so that machine and controller, including the safety-related components, are easy to use and the user is not tempted to act in a dangerous way.
- ▶ Apply the safety requirements in order to comply with the ergonomic principles of ISO 12100-2.
- ▶ Use semi formal procedures to describe the data and control flow, e.g. status diagram or program flow diagram.

Structure the program to generate a consistent and understandable frame in which different processes can easily be found.

- ▶ Use templates for typical programs and functions.
- ▶ The complete application should be called by the program block PLC_PRG. Nothing else should be programmed in the PLC_PRG (no logic processing).
- ▶ Implement safety functions separately from pure control functions, i.e. in their own program and function blocks.
- ▶ Use safety functions from validated safety-related POU libraries.
- ▶ Write safety-related functions blocks (FBs) with code lengths as minimized as possible.
- ▶ Within the function block (FB) the code should be executed with only one input jump and only output jump.
- ▶ Describe the tasks of the functions in the comment.
- ▶ For clear differentiation precede the names of safety functions with an "S_".
- ▶ Comment each program section of the source code to facilitate updates, checks and corrections.
- Architecture model of 3 stages: Inputs → Processing → Outputs.
→ e.g. ISO 13849, annex J
→ following graphics:



- ▶ Assign each safety output to only one program part each. No assignments to several program parts!
- ▶ Processing of the program is NOT to depend on variables such as jump addresses which are calculated during the runtime, otherwise there is the risk of random faults in the program processing. However, conditional jumps are permitted.
- ▶ Use methods for detection of external failures and for defensive programming within input, processing and output blocks which lead to the safe state.
- ▶ Activate the controller-specific options for monitoring the total runtime (→ Watchdog behaviour (→ page 109)) and set them significantly below the error tolerance time (→ Safety considerations (→ page 31))

3.3.5 Rules on SRASW and non safety-related software in one component

4125

If SRASW and non safety-related software are combined in one component (otherwise impossible in the SafetyController):

- ▶ Code SRASW and non safety-related software in different function blocks with well-defined data links.
- ▶ Safety-related functions are not to call any non safety-related functions. Check this in CoDeSys with the function [Project] > [Show Call Tree].
- ▶ Non safety-related functions are to activate only standard safety-related functions. Check this in CoDeSys with the function [Project] > [Show Call Tree].
- ▶ There shall be no logical combination of non safety-related and safety-related data which could lead to downgrading of the integrity of the safety-related signals.

Example: OR function of a safety-related signal and a non safety-related signal where the result controls safety-related signals.

3.3.6 Rules on the software implementation / coding

4126

- ▶ The code must be readable, understandable and testable. Therefore **symbolic** variables are to be used instead of explicit hardware addresses.
- ▶ Use justified or accepted programming guidelines.
- ▶ On the application layer use data integrity and plausibility tests, e.g. range checks: "defensive programming".
- ▶ Test the code by means of simulation.
- ▶ For PL d (or PL e): Verify the code by control and data flow analysis.

3.3.7 Rules on the safety-related function blocks

4127

- ▶ Function blocks validated by **ifm** are preferred.
- ▶ Check whether the accepted operating conditions for these validated blocks comply with the conditions of the program.
- ▶ Limit the size of the coded block to the following guideline values:
 - Parameters: max. 8 digital + 2 INT inputs +1 output, (+ diagnosis and status signals),
 - Function code: max. 10 local variables, max. 20 Boolean equations.
- ▶ Function blocks should not modify the values of the global variables.
 - Rules on the use of variables (→ page [26](#)).
- ▶ Check a digital value by means of a specified comparative test to ensure the scope of validity.
- ▶ Inconsistencies of variables to be processed should be detected by this function block and not afterwards outside the function block.
- ▶ The error code of a block is to be externally accessible to differentiate the error from others.
- ▶ After noticing the fault, comment the possible error code and the status of the block in the source code.
- ▶ For the error case comment the reset of the block or the recovery of the normal status in the source code.

3.3.8 Rules on the use of variables

4128

- ▶ Each output and each variable is to be switched on only at ONE position and switched off only at ONE position in the program (centralised conditions).
- ▶ Variables which are used with read access several times during a cycle and which are written by another task should be copied, at the beginning of the cycle, to a separated variable which is nowhere else changed. Consequence: The values of the variables remain consistent in the cycle.
- ▶ Protect large variables requiring several cycles for reading or writing (e.g. in case of interruptions of read or write processes) against inconsistencies.
- ▶ Assign each used address with "AT" in the declaration to a variable. Never use IEC addresses directly in the program code!
- ▶ Mark the input / output variable by a prefix (e.g. "I_" / "O_" and define it separately in the declaration, e.g.:

```
VAR_INPUT
  I_VARIABLE: WORD; (* input variable *)
  I_...
END_VAR
VAR_OUTPUT
  O_VARIABLE: WORD; (* output variable *)
  O_...
END_VAR
```

- ▶ For variables the memory is automatically assigned suitably. Therefore, if possible, DO NOT use concrete IEC addresses for flag "%M..." because of the error rate during assignment.
- ▶ If possible, DO NOT use addresses several times because of unclear side effects. If access is to be made by word or by bit, define a variable for the word and access the bit by means of the bit access `variable.bitnumber`. Examples:

Good example:	Bad example:
<pre>VAR_CONSTANT EnableBit: INT:=0; END_VAR VAR_GLOBAL Flags AT %QW12: WORD; END_VAR Flags:=0; Flags.0:=TRUE;</pre>	<pre>VAR_GLOBAL Flags AT %QW12: WORD; Enable AT %QX12.0: BOOL; END_VAR Flags:=0; Enable:=TRUE;</pre>

- ▶ Non safety-related POU's must not have writing access to safety-related variables. Check this with the function [Project] > [Deliver cross-link list] in CoDeSys.
- ▶ Safety-related POU's must not have writing access to non safety-related variables. Check this with the function [Project] > [Deliver cross-link list] in CoDeSys.
- ▶ For safety-related retain variables provide explicit test cases for the power down case.
- ▶ Use an own declaration line for each variable. No enumeration of identical variable types in the same declaration line! → Examples:

Good example:	Bad example:
<pre>VAR A: BOOL; (* 1st Variable *) B: BOOL; (* 2nd Variable *) C: BOOL; (* 3rd Variable *) END_VAR</pre>	<pre>VAR A, B, C: BOOL; (* Some Variables *) END_VAR</pre>

- ▶ Assign a unique and self-explanatory name and describe it in the comment of the source text for each global variable, input and output.
- ▶ Precede the name of global variables with a "G_" for clear differentiation.
Precede the name of safety-related global variables with a "GS_".
Precede the name of safety-related variables with an "S_".
- ▶ Check values of variables on plausibility:
Never compare for "equal" (=) but for "greater than" (\geq) or "smaller than" (\leq) because exactly "equal" is actually never achieved or found in the measuring cycle.

3.3.9 Rules on the use of data types

4130

Among the data types defined in CoDeSys the following are approved for safety-related applications:

Data type	approved for safety-related applications
BOOL	yes
BYTE SINT USINT	yes
WORD INT UINT	yes
DWORD DINT UDINT	yes
TIME TOD DATE DT	yes
STRING	conditionally: use does not make sense due to missing safety-related input/output units.
LREAL REAL	conditionally: error-prone due to rounding errors thus no check with EQ operator possible. Pay attention to invalid operations!
ARRAY	conditionally: only with explicit range check!
STUCT	yes
Enumerated types	yes
Sub-data types	yes
POINTER	conditionally: no pointer arithmetic! Range check! New assignment of the pointer value at the beginning of each cycle!

- ▶ An explicit range check of the index should be carried out prior to each access to an array.
- ▶ In case of an index above or below the range which cannot be explained by the application the controller is to be put in the safe state.

3.3.10 Rules for testing safety-related software

4131

- ▶ Carry out a complete function test for all parts of the application software (test input must be deactivated!).

The appropriate validation procedure is black box testing of the functional behaviour and the performance criteria, e.g. timing behaviour.

For PL d (or PL e) it is recommended to execute test cases based on boundary value analyses.

We recommend a test planning. The test planning should contain test cases with completion criteria and required tools.

I/O tests must ensure that the safety-related signals are correctly used in the SRASW.

3.3.11 Rules for documenting safety-related software

4133

- ▶ All life cycle and modification activities are to be documented.
- ▶ Documentation must be complete, available, readable and understandable.
- ▶ Documentation within the source text must contain module headers with the following information:
 - indication of a legal entity,
 - description of the functions and the inputs and outputs,
 - version of the FB library used and
 - sufficient documented networks, instructions and declaration lines.

3.3.12 Rules on the verification of safety-related software

4134

E.g. verification, inspection, walkthrough or other appropriate activities.

3.3.13 Rules for subsequent program modifications

3838

- ▶ Upon approval of the application software in safety-related applications no more modifications may be made in this software.
 - ▶ The application software may only be used with an unchanged (after approval) operating system software.
 - ▶ Upon approval only the HEX file `name_of_projectfile.H86` may be loaded via the downloader software in the controller modules.
- chapter Creating application program (→ page [111](#)).
- ▶ If subsequent modifications are made, the entire application software must be tested and certified again.
 - After modifying an SRASW, an impact analysis must be carried out to ensure compliance with the specification.
 - After modifying, carry out appropriate life cycle activities.
 - Check the access rights regarding the modifications. Document the history of modifications.

Under the following conditions the new certification may not be necessary:

- a new risk assessment was made for the change,
- NO safety-related elements were changed, added or removed,
- the change was correctly documented.

3.4 SafetyController

Contents

ExtendedSafetyController CR7200/CR7201	29
Safe state	29
Safety-related inputs and outputs	29
Fail-safe sensors and safety signal transmitters	30
Test input.....	30
Use in applications up to CAT 3 / PL d	31

3841

The mobile controller SafetyController is a single-channel controller which meets the following requirements:

Safety class	according to the standard
Performance level PL d / CAT 3	ISO 13849-1 (→ standard ISO 13849 (→ page 13))
Safety integrity level SIL 2	IEC 62061-1

→ chapter Safety concept (→ page [55](#))

3.4.1 ExtendedSafetyController CR7200/CR7201

4036

In this special version of the SafetyController two PLC modules are integrated in one housing. They are internally connected via a bus.

If both PLC modules are to be used for safe applications, both PLC modules must be loaded with an own independent program. Otherwise the secondary control device (slave) cannot be used for safety-related functions.

→ chapter Operating modes of the ExtendedController (→ page [103](#))

The internal interface is only available to non safety-related data exchange. If required, safe communication is implemented externally via CANopen Safety.

→ chapter CANopen Safety in safety-related applications (→ page [231](#))

3.4.2 Safe state

3829

The safe state of an output with safety function is the power-free status (L signal, "0").

This state must be implemented via 2 separate and independent switch-off modes. To do so, approved switching elements are to be used.

3.4.3 Safety-related inputs and outputs

3830

Safety-related inputs and outputs must be used redundantly. That means:

- redundant connection of signal transmitters to the inputs,
- redundant outputs as second switch-off mode in the application (e.g. hydraulic valve and pump).

→ chapter Safety concept (→ page [55](#))

3.4.4 Fail-safe sensors and safety signal transmitters

3831

The binary safety signal transmitters must be connected to two different input groups. To do so, the inputs marked as safe must be used as far as possible. For the redundancy the analogue channels (configured as digital inputs) must be used.

→ chapter Function configuration of the inputs and outputs (→ page [78](#))

Analogue fail-safe sensors must be connected to the channels marked as safe, as far as possible with different input signals (current / voltage) in a diverse manner.

If fail-safe inductive sensors are connected, the outputs and inputs provided must be used.

→ chapter Inputs for fail-safe inductive sensors (→ page [83](#))

3.4.5 Test input

3837

The test input must be set if e.g. the software is to be loaded into the controller (program download).

→ chapter TEST mode (→ page [60](#)).

During operation of the application the test input must not be used.

If outputs are configured as safety-related (MODE byte = OUT_SAFETY), they cannot be used when the test input is active.

The safety-related outputs are only available after the following actions:

- when the test input is deactivated and
- a reset of the controller has been carried out (power off and on).

Nevertheless, to enable software monitoring (no program download) with the programming system CoDeSys for maintenance purposes, SET_DEBUG (→ page [352](#)) is provided.

NOTE

Risk of misuse and malfunction!

SET_DEBUG should be accessible only for authorised persons, e.g. locked with a key switch.

3.4.6 Use in applications up to CAT 3 / PL d

3828

If the SafetyController is used in applications up to safety category CAT 3 or performance level PL d or safety-integrity level SIL CL 2, the special conditions and obligations described below must be adhered to.

Configuration and use in safety-related applications must be based on a risk assessment.

In addition, the following points must be adhered to (→ Safety considerations (→ page [31](#))).

→ also chapter Safety-related application software (SRASW) (→ page [22](#))

Safety considerations

3833

On the basis of the monitoring functions of the above-mentioned points implemented in the operating system it must be assessed if with corresponding design of the safety system and the application software process-dependent safety functions are met according to PL d.

Fault tolerance time

In this context, especially the fault tolerance time of the process is to be assessed.

The max. time it may take between the occurrence of a fault and the establishment of the safe state in the application without having to assume a danger for people.

The max. cycle time of the application program (in the worst case 100 ms, → Watchdog (→ page [109](#))) and the possible delay and response times due to switching elements have to be considered.

The resulting total time must be smaller than the fault tolerance time of the application.

First fault occurrence time

Moreover, the first fault occurrence time must be taken into account.

Time until the first failure of a safety element.

The operating system verifies the controller by means of the internal monitoring and test routines within a period of max. 30 s.

This "test cycle time" must be smaller than the statistical first fault occurrence time for the application.

Operating system and software versions

835

The number of the operating system and the article number of the device must correspond, e.g.:
CR7020_V050100.H86 for CR7020.

! NOTE

The software versions suitable for the selected target must always be used:

- operating system (CRnnnn_Vxxyzz.H86 / CRnnnn_Vxxyzz.HEX)
- PLC configuration (CRnnnn_Vxx.CFG)
- device library (ifm_CRnnnn_Vxxyzz.LIB)
- and the further files (→ chapter Overview of the files and libraries used (→ page [376](#)))

CRnnnn	device article number
Vxx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "CR0032") and the software version number "xx" (e.g. "02") must always have the same value! Otherwise the device goes to the STOP mode.

The values for "yy" (release number) and "zz" (patch number) do **not** have to match.

IMPORTANT: the following files must also be loaded:

- the internal libraries (created in IEC 1131) required for the project,
- the configuration files (*.CFG)
- and the target files (*.TRG).

Principally the most important data of the operating system is automatically monitored via a checksum (CRC check). For safety-related data which is generated by the application program CHECK_DATA (→ page [359](#)) can additionally be used.

3.5 Safety functions

Contents

SAFE_ANALOG_OK (FB).....	34
SAFE_FREQUENCY_OK (FB).....	36
SAFE_INPUTS_OK (FB).....	38
SAFETY_SWITCH (FB).....	41

907

For safety functions of the SafetyController we provide the following certified function blocks:

In the safety library `_ifm_SafetyIO_Vxyxyz.LIB`:

- SAFE_ANALOG_OK (→ page [34](#))
- SAFE_FREQUENCY_OK (→ page [36](#))
- SAFE_INPUTS_OK (→ page [38](#))

In the device library `_ifm_CR7xxx_Vxyxyz.LIB`:

- SAFETY_SWITCH (→ page [41](#))
- CAN_SAFETY_TRANSMIT (→ page [237](#)) *)
- CAN_SAFETY_RECEIVE (→ page [239](#)) *)

❗ NOTE

CAN SAFETY FBs need 2 11-bit operated CAN interfaces at the same time.

When CAN SAFETY FBs are used the 2nd CAN interface can therefore not be used for SAE J1939 FBs (29 bits).

*) description of CANopen Safety and its functions:
→ chapter CANopen Safety in safety-related applications (→ page [231](#)).

3.5.1 SAFE_ANALOG_OK (FB)

3863

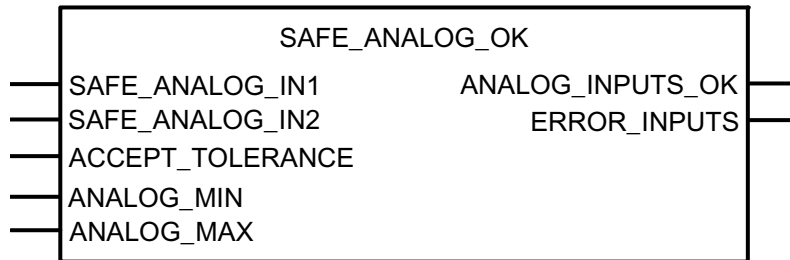
Contained in the library:

`ifm_SafetyIO_Vxxyyzz.LIB`

Available for the following devices:

- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Symbol in CoDeSys:



Description

3886

SAFE_ANALOG_OK monitors safety-related analogue input signals.

In safety-related applications input signals have to be evaluated redundantly and as diversified as possible. This FB evaluates 2 input channels (if possible from different input groups) and compares if the measured values are within defined set tolerance and within the defined value range.

The two channels should each be processed with INPUT_ANALOG (→ page [251](#)) Furthermore, it is useful to evaluate the signal voltage only in a limited range (e.g. 10...90 %). This allows to detect the following errors:

- short to ground (< 10 %)
- wire break (< 10 %)
- short to voltage supply (> 90 %)
- short circuit (< 10 %)

The result is also provided redundantly:

Comparison of inputs shows	ANALOG_INPUTS_OK	ERROR_INPUTS
Values within tolerance and within value range	TRUE	FALSE
Values within tolerance and out of value range	FALSE	TRUE
Values out of tolerance and within value range	FALSE	TRUE
Values out of tolerance and out of value range	FALSE	TRUE

→ Example for SAFE_ANALOG_OK (→ page [35](#))

Parameters of the inputs

3889

Parameter	Data type	Description
SAFE_ANALOG_IN1	DWORD	safety-related analogue value 1
SAFE_ANALOG_IN2	DWORD	safety-related analogue value 2 (reference value)
ACCEPT_TOLERANCE	BYTE	permissible deviation analogue value 1 to analogue value 2 in [%] value range = 0...100 %
ANALOG_MIN	WORD	bottom limit of the analogue input value (e.g. wire-break detection)
ANALOG_MAX	WORD	upper limit of the analogue input value (e.g. detection short to supply voltage)

Parameters of the outputs

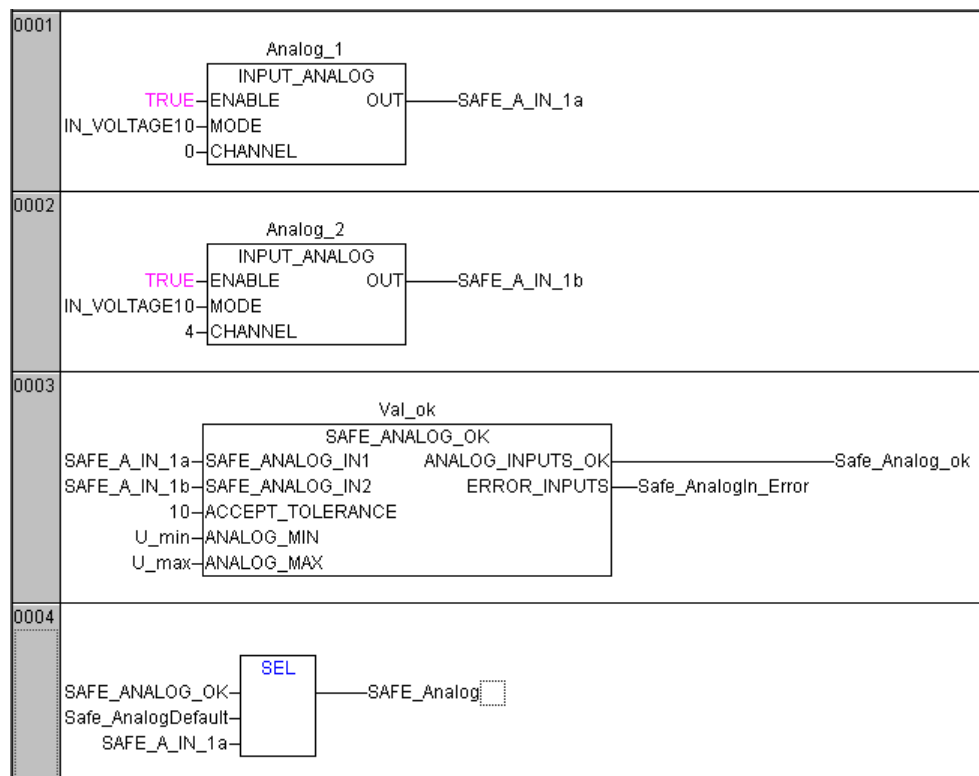
3893

Parameter	Data type	Description
ANALOG_INPUTS_OK	BOOL	TRUE: values within tolerance and within value range FALSE: values out of tolerance and/or out of value range
ERROR_INPUTS	BOOL	TRUE: values out of tolerance and/or out of value range FALSE: values within tolerance and within value range

Example: SAFE_ANALOG_OK

3887

In the following example for the redundant use of input signals **SAFE_ANALOG_OK** (→ page 34) (from library `ifm_SafetyIO_Vxxyzz.Lib`) compares the two analogue values **SAFE_A_IN_1a** and **SAFE_A_IN_1b**. If the difference is smaller than or equal to the value of **ACCEPT_TOLERANCE** the two analogue values are accepted as equal and can be further processed. If not, the error message **Safe_AnalogIn_Error** is given.



3.5.2 SAFE_FREQUENCY_OK (FB)

3865

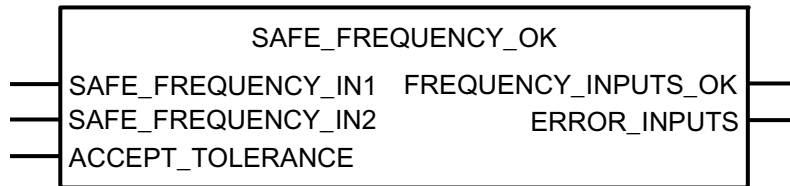
Contained in the library:

`ifm_SafetyIO_Vxxyyzz.LIB`

Available for the following devices:

- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Symbol in CoDeSys:



Description

3890

SAFE_FREQUENCY_OK monitors safety-related frequency signals.

In safety-related applications input signals have to be evaluated redundantly. This FB evaluates 2 input channels and compares if the measured values are within the set tolerance and within the defined value range.

The input channels should be in two different input groups. One input channel should be measured with FREQUENCY (→ page 259) and the other channel with PERIOD (→ page 261). This will meet the demands for redundancy and diversity.

The result is also provided redundantly:

Comparison of inputs shows	FREQUENCY_INPUTS_OK	ERROR_INPUTS
Values within tolerance	TRUE	FALSE
Values out of tolerance	FALSE	TRUE

→ Example for SAFE_FREQUENCY_OK (→ page 37)

Applications

3802

Due to the different measuring methods errors can occur when the frequency is determined:

FREQUENCY (→ page 259) is suited for frequencies between 0.1...50 kHz; the error is reduced at high frequencies.

PERIOD (→ page 261) carries out the period measurement. It is thus suitable for frequencies lower than 1 kHz. The measurement of higher frequencies has a strong impact on the cycle time. This has to be taken into account when designing the application software.

As a consequence, a safe measurement of frequencies is only possible between 100...1000 Hz.

Parameters of the inputs

3894

Parameter	Data type	Description
SAFE_FREQUENCY_IN1	DWORD	safety-related frequency value 1
SAFE_FREQUENCY_IN2	DWORD	safety-related frequency value 2 (reference value)
ACCEPT_TOLERANCE	BYTE	permissible deviation frequency value 1 to frequency value 2 in [%] value range = 0...100 %

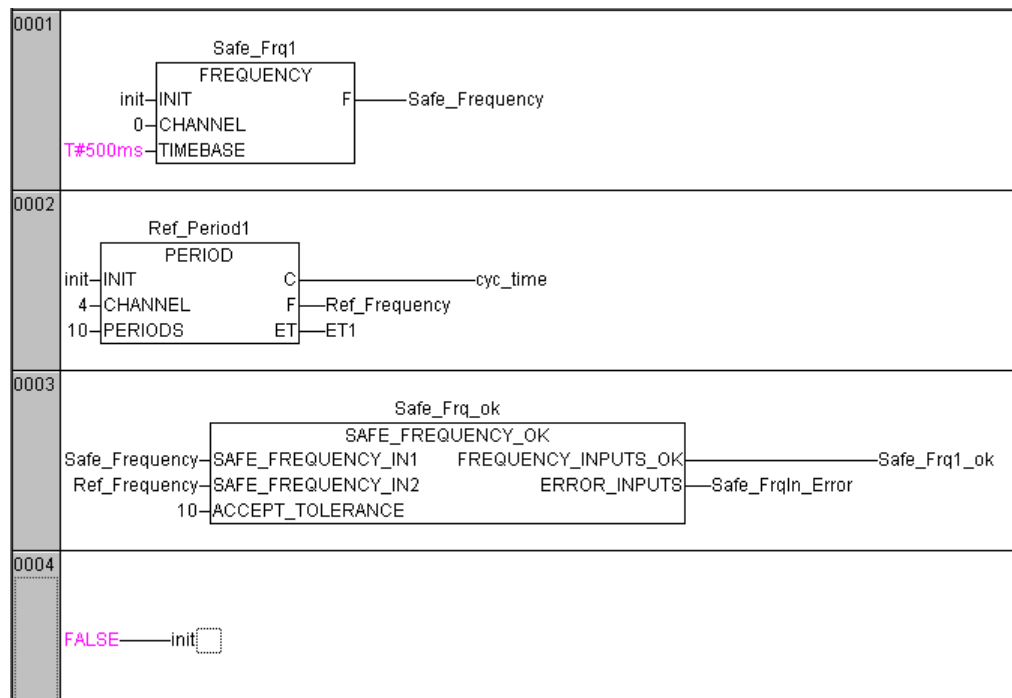
Parameters of the outputs

3895

Parameter	Data type	Description
ANALOG_FREQUENCY_OK	BOOL	TRUE: values within tolerance FALSE: values out of tolerance
ERROR_INPUTS	BOOL	TRUE: values out of tolerance FALSE: values within tolerance

Example: SAFE_FREQUENCY_OK

3891



In the example above **SAFE_FREQUENCY_OK** (→ page 36) compares the two frequency values **SAFE_frequency** und **REF_frequency**. If the difference is smaller than or equal to the value of **ACCEPT_TOLERANCE** the two frequency values are accepted as equal and can be further processed.

3.5.3 SAFE_INPUTS_OK (FB)

3867

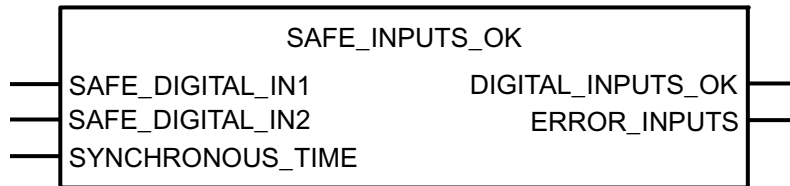
Contained in the library:

ifm_SafetyIO_Vxxyyzz.LIB

Available for the following devices:

- SafetyController: CR7nnn

Symbol in CoDeSys:



Description

3897

SAFE_INPUTS_OK monitors safety-related digital input signals.

In safety-related applications input signals have to be evaluated redundantly and as diversified as possible. This FB evaluates 2 digital inputs (if possible from different input groups) and checks if the signals have switched synchronously, i.e. within the stated SYNCHRONOUS_TIME.

Start behavior of the SYNCHRONOUS_TIME:

DIGITAL_INPUTS_OK	SYNCHRONOUS_TIME starts...
FALSE	... if both SAFE_DIGITAL_IN signals were switched off AND since one SAFE_DIGITAL_IN signal is switched on (edge FALSE ⇒ TRUE).
TRUE	... since one SAFE_DIGITAL_IN signal is switched off (edge TRUE ⇒ FALSE).

The result is also provided redundantly:

Comparison of inputs shows	DIGITAL_INPUTS_OK	ERROR_INPUTS	
If DIGITAL_INPUTS_OK = FALSE: Both SAFE_DIGITAL_IN signals are switched on within the SYNCHRONOUS_TIME.	FALSE ⇒ TRUE *)	FALSE	Green
If DIGITAL_INPUTS_OK = FALSE: One SAFE_DIGITAL_IN signal is switched on within the SYNCHRONOUS_TIME.	FALSE	FALSE ⇒ TRUE *)	Red
If DIGITAL_INPUTS_OK = TRUE: One SAFE_DIGITAL_IN signal is switched off within the SYNCHRONOUS_TIME.	TRUE ⇒ FALSE (at once)	FALSE ⇒ TRUE *)	Red
If DIGITAL_INPUTS_OK = TRUE: Both SAFE_DIGITAL_IN signals are switched off within the SYNCHRONOUS_TIME.	TRUE ⇒ FALSE (at once)	FALSE	Green
If DIGITAL_INPUTS_OK = FALSE AND the SYNCHRONOUS_TIME passed by: Both SAFE_DIGITAL_IN signals are switched off.	FALSE	FALSE	Green

*) switches not until after SYNCHRONOUS_TIME passed by.

→ Example: SAFE_INPUTS_OK (→ page [40](#))

WARNING

Risk for people if one of the operating elements on the SAFE_DIGITAL_IN inputs is mechanically stuck (e.g. tampering).

- ▶ The user has to ensure that the operating elements always function perfectly and cannot be tampered with.

SAFE_INPUTS_OK monitors e.g. switching signals of positively guided contacts in e-stops or the synchronous switching of two-hand controls.

Antivalent contact pairs (one normally closed and normally open contact each) can also be processed by inverting the input signals. Antivalent contact pairs additionally allow the detection of wiring errors such as cross fault.

If e-stop or two-hand control are not used regularly they need to be tested manually at defined intervals. This ensures that an error (e.g. in the wiring or the e-stop) does not remain undetected.

NOTE

Max. permissible delay times SYNCHRONOUS_TIME for typical 2-channel input signals:

e-stop	max. 100 ms
two-hand control (to CAT 3)	max. 500 ms

Parameters of the inputs

3900

Parameter	Data type	Description
SAFE_DIGITAL_IN1	BOOL	safety-related input signal 1
SAFE_DIGITAL_IN2	BOOL	safety-related input signal 2 (reference value)
SYNCHRONOUS_TIME	TIME	max. permissible delay time of the two input signals in relation to each other

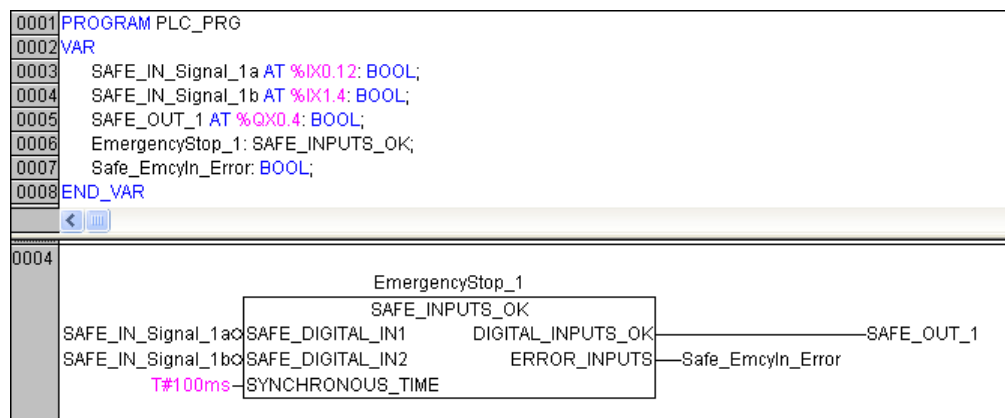
Parameters of the outputs

3901

Parameter	Data type	Description
DIGITAL_INPUTS_OK	BOOL	TRUE: signals are synchronous = within the SYNCHRONOUS_TIME FALSE: signals are not synchronous = out of the SYNCHRONOUS_TIME OR: no input signal (both = FALSE)
ERROR_INPUTS	BOOL	TRUE: signals are not synchronous = out of the SYNCHRONOUS_TIME FALSE: no error

Example: SAFE_INPUTS_OK

3898



In this example for a 2-channel e-stop 2 digital inputs from different inputs groups have to switch on simultaneously within 100 ms to switch on the output.

In the case of a complementary circuit (e.g. for a protective guard) the normally closed contact (e.g. signal_1b) has to be scanned in a negated way, the normally open contact (e.g. signal_1a) without negation.

3.5.4 SAFETY_SWITCH (FB)

3869

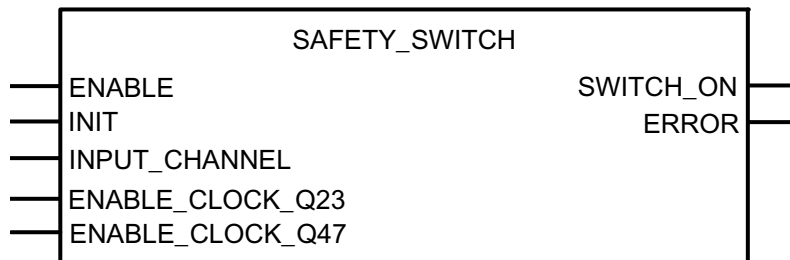
Contained in the library:

`ifm_CR7nnn_Vxxyyzz.LIB`

Available for the following devices:

- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Symbol in CoDeSys:



NOTE regarding CR7505/CR7506: Input `ENABLE_CLOCK_Q47` not available.

Description

3896

`SAFETY_SWITCH` evaluates the signals of inductive fail-safe sensors.

The FB has to be integrated once for each connected sensor and has to be initialised during program start once for one cycle.

The configuration of the FB (selected input channel and clock output) is taken over during initialisation and is thus firmly set. The configuration of the FB cannot be changed during the processing of the program.

The FB can be deactivated via input `ENABLE`.

- > If the SafetyController reads the generated clock signal with the correct chronological sequence at the configured output is output `SWITCH_ON` set to `TRUE` when the sensor is damped.
- > In the case of an error the `ERROR` output is set to `TRUE` and output `SWITCH_ON` is set to `FALSE` simultaneously.
- > The output `SWITCH_ON` becomes `TRUE` again as soon as an error-free sensor signal is received.

The result is provided redundantly:

Comparison of inputs shows	<code>SWITCH_ON</code>	<code>ERROR</code>
Received clock signal (<code>INPUT_CHANNEL</code>) corresponds to transmitted clock signal (<code>Q23/Q47</code>).	<code>TRUE</code>	<code>FALSE</code>
Received clock signal (<code>INPUT_CHANNEL</code>) does not correspond to transmitted clock signal (<code>Q23/Q47</code>).	<code>FALSE</code>	<code>TRUE</code>

The safe output signal `SWITCH_ON` can be evaluated in the application.

NOTICE

- > The result can vary in each PLC cycle, corresponding to the signal constellation.
- ▶ The application programmer has to evaluate a signalled error in the same cycle.
- ▶ In case of a fault, the application programmer has to bring the machine / installation into the safe state.

→ Example for `SAFETY_SWITCH` (→ page [43](#))

Parameters of the inputs

3905

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
INPUT_CHANNEL	BYTE	input channel for evaluating the fail-safe sensor: 0 = I24 / %IX1.4 1 = I25 / %IX1.5 2 = I26 / %IX1.6 3 = I27 / %IX1.7
ENABLE_CLOCK_Q23	BOOL	the associated clock signal for the fail-safe sensor comes from output Q23 / %QX0.7
ENABLE_CLOCK_Q47 *)	BOOL	the associated clock signal for the fail-safe sensor comes from output Q47 / %QX1.7 *)

*) not for CR7505, CR7506

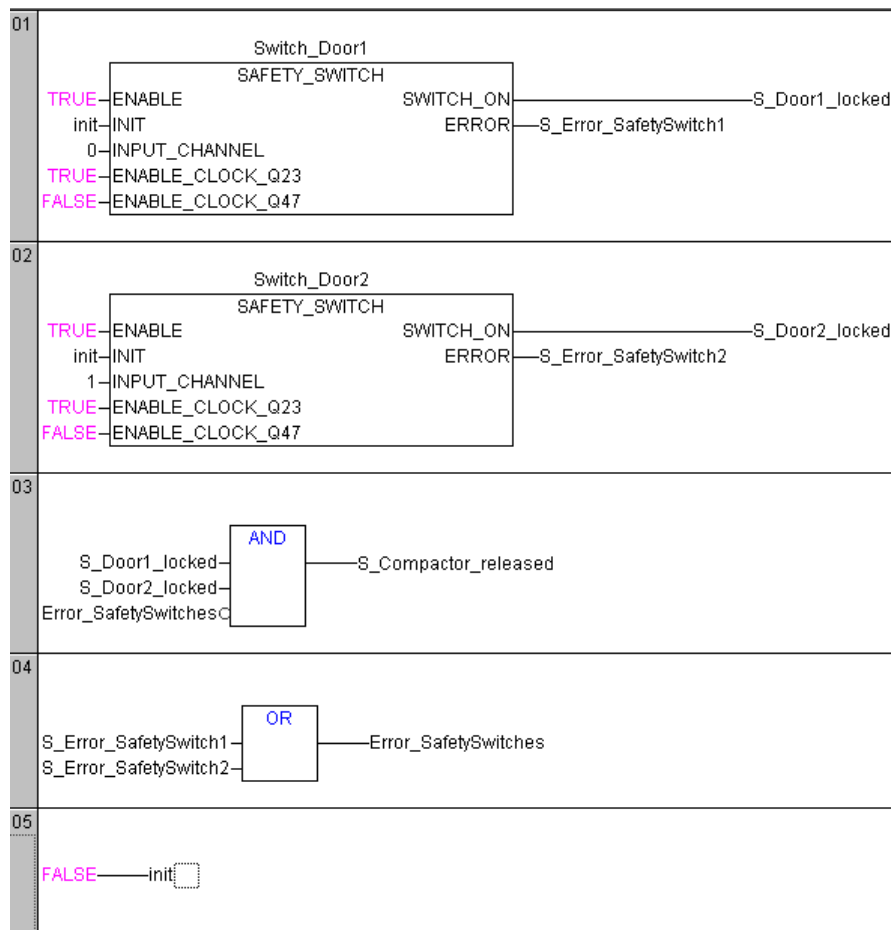
Parameters of the outputs

3906

Parameter	Data type	Description
SWITCH_ON	BOOL	TRUE: inductive sensor is damped and generates a safe input signal FALSE: sensor is not damped
ERROR	BOOL	TRUE: sensor signal is faulty OR: sensor is manipulated FALSE: no faulty sensor signal

Example: SAFETY_SWITCH

3902



In this example 2 protective guards on a waste compactor are monitored:

- Protective guard 1, connected to input channel 0 (= I24 / %IX1.4)
- Protective guard 2, connected to input channel 1 (= I25 / %IX1.5)

Only when both protective guards are closed safely does the controller release the waste compactor.

4 System description

Contents

Information concerning the device	44
Information concerning the software	46
PLC configuration.....	47
Monitoring concept.....	48

975

4.1 Information concerning the device

1324

This manual describes the **ecomatmobile** controller family of **ifm electronic gmbh** with a 16-bit microcontroller for mobile vehicles:

- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

4.1.1 Test basis for certification

4092

Testing and certification were carried out on the basis of the following standards and specifications:

ISO 13849-1 / 03.2007

Safety of machines - safety-related parts of control systems

Part 1: General design principles

4.1.2 Functions and features

3747

The freely programmable controllers of the "SafetyController **ecomatmobile**" series are rated for use under difficult conditions, e.g.:

- extended temperature range,
- strong vibration,
- intensive EMC interference.

The controllers are thus suited for direct installation in machines in mobile and robust applications. By their specification the inputs and outputs are specially rated for this use.

Integrated hardware and software functions (operating system) offer already high protection. In addition, special hardware and software functions are integrated in the certified controllers for safety-related applications to enable use as a safety controller.

The SafetyController is approved for safety-related tasks according to the protection of persons if the corresponding system check routines are integrated in the operating system and the application software.

Achievable safety class

4064

Depending on the use of the hardware or its external wiring (→ chapter Hardware structure (→ page [48](#))) and the structure of the application program (→ chapter Safety concept (→ page [55](#))), the following safety class can be achieved with the certified SafetyControllers:

Safety class	according to the standard
Performance level PL d	ISO 13849-1 (→ standard ISO 13849 (→ page 13))
Safety integrity level SIL CL 2	IEC 62061-1

4094

The final classification may only be effected upon a risk assessment of the application. Approval of hardware and software must be obtained from the corresponding supervisory organisations.

NOTE

Principally only certified operating systems can and may be used for safety-related applications.

The user is responsible for the reliable function of the application programs he designed. If necessary, he must obtain an approval from the corresponding supervisory and test organisations according to the national regulations.

4.2 Information concerning the software

2730

The device operates with CoDeSys, version 2.3.9.1 or higher.

In the "programming manual CoDeSys 2.3" you will find more details about how to use the programming system "CoDeSys for Automation Alliance". This manual can be downloaded free of charge from ifm's website at:

→ www.ifm.com > select your country > [Service] > [Download] > [Control systems]

→ ifm-CD "Software, tools and documentation"

The application software conforming to IEC 61131-3 can be easily designed by the user with the programming system CoDeSys (→ www.3s-software.com). Before using this software on the PC please note the following minimal system requirements:

- CPU Pentium II, 500 MHz
- Memory (RAM) 128 MB, recommended: 256 MB
- Free hard disc required (HD) 100 MB
- Runtime system platform Windows 2000 or higher
NOTE: Not yet released for the 64-bit platforms of Windows Vista and Windows 7
- CD ROM drive

Moreover the user must take into account which software version is used (in particular for the operating system and the function libraries).

NOTE

The software versions suitable for the selected target must always be used:

- operating system (CRnnnn_Vxxyzz.H86 / CRnnnn_Vxxyzz.HEX)
- PLC configuration (CRnnnn_Vxx.CFG)
- device library (ifm_CRnnnn_Vxxyzz.LIB)
- and the further files (→ chapter Overview of the files and libraries used (→ page [376](#)))

CRnnnn	device article number
Vxx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "CR0032") and the software version number "xx" (e.g. "02") must always have the same value! Otherwise the device goes to the STOP mode.

The values for "yy" (release number) and "zz" (patch number) do **not** have to match.

IMPORTANT: the following files must also be loaded:

- the internal libraries (created in IEC 1131) required for the project,
- the configuration files (*.CFG)
- and the target files (*.TRG).

Important: the following devices must have at least the here listed targets:

Device	Target at least version ...
BasicController: CR040n	V01
BasicDisplay: CR0451	V01
CabinetController: CR030n	V05
ClassicController: CR0020, CR0505	V05
ClassicController: CR0032	V02
ExtendedController: CR0200	V05
ExtendedController: CR0232	V01
SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506	V05
SafetyController: CR7032, CR7232	V01
SafetyController: CR7nnn	V05

WARNING

The user is responsible for the reliable function of the application programs he designed. If necessary, he must additionally carry out an approval test by corresponding supervisory and test organisations according to the national regulations.

4.3 PLC configuration

1797

The control system **ecomatmobile** is a device concept for series use. This means that the devices can be configured in an optimum manner for the applications. If necessary, special functions and hardware solutions can be implemented. In addition, the current version of the **ecomatmobile** software can be downloaded from our website at: www.ifm.com.

→ Setup the target (→ page [62](#))

Before using the devices it must be checked whether certain functions, hardware options, inputs and outputs described in the documentation are available in the hardware.

991

3772

4.4.2 Operating principle of the delayed switch-off

Contents

Connect terminal VBBS (23) to the ignition switch	49
Connect terminal VBBO (5) to battery (not switched)	49
Latching	49

993

If the **ecomatmobile** controllers are disconnected from the supply voltage (ignition off), all outputs are normally switched off at once, input signals are no longer read and processing of the controller software (operating system and application program) is interrupted. This happens irrespective of the current program step of the controller.

If this is not requested, the controller must be switched off via the program. After switch-off of the ignition this enables, for example, saving of memory states.

The ClassicControllers can be switched off via the program by means of a corresponding connection of the supply voltage inputs and the evaluation of the related system flags. The block diagram in the chapter Hardware set-up (→ page 48) shows the context of the individual current paths.

Connect terminal VBBS (23) to the ignition switch

994

Via terminal 23 the controller is supplied and can be switched off by an ignition switch.

In automotive engineering the potential is called "clamp 15".

This terminal is monitored internally. If no supply voltage is applied, the system flag CLAMP_15 is set to FALSE. The reset of the flag CLAMP_15 can be monitored by the application program.

Connect terminal VBBO (5) to battery (not switched)

995

Up to 12 outputs of the output group VBB_o can be supplied via terminal 5. At the same time latching of the control electronics is supplied via this terminal.

Latching

996

Latching is active if voltage is applied to VBB_o **and** the system flag RELAY_CLAMP_15 (and so the relay [Clamp]) is set.

If the system flag RELAY_CLAMP_15 is reset, the relay [Clamp] is de-energised. If at this moment no voltage is applied to terminal 23, latching is removed and the controller switches off completely.

4.4.3 Operating principle of the monitoring concept

Contents

Monitoring of the supply voltage VBBs	51
Monitoring and securing mechanisms.....	52

997

During program processing the monitoring relay is completely controlled via the software by the user. So a parallel contact of the safety chain, for example, can be evaluated as an input signal and the monitoring relay can be switched off accordingly. To be on the safe side, the corresponding applicable national regulations must be complied with.

If an error occurs during program processing, the relay can be switched off using the system flag bit ERROR to disconnect critical plant sections.

By resetting the system flag bit RELAIS (via the system flag bit ERROR or directly) all outputs are switched off. The outputs in the current path VBB_R are disconnected directly by means of the monitoring relay. So the outputs in the current path VBB_O are only disconnected via the software.

WARNING

Danger due to unintentional and dangerous start of machine or plant sections!

- When creating the program, the programmer must ensure that no unintentional and dangerous start of machines or plant sections after a fault (e.g. e-stop) and the following fault elimination can occur.
- To do so, the required outputs must be additionally switched off and the logic states must be linked to the relay state and evaluated.

If an output to be monitored is continuously switched and the contact of the monitoring relay is stuck, the corresponding output cannot be switched off!

NOTE

If a watchdog error occurs, the program processing is interrupted automatically and the controller is reset. The controller then starts again as after power on.

Monitoring of the supply voltage VBBs

6752

Available for the following devices:

- SafetyController: CR7021, CR7201, CR7506

The undervoltage detection is not carried out on the terminal voltage VBBx but directly on the supply voltage VBB_s.

In case of a fault we differentiate 2 scenarios:

1) The terminal voltage VBBx falls below the limit value of 8 V:

- > As long as the supply voltage VBB_s remains in the accepted range of > 8 V the controller can be operated.
- By a corresponding design of the application software (customer's responsibility!) the system can be configured so that the application continues to operate when the terminal voltage VBBx recovers and is in the regular range again.

2) The supply voltage VBBs falls below the limit value of 8 V:

- > The controller detects undervoltage. The CPU stops the watchdog.
- > After the watchdog time (→ page [109](#)) has elapsed, the internal controllers are reset.
- > A restart of the controller is not carried out before the supply voltages are above the limit value again.

Monitoring and securing mechanisms

Contents

After application of the supply voltage	52
If runtime system / application is running	52
If the TEST pin is not active	53
One-time mechanisms	53

3926

For the ClassicController family the following monitoring activities are automatically carried out:

After application of the supply voltage

3927

After application of the supply voltage (controller is in the boot loader) the following tests are carried out in the device:

- > RAM test (one-time)
- > Supply voltage > 10 V DC
- > System data consistency
- > CRC of the boot loader
- > CRC of the runtime system
- > CRC of the application
- > Memory error:
 - If the test is running: flag ERROR_MEMORY = TRUE
(can be evaluated as from the first cycle).
 - If the test is not running: red LED is lit.

If runtime system / application is running

3928

then the following tests are cyclically carried out:

- > Triggering of the watchdog (100 ms)
Then continuous program check watchdog
- > Continuous temperature check
In case of a fault: system flag ERROR_TEMPERATURE = TRUE
- > Continuous voltage monitoring
In case of a fault: system flag ERROR_POWER = TRUE or ERROR_VBBR = TRUE
- > Continuous CAN bus monitoring
- > Continuous system data monitoring:
 - program loaded
 - operating mode RUN / STOP,
 - runtime system loaded,
 - node ID,
 - baud rate of CAN and RS232.
- > In the operating mode RUN:

Cyclical I/O diagnosis:

 - short circuit,
 - wire break,
 - overload (current) of the inputs and outputs,
 - cross fault (only for SafetyController).

Only for SafetyController:

- > Monitoring of the PIC controller (PIC = coprocessor of the controller):
 - PIC available and active,
 - quartz clock signal
- > RELAY test
- > Monitoring of the memory
- > Controller command check

In case of a fault:

- > red LED is lit,
- > Controller passes to the STOP mode.

If the TEST pin is not active

3929

- > Write protection for system data in FRAM, e.g.:
 - runtime system loaded,
 - calibration data.
 Implemented via hardware and software.
- > Write protection for application program (in the flash memory)
- > DEBUG mode

One-time mechanisms

3930

- > CRC monitoring during download or upload.
- > It must be checked that the runtime system and the application are assigned to the same device.

Safety-related processing of the memory areas

3932

For the downloader from version V05.10.01 onwards, the memory areas for retain data, user flash, data flash as well as EEPROM or FRAM data are monitored as follows:

Upload without CRC	Upload with CRC
If the downloader detects a safety controller CR7nnn during the login, a warning is displayed. The checksum of the last 2 bytes of the memory is ignored. A checksum is appended to the end of the H86 file.	It is expected that the last 2 bytes of the memory area contain a checksum. If this checksum is not correct (or missing), the upload is aborted and no file is created. A checksum is appended to the end of the H86 file.
Download without CRC	Download with CRC
If the downloader detects a safety controller CR7nnn during the login, a warning is displayed. The checksum at the end of the H86 file is ignored but nevertheless transmitted. If the checksum at the end of the H86 file was wrong, the checksum in the last 2 bytes of the memory is also wrong after the download.	It is expected that the last 2 bytes of the H86 file contain a checksum. If this checksum is not correct, no download is carried out. After the download the checksum is checked again in the controller. If this checksum was not correct, the downloader generates an error message (not for the EEPROM area).

4.4.4 Feedback in case of externally supplied outputs

2422

In some applications actuators are not only controlled by outputs of the PLC but additionally by external switches. In such cases the externally supplied outputs must be protected with blocking diodes (→ see graphics below).

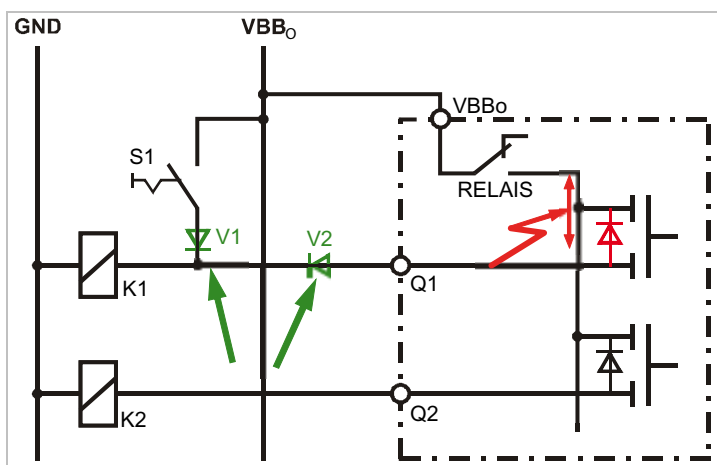
ATTENTION

Destruction of outputs if there is inadmissible feedback!

If actuators are externally controlled, the corresponding output bar must not become potential-free (e.g. for RELAIS = FALSE).

Otherwise the terminal voltage VBB_x is fed back to the potential bar of the output group via the protective diode integrated in the output driver. A possibly set output thus triggers its connected load. The load current destroys the output which feeds back.

► Protect externally supplied outputs by means of blocking diodes!



Example:

Without the blocking diodes V1+V2 an external switch S1 VBB_0 feeds from the output Q1 to the internal potential bar of the outputs via the internal protective diode (red).

If output Q2 = TRUE, K2 is supplied with voltage from Q1 via the protective diode despite RELAIS = FALSE. Due to overload this protective diode burns out and output Q1 is destroyed!

► Insert the blocking diodes V1+V2 (→ green arrows)!

Graphics: Example wiring with blocking diodes due to the danger of feedback

NOTE

Help for externally supplied outputs

► The externally supplied outputs must be decoupled via diodes so that no external voltage is applied to the output terminal.

4.4.5 Safety concept

Contents

System test.....	55
Software structure	55
Operating system	56
Application program	56
Maximum program cycle time	56

3776

The following chapters describe the safety concept of the hardware and its application in safety-related applications. If the inputs and outputs are correspondingly selected and wired, certified SafetyControllers can be used in applications up to PL d.

WARNING

Warning of loss of the safety category!

Principally a second switch-off mode must be available for applications to PL c (and higher) if the dangerous failure is not signalled in due time (warning, alarm, display, etc.). For this purpose an additional relay is available in the SafetyController. Only those outputs switched off via this monitoring relay and having extended diagnosis possibilities can be identified in the configuration overviews by the marking "Safe outputs" and the reference to the relay contact. → data sheet!

The analysis of the safety system must show whether a safety-related output must be designed as redundant or whether monitoring and testing as described above are sufficient.

Further, the analysis must show whether switch-off via the internal relay is sufficient in case of a fault or whether a second output (electrical or hydraulic) must be used for the redundant switch-off.

If e.g. a cable harness to an external valve does not contain a supply line or if a short-circuit to GND is harmless in terms of safety, it is sufficient to switch off the output via the internal relay in case of a fault.

System test

3779

All software parts in the controller are monitored to the extent possible by the operating system and the additional internal processor. This allows to detect and react to errors such as exceeded runtimes in case of incorrect processing of the program.

When switching on the controller, all hardware and software parts are tested. These internal tests and monitoring activities are periodically repeated. For these tests the occurrence time of the first error of 30 s is adhered to. So all functional parts of the controller are checked independently of the user program.

Software structure

3780

The software in the controller consists of the operating system and the application software. By means of checksums these parts are cyclically checked for correctness, individually and as a whole. The checksums are automatically generated and appended to the software parts.

Operating system

3781

The user receives the operating system together with the programming system. He must (normally) load the operating system only once in the controller.

The number of the operating system and the hardware must correspond, e.g.:
CR7020_V050100.H86 for CR7020.

Application program

3782

The application program is created on site. The structure must comply with the required safety class. It may only be loaded in the controller after the operating system has been loaded.

- When creating the application program observe version consistency of operating system (*.H86), PLC configuration (*.CFG) and libraries (*.LIB)!

Maximum program cycle time

3783

The maximum program cycle time of an application program must not exceed 100 ms. Longer times can result in triggering the watchdog and thus causing a fatal error (> red LED is lit).

Typically the cycle time should not be longer than 50 ms to ensure sufficient excess gain.

5 Operating states and operating system

Contents

Operating states	57
Status LED	58
Load the operating system.....	59
Operating modes.....	60

1074

5.1 Operating states

1075

After power on the **ecomatmobile** controller can be in one of five possible operating states:

5.1.1 INIT state (Reset)

1076

This state is passed through after every power on reset:

- > The operating system is initialised.
- > Various checks are carried out, e.g. waiting for correctly power supply voltage.
- > This temporary state is replaced by the RUN or STOP state.
- > The LED lights yellow.

Change out of this state possible into one of the following states:

- RUN
- FATAL ERROR
- STOP

5.1.2 STOP state

1078

This state is reached in the following cases:

- From the INIT state if no program is loaded
- From the RUN state if:
 - the STOP command is sent via the interface
 - AND: operating mode = Test (→ chapter TEST mode (→ page [60](#)))

5.1.3 Fatal error

1079

The **ecomatmobile** controller goes to this state if a non tolerable error was found. This state can only be left by a reset.

- > The LED lights red.

5.1.4 RUN state

1077

This state is reached in the following cases:

- From the INIT state (autostart)
- From the STOP state by the RUN command
 - only for the operating mode = Test (→ chapter TEST mode (→ page 60))

5.1.5 No operating system

1080

No operating system was loaded, the controller is in the boot loading state. Before loading the application software the operating system must be downloaded.

> The LED flashes green (quickly).

5.2 Status LED

1430

The operating states are indicated by the integrated status LED (default setting).

LED colour	Flashing frequency	Description
off	permanently out	no operating voltage
green	5 Hz	no operating system loaded
green	2 Hz	RUN state
green	permanently on	STOP state
red	2 Hz	RUN state with error
red	permanently on	fatal error
yellow/orange	briefly on	initialisation or reset checks

The operating states STOP and RUN can be changed by the programming system.

For this controller the status LED can also be set by the application program. To do so, the following system variables are used:

LED	LED colour for "active" (= on)
LED_X	LED colour for "pause" (= out)
LED_COLOR	colour constant from the data structure "LED colour" allowed: LED_GREEN, LED_BLUE, LED_RED, LED_WHITE, LED_MAGENTA, LED_CYAN, LED_YELLOW, LED_BLACK (= LED out)
LED_MODE	flashing frequency from the data structure "LED_MODES" allowed: LED_2HZ, LED_1HZ, LED_05HZ, LED_0HZ (permanently)

NOTE

In case of an error the LED colour RED is set by the operating system. Therefore this colour should not be used by the application.

If the colours and/or flashing modes are changed by the application program, the above-mentioned table (default setting) is no longer valid.

5.3 Load the operating system

2733

On delivery of the **ecomatmobile** controller no operating system is normally loaded (LED flashes green at 5 Hz). Only the boot loader is active in this operating mode. It provides the minimum functions for loading the operating system (e.g. RS232, CAN).

Normally it is necessary to download the operating system only once. The application program can then be loaded to the controller (also several times) without influencing the operating system.

Advantage:

- No EPROM replacement is necessary for an update of the operating system.

The operating system is provided with this documentation on a separate data carrier. In addition, the current version can be downloaded from the website of **ifm electronic gmbh** at:

→ www.ifm.com > select your country > [Service] > [Download] > [Control systems]

NOTE

The software versions suitable for the selected target must always be used:

- operating system (CRnnnn_Vxxyzz.H86 / CRnnnn_Vxxyzz.HEX)
- PLC configuration (CRnnnn_Vxx.CFG)
- device library (ifm_CRnnnn_Vxxyzz.LIB)
- and the further files (→ chapter Overview of the files and libraries used (→ page [376](#)))

CRnnnn	device article number
Vxx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "CR0032") and the software version number "xx" (e.g. "02") must always have the same value! Otherwise the device goes to the STOP mode.

The values for "yy" (release number) and "zz" (patch number) do **not** have to match.

IMPORTANT: the following files must also be loaded:

- the internal libraries (created in IEC 1131) required for the project,
- the configuration files (*.CFG)
- and the target files (*.TRG).

The operating system is transferred to the device using the separate program "downloader". (The downloader is on the **ecomatmobile** CD "Software, Tools and Documentation" or can be downloaded from **ifm's** website, if necessary).

Normally the application program is loaded to the device via the programming system. But it can also be loaded using the downloader if it was first read from the device (→ upload).

5.4 Operating modes

1083

Independent of the operating states the **ecomatmobile** controller can be operated in different modes. The corresponding control bits can be set and reset with the programming software CoDeSys (window: Global Variables) via the application software or in test mode (→ chapter TEST mode (→ page [60](#))).

5.4.1 TEST mode

1084

This operating mode is reached by applying a high level (supply voltage) to the test input (→ installation instructions, chapter "wiring"). The **ecomatmobile** controller can now receive commands via one of the interfaces in the RUN or STOP mode and, for example, communicate with the programming system. Moreover the software can only be downloaded to the controller in this operating state.

The state of the application program can be queried via the flag TEST.

NOTICE

Loss of the stored software possible!

In the test mode there is no protection of the stored operating system and application software.

5.4.2 SERIAL_MODE

1085

The serial interface is available for the exchange of data in the application. Debugging the application software is then only possible via the CAN interface.

For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

This function is switched off as standard (FALSE). Via the flag SERIAL_MODE the state can be controlled and queried via the application program or the programming system.

→ chapter Use of the serial interface (→ page [327](#))

5.4.3 DEBUG mode

1086

If the input DEBUG of SET_DEBUG (→ page [352](#)) is set to TRUE, the programming system or the downloader, for example, can communicate with the controller and execute system commands (e.g. for service functions via the GSM modem CANremote).

In this operating mode a software download is not possible because the test input (→ chapter (→ page [60](#))) is not connected to supply voltage.

6 Configurations

Contents

Set up programming system	61
Function configuration of the inputs and outputs	78
Hints to wiring diagrams	101
Operating modes of the ExtendedSafetyController	103

1016

The device configurations described in the corresponding installation instructions and in the annex (→ page 368) to this documentation are used for standard devices (stock items). They fulfil the requested specifications of most applications.

Depending on the customer requirements for series use it is, however, also possible to use other device configurations, e.g. with respect to the inputs/outputs and analogue channels.

WARNING

Property damage or bodily injury possible due to malfunctions!

The software functions described in this documentation only apply to the standard configurations. In case of use of customer-specific devices:

- ▶ Note the special hardware versions and additional remarks (additional documentation) on use of the software.

Installation of the files and libraries in the device:

Factory setting: the device contains only the boot loader.

- ▶ Load the operating system (*.H86 or *.HEX)
- ▶ Create the project (*.PRO) in the PC: enter the target (*.TRG)
- ▶ Additionally depending on device and target:
Define the PLC configuration (*.CFG)
- > CoDeSys integrates the files belonging to the target into the project:
*.TRG, *.CFG, *.CHM, *.INI, *.LIB
- ▶ If required, add further libraries to the project (*.LIB).

Certain libraries automatically integrate further libraries into the project.

Some FBs in ifm libraries (ifm_*.LIB) e.g. are based on FBs in CoDeSys libraries (3S_*.LIB).

6.1 Set up programming system

Contents

Set up programming system manually	61
Set up programming system via templates	65
ifm demo programs	75

3968

6.1.1 Set up programming system manually

Contents

Setup the target	62
Activating the PLC configuration	63

3963

Setup the target

2687

When creating a new project in CoDeSys® the target file corresponding to the controller must be loaded. It is selected in the dialogue window for all hardware and acts as an interface to the hardware for the programming system.

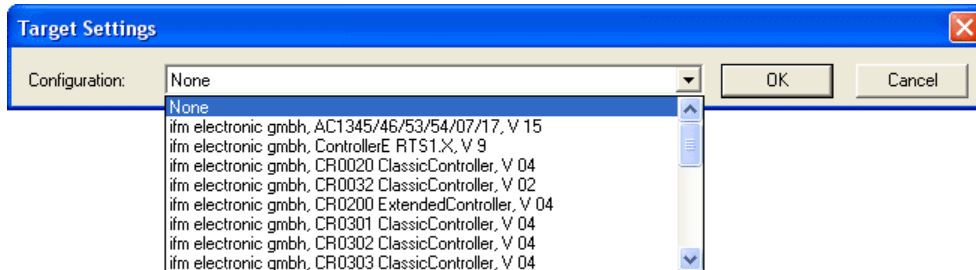


Figure: Target system settings

At the same time, all important libraries and the PLC configuration are loaded when selecting the target. These can be removed by the programmer or complemented by further libraries, if necessary.

NOTE

The software versions suitable for the selected target must always be used:

- operating system (CRnnnn_Vxxyzz.H86 / CRnnnn_Vxxyzz.HEX)
- PLC configuration (CRnnnn_Vxx.CFG)
- device library (ifm_CRnnnn_Vxxyzz.LIB)
- and the further files (→ chapter Overview of the files and libraries used (→ page [376](#)))

CRnnnn	device article number
Vxx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "CR0032") and the software version number "xx" (e.g. "02") must always have the same value! Otherwise the device goes to the STOP mode.

The values for "yy" (release number) and "zz" (patch number) do **not** have to match.

IMPORTANT: the following files must also be loaded:

- the internal libraries (created in IEC 1131) required for the project,
- the configuration files (*.CFG)
- and the target files (*.TRG).

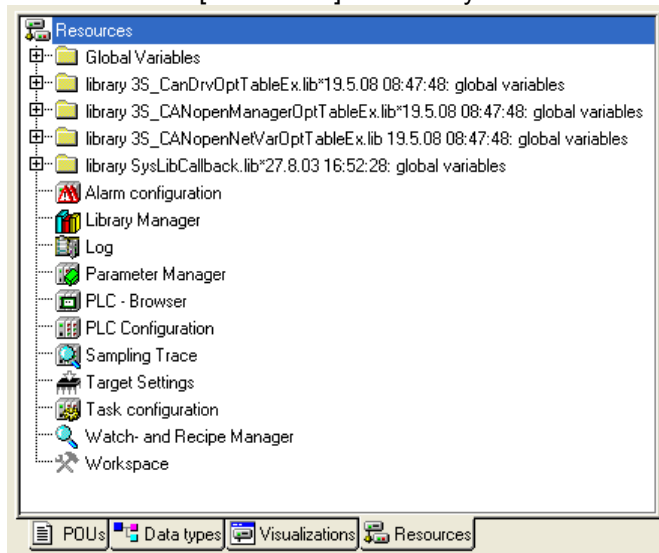
Activating the PLC configuration

2688

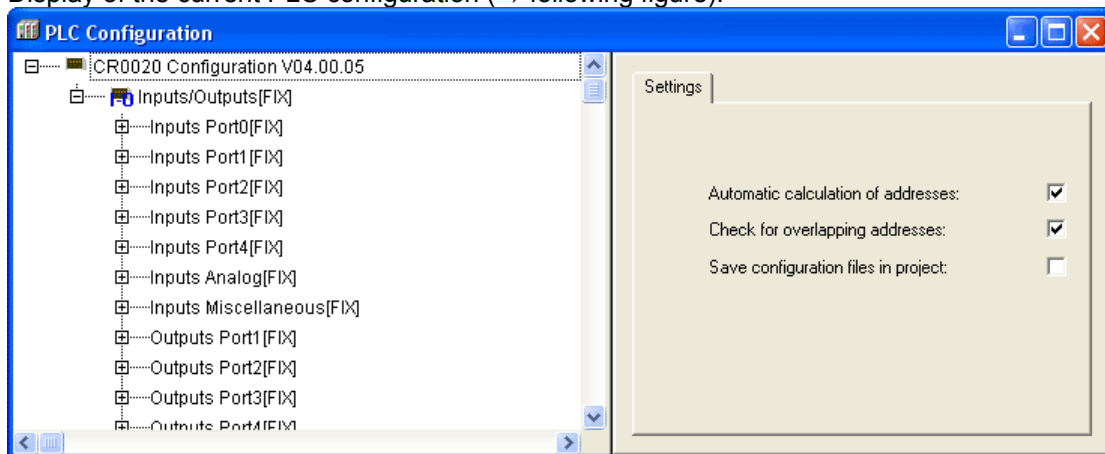
During the configuration of the programming system (→ previous section) automatically also the PLC configuration was carried out.

The point [PLC Configuration] is reached via the tab [Resources]. Double-click on [PLC Configuration] to open the corresponding window.

- Click on the tab [Resources] in CoDeSys:

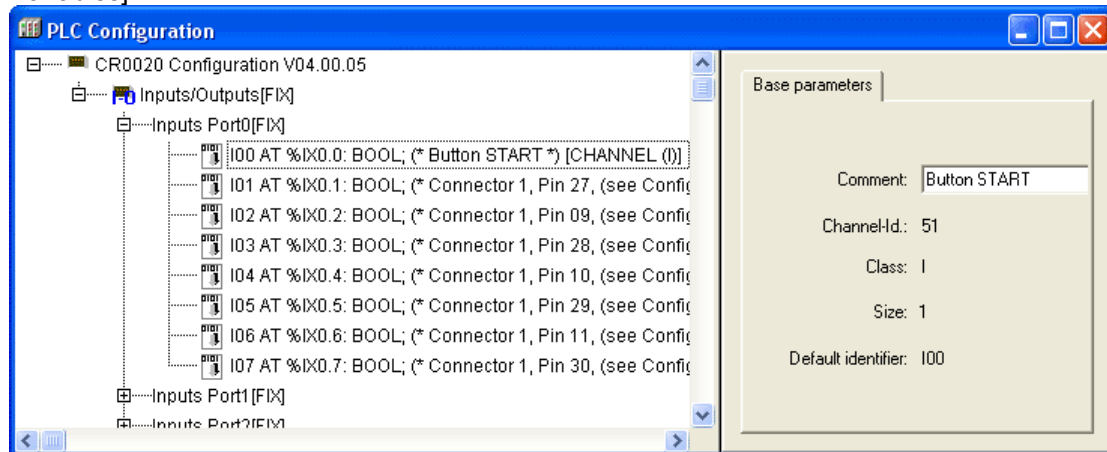


- Double-click on [PLC Configuration] in the left column.
- > Display of the current PLC configuration (→ following figure):



Based on the configuration the following is available in the program environment for the user:

- All important system and error flags
Depending on the application and the application program, these flags must be processed and evaluated. Access is made via their symbolic names.
- The structure of the inputs and outputs
These can be directly symbolically designated (highly recommended!) in the window [PLC Configuration] (example → figure below) and are available in the whole project as [Global Variables].



6.1.2 Set up programming system via templates

Contents

About the ifm templates	68
Supplement project with further functions	72

3977

ifm offers ready-to-use templates (program templates) for a fast, simple, and complete setting up of the programming system.

NOTE

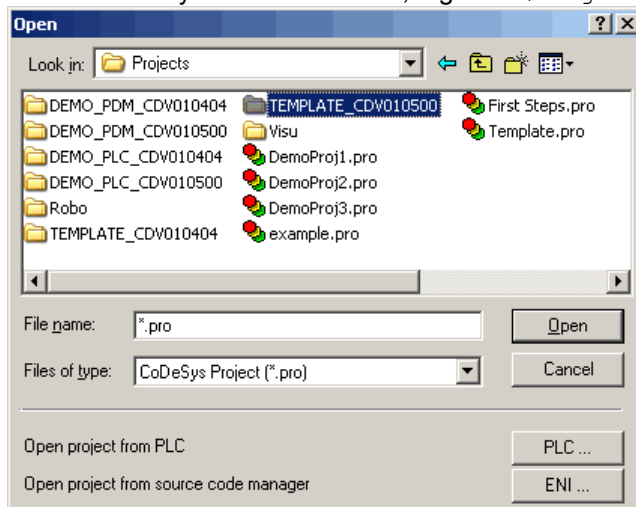
When installing the **ecomatmobile** CD "Software, Tools and Documentation", projects with templates have been stored in the program directory of your PC:

...\\ifm electronic\\CoDeSys V...\\Projects\\Template_CDVxyyyzz

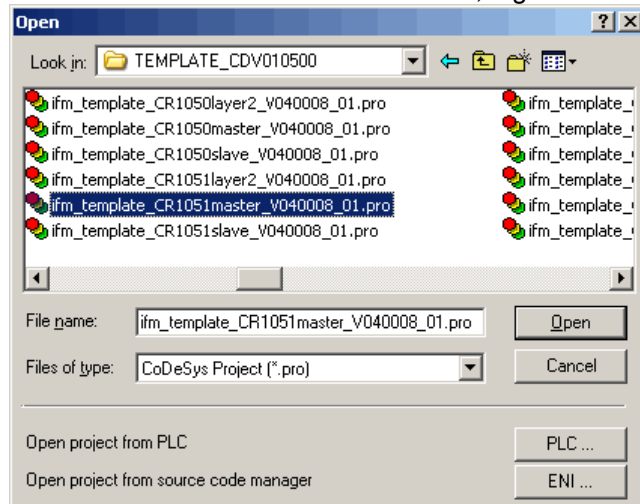
- ▶ Open the requested template in CoDeSys via:
[File] > [New from template...]
- > CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page [65](#))

How do you set up the programming system fast and simply?

- ▶ In the CoDeSys menu select: [File] > [New from template...]
- ▶ Select directory of the current CD, e.g. ...\\Projects\\TEMPLATE_CDV010500:



- Find article number of the unit in the list, e.g. CR2500 as CANopen master:



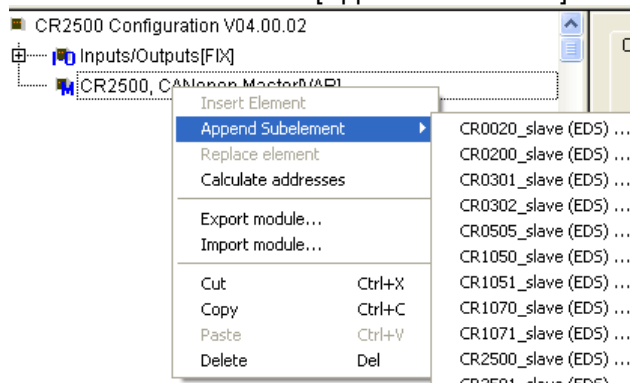
- How is the CAN network organised?
Do you want to work on layer 2 basis or is there a master with several slaves (for CANopen)?
(Here an example: CANopen-Slave, → figure above)
- Confirm the selection with [Open].
- > A new CoDeSys project is generated with the following folder structure (left):

Example for CR2500 as CANopen master:	Another example for CR1051 as CANopen slave:

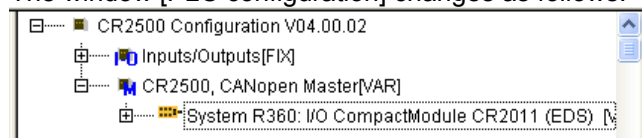
(via the folder structures in Templates → Section About the ifm Templates (→ page 68)).

- Save the new project with [file] > [Save as...], and define suitable directory and project name.
- Configuration of the CAN network in the project:
Double click the element [PLC configuration] above the tabulator [resources] in the CoDeSys project.

- ▶ **Right mouse click** in the entry [CR2500, CANopen Master]
- ▶ Click in the context menu [Append subelement]:



- > A list of all available EDS files appears in the extended context menu.
- ▶ Select requested element, e.g. "System R360: I/O CompactModule CR2011 (EDS)".
The EDS files are in directory C:\...\CoDeSys V...\Library\PLCConf\.
- > The window [PLC configuration] changes as follows:



- ▶ Set CAN parameters, PDO mapping and SDOs for the entered slave according to the requirements. Note: Better deselect [Create all SDOs].
- ▶ With further slaves proceed as described above.
- ▶ Save the project!

This should be a sufficient description of your project. You want to supplement this project with further elements and functions?

→ chapter Supplement project with further functions (→ page [72](#))

About the ifm templates

Contents

Folder structure in general	68
Programs and functions in the folders of the templates	69
Structure of the visualisations in the templates	71

3981

As a rule the following templates are offered for each unit:

- `ifm_template_CRnnnnLayer2_Vxyyyzz.pro` for the operation of the unit with CAN layer 2
- `ifm_template_CRnnnnMaster_Vxyyyzz.pro` for the operation of the unit as CAN master
- `ifm_template_CRnnnnSlave_Vxyyyzz.pro` for the operation of the unit as CAN slave

The templates described here are for:

- CoDeSys from version 2.3.9.6
- on the **ecomatmobile**-CD from version 010500

The templates all have the same structures.

The selection of this program template for CAN operation already is an important basis for a functioning program.

Folder structure in general

3978

The POU's are sorted in the following folders:

Folder	Description
CAN_OPEN	for Controller and PDM, CAN operation as master or slave: contains the FBs for CANopen.
I_O_CONFIGURATION	for Controller, CAN operation with layer 2 or as master or slave: FBs for parameter setting of the operating modes of the inputs and outputs.
PDM_COM_LAYER2	for Controller, CAN operation as layer 2 or as slave: FBs for basis communication via layer 2 between PLC and PDM.
CONTROL_CR10nn	for PDM, CAN operation with layer 2 or as master or slave: Contains FBs for image and key control during operation.
PDM_DISPLAY_SETTINGS	for PDM, CAN operation with layer 2 or as master or slave: Contains FBs for adjusting the monitor.

Programs and functions in the folders of the templates

3980

The above folders contain the following programs and function blocks (all = POU's):

POUs in the folder CAN_OPEN	Description
CANopen	for Controller and PDM, CAN operation as master: Contains the following parameterised POU's: - CAN1_MASTER_EMCI_HANDLER (→ CANx_MASTER_EMCI_HANDLER (→ page 205)), - CAN1_MASTER_STATUS (→ CANx_MASTER_STATUS (→ page 210)), - SELECT_NODESTATE (→ down).
CANopen	for Controller and PDM, CAN operation as slave: Contains the following parameterised POU's: - CAN1_SLAVE_EMCI_HANDLER (→ CANx_SLAVE_EMCI_HANDLER (→ page 218)), - CAN1_SLAVE_STATUS (→ CANx_SLAVE_STATUS (→ page 223)), - SELECT_NODESTATE (→ down).
Objekt1xxxh	for Controller and PDM, CAN operation as slave: Contains the values [STRING] for the following parameters: - ManufacturerDeviceName, e.g.: 'CR1051' - ManufacturerHardwareVersion, e.g.: 'HW_Ver 1.0' - ManufacturerSoftwareVersion, e.g.: 'SW_Ver 1.0'
SELECT_NODESTATE	for PDM, CAN operation as master or slave: Converts the value of the node status [BYTE] into the corresponding text [STRING]: 4 → 'STOPPED' 5 → 'OPERATIONAL' 127 → 'PRE-OPERATIONAL'
POUs in the folder I_O_CONFIGURATION	Description
CONF_IO_CRnnnn	for Controller, CAN operation with layer 2 or as master or slave: Parameterises the operating modes of the inputs and outputs.
POUs in the folder PDM_COM_LAYER2	Description
PLC_TO_PDM	for Controller, CAN operation with layer 2 or as slave: Organises the communication from the Controller to the PDM: - monitors the transmission time, - transmits control data for image change, input values etc.
TO_PDM	for Controller, CAN operation with layer 2 or as slave: Organises the signals for LEDs and keys between Controller and PDM. Contains the following parameterised POU's: - PACK (→ 3S), - PLC_TO_PDM (→ up), - UNPACK (→ 3S).

POUs in the folder CONTROL_CR10nn	Description
CONTROL_PDM	<p>for PDM, CAN operation with layer 2 or as master or slave:</p> <p>Organises the image control in the PDM.</p> <p>Contains the following parameterised POU:</p> <ul style="list-style-type: none"> - PACK (→ 3S), - PDM_MAIN_MAPPER (→ PDM_MAIN_MAPPER), - PDM_PAGECONTROL (→ PDM_PAGECONTROL), - PDM_TO_PLC (→ down), - SELECT_PAGE (→ down).
PDM_TO_PLC	<p>for PDM, CAN operation with layer 2:</p> <p>Organises the communication from the PDM to the Controller:</p> <ul style="list-style-type: none"> - monitors the transmission time, - transmits control data for image change, input values etc. <p>Contains the following parameterised POU:</p> <ul style="list-style-type: none"> - CAN_1_TRANSMIT (→ CAN_x_TRANSMIT), - CAN_1_RECEIVE (→ CAN_x_RECEIVE).
RT_SOFT_KEYS	<p>for PDM, CAN operation with layer 2 or as master or slave:</p> <p>Provides the rising edges of the (virtual) key signals in the PDM. As many variables as desired (as virtual keys) can be mapped on the global variable SoftKeyGlobal when e.g. a program part is to be copied from a CR1050 to a CR1055. It contains only the keys F1...F3:</p> <p>→ For the virtual keys F4...F6 variables have to be created. Map these self-created variables on the global softkeys. Work only with the global softkeys in the program. Advantage: Adaptations are only required in one place.</p>
SELECT_PAGE	<p>for PDM, CAN operation with layer 2 or as master or slave:</p> <p>Organises the selection of the visualisations.</p> <p>Contains the following parameterised POU:</p> <ul style="list-style-type: none"> - RT_SOFT_KEYS (→ up).
POUs in the folder PDM_DISPLAY_SETTINGS	Description
CHANGE_BRIGHTNESS	<p>for PDM, CAN operation with layer 2 or as master or slave:</p> <p>Organises brightness / contrast of the monitor.</p>
DISPLAY_SETTINGS	<p>for PDM, CAN operation with layer 2 or as master or slave:</p> <p>Sets the real-time clock, controls brightness / contrast of the monitor, shows the software version.</p> <p>Contains the following parameterised POU:</p> <ul style="list-style-type: none"> - CHANGE_BRIGHTNESS (→ up), - CurTimeEx (→ 3S), - PDM_SET_RTC (→ PDM_SET_RTC), - READ_SOFTWARE_VERS (→ down), (→ 3S).
READ_SOFTWARE_VERS	<p>for PDM, CAN operation with layer 2 or as master or slave:</p> <p>Shows the software version.</p> <p>Contains the following parameterised POU:</p> <ul style="list-style-type: none"> - DEVICE_KERNEL_VERSION1 (→ DEVICE_KERNEL_VERSION1), - DEVICE_RUNTIME_VERSION (→ DEVICE_RUNTIME_VERSION), - LEFT (→ 3S).

POUs in the root directory	Description
PLC_CYCLE	for Controller, CAN operation with layer 2 or as master or slave: Determines the cycle time of the PLC in the unit.
PDM_CYCLE_MS	for PDM, CAN operation with layer 2 or as master or slave: Determines the cycle time of the PLC in the unit.
PLC_PRG	for Controller and PDM, CAN operation with layer 2 or as master or slave: Main program This is where further program elements are included.

Structure of the visualisations in the templates

3979

Available for the following devices:

- BasicDisplay: CR0451
- PDM: CR10nn

The visualisations are structured in folders as follows:

Folder	Image no.	Description contents
START_PAGE	P00001	Setting / display of... - node ID - CAN baud rate - status - GuardErrorNode - PLC cycle time
__MAIN_MENUES	P00010	Menu screen: - Display setup
___MAIN_MENU_1		
_____DISPLAY_SETUP		
_____1_DISPLAY_SETUP1	P65000	Menu screen: - Software version - brightness / contrast - display / set real-time clock
_____1_SOFTWARE_VERSION	P65010	Display of the software version.
_____2_BRIGHTNESS	P65020	Adjustment of brightness / contrast
_____3_SET_RTC	P65030	Display / set real-time clock

In the templates we have organised the image numbers in steps of 10. This way you can switch into different language versions of the visualisations by means of an image number offset.

Supplement project with further functions

3987

You have created a project using an **ifm** template and you have defined the CAN network. Now you want to add further functions to this project.

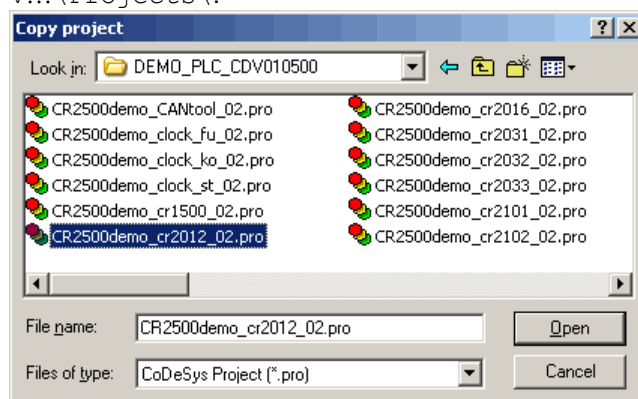
For the example we take a CabinetController CR2500 as CAN open Master to which an I/O CabinetModule CR2011 and an I/O CompactModule are connected as slaves:



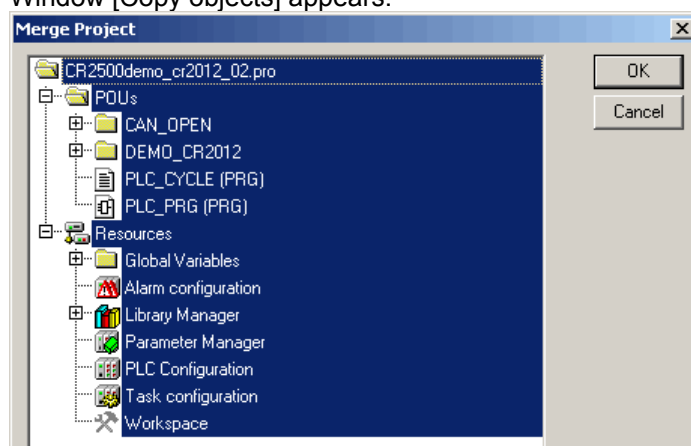
Example: PLC configuration

A joystick is connected to the CR2012 which is to trigger a PWM output on the CR2032. How is that achieved in a fast and simple way?

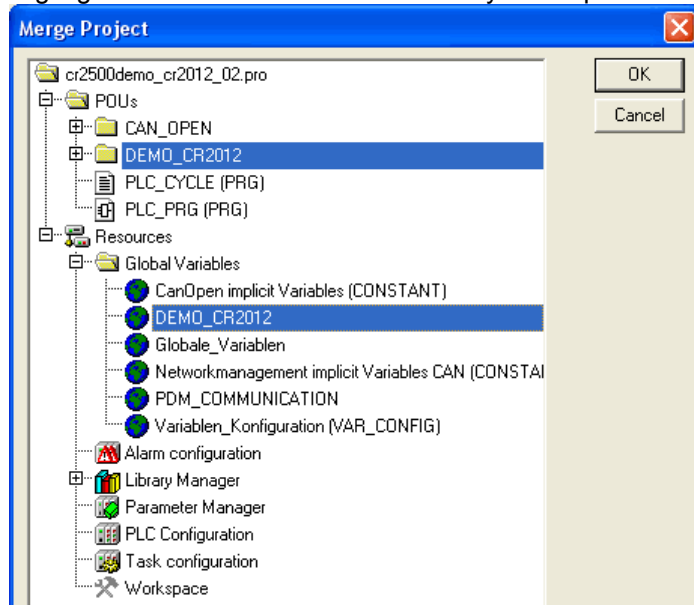
- ▶ Save CoDeSys project!
- ▶ In CoDeSys use [Project] > [Copy...] to open the project containing the requested function:
e.g. CR2500demo_CR2012_02.pro from directory DEMO_PLC_CDV... under C:\...\CoDeSys V...\Projects\:



- ▶ Confirm the selection with [Open].
- > Window [Copy objects] appears:



- Highlight the elements which contain only the requested function, in this case e.g.:

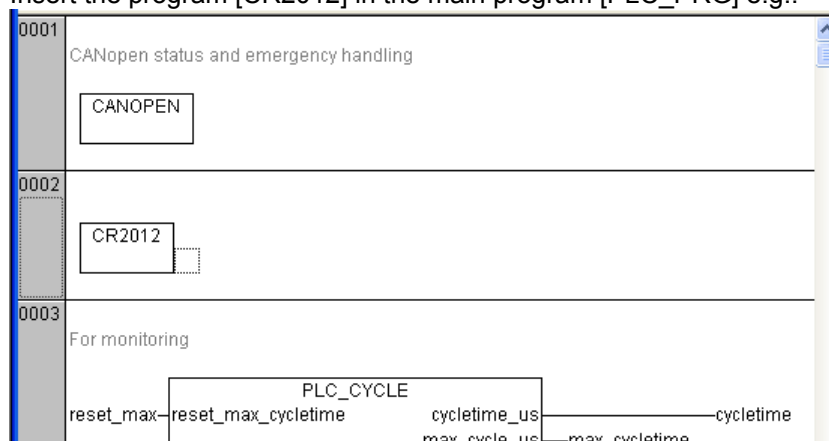


NOTE: In other cases libraries and/or visualisations might be required.

- Confirm the selection with [OK].
- > In our example project the elements selected in the demo project have been added:

POUs:	Resources:
<ul style="list-style-type: none"> CAN_OPEN <ul style="list-style-type: none"> CANOPEN (PRG) DEMO_CR2012 <ul style="list-style-type: none"> CR2012 (PRG) CR2012_DIAI (FB) PLC_CYCLE (PRG) PLC_PRG (PRG) 	<ul style="list-style-type: none"> Global Variables <ul style="list-style-type: none"> CanOpen implicit Variables (CONSTANT) DEMO_CR2012 Globale_Variablen Networkmanagement implicit Variables CAN PDM_COMMUNICATION Variablen_Konfiguration (VAR_CONFIG)

- Insert the program [CR2012] in the main program [PLC_PRG] e.g.:



- The comments of the POU's and global variables usually contain information on how the individual elements have to be configured, included or excluded. This information has to be followed.
- Adapt input and output variables as well as parameters and possible visualisations to your own conditions.
- [Project] > [Save] and [Project] > [Rebuild all].

- ▶ After possibly required corrections and addition of missing libraries (→ Error messages after rebuild) save the project again.
- ▶ Follow this principle to step by step (!) add further functions from other projects and check the results.
- ▶ [Project] > [Save] and
[Project] > [Rebuild all].

6.1.3 ifm demo programs

Contents

Demo program for controller	75
Demo programs for PDM and BasicDisplay	76

3982

In directory DEMO_PLC_CDV... (for Controller) or DEMO_PDM_CDV... (für PDMs) under C:\...\CoDeSys V...\Projects\ we explain certain functions in tested demo programs. If required, these functions can be implemented in own projects. Structures and variables of the **ifm** demos match those in the **ifm** templates.

Each demo program shows just **one** topic. For the Controller as well some visualisations are shown which demonstrate the tested function on the PC screen.

Comments in the POU's and in the variable lists help you adapt the demo to your project.

If not stated otherwise the demo programs apply to all controllers or to all PDMs.

The demo programs described here apply for:

- CoDeSys from version 2.3.9.6
- on the **ecomatmobile** CD from version 010500

Demo program for controller

3995

Demo program	Function
CR2500Demo_CanTool_xx.pro	separate for PDM360, PDM360compact, PDM360smart and Controller: Contains FBs to set and analyse the CAN interface.
CR2500Demo_ClockFu_xx.pro CR2500Demo_ClockKo_xx.pro CR2500Demo_ClockSt_xx.pro	Clock generator for Controller as a function of a value on an analogue input: Fu = in function block diagram K0 = in ladder diagram St = in structured text
CR2500Demo_CR1500_xx.pro	Connection of a keypad module CR1500 as slave of a Controller (CANopen master).
CR2500Demo_CR2012_xx.pro	I/O cabinet module CR2012 as slave of a Controller (CANopen master), Connection of a joystick with direction switch and reference medium voltage.
CR2500Demo_CR2016_xx.pro	I/O cabinet module CR2016 as slave of a Controller (CANopen master), 4 x frequency input, 4 x digital input high side, 4 x digital input low side, 4 x analogue input ratiometric, 4 x PWM1000 output and 12 x digital output.
CR2500Demo_CR2031_xx.pro	I/O compact module CR2031 as slave of a Controller (CANopen master), Current measurement on the PWM outputs
CR2500Demo_CR2032_xx.pro	I/O compact module CR2032 as slave of a Controller (CANopen master), 4 x digital input, 4 x digital input analogue evaluation, 4 x digital output, 4 x PWM output.
CR2500Demo_CR2033_xx.pro	I/O compact module CR2033 as slave of a Controller (CANopen master), 4 x digital input, 4 x digital input analogue evaluation, 4 x digital output,
CR2500Demo_CR2101_xx.pro	Inclination sensor CR2101 as slave of a Controller (CANopen master).
CR2500Demo_CR2102_xx.pro	Inclination sensor CR2102 as slave of a Controller (CANopen master).

Demo program	Function
CR2500Demo_CR2511_xx.pro	I/O smart module CR2511 as slave of a Controller (CANopen master), 8 x PWM output current-controlled.
CR2500Demo_CR2512_xx.pro	I/O smart module CR2512 as slave of a Controller (CANopen master), 8 x PWM output. Display of the current current for each channel pair.
CR2500Demo_CR2513_xx.pro	I/O smart module CR2513 as slave of a Controller (CANopen master), 4 x digital input, 4 x digital output, 4 x analogue input 0...10 V.
CR2500Demo_Interrupt_xx.pro	Example with SET_INTERRUPT_XMS (→ page 362).
CR2500Demo_Operating_hours_xx.pro	Example of an operating hours counter with interface to a PDM.
CR2500Demo_PWM_xx.pro	Converts a potentiometer value on an input into a normed value on an output with the following POU's: - INPUT_VOLTAGE (→ page 253), - NORM (→ page 256), - PWM100 (→ page 281).
CR2500Demo_RS232_xx.pro	Example for the reception of data on the serial interface by means of the Windows hyper terminal.
StartersetDemo.pro StartersetDemo2.pro StartersetDemo2_fertig.pro	Various e-learning exercises with the starter set EC2074.

_xx = indication of the demo version

Demo programs for PDM and BasicDisplay

3996

Demo program	Function
CR1051Demo_CanTool_xx.pro CR1053Demo_CanTool_xx.pro CR1071Demo_CanTool_xx.pro	separate for PDM360, PDM360compact, PDM360smart and Controller: Contains FBs to set and analyse the CAN interface.
CR1051Demo_Input_Character_xx.pro	Allows to enter any character in a character string: - capital letters, - small letters, - special characters, - figures. Selection of the characters via encoder. Example also suited for e.g. entering a password. Figure P01000: Selection and takeover of characters
CR1051Demo_Input_Lib_xx.pro	Demo of INPUT_INT from the library ifm_pdm_input_Vxxyyzz (possible alternative to 3S standard). Select and set values via encoder. Figure P10000: 6 values INT Figure P10010: 2 values INT Figure P10020: 1 value REAL
CR1051Demo_Linear_logging_on_flash_intern_xx.pro	Writes a CVS data block with the contents of a CAN message in the internal flash memory (/home/project/daten.csv), when [F3] is pressed or a CAN message is received on ID 100. When the defined memory range is full the recording of the data is finished. POUs used: - WRITE_CSV_8BYTE, - SYNC. Figure P35010: Display of data information Figure P35020: Display of current data record Figure P35030: Display of list of 10 data records

Demo program	Function
CR1051Demo_O2M_1Cam_xx.pro	<p>Connection of 1 camera O2M100 to the monitor with CAM_O2M. Switching between partial screen and full screen.</p> <p>Figure 39000: Selection menu Figure 39010: Camera image + text box Figure 39020: Camera image as full screen Figure 39030: Visualisation only</p>
CR1051Demo_O2M_2Cam_xx.pro	<p>Connection of 2 cameras O2M100 to the monitor with CAM_O2M. Switching between the cameras and between partial screen and full screen.</p> <p>Figure 39000: Selection menu Figure 39010: Camera image + text box Figure 39020: Camera image as full screen Figure 39030: Visualisation only</p>
CR1051Demo_Powerdown_Retain_bin_xx.pro	<p>Example with PDM_POWER_DOWN from the library <code>ifm_CR1051_Vxxyyzz.Lib</code>, to save retain variable in the file <code>Retain.bin</code>. Simulation of ShutDown with [F3].</p>
CR1051Demo_Powerdown_Retain_bin2_xx.pro	<p>Example with PDM_POWER_DOWN from the library <code>ifm_CR1051_Vxxyyzz.Lib</code>, to save retain variable in the file <code>Retain.bin</code>. Simulation of ShutDown with [F3].</p>
CR1051Demo_Powerdown_Retain_cus_t_xx.pro	<p>Example with PDM_POWER_DOWN and the PDM_READ_RETAIN from the library <code>ifm_CR1051_Vxxyyzz.Lib</code>, to save retain variable in the file <code>/home/project/myretain.bin</code>. Simulation of ShutDown with [F3].</p>
CR1051Demo_Read_Textline_xx.pro	<p>The example program reads 7 text lines at a time from the PDM file system using READ_TEXTLINE.</p> <p>Figure P01000: Display of read text</p>
CR1051Demo_Real_in_xx.pro	<p>Simple example for entering a REAL value in the PDM.</p> <p>Figure P01000: Enter and display REAL value</p>
CR1051Demo_Ringlogging_on_flash_intern_xx.pro	<p>Writes a CVS data block in the internal flash memory when [F3] is pressed or a CAN message is received on ID 100. The file names can be freely defined. When the defined memory range is full the recording of the data starts again.</p> <p>POUs used:</p> <ul style="list-style-type: none"> - WRITE_CSV_8BYTE, - SYNC. <p>Figure P35010: Display of data information Figure P35020: Display of current data record Figure P35030: Display of list of 8 data records</p>
CR1051Demo_Ringlogging_on_flash_pcmcia_xx.pro	<p>Writes a CVS data block on the PCMCIA card when [F3] is pressed or a CAN message is received on ID 100. The file names can be freely defined. When the defined memory range is full the recording of the data starts again.</p> <p>POUs used:</p> <ul style="list-style-type: none"> - WRITE_CSV_8BYTE, - OPEN_PCMCIA, - SYNC. <p>Figure P35010: Display of data information Figure P35020: Display of current data record Figure P35030: Display of list of 8 data records</p>
CR1051Demo_RW-Parameter_xx.pro	<p>In a list parameters can be selected and changed.</p> <p>Example with the following POUs:</p> <ul style="list-style-type: none"> - READ_PARAMETER_WORD, - WRITE_PARAMETER_WORD. <p>Figure P35010: List of 20 parameters</p>

_xx = indication of the demo version

6.2 Function configuration of the inputs and outputs

Contents

Configure inputs	79
Configure outputs	92

1375

For some devices of the **ecomatmobile** controller family, additional diagnostic functions can be activated for the inputs and outputs. So the corresponding input and output signal can be monitored and the application program can react in case of a fault.

Depending on the input and output, certain marginal conditions must be taken into account when using the diagnosis:

- It must be checked by means of the data sheet if the device used has the described input and output groups.
- Constants are predefined (e.g. IN_DIGITAL_H) in the device libraries (e.g. `ifm_CR0020_Vx.LIB`) for the configuration of the inputs and outputs.
For details → Possible operating modes inputs / outputs (→ page [370](#)).

Only for CRn2nn: The ExtendedController (or ExtendedSafetyController) is configured via the same system flags as the ClassicController (or SafetyController). If it is used in the operating mode 2 (→ chapter Operating modes of the ExtendedController (→ page [103](#))) the designations of the inputs and outputs in the second controller are indicated by an appended _E.

6.2.1 Configure inputs

Contents

Digital inputs.....	79
Digital safety inputs	80
Inputs for fail-safe inductive sensors.....	83
Fast inputs.....	86
Fast safety inputs	86
Analogue inputs.....	88
Use of analogue inputs for digital signals	89
Input group I0 (ANALOG0...7 or %IX0.0...%IX0.7)	90
Input group I1...I4 (%IX0.8...%IX2.7)	91

3973

Digital inputs

1015

Depending on the device, the digital inputs can be configured differently. In addition to the protective mechanisms against interference, the digital inputs are internally evaluated via an analogue stage. This enables diagnosis of the input signals. But in the application software the switching signal is directly available as bit information. For some of these inputs (CRnn32: for all inputs) the potential can be selected.

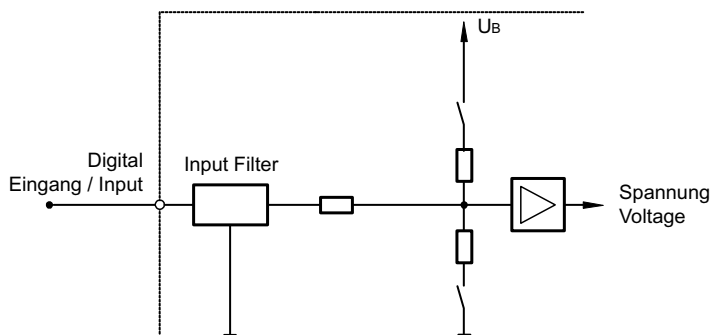
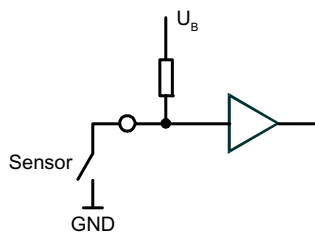
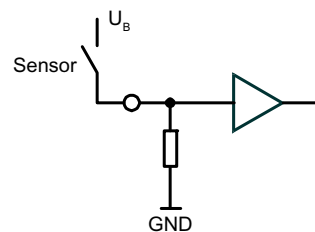


Figure: Block diagram high/low side input for negative and positive sensor signals



High side input for negative sensor signal



Low side input for positive sensor signal

Digital safety inputs

3791

In the SafetyController only the following inputs are permitted for safety functions (→ data sheet):

SafetyController	input addresss	Number safety inputs
CR7020, CR7021	%IX0.00...%IX0.07	8
	%IX0.12...%IX0.15	4
	%IX1.04...%IX1.07	4
CR7505, CR7506	%IX0.00...%IX0.07	8
	%IX0.12...%IX0.15	4
	%IX1.04...%IX1.07	4
CR7200, CR7201	%IX0.00...%IX0.07	8
	%IX0.12...%IX0.15	4
	%IX1.04...%IX1.07	4
	%IX32.00...%IX32.07	8 *)
	%IX32.12...%IX32.15	4 *)
	%IX33.04...%IX33.07	4 *)

*) only for a separate programming of CPU1 and CPU2

NOTE

Only when the digital inputs of the operating system are activated for the analogue inputs can these digital inputs be used for applications to Performance Level PL d (→ Section The risk graph to ISO 13849 (→ page 18)) (safety-integrity level SIL CL 2). To do so, the operating mode of the input in question has to be set to IN_SAFETY. This activates the automatic monitoring and testing of the digital input.

If an error is detected during this internal testing, the corresponding bit (only for safety inputs) in the error flag ERROR_IO and the error flags ERROR_ANALOG and ERROR_IO are set.

Errors in the wiring (short circuit, wire break) or in the sensor are NOT detected by these tests. Cause: for digital signals (e.g. of mechanical switches) only the states 0 (no voltage applied) and 1 (voltage applied) are possible.

Therefore the input signals must be connected to the controller in a redundant and diverse manner as well as via separate cables and processed by the application software in a redundant and diverse manner. In addition, the inputs should be in different input groups (if possible, except for mere analogue signals). Using the diagnostic function does not release the user from this signal processing.

NOTE

SafetyControllers do NOT support the diagnosis ...

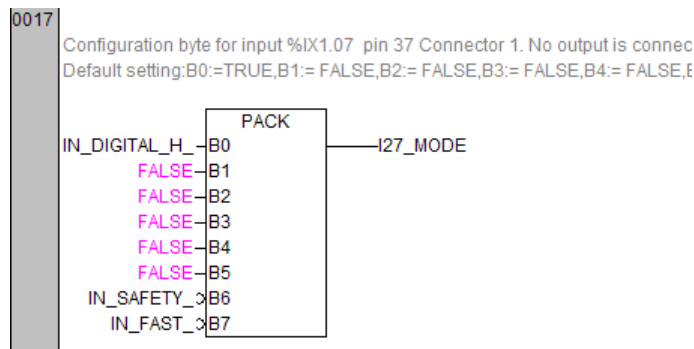
- via an additional resistor circuit for mechanical switches or
- of sensors according to NAMUR.

For safety-related signals preferably only inputs without parallel outputs should be set. For redundant processing the input channels from the second input group (group of four) must be used.

NOTE

To monitor two-channel safety devices (e.g. e-stop) SAFE_INPUTS_OK (→ page 38) must be used. If the safety device set up with this FB is not regularly used, it must be tested manually at defined intervals. This ensures that an error (e.g. in the wiring or the e-stop) is detected.

The following **example** shows the manual configuration of the input %IX1.07 in the SafetyController:



Screenshot: manual configuration of the input %IX1.07 in the SafetyController

If more than 8 (16) safety-related inputs are needed in the application, redundant processing can also be carried out alternatively via the following inputs:

CR7020, CR7021	%IX1.08...%IX1.15	
CR7505, CR7506	---	
CR7200, CR7201	%IX1.08...%IX1.15	%IX33.08...%IX33.15

Using a safety-related channel is imperative for the first input:

CR7020, CR7021	%IX0.12...%IX0.15 %IX1.04...%IX1.07	
CR7505, CR7506	%IX0.12...%IX0.15 %IX1.04...%IX1.07	
CR7200, CR7201	%IX0.12...%IX0.15 %IX1.04...%IX1.07	%IX32.12...%IX32.15 *) %IX33.04...%IX33.07 *)

On no account is redundant processing only with the input channels %IX1.08...%IX1.15 (%IX33.08...%IX33.15) allowed.

*) Only for a separate programming of CPU1 and CPU2

Connect e-stop

3792

In general, e-stops can also be connected to the SafetyController and can be processed directly via the controller. Fail-safe sensors have to be connected via two channels and in a diverse way. Switch-off has to be done via the safety-related outputs.

NOTE

To monitor two-channel safety devices (e.g. e-stop) **SAFE_INPUTS_OK** (→ page 38) must be used. If the safety device set up with this FB is not regularly used, it must be tested manually at defined intervals. This ensures that an error (e.g. in the wiring or the e-stop) is detected.

→ Example: **SAFE_INPUTS_OK** (→ page 40)

Plausibility check via the process

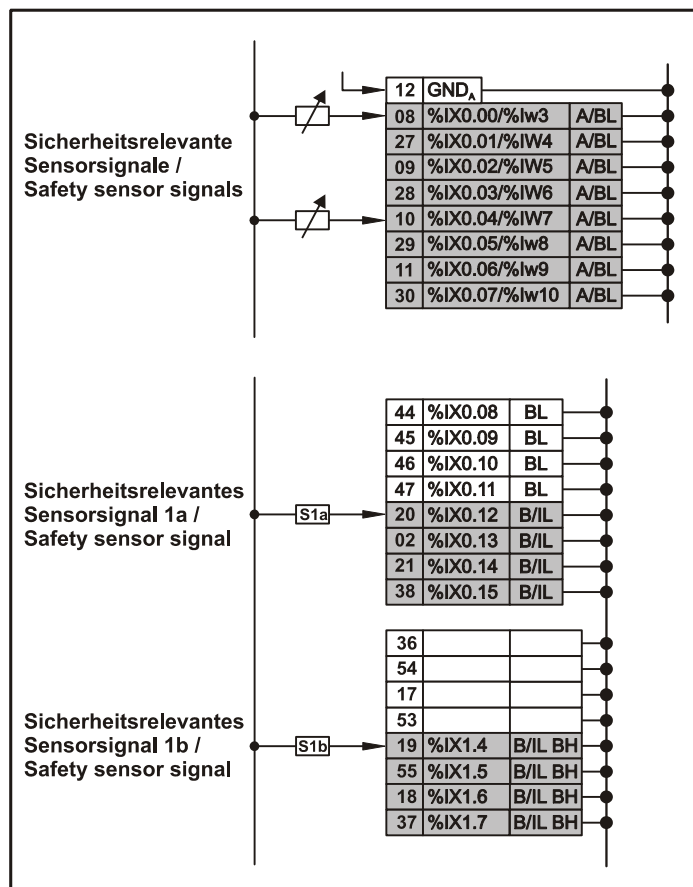
3795

If the application permits, sufficient fault safety can be achieved by the following measures:

- Selection of suitable sensing element (mechanical or electronic),
- proper installation and
- checking of certain components with regard to plausibility.

This makes the installation of two identical sensing elements in one mounting position obsolete.

Wiring examples:



Example analogue sensor:
Input 08 with voltage signal,
Input 10 with current signal
of the same sensor.

Example digital sensor:
Input 20 with normally open signal,
Input 19 with normally closed signal
of the same sensor.

Inputs for fail-safe inductive sensors

3805

With the SafetyController up to 4 safety chains (for the ExtendedSafetyController up to 8 safety chains) each consisting of max. 9 inductive fail-safe sensors in series (i.e. up to 36 or 72 fail-safe sensors) can be triggered and processed. An additional evaluation electronics is not required. The generation of the required test signal and the processing of the sensor output signals are handled directly via the SafetyController.

Fail-safe sensors have to be monitored with SAFETY_SWITCH (→ page [41](#))

NOTE

At present the SafetyController only supports the following unit types:

- order no. GG505S, cylindrical, M18, type GIGA
- order no. GI505S, cylindrical, M30, type GIIA
- order no. GM504S, rectangular, type GIMC
- order no. GM505S, rectangular, type GIMC

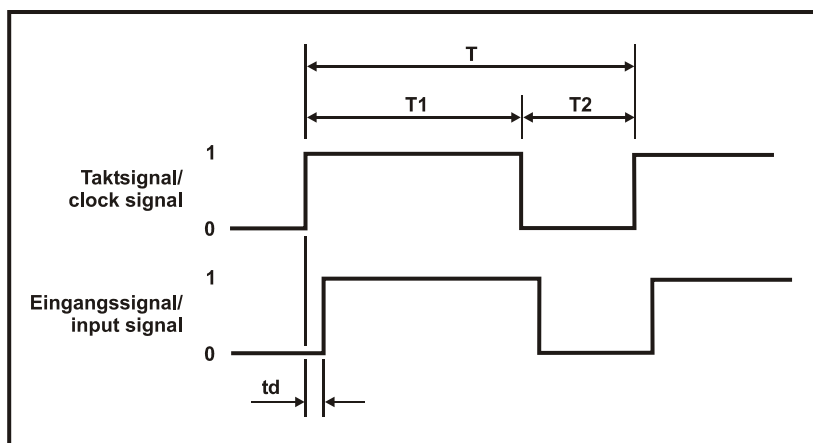
All units have to be supplied with 24 V DC. The use of fail-safe sensors in on-board systems of 12 V DC is not possible.

Operating principle

3806

The fail-safe sensors are supplied with a supply voltage of 24 V DC. In addition, the sensor has to get a clock signal from the controller. By means of the clock signal wiring errors (wire break, short circuit, and cross fault) and a simple defeating of the sensor (e.g. by bridging clock signal and control input) are detected. The sensor monitors and evaluates clock signals generated in the controller. In addition, the sensor monitors the supply voltage and the proper positioning of the damping element.

When the sensor detects no error, the clock signal is provided again to the SafetyController as input signal with a delay of approx. 1.5 ms (time t_d). The time offset and the correct signal form are monitored and evaluated by the controller. If everything is correct, the output of the software function is switched on and can be further processed as a digital input signal.



Typical response times of the SafetyController (without response time of the sensor):

T	= 250 ms + 2 x cycle time
T1	= 200 ms + 1 x cycle time
T2	= 50 ms + 1 x cycle time
Td	≈1.5 ms
Response time on safety request (SWITCH_ON = FALSE)	= max. 50 ms + 1 x cycle time (typ. cycle time)
Response time to the rising edge of the sensor signal (sensor damped)	= max. 250 ms + 2 x cycle time (typ. 100 ms)

For further technical data → see unit description of the individual sensors.

Permissible clock outputs:

CR7020, CR7021	Q23 / %QX0.07 Q47 / %QX1.07	
CR7505, CR7506	Q23 / %QX0.07	
CR7200, CR7201	Q23 / %QX0.07 Q47 / %QX1.07	Q55 / %QX32.07 Q99 / %QX33.07

Permissible signal inputs for inductive fail-safe sensors:

CR7020, CR7021, CR7505, CR7506	I24 / %IX1.04 I25 / %IX1.05 I26 / %IX1.06 I27 / %IX1.07	
CR7200, CR7201	I24 / %IX1.04 I25 / %IX1.05 I26 / %IX1.06 I27 / %IX1.07	I56 / %IX33.04 I57 / %IX33.05 I58 / %IX33.06 I59 / %IX33.07

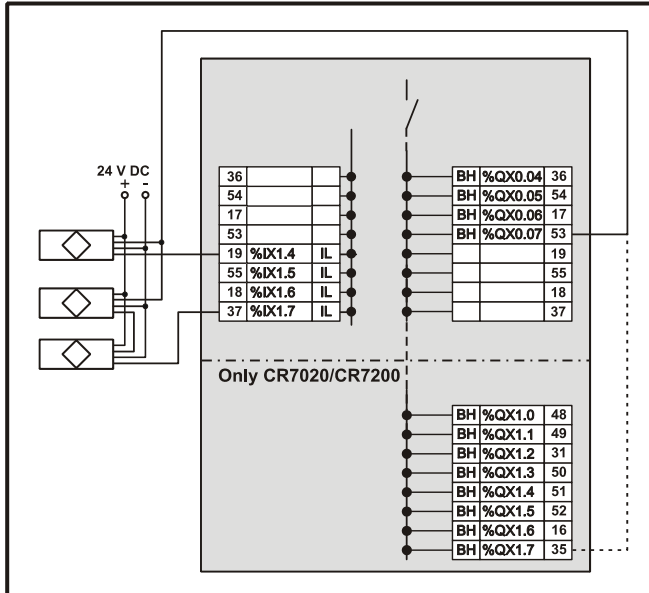
Number of independent safety chains / fail-safe sensors:

CR7020, CR7021, CR7505, CR7506	4 safety chains with 9 fail-safe sensors each = 36 fail-safe sensors
CR7200, CR7201	8 safety chains with 9 fail-safe sensors each = 72 fail-safe sensors

Example fail-safe sensor

3808

Connection of 2 safety chains with a total of 3 inductive fail-safe sensors:



Fail-safe sensors have to be monitored with SAFETY_SWITCH (→ page [41](#)).

Fast inputs

1018

In addition, the **ecomatmobile** controllers have up to 16 fast counter/pulse inputs for an input frequency up to 50 kHz (→ data sheet). If, for example, mechanical switches are connected to these inputs, there may be faulty signals in the controller due to contact bouncing. Using the application software, these "faulty signals" must be filtered if necessary.

Furthermore it has to be noted whether the pulse inputs are designed for frequency measurement (FRQx) and/or period measurement (CYLx) (→ data sheet).

The following FBs, for example, can be used here:

On FRQx inputs:

- Frequency measurement with FREQUENCY (→ page [259](#))
- Fast counter with FAST_COUNT (→ page [270](#))

On CYLx inputs:

- Period measurement with PERIOD (→ page [261](#)) or with PERIOD_RATIO (→ page [263](#))
- Phase position of 2 fast inputs compared via PHASE (→ page [265](#))

Info

When using these units, the parameterised inputs and outputs are automatically configured, so the programmer of the application does not have to do this.

Fast safety inputs

3797

Fast inputs for safety functions are (→ data sheet):

CR7020, CR7021, CR7505, CR7506	%IX0.12...%IX0.15 %IX1.04...%IX1.07	
CR7200, CR7201	%IX0.12...%IX0.15 %IX1.04...%IX1.07	%IX32.12...%IX32.15 %IX33.04...%IX33.07

NOTE

Only if FREQUENCY (→ page [259](#)) and/or PERIOD (→ page [261](#)) are used for the fast inputs (→ Table above) and the measured frequencies are compared via SAFE_FREQUENCY_OK (→ page [36](#)) can these fast inputs be used for applications up to Performance Level PI d (→ chapter The risk graph to ISO 13849 (→ page [18](#))) (safety-integrity level SIL CL 2).

Errors in the wiring (short circuit, wire break) or in the sensor are NOT detected by these tests. Cause: for digital signals (e.g. of mechanical switches) only the states 0 (no voltage applied) and 1 (voltage applied) are possible.

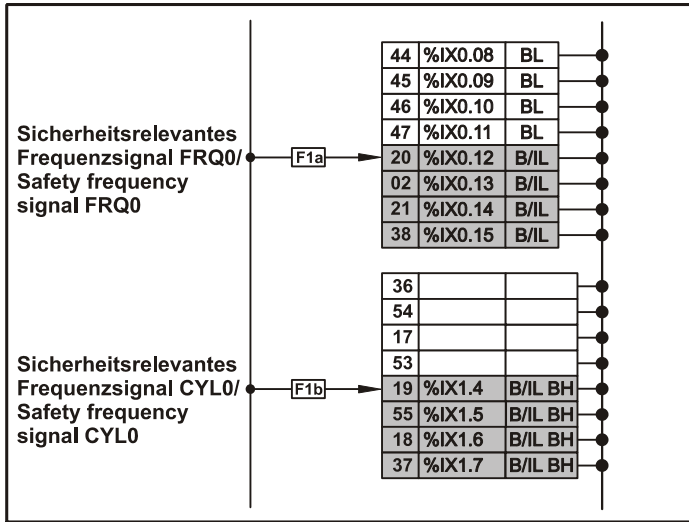
Therefore the input signals must be connected to the controller in a redundant and diverse manner as well as via separate cables and processed by the application software in a redundant and diverse manner. In addition, the inputs should be in different input groups (if possible, except for mere analogue signals). Using the diagnostic function does not release the user from this signal processing.

Measuring methods for fast inputs

3798

In the case of safety-related frequency measurements the signal frequency has to be determined in two different ways (diversity) in addition to the external wiring. Depending on the selected software function (→ library `CR7nnn_Vxyzzz.LIB`) different hardware parts are used in the SafetyController. FREQUENCY (→ page 259) determines the frequency on the basis of the internal hardware counter; PERIOD (→ page 261) by means of an internal timer. The result of these different measuring methods has to be compared with `SAFE_FREQUENCY_OK` (→ page 36) in the application program.

Wiring example:



→ Example for `SAFE_FREQUENCY_OK` (→ page 37)

Applications

3802

Due to the different measuring methods errors can occur when the frequency is determined:

FREQUENCY (→ page 259) is suited for frequencies between 0.1...50 kHz; the error is reduced at high frequencies.

PERIOD (→ page 261) carries out the period measurement. It is thus suitable for frequencies lower than 1 kHz. The measurement of higher frequencies has a strong impact on the cycle time. This has to be taken into account when designing the application software.

As a consequence, a safe measurement of frequencies is only possible between 100...1000 Hz.

Safety aspects

3803

In the safety considerations errors in the reference measurement of up to 25 % can be tolerated as the reference value is only used as function control of the measuring channel. The frequency value for the application has to be derived from the "exact" measurement.

Use as digital inputs

3804

The permissible high input frequencies also ensure the detection of faulty signals, e.g. bouncing contacts of mechanical switches. If necessary, this has to be suppressed in the application software.

Analogue inputs

1369

The analogue inputs can be configured via the application program. The measuring range can be set as follows:

- current input 0...20 mA
- voltage input 0...10 V
- voltage input 0...30 / 32 V

If in the operating mode "0...30 / 32 V" the supply voltage is read back, the measurement can also be performed ratiometrically. This means potentiometers or joysticks can be evaluated without additional reference voltage. A fluctuation of the supply voltage then has no influence on this measured value.

As an alternative, an analogue channel can also be evaluated digitally.

NOTE

In case of ratiometric measurement the connected sensors should be supplied via the same voltage source as the controller. So, faulty measurements caused by offset voltage are avoided.

In case of digital evaluation the higher input resistance must be taken into account.

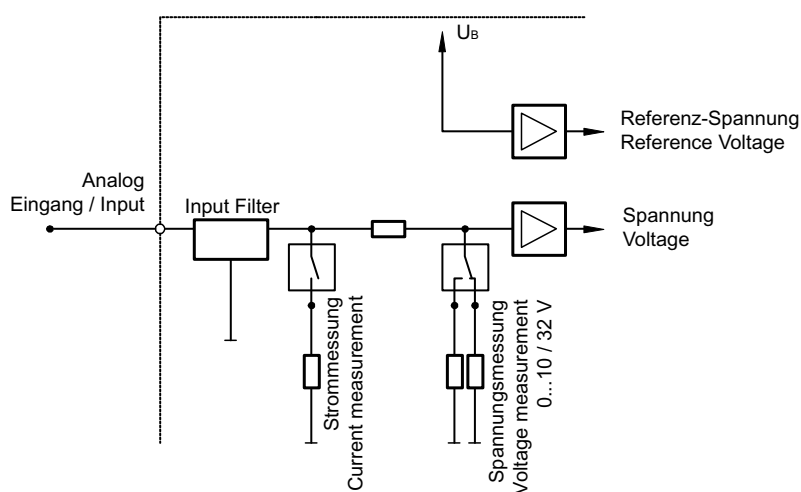


Figure: block diagram of the analogue inputs

Use of analogue inputs for digital signals

3789

The analogue inputs can also be used for the evaluation of digital signals. To do so, operating mode IN_DIGITAL_H has to be set for the selected input.

NOTE

If digital safety signals are to be evaluated via analogue inputs (Mode = IN_SAFETY), this signals should only be statis signals (e.g. mechanical switches).

For the evaluation of frequencies the safety frequencies %IX0.12...%IX0.15 und %IX1.4...%IX1.7 have to be used.

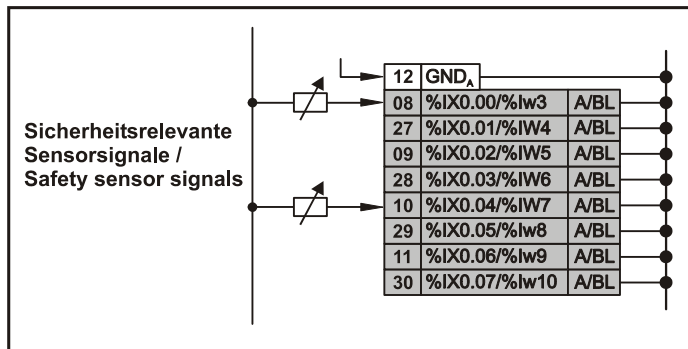
3785

NOTE

Only when the monitoring functions of the operating system are activated for the analogue inputs can these analogue inputs be used for applications to Performance Level PL d (→ Section The risk graph to ISO 13849 (→ page 18)) (safety-integrity level SIL CL 2). To do so, the operating mode of the input in question has to be set to IN_SAFETY. This activates the automatic monitoring and testing of the analogue / digital converter.

If an error is detected during this internal testing, the corresponding bit in the error flag ERROR_I0 and the error flags ERROR_ANALOG and ERROR_IO are set.

Errors in the wiring (short circuit, wire break) or in the sensor are NOT detected by these tests. Therefore analogue input signals must be connected to the controller in a redundant (and if possible diverse) manner as well as via separate cables and processed by the application software in a redundant (and if possible diverse) manner.



Example analogue sensor:
input 08 with voltage signal,
input 10 with current signal
of the same sensor.

Graphics: connection example analogue safety inputs

Furthermore, it is useful to evaluate the signal voltage only in a limited range (e.g. 10...90 %). This allows to detect the following errors:

- short to ground (< 10 %)
- wire break (< 10 %)
- short to voltage supply (> 90 %)
- short circuit (< 10 %)

In addition, analogue safety-related input signals must be evaluated with SAFE_ANALOG_OK (→ page 34).

Input group I0 (ANALOG0...7 or %IX0.0...%IX0.7)

1376

These inputs are a group of analogue channels which can also be evaluated digitally.

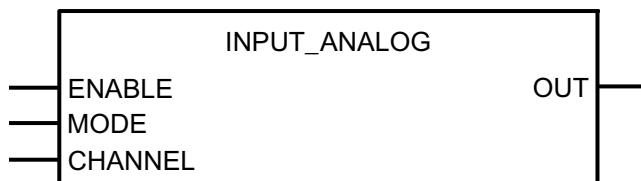
If used as analogue channels, they have diagnostic capabilities at all times via the permanent analogue value in the system variables ANALOG0...ANALOG7 (or ANALOG0_E...ANALOG7_E).

If the analogue inputs are configured for current measurement, the device switches to the safe voltage measurement range (0...30V DC) and the corresponding error bit in the flag byte ERROR_I0 is set when the final value (> 21 mA) is exceeded. When the value is again below the limit value, the input automatically switches back to the current measurement range.

Info

When using the analogue input functions the diagnosis does not have to be activated via the system variable I0x_MODE.

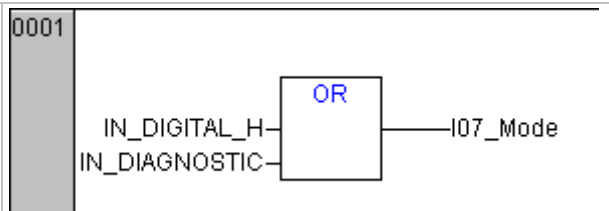
The configuration of the inputs and outputs is carried out via the application software in the latest generation of **ecomatmobile** controllers. INPUT_ANALOG (→ page 251) configures the operating mode of the selected analogue channel via the input MODE. Accordingly, the function of the PWM channels is also set via FBs (→ following example).



As an alternative the inputs and outputs can also be directly set by setting a system variable Ixx_MODE.

Example:

The assignment sets the selected input to the operating mode IN_DIGITAL_H with diagnosis:



If the diagnosis is to be used, it must be activated in addition. The system flag bit DIAGNOSE indicates wire break or short circuit of the input signal as group error.

WARNING

Property damage or bodily injury due to malfunctions possible!

► Do not use **any** sensors with diagnostic capabilities to NAMUR with this input group.

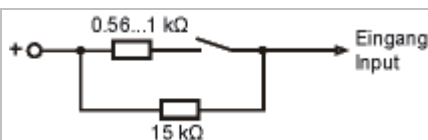


Figure: non-electronic switches

To monitor the input signals of non-electronic switches, they must be equipped with an additional resistor connection.

Input group I1...I4 (%IX0.8...%IX2.7)

1377

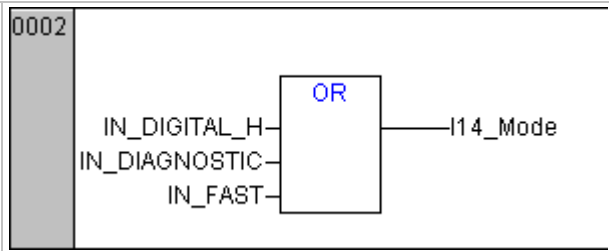
These inputs are digital inputs with internal analogue evaluation for diagnosis. In addition, a part of these inputs can be configured for negative input signals and frequency measurement (only positive input signals).

In principle, negative input signals have no diagnostic capabilities.

The configuration of these inputs is carried out via the system variables I1x_MODE...I4x_MODE. If the diagnosis is to be used, it must be activated in addition. The system flag bit DIAGNOSE indicates wire break or short circuit of the input signal as group error.

Example:

The following assignment sets the selected input to the operating modes IN_DIGITAL_H, fast input and input with diagnosis:



NOTE

Sensors with diagnostic capabilities to NAMUR can be used on all inputs. In this case, no additional resistor connection is required.

To use the diagnostic function for inputs of the group I4 (%IX2.0...%IX2.7), the corresponding outputs (%QX1.0...%QX1.7) must be switched off via the system flags Q4x_MODE. To do so, use the constant OUT_NOMODE.

On delivery, all 8 outputs are switched off.

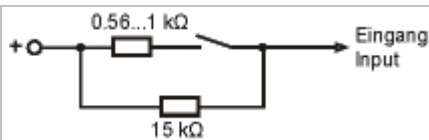


Figure: non-electronic switches

To monitor the input signals of non-electronic switches, they must be equipped with an additional resistor connection.

6.2.2 Configure outputs

Contents

Digital and PWM outputs.....	92
Output group Q1Q2 (%QX0.0...%QX0.7)	96
Output group Q3 (%QX0.8...%QX0.15)	98
Output group Q4 (%QX1.0...%QX1.7)	99

3976

Digital and PWM outputs

1346

Three types of controller outputs can be distinguished:

- High side digital outputs with and without diagnostic function,
- High side digital outputs with and without diagnostic function and additional PWM mode,
- Low side digital outputs with and without diagnostic function,
- PWM outputs which can be operated with and without current control function. Current-controlled PWM outputs are mainly used for triggering proportional hydraulic functions.

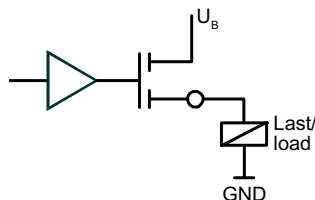
⚠ WARNING

Property damage or bodily injury due to malfunctions possible!

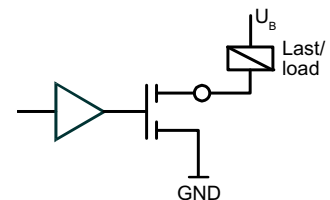
Outputs which are operated in the PWM mode do not support any diagnostic functions and no ERROR flags are set. This is due to the structure of the outputs.

OUT_OVERLOAD_PROTECTION is not active in this mode!

The **ecomatmobile** controllers operate either with high or low side outputs. So, a maximum of 2 H-bridges, e.g. for triggering electric motors, can be implemented in these devices.



High side output for positive output signal



Low side output for negative output signal

⚠ WARNING

Property damage or bodily injury due to malfunctions possible!

The outputs with read back function (outputs with diagnostic capabilities) are to be preferred for safety-related applications, i.e. group VBB_R.

📌 NOTE

If an output is switched off in case of a fault (e.g. short circuit) via the hardware (by means of a fuse), the logic state created by the application program does not change.

To set the outputs again after removal of the peripheral fault, the outputs must first be logically reset in the application program and then set again if required.

Behaviour in case of short circuit, permanent overload or wire break:

(applies as from the hardware version AH, however **not** in the safety mode)

- > System flag ERROR_SHORT_Qx (in case of short circuit or overload) or ERROR_BREAK_Qx (in case of wire break) becomes active.
- > Only in case of short circuit/overload: the operating system deactivates the affected output driver. The logic of the affected output remains TRUE.
After a waiting time the output is activated again, which can lead to periodic switching to short circuit.
The waiting time increases with the (over)load of the output.
Switch-on time in case of short circuit typically 50 µs, considerably longer in case of overload.
- ▶ Evaluate the error flag in the application program!
Reset the output logic, stop the machine if necessary.
If required, switch off the output group VBB_R via RELAY=FALSE (e.g. via ERROR=TRUE).

After fault elimination:

- ▶ Reset the error flag ERROR_..._Qx.
- > The monitoring relay re-enables the output group VBB_R.
- ▶ New setting of the output or restart of the machine.

Digital outputs for safety functions

3811

! NOTE

Only the outputs marked "safe" in the data sheet must be used for applications up to performance level PL d (→ chapter The risk graph to ISO 13849 (→ page 18)) (safety-integrity level SIL CL 2). Only these outputs have diagnostic and monitoring functions (short circuit, wire break, and cross fault) described below.

The monitoring functions of the operating systems that can be activated by the application software have to be used. The application software has to evaluate the error and feedback messages and has to react accordingly.

If an error is detected during this testing, the corresponding bits are set in the error bytes ERROR_BREAK_..., ERROR_SHORT_..., possibly ERROR_OUTPUTBLANKING and the error flag ERROR_IO.

To activate all monitoring functions the bits OUT_SAFETY and OUT_DIAGNOSTIC must be set in the mode byte of the corresponding output.

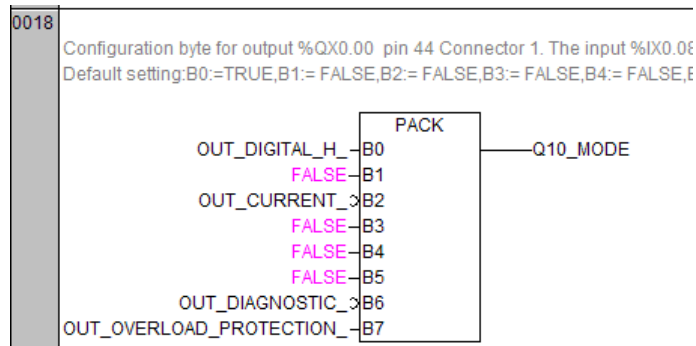
In case of an error switching off the outputs is one of the most important features of machine controllers. The switched-off output (no energy) is considered as safe state.

Therefore the continuous monitoring of the connected actuators for the following errors is absolutely necessary:

- wire break,
- short circuit to supply voltage,
- short circuit to ground and
- cross fault between each other.

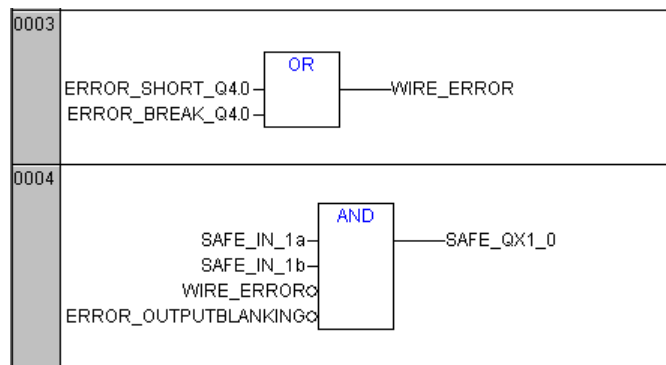
For the errors indicated above, the SafetyController provides outputs with diagnostic capabilities which the operating system partly checks automatically and the user has to evaluate in the application software.

The following **example** shows the manual configuration of the output %QX0.00 in the SafetyController.



Screenshot: manual configuration of output %QX0.00 in the SafetyController

The following **example** shows monitoring of a safe output for various cable faults (short circuit, break, cross fault) via the program:



Screenshot: monitoring of a safe output for wire damage in the SafetyController

Behaviour (in the safety mode) in case of short circuit, permanent overload, wire break or cross fault.

- > System flag ERROR_SHORT_Qx (in case of short circuit or overload) or ERROR_BREAK_Qx (in case of wire break) or ERROR_OUTPUTBLANKING (in case of cross fault) as well as group error flag ERROR_IO and ERROR become active.
- > The operating system deactivates the affected output driver.
The logic of the affected output remains TRUE.
- > The monitoring relay switches off the output group VBB_R.
- > The LED lights in red.
- ▶ Evaluate the error flag in the application program!
Stop the machine.
- ▶ Switch off the controller.

After fault elimination:

- ▶ Switch on the controller again.
- > The monitoring relay re-enables the output group VBB_R.
- ▶ Restart the machine.

Depending on the output group the internal structure of the output channels is different,
→ chapter Function configuration of the inputs and outputs (→ page [78](#)).

Safety outputs for PWM and PWMi

3819

In the SafetyController only the following outputs are permitted for safety functions (→ data sheet):

CR7020, CR7021	%QX0.04...%QX0.07 *) %QX1.00...%QX1.07	
CR7505, CR7506	%QX0.04...%QX0.07 *)	
CR7200, CR7201	%QX0.04...%QX0.07 *) %QX1.00...%QX1.07	%QX32.04...%QX32.07 *) %QX33.00...%QX33.07

*) outputs suited for PWMi

NOTE

No internal testing!

Due to the function principle there is no system internal monitoring and testing for these outputs. If PWMi outputs (→ Table above) are to be used for safety purposes the plausibility of the signals has to be monitored via the application and the application program.

If an output is used as a PWM or a current-controlled PWM output, the criss-fault detection of the output in question must NOT be activated (MODE byte OUT_SAFETY is not set).

Example:

When the PWM function is used the current can be read back via OUTPUT_CURRENT (→ page [291](#)).

When the current-controlled PWM outputs are used it has to be ensured that the output is only triggered within permissible limits. The plausibility can be monitored e.g. with additional sensors (→ figure).

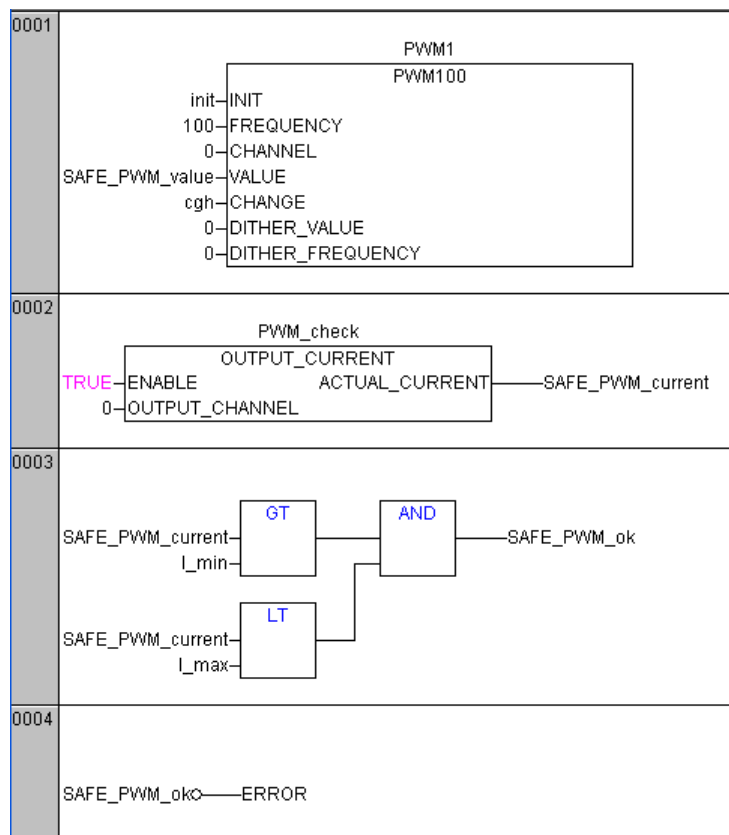


Figure: Monitoring a PWMi output with OUTPUT_CURRENT and additional sensor (input SAFE_PWM_current)

Output group Q1Q2 (%QX0.0...%QX0.7)

1378

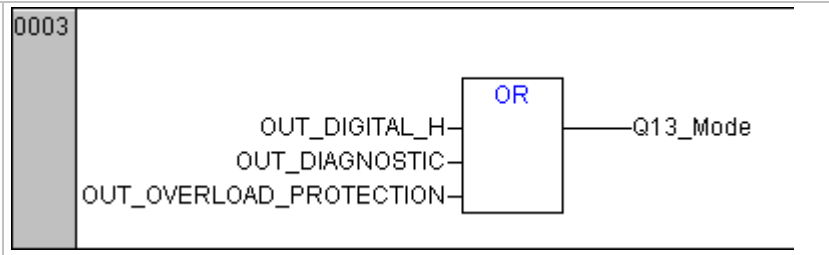
These outputs have two functions. When used as PWM outputs, the diagnosis is implemented via the integrated current measurement channels, which are also used for the current-controlled output functions.

Using OUTPUT_CURRENT (→ page 291) load currents ≥ 100 mA can be indicated.

When used as digital output, configuration is carried out using the system variables Q1x_MODE...Q2x_MODE. If the diagnosis is to be used, it must be activated in addition. Wire break and short circuit of the output signal are indicated separately via the system variables ERROR_BREAK_Q1Q2 and ERROR_SHORT_Q1Q2. The individual output error bits can be masked in the application program, if necessary.

Example:

The assignment sets the selected output to the operating mode OUT_DIGITAL_H with diagnosis. The overload protection is activated (default state).



NOTE

To protect the internal measuring resistors, OUT_OVERLOAD_PROTECTION should always be active (max. measurement current 4.1 A).

IMPORTANT: For the limit values please make sure to adhere to the data sheet!

OUT_OVERLOAD_PROTECTION is not supported in the pure PWM mode.

Wire break and short circuit detection are active when the output is switched **on**.

Safety outputs %QX0.04...0.07 (%QX32.04...32.07)

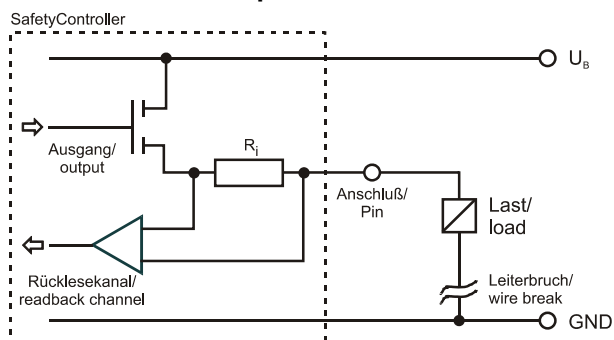
3813

In the SafetyController only the following outputs are permitted for safety functions (→ data sheet):

CR7020, CR7021	%QX0.04...%QX0.07 %QX1.00...%QX1.07 *)	
CR7505, CR7506	%QX0.04...%QX0.07	
CR7200, CR7201	%QX0.04...%QX0.07 %QX1.00...%QX1.07 *)	%QX32.04...%QX32.07 %QX33.00...%QX33.07 *)

*) → Next chapter

Structure of the output channels %QX0.04...%QX0.07 (%QX32.04...%QX32.07)



Monitoring for wire break

Wire-break detection is done via the read back channel. When the output is switched wire break is detected when no current flows on the resistor R_i (no voltage drops). Without wire break the load current flows through the series resistor R_i generating a voltage drop which is evaluated via the read back channel.

Info

The error bit in the system flag byte ERROR_BREAK.... is only set for an output when the state is **output ON**.

Monitoring for short circuit

Short-circuit detection is done via the read back channel. When the output is switched a short circuit against GND is detected when the supply voltage drops over the series resistor.

Info

The error bit in the system flag byte ERROR_SHORT.... is only set for an output when the state is **output ON**.

Monitoring for cross fault

Depending on the result of the risk assessment of the application the outputs must be tested for the following errors:

- cross fault between each other and
- short to supply voltage.

To do so, a short switch-off pulse (approx. 200 μ s) is automatically applied to the monitored outputs (= outputs which can be read back) by the operating system of the controller. This is read back and evaluated by the integrated diagnostic channels. This testing is carried out cyclically (approx. every 30 s) during the whole controller testing and monitoring.

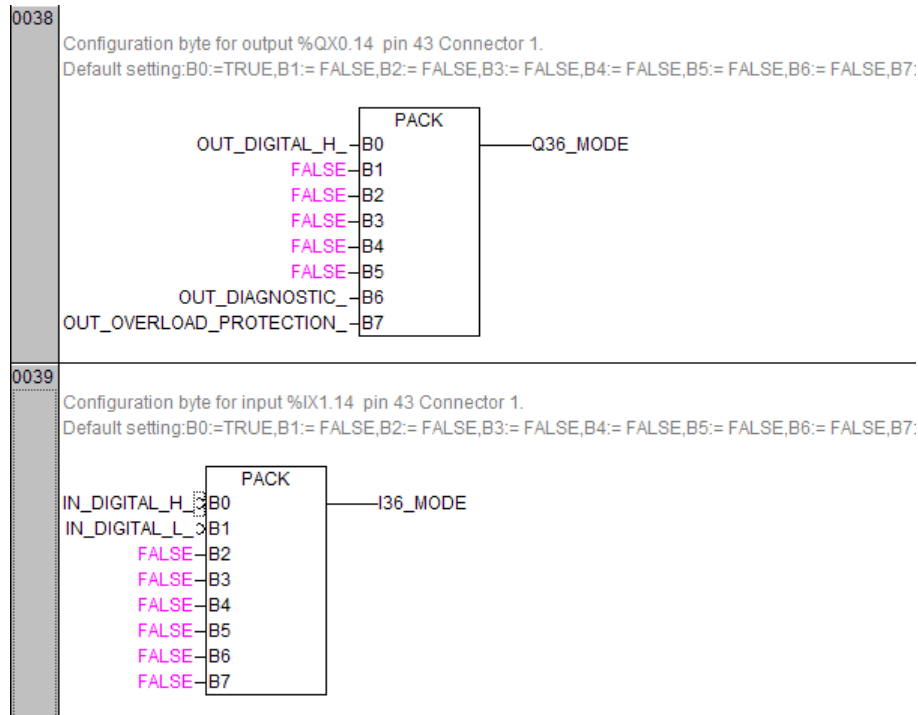
When the output is active the cross fault is detected in the safety-related outputs by this testing.

An error detected by the testing is indicated via the error bit ERROR_OUTPUTBLANKING. With a further diagnosis (see above) the exact error can be determined more precisely.

Output group Q3 (%QX0.8...%QX0.15)

1379

The configuration of these outputs is carried out via the system variables Q3x_MODE. If the diagnosis is to be used, it must be activated in addition. At the same time, the corresponding input must be deactivated by setting the system flag I3x_MODE to IN_NOMODE.



Example: The assignments on the right deactivate the input and set the selected output to the operating mode "OUT_DIGITAL_H with diagnosis".

Wire break and short circuit of the output signal are indicated separately via the system variables ERROR_BREAK_Q3 and ERROR_SHORT_Q3. The individual output error bits can be masked in the application program, if necessary.

IMPORTANT: For the limit values please make sure to adhere to the data sheet!

The wire break detection is active when the output is switched **off**.

The short circuit detection is active when the output is switched **on**.

Output group Q4 (%QX1.0...%QX1.7)

1380

On delivery, this output group is deactivated to enable diagnosis via the inputs. The outputs must be activated in order to be used.

The configuration of these outputs is carried out via the system variables Q4x_MODE. If the diagnosis is to be used, it must be activated in addition. At the same time, the corresponding input must be deactivated by setting the system flag I4x_MODE to IN_NOMODE.

Wire break and short circuit of the output signal are indicated separately via the system variables ERROR_BREAK_Q4 and ERROR_SHORT_Q4. The individual output error bits can be masked in the application program, if necessary.

To implement an H-bridge function, the outputs %QX1.1/2/5/6 can be switched to the mode OUT_DIGITAL_L in addition.

IMPORTANT: For the limit values please make sure to adhere to the data sheet!

The wire break detection is active when the output is switched **off**.

The short circuit detection is active when the output is switched **on**.

Safety outputs %QX1.00...1.07 (%QX33.00...33.07)

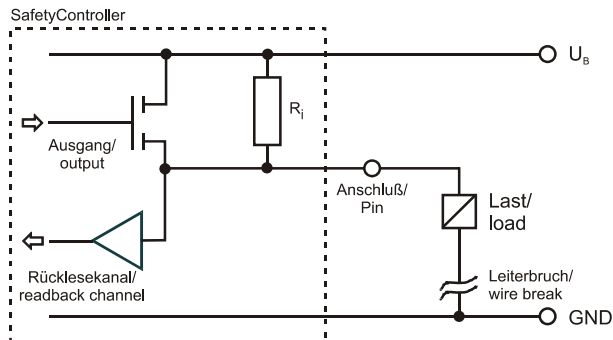
3816

In the SafetyController only the following outputs are permitted for safety functions (→ data sheet):

CR7020, CR7021	%QX0.04...%QX0.07 *) %QX1.00...%QX1.07	
CR7505, CR7506	%QX0.04...%QX0.07 *)	
CR7200, CR7201	%QX0.04...%QX0.07 *) %QX1.00...%QX1.07	%QX32.04...%QX32.07 *) %QX33.00...%QX33.07

*) → Previous chapter

Structure of the output channels %QX1.00...%QX1.07 (%QX33.00...%QX33.07)



Monitoring for wire break

Wire-break detection is done via the read back channel. When the output is blocked wire break is detected when the resistor Ri pulls the read back channel to HIGH potential (VBB). Without the wire break the low-resistance load (RL < 10 kΩ) would force a LOW (logical 0).

Info

The error bit in the system flag byte ERROR_BREAK.... is only set for an output when the state is **output OFF**.

Monitoring for short circuit

Short-circuit detection is done via the read back channel. When the output is switched short circuit against GND is detected when the read back channel is pulled to LOW potential (GND).



Info

The error bit in the system flag byte ERROR_SHORT.... is only set for an output when the state is **output ON**.

Monitoring for cross fault

Depending on the result of the risk assessment of the application the outputs must be tested for the following errors:

- cross fault between each other and
- short to supply voltage.

To do so, a short switch-off pulse (approx. 200 µs) is automatically applied to the monitored outputs (= outputs which can be read back) by the operating system of the controller. This is read back and evaluated by the integrated diagnostic channels. This testing is carried out cyclically (approx. every 30 s) during the whole controller testing and monitoring.

When the output is active the cross fault is detected in the safety-related outputs by this testing.

An error detected by the testing is indicated via the error bit ERROR_OUTPUTBLANKING. With a further diagnosis (see above) the exact error can be determined more precisely.



NOTE

Only limited diagnostic possibilities (no short circuit protection) are available in the configuration LowSide for the outputs %QX1.01, %QX1.02, %QX1.05, %QX1.06 (%QX33.02, %QX33.03, %QX33.05, %QX33.06). Therefore this configuration is not suited for safety signals.

To activate the testing the bits OUT_SAFETY and OUT_DIAGNOSTIC must be set in the mode byte of the corresponding output.

6.3 Hints to wiring diagrams

1426

The wiring diagrams (→ installation instructions of the controllers, chapter "Wiring") show the standard device configurations. The wiring diagrams help allocate the input and output channels to the IEC addresses and the device terminals.

Examples:

12 GND_A

12	Terminal number
GND _A	Terminal designation

30 %IX0.7 BL

30	Terminal number
%IX0.7	IEC address for a binary input
BL	Hardware version of the input, here: Binary Low side

47 %QX0.3 BH/PH

47	Terminal number
%QX0.3	IEC address for a binary output
BH/PH	Hardware version of the output, here: Binary High side or PWMHigh side

The different abbreviations have the following meaning:

A	Analogue input
BH	Binary input/output, high side
BL	Binary input/output, low side
CYL	Input period measurement
ENC	Input encoder signals
FRQ	Frequency input
H-bridge	Output with H-bridge function
PWM	P ulse- w idth m odulated signal
PWM _I	PWM output with current measurement
IH	Pulse/counter input, high side
IL	Pulse/counter input, low side
R	Read back channel for one output

Allocation of the input/output channels:

Depending on the device configuration there is one input and/or one output on a device terminal (→ catalogue, installation instructions or data sheet of the corresponding device).

! NOTE

Contacts of Reed relays may be clogged (reversibly) if connected to the device inputs without series resistor.

- ▶ **Remedy:** Install a series resistor for the Reed relay:
Series resistor = max. input voltage / permissible current in the Reed relay
Example: 32 V / 500 mA = 64 Ohm
- ▶ The series resistor must not exceed 5 % of the input resistance RE of the device input (→ data sheet). Otherwise, the signal will not be detected as TRUE.
Example:
RE = 3 000 Ohm
⇒ max. series resistor = 150 Ohm

6.4 Operating modes of the ExtendedSafetyController

Contents

Operating mode master/master	104
Operating mode master/slave	105

3822

The ExtendedController can be operated in two different ways:

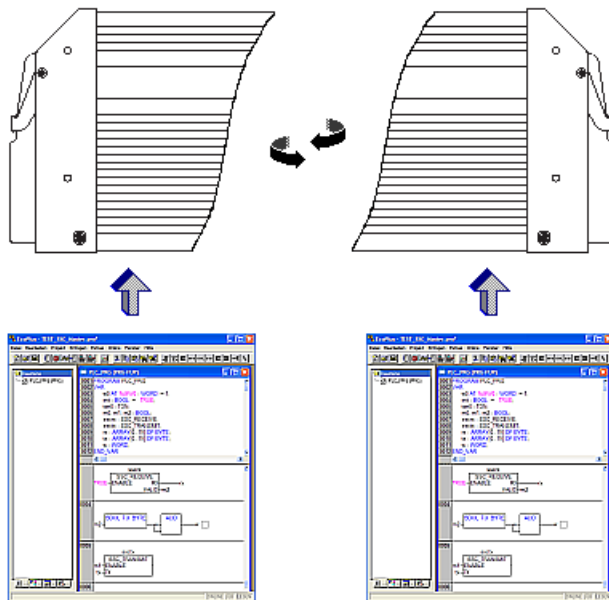
- as master/master:
two separate applications in the **two** controller halves.
Both PLC modes are permissible as safety controller.
Safety-related communication only permissible externally via CANopen-Safety.
→ chapter CANopen Safety in safety-related applications (→ page [231](#))
- as master/slave:
only **one** controller half (which half can be freely defined) with a complete application program.
The secondary PLC module is **not** permissible for safety signals.

The internal interface is only available to non safety-related data exchange.

6.4.1 Operating mode master/master

3823

The use of the ExtendedSafetyController as a safety controller is only permitted in operating mode master/master.



Graphics: Both controller halves used as master/master

In **variant 1** two separate applications are loaded to the two controller halves. They operate completely independently of and asynchronously to each other in a master/master operating mode (not to be confused with CANopen master).

The inputs and outputs are addressed in both controller halves via the same system variables and system functions.

If requested, the internal interface can be used to exchange non-safety-related data between the two halves. For this purpose SSC_TRANSMIT (→ page 337) and SSC_RECEIVE (→ page 335) are included in the application programs (→ description in chapter Data access and data check (→ page 351)).

NOTE

When installing the **ecomatmobile** CD "Software, Tools and Documentation", projects with templates have been stored in the program directory of your PC:

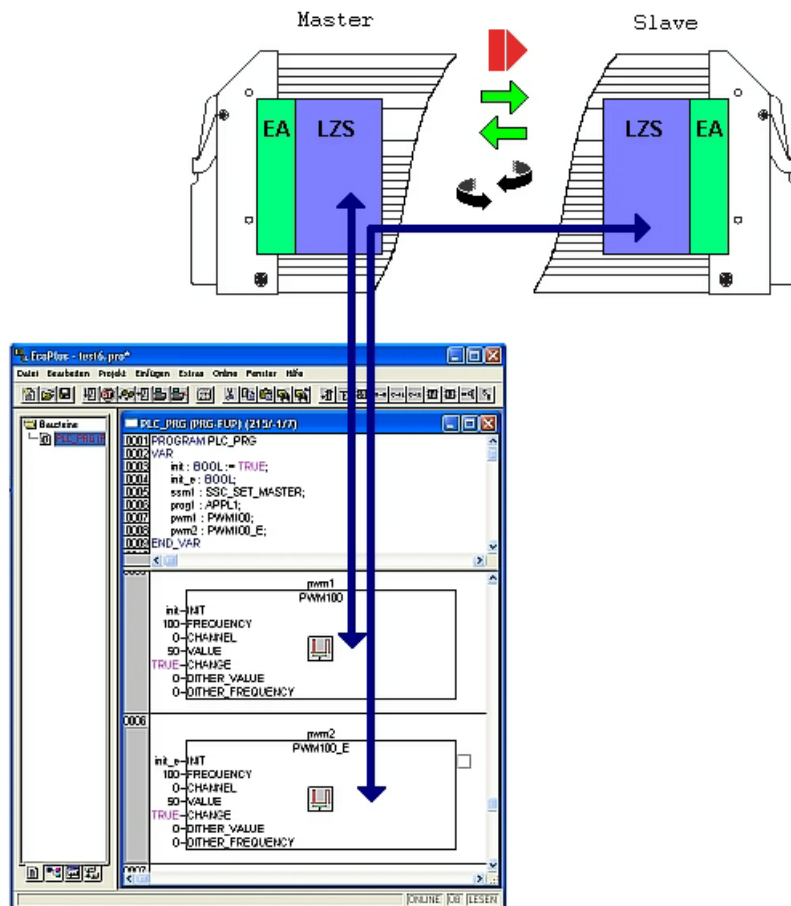
...\\ifm electronic\\CoDeSys V...\\Projects\\Template_CDVxyyzz

- Open the requested template in CoDeSys via:
[File] > [New from template...]
- > CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page 65)

6.4.2 Operating mode master/slave

3824

Not permissible for safety controller.



Graphics: Only one controller half used

Legend:



Synchronisation



Exchange of input/output variables and system flags



Exchange of messages of the application programs

In the **operating mode master/slave** only one controller half (it can be freely defined which one) is loaded with a complete application program. The ExtendedController now behaves like **one** controller. The two controller halves operate in the master/slave operating mode. The inputs and outputs are processed synchronously. To do so, **SSC_SET_MASTER** must be integrated in the application program. The FB assumes the function of initialising the slave.

Furthermore, a little dummy program is loaded into the slave by the master. To do so, this program block from the slave library `ifm_CR0020_DUMMY_Vxxyyzz.LIB` must be integrated into the application program.

Here, too, the same system variables and system units are addressed. For differentiation, the names of the variables and FBs of the second controller half are extended by an _E (for **E**xtended). The data exchange between the two halves of the controller is automatically carried out via the internal interface.

! NOTE

For the 2nd controller half (slave) only a part of the functions of the master controller is available.

This operating mode is NOT permissible for a safety controller.

! NOTE

When installing the **ecomatmobile** CD "Software, Tools and Documentation", projects with templates have been stored in the program directory of your PC:

...\ifm_electronic\CoDeSys V...\Projects\Template_CDVxyzz

- ▶ Open the requested template in CoDeSys via:
[File] > [New from template...]
- > CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page [65](#))

7 Limitations and programming notes

Contents

Limits of the device.....	107
Programming notes for CoDeSys projects.....	110

3058

Here we show you the limits of the device and help you with programming notes.

7.1 Limits of the device

7358

NOTE

Note the limits of the device! → data sheet

7.1.1 CPU frequency

8005

► It must also be taken into account which CPU is used in the device:

Controller family / article no.	CPU frequency [MHz]
BasicController: CR040n	50
CabinetController: CR0301, CR0302	20
CabinetController: CR0303	40
ClassicController: CR0020, CR0505	40
ClassicController: CR0032	150
ExtendedController: CR0200	40
ExtendedController: CR0232	150
SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506	40
SmartController: CR25nn	20

Monitor family / article no.	CPU frequency [MHz]
BasicDisplay: CR0451	50
PDM360: CR1050, CR1051, CR1060	50
PDM360compact: CR1052, CR1053, CR1055, CR1056	50
PDM360NG: CR108n	400
PDM360smart: CR1070, CR1071	20

The higher the CPU frequency, the higher the performance when complex units are used at the same time.

7.1.2 Above-average stress

1497

The following FBs, for example, utilise the system resources above average:

Function block	Above average load
CYCLE, PERIOD, PERIOD_RATIO, PHASE	Use of several measuring channels with a high input frequency
OUTPUT_CURRENT_CONTROL, OCC_TASK	Simultaneous use of several current controllers
CAN interface	High baud rate (> 250 kbits) with a high bus load
PWM, PWM1000	Many PWM channels at the same time. In particular the channels as from 4 are much more time critical
INC_ENCODER	Many encoder channels at the same time
SSC interface	High data traffic on the internal interface of the ExtendedController

The FBs listed above as examples trigger system interrupts. This means: Each activation prolongs the cycle time of the application program.

The following indications should be seen as reference values:

7.1.3 Limits of the SafetyController

1507

Current controller	max. 8 (max. 2x 8) *)	If possible, do not use any other performance-affecting functions
CYCLE, PERIOD, PERIOD_RATIO, PHASE	1 channel	Input frequency ≤ 10 kHz
	4 channels	Input frequency ≤ 2 kHz
INC_ENCODER	max. 4 (max. 2x 4) *)	If possible, do not use any other performance-affecting functions!
SSC interface *)		Optimisation of the data volume

*) only ExtendedSafetyControllers: CR7200, CR7201

! NOTE

- Set the baud rate of the CAN interfaces to max. 250 kBaud!
- Otherwise data can get lost in the 24-hour operation. This means:
- > serious errors and
 - > controller goes to the stop mode.

ATTENTION

Risk that the controller works too slowly! Cycle time must not become too long!

- When the application program is designed the above-mentioned recommendations must be complied with and tested. If necessary, the cycle time must be optimised by restructuring the software and the system set-up.

7.1.4 Watchdog behaviour

1490

For all **ecomatmobile** controllers the program runtime is monitored by a watchdog. If the maximum watchdog time is exceeded, the controller carries out a reset and starts again (SafetyController: controller remains in the reset; LED goes out).

Depending on the hardware the individual controllers have a different time behaviour:

Controller	Watchdog [ms]
BasicController: CR040n	100
BasicDisplay: CR0451	100
CabinetController: CR030n	100...200
ClassicController: CR0020, CR0032, CR0505	100
ExtendedController: CR0200, CR0232	100
PCB controller: CS0015	100...200
SafetyController: CR7nnn	100
SmartController: CR25nn	100...200
PDM360: CR1050, CR1051, CR1060	no watchdog
PDM360compact: CR1052, CR1053, CR1055, CR1056	no watchdog
PDM360NG: CR108n	no watchdog
PDM360smart: CR1070, CR1071	100...200

7.1.5 Available memory (CR7nnn)

4403

Physical memory	Physically existing FLASH memory (non-volatile, slow memory)	2 Mbytes
	Physically existing RAM (volatile, fast memory)	512 Kbytes
	Physically existing EEPROM (non-volatile, slow memory)	---
	Physically existing FRAM (non-volatile, fast memory)	32 Kbytes
Use of the FLASH memory	Memory reserved for the code of the IEC application	704 Kbytes
	Memory for data other than the IEC application that can be written by the user such as files, bitmaps, fonts	1 Mbytes
	Memory for data other than the IEC application that can be processed by the user by means of FBs such as FLASHREAD, FLASHWRITE	64 Kbytes
RAM	Memory for the data in the RAM reserved for the IEC application	180 Kbytes
Remanent memory	Memory for the data declared as VAR_RETAIN in the IEC application	1 Kbyte
	Memory for the flags agreed as RETAIN in the IEC application	256 bytes
	Remanent memory freely available to the user. Access is made via FRAMREAD, FRAMWRITE.	16 Kbytes
	FRAM freely available to the user. Access is made via the address operator.	---

7.2 Programming notes for CoDeSys projects

Contents

FB, FUN, PRG in CoDeSys	110
Note the cycle time!.....	110
Creating application program	111
Save	112
Using ifm downloader	112
Certification and distribution of the safety-related software	112
Changing the safety-relevant software after certification	113

7426

Here you receive tips how to program the device.

- See the notes in the CoDeSys programming manual
→ **ifm** CD "Software, tools and documentation".

7.2.1 FB, FUN, PRG in CoDeSys

8473

In CoDeSys we differentiate between the following unit types:

FB = function block

- A FB may have several inputs and several outputs.
- A FB may be called several times within a project.
- For every call you must declare an instance.

FCT = function

- A function may have several inputs but only one output.
- The output is of the same data type as the function itself.

PRG = program

- A PRG may have several inputs and several outputs.
- A PRG may be called only once within a project.

! NOTE

According to IEC: function blocks must NOT be called within a function.
Otherwise: During the executing the application program will crash.

7.2.2 Note the cycle time!

8006

For the programmable devices from the controller family **ecomatmobile** numerous functions are available which enable use of the devices in a wide range of applications.

As these units use more or fewer system resources depending on their complexity it is not always possible to use all units at the same time and several times.

NOTICE

Risk that the controller acts too slowly! Cycle time must not become too long!

- When designing the application program the above-mentioned recommendations must be complied with and tested. If necessary, the cycle time must be optimised by restructuring the software and the system set-up.

7.2.3 Creating application program

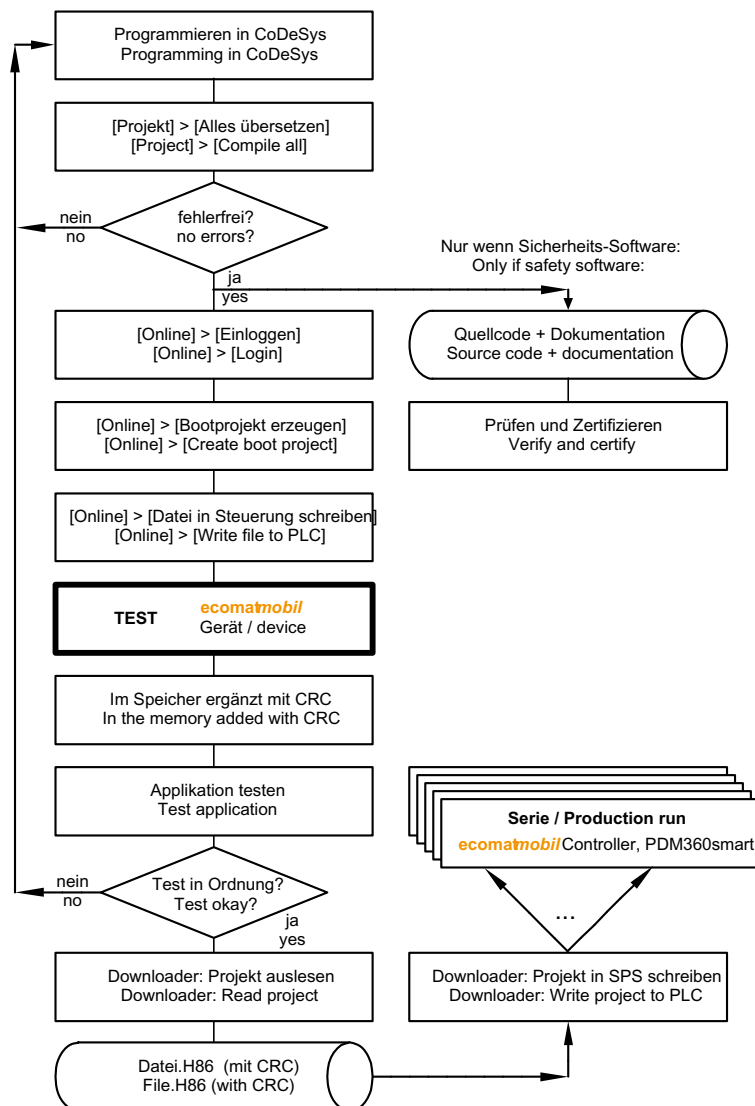
8007

The application program is generated by the CoDeSys programming system and loaded in the controller several times during the program development for testing:

In CoDeSys: [Online] > [Write file in the controller].

For each such download via CoDeSys the source code is translated again. The result is that each time a new checksum is formed in the controller memory. This process is also permissible for safety controllers until the release of the software.

At least for safety-related applications the software and its checksum have to be identical for the series production of the machine.



Graphics: Creation and distribution of the (certified) software

7.2.4 Save

7430

Applies only to the following devices:

- Controller CR0032, CR0232
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360NG: CR108n

! NOTE

Only files in the flash memory (or EEPROM) are protected against power failure.

Always save the related boot project together with your CoDeSys project in the device.

- Menu [Online] > [Create boot project] (this must be carried out again after every change!).
- > After a reboot, the device starts with the boot project last saved.

7.2.5 Using ifm downloader

8008

The **ifm** downloader serves for easy transfer of the program code from the programming station to the controller. As a matter of principle each application software can be copied to the controllers using the **ifm** downloader. Advantage: A programming system with CoDeSys licence is not required.

Safety-related application software MUST be copied to the controllers using the **ifm** downloader so as not to falsify the checksum by which the software has been identified.

! NOTE

The **ifm** downloader cannot be used for the following devices:

- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360: CR1050, CR1051, CR1060,
- PDM360compact: CR1052, CR1053, CR1055, CR1056,
- PDM360NG: CR108n

7.2.6 Certification and distribution of the safety-related software

8009

Only safety-related application software must be certified before it is copied to the series machine and used.

- Saving the approved software
After completion of program development and approval of the entire system by the responsible certification body (e.g. TÜV, BiA) the latest version of the application program loaded in the controller using the **ifm** downloader has to be read from the controller and saved on a data carrier using the name `name_of_the_project_file.H86`. Only this process ensures that the application software and its checksums are stored.
- Download of the approved software.
To equip all machines of a series production with an identical software only this file may be loaded in the controllers using the **ifm** downloader.
- An error in the data of this file is automatically recognised by the integrated checksum when loaded again using the **ifm** downloader.

7.2.7 Changing the safety-relevant software after certification

8010

Changes to the application software using the CoDeSys programming system automatically create a new application file which may only be copied to the safety-related devices after a new certification. To do so, follow again the process described above!

Under the following conditions the new certification may not be necessary:

- a new risk assessment was made for the change,
- NO safety-related elements were changed, added or removed,
- the change was correctly documented.

8 Error messages

Contents

Slight errors	114
Serious errors	115
CAN error	116
Fatal errors	116
Response to the system error	117

2259

If errors are detected while the system is monitored, the PLC reacts. The PLC reaction depends on how serious the error is.

We distinguish:

- Slight errors
- Serious errors
- CAN errors
- Fatal errors

The error flags are not automatically reset by the operating system. This must be done in the application program after analysis and rectification of the errors.

In case of a fault it must be decided depending on the application whether the outputs may be switched on again by switching on again the relay by means of a reset of the ERROR bit.

It is also possible to set the ERROR bit via the application program in case of "freely defined errors".

→ also chapter System flags (→ page [374](#))

8.1 Slight errors

2260

Slight errors are only signalled to the application program. It is up to the application programmer to react to these errors. As minimum reaction the error flag should be reset.

Error message	Type	Description
ERROR_BREAK_Qx	BYTE	Error wire break
ERROR_Ix	BYTE	Peripheral error on the input group x
ERROR_SHORT_Qx	BYTE	Error short circuit

Ix or Qx stands for the input/output group x (word 0...x, depending on the device).

If an input is configured as IN_SAFETY or an output is configured as OUT_SAFETY this error lead to setting a serious error → chapter Serious errors (→ page [115](#)).

8.2 Serious errors

2261

In case of serious errors the ERROR bit can be additionally set. At the same time this also has the effect that the operation LED lights red and the monitoring relays are de-energised. As a result of this, the outputs protected by the relays are also switched off (but not necessarily set to FALSE!).

NOTE

If the outputs are switched off by the relays, the signal states remain unchanged internally. The application programmer must evaluate the ERROR bit and therefore logically switch off the outputs.

Error message	Type	Description
ERROR ¹⁾	BOOL	Set ERROR bit / switch off the relay
ERROR_ANALOG	BOOL	Error in analogue conversion
ERROR_BREAK_Qx ¹⁾	BYTE	Wire break error on output group x
ERROR_CAN_SAFETY	BOOL	SCT, SRVT and data error
ERROR_IO	BOOL	Group error wire break, short circuit, cross fault
ERROR_Ix ²⁾	BYTE	Peripheral error on the input group x
ERROR_OUTPUTBLANKING	BOOL	Cross fault on one of the safety outputs
ERROR_POWER	BOOL	Error undervoltage / overvoltage
ERROR_SHORT_Qx ²⁾	BYTE	Short circuit error on output group x
ERROR_TEMPERATURE	BOOL	Error excess temperature (inside > 85 °C)
ERROR_VBBO	BOOL	Missing voltage on terminal VBB _O
ERROR_VBBR	BOOL	Missing voltage on terminal VBB _R

Ix/Qx stands for the input/output group (word 0...x, depending on the device).

¹⁾ By setting the ERROR system flag the ERROR output (terminal 13) is set to FALSE. In the "error-free state" the output ERROR = TRUE (negative logic).

²⁾ This error message is only seen as a "serious error" if the respective input was configured as IN_SAFETY or the respective output as OUT_SAFETY.

The application program continues running. Communication via the interfaces, e.g. for troubleshooting, is therefore possible.

NOTE

If a serious error occurs, no further diagnosis (wire break, short circuit) of the inputs / outputs can be carried out. Therefore all error bits, for example, must first be reset. A further error analysis must be carried out by an error routine in the application program.

8.3 CAN error

2262

If the mechanism of the safe data transmission via the CAN bus is selected (CANopen safety), all detected errors lead to an error message in the sender (producer) and receiver (consumer) of the data.

The SafetyController is brought to the safe state (serious error). In addition the system flag `ERROR_CAN_SAFETY` is set and all outputs (and the relay) are switched off. The LED lights red.

The application program continues running. Communication via the interfaces, e.g. for troubleshooting, is therefore possible.

With or without CANopen safety the CAN errors can be monitored via the CAN system flags.

Error message	Type	Description
<code>CANx_BUSOFF</code>	BOOL	CAN interface x: Interface is not on the bus
<code>CANx_ERRORCOUNTER_RX ¹)</code>	BYTE	CAN interface x: Error counter reception
<code>CANx_ERRORCOUNTER_TX ¹)</code>	BYTE	CAN interface x: Error counter transmission
<code>CANx_LASTERROR ¹)</code>	BYTE	CAN interface x: Error number of the last CAN transmission: 0= no error ≠0 → CAN specification → LEC
<code>CANx_WARNING</code>	BOOL	CAN interface x: Warning threshold reached (≥ 96)

CANx stands for the number of the CAN interface (CAN 1...x, depending on the device).

¹) Access to this flags requires detailed knowledge of the CAN controller and is normally not required.

8.4 Fatal errors

2263

If a "fatal error" occurs:

- > The PLC is completely stopped,
- > all outputs are switched off,
- > processing of the software is stopped,
- > no communication is possible any more,
- > the LED lights red.

Error message	Type	Description
<code>ERROR_ADDRESS</code>	BOOL	Addressing error
<code>ERROR_CO_CPU</code>	BOOL	Error in the Co processor
<code>ERROR_CPU</code>	BOOL	CPU error
<code>ERROR_DATA</code>	BOOL	System data faulty
<code>ERROR_INSTRUCTION_TIME</code>	BOOL	Processing of instruction too long
<code>ERROR_MEMORY</code>	BOOL	Memory error
<code>ERROR_RELAIS</code>	BOOL	Error relay triggering
<code>ERROR_TIME_BASE</code>	BOOL	Error internal system time

NOTE

If the test input (pin 24) is active, a "fatal error" is treated like a "serious error". The outputs are switched off and the LED lights red. However, communication for a further error diagnosis is possible because the application program continues.

8.5 Response to the system error

1445

In principle, the programmer is responsible to react to the error flags (system flags) in the application program.

The specific error bits and bytes should be processed in the application program. An error description is provided via the error flag. These error bits/bytes can be further processed if necessary.

In principle, all error flags must be reset by the application program. Without explicit reset of the error flags the flags remain set with the corresponding effect on the application program.

In case of serious errors the system flag bit ERROR can also be set. At the same time this also has the effect that the operation LED (if available) lights red, the ERROR output is set to FALSE and the monitoring relays (if available) are de-energised. So the outputs protected via these relays are switched off.

8.5.1 Notes on devices with monitoring relay

1446

Available for the following devices:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Using the logic function via the system flag RELAIS or RELAY_CLAMP_15 (→ chapter Latching (→ page 49)) all other outputs are also switched off.

Depending on the application it must now be decided whether by resetting the system flag bit ERROR the relay – and so also the outputs – may be switched on again.

In addition it is also possible to set the system flag bit ERROR as "defined error" by the application program.

NOTICE

Premature wear of the relay contacts possible.

- ▶ Only use this function for a general switch-off of the outputs in case of an "emergency".
- ▶ In normal operation switch off the relays only without load!
To do so, first switch off the outputs via the application program!

8.5.2 Example process for response to a system error

1447

The system determines an excessive temperature in the controller.

The operating system sets the error bit ERROR_TEMPERATURE.

The application program recognises this state by querying the corresponding bits.

> The application program switches off outputs.

If necessary, the error bit ERROR can be set additionally via the application program.

> Consequences:

- operation LED flashes red
- safety relay is de-energised
- supply voltage of all outputs is switched off
- level of the output ERROR*) is LOW

► Rectify the cause of the error.

> The operating system resets the error bit ERROR_TEMPERATURE.

► If set, the error bit ERROR must be deleted via the application program.

> The relay is energised again and the LED flashes green again.

*) Output not available for CR0301, CR0302, CS0015.

9 Using CAN

Contents

General about CAN	119
Physical connection of CAN	121
Exchange of CAN data	125
Description of the CAN standard program units	129
CAN units acc. to SAE J1939	154
ifm CANopen library	170
CANopen Safety in safety-related applications	231
CAN errors and error handling	242

1163

9.1 General about CAN

Contents

Topology	119
CAN interfaces	120
System configuration	120

1164

The CAN bus (**C**ontroller **A**rea **N**etwork) belongs to the fieldbuses.

It is an asynchronous serial bus system which was developed for the networking of control devices in automobiles by Bosch in 1983 and presented together with Intel in 1985 to reduce cable harnesses (up to 2 km per vehicle) thus saving weight.

9.1.1 Topology

1244

The CAN network is set up in a line structure. A limited number of spurs is allowed. Moreover, a ring type bus (infotainment area) and a star type bus (central locking) are possible. Compared to the line type bus both variants have one disadvantage:

- In the ring type bus all control devices are connected in series so that the complete bus fails if one control device fails.
- The star type bus is mostly controlled by a central processor as all information must flow through this processor. Consequently no information can be transferred if the central processor fails. If an individual control device fails, the bus continues to function.

The linear bus has the advantage that all control devices are in parallel of a central cable. Only if this fails, the bus no longer functions.

NOTE

The line must be terminated at its two ends using a terminating resistor of 120 Ω to prevent corruption of the signal quality.

The devices of **ifm electronic** equipped with a CAN interface have no terminating resistors.

The disadvantage of spurs and star-type bus is that the wave resistance is difficult to determine. In the worst case the bus no longer functions.

For a high-speed bus (> 125 kbits/s) 2 terminating resistors of 120 Ω (between CAN_HIGH and CAN_LOW) must additionally be used at the cable ends.

9.1.2 CAN interfaces

269

The controllers have several CAN interfaces depending on the hardware structure. In principle, all interfaces can be used with the following functions independently of each other:

- CAN at level 2 (layer 2)
- CANopen (→ page [170](#)) protocol to CiA 301/401 for master/slave operation (via CoDeSys)
- CAN Network variables (→ page [198](#)) (via CoDeSys)
- Protocol SAE J1939 (→ page [154](#)) (for engine management)
- Bus load detection
- Error frame counter
- Download interface
- 100 % bus load without package loss

Which CAN interface of the device has which potential, → data sheet of the device.

Informative: more interesting CAN protocols:

- "Truck & Trailer Interface" to ISO 11992 (only available for SmartController CR2051)
- ISOBUS to ISO 11783 for agricultural machines
- NMEA 2000 for maritime applications
- CANopen truck gateway to CiA 413 (conversion between ISO 11992 and SAE J1939)

9.1.3 System configuration

1167

The controllers are delivered with the download identifier 127. The download system uses this identifier (= ID) for the first communication with a non configured module via CAN. The download ID can be set via the PLC browser of the programming system, the downloader or the application program.

As the download mechanism works on the basis of the CANopen SDO service (even if the controller is not operated in the CANopen mode) all controllers in the network must have a unique identifier. The actual COB IDs are derived from the module numbers according to the "predefined connection set". Only one non configured module is allowed to be connected to the network at a time. After assignment of the new participant number 1...126, a download or debugging can be carried out and then another device can be connected to the system.

The download ID is set irrespective of the CANopen identifier. Ensure that these IDs do not overlap with the download IDs or the CANopen node numbers of the other controllers or network participants.

Controller program download		CANopen	
ID	COB ID SDO	Node ID	COB ID SDO
1...127	TX: 580 ₁₆ + download ID	1...127	TX: 580 ₁₆ + node ID
	RX: 600 ₁₆ + download ID		RX: 600 ₁₆ + node ID

NOTE

The CAN download ID of the device must match the CAN download ID set in CoDeSys!
In the CAN network the CAN download IDs must be unique!

9.2 Physical connection of CAN

Contents

Network structure	121
CAN bus level.....	122
CAN bus level according to ISO 11992-1	122
Bus cable length.....	123
Wire cross-sections	124

1177

The mechanisms of the data transmission and error handling described in the chapters Exchange of CAN data (→ page [125](#)) and CAN errors (→ page [242](#)) are directly implemented in the CAN controller. ISO 11898 describes the physical connection of the individual CAN participants in layer 1.

9.2.1 Network structure

1178

The ISO 11898 standard assumes a line structure of the CAN network.

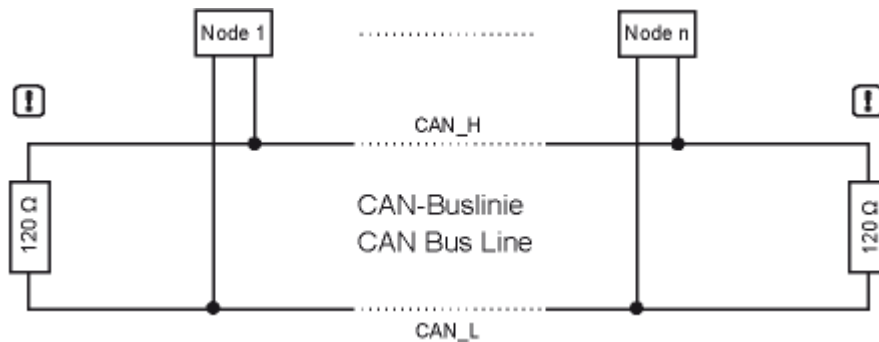


Figure: CAN network line structure

NOTE

The line must be terminated at its two ends using a terminating resistor of 120 Ω to prevent corruption of the signal quality.

The devices of **ifm electronic** equipped with a CAN interface have no terminating resistors.

Spurs

Ideally no spur should lead to the bus participants (node 1 ... node n) because reflections occur depending on the total cable length and the time-related processes on the bus. To avoid system errors, spurs to a bus participant (e.g. I/O module) should not exceed a certain length. 2 m spurs (referred to 125 kbits/s) are considered to be uncritical. The sum of all spurs in the whole system should not exceed 30 m. In special cases the cable lengths of the line and spurs must be calculated exactly.

9.2.2 CAN bus level

1179

The CAN bus is in the inactive (recessive) state if the output transistor pairs are switched off in all bus participants. If at least one transistor pair is switched on, a bit is transferred to the bus. This activates the bus (dominant). A current flows through the terminating resistors and generates a difference voltage between the two bus cables. The recessive and dominant states are converted into voltages in the bus nodes and detected by the receiver circuits.

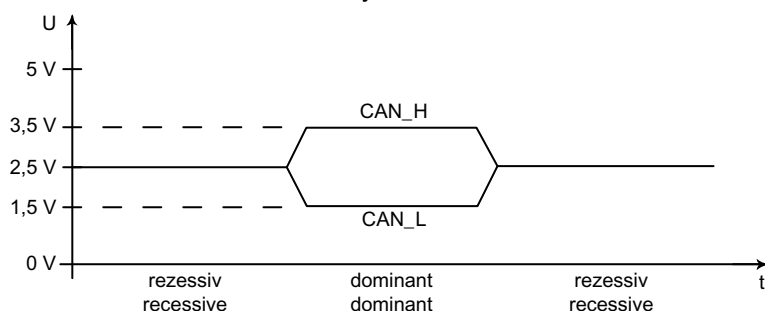


Figure: CAN bus level

This differential transmission with common return considerably improves the transmission security. Noise voltages which interfere with the system externally or shifts of the ground potential influence both signal cables with the same interference. These influences are therefore not considered when the difference is formed in the receiver.

9.2.3 CAN bus level according to ISO 11992-1

1182

Available for the following devices: only SmartController: CR2501 on the 2nd CAN interface.

The physical layer of the ISO 11992-1 is different from ISO 11898 in its higher voltage level. The networks are implemented as point-to-point connection. The terminating networks have already been integrated.

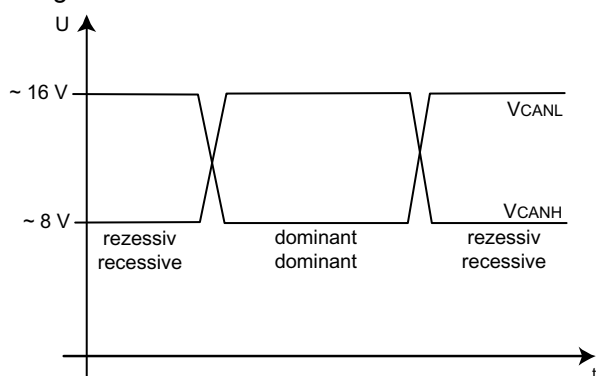


Figure: voltage level to ISO 11992-1 (here: 12 V system)

9.2.4 Bus cable length

1180

The length of the bus cable depends on:

- type of the bus cable (cable, connector),
- cable resistance,
- required transmission rate (baud rate),
- length of the spurs.

To simplify matters, the following dependence between bus length and baud rate can be assumed:

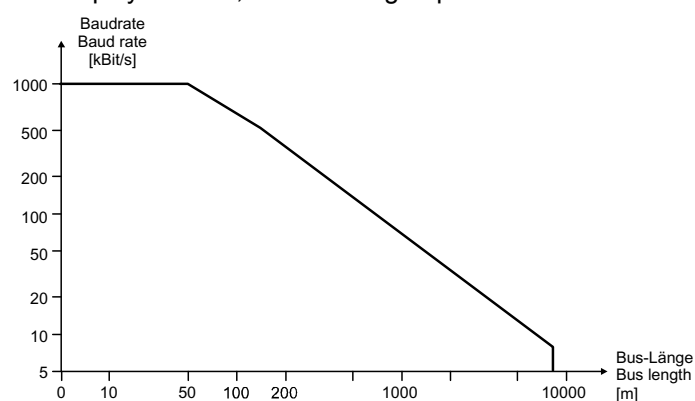


Figure: bus cable length

Baud rate [kBit/s]	Bus length [m]	Bit length nominal [μs]
1 000	30	1
800	50	1.25
500	100	2
250	250	4
125	500	8
62.5	1 000	20
20	2 500	50
10	5 000	100

Table: Dependencies bus length / baud rate / bit time

9.2.5 Wire cross-sections

1181

For the layout of the CAN network the wire cross-section of the bus cable used must also be taken into account. The following table describes the dependence of the wire cross-section referred to the cable length and the number of the connected nodes.

Cable length [m]	Wire cross-section at 32 nodes [mm ²]	Wire cross-section at 64 nodes [mm ²]	Wire cross-section at 100 nodes [mm ²]
≤ 100	0.25	0.25	0.25
≤ 250	0.34	0.50	0.50
≤ 500	0.75	0.75	1.00

Depending on the EMC requirements the bus cables can be laid out as follows:

- in parallel,
- as twisted pair
- and/or shielded.

9.3 Exchange of CAN data

Contents

Hints	126
Data reception	128
Data transmission.....	128

1168

CAN data is exchanged via the CAN protocol of the link layer (level 2) of the seven-layer ISO/OSI reference model specified in the international standard ISO 11898.

Every bus participant can transmit messages (multimaster capability). The exchange of data functions similarly to radio. Data is transferred on the bus without transmitter or address. The data is only marked by the identifier. It is the task of every participant to receive the transmitted data and to check by means of the identifier whether the data is relevant for this participant. This procedure is carried out automatically by the CAN controller together with the operating system.

For the normal exchange of CAN data the programmer only has to make the data objects with their identifiers known to the system when designing the software. This is done via the following FBs:

- CANx_RECEIVE (→ page [145](#)) (receive CAN data) and
- CANx_TRANSMIT (→ page [143](#)) (transmit CAN data).

Using these FBs the following units are combined into a data object:

- RAM address of the useful data,
- data type,
- selected identifier (ID).

These data objects participate in the exchange of data via the CAN bus. The transmit and receive objects can be defined from all valid IEC data types (e.g. BOOL, WORD, INT, ARRAY).

The CAN message consists of a CAN identifier (CAN-ID (→ page [126](#))) and maximum 8 data bytes. The ID does not represent the transmit or receive module but identifies the message. To transmit data it is necessary that a transmit object is declared in the transmit module and a receive object in at least one other module. Both declarations must be assigned to the same identifier.

9.3.1 Hints

8394

CAN-ID

1166

Depending of the CAN-ID the following CAN identifiers are free available for the data transfer:

CAN-ID base	CAN-ID extended
11 bits	29 bits
2 047 CAN identifiers	536 870 912 CAN identifiers
Standard applications	Motor management (SAE J1939), Truck & Trailer interface (ISO 11992)

! NOTE

In some devices the 29 bits CAN-ID is not available for all CAN interfaces, → data sheet.

Example 11 bits CAN-ID (base):

S O F	CAN-ID base Bit 28 ... Bit 18										R T R	I D E
0	0	0	0	0	1	1	1	1	1	1	0	0
	0				7					F		

Example 29 bits CAN-ID (extended):

S O F	CAN-ID base Bit 28 ... Bit 18											S R R	I D E	CAN-ID extended Bit 17 ... Bit 0														R T R
	0	0	0	0	1	1	1	1	1	1	1			1	1	0	0	0	0	0	0	0	0	0	0	0	0	
	0	1			F			C				0				0		0			0							

Legend:

SOF = **S**tart **o**f frame

Edge of recessive to dominant

RTR = **R**emote **t**ransmission **r**equ^{est}

dominant: This message sends data

recessive: This message requests data

IDE = **I**dentifier **e**xtension **f**lag

dominant: After this control bits follows

recessive: After this the second part of the 29 bits identifier follows

SRR = **S**ubstitute **r**emote **r**equ^{est}

recessive: Extended CAN-ID: Replaces the RTR bit at this position

Summary CAN / CANopen

3956

- The COB ID of the network variables must differ from the CANopen Device ID in the controller configuration and from the IDs of the FBs CANx_TRANSMIT and CANx_RECEIVE!
- If more than 8 bytes of network variables are put into one COB ID, CANopen automatically expands the data packet to several successive COB IDs. This can lead to conflicts with manually defined COB IDs!
- Network variables cannot transport any string variables.
- Network variables can be transported...
 - if a variable becomes TRUE (Event),
 - in case of data changes in the network variable or
 - cyclically when the timer has elapsed.
- The interval time is the period between transmissions if cyclical transmission has been selected. The minimum distance is the waiting time between two transmissions, if the variable changes too often.
- To reduce the bus load, split the messages via network variables or CANx_TRANSMIT to several plc cycles using several events.
- Each call of CANx_TRANSMIT or CANx_RECEIVE generates a message packet of 8 bytes.
- In the controller configuration the values for [Com Cycle Period] and [Sync. Window Length] should be identical. These values must be higher than the plc cycle time.
- If [Com Cycle Period] is selected for a slave, the slave searches for a Sync object of the master during exactly this period. This is why the value for [Com Cycle Period] must be higher than the [Master Synch Time].
- We recommend to select "optional startup" for slaves and "automatic startup" for the network. This reduces unnecessary bus load and allows a briefly lost slave to integrate into the network again.
- Since we have no inhibit timer, we recommend to set analogue inputs to "synchronous transmission" to avoid bus overload.
- Binary inputs, especially the irregularly switching ones, should best be set to "asynchronous transmission" using an event timer.
- To be considered during the monitoring of the slave status:
 - after the start of the slaves it takes a while until the slaves are operational.
 - When the system is switched off, slaves can indicate an incorrect status change due to early voltage loss.

9.3.2 Data reception

1169

In principle the received data objects are automatically stored in a buffer (i.e. without influence of the user).

Each identifier has such a buffer (queue). Depending on the application software this buffer is emptied according to the FiFo principle (**F**irst **I**n, **F**irst **O**ut) via CANx_RECEIVE (→ page [145](#)).

9.3.3 Data transmission

1170

By calling CANx_TRANSMIT (→ page [143](#)) the application program transfers exactly one CAN message to the CAN controller. As feedback you are informed whether the message was successfully transferred to the CAN controller. Which then automatically carries out the actual transfer of the data on the CAN bus.

The transmit order is rejected if the controller is not ready because it is in the process of transferring a data object. The transmit order must then be repeated by the application program. This information is indicated by a bit.

If several CAN messages are ready for transmission, the message with the lowest ID is transmitted first. Therefore, the programmer must assign the CAN ID (→ page [126](#)) very carefully.

9.4 Description of the CAN standard program units

Contents

CAN1_BAUDRATE (FB)	130
CAN1_DOWNLOADID (FB)	132
CAN1_EXT (FB)	134
CAN1_EXT_TRANSMIT (FB)	136
CAN1_EXT_RECEIVE (FB)	138
CAN1_EXT_ERRORHANDLER (FB)	140
CAN2 (FB)	141
CANx_TRANSMIT (FB)	143
CANx_RECEIVE (FB)	145
CANx_RECEIVE_RANGE (FB)	147
CANx_EXT_RECEIVE_ALL (FB)	150
CANx_ERRORHANDLER (FB)	152

1186

The CAN FBs are described for use in the application program.

NOTE

To use the full capacity of CAN it is absolutely necessary for the programmer to define an exact **bus concept** before starting to work:

- How many data objects are needed with what identifiers?
- How is the **ecomatmobile** device to react to possible CAN errors?
- How often must data be transmitted? CANx_TRANSMIT (→ page 143) and CANx_RECEIVE (→ page 145) must be called accordingly.
- Check whether the transmit orders were successfully assigned to CANx_TRANSMIT (output RESULT) or ensure that the received data is read from the data buffer of the queue using CANx_RECEIVE and processed in the rest of the program immediately.

To be able to set up a communication connection, the same transmission rate (baud rate) must first be set for all participants of the CAN network. For the controller this is done using CAN1_BAUDRATE (→ page 130) (for the 1st CAN interface) or via CAN2 (→ page 141) (for the 2nd CAN interface).

Irrespective of whether the devices support one or several CAN interfaces the FBs related to the interface are specified by a number in the CAN FB (e.g. CAN1_TRANSMIT or CAN2_RECEIVE). To simplify matters the designation (e.g. CANx_TRANSMIT) is used for all variants in the documentation.

NOTE

When installing the **ecomatmobile** CD "Software, Tools and Documentation", projects with templates have been stored in the program directory of your PC:

```
...\\ifm electronic\\CoDeSys V...\\Projects\\Template_CDVxxyyzz
```

- Open the requested template in CoDeSys via:
[File] > [New from template...]
- > CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page 65)

In this example data objects are exchanged with other CAN participants via the identifiers 1 and 2. To do so, a receive identifier must exist for the transmit identifier (or vice versa) in the other participant.

9.4.1 CAN1_BAUDRATE (FB)

651

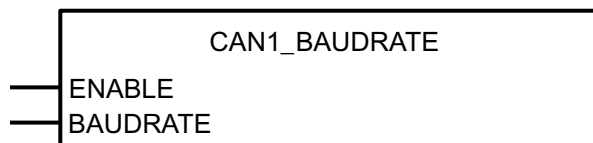
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

654

CAN1_BAUDRATE sets the transmission rate for the bus participant.

- To do so, the corresponding value in kbits/s is entered at the input BAUDRATE.
- > After executing the FB the new value is stored in the device and will even be available after a power failure.

ATTENTION

Please note for CR250n, CR0301, CR0302 and CS0015:

The EEPROM memory module may be destroyed by the permanent use of this unit!

- Only carry out the unit **once** during initialisation in the first program cycle!
- Afterwards block the unit again with ENABLE = FALSE!

NOTE

The new baud rate will become effective on RESET (voltage OFF/ON or soft reset).

ExtendedController: In the slave module, the new baud rate will become effective after voltage OFF/ON.

Parameters of the inputs

655

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
BAUDRATE	WORD	Baud rate [kbits/s] permissible values: 50, 100, 125, 250, 500, 1000 preset value = 125 kbits/s

9.4.2 CAN1_DOWNLOADID (FB)

645

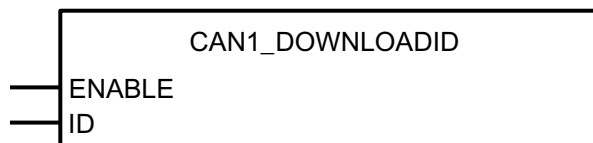
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

648

CAN1_DOWNLOADID sets the download identifier for the first CAN interface.

Using the FB the communication identifier for the program download and for debugging can be set. The new value is entered when the input ENABLE is set to TRUE. The new download ID will become effective after voltage OFF/ON or after a soft reset.

ATTENTION

Please note for CR250n, CR0301, CR0302 and CS0015:

The EEPROM memory module may be destroyed by the permanent use of this unit!

- ▶ Only carry out the unit **once** during initialisation in the first program cycle!
- ▶ Afterwards block the unit again with ENABLE = FALSE!

NOTE

Make sure that a different download ID is entered for each device in the same network!

If the device is operated in the CANopen network, the download ID must not coincide with any module ID (node number) of the other participants, either!

ExtendedController: In the slave module the download ID becomes effective after voltage OFF/ON.

Parameters of the inputs

649

Parameter	Data type	Description
ENABLE	BOOL	TRUE (or only 1 cycle): ID is set FALSE: unit is not executed
ID	BYTE	download identifier permissible values: 1...127

9.4.3 CAN1_EXT (FB)

4192

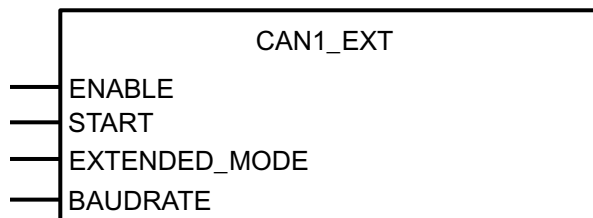
Contained in the library:

ifm_CAN1_EXT_Vxxxxxx.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4333

CAN1_EXT initialises the first CAN interface for the extended identifier (29 bits).

The FB has to be retrieved if the first CAN interface e.g. with the function libraries for SAE J1939 (→ page [154](#)) is to be used.

A change of the baud rate will become effective after voltage OFF/ON. The baud rates of CAN 1 and CAN 2 can be set differently.

The input START is only set for one cycle during reboot or restart of the interface.

NOTE

The FB must be executed **before** CAN1_EXT_... .

Parameters of the inputs

4334

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
START	BOOL	TRUE (in the 1st cycle): interface is initialised FALSE: initialisation cycle completed
EXTENDED_MODE	BOOL	TRUE: identifier of the 1st CAN interface operates with 29 bits FALSE: identifier of the 1st CAN interface operates with 11 bits
BAUDRATE	WORD	baud rate [kbits/s] permissible values = 50, 100, 125, 250, 500, 1000 preset value = 125 kbits/s

9.4.4 CAN1_EXT_TRANSMIT (FB)

4307

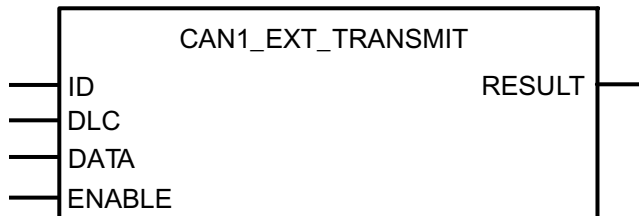
Contained in the library:

`ifm_CAN1_EXT_Vxxxxxxx.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4337

CAN1_EXT_TRANSMIT transfers a CAN data object (message) to the CAN controller for transmission.

The FB is called for each data object in the program cycle; this is done several times in case of long program cycles. The programmer must ensure by evaluating the output RESULT that his transmit order was accepted. To put it simply, at 125 kbits/s one transmit order can be executed per 1 ms.

The execution of the FB can be temporarily blocked via the input ENABLE = FALSE. This can, for example, prevent a bus overload.

Several data objects can be transmitted virtually at the same time if a flag is assigned to each data object and controls the execution of the FB via the ENABLE input.

NOTE

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page [134](#)).

Parameters of the inputs

4380

Parameter	Data type	Description
ID	DWORD	number of the data object identifier permissible values: 11-bit ID = 0...2 047, 29-bit ID = 0...536 870 911
DLC	BYTE	number of bytes to be transmitted from the array DATA permissible values = 0...8
DATA	ARRAY[0...7] OF BYTE	the array contains max. 8 data bytes
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active

Parameters of the outputs

614

Parameter	Data type	Description
RESULT	BOOL	TRUE (only 1 cycle): the unit has accepted the transmit order

9.4.5 CAN1_EXT_RECEIVE (FB)

4302

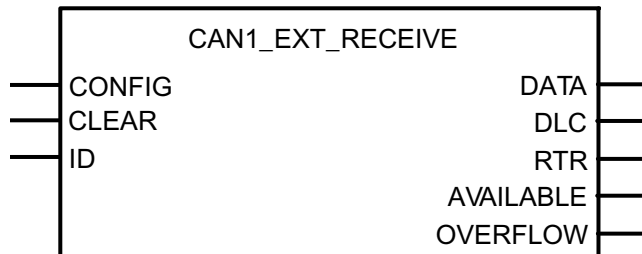
Contained in the library:

`ifm_CAN1_EXT_Vxyzzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4336

CAN1_EXT_RECEIVE configures a data receive object and reads the receive buffer of the data object.

The FB must be called once for each data object during initialisation to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CAN1_EXT_RECEIVE is called for reading the corresponding receive buffer, this is done several times in case of long program cycles. The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed.

Each call of the FB decrements the byte AVAILABLE by 1. If the value of AVAILABLE is 0, there is no data in the buffer.

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE at least 1 data object has been lost.

❗ NOTE

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page [134](#)).

Parameters of the inputs

2172

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only for 1 cycle): configure data object FALSE: this function is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed
ID	WORD	number of the data object identifier permissible values normal frame = 0...2 047 (2^{11}) permissible values extended frame = 0...536 870 912 (2^{29})

Parameters of the outputs

632

Parameter	Data type	Description
DATA	ARRAY[0...7] OF BYTES	the array contains a maximum of 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
RTR	BOOL	not supported
AVAILABLE	BYTE	number of received messages
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

9.4.6 CAN1_EXT_ERRORHANDLER (FB)

4195

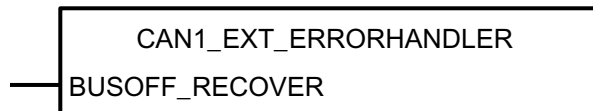
Contained in the library:

ifm_CAN1_EXT_Vxyyz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4335

CAN1_EXT_ERRORHANDLER monitors the first CAN interface and evaluates the CAN errors. If a certain number of transmission errors occurs, the CAN participant becomes error passive. If the error frequency decreases, the participant becomes error active again (= normal condition).

If a participant already is error passive and still transmission errors occur, it is disconnected from the bus (= bus off) and the error bit CANx_BUSOFF is set. Returning to the bus is only possible if the "bus off" condition has been removed (signal BUSOFF_RECOVER).

Afterwards, the error bit CANx_BUSOFF must be reset in the application program.

NOTE

If the automatic bus recover function is to be used (default setting) CAN1_EXT_ERRORHANDLER must **not** be integrated and instanced in the program!

Parameters of the inputs

2177

Parameter	Data type	Description
BUSOFF_RECOVER	BOOL	<p>TRUE (only for 1 cycle):</p> <ul style="list-style-type: none"> > reboot of the CAN interface x > remedy "bus off" status <p>FALSE: this function is not executed</p>

9.4.7 CAN2 (FB)

639

(can only be used for devices with a 2nd CAN interface)

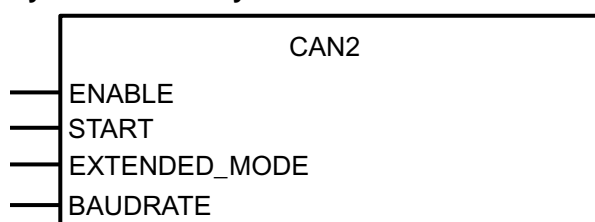
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

642

CAN2 initialises the 2nd CAN interface.

The FB must be called if the 2nd CAN interface is to be used.

A change of the baud rate will become effective after voltage OFF/ON. The baud rates of CAN 1 and CAN 2 can be set differently.

The input START is only set for one cycle during reboot or restart of the interface.

For the 2nd CAN interface the libraries for SAE J1939 (→ page [154](#)) and ISO 11992, among others, are available. The FBs to ISO 11992 are only available in the CR2501 on the 2nd CAN interface.

NOTE

The FB must be executed **before** CAN2... .

Parameters of the inputs

643

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
START	BOOL	TRUE (in the 1st cycle): interface is initialised FALSE: initialisation cycle completed
EXTENDED_MODE	BOOL	TRUE: identifier of the 2nd CAN interface operates with 29 bits FALSE: identifier of the 2nd CAN interface operates with 11 bits
BAUDRATE	WORD	Baud rate [kbits/s] permissible values: 50, 100, 125, 250, 500, 800, 1000 preset value = 125 kbits/s

9.4.8 CANx_TRANSMIT (FB)

609

x = number 1...n of the CAN interface (depending on the device, → data sheet)

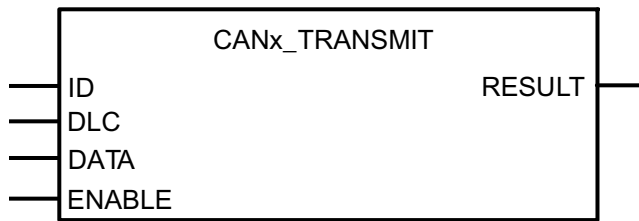
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- **POU not for safety signals!**
(For safety signals → CAN_SAFETY_TRANSMIT (→ page [237](#)))
- SmartController: CR25nn

Symbol in CoDeSys:



Description

612

CANx_TRANSMIT transmits a CAN data object (message) to the CAN controller for transmission.

The FB is called for each data object in the program cycle, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the FB output RESULT that his transmit order was accepted. Simplified it can be said that at 125 kbits/s one transmit order can be executed per ms.

The execution of the FB can be temporarily blocked (ENABLE = FALSE) via the input ENABLE. So, for example a bus overload can be prevented.

Several data objects can be transmitted virtually at the same time if a flag is assigned to each data object and controls the execution of the FB via the ENABLE input.

NOTE

If CAN2_TRANSMIT is to be used, the second CAN interface must be initialised first using CAN2 (→ page [141](#)).

Parameters of the inputs

613

Parameter	Data type	Description
ID	WORD	number of the data object identifier permissible values = 0...2 047
DLC	BYTE	number of bytes to be transmitted from the array DATA permissible values = 0...8
DATA	ARRAY[0...7] OF BYTES	the array contains a maximum of 8 data bytes
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active

Parameters of the outputs

614

Parameter	Data type	Description
RESULT	BOOL	TRUE (only 1 cycle): the unit has accepted the transmit order

9.4.9 CANx_RECEIVE (FB)

627

x = number 1...n of the CAN interface (depending on the device, → data sheet)

Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

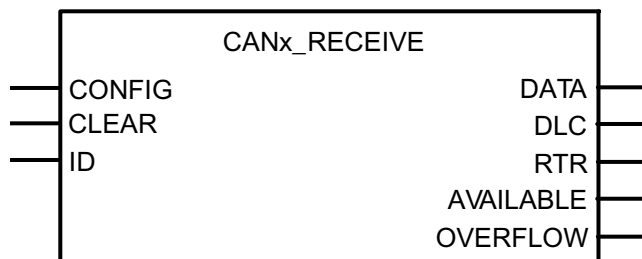
Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

POU not for safety signals!

(For safety signals → CAN_SAFETY_RECEIVE (→ page [239](#)))

Symbol in CoDeSys:



Description

630

CANx_RECEIVE configures a data receive object and reads the receive buffer of the data object.

The FB must be called once for each data object during initialisation, in order to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CANx_RECEIVE is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed.

Each call of the FB decrements the byte AVAILABLE by 1. If the value of AVAILABLE is 0, there is no data in the buffer.

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE at least 1 data object has been lost.

NOTE

If CAN2_RECEIVE is to be used, the second CAN interface must be initialised first using CAN2 (→ page [141](#)).

Parameters of the inputs

631

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only 1 cycle): Configure data object FALSE: unit is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed
ID	WORD	number of the data object identifier permissible values = 0...2 047

Parameters of the outputs

632

Parameter	Data type	Description
DATA	ARRAY[0...7] OF BYTES	the array contains a maximum of 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
RTR	BOOL	not supported
AVAILABLE	BYTE	number of received messages
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

9.4.10 CANx_RECEIVE_RANGE (FB)

4179

x = number 1...n of the CAN interface (depending on the device, → data sheet)

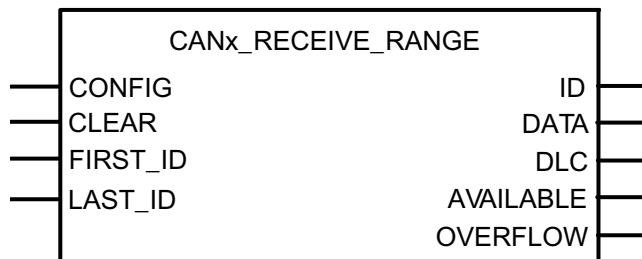
Contained in the library:

from ifm_CRnnnn_V05yyzz.LIB onwards

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- **POU not for safety signals!**
(For safety signals → CAN_SAFETY_RECEIVE (→ page [239](#)))
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2295

CANx_RECEIVE_RANGE configures a sequence of data receive objects and reads the receive buffer of the data objects.

For the first CAN interface max. 2048 IDs per bit are possible.

For the second CAN interface max. 256 IDs per 11 OR 29 bits are possible.

The second CAN interface requires a long initialisation time. To ensure that the watchdog does not react, the process should be distributed to several cycles in the case of bigger ranges.

→ Example (→ page [149](#)).

The FB must be called once for each sequence of data objects during initialisation to inform the CAN controller about the identifiers of the data objects.

The FB must NOT be mixed with CANx_RECEIVE (→ page [145](#)) or CANx_RECEIVE_RANGE for the same IDs at the same CAN interfaces.

In the further program cycle CANx_RECEIVE_RANGE is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer has to ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from buffer SOFORT and are further processed as the data are only available for one cycle.

Each call of the FB decrements the byte AVAILABLE by 1. If the value of AVAILABLE is 0, there is no data in the buffer.

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE, at least 1 data object has been lost.

Receive buffer: max. 16 software buffers per identifier.

Parameters of the inputs

2290

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only for 1 cycle): configure data object FALSE: this function is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed
FIRST_ID	CAN1: WORD CAN2: DWORD	number of the first data object identifier of the sequence permissible values normal frame = 0...2 047 (2^{11}) permissible values extended frame = 0...536 870 912 (2^{29})
LAST_ID	CAN1: WORD CAN2: DWORD	number of the last data object identifier of the sequence permissible values normal frame = 0...2 047 (2^{11}) permissible values extended frame = 0...536 870 912 (2^{29}) LAST_ID has to be bigger than FIRST_ID.

Parameters of the outputs

4381

Parameter	Data type	Description
ID	CAN1: WORD CAN2: DWORD	ID of the transmitted data object
DATA	ARRAY[0...7] OF BYTE	the array contains max. 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
AVAILABLE	BYTE	number of messages in the buffer
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

Example: Initialisation of CANx_RECEIVE_RANGE in 4 cycles

2294

```

PLC_PRG (PRG-ST) (-1/181/-1/88)
0001 PROGRAM PLC_PRG
0002 VAR
0003   init : BOOL := FALSE;
0004   initstep : WORD := 1;
0005   can20 : CAN2;
0006   cr2 : CAN2_RECEIVE_RANGE;
0007   cnt : WORD;
0008 END_VAR
0009
0010 (* CAN2 init *)
0011 can20(ENABLE := TRUE, START := init, EXTENDED_MODE := FALSE, BAUDRATE := 125);
0012
0013 (* CAN2_RECEIVE_RANGE in mehreren Steps initialisieren *)
0014 CASE initstep OF
0015   1:
0016     cr2(CONFIG := TRUE, CLEAR := FALSE, FIRST_ID := 16#100, LAST_ID := 16#10F, ID => , DATA => , DLC => , AVAILABLE => , OVERFLOW => );
0017     initstep := initstep + 1;
0018   2:
0019     cr2(CONFIG := TRUE, CLEAR := FALSE, FIRST_ID := 16#110, LAST_ID := 16#11F, ID => , DATA => , DLC => , AVAILABLE => , OVERFLOW => );
0020     initstep := initstep + 1;
0021   3:
0022     cr2(CONFIG := TRUE, CLEAR := FALSE, FIRST_ID := 16#120, LAST_ID := 16#12F, ID => , DATA => , DLC => , AVAILABLE => , OVERFLOW => );
0023     initstep := initstep + 1;
0024   4:
0025     cr2(CONFIG := TRUE, CLEAR := FALSE, FIRST_ID := 16#130, LAST_ID := 16#13F, ID => , DATA => , DLC => , AVAILABLE => , OVERFLOW => );
0026     initstep := initstep + 1;
0027 ELSE
0028   cr2(CONFIG := FALSE, CLEAR := FALSE, FIRST_ID := 16#100, LAST_ID := 16#100, ID => , DATA => , DLC => , AVAILABLE => , OVERFLOW => );
0029 END_CASE
0030
0031 init := FALSE;
0032
0033 (* Test *)
0034 IF cr2.available > 0 THEN
0035   cnt := cnt + 1;
0036 END_IF

```

9.4.11 CANx_EXT_RECEIVE_ALL (FB)

4183

x = number 1...n of the CAN interface (depending on the device, → data sheet)

Contained in the library:

For CAN interface 1: ifm_CAN1_EXT_Vxxyyzz.LIB

For CAN interface 2...n: ifm_CRnnnn_Vxxyyzz.LIB

Available for the following devices:

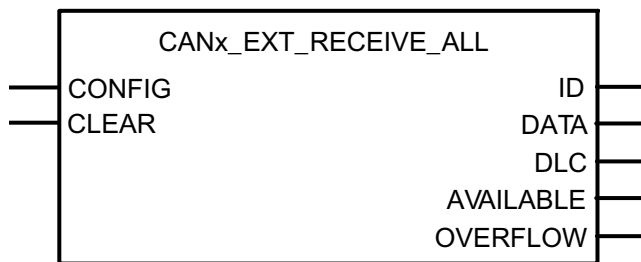
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

POU not for safety signals!

(For safety signals → CAN_SAFETY_RECEIVE (→ page [239](#)))

- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4326

CANx_EXT_RECEIVE_ALL configures all data receive objects and reads the receive buffer of the data objects.

The FB must be called once during initialisation to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CANx_EXT_RECEIVE_ALL is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed.

Each call of the FB decrements the byte AVAILABLE by 1. If the value of AVAILABLE is 0, there is no data in the buffer.

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE at least 1 data object has been lost.

Receive buffer: max. 16 software buffers per identifier.

Parameters of the inputs

4329

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only for 1 cycle): configure data object FALSE: unit is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed

Parameters of the outputs

2292

Parameter	Data type	Description
ID	DWORD	ID of the transmitted data object
DATA	ARRAY[0...7] OF BYTE	the array contains max. 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
AVAILABLE	BYTE	number of messages in the buffer
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

9.4.12 CANx_ERRORHANDLER (FB)

633

x = number 1...n of the CAN interface (depending on the device, → data sheet)

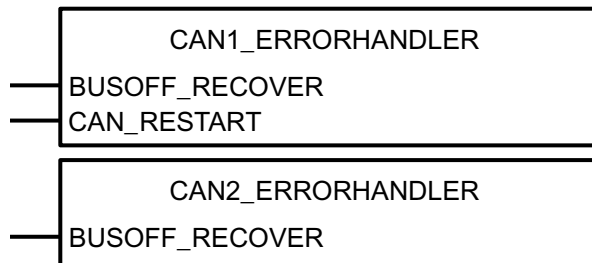
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

636

Error routine for monitoring the CAN interfaces

CANx_ERRORHANDLER monitors the CAN interfaces and evaluates the CAN errors. If a certain number of transmission errors occurs, the CAN participant becomes error passive. If the error frequency decreases, the participant becomes error active again (= normal condition).

If a participant already is error passive and still transmission errors occur, it is disconnected from the bus (= bus off) and the error bit CANx_BUSOFF is set. Returning to the bus is only possible if the "bus off" condition has been removed (signal BUSOFF_RECOVER).

The input CAN_RESTART is used for rectifying other CAN errors. The CAN interface is reinitialised.

Afterwards, the error bit must be reset in the application program.

The procedures for the restart of the interfaces are different:

- For CAN interface 1 or devices with only one CAN interface:
set the input CAN_RESTART = TRUE (only 1 cycle)
- For CAN interface 2:
set the input START = TRUE (only 1 cycle) in CAN2 (→ page [141](#))

NOTE

In principle, CAN2 (→ page [141](#)) must be executed to initialise the second CAN interface, before FBs can be used for it.

If the automatic bus recover function is to be used (default setting) CANx_ERRORHANDLER must **not** be integrated and instanced in the program!

Parameters of the inputs

637

Parameter	Data type	Description
BUSOFF_RECOVER	BOOL	TRUE (only 1 cycle): remedy 'bus off' status FALSE: this function is not executed
CAN_RESTART	BOOL	TRUE (only 1 cycle): completely reinitialise CAN interface 1 FALSE: this function is not executed

9.5 CAN units acc. to SAE J1939

Contents

CAN for the drive engineering	154
Units for SAE J1939	158

7482

The network protocol SAE J1939 describes the communication on a CAN bus in utility vehicles for submitting diagnosis data (e.g. motor speed, temperature) and control information.

9.5.1 CAN for the drive engineering

Contents

Identifier acc. to SAE J1939	155
Example: detailed message documentation	156
Example: short message documentation	157

7678

With the standard SAE J1939 the CiA bietet offers to the user a CAN bus protocol for the drive engineering. For this protocol the CAN controller of the 2nd interface is switched to the "extended mode". This means that the CAN messages are transferred with a 29-bit identifier. Due to the longer identifier numerous messages can be directly assigned to the identifier.

For writing the protocol this advantage was used and certain messages were combined in ID groups. The ID assignment is specified in the standards SAE J1939 and ISO 11992. The protocol of ISO 11992 is based on the protocol of SAE J1939.

Standard	Application area
SAE J1939	Drive management
ISO 11992	"Truck & Trailer Interface"

The 29-bit identifier consists of two parts:

- an 11-bit ID and
- an 18-bit ID.

As for the software protocol the two standards do not differ because ISO 11992 is based on SAE J1939. Concerning the hardware interface, however, there is one difference: higher voltage level for ISO 11992.

! NOTE

To use the functions to SAE J1939 the protocol description of the aggregate manufacturer (e.g. for motors, gears) is definitely needed. For the messages implemented in the aggregate control device this description must be used because not every manufacturer implements all messages or implementation is not useful for all aggregates.

The following information and tools should be available to develop programs for functions to SAE J1939:

- List of the data to be used by the aggregates
- Overview list of the aggregate manufacturer with all relevant data
- CAN monitor with 29-bit support
- If required, the standard SAE J1939

Identifier acc. to SAE J1939

7675

For the data exchange with SAE J1939 the 29 bit identifiers are determinant. This identifier is pictured schematically as follows:

A	SOF	Identifier 11 bits											SRR	IDE	Identifier 18 bits																		RTR		
		Priority			R	DP	PDU format (PF) 6+2 bits								SRR	IDE	still PF		PDU specific (PS) destination address group extern or proprietary										Source address						
B	SOF	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	RTR						
C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33		
D		28	27	26	25	24	23	22	21	20	19	18			17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			

Legend:

A = CAN extended message format

B = J1939 message format

C = J1939 message bit position

D = CAN 29 bit ID position

SOF = **S**tart **o**f frame

SRR = **S**ubstitute remote **r**equ~~e~~st

IDE = **I**dentifier **e**xtension flag

RTR = **R**emote **t**ransmission **r**equ~~e~~st

PDU = **P**rotocol **D**ata **U**nit

PGN = **P**arameter **G**roup **N**umber = PDU format (PF) + PDU source (PS)

(→ CAN-ID (→ page [126](#)))

To do so, the 3 essentially communication methods with SAE J1939 are to be respected:

- destination specific communication with PDU1 (PDU format 0...239)
- broadcast communication with PDU2 (PDU format 240...255)
- proprietary communication with PDU1 or PDU2

Example: detailed message documentation

7679

ETC1: Electronic Transmission Controller #1 (3.3.5) 0CF00203₁₆

Transmission repetition rate	RPT	10 ms
Data length	LEN	8 Bytes
PDU format	PF	240
PDU specific	PS	2
Default priority	PRIO	3
Data Page	PG	0
Source Address	SA	3
Parameter group number	PGN	00F002 ₁₆
Identifier	ID	0CF00203 ₁₆
Data Field	SRC	The meaning of the data bytes 1...8 is not further described. It can be seen from the manufacturer's documentation.

As in the example of the manufacturer all relevant data has already been prepared, it can be directly transferred to the FBs.

Meaning:

Designation in the manufacturer's documentation	Unit input library function	Example value
Transmission repetition rate	RPT	T#10ms
Data length	LEN	8
PDU format	PF	240
PDU specific	PS	2
Default priority	PRIO	3
Data page	PG	0
Source address / destination address	SA / DA	3
Data field	SRC / DST	array address

Depending on the required function the corresponding values are set. For the fields SA / DA or SRC / DST the meaning (but not the value) changes according to the receive or transmit function.

The individual data bytes must be read from the array and processed according to their meaning.

Example: short message documentation

7680

But even if the aggregate manufacturer only provides a short documentation, the function parameters can be derived from the identifier. In addition to the ID, the "transmission repetition rate" and the meaning of the data fields are also always needed.

If the protocol messages are not manufacturer-specific, the standard SAE J1939 or ISO 11992 can also serve as information source.

Structure of the identifier 0CF00203₁₆:

PRIO, reserved, PG		PF + PS				SA / DA	
0	C	F	0	0	2	0	3

As these values are hexadecimal numbers of which individual bits are sometimes needed, the numbers must be further broken down:

SA / DA		Source / Destination Address (hexadecimal)		Source / Destination Address (decimal)	
0	3	00	03	0	3

PF		PDU format (PF) (hexadecimal)		PDU format (PF) (decimal)	
F	0	0F	00	16	0

PS		PDU specific (PS) (hexadecimal)		PDU specific (PS) (decimal)	
0	2	00	02	0	2

PRIO, reserved, PG		PRIO, reserved, PG (binary)	
0	C	0000	1100

Out of the 8 bits (0C₁₆) only the 5 least significant bits are needed:

Not necessary			Priority			res.	PG
x	x	x	0 ₂	1 ₂	1 ₂	0 ₂	0 ₂
			03 ₁₀			0 ₁₀	0 ₁₀

Further typical combinations for "PRIO, reserve., PG"

18₁₆:

Not necessary			priority			res.	PG
x	x	x	1 ₂	1 ₂	0 ₂	0 ₂	0 ₂
			6 ₁₀			0 ₁₀	0 ₁₀

1C₁₆:

Not necessary			priority			res.	PG
x	x	x	1 ₂	1 ₂	1 ₂	0 ₂	0 ₂
			7 ₁₀			0 ₁₀	0 ₁₀

9.5.2 Units for SAE J1939

Contents

J1939_x (FB).....	159
J1939_x_RECEIVE (FB).....	160
J1939_x_TRANSMIT (FB).....	162
J1939_x_RESPONSE (FB).....	164
J1939_x_SPECIFIC_REQUEST (FB).....	166
J1939_x_GLOBAL_REQUEST (FB).....	168

8566

Here you find funktion blocks of the CAN function for SAE J1939.

J1939_x (FB)

4311

x = number 1...n of the CAN interface (depending on the device, → data sheet)

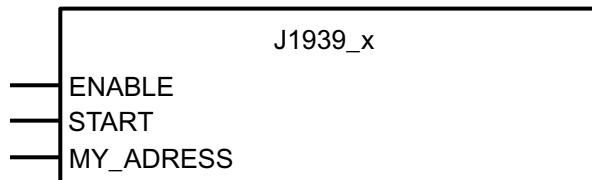
Contained in the library:

ifm_J1939_x_Vxxyyzz.LIB

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

435

J1939_x serves as protocol handler for the communication profile SAE J1939.

To handle the communication, the protocol handler must be called in each program cycle. To do so, the input ENABLE is set to TRUE.

The protocol handler is started if the input START is set to TRUE for one cycle.

Using MY_ADDRESS, a device address is assigned to the controller. It must differ from the addresses of the other J1939 bus participants. It can then be read by other bus participants.

NOTE

J1939 communication via the 1st CAN interface:

- First initialise the interface via CAN1_EXT (→ page [134](#))!

J1939 communication via the 2nd CAN interface:

- Initialise the interface first with CAN2 (→ page [141](#))!

Parameters of the inputs

469

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
START	BOOL	TRUE (only for 1 cycle): protocol handler started FALSE: during further processing of the program
MY_ADDRESS	BYTE	device address of the controller

J1939_x_RECEIVE (FB)

4317

x = number 1...n of the CAN interface (depending on the device, → data sheet)

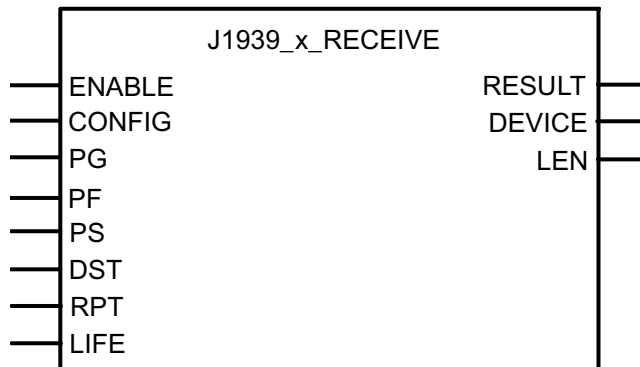
Contained in the library:

`ifm_J1939_x_Vxxyyzz.LIB`

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2288

J1939_x_RECEIVE serves for receiving one individual message or a block of messages.

To do so, the FB must be initialised for one cycle via the input CONFIG. During initialisation, the parameters PG, PF, PS, RPT, LIFE and the memory address of the data array DST are assigned.

- ▶ The address must be determined by means of the operator ADR and assigned to the FB.
- ▶ The receipt of data must be evaluated via the RESULT byte. If RESULT = 1 the data can be read from the memory address assigned via DST and can be further processed.
- > When a new message is received, the data in the memory address DST is overwritten.
- > The number of received message bytes is indicated via the output LEN.
- > If RESULT = 3, no valid messages have been received in the indicated time window (LIFE * RPT).

NOTE

This block must also be used if the messages are requested using J1939_..._REQUEST.

Parameters of the inputs

457

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
CONFIG	BOOL	TRUE (only for 1 cycle): for the configuration of the data object FALSE: during further processing of the program
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
DST	DWORD	target address of the array in which the received data is stored
RPT	TIME	monitoring time Within this time window the messages must be received repeatedly. Otherwise, an error will be signalled. If no monitoring is requested, RPT must be set to T#0s.
LIFE	BYTE	number of permissible faulty monitoring calls

Parameters of the outputs

458

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data has been received 3 = signalling of errors: nothing has been received during the time window (LIFE*RPT)
DEVICE	BYTE	device address of the sender
LEN	WORD	number of bytes received

J1939_x_TRANSMIT (FB)

4324

x = number 1...n of the CAN interface (depending on the device, → data sheet)

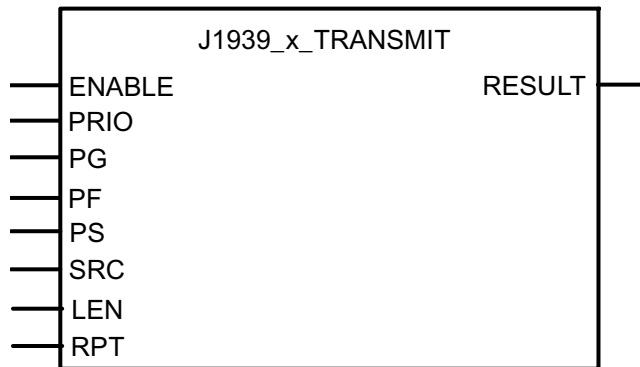
Contained in the library:

`ifm_J1939_x_Vxyyz.LIB`

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2298

J1939_x_TRANSMIT is responsible for transmitting individual messages or blocks of messages. To do so, the parameters PG, PF, PS, RPT and the address of the data array SRC are assigned to the FB.

- The address must be determined by means of the operator ADR and assigned to the FB.
- In addition, the number of data bytes to be transmitted and the priority (typically 3, 6 or 7) must be assigned.

Given that the transmission of data is processed via several control cycles, the process must be evaluated via the RESULT byte. All data has been transmitted if RESULT = 1.

Info

If more than 8 bytes are to be sent, a "multi package transfer" is carried out.

Parameters of the inputs

439

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
PRI0	BYTE	message priority (0...7)
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
SRC	DWORD	memory address of the data array whose content is to be transmitted
LEN	WORD	number of bytes to be transmitted
RPT	TIME	repeat time during which the data messages are transmitted cyclically

Parameters of the outputs

440

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data transmission completed 2 = unit active (data transmission) 3 = error, data cannot be sent

J1939_x_RESPONSE (FB)

4320

x = number 1...n of the CAN interface (depending on the device, → data sheet)

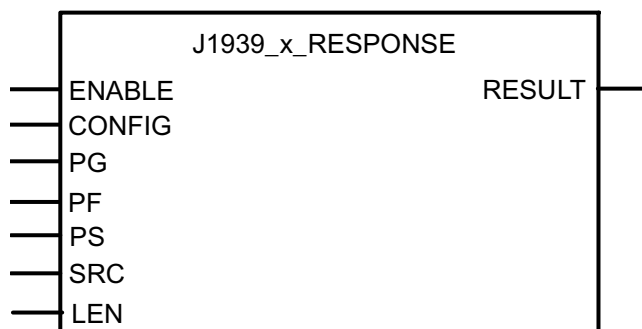
Contained in the library:

ifm_J1939_x_Vxxyyzz.LIB

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2299

J1939_x_RESPONSE handles the automatic response to a request message.

This FB is responsible for the automatic sending of messages to "Global Requests" and "Specific Requests". To do so, the FB must be initialised for one cycle via the input CONFIG.

The parameters PG, PF, PS, RPT and the address of the data array SRC are assigned to the FB.

- The address must be determined by means of the operator ADR and assigned to the FB.
- In addition, the number of data bytes to be transmitted is assigned.

Parameters of the inputs

451

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
CONFIG	BOOL	TRUE (only for 1 cycle): for the configuration of the data object FALSE: during further processing of the program
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
SRC	DWORD	memory address of the data array whose content is to be transmitted
LEN	WORD	number of bytes to be transmitted

Parameters of the outputs

440

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data transmission completed 2 = unit active (data transmission) 3 = error, data cannot be sent

J1939_x_SPECIFIC_REQUEST (FB)

4322

x = number 1...n of the CAN interface (depending on the device, → data sheet)

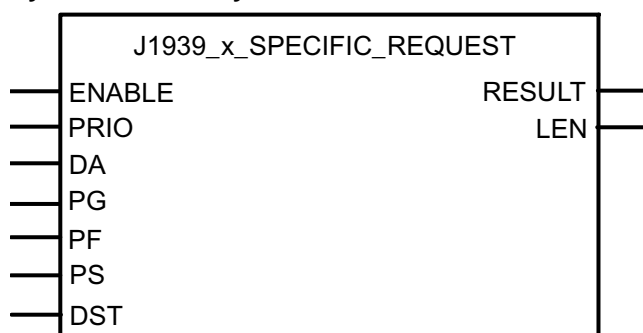
Contained in the library:

ifm_J1939_x_Vxxyyyzz.LIB

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2300

J1939_x_SPECIFIC_REQUEST is responsible for the automatic requesting of individual messages from a specific J1939 network participant. To do so, the logical device address DA, the parameters PG, PF, PS and the address of the array DST in which the received data is stored are assigned to the FB.

- ▶ The address must be determined by means of the operator ADR and assigned to the FB.
 - ▶ In addition, the priority (typically 3, 6 or 7) must be assigned.
 - ▶ Given that the request of data can be handled via several control cycles, this process must be evaluated via the RESULT byte. All data has been received if RESULT = 1.
- > The output LEN indicates how many data bytes have been received.

Parameters of the inputs

445

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
PRI0	BYTE	priority (0...7)
DA	BYTE	logical address (target address) of the called device
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
DST	DWORD	target address of the array in which the received data is stored

Parameters of the outputs

446

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data transmission completed 2 = unit active (data transmission) 3 = error, data cannot be sent
LEN	WORD	number of data bytes received

J1939_x_GLOBAL_REQUEST (FB)

4315

x = number 1...n of the CAN interface (depending on the device, → data sheet)

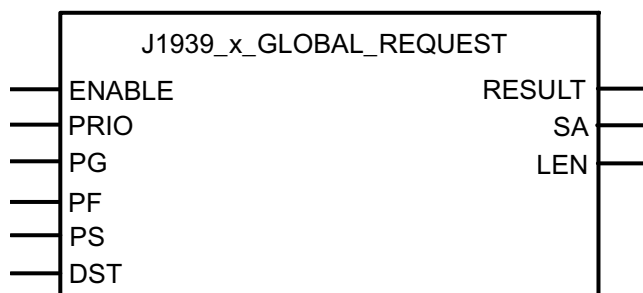
Contained in the library:

ifm_J1939_x_Vxxyyyzz.LIB

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2301

J1939_x_GLOBAL_REQUEST is responsible for the automatic requesting of individual messages from all (global) active J1939 network participants. To do so, the logical device address DA, the parameters PG, PF, PS and the address of the array DST in which the received data is stored are assigned to the FB.

- ▶ The address must be determined by means of the operator ADR and assigned to the FB.
- ▶ In addition, the priority (typically 3, 6 or 7) must be assigned.
- ▶ Given that the request of data can be handled via several control cycles, this process must be evaluated via the RESULT byte. All data has been received if RESULT = 1.
- > The output LEN indicates how many data bytes have been received.

Parameters of the inputs

463

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
PRI0	BYTE	priority (0...7)
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
DST	DWORD	target address of the array in which the received data is stored

Parameters of the outputs

464

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data transmission completed 2 = unit active (data transmission) 3 = error, data cannot be sent
SA	BYTE	logical device address (sender address) of the called device
LEN	WORD	number of data bytes received

9.6 ifm CANopen library

Contents

Technical about CANopen	171
Units for CANopen	204

1856

NOTE

The following devices support CANopen only for the 1st CAN interface:

- Controller CR0020, CR200, CR0301, CR0302, CR0303, CR0505, CR2500, CR2501, CR2502, CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- PDM360smart: CR1070, CR1071

If the CAN master has already been added, the device can no longer be used as a CAN device via CoDeSys.

Implementation of a separate protocol on interface 2 or using the protocol to SAE J1939 or ISO11992 is possible at any time.

The following devices can be used on all CAN interfaces with all protocols:

- BasicController: CR0403
- Controller CR0032, CR0232
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360NG: CR108n

The following devices support not CANopen:

- BasicController: CR0401, CR0402

CANopen network configuration, status and error handling

For all programmable devices the CANopen interface of CoDeSys is used. Whereas the network configuration and parameter setting of the connected devices are directly carried out via the programming software, the error messages can only be reached via nested variable structures in the CANopen stack. The documentation below shows you the structure and use of the network configuration and describes the units of the *ifm* CANopen device libraries.

The chapters CANopen support by CoDeSys (→ page [171](#)), CANopen master (→ page [173](#)), CAN device (→ page [190](#)) and CAN network variables (→ page [198](#)) describe the internal units of the CoDeSys CANopen stacks and their use. They also give information of how to use the network configurator.

The chapters concerning the libraries `ifm_CRnnnn_CANopenMaster_Vxxyyzz.lib` and `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` describe all units for error handling and polling the device status when used as master or slave (CAN device).

NOTE

Irrespective of the device used the structure of the function interfaces of all libraries is the same. The slight differences (e.g. CANOPEN_LED_STATUS) are directly described in the corresponding FBs.

It is absolutely necessary to use only the corresponding device-specific library. The context can be seen from the integrated article number of the device, e.g.:

CR0020: → `ifm_CR0020_CANopenMaster_V040003.lib`

→ chapter Setup the target (→ page [62](#))

When other libraries are used the device can no longer function correctly.

9.6.1 Technical about CANopen

Contents

CANopen support by CoDeSys.....	171
CANopen master.....	173
CAN device	190
CAN network variables.....	198

7773

CANopen support by CoDeSys

1857

General information about CANopen with CoDeSys

2075

CoDeSys is one of the leading systems for programming control systems to the international standard IEC 61131. To make CoDeSys more interesting for users many important functions were integrated in the programming system, among them a configurator for CANopen. This CANopen configurator enables configuration of CANopen networks (with some restrictions) under CoDeSys.

CANopen is implemented as a CoDeSys library in IEC 61131-3. The library is based on simple basic CAN functions called CAN driver.

Implementation of the CANopen functions as CoDeSys library enables simple scaling of the target system. The CANopen function only uses target system resources if the function is really used. To use target system resources carefully CoDeSys automatically generates a data basis for the CANopen master function which exactly corresponds to the configuration.

From the CoDeSys programming system version 2.3.6.0 onwards an **ecomat mobile** controller can be used as CANopen master and slave (CAN device).

❗ NOTE:

For all **ecomat mobile** controllers and the PDM360smart you must use CANopen libraries with the following addition:

- For CR0032 target version up to V01, all other devices up to V04.00.05: "**OptTable**"
- For CR0032 target version from V02 onwards, all other devices from V05 onwards: "**OptTableEx**"

If a new project is created, these libraries are in general automatically loaded. If you add the libraries via the library manager, you must ensure a correct selection.

The CANopen libraries without this addition are used for all other programmable devices (e.g. PDM360compact).

CANopen terms and implementation

1858

According to the CANopen specification there are no masters and slaves in a CAN network. Instead of this there is an NMT master (NMT = network management), a configuration master, etc. according to CANopen. It is always assumed that all participants of a CAN network have equal rights.

Implementation assumes that a CAN network serves as periphery of a CoDeSys programmable controller. As a result of this an *ecomatmobile* controller or a PDM360 display is called CAN master in the CAN configurator of CoDeSys. This master is an NMT master and configuration master. Normally the master ensures that the network is put into operation. The master takes the initiative to start the individual nodes (= network nodes) known via the configuration. These nodes are called slaves.

To bring the master closer to the status of a CANopen node an object directory was introduced for the master. The master can also act as an SDO server (SDO = Service Data Object) and not only as SDO client in the configuration phase of the slaves.

'Addresses' in CANopen

3952

In CANopen there are different types of addresses (IDs):

- COB ID
The **CAN Object Identifier** addresses the message (= the CAN object) in the list of devices. Identical messages have the same COB ID. The COB ID entries in the object directory contain the CAN identifier (CAN ID) among others.
- CAN ID
The **CAN Identifier** identifies CAN messages in the complete network. The CAN ID is part of the COB ID in the object directory.
- Node ID
The **Node Identifier** identifies the CANopen devices in the complete network. The Node ID is part of some predefined CAN IDs (lower 7 bits).

CANopen master

Contents

Differentiation from other CANopen libraries	173
Create a CANopen project	174
Add and configure CANopen slaves	177
Master at runtime	180
Start the network	182
Network states	183

1859

Differentiation from other CANopen libraries

1990

The CANopen library implemented by 3S (Smart Software Solutions) differentiates from the systems on the market in various points. It was not developed to make other libraries of renowned manufacturers unnecessary but was deliberately optimised for use with the CoDeSys programming and runtime system.

The libraries are based on the specifications of CiA DS301, V402.

For users the advantages of the CoDeSys CANopen library are as follows:

- Implementation is independent of the target system and can therefore be directly used on every controller programmable with CoDeSys.
- The complete system contains the CANopen configurator and integration in the development system.
- The CANopen functionality is reloadable. This means that the CANopen FBs can be loaded and updated without changing the operating system.
- The resources of the target system are used carefully. Memory is allocated depending on the used configuration, not for a maximum configuration.
- Automatic updating of the inputs and outputs without additional measures.

The following functions defined in CANopen are at present supported by the **ifm** CANopen library:

- **Transmitting PDOs:** master transmits to slaves (slave = node, device)
Transmitting event-controlled (i.e. in case of a change), time-controlled (RepeatTimer) or as synchronous PDOs, i.e. always when a SYNC was transmitted by the master. An external SYNC source can also be used to initiate transmission of synchronous PDOs.
- **Receiving PDOs:** master receives from slave
Depending on the slave: event-controlled, request-controlled, acyclic and cyclic.
- **PDO mapping**
Assignment between a local object directory and PDOs from/to the CAN device (if supported by the slave).
- **Transmitting and receiving SDOs** (unsegmented, i.e. 4 bytes per entry in the object directory)
Automatic configuration of all slaves via SDOs at the system start.
Application-controlled transmission and reception of SDOs to/from configured slaves.
- **Synchronisation**
Automatic transmission of SYNC messages by the CANopen master.
- **Nodeguarding**
Automatic transmission of guarding messages and lifetime monitoring for every slave configured accordingly.
We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.
- **Heartbeat**
Automatic transmission and monitoring of heartbeat messages.

- **Emergency**
Reception of emergency messages from the configured slaves and message storage.
- Set **Node-ID** and **baud rate** in the slaves
By calling a simple function, node ID and baud rate of a slave can be set at runtime of the application.

The following functions defined in CANopen are at present **not** supported by the CANopen 3S (Smart Software Solutions) library:

- Dynamic identifier assignment,
- Dynamic SDO connections,
- SDO transfer block by block, segmented SDO transfer (the functionality can be implemented via CANx_SDO_READ (→ page 227) and CANx_SDO_WRITE (→ page 229) in the corresponding ifm device library).
- All options of the CANopen protocol which are not mentioned above.

Create a CANopen project

1860

Below the creation of a new project with a CANopen master is completely described step by step. It is assumed that you have already installed CoDeSys on your processor and the Target and EDS files have also been correctly installed or copied.

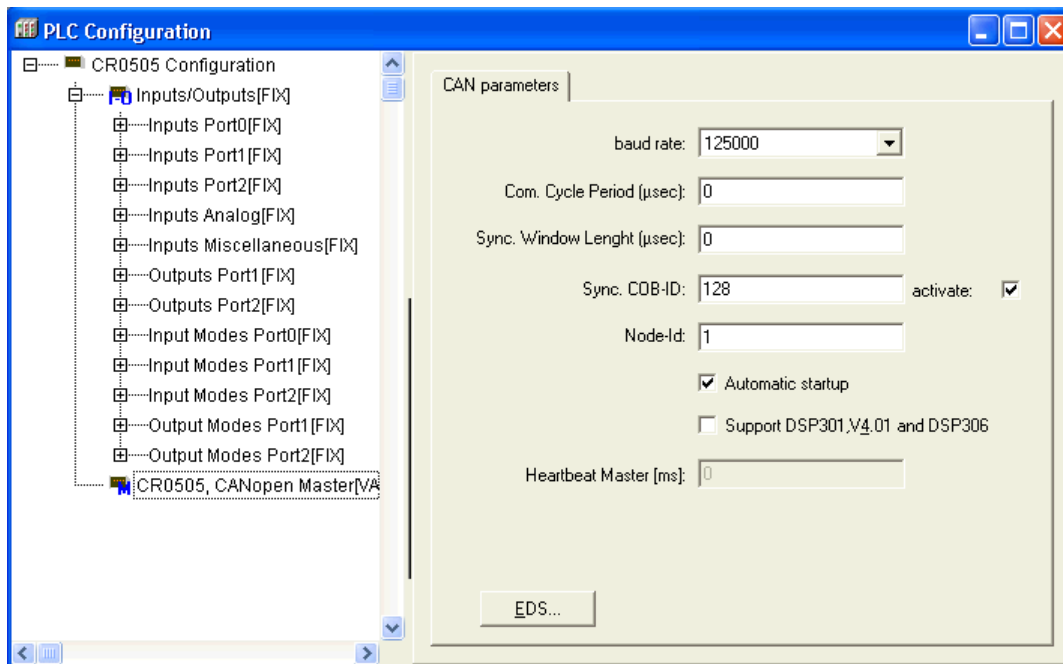
- ▶ A more detailed description for setting and using the dialogue [controller and CANopen configuration] is given in the CoDeSys manual under [Resources] > [PLC Configuration] or in the Online help.
- > After creation of a new project (→ chapter Setup the target (→ page 62)) the CANopen master must first be added to the controller configuration via [Insert] > [Append subelement]. For controllers with 2 or more CAN interfaces interface 1 is automatically configured for the master.
- > The following libraries and software modules are automatically integrated:
 - The `Standard.LIB` which provides the standard functions for the controller defined in IEC 61131.
 - The `3S_CanOpenManager.LIB` which provides the CANopen basic functionalities (possibly `3S_CanOpenManagerOptTable.LIB` for the C167 controller)
 - One or several of the libraries `3S_CANopenNetVar.LIB`, `3S_CANopenDevice.LIB` and `3S_CANopenMaster.LIB` (possibly `3S_...OptTable.LIB` for the C167 controller) depending on the requested functionality
 - The system libraries `SysLibSem.LIB` and `SysLibCallback.LIB`
 - To use the prepared network diagnostic, status and EMCY functions, the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` must be manually added to the library manager. Without this library the network information must be directly read from the nested structures of the CoDeSys CANopen libraries.
- > The following libraries and software modules must still be integrated:
 - The device library for the corresponding hardware, e.g. `ifm_CR0020_Vxxyyzz.LIB`. This library provides all device-specific functions.
 - EDS files for all slaves to be operated on the network. The EDS files are provided for all CANopen slaves by **ifm electronic**. → chapter Set up programming system via templates (→ page 65)
For the EDS files of other manufacturers' nodes contact the corresponding manufacturer.

Tab [CAN parameters]

1967

The most important parameters for the master can be set in this dialogue window. If necessary, the contents of the master EDS file can be viewed via the button [EDS...]. This button is only indicated if the EDS file (e.g. CR0020MasterODEntry.EDS) is in the directory ... \CoDeSys V2.3\Library\PLCConf.

During the compilation of the application program the object directory of the master is automatically generated from this EDS file.



Baud rate

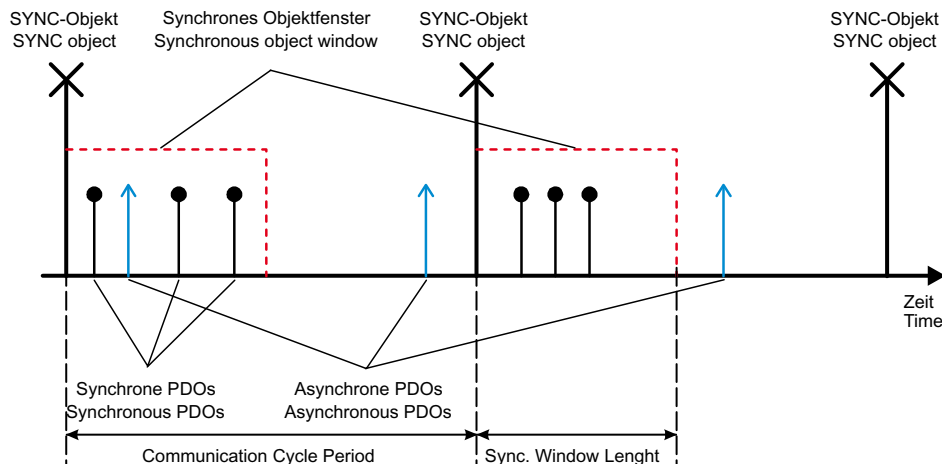
Select the baud rate for the master. It must correspond to the transmission speed of the other network participants.

Communication Cycle Period/Sync. Window Length

After expiry of the [Communication Cycle Period] a SYNC message is transmitted by the master.

The [Sync. Window Length] indicates the time during which synchronous PDOs are transmitted by the other network participants and must be received by the master.

As in most applications no special requirements are made for the SYNC object, the same time can be set for [Communication Cycle Period] and [Sync. Window Length]. Please ensure the time is entered in [μ s] (the value 50 000 corresponds to 50 ms).



Sync. COB ID

In this field the identifier for the SYNC message can be set. It is always transmitted after the communication cycle period has elapsed. The default value is 128 and should normally not be changed. To activate transmission of the SYNC message, the checkbox [activate] must be set.

! NOTE

The SYNC message is always generated at the start of a program cycle. The inputs are then read, the program is processed, the outputs are written to and then all synchronous PDOs are transmitted.

Please note that the SYNC time becomes longer if the set SYNC time is shorter than the program cycle time.

Example: communication cycle period = 10 ms and program cycle time = 30 ms.
The SYNC message is only transmitted after 30 ms.

Node ID

Enter the node number (not the download ID!) of the master in this field. The node number may only occur once in the network, otherwise the communication is disturbed.

Automatic startup

After successful configuration the network and the connected nodes are set to the state [operational] and then started.

If the checkbox is not activated, the network must be started manually.

Heartbeat

If the other participants in the network support heartbeat, the option [support DSP301, V4.01...] can be selected. If necessary, the master can generate its own heartbeat signal after the set time has elapsed.

Add and configure CANopen slaves

Contents

Tab [CAN parameters]	178
Tab [Receive PDO-Mapping] and [Send PDO-Mapping]	179
Tab [Service Data Objects]	179

1861

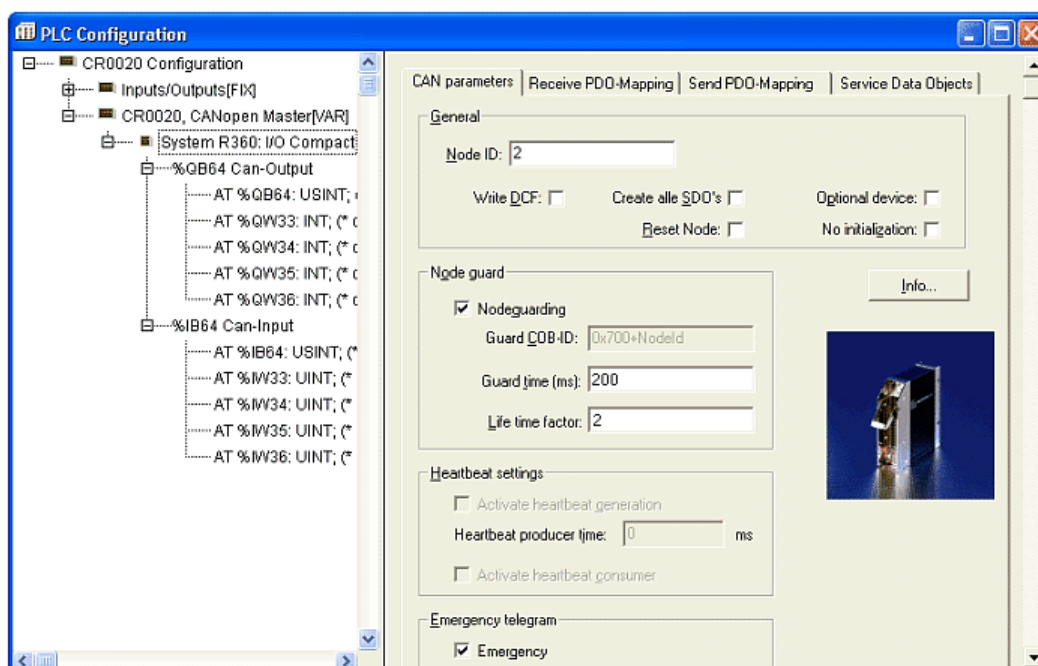
Next you can add the CAN slaves. To do so, you must call again the dialogue in the controller configuration [Insert] > [Append subelement]. A list of the CANopen device descriptions (EDS files) stored in the directory PLC_CONF is available. By selecting the corresponding device it is directly added to the tree of the controller configuration.

NOTE

If a slave is added via the configuration dialogue in CoDeSys, source code is dynamically integrated in the application program for every node. At the same time every additionally inserted slave extends the cycle time of the application program. This means: In a network with many slaves the master can process no further time-critical tasks (e.g. FB OCC_TASK).

A network with 27 slaves has a basic cycle time of 30 ms.

Please note that the maximum time for a PLC cycle of approx. 50 ms should not be exceeded (watchdog time: 100 ms).



Tab [CAN parameters]

1968

Node ID

The node ID is used to clearly identify the CAN module and corresponds to the number on the module set between 1 and 127. The ID is entered decimally and is automatically increased by 1 if a new module is added.

Write DCF

If [Write DCF] is activated, a DCF file is created after adding an EDS file to the set directory for compilation files. The name of the DCF file consists of the name of the EDS file and appended node ID.

Create all SDO's

If this option is activated, SDOs are generated for all communication objects. (Default values are not written again!)

Node reset

The slave is reset ("load") as soon as the configuration is loaded to the controller.

Optional device

If the option [optional device] is activated, the master tries only once to read from this node. In case of a missing response, the node is ignored and the master goes to the normal operating state.

If the slave is connected to the network and detected at a later point in time, it is automatically started. To do so, you must have selected the option [Automatic startup] in the CAN parameters of the master.

No initialization

If this option is activated, the master immediately takes the node into operation without transmitting configuration SDOs. (Nevertheless, the SDO data is generated and stored in the controller.)

Nodeguarding / heartbeat settings

Depending on the device [nodeguarding] and [life time factor] or [heartbeat] must be set.

We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.

Emergency telegram

This option is normally selected. The EMCY messages are transferred with the specified identifier.

Communication cycle

In special applications a monitoring time for the SYNC messages generated by the master can be set here. Please note that this time must be longer than the SYNC time of the master. The optimum value must be determined experimentally, if necessary.

In most cases nodeguarding and heartbeat are sufficient for node monitoring.

Tab [Receive PDO-Mapping] and [Send PDO-Mapping]

1969

With the tabs [Receive PDO-Mapping] and [Send PDO-Mapping] in the configuration dialogue of a CAN module the module mapping (assignment between local object directory and PDOs from/to the CAN device) described in the EDS file can be changed (if supported by the CAN module).

All [mappable] objects of the EDS file are available on the left and can be added to or removed from the PDOs (Process Data Objects) on the right. The [StandardDataTypes] can be added to generate spaces in the PDO.

Insert

With the button [Insert] you can generate more PDOs and insert the corresponding objects. The inputs and outputs are assigned to the IEC addresses via the inserted PDOs. In the controller configuration the settings made can be seen after closing the dialogue. The individual objects can be given symbolic names.

Properties

The PDO properties defined in the standard can be edited in a dialogue via properties.

COB-ID	Every PDO message requires a clear COB ID (communication object identifier). If an option is not supported by the module or the value must not be changed, the field is grey and cannot be edited.
Inhibit Time	The inhibit time (100 µs) is the minimum time between two messages of this PDO so that the messages which are transferred when the value is changed are not transmitted too often. The unit is 100 µs.
Transmission Type	<p>For transmission type you receive a selection of possible transmission modes for this module:</p> <p>acyclic – synchronous After a change the PDO is transferred with the next SYNC.</p> <p>cyclic – synchronous The PDO is transferred synchronously. [Number of SYNCs] indicates the number of the synchronisation messages between two transmissions of this PDO.</p> <p>asynchronous – device profile specific The PDO is transmitted on event, i.e. when the value is changed. The device profile defines which data can be transferred in this way.</p> <p>asynchronous – manufacturer specific The PDO is transmitted on event, i.e. when the value is changed. The device manufacturer defines which data is transferred in this way.</p> <p>(a)synchronous – RTR only These services are not implemented.</p> <p>Number of SYNCs Depending on the transmission type this field can be edited to enter the number of synchronisation messages (definition in the CAN parameter dialogue of [Com. Cycle Period], [Sync Window Length], [Sync. COB ID]) after which the PDO is to be transmitted again.</p> <p>Event-Time Depending on the transmission type the period in milliseconds [ms] required between two transmissions of the PDO is indicated in this field.</p>

Tab [Service Data Objects]

1970

Index, name, value, type and default

Here all objects of the EDS or DCF file are listed which are in the range from index 2000₁₆ to 9FFF₁₆ and defined as writable. Index, name, value, type and default are indicated for every object. The value can be changed. Select the value and press the [space bar]. After the change you can confirm the value with the button [Enter] or reject it with [ESC].

For the initialisation of the CAN bus the set values are transferred as SDOs (Service Data Object) to the CAN module thus having direct influence on the object directory of the CAN slave. Normally they are written again at every start of the application program – irrespective of whether they are permanently stored in the CAN device.

Master at runtime

Contents

Reset of all configured slaves on the bus at the system start.....	180
Polling of the slave device type.....	180
Configuration of all correctly detected devices	180
Automatic configuration of slaves	180
Start of all correctly configured slaves	181
Cyclical transmission of the SYNC message.....	181
Nodeguarding with lifetime monitoring.....	181
Heartbeat from the master to the slaves.....	181
Reception of emergency messages.....	181

8569

Here you find information about the functionality of the CANopen master libraries at runtime.

The CANopen master library provides the CoDeSys application with implicit services which are sufficient for most applications. These services are integrated for users in a transparent manner and are available in the application without additional calls. The following description assumes that the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` was manually added to the library manager to use the network diagnostic, status and EMCY functions.

Services of the CANopen master library:

Reset of all configured slaves on the bus at the system start

8570

To reset the slaves, the NMT command "Reset Remote Node" is used as standard explicitly for every slave separately. (NMT stands for **N**etwork **M**anagement according to CANopen. The individual commands are described in the CAN document DSP301.) In order to avoid overload of slaves having less powerful CAN controllers it is useful to reset the slaves using the command "All Remote Nodes". The service is performed for **all** configured slaves using `CANx_MASTER_STATUS` (→ page [210](#)) with `GLOBAL_START=TRUE`. If the slaves are to be reset **individually**, this input must be set to `FALSE`.

Polling of the slave device type

8571
8021

Polling of the slave device type using SDO (polling for object 100016) and comparison with the configured slave ID:

Indication of an error status for the slaves from which a wrong device type was received. The request is repeated after 0.5 s if ...

- no device type was received
- AND the slave was **not** identified as optional in the configuration
- AND the timeout has **not** elapsed.

Configuration of all correctly detected devices

8572
8022

Every SDO is monitored for a response and repeated if the slave does not respond within the monitoring time.

Automatic configuration of slaves

8573
8023

Automatic configuration of slaves using SDOs while the bus is in operation:
Prerequisite: The slave logged in the master via a bootstrap message.

Start of all correctly configured slaves

8574

Start of all correctly configured slaves after the end of the configuration of the corresponding slave:

To start the slaves the NMT command "Start remote node" is normally used. As for the "reset" this command can be replaced by "Start All Remote Nodes".

The service can be called via `CANx_Master_STATUS` with `GLOBAL_START=TRUE`.

Cyclical transmission of the SYNC message

8575
8025

This value can only be set during the configuration.

Nodeguarding with lifetime monitoring

8576

Setting of nodeguarding with lifetime monitoring for every slave possible:

The error status can be monitored for max. 8 slaves via `CANx_MASTER_STATUS` with `ERROR_CONTROL=TRUE`.

We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.

Heartbeat from the master to the slaves

8577

The error status can be monitored for max. 8 slaves via `CANx_MASTER_STATUS` with `ERROR_CONTROL=TRUE`.

Reception of emergency messages

8578

Reception of emergency messages for every slave, the emergency messages received last are stored separately for every slave:

The error messages can be read via `CANx_MASTER_STATUS` with `EMERGENCY_OBJECT_SLAVES=TRUE`.

In addition this FB provides the EMCY message generated last on the output `GET_EMERGENCY`.

Start the network

1863

Here you find information about how to start the CANopen network.

After downloading the project to the controller or a reset of the application the master starts up the CAN network again. This always happens in the same order of actions:

- All slaves are reset unless they are marked as "No initialization" in the configurator. They are reset individually using the NMT command "Reset Node" (81_{16}) with the node ID of the slave. If the flag GLOBAL_START was set via CANx_MASTER_STATUS (→ page 210), the command is used once with the node ID 0 to start up the network.
- All slaves are configured. To do so, the object 1000_{16} of the slave is polled.
 - If the slave responds within the monitoring time of 0.5 s, the next configuration SDO is transmitted.
 - If a slave is marked as "optional" and does not respond to the polling for object 1000_{16} within the monitoring time, it is marked as not available and no further SDOs are transmitted to it.
 - If a slave responds to the polling for object 1000_{16} with a type other than the configured one (in the lower 16 bits), it is configured but marked as a wrong type.
- All SDOs are repeated as long as a response of the slave was seen within the monitoring time. Here the application can monitor start-up of the individual slaves and possibly react by setting the flag SET_TIMEOUT_STATE in the NODE_STATE_SLAVE array of CANx_MASTER_STATUS.
- If the master configured a heartbeat time unequal to 0, the heartbeat is generated immediately after the start of the master controller.
- After all slaves have received their configuration SDOs, guarding starts for slaves with configured nodeguarding.
- If the master was configured to [Automatic startup], all slaves are now started individually by the master. To do so, the NMT command "Start Remote Node" (1_{16}) is used. If the flag GLOBAL_START was set via CANx_Master_STATUS, the command is used with the node ID 0 and so all slaves are started with "Start all Nodes".
- All configured TX-PDOs are transmitted at least once (for the slaves RX-PDOs).
- If [Automatic startup] is deactivated, the slaves must be started separately via the flag START_NODE in the NODE_STATE_SLAVE array or via the input GLOBAL_START of CANx_MASTER_STATUS.

Network states

Contents

Boot up of the CANopen master	183
Boot up of the CANopen slaves	184
Start-up of the network without [Automatic startup]	186
The object directory of the CANopen master	188

1864

Here you read how to interpret the states of the CANopen network and how to react.

For the start-up (→ page [182](#)) of the CANopen network and during operation the individual functions of the library pass different states.

NOTE

In the monitor mode (online mode) of CoDeSys the states of the CAN network can be seen in the global variable list "CANopen implicit variables". This requires exact knowledge of CANopen and the structure of the CoDeSys CANopen libraries.

To facilitate access CANx_MASTER_STATUS (→ page [210](#)) from the library
ifm_CRnnnn_CANopenMaster_Vxyyzz.LIB is available.

Boot up of the CANopen master

1971

During boot-up of the CAN network the master passes different states which can be read via the output NODE_STATE of CANx_MASTER_STATUS (→ page [210](#)).

State	Description
0, 1, 2	These states are automatically passed by the master and in the first cycles after a PLC start.
3	State 3 of the master is maintained for some time. In state 3 the master configures its slaves. To do so, all SDOs generated by the configurator are transmitted to the slaves one after the other.
5	After transmission of all SDOs to the slaves the master goes to state 5 and remains in this state. State 5 is the normal operating state for the master.

Whenever a slave does not respond to an SDO request (upload or download), the request is repeated. The master leaves state 3, as described above, but not before all SDOs have been transmitted successfully. So it can be detected whether a slave is missing or whether the master has not correctly received all SDOs. It is of no importance for the master whether a slave responds with an acknowledgement or an abort. It is only important for the master whether he received a response at all.

An exception is a slave marked as "optional". Optional slaves are asked for their 1000_n object only once. If they do not respond within 0.5 s, the slave is first ignored by the master and the master goes to state 5 without further reaction of this slave.

Boot up of the CANopen slaves

1972

You can read the states of a slave via the array `NODE_STATE_SLAVE` of `CANx_MASTER_STATUS` (→ page 210). During boot up of the CAN network the slave passes the states -1, 1 and 2 automatically. The states have to be interpreted as follows:

State	Description
-1	The slave is reset by the NMT message "Reset Node" and automatically goes to state 1.
1	After max. 2 s or immediately on reception of its boot up message the slave goes to state 2.
2	After a delay of 0.5 s the slave automatically goes to state 3. This period corresponds to the experience that many CANopen devices are not immediately ready to receive their configuration SDOs after transmission of their boot up message.
3	<p>The slave is configured in state 3. The slave remains in state 3 as long as it has received all SDOs generated by the configurator. It is not important whether during the slave configuration the response to SDO transfers is abort (error) or whether the response to all SDO transfers is no error. Only the response as such received by the slave is important – not its contents.</p> <p>If in the configurator the option "Reset node" has been activated, a new reset of the node is carried out after transmitting the object 1011₁₆ sub-index 1 which then contains the value "load". The slave is then polled again with the upload of the object 1000₁₆.</p> <p>Slaves with a problem during the configuration phase remain in state 3 or directly go to an error state (state > 5) after the configuration phase.</p>

After passing the configuration phase, the slave can go to the following states:

State	Description
4	A node always goes to state 4 except for the following cases: it is an "optional" slave and it was detected as non available on the bus (polling for object 1000 ₁₆) or the slave is present but reacted to the polling for object 1000 ₁₆ with a type in the lower 16 bits other than expected by the configurator.
5	<p>State 5 is the normal operating state of the slave.</p> <p>If the master was configured to "Automatic startup", the slave starts in state 4 (i.e. a "start node" NMT message is generated) and the slave goes automatically to state 5.</p> <p>If the flag <code>GLOBAL_START</code> of <code>CANx_MASTER_STATUS</code> was set by the application, the master waits until all slaves are in state 4. All slaves are then started with the NMT command "Start All Nodes".</p>
97	<p>A node goes to state 97 if it is optional (optional device in the CAN configuration) and has not reacted to the SDO polling for object 1000₁₆.</p> <p>If the slave is connected to the network and detected at a later point in time, it is automatically started. To do so, you must have selected the option "Automatic startup" in the CAN parameters of the master.</p>
98	A node goes to state 98 if the device type (object 1000 ₁₆) does not correspond to the configured type.

If the slave is in state 4 or higher, nodeguard messages are transmitted to the slave if nodeguarding was configured.

Nodeguarding / heartbeat error

1973

State	Description
99	<p>In case of a nodeguarding timeout the variable NODE_STATE in the array NODE_STATE_SLAVE of CANx_MASTER_STATUS (→ page 210) is set to 99.</p> <p>As soon as the node reacts again to nodeguard requests and the option [Automatic startup] is activated, it is automatically started by the master. Depending on the status contained in the response to the nodeguard requests, the node is newly configured or only started.</p> <p>To start the slave manually it is sufficient to use the method "NodeStart".</p>

The same applies to heartbeat errors.

The current CANopen state of a node can be called via the structure element LAST_STATE from the array NODE_STATE_SLAVE of CANx_MASTER_STATUS.

State	Description
0	The node is in the boot up state.
4	The node is in the PREPARED state.
5	The node is in the OPERATIONAL state.
127	The node is in the PRE-OPERATIONAL state.

9.6.2 Start-up of the network without [Automatic startup]

Sometimes it is necessary that the application determines the instant to start the CANopen slaves. To do so, the option [Automatic startup] of the CAN master must be deactivated in the configuration. It is then up to the application to start the slaves.

To start a slave via the application, the structure element `START_NODE` in the array `NODE_STATE_SLAVES` must be set.

The array is assigned to `CANx_MASTER_STATUS` via the ADR operator.

Start-up of the network without [Automatic startup]

Contents	
Starting the network with <code>GLOBAL_START</code>	186
Starting the network with <code>START_ALL_NODES</code>	186
Initialisation of the network with <code>RESET_ALL_NODES</code>	187
Access to the status of the CANopen master	187

8583

Sometimes it is necessary that the application determines the instant to start the CANopen slaves.

Starting the network with `GLOBAL_START`

1974

In a CAN network with many participants (in most cases more than 8) it often happens that NMT messages in quick succession are not detected by all (mostly slow) IO nodes (e.g. CompactModules CR2013). The reason for this is that these nodes must listen to all messages with the ID 0. NMT messages transmitted at too short intervals overload the receive buffer of such nodes.

A help for this is to reduce the number of NMT messages in quick succession.

- ▶ To do so, set the input `GLOBAL_START` of `CANx_Master_STATUS` (→ page 210) to `TRUE` (with [Automatic startup]).
- > The CANopen master library uses the command "Start All Nodes" instead of starting all nodes individually using the command "Start Node".
- > `GLOBAL_START` is executed only once when the network is initialised.
- > If this input is set, the controller also starts nodes with status 98 (see above). However, the PDOs for these nodes remain deactivated.

Starting the network with `START_ALL_NODES`

1975

If the network is not automatically started with `GLOBAL_START` of `CANx_Master_STATUS` (→ page 210), it can be started at any time, i.e. every node one after the other. If this is not requested, the option is as follows:

- ▶ Set the input `START_ALL_NODES` of `CANx_Master_STATUS` to `TRUE`.
`START_ALL_NODES` is typically set by the application program at runtime.
- > If this input is set, nodes with status 98 (see above) are started. However, the PDOs for these nodes remain deactivated.

Initialisation of the network with RESET_ALL_NODES

1976

The same reasons which apply to the command START_ALL_NODES also apply to the NMT command RESET_ALL_NODES (instead of RESET_NODES for every individual node).

- To do so, the input RESET_ALL_NODES of CANx_MASTER_STATUS (→ page [210](#)) must be set to TRUE.
- > This resets all nodes once at the same time.

Access to the status of the CANopen master

1977

You should poll the status of the master so that the application code is not processed before the IO network is ready. The following code fragment example shows one option:

Variable declaration

```
VAR
    FB_MasterStatus := CR0020_MASTER_STATUS;
    :
END_VAR
```

program code

```
If    FB_MasterStatus.NODE_STATE = 5 THEN
    <application code>
END_IF
```

By setting the flag TIME_OUT_STATE in the array NODE_STATE_SLAVE of CANx_Master_STATUS (→ page [210](#)) the application can react and, for example, jump the non configurable node.

The object directory of the CANopen master

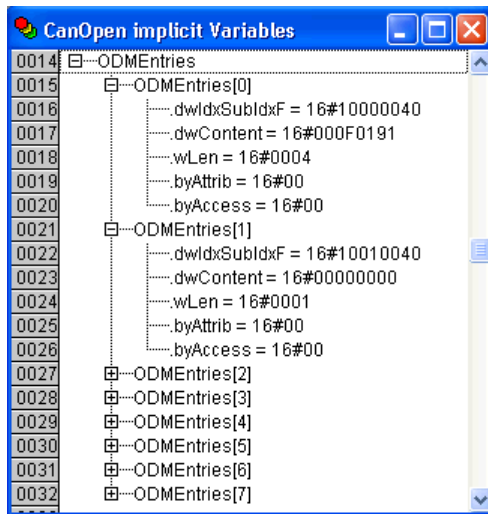
1978

In some cases it is helpful if the CAN master has its own object directory. This enables, for example, the exchange of data of the application with other CAN nodes.

The object directory of the master is generated using an EDS file named `CRnnnnMasterODEntry.EDS` during compilation and is given default values. This EDS file is stored in the directory `CoDeSys Vn\Library\PLCconf`. The content of the EDS file can be viewed via the button [EDS...] in the configuration window [CAN parameters].

Even if the object directory is not available, the master can be used without restrictions.

The object directory is accessed by the application via an array with the following structure:



Structure element	Description
dwIdxSubIdxF	<p>Structure of the component 16#iiii:ss:ff:</p> <p>iiii – index (2 bytes, bits 16...31), ldx</p> <p>ss – sub-index (1 byte, bits 8...15), SubIdx</p> <p>ff – flags (1 byte, bits 0...7), F</p> <p>Meaning of the flag bits:</p> <ul style="list-style-type: none"> bit 0: write bit 1: content is a pointer to an address bit 2: mappable bit 3: swap bit 4: signed value bit 5: floating point bit 6: contains more sub-indices
dwContent	contains the contents of the entry
wLen	length of the data
byAttrib	initially intended as access authorisation can be freely used by the application of the master
byAccess	in the past access authorisation can be freely used by the application of the master

On the platform CoDeSys has no editor for this object directory.

The EDS file only determines the objects used to create the object directory. The entries are always generated with length 4 and the flags (least significant byte of the component of an object directory entry `dwIdxSubIdxF`) are always given the value 1. This means both bytes have the value 41_{16} .

If an object directory is available in the master, the master can act as SDO server in the network. Whenever a client accesses an entry of the object directory by writing, this is indicated to the application via the flag `OD_CHANGED` in `CANx_MASTER_STATUS` (→ page 210). After evaluation this flag must be reset.

The application can use the object directory by directly writing to or reading the entries or by pointing the entries to IEC variables. This means: when reading/writing to another node these IEC variables are directly accessed.

If index and sub-index of the object directory are known, an entry can be addressed as follows:

```
I := GetODMEntryValue(16#iiiiiss00, pCanOpenMaster[0].wODMFirstIdx,
pCanOpenMaster[0].wODMFirstIdx + pCanOpenMaster[0]. wODMCount;
```

For "iii" the index must be used and for "ss" the sub-index (as hex values).

The number of the array entry is available in `I`. You can now directly access the components of the entry.

It is sufficient to enter address, length and flags so that this entry can be directly transferred to an IEC variable:

```
ODMEntries[I].dwContent := ADR(<variable name>);
ODMEntries[I].wLen := sizeof(<variable name>);
ODMEntries[I]. dwIdxSubIdxF := ODMEntries[I]. dwIdxSubIdxF OR
OD_ENTRYFLG_WRITE OR OD_ENTRYFLG_ISPOINTER;
```

It is sufficient to change the content of "dwContent" to change only the content of the entry.

CAN device

Contents

Functionality of the CAN device library	190
CAN device configuration.....	191
Access to the CAN device at runtime.....	197

1865

CAN device is another name for a CANopen slave or CANopen node.

A CoDeSys programmable controller can also be a CANopen slave in a CAN network.

Functionality of the CAN device library

1979

The CAN device library in combination with the CANopen configurator provides the user with the following options:

- In CoDeSys configuration of the properties for nodeguarding/heartbeat, emergency, node ID and baud rate at which the device is to operate.
- Together with the parameter manager in CoDeSys, a default PDO mapping can be created which can be changed by the master at runtime. The PDO mapping is changed by the master during the configuration phase. By means of mapping IEC variables of the application can be mapped to PDOs. This means IEC variables are assigned to the PDOs to be able to easily evaluate them in the application program.
- The CAN device library provides an object directory. The size of this object directory is defined while compiling CoDeSys. This directory contains all objects which describe the CAN device and in addition the objects defined by the parameter manager. In the parameter manager only the list types parameters and variables can be used for the CAN device.
- The library manages the access to the object directory, i.e. it acts as SDO server on the bus.
- The library monitors nodeguarding or the heartbeat consumer time (always only of one producer) and sets corresponding error flags for the application.
- An EDS file can be generated which describes the configured properties of the CAN device so that the device can be integrated and configured as a slave under a CAN master.

The CAN device library explicitly does not provide the following functionalities described in CANopen (all options of the CANopen protocol which are not indicated here or in the above section are not implemented either):

- Dynamic SDO and PDO identifiers
- SDO block transfer
- Automatic generation of emergency messages. Emergency messages must always be generated by the application using `CANx_SLAVE_EMCY_HANDLER` (→ page [218](#)) and `CANx_SLAVE_SEND_EMERGENCY` (→ page [220](#)). To do so, the library `ifm_CRnnnn_CANopenSlave_Vxyyzz.LIB` provides these FBs.
- Dynamic changes of the PDO properties are currently only accepted on arrival of a StartNode NMT message, not with the mechanisms defined in CANopen.

CAN device configuration

Contents

Tab [Base settings].....	191
Tab [CAN settings]	193
Tab [Default PDO mapping]	194
Changing the standard mapping by the master configuration	196

1980

To use the controller as CANopen slave (device) the CANopen slave must first be added via [Insert] > [Append subelement]. For controllers with 2 or more CAN interfaces the CAN interface 1 is automatically configured as a slave. All required libraries are automatically added to the library manager.

Tab [Base settings]

1981

Bus identifier

is currently not used.

Name of updatetask

Name of the task where the CAN device is called.

Generate EDS file

If an EDS file is to be generated from the settings to be able to add the CAN device to any master configuration, the option [Generate EDS file] must be activated and the name of a file must be indicated. As an option a template file can be indicated whose entries are added to the EDS file of the CAN device. In case of overlapping the template definitions are not overwritten.

Example of an object directory

1991

The following entries could for example be in the object directory:

```
[FileInfo]
FileName=D:\CoDeSys\lib2\plcconf\MyTest.eds
FileVersion=1
FileRevision=1
Description=EDS for CoDeSys-Project:
D:\CoDeSys\CANopenTestprojekte\TestHeartbeatODsettings_Device.pro
CreationTime=13:59
CreationDate=09-07-2005
CreatedBy=CoDeSys
ModificationTime=13:59
ModificationDate=09-07-2005
```

```

ModifiedBy=CoDeSys
[DeviceInfo]
VendorName=3S Smart Software Solutions GmbH
ProductName=TestHeartbeatODsettings_Device
ProductNumber=0x33535F44
ProductVersion=1
ProductRevision=1
OrderCode=xxxx.yyyy.zzzz
LMT_ManufacturerName=3S GmbH
LMT_ProductName=3S_Dev
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_100=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1
SimpleBootUpMaster=1
SimpleBootUpSlave=0
ExtendedBootUpMaster=1
ExtendedBootUpSlave=0

```

...

```

[1018sub0]
ParameterName=Number of entries
ObjectType=0x7
DataType=0x5
AccessType=ro
DefaultValue=2
PDOMapping=0
[1018sub1]
ParameterName=VendorID
ObjectType=0x7
DataType=0x7
AccessType=ro
DefaultValue=0x0
PDOMapping=0
[1018sub2]
ParameterName=Product Code
ObjectType=0x7
DataType=0x7
AccessType=ro
DefaultValue=0x0
PDOMapping=0

```

For the meaning of the individual objects please see the CANopen specification DS301.

In addition to the prescribed entries, the EDS file contains the definitions for SYNC, guarding, emergency and heartbeat. If these objects are not used, the values are set to 0 (preset). But as the objects are present in the object directory of the slave at runtime, they are written to in the EDS file.

The same goes for the entries for the communication and mapping parameters. All 8 possible sub-indices of the mapping objects 16xx₁₆ or 1Axx₁₆ are present, but possibly not considered in the sub-index 0.

NOTE: Bit mapping is not supported by the library!

Tab [CAN settings]

1982

Base settings | **CAN settings** | Default PDO mapping

Node id: Device Type:

Baud rate:

☐ Automatic startup

Node guard

☒ Nodeguarding

Guard COB-ID:

Guard time (ms):

Life time factor:

Heartbeat settings

☒ Activate heartbeat generation

Heartbeat producer time: ms

☒ Activate heartbeat consumer

Heartbeat Consumer Time: ms Consumer ID:

Emergency telegram

☒ Emergency

COB-ID:

Here you can set the **node ID** and the **baud rate**.

Device type

(this is the default value of the object 1000₁₆ entered in the EDS) has 191₁₆ as default value (standard IO device) and can be freely changed.

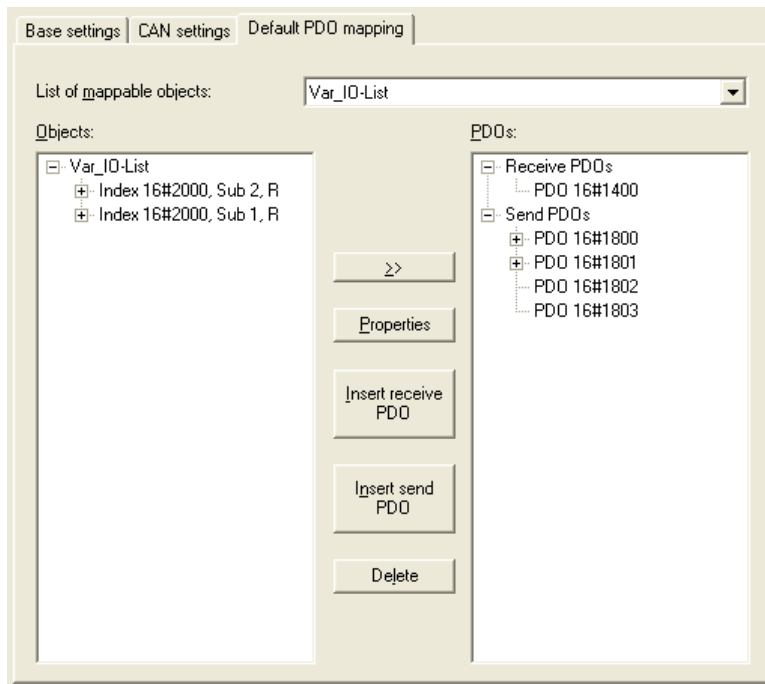
The index of the CAN controller results from the position of the CAN device in the controller configuration.

The **nodeguarding** parameters, the **heartbeat** parameters and the emergency COB ID can also be defined in this tab. The CAN device can only be configured for the monitoring of a heartbeat.

We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.

Tab [Default PDO mapping]

1983

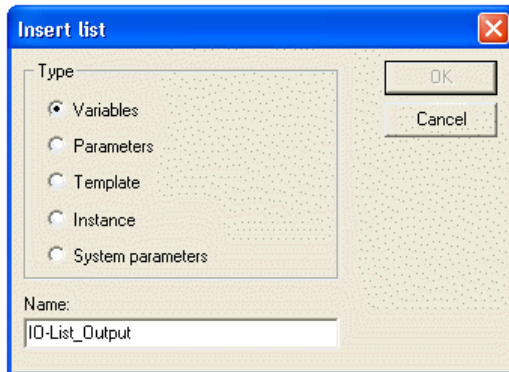


In this tab the assignment between local object directory (OD editor) and PDOs transmitted/received by the CAN device can be defined. Such an assignment is called "mapping".

In the object directory entries used (variable OD) the connection to variables of the application is made between object index/sub-index. You only have to ensure that the sub-index 0 of an index containing more than one sub-index contains the information concerning the number of the sub-indices.

Example list of variables

The data for the variable PLC_PRG.a is to be received on the first receive PDO (COB ID = 512 + node ID) of the CAN device.



Info

[Variables] and [parameters] can be selected as list type.

For the exchange of data (e.g. via PDOs or other entries in the object directory) a variable list is created.

The parameter list should be used if you do not want to link object directory entries to application variables. For the parameter list only the index 1006₁₆ / SubIdx 0 is currently predefined. In this entry the value for the "Com. Cycle Period" can be entered by the master. This signals the absence of the SYNC message.

So you have to create a variable list in the object directory (parameter manager) and link an index/sub-index to the variable PLC_PRG.a.

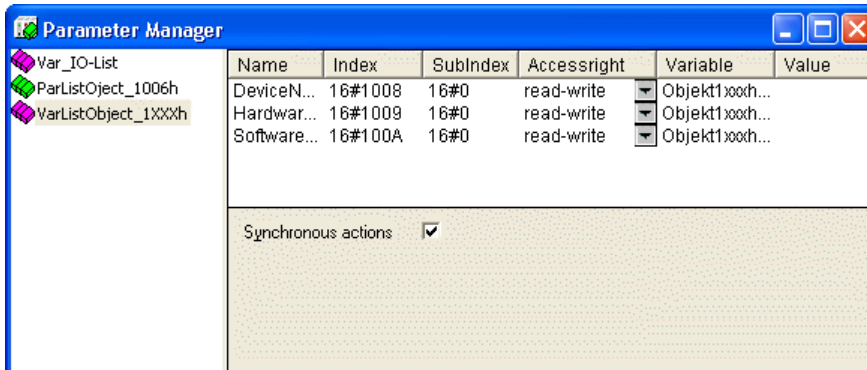
- ▶ To do so, add a line to the variable list (a click on the right mouse button opens the context menu) and enter a variable name (any name) as well as the index and sub-index.
- ▶ The only allowed access right for a receive PDO is [write only].
- ▶ Enter "PLC_PRG.a" in the column [variable] or press [F2] and select the variable.

NOTE

Data to be read by the CAN master (e.g. inputs, system variables) must have the access right [read only].

Data to be written by the CAN master (e.g. outputs in the slave) must have the access right [write only]. SDO parameters to be written and at the same time to be read from and written to the slave application by the CAN master must have the access right [read-write].

To be able to open the parameter manager the parameter manager must be activated in the target settings under [Network functionality]. The areas for index/sub-index already contain sensible values and should not be changed.



In the default PDO mapping of the CAN device the index/sub-index entry is then assigned to a receive PDO as mapping entry. The PDO properties can be defined via the dialogue known from Add and configure CANopen slaves (→ page 177).

Only objects from the parameter manager with the attributes [read only] or [write only] are marked in the possibly generated EDS file as mappable (= can be assigned) and occur in the list of the mappable objects. All other objects are not marked as mappable in the EDS file.

NOTE

If more than 8 data bytes are mapped to a PDO, the next free identifiers are then automatically used until all data bytes can be transferred.

To obtain a clear structure of the identifiers used you should add the correct number of the receive and transmit PDOs and assign them the variable bytes from the list.

Changing the standard mapping by the master configuration

1984

You can change the default PDO mapping (in the CAN device configuration) within certain limits by the master.

The rule applies that the CAN device cannot recreate entries in the object directory which are not yet available in the standard mapping (default PDO mapping in the CAN device configuration). For a PDO, for example, which contains a mapped object in the default PDO mapping no second object can be mapped in the master configuration.

So the mapping changed by the master configuration can at most contain the PDOs available in the standard mapping. Within these PDOs there are 8 mapping entries (sub-indices).

Possible errors which may occur are not displayed, i.e. the supernumerary PDO definitions / supernumerary mapping entries are processed as if not present.

In the master the PDOs must always be created starting from 1400₁₆ (receive PDO communication parameter) or 1800₁₆ (transmit PDO communication parameter) and follow each other without interruption.

Access to the CAN device at runtime

1985

Setting of the node numbers and the baud rate of a CAN device

1986

For the CAN device the node number and the baud rate can be set at runtime of the application program.

- ▶ For setting the **node number** CANx_SLAVE_NODEID (→ page [217](#)) of the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` is used.
- ▶ For setting the **baud rate** CAN1_BAUDRATE (→ page [130](#)) or CAN1_EXT (→ page [134](#)) or CANx of the corresponding device library is used for the controllers and the PDM360smart. For PDM360 or PDM360compact CANx_SLAVE_BAUDRATE is available via the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib`.

Access to the OD entries by the application program

1987

As standard, there are entries in the object directory which are mapped to variables (parameter manager).

However, there are also automatically generated entries of the CAN device which cannot be mapped to the contents of a variable via the parameter manager. Via CANx_SLAVE_STATUS (→ page [223](#)) these entries are available in the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`.

Change the PDO properties at runtime

1988

If the properties of a PDO are to be changed at runtime, this is done by another node via SDO write access as described by CANopen.

As an alternative, it is possible to directly write a new property, e.g. the "event time" of a send PDO and then transmit a command "StartNode-NMT" to the node although it has already been started. As a result of this the device reinterprets the values in the object directory.

Transmit emergency messages via the application program

1989

To transmit an emergency message via the application program CANx_SLAVE_EM CY_HANDLER (→ page [218](#)) and CANx_SLAVE_SEND_EMERGENCY (→ page [220](#)) can be used. The library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` provides these functions.

CAN network variables

Contents

General information.....	198
Configuration of CAN network variables	199
Particularities for network variables	203

1868

General information

2076

Network variables

Network variables are one option to exchange data between two or several controllers. For users the mechanism should be easy to use. At present network variables are implemented on the basis of CAN and UDP. The variable values are automatically exchanged on the basis of broadcast messages. In UDP they are implemented as broadcast messages, in CAN as PDOs. These services are not confirmed by the protocol, i.e. it is not checked whether the receiver receives the message. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

Object directory

The object directory is another option to exchange variables. This is a 1 to 1 connection using a confirmed protocol. The user can check whether the message arrived at the receiver. The exchange is not carried out automatically but via the call of FBs from the application program.

→ chapter The object directory of the CANopen master (→ page [188](#))

Configuration of CAN network variables

Contents

Settings in the target settings.....	199
Settings in the global variable lists	200

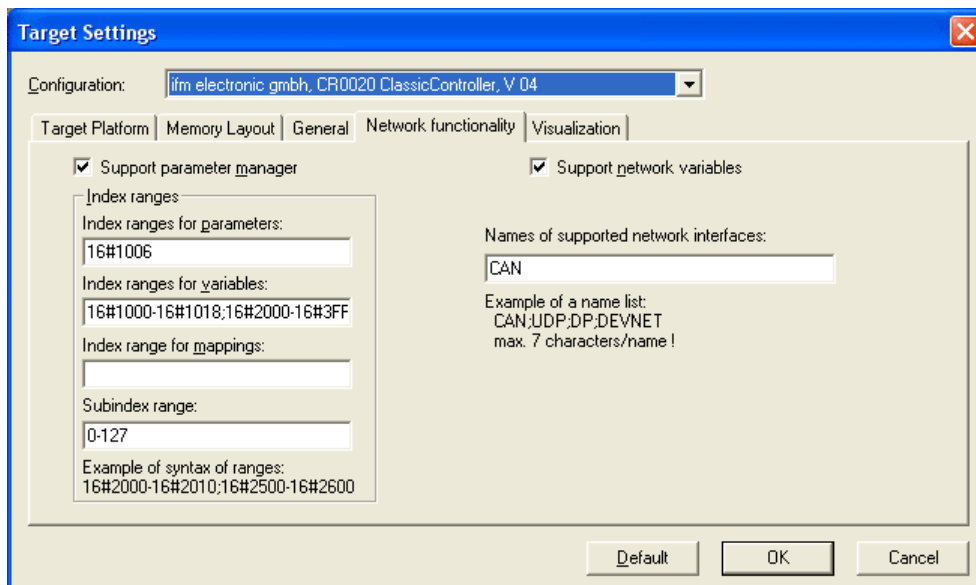
1869

To use the network variables with CoDeSys you need the libraries `3s_CanDrv.lib`, `3S_CANopenManager.lib` and `3S_CANopenNetVar.lib`. You also need the library `SysLibCallback.lib`.

CoDeSys automatically generates the required initialisation code and the call of the network blocks at the start and end of the cycle.

Settings in the target settings

1994

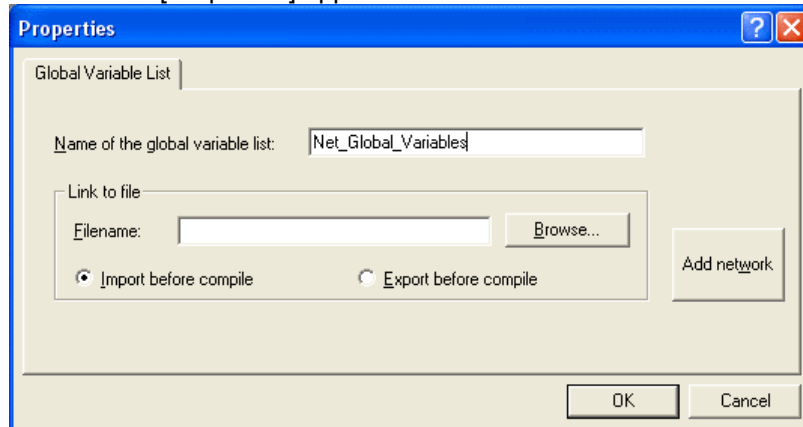


- ▶ Select the dialogue box [Target settings].
- ▶ Select the tab [Network functionality].
- ▶ Activate the check box [Support network variables].
- ▶ Enter the name of the requested network, here CAN, in [Names of supported network interfaces].
- ▶ To use network variables you must also add a CAN master or CAN slave (device) to the controller configuration.
- ▶ Please note the particularities when using network variables for the corresponding device types.
→ Chapter Particularities for network variables (→ page [203](#))

Settings in the global variable lists

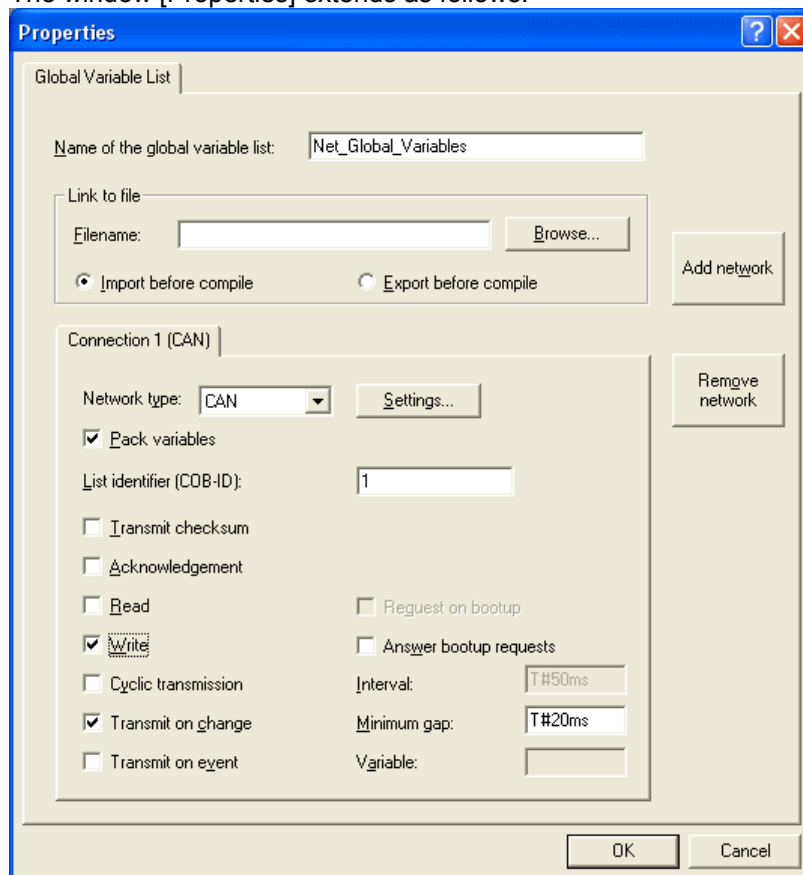
1995

- Create a new global variable list. In this list the variables to be exchanged with other controllers are defined.
- Open the dialogue with the menu point [Object Properties].
- > The window [Properties] appears:



If you want to define the network properties:

- Click the button [Add network].
If you have configured several network connections, you can also configure here several connections per variable list.
- > The window [Properties] extends as follows:



Meaning of the options:

Network type

As network type you can enter one of the network names indicated in the target settings.

If you click on the button [Settings] next to it, you can select the CAN interface:

1. CAN interface: value = 0
 2. CAN interface: value = 1
- etc.

Pack variables

If this option is activated with [v], the variables are combined, if possible, in one transmission unit. For CAN the size of a transmission unit is 8 bytes. If it is not possible to include all variables of the list in one transmission unit, several transmission units are formed for this list.

If the option is not activated, every variable has its own transmission unit.

If [Transmit on change] is configured, it is checked separately for every transmission unit whether it has been changed and must be transmitted.

List identifier (COB-ID)

The basic identifier is used as a unique identification to exchange variable lists of different projects. Variable lists with identical basic identifier are exchanged. Ensure that the definitions of the variable lists with the same basic identifier match in the different projects.

❗ NOTE

In CAN networks the basic identifier is directly used as COB-ID of the CAN messages. It is not checked whether the identifier is also used in the remaining CAN configuration.

To ensure a correct exchange of data between two controllers the global variable lists in the two projects must match. To ensure this you can use the feature [Link to file]. A project can export the variable list file before compilation, the other projects should import this file before compilation.

In addition to simple data types a variable list can also contain structures and arrays. The elements of these combined data types are transmitted separately.

Strings must not be transmitted via network variables as otherwise a runtime error will occur and the watchdog will be activated.

If a variable list is larger than a PDO of the corresponding network, the data is split up to several PDOs. Therefore it cannot be ensured that all data of the variable list is received in **one** cycle. Parts of the variable list can be received in different cycles. This is also possible for variables with structure and array types.

Transmit checksum

This option is not supported.

Acknowledgement

This option is not supported.

Read

The variable values of one (or several) controllers are read.

Write

The variables of this list are transmitted to other controllers.

NOTE

You should only select one of these options for every variable list, i.e. either only read or only write.

If you want to read or write several variables of a project, please use several variable lists (one for reading, one for writing).

In a network the same variable list should only be exchanged between two participants.

Cyclic transmission

Only valid if [write] is activated. The values are transmitted in the specified [interval] irrespective of whether they have changed.

Transmit on change

The variable values are only transmitted if one of the values has been changed. With [Minimum gap] (value > 0) a minimum time between the message packages can be defined.

Transmit on event

If this option is selected, the CAN message is only transmitted if the indicated binary [variable] is set to TRUE. This variable cannot be selected from the list of the defined variables via the input help.

Particularities for network variables

1992

Device	Description
ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506	<p>Network variables are only supported on interface 1 (enter the value 0).</p> <p>CAN master Transmit and receive lists are processed directly. You only have to make the settings described above.</p> <p>CAN device Transmit lists are processed directly. For receive lists you must also map the identifier area in the object directory to receive PDOs. It is sufficient to create only two receive PDOs and to assign the first object the first identifier and the second object the last identifier. If the network variables are only transferred to one identifier, you only have to create one receive PDO with this identifier.</p> <p>Important! Please note that the identifier of the network variables and of the receive PDOs must be entered as decimal value.</p>
ClassicController: CR0032 ExtendedController: CR0232	<p>Network variables are supported on all CAN interfaces. (All other informations as above)</p>
BasicController: CR040n	<p>Network variables are supported on all CAN interfaces. (All other informations as above)</p>
BasicDisplay: CR0451	<p>Network variables are supported only on CAN interface 1 (enter value = 0). (All other informations as above)</p>
PDM360smart: CR1070, CR1071	<p>Only one interface is available (enter value = 0).</p> <p>CAN master Transmit and receive lists are processed directly. You only have to make the settings described above.</p> <p>CAN device Transmit lists are processed directly. For receive lists you must additionally map the identifier area in the object directory to receive PDOs. It is sufficient to create only two receive PDOs and to assign the first object the first identifier and the second object the last identifier. If the network variables are only transferred to one identifier, you only have to create one receive PDO with this identifier.</p> <p>Important! Please note that the identifier of the network variables and of the receive PDOs must be entered as decimal value.</p>
PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056	<p>Network variables are supported on interface 1 (value = 0) and 2 (value = 1).</p> <p>CAN master Transmit and receive lists are processed directly. You only have to make the settings described above.</p> <p>CAN device Transmit and receive lists are processed directly. You only have to make the settings described above.</p> <p>Important! If [support network variables] is selected in the PDM360 or PDM360compact, you must at least create one variable in the global variable list and call it once in the application program. Otherwise the following error message is generated when compiling the program: Error 4601: Network variables 'CAN': No cyclic or freewheeling task for network variable exchange found.</p>
PDM360NG: CR108n	???

9.6.3 Units for CANopen

Contents

Library for the CANopen master	204
Library for the CANopen slave	216
Further ifm libraries for CANopen	226

8587

Library for the CANopen master

Contents

CANx_MASTER_EMCY_HANDLER (FB)	205
CANx_MASTER_SEND_EMERGENCY (FB)	207
CANx_MASTER_STATUS (FB)	210

1870

The library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` provides a number of FBs for the CANopen master which will be explained below.

CANx_MASTER_EMCY_HANDLER (FB)

2006

x = number 1...n of the CAN interface (depending on the device, → data sheet)

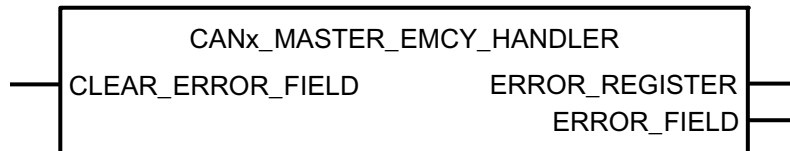
Contained in the library:

`ifm_CRnnnn_CANopenMaster_Vxyyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2009

CANx_MASTER_EMCY_HANDLER monitors the device-specific error status of the master. The FB must be called in the following cases:

- the error status is to be transmitted to the network and
- the error messages of the application are to be stored in the object directory.

NOTE

If application-specific error messages are to be stored in the object directory, CANx_MASTER_EMCY_HANDLER must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY (→ page [207](#)).

Parameters of the inputs

2010

Parameter	Data type	Description
CLEAR_ERROR_FIELD	BOOL	TRUE: deletes the contents of the array ERROR_FIELD FALSE: this function is not executed

Parameters of the outputs

2011

Parameter	Data type	Description
ERROR_REGISTER	BYTE	shows the content of the object directory index 1001 ₁₆ (Error Register)
ERROR_FIELD	ARRAY [0...5] OF WORD	the array [0...5] shows the contents of the object directory index 1003 ₁₆ (Error Field) - ERROR_FIELD[0]: number of stored errors - ERROR_FIELD[1...5]: stored errors, the most recent error is in index [1]

CANx_MASTER_SEND_EMERGENCY (FB)

2012

x = number 1...n of the CAN interface (depending on the device, → data sheet)

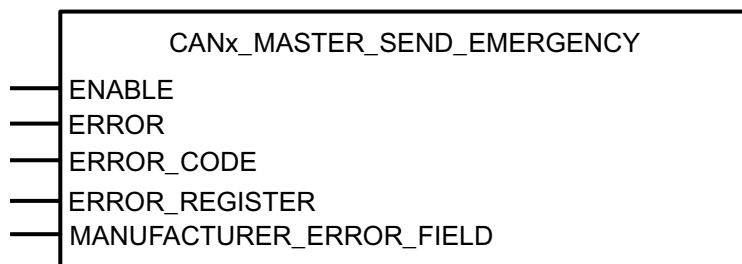
Contained in the library:

`ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2015

CANx_MASTER_SEND_EMERGENCY transmits application-specific error states. The FB is called if the error status is to be transmitted to other devices in the network.

❗ NOTE

If application-specific error messages are to be stored in the object directory, CANx_MASTER_EMCY_HANDLER (→ page [205](#)) must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY.

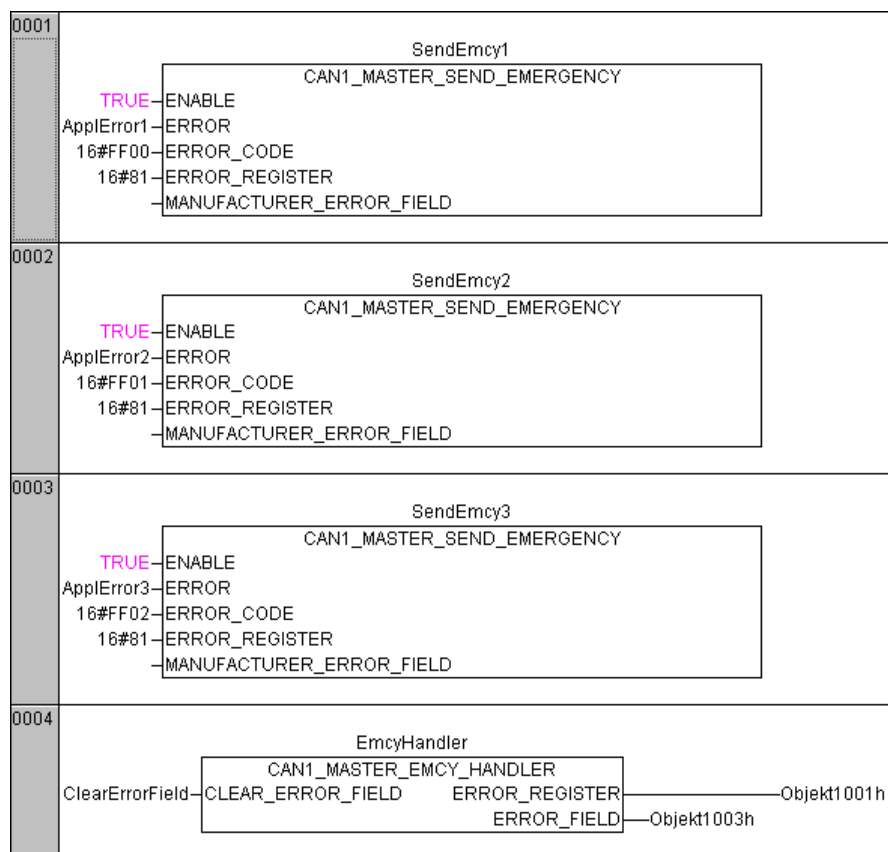
Parameters of the inputs

2016

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
ERROR	BOOL	FALSE → TRUE (edge): transmits the given error code TRUE → FALSE (edge) AND the fault is no longer indicated: the message that there is no error is sent after a delay of approx. 1 s
ERROR_CODE	WORD	The error code provides detailed information about the detected fault. The values should be entered according to the CANopen specification. → chapter Overview CANopen error codes (→ page 247)
ERROR_REGISTER	BYTE	This object reflects the general error state of the CANopen network participant. The values should be entered according to the CANopen specification.
MANUFACTURER_ERROR_FIELD	ARRAY [0...4] OF BYTE	Here, up to 5 bytes of application-specific error information can be entered. The format can be freely selected.

Example: CANx_MASTER_SEND_EMERGENCY

2018



In this example 3 error messages will be generated subsequently:

1. ApplError1, Code = FF00₁₆ in the error register 81₁₆
2. ApplError2, Code = FF01₁₆ in the error register 81₁₆
3. ApplError3, Code = FF02₁₆ in the error register 81₁₆

CAN1_MASTER_EMCY_HANDLER sends the error messages to the error register "Object 1001₁₆" in the error array "Object 1003₁₆".

CANx_MASTER_STATUS (FB)

2021

x = number 1...n of the CAN interface (depending on the device, → data sheet)

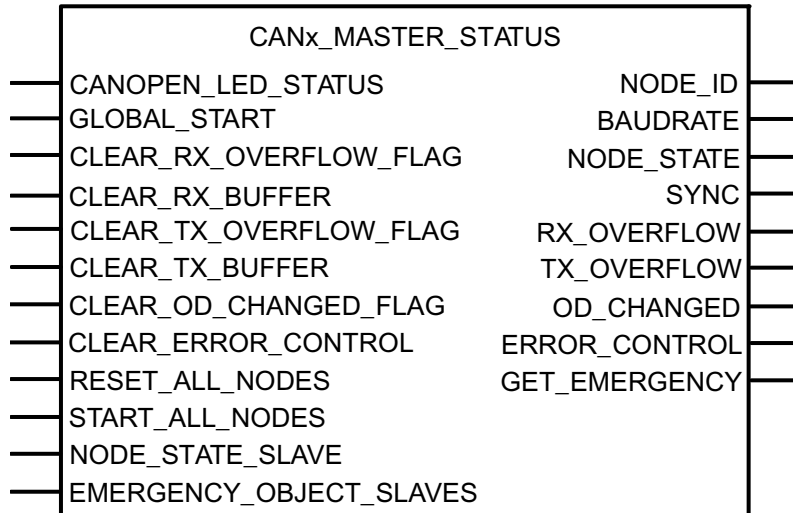
Contained in the library:

`ifm_CRnnnn_CANopenMaster_Vxyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2024

Status indication of the device used with CANopen.

CANx_MASTER_STATUS shows the status of the device used as CANopen master. Furthermore, the status of the network and of the connected slaves can be monitored.

The FB simplifies the use of the CoDeSys CANopen master libraries. We urgently recommend to carry out the evaluation of the network status and of the error messages via this FB.

Parameters of the inputs

2025

Parameter	Data type	Description
CANOPEN_LED_STATUS	BOOL	(input not available for PDM devices) TRUE: the status LED of the controller is switched to the mode "CANopen": flashing frequency 0.5 Hz = pre-operational flashing frequency 2.0 Hz = operational The other diagnostic LED signals are not changed by this operating mode.
GLOBAL_START	BOOL	TRUE: all connected network participants (slaves) are started simultaneously during network initialisation. FALSE: the connected network participants are started one after the other. further information → chapter Starting the network with GLOBAL_START (→ page 186)
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "receive buffer overflow" FALSE: this function is not executed
CLEAR_RX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the receive buffer FALSE: this function is not executed
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "transmit buffer overflow" FALSE: this function is not executed
CLEAR_TX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the transmit buffer FALSE: this function is not executed
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE → TRUE (edge): delete flag "data in the object directory changed" FALSE: this function is not executed
CLEAR_ERROR_CONTROL	BOOL	FALSE → TRUE (edge): delete the guard error list (ERROR_CONTROL) FALSE: this function is not executed
RESET_ALL_NODES	BOOL	FALSE → TRUE (edge): reset all nodes FALSE: this function is not executed
START_ALL_NODES	BOOL	TRUE: All connected network participants (slaves) are started simultaneously at runtime of the application program. FALSE: The connected network participants must be started one after the other further information → chapter Starting the network with START_ALL_NODES (→ page 186)

Parameter	Data type	Description
NODE_STATE_SLAVE	ARRAY [0...MAX_NODEINDEX] STRUCT NODE_STATE	<p>To determine the status of a single network node the global array "NodeStateList" can be used. The array then contains the following elements:</p> <p>NodeStateList[n].NODE_ID: node number of the slave</p> <p>NodeStateList[n].NODE_STATE: current status from the master's point of view</p> <p>NodeStateList[n].LAST_STATE: the CANopen status of the node</p> <p>NodeStateList[n].RESET_NODE: TRUE: reset slave</p> <p>NodeStateList[n].START_NODE: TRUE: start slave</p> <p>NodeStateList[n].PREOP_NODE: TRUE: set slave to pre-operation mode</p> <p>NodeStateList[n].SET_TIMEOUT_STATE: TRUE: set timeout for cancelling the configuration</p> <p>NodeStateList[n].SET_NODE_STATE: TRUE: set new node status</p> <p>example code → chapter Example: CANx_MASTER_STATUS (→ page 214)</p> <p>further information → chapter Master at runtime (→ page 180)</p>
EMERGENCY_OBJECT_SLAVES	ARRAY [0...MAX_NODEINDEX] STRUCT EMERGENCY_MESSAGE	<p>To obtain a list of the most recent occurred error messages of all network nodes the global array "NodeEmergencyList" can be used. The array then contains the following elements:</p> <p>NodeEmergencyList[n].NODE_ID: node number of the slave</p> <p>NodeEmergencyList[n].ERROR_CODE: error code</p> <p>NodeEmergencyList[n].ERROR_REGISTER: error register</p> <p>NodeEmergencyList[n].MANUFACTURER_ERROR_FIELD[0..4]: manufacturer-specific error field</p> <p>further information → chapter Access to the structures at runtime of the application (→ page 215)</p>

Parameters of the outputs

2029

Parameter	Data type	Description
NODE_ID	BYTE	node ID of the master
BAUDRATE	WORD	baud rate of the master
NODE_STATE	INT	current status of the master
SYNC	BOOL	SYNC signal of the master This is set in the tab [CAN parameters] (→ page 175) of the master depending on the set time [Com. Cycle Period].
RX_OVERFLOW	BOOL	error flag "receive buffer overflow"
TX_OVERFLOW	BOOL	error flag "transmit buffer overflow"
OD_CHANGED	BOOL	flag "object directory master was changed"
ERROR_CONTROL	ARRAY [0...7] OF BYTE	The array contains a list (max. 8) of the missing network nodes (guard or heartbeat error). further information → chapter Access to the structures at runtime (See "Access to the structures at runtime of the application" → page 215)
GET_EMERGENCY	STRUCT EMERGENCY_MESSAGE	at the output the data for the structure EMERGENCY_MESSAGE are available the most recent error message of a network node is always displayed To obtain a list of all occurred errors, the array "EMERGENCY_OBJECT_SLAVES" must be evaluated.

Parameters of internal structures

2030

Below are the structures of the arrays used in this FB.

Parameter	Data type	Description
CANx_EMERGENCY_MESSAGE	STRUCT	NODE_ID: BYTE ERROR_CODE: WORD ERROR_REGISTER: BYTE MANUFACTURER_ERROR_FIELD: ARRAY[0..4] OF BYTE The structure is defined by the global variables of the library <code>ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB</code> .
CANx_NODE_STATE	STRUCT	NODE_ID: BYTE NODE_STATE: BYTE LAST_STATE: BYTE RESET_NODE: BOOL START_NODE: BOOL PREOP_NODE: BOOL SET_TIMEOUT_STATE: BOOL SET_NODE_STATE: BOOL The structure is defined by the global variables of the library <code>ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB</code> .

Detailed description of the functionalities of the CANopen master and the mechanisms → chapter CANopen master (→ page [173](#)).

Using the controller CR0020 as an example the following code fragments show the use of CANx_MASTER_STATUS (→ page [210](#)).

Example: CANx_MASTER_STATUS

2031

Slave information

2033

To be able to access the information of the individual CANopen nodes, an array for the corresponding structure must be generated. The structures are contained in the library. You can see them under "Data types" in the library manager.

The number of the array elements is determined by the global variable MAX_NODEINDEX which is automatically generated by the CANopen stack. It contains the number of the slaves minus 1 indicated in the network configurator.

NOTE

The numbers of the array elements do **not** correspond to the node ID. The identifier can be read from the corresponding structure under NODE_ID.

```

0001 PROGRAM MasterStatus
0002 VAR
0003   Status: CR0020_MASTER_STATUS;
0004   LedStatus: BOOL := TRUE;
0005   GlobalStartNodes: BOOL := TRUE;
0006   ClearRxOverflowFlag: BOOL;
0007   ClearRxBuffer: BOOL;
0008   ClearTxOverflowFlag: BOOL;
0009   ClearTxBuffer: BOOL;
0010   ClearOdChanged: BOOL;
0011   ClearErrorControl: BOOL;
0012   ResetAllNodes: BOOL;
0013   StartAllNodes: BOOL;
0014   NodeId: BYTE;
0015   Baudrate: WORD;
0016   NodeState: INT;
0017   Sync: BOOL;
0018   RxOverflow: BOOL;
0019   TxOverflow: BOOL;
0020   OdChanged: BOOL;
0021   GuardHeartbeatErrorArray: ARRAY[0..7] OF BYTE;
0022   GetEmergency: EMERGENCY_MESSAGE;
0023 END_VAR

```

Structure node status

2034

```

TYPE CAN1_NODE_STATE :
STRUCT
  NODE_ID: BYTE;
  NODE_STATE: BYTE;
  LAST_STATE: BYTE;
  RESET_NODE: BOOL;
  START_NODE: BOOL;
  PREOP_NODE: BOOL;
  SET_TIMEOUT_STATE: BOOL;
  SET_NODE_STATE: BOOL;
END_STRUCT
END_TYPE

```

Structure Emergency_Message

2035

```

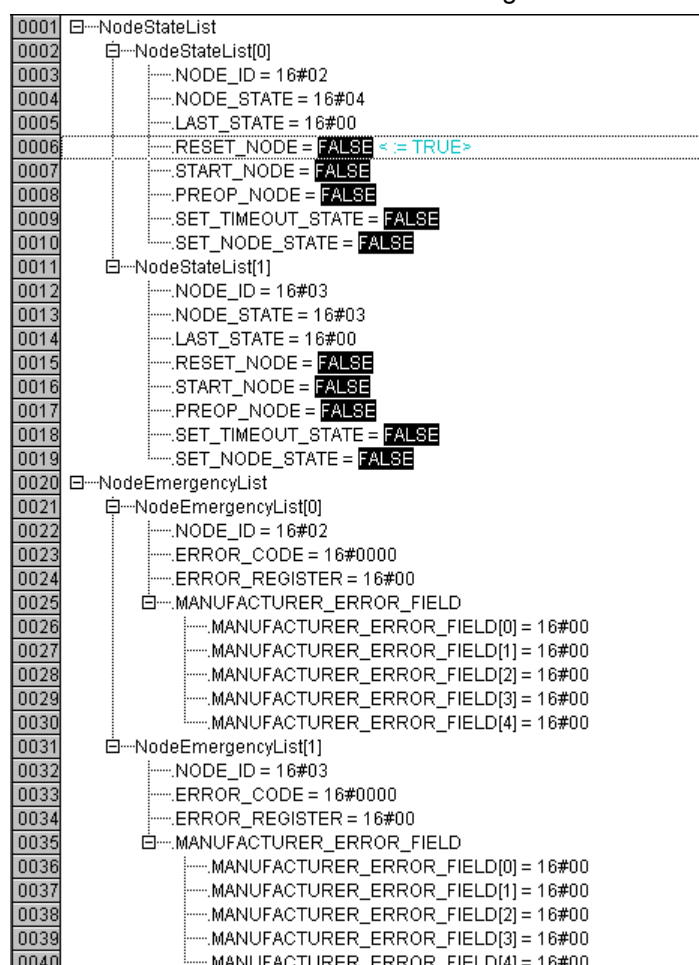
TYPE CAN1_EMERGENCY_MESSAGE :
STRUCT
  NODE_ID: BYTE;
  ERROR_CODE: WORD;
  ERROR_REGISTER: BYTE;
  MANUFACTURER_ERROR_FIELD: ARRAY[0..4] OF BYTE;
END_STRUCT
END_TYPE

```

Access to the structures at runtime of the application

2036

At runtime you can access the corresponding array element via the global variables of the library and therefore read the status or EMCY messages or reset the node.



If `ResetSingleNodeArray[0].RESET_NODE` is set to `TRUE` for a short time in the example given above, the first node is reset in the configuration tree.

Further information concerning the possible error codes → chapter CAN errors and error handling (→ page [242](#)).

Library for the CANopen slave

Contents

CANx_SLAVE_NODEID (FB)	217
CANx_SLAVE_EMCY_HANDLER (FB).....	218
CANx_SLAVE_SEND_EMERGENCY (FB).....	220
CANx_SLAVE_STATUS (FB)	223

1874

The library `ifm_CRnnnn_CANopenSlave_Vxyxyz.LIB` provides a number of FBs for the CANopen slave (= CANopen device = CANopen node) which will be explained below.

CANx_SLAVE_NODEID (FB)

2044

x = number 1...n of the CAN interface (depending on the device, → data sheet)

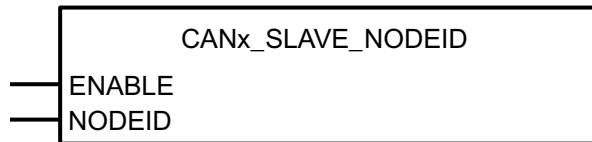
Contained in the library:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2049

CANx_SLAVE_NODEID enables the setting of the node ID of a CAN device (slave) at runtime of the application program.

Normally, the FB is called once during initialisation of the controller, in the first cycle. Afterwards, the input ENABLE is set to FALSE again.

Parameters of the inputs

2047

Parameter	Data type	Description
ENABLE	BOOL	FALSE → TRUE (edge): set node ID FALSE: unit is not executed
NODEID	BYTE	value of the new node number

CANx_SLAVE_EMCY_HANDLER (FB)

2050

x = number 1...n of the CAN interface (depending on the device, → data sheet)

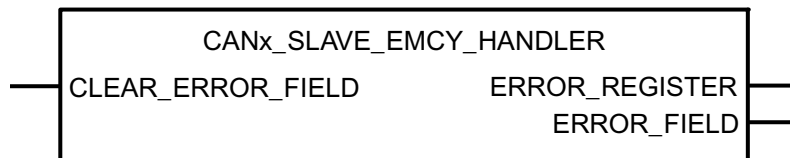
Contained in the library:

`ifm_CRnnnn_CANopenSlave_Vxyyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2053

CANx_SLAVE_EMCY_HANDLER monitors the device-specific error status (device operated as slave).

The FB must be called in the following cases:

- the error status is to be transmitted to the CAN network and
- the error messages of the application are to be stored in the object directory.

❗ NOTE

If application-specific error messages are to be stored in the object directory, CANx_SLAVE_EMCY_HANDLER must be called **after** (repeatedly) calling CANx_SLAVE_SEND_EMERGENCY (→ page [220](#)).

Parameters of the inputs

2054

Parameter	Data type	Description
CLEAR_ERROR_FIELD	BOOL	FALSE → TRUE (edge): delete ERROR FIELD FALSE: unit is not executed

Parameters of the outputs

2055

Parameter	Data type	Description
ERROR_REGISTER	BYTE	shows the contents of the object directory index 1001 ₁₆ (Error Register).
ERROR_FIELD	ARRAY [0...5] OF WORD	the array [0...5] shows the contents of the object directory index 1003 ₁₆ (Error Field): - ERROR_FIELD[0]: Number of stored errors - ERROR_FIELD[1...5]: stored errors, the most recent error is in index [1]

CANx_SLAVE_SEND_EMERGENCY (FB)

2056

x = number 1...n of the CAN interface (depending on the device, → data sheet)

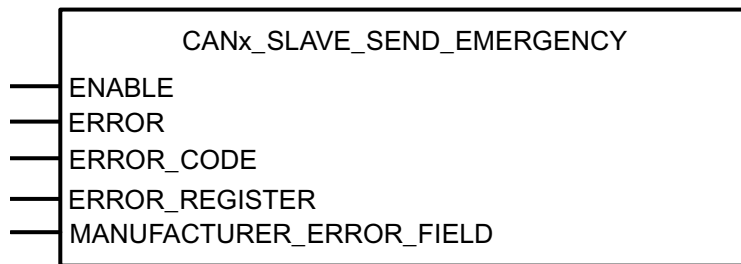
Contained in the library:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2059

Using `CANx_SLAVE_SEND_EMERGENCY` application-specific error states are transmitted. These are error messages which are to be sent in addition to the device-internal error messages (e.g. short circuit on the output).

The FB is called if the error status is to be transmitted to other devices in the network.

NOTE

If application-specific error messages are to be stored in the object directory, `CANx_SLAVE_EMCY_HANDLER` (→ page [218](#)) must be called **after** (repeatedly) calling `CANx_SLAVE_SEND_EMERGENCY`.

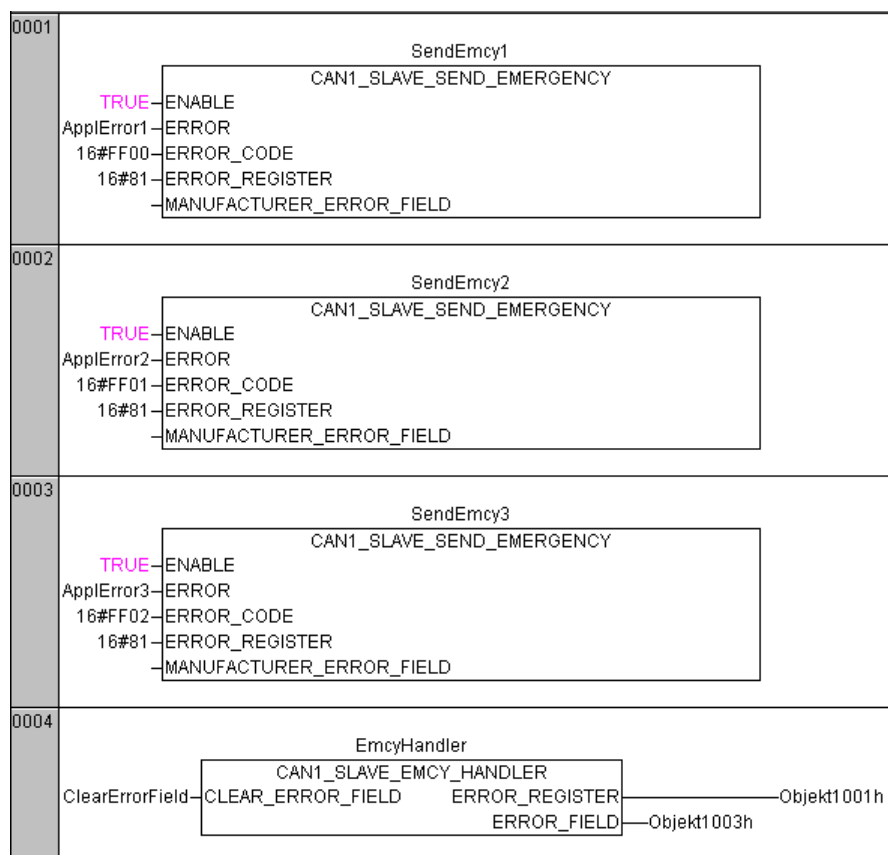
Parameters of the inputs

2060

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
ERROR	BOOL	FALSE → TRUE (edge): transmits the given error code TRUE → FALSE (edge) AND the fault is no longer indicated: the message that there is no error is sent after a delay of approx. 1 s
ERROR_CODE	WORD	The error code provides detailed information about the detected fault. The values should be entered according to the CANopen specification. → chapter Overview of the CANopen error codes (→ page 247)
ERROR_REGISTER	BYTE	This object reflects the general error state of the CANopen network participant. The values should be entered according to the CANopen specification.
MANUFACTURER_ERROR_FIELD	ARRAY [0...4] OF BYTE	Here, up to 5 bytes of application-specific error information can be entered. The format can be freely selected.

Example: CANx_SLAVE_SEND_EMERGENCY

2062



In this example 3 error messages will be generated subsequently:

1. AppError1, Code = FF00₁₆ in the error register 81₁₆
2. AppError2, Code = FF01₁₆ in the error register 81₁₆
3. AppError3, Code = FF02₁₆ in the error register 81₁₆

CAN1_SLAVE_EMCY_HANDLER sends the error messages to the error register "Object 1001₁₆" in the error array "Object 1003₁₆".

CANx_SLAVE_STATUS (FB)

2063

x = number 1...n of the CAN interface (depending on the device, → data sheet)

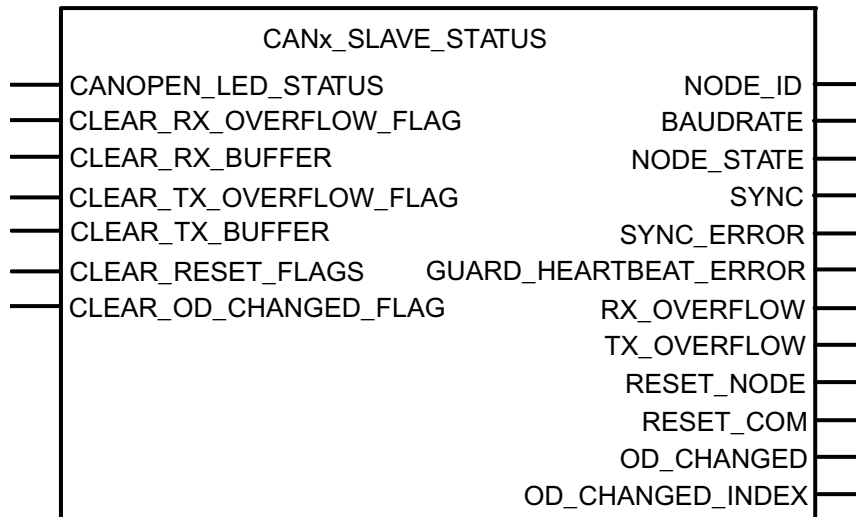
Contained in the library:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

2066

CANx_SLAVE_STATUS shows the status of the device used as CANopen slave. The FB simplifies the use of the CoDeSys CAN device libraries. We urgently recommend to carry out the evaluation of the network status via this FB.

Info

For a detailed description of the FBs of the CANopen slave and the mechanisms:
→ chapter CANopen device (→ page [190](#)).

At runtime you can then access the individual outputs of the block to obtain a status overview.

Example:

```

0001 PROGRAM SlaveStatus
0002 VAR
0003     SlaveStatus: CR0505_SLAVE_STATUS;
0004     LedStatus: BOOL := TRUE;
0005     ClearRxOverflowFlag: BOOL;
0006     ClearRxBuffer: BOOL;
0007     ClearTxOverflowFlag: BOOL;
0008     ClearTxBuffer: BOOL;
0009     ClearResetFlags: BOOL;
0010     ClearOdChanged: BOOL;
0011     NodeId: BYTE;
0012     Baudrate: WORD;
0013     NodeState: BYTE;
0014     Sync: BOOL;
0015     SyncError: BOOL;
0016     GuardHeartbeatError: BOOL;
0017     RxOverflow: BOOL;
0018     TxOverflow: BOOL;
0019     ResetNode: BOOL;
0020     ResetCom: BOOL;
0021     OdChanged: BOOL;
0022     OdChangedIndex: INT;
0023 END_VAR
    
```

Parameters of the inputs

2067

Parameter	Data type	Description
GLOBAL_START	BOOL	TRUE: all connected network participants (slaves) are started simultaneously during network initialisation FALSE: the connected network participants (slaves) are started one after the other further information → chapter Starting the network with GLOBAL_START (→ page 186)
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "receive buffer overflow" FALSE: this function is not executed
CLEAR_RX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the receive buffer FALSE: this function is not executed
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "transmit buffer overflow" FALSE: this function is not executed
CLEAR_TX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the transmit buffer FALSE: this function is not executed
CLEAR_RESET_FLAG	BOOL	FALSE → TRUE (edge): delete the flags "nodes reset" and "communications interface reset" FALSE: this function is not executed
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE → TRUE (edge): delete the flags "data in the object directory changed" and "index position" FALSE: this function is not executed

Parameters of the outputs

2068

Parameter	Data type	Description
NODE_ID	BYTE	ode ID of the slave
BAUDRATE	WORD	baud rate of the slave
NODE_STATE	BYTE	current status of the slave
SYNC	BOOL	received SYNC signal of the master
SYNC_ERROR	BOOL	no SYNC signal of the master received OR: the set SYNC time (ComCyclePeriod in the master) was exceeded
GUARD_HEARTBEAT_ERROR	BOOL	no guard or heartbeat signal of the master received OR: the set times were exceeded
RX_OVERFLOW	BOOL	error flag "receive buffer overflow"
TX_OVERFLOW	BOOL	error flag "transmit buffer overflow"
RESET_NODE	BOOL	the CAN stack of the slave was reset by the master This flag can be evaluated by the application and, if necessary, be used for further reactions.
RESET_COM	BOOL	the communication interface of the CAN stack was reset by the master This flag can be evaluated by the application and, if necessary, be used for further reactions.
OD_CHANGED	BOOL	flag "object directory master was changed"
OD_CHANGED_INDEX	INT	the output shows the changed index of the object directory

Further ifm libraries for CANopen

Contents

CANx_SDO_READ (FB)	227
CANx_SDO_WRITE (FB)	229

2071

Here we present further **ifm** FBs which are sensible additions for CANopen.

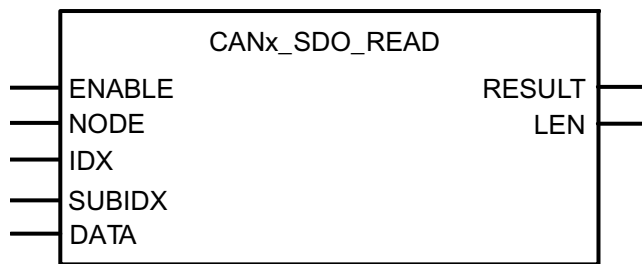
CANx_SDO_READ (FB)

621

x = number 1...n of the CAN interface (depending on the device, → data sheet)

Available for the following devices:	Contained in the library:	
	ifm_CRnnnn_Vxxyzz.LIB	ifm_CANx_SDO_Vxxyzz.LIB
CabinetController: CR030n	X	--
ClassicController: CR0020, CR0032, CR0505	X	--
ExtendedController: CR0200, CR0232	X	--
PCB controller: CS0015	X	--
SafetyController: CR7nnn	X	--
SmartController: CR25nn	X	--
PDM360: CR1050, CR1051, CR1060	--	X
PDM360compact: CR1052, CR1053, CR1055, CR1056	--	X
PDM360smart: CR1070, CR1071	X	--

Symbol in CoDeSys:



Description

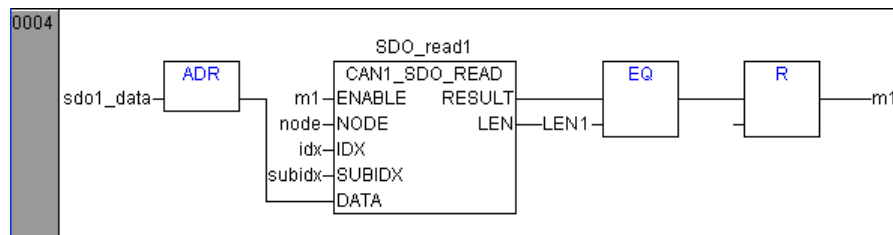
624

CANx_SDO_READ reads the SDO (→ page [179](#)) with the indicated indexes from the node.

By means of these, the entries in the object directory can be read. So it is possible to selectively read the node parameters.

all ecomatmobile controllers PCB controller: CS0015 PDM360smart: CR1070, CR1071	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
From the device library ifm_CRnnnn_Vxxyzz.LIB	From the device library ifm_CANx_SDO_Vxxyzz.LIB
Prerequisite: Node must be in the mode "Pre-Operational" or "Operational".	Prerequisite: The node must be in the mode "CANopen master" or "CAN device".
For controllers, only CAN1_SDO_READ is available.	

Example:



Parameters of the inputs

625

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
NODE	BYTE	number of the node
IDX	WORD	index in object directory
SUBIDX	BYTE	sub-index referred to the index in the object directory
DATA	DWORD	address of the receive data array permissible length = 0...255 transmission with ADR operator

Parameters of the outputs

626

Parameter	Data type	Description
RESULT	BYTE	0 = unit inactive 1 = execution of the unit completed 2 = unit active 3 = unit has not been executed
LEN	WORD	length of the entry in "number of bytes" The value for LEN must correspond to the length of the receive array. Otherwise, problems with SDO communication will occur.

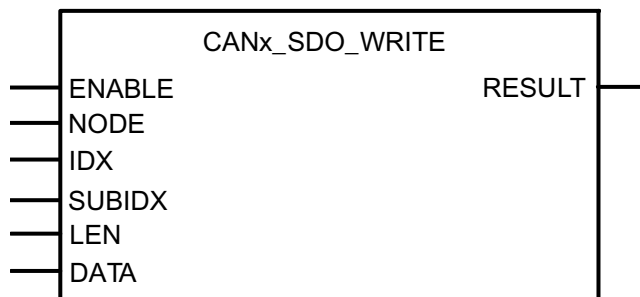
CANx_SDO_WRITE (FB)

615

x = number 1...n of the CAN interface (depending on the device, → data sheet)

Available for the following devices:	Contained in the library:	
	ifm_CRnnnn_Vxxyzz.LIB	ifm_CANx_SDO_Vxxyzz.LIB
CabinetController: CR030n	X	--
ClassicController: CR0020, CR0032, CR0505	X	--
ExtendedController: CR0200, CR0232	X	--
PCB controller: CS0015	X	--
SafetyController: CR7nnn	X	--
SmartController: CR25nn	X	--
PDM360: CR1050, CR1051, CR1060	--	X
PDM360compact: CR1052, CR1053, CR1055, CR1056	--	X
PDM360smart: CR1070, CR1071	X	--

Symbol in CoDeSys:



Description

618

CANx_SDO_WRITE writes the SDO (→ page [179](#)) with the specified indexes to the node.

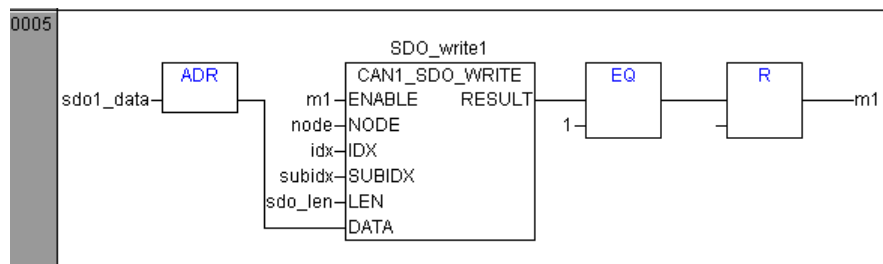
Using this FB, the entries can be written to the object directory. So it is possible to selectively set the node parameters.

all ecomatmobile controllers PCB controller: CS0015 PDM360smart: CR1070, CR1071	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
From the device library ifm_CRnnnn_Vxxyzz.LIB	From the device library ifm_CANx_SDO_Vxxyzz.LIB
Prerequisite: the node must be in the state "Pre-Operational" or "Operational" and in the mode "CANopen master".	Prerequisite: The node must be in the mode "CANopen master" or "CAN device".
For controllers, there only is CAN1_SDO_WRITE available.	

NOTE

The value for LEN must correspond to the length of the transmit array. Otherwise, problems with SDO communication will occur.

Example:



Parameters of the inputs

619

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
NODE	BYTE	number of the node
IDX	WORD	index in object directory
SUBIDX	BYTE	sub-index referred to the index in the object directory.
LEN	WORD	length of the entry in "number of bytes" The value for LEN must correspond to the length of the transmit array. Otherwise, problems with SDO communication will occur.
DATA	DWORD	address of the transmit data array permissible length = 0...255 transmission with ADR operator

Parameters of the outputs

620

Parameter	Data type	Description
RESULT	BYTE	0 = unit inactive 1 = execution of the unit stopped 2 = unit active 3 = unit has not been executed

9.7 CANopen Safety in safety-related applications

Contents

General notes and explanations on CANopen Safety	231
CANopen for safety-related communication	232
Functions for CANopen Safety	236

3846

9.7.1 General notes and explanations on CANopen Safety

3848

A working group of the user organisation "CAN in Automation" (CiA) elaborated an extension of the CANopen communication profile (CiA DS 304). This allows the following simultaneous communication options on the same bus cable:

- "normal" communication between CAN bus participants (e.g. I/O modules and a controller),
- exchange safe data between safety CAN bus participants.

Prerequisite for this simultaneous communication option: the bus participants that generate or read this safety-related data must meet the following conditions:

- the bus participants support the respective CAN mechanisms and
- the hardware structure of the bus participants is according to the respective performance level PL_r (→ chapter Notes on safety-related applications (→ page [11](#)).

As for the design of a safety controller and the implemented application software the correctness of the data has to be ensured on a "safe bus system" as well. If an error occurs during communication, there has to be a reaction within a sufficiently short period of time and the machine has to be passed into a safe state.

At the same time the implemented safety functions must not affect the "normal" communication of the not safety-related bus participants.

9.7.2 CANopen for safety-related communication

Contents

Safety-related data objects SRDOs	233
Safeguard cycle time SCT	233
Safety-related object validation time SRVT	233
Global failsafe command GFC	234
Processing of the SRDO in the SafetyController	234
Predefined identifiers for CANopen-Safety	235

3851

The entire "safe" communication CANopen Safety is based on the standard CAN mechanisms and is integrated in the CANopen communication profiles. This allows to exchange the following data simultaneously on one bus cable:

- "normal" data between non-safe participants,
- "normal" data between non-safe and safe participants,
- safe data between safe participants.

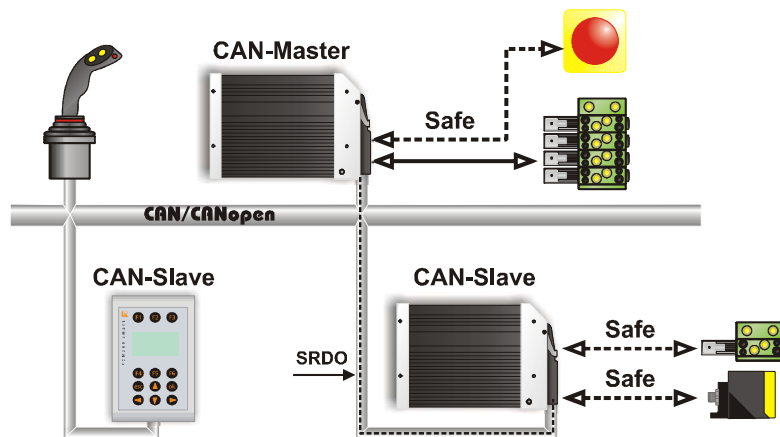


Figure: Parallel communication of normal and safe CAN bus participants

The services, protocol mechanisms and units described below complete the application and communication profile CANopen. These CAN mechanisms only guarantee the safe exchange of data between safety-related bus participants.

NOTE

The programmer has the sole responsibility for safe data processing in the application software.

Safety-related data objects SRDOs

3853

Safe data is exchanged via SRDOs (**S**afety-**R**elated **D**ata **O**bjects). An SRDO always consists of two CAN messages with different identifiers:

- message 1 contains the original user data,
- message 2 contains the same data which are inverted bit by bit.

The data is read and compared via the operating system. In case of a correct transmission of the data the application then has the original data for further processing.

The following is monitored:

- data falsified during transmission,
- cyclic update (SCT) of the SRDOs and
- the correct distance (SRVT) between the redundant message 2 and the original message.

Due to the identifier structure of CANopen the number of SRDOs which can be sent in a network of safety-related participants (producers) is limited to 64 (usually 32 receive and 32 transmit identifiers). The number of participants receiving this data (consumers) is only limited by the network structure and the general CANopen mechanisms.

Safeguard cycle time SCT

3854

In CANopen safety the **Safeguard Cycle Time (SCT)** monitors the correct function of the periodic transmission (data refresh) of the SRDOs. The data must have been repeated within the set time to be valid. Otherwise the receiving controller signals a fault and passes into the safe state (= outputs switched off).

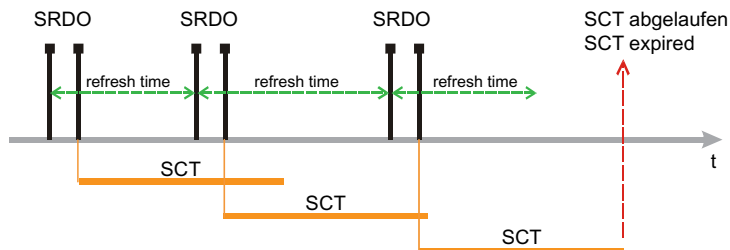


Figure: SCT monitors data refresh

Safety-related object validation time SRVT

3856

The SRVT (**S**afety-**R**elated **O**bject **V**alidation **T**ime) ensures with CANopen safety that the time between the SRDO-message pairs is adhered to.

Only if the redundant, inverted message has been transmitted after the original message within the SRVT set are the transmitted data valid. Otherwise the receiving controller signals a fault and will pass into the safe state (= outputs switched off).

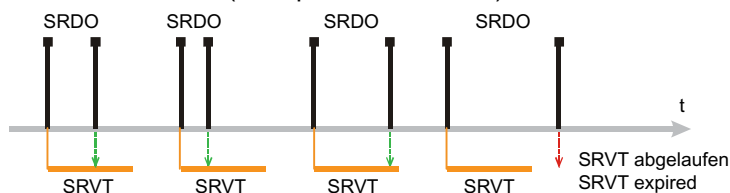


Figure: SRVT monitors the delay between the SRDO messages

Global failsafe command GFC

3858

The **Global failsafe command** (GFC) can be sent to increase the response time of the complete CANopen system. The message is event-oriented and non-safe. This means that the message is only sent once by the producer. The GFC is the same for all network participants (same identifier) and contains no data. GFC can for example be used to transmit an early warning to all safety-related participants in the network.

NOTE

Since the GFC is transmitted without safety, it must not be considered for the calculation of the first fault occurrence time.

Processing of the SRDO in the SafetyController

3859

For a safe processing of the SRDO data in the SafetyController this data has to be read **via 2 CAN interfaces**. The differences in the hardware and software interface guarantee that the transmitted data can be transferred correctly to the following application process and can be processed further.

In a CAN bus system where safety-related data is also to be transmitted, the bus cable is connected to the two CAN interfaces. Using CAN_SAFETY_TRANSMIT (→ page 237) and CAN_SAFETY_RECEIVE (→ page 239) the data is read via both interfaces and tested according to the "CANopen framework for safety-relevant communication" DS 304 and transferred to the application software.

Additional protocols (e.g. CAN Layer 2 or CANopen via CAN interface 1) are transferred to CANopen Safety and processed in "parallel".

NOTE

In connection with CANopen-Safety the CAN interface 1 must be used for CAN Layer 2!

A protocol using the 29-bit identifier (e.g. the protocol SAE J 1939 for CAN interface 2) canNOT be used in connection with CANopen-Safety.

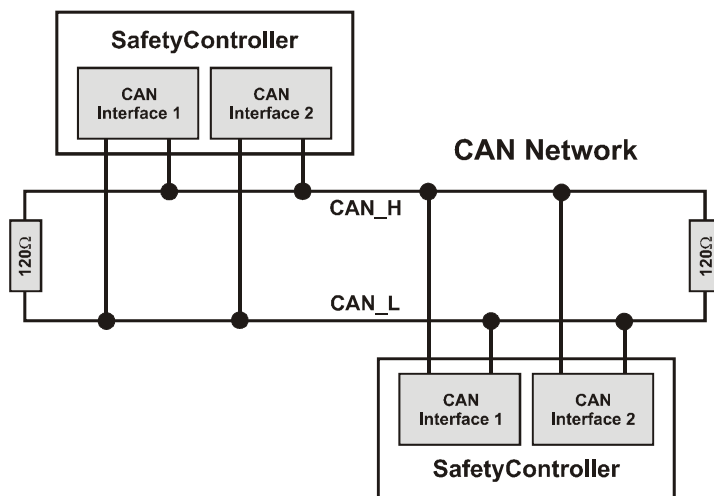


Figure: CANopen Safety participants need 2 CAN interfaces for safe data transmission

Since the SafetyController is a freely programmable device and due to the internal design of the different CANopen interfaces implemented, no object directory (OD) was implemented for the exchange of the CANopen Safety parameters.

All settings, network commands and the data transfer are processed via CAN_SAFETY_TRANSMIT (→ page 237) and CAN_SAFETY_RECEIVE (→ page 239).

This means that an external configuration tool or a CANopen master canNOT set the parameters via SDO.

Predefined identifiers for CANopen-Safety

3861

Usually predefined identifiers are used in CANopen systems. For CANopen Safety an identifier range was also fixed. It is also integrated into the pre-defined connection set of CANopen.

! NOTE

It is urgently recommended only to use the identifiers of the predefined connection set since otherwise the overview of the system is lost.

The application programmer must in any case see for himself if the CAN messages are without any conflict.

As described above, an SRDO always consists of a message **pair**. Message 1 contains the regular data and an identifier with an uneven value. Message 2 contains the inverted data and an identifier with an even value. This structure is absolutely necessary and must be adhered to.

The pre-defined connection set assumes that any safety-related participant transmits a transmit and a receive message.

The SafetyController supports up to 8 TX-SRDOs and 8 RX-SRDOs from the following identifier range:

Object	CAN identifier			
	Normal data		inverted data	
	dec.	hex.	dec.	hex.
TX-SRDO SafetyController 1	257	101	258	102
RX-SRDO SafetyController 2	259	103	260	104
	261	105	262	106

	319	13F	320	140
	321	141	322	142
RX-SRDO SafetyController 1	323	143	324	144
	325	145	326	146

	383	17F	384	180
	385	181	386	182
TX-SRDO SafetyController 2	387	183	388	184

Example:

As transmit SRDO the identifier combination 257₁₀ and 258₁₀ can for example be used as well as the identifier combination 321₁₀ and 322₁₀ as receive SRDO.

In a second SafetyController the identifier combination 257₁₀ and 258₁₀ must be used as receive SRDO and the identifier combination 321₁₀ and 322₁₀ as transmit SRDO.

9.7.3 Functions for CANopen Safety

Contents

CAN_SAFETY_TRANSMIT (FB)	237
CAN_SAFETY_RECEIVE (FB).....	239

3914

For safety functions of the SafetyController we provide the following certified CAN FBs:

- CAN_SAFETY_TRANSMIT (→ page [237](#))
- CAN_SAFETY_RECEIVE (→ page [239](#))

NOTE

CAN SAFETY FBs need 2 11-bit operated CAN interfaces at the same time.

When CAN SAFETY FBs are used the 2nd CAN interface can therefore not be used for SAE J1939 FBs (29 bits).

→ also further safety functions (→ page [33](#))

CAN_SAFETY_TRANSMIT (FB)

3871

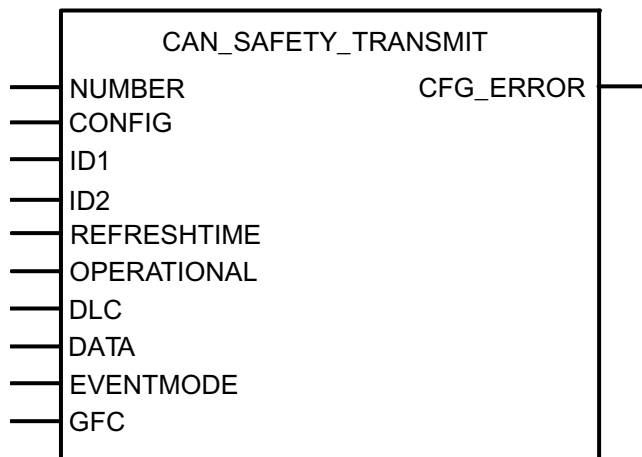
Contained in the library:

ifm_CR7nnn_Vxxyyzz.LIB

Available for the following devices:

- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Symbol in CoDeSys:



Description

3909

CAN_SAFETY_TRANSMIT transmits a safe CAN message (SRDO-TX).

The FB initialises, configures, and transmits an SRDO. The FB has to be used in the following sequence:

- ▶ The configuration set with NUMBER, ID1, ID2 and REFRESHTIME is submitted to the FB with CONFIG = TRUE.
- ▶ Start communication with OPERATIONAL = TRUE.
- > The configuration is fixed and secured by a checksum.

Max. 8 Transmit-SRDOs can be initialised in an application program.

- > Each millisecond 1 Transmit-SRDO and 1 Receive-SRDO are processed. This means: If only 1 SRDO is defined in the program, it can be transmitted every ms. In the case of 8 SRDOs each object is processed only every 8 ms.

Depending on the bus load (further CAN messages independent of CANopen-Safety) an SRVT of 16...24 ms has to be set in the case of 8 Transmit-SRDOs. If the bus load is very high the time between the normal and the inverted data telegrams increases, resulting in an even longer SRVT.

Recommend settings in the case of a very high bus load due to CANopen and 8 SRDOs:

REFRESHTIME = 100 ms

SCT = 150 ms

SRVT = 40 ms

Transmission of analogue data via an SRDO:

If analogue values are transmitted with an SRDO, input EVENTMODE should be set to FALSE. Otherwise there might be an extreme load on the CAN bus.

NOTE

CAN SAFETY FBs need 2 11-bit operated CAN interfaces at the same time.

When CAN SAFETY FBs are used the 2nd CAN interface can therefore not be used for SAE J1939 FBs (29 bits).

Parameters of the inputs

3916

Parameter	Data type	Description
NUMBER	BYTE	number of the SRDO value range = 0...7
CONFIG	BOOL	TRUE: configuration is written if OPERATIONAL = FALSE simultaneously FALSE: configuration remains unchanged
ID1	WORD	SRDO-ID1 for the original data
ID2	WORD	SRDO-ID2 for the inverted data
REFRESHTIME	TIME	time by which the SRDO is sent at the latest The REFRESHTIME has to be shorter than the SCT stated for the receive CAN_SAFETY_RECEIVE.
OPERATIONAL	BOOL	TRUE: the SRDO is transmitted cyclically and safe the FB CANNOT be reconfigured FALSE: the SRDO is not transmitted the FB can be configured
DLC	BYTE	number of bytes to be transmitted from the array DATA Value range 0...8
DATA	ARRAY[0...7] OF BYTE	the matrix contains the data to be transmitted
EVENTMODE	BOOL	TRUE: The data is transmitted in an event-driven manner, i.e. a modification to the data array causes an immediate transmission. If no modification is made, the data will be transmitted after the REFRESHTIME has elapsed, at the latest. FALSE: The data is transmitted cyclically with the time REFRESHTIME.
GFC	BOOL	Edge FALSE → TRUE: FB transmits the "Global failsafe command" once

Parameters of the outputs

3917

Parameter	Data type	Description
CFG_ERROR	BYTE	result of the configuration: 0 = no error 1 = faulty identifier pair 2 = invalid SRDO number 3 = configuration attempt in operational mode

CAN_SAFETY_RECEIVE (FB)

3873

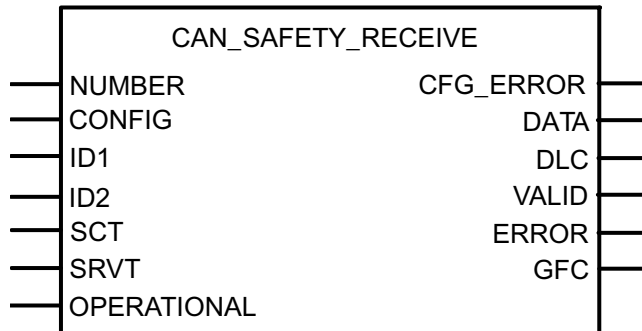
Contained in the library:

ifm_CR7nnn_Vxxyyzz.LIB

Available for the following devices:

- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Symbol in CoDeSys:



Description

3910

CAN_SAFETY_RECEIVE receives a safe CAN message (SRDO-TX).

The FB initialises, configures, and receives an SRDO. The FB has to be used in the following sequence:

- ▶ The configuration set with NUMBER, ID1, ID2, SCT, and SRVT is submitted to the FB with CONFIG = TRUE.
- ▶ Start communication with OPERATIONAL = TRUE.
- > The configuration is fixed and secured by a checksum.
- > Received data is stored in the array and output VALID is set to TRUE for one cycle. The data has to be read and evaluated safely in the application software.
- > If the data is received in a faulty state or outside the defined time limits the controller passes into the safe state (all outputs off).

Max. 8 Receive-SRDOs can be initialised in an application program.

Each millisecond 1 Transmit-SRDO and 1 Receive-SRDO can be processed. This means: If only 1 SRDO is defined in the program, it can be transmitted every ms. In the case of 8 SRDOs each object is processed only every 8 ms.

Depending on the bus load (further CAN messages independent of CANopen-Safety) an SRVT of 16...24 ms has to be set in the case of 8 Transmit-SRDOs. If the bus load is very high the time between the normal and the inverted data telegrams increases, resulting in an even longer SRVT.

Recommend settings in the case of a very high bus load due to CANopen and 8 SRDOs:

REFRESHTIME = 100 ms

SCT = 150 ms

SRVT = 40 ms

NOTE

Before the unit is activated (OPERATIONAL = TRUE), valid CANopen safety data (correct identifier, correct order, etc.) must be transmitted on the bus.

Otherwise the fault monitoring of the FB will be activated, the ERROR output set and the controller will pass into the safe state. Then, a safe CAN communication is no longer possible.

NOTE

CAN SAFETY FBs need 2 11-bit operated CAN interfaces at the same time.

When CAN SAFETY FBs are used the 2nd CAN interface can therefore not be used for SAE J1939 FBs (29 bits).

Parameters of the inputs

3918

Parameter	Data type	Description
NUMBER	BYTE	number of the SRDO. value range 0...7
CONFIG	BOOL	TRUE: configuration is written if OPERATIONAL = FALSE simultaneously. FALSE: configuration remains unchanged.
ID1	WORD	SRDO-ID1 for the original data.
ID2	WORD	SRDO-ID2 for the inverted data.
SCT	TIME	Max. time during which the SRDO shall be received cyclically. The SCT value has to be longer than the REFRESHTIME for CAN_SAFETY_TRANSMIT (→ page 237). If SCT is exceeded the output ERROR is set to TRUE and the controller passes in the "safe state".
SRVT	TIME	Max. time during which the 2nd message with the inverted data has to be received. If SRVT is exceeded the output ERROR is set to TRUE and the controller passes in the "safe state".
OPERATIONAL	BOOL	TRUE: the SRDO can be received safely the FB monitors the data for correctness as well as the SCT and the SRVT the FB CANNOT be reconfigured FALSE: the SRDO is not received the FB can be configured

Parameters of the outputs

3919

Parameter	Data type	Description
CFG_ERROR	BYTE	result of the configuration: 0 = no error 1 = faulty identifier pair 2 = invalid SRDO number 3 = configuration attempt in operational mode
DATA	ARRAY[0...7] OF BYTE	the matrix contains the data to be transmitted
DLC	BYTE	length of CAN message = number of transmitted bytes the data bytes can be read from the array
VALID	BOOL	TRUE (for one cycle): new valid data has been received FALSE: running operation
ERROR	BOOL	Group error: > Faulty data transmission, SCT or SRVT were exceeded. > At the same time, error flag ERROR_CAN_SAFETY is set. > The controller passes into the safe state. > The outputs switch off.
GFC	BOOL	TRUE (for one cycle): FB receives the "Global failsafe command"

9.8 CAN errors and error handling

Contents

CAN errors	242
Structure of an EMCY message.....	245
Overview CANopen error codes	247

1171

The error mechanisms described are automatically processed by the CAN controller integrated in the controller. This cannot be influenced by the user. (Depending on the application) the user should react to signalled errors in the application software.

Goal of the CAN error mechanisms:

- Ensuring uniform data objects in the complete CAN network
- Permanent functionality of the network even in case of a faulty CAN participant
- Differentiation between temporary and permanent disturbance of a CAN participant
- Localisation and self-deactivation of a faulty participant in 2 steps:
 - error passive
 - disconnection from the bus (bus off)
This gives a temporarily disturbed participant a "rest".

To give the interested user an overview of the behaviour of the CAN controller in case of an error, error handling is easily described below. After error detection the information is automatically prepared and made available to the programmer as CAN error bits in the application software.

9.8.1 CAN errors

Contents

Error message	242
Error counter	243
Participant, error active	243
Participant, error passive	243
Participant, bus off.....	244

8589

Error message

1172

If a bus participant detects an error condition, it immediately transmits an error flag. The transmission is then aborted or the correct messages already received by other participants are rejected. This ensures that correct and uniform data is available to all participants. Since the error flag is directly transmitted the sender can immediately start to repeat the disturbed message as opposed to other fieldbus systems (they wait until a defined acknowledgement time has elapsed). This is one of the most important features of CAN.

One of the basic problems of serial data transmission is that a permanently disturbed or faulty bus participant can block the complete system. Error handling for CAN would increase such a risk. To exclude this, a mechanism is required which detects the fault of a participant and disconnects this participant from the bus, if necessary.

Error counter

1173

A transmit and receive error counter are integrated in the CAN controller. They are counted up (incremented) for every faulty transmit or receive operation. If a transmission was correct, these counters are counted down (decremented).

However, the error counters are more incremented in case of an error than decremented in case of success. Over a defined period this can lead to a considerable increase of the counts even if the number of the undisturbed messages is greater than the number of the disturbed messages. Longer undisturbed periods slowly reduce the counts. So the counts indicate the relative frequency of disturbed messages.

If the participant itself is the first to detect errors (= self-inflicted errors), the error is more severely "punished" for this participant than for other bus participants. To do so, the counter is incremented by a higher amount.

If the count of a participant exceeds a defined value, it can be assumed that this participant is faulty. To prevent this participant from disturbing bus communication by active error messages (error active), it is switched to "error passive".

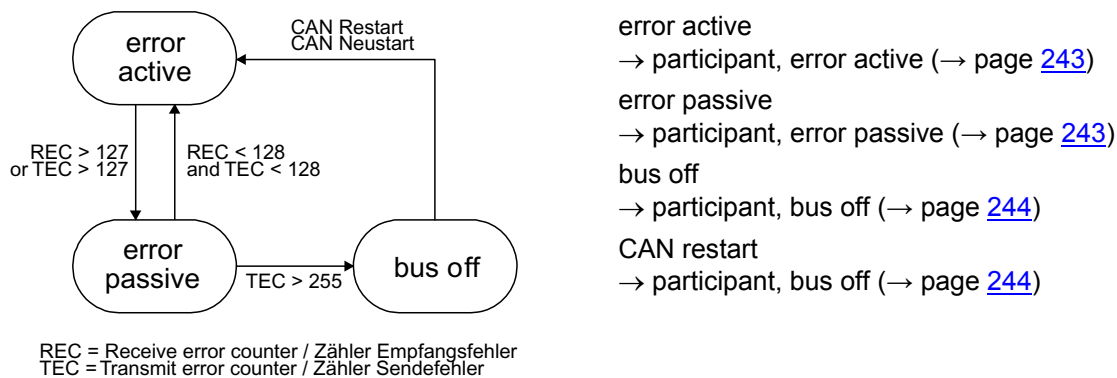


Figure: mechanism of the error counter

Participant, error active

1174

An error active participant participates in the bus communication without restriction and is allowed to signal detected errors by transmitting the active error flag. As already described the transmitted message is destroyed.

Participant, error passive

1175

An error passive participant can also communicate without restriction. However, it is only allowed to identify a detected error by a passive error flag, which does not interfere with the bus communication. An error passive participant becomes error active again if it is below a defined count value.

To inform the user about incrementing of the error counter, the system variable CANx_WARNING is set if the value of the error counter is ≥ 96 . In this state the participant is still error active.

Participant, bus off

1176

If the error count value continues to be incremented, the participant is disconnected from the bus (bus off) after exceeding a maximum count value.

To indicate this state the flag CANx_BUSOFF is set in the application program.

! NOTE

The error CANx_BUSOFF is automatically handled and reset by the operating system. If the error is to be handled or evaluated more precisely via the application program, CANx_ERRORHANDLER (→ page [152](#)) must be used. The error CANx_BUSOFF must then be reset explicitly by the application program.

9.8.2 Structure of an EMCY message

Contents

A distinction is made between the following errors:	245
Structure of an error message	245
Identifier	245
EMCY error code	246
Object 0x1003 (error field)	246
Signalling of device errors	246

8591

Under CANopen error states are indicated via a simple standardised mechanism. For a CANopen device every occurrence of an error is indicated via a special message which details the error.

If an error or its cause disappears after a certain time, this event is also indicated via the EMCY message. The errors occurred last are stored in the object directory (object 1003₁₆) and can be read via an SDO access (→ CANx_SDO_READ (→ page [227](#))). In addition, the current error situation is reflected in the error register (object 1001₁₆).

A distinction is made between the following errors:

8046

Communication error

- The CAN controller signals CAN errors.
(The frequent occurrence is an indication of physical problems. These errors can considerably affect the transmission behaviour and thus the data rate of a network.)
- Life guarding or heartbeat error

Application error

- Short circuit or wire break
- Temperature too high

Structure of an error message

8047

The structure of an error message (EMCY message) is as follows:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
EMCY error code as entered in the object 1003 ₁₆		object 1001 ₁₆	manufacturer-specific information				

Identifier

8048

The identifier for the error message consists of the sum of the following elements:

EMCY default identifier 128 (80₁₆)

+

node ID

EMCY error code

8049

It gives detailed information which error occurred. A list of possible error codes has already been defined in the communication profile. Error codes which only apply to a certain device class are defined in the corresponding device profile of this device class.

Object 0x1003 (error field)

8050

The object 1003₁₆ represents the error memory of a device. The sub-indices contain the errors occurred last which triggered an error message.

If a new error occurs, its EMCY error code is always stored in the sub-index 1₁₆. All other older errors are moved back one position in the error memory, i.e. the sub-index is incremented by 1. If all supported sub-indices are used, the oldest error is deleted. The sub-index 0₁₆ is increased to the number of the stored errors. After all errors have been rectified the value "0" is written to the error field of the sub-index 1₁₆.

To delete the error memory the value "0" can be written to the sub-index 0₁₆. Other values must not be entered.

Signalling of device errors

1880

As described, EMCY messages are transmitted if errors occur in a device. In contrast to programmable devices error messages are automatically transmitted by decentralised input/output modules (e.g. CompactModules CR2033).

Corresponding error codes → corresponding device manual.

Programmable devices only generate an EMCY message automatically (e.g. short circuit on an output) if CANx_MASTER_EMCY_HANDLER (→ page [205](#)) or CANx_SLAVE_EMCY_HANDLER (→ page [218](#)) is integrated in the application program.

Overview of the automatically transmitted EMCY error codes for all ifm devices programmable with CoDeSys → chapter Overview of the CANopen error codes (→ page [247](#)).

If in addition application-specific errors are to be transmitted by the application program, CANx_MASTER_SEND_EMERGENCY (→ page [207](#)) or CANx_SLAVE_SEND_EMERGENCY (→ page [220](#)) are used.

9.8.3 Overview CANopen error codes

8545

Error Code (hex)	Meaning
00xx	Reset or no error
10xx	Generic error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains voltage
32xx	Voltage inside the device
33xx	Output voltage
40xx	Temperature
41xx	Ambient temperature
42xx	Device temperature
50xx	Device hardware
60xx	Device software
61xx	Internal software
62xx	User software
63xx	Data set
70xx	Additional modules
80xx	Monitoring
81xx	Communication
8110	CAN overrun-objects lost
8120	CAN in error passiv mode
8130	Life guard error or heartbeat error
8140	Recovered from bus off
8150	Transmit COB-ID collision
82xx	Protocol error
8210	PDO not proceeded due to length error
8220	PDO length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

Object 0x1001 (error register)

8547

This object reflects the general error state of a CANopen device. The device is to be considered as error free if the object 1001₁₆ signals no error any more.

Bit	Meaning
0	Generic Error
1	Current
2	Voltage
3	Temperature
4	Communication Error
5	Device Profile specific
6	Reserved – always 0
7	manufacturer specific

Manufacturer specific information

8548

A device manufacturer can indicate additional error information. The format can be freely selected.

Example:

In a device two errors occur and are signalled via the bus:

- Short circuit of the outputs:

Error code 2300₁₆,

the value 03₁₆ (0000 0011₂) is entered in the object 1001₁₆
(generic error and current error)

- CAN overrun:

Error code 8110₁₆,

the value 13₁₆ (0001 0011₂) is entered in the object 1001₁₆
(generic error, current error and communication error)

>> CAN overrun processed:

Error code 0000₁₆,

the value 03₁₆ (0000 0011₂) is entered in the object 1001₁₆
(generic error, current error, communication error reset)

It can be seen only from this information that the communication error is no longer present.

Overview CANopen EMCY codes (C16)

2669

All indications (hex) for the 1st CAN interface

EMCY code object 1003 ₁₆		Object 1001 ₁₆	Manufacturer-specific information					
Byte 0	1	2	3	4	5	6	7	Description
00	21	03	I0	I1	I2	I3	I4	Diagnosis inputs
01	21	03	I0_E	I1_E	I2_E	I3_E	I4_E	Diagnosis inputs *)
00	23	03	Q1Q2	Q3	Q4			Outputs interruption
01	23	03	Q1Q2_E	Q3_E	Q4_E			Outputs interruption *)
02	23	03	Q1Q2	Q3	Q4			Outputs short circuit
03	23	03	Q1Q2_E	Q3_E	Q4_E			Outputs short circuit *)
00	31	05						Terminal voltage VBB0/VBBs
01	31	05						Terminal voltage VBB0/VBBs *)
00	33	05						Output voltage VBBr
01	33	05						Output voltage VBBr *)
00	42	09						Excess temperature
01	42	09						Excess temperature *)
00	61	11						Memory error
01	61	11						Memory error *)
00	80	11						CAN1 monitoring SYNC error (only slave)
00	81	11						CAN1 warning threshold (> 96)
10	81	11						CAN1 receive buffer overrun
11	81	11						CAN1 transmit buffer overrun
30	81	11						CAN1 guard/heartbeat error (only slave)

*) ExtendedController

10 In-/output functions

Contents

Processing analogue input values	250
Adapting analogue values	255
Counter functions for frequency and period measurement.....	258
PWM functions	272
Controller functions	313

1590

In this chapter you will find FBs which allow you to read and process the signals of the in- and outputs.

10.1 Processing analogue input values

Contents

INPUT_ANALOG (FB).....	251
INPUT_VOLTAGE (FB)	253
INPUT_CURRENT (FB).....	254

1602

In this chapter we show you FBs which allow you to read and process the values of analogue voltages or currents at the device input.

10.1.1 INPUT_ANALOG (FB)

519

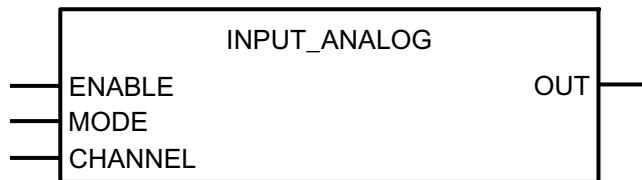
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
(For safety signals use SAFE_ANALOG_OK (→ page 34) in addition!)
- SmartController: CR25nn

Symbol in CoDeSys:



Description

522

INPUT_ANALOG enables current and voltage measurements at the analogue channels.

The FB provides the current analogue value at the selected analogue channel. The measurement and the output value result from the operating mode specified via MODE (digital input, 0...20 mA, 0...10 V, 0...30 V).

MODE	Input operating mode	Output OUT	Unit
IN_DIGITAL_H	digital input	0 / 1	---
IN_CURRENT	current input	0...20 000	µA
IN_VOLTAGE10	voltage input	0...10 000	mV
IN_VOLTAGE30	voltage input	0...30 000	mV
IN_VOLTAGE32	voltage input	0...32 000	mV
IN_RATIO	voltage input ratiometric	0...1 000	‰

For parameter setting of the operating mode, the indicated global system variables should be used. The analogue values are provided as standardised values.

NOTE

When using this FB you must set the system variable RELAIS.
Otherwise the internal reference voltages are missed for the current measurement.

Parameters of the inputs

523

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
MODE	BYTE	IN_DIGITAL_H digital input IN_CURRENT current input 0...20 000 µA IN_VOLTAGE10 voltage input 0...10 000 mV IN_VOLTAGE30 voltage input 0...30 000 mV IN_VOLTAGE32 voltage input 0...32 000 mV IN_RATIO ratiometric analogue input
INPUT_CHANNEL	BYTE	input channel

Parameters of the outputs

524

Parameter	Data type	Description
OUT	WORD	output value

10.1.2 INPUT_VOLTAGE (FB)

507

Contained in the library:

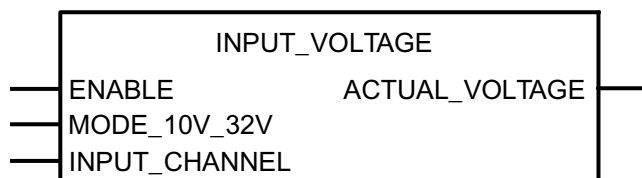
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

510

INPUT_VOLTAGE processes analogue voltages measured on the analogue channels.

- > The FB returns the current input voltage in [mV] on the selected analogue channel. The measurement refers to the voltage range defined via MODE_10V_32V (10 000 mV or 32 000 mV).

Info

INPUT_VOLTAGE is a compatibility FB for older programs. In new programs, the more powerful INPUT_ANALOG (→ page [251](#)) should be used.

Parameters of the inputs

511

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
MODE_10V_32V	BOOL	TRUE: voltage range 0...32 V FALSE: voltage range 0...10 V
INPUT_CHANNEL	BYTE	input channel

Parameters of the outputs

512

Parameter	Data type	Description
ACTUAL_VOLTAGE	WORD	output voltage in [mV]

10.1.3 INPUT_CURRENT (FB)

513

Contained in the library:

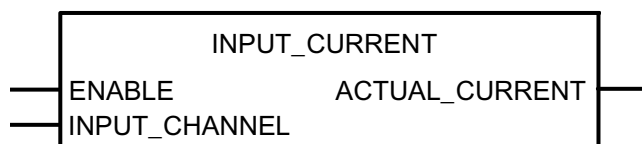
ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

516

INPUT_CURRENT returns the actual input current in [μA] at the analogue current inputs.

Info

INPUT_CURRENT is a compatibility FB for older programs. In new programs, the more powerful INPUT_ANALOG (→ page [251](#)) should be used.

Parameters of the inputs

517

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
INPUT_CHANNEL	BYTE	analogue current inputs 4...7

Parameters of the outputs

518

Parameter	Data type	Description
ACTUAL_CURRENT	WORD	input current in [μA]

10.2 Adapting analogue values

Contents

NORM (FB)	256
-----------------	-----

1603

If the values of analogue inputs or the results of analogue functions must be adapted, the following FBs will help you.

10.2.1 NORM (FB)

401

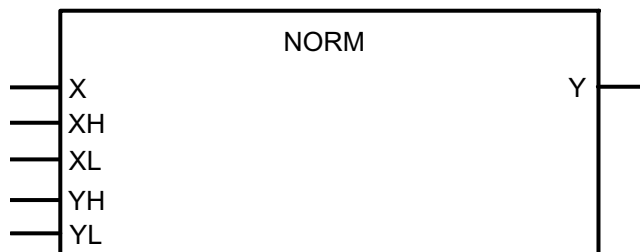
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

404

NORM normalises a value within defined limits to a value with new limits.

The FB normalises a value of type WORD within the limits of XH and XL to an output value within the limits of YH and YL. This FB is for example used for generating PWM values from analogue input values.

NOTE

The value for X must be in the defined input range between XL and XH (there is no internal plausibility check of the value).

Due to rounding errors the normalised value can deviate by 1.

If the limits (XH/XL or YH/YL) are defined in an inverted manner, normalisation is also done in an inverted manner.

Parameters of the inputs

405

Parameter	Data type	Description
X	WORD	current input value
XH	WORD	upper limit of input value range
XL	WORD	lower limit of input value range
YH	WORD	upper limit of output value range
YL	WORD	lower limit of output value range

Parameters of the outputs

406

Parameter	Data type	Description
Y	WORD	normalised value

Example 1

407

lower limit value input	0	XL
upper limit value input	100	XH
lower limit value output	0	YL
upper limit value output	2000	YH

then the FB converts the input signal for example as follows:

from X =	50	0	100	75
to Y =	1000	0	2000	1500

Example 2

408

lower limit value input	2000	XL
upper limit value input	0	XH
lower limit value output	0	YL
upper limit value output	100	YH

then the FB converts the input signal for example as follows:

from X =	1000	0	2000	1500
to Y =	50	100	0	25

10.3 Counter functions for frequency and period measurement

Contents

Applications	258
Use as digital inputs	258

1591

Depending on the controller up to 16 fast inputs are supported which can process input frequencies of up to 30 kHz. Further to the pure frequency measurement at the inputs FRQ, the inputs ENC can be also used to evaluate incremental encoders (counter function) with a maximum frequency of 10 kHz. The inputs CYL are used for period measurement of slow signals.

Input	Frequency [kHz]	Description
FRQ 0 / ENC 0	30 / 10	frequency measurement / encoder 1, channel A
FRQ 1 / ENC 0	30 / 10	frequency measurement / encoder 1, channel B
FRQ 2 / ENC 1	30 / 10	frequency measurement / encoder 2, channel A
FRQ 3 / ENC 1	30 / 10	frequency measurement / encoder 2, channel B
CYL 0 / ENC 2	10	period measurement / encoder 3, channel A
CYL 1 / ENC 2	10	period measurement / encoder 3, channel B
CYL 2 / ENC 3	10	period measurement / encoder 4, channel A
CYL 3 / ENC 3	10	period measurement / encoder 4, channel B

The following functions are available for easy evaluation:

10.3.1 Applications

1592

It must be taken into account that the different measuring methods can cause errors in the frequency detection.

FREQUENCY (→ page 259) is suitable for frequencies between 100 Hz and 30 kHz; the error decreases at high frequencies.

PERIOD (→ page 261) carries out a period measurement. It is thus suitable for frequencies lower than 1000 Hz. In principle it can also measure higher frequencies, but this has a significant impact on the cycle time. This must be taken into account when setting up the application software.

10.3.2 Use as digital inputs

Contents

FREQUENCY (FB)	259
PERIOD (FB)	261
PERIOD_RATIO (FB)	263
PHASE (FB)	265
INC_ENCODER (FB)	267
FAST_COUNT (FB)	270

1593

If the fast inputs (FRQx / CYLx) are used as "normal" digital inputs, the increased sensitivity to interfering pulses must be taken into account (e.g. contact bouncing for mechanical contacts). The standard digital input has an input frequency of 50 Hz. If necessary, the input signal must be debounced by means of the software.

FREQUENCY (FB)

537

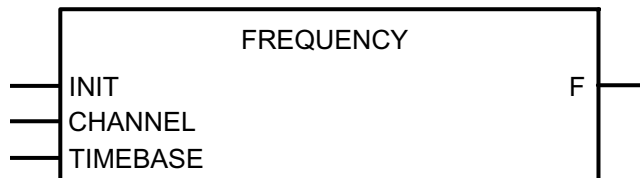
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
(For safety signals use SAFE_FREQUENCY_OK (→ page 36) together with PERIOD (→ page 261)!))
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

540

FREQUENCY measures the signal frequency at the indicated channel. Maximum input frequency → data sheet.

This FB measures the frequency of the signal at the selected CHANNEL. To do so, the positive edge is evaluated. Depending on the TIMEBASE, frequency measurements can be carried out in a wide value range. High frequencies require a short time base, low frequencies a correspondingly longer time base. The frequency is provided directly in [Hz].

! NOTE

For FREQUENCY only the inputs FRQ0...FRQ3 can be used.

Parameters of the inputs

541

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
TIMEBASE	TIME	time base

NOTE

The FB may provide wrong values before initialisation.

► Only evaluate the output if the FB has been initialised.

Parameters of the outputs

542

Parameter	Data type	Description
F	REAL	frequency in [Hz]

PERIOD (FB)

370

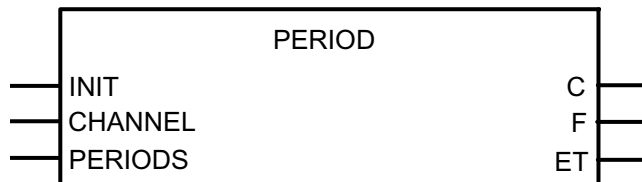
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
(For safety signals use SAFE_FREQUENCY_OK (→ page 36) together with FREQUENCY (→ page 259) in addition!)
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

373

PERIOD measures the frequency and the cycle period (cycle time) in [μs] at the indicated channel. Maximum input frequency → data sheet.

This FB measures the frequency and the cycle time of the signal at the selected CHANNEL. To calculate, all positive edges are evaluated and the average value is determined by means of the number of indicated PERIODS.

In case of low frequencies there will be inaccuracies when using FREQUENCY. To avoid this, PERIOD can be used. The cycle time is directly indicated in [μs].

The maximum measuring range is approx. 71 min.

NOTE

For PERIOD only the inputs CYL0...CYL3 can be used.

Frequencies < 0.5 Hz are no longer clearly indicated!

Parameters of the inputs

374

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
PERIODS	BYTE	number of periods to be compared

! NOTE

The FB may provide wrong values before initialisation. Do not evaluate the output before the FB has been initialised.

We urgently recommend to initialise all required instances of this FB at the same time. Otherwise, wrong values may be provided.

Parameters of the outputs

375

Parameter	Data type	Description
C	DWORD	cycle time of the detected periods in [μs]
F	REAL	frequency of the detected periods in [Hz]
ET	TIME	time elapsed since the beginning of the period measurement (can be used for very slow signals)

PERIOD_RATIO (FB)

364

Contained in the library:

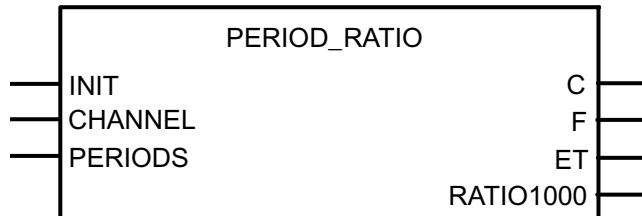
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

367

PERIOD_RATIO measures the frequency and the cycle period (cycle time) in [μs] during the indicated periods at the indicated channel. In addition, the mark-to-space ratio is indicated in per mill. Maximum input frequency → data sheet.

This FB measures the frequency and the cycle time of the signal at the selected CHANNEL. To calculate, all positive edges are evaluated and the average value is determined by means of the number of indicated PERIODS. In addition, the mark-to-space ratio is indicated in [%].

For example: In case of a signal ratio of 25 ms high level and 75 ms low level the value RATIO1000 is provided as 250 ‰.

In case of low frequencies there will be inaccuracies when using FREQUENCY. To avoid this, PERIOD_RATIO can be used. The cycle time is directly indicated in [μs].

The maximum measuring range is approx. 71 min.

NOTE

For PERIOD_RATIO only the inputs CYL0...CYL3 can be used.

The output RATIO1000 provides the value 0 for a mark-to-space ratio of 100 % (input signal permanently at supply voltage).

Frequencies < 0.05 Hz are no longer clearly indicated!

Parameters of the inputs

368

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
PERIODS	BYTE	number of periods to be compared

! NOTE

The FB may provide wrong values before initialisation. Do not evaluate the output before the FB has been initialised.

We urgently recommend to initialise all required instances of this FB at the same time. Otherwise, wrong values may be provided.

Parameters of the outputs

369

Parameter	Data type	Description
C	DWORD	cycle time of the detected periods in [μs]
F	REAL	frequency of the detected periods in [Hz]
ET	TIME	time elapsed since the beginning of the last change in state of the input signal (can be used for very slow signals)
RATIO1000	WORD	mark-to-space ratio in [‰]

PHASE (FB)

358

Contained in the library:

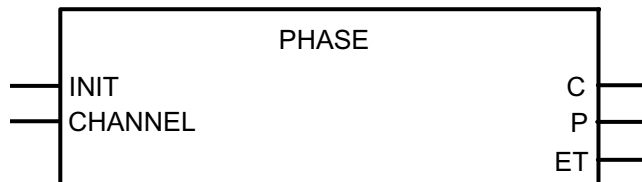
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

361

PHASE reads a pair of channels with fast inputs and compares the phase position of the signals. Maximum input frequency → data sheet.

This FB compares a pair of channels with fast inputs so that the phase position of two signals towards each other can be evaluated. An evaluation of the cycle period is possible even in the range of seconds.

! NOTE

For frequencies lower than 15 Hz a cycle period or phase shift of 0 is indicated.

Parameters of the inputs

362

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the input channel pair (0...1): 0 = channel pair 0 = inputs 0 + 1 1 = channel pair 1 = inputs 2 + 3

! NOTE

The FB may provide wrong values before initialisation. Do not evaluate the output before the FB has been initialised.

We urgently recommend to program an own instance of this FB for each channel to be evaluated. Otherwise, wrong values may be provided.

Parameters of the outputs

363

Parameter	Data type	Description
C	DWORD	cycle period in [μ s]
P	INT	angle of the phase shift (0...360 °)
ET	TIME	time elapsed since the beginning of the period measurement (can be used for very slow signals)

INC_ENCODER (FB)

4187

Contained in the library:

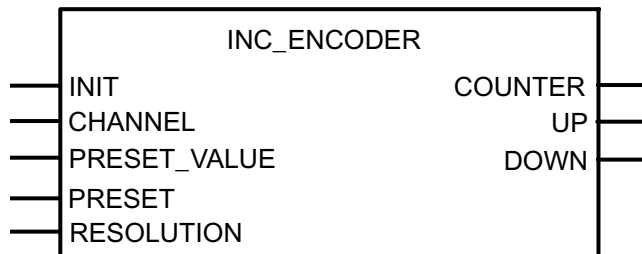
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

4330
2602

INC_ENCODER handles up/down counter functions for the evaluation of encoders.

Two frequency inputs form the input pair which is evaluated by means of the FB. The following table shows the permissible limit frequencies and the max. number of incremental encoders that can be connected:

Device	Limit frequency	max. number of encoders
BasicController: CR040n	-???- kHz	2
CabinetController: CR030n	10 kHz	2
ClassicController: CR0020, CR0505	10 kHz	4
ClassicController: CR0032	30 kHz	8
ExtendedController: CR0200	10 kHz	8
ExtendedController: CR0232	30 kHz	16
PCB controller: CS0015	0,5 kHz	2
SafetyController: CR7020, CR7021, CR7505, CR7506	10 kHz	4
SafetyController: CR7032	30 kHz	8
ExtendedSafetyController: CR7200, CR7201	10 kHz	8
ExtendedSafetyController: CR7232	30 kHz	16
SmartController: CR25nn	10 kHz	2
PDM360smart: CR1071	1 kHz	2

NOTE

Depending on the further load on the unit the limit frequency might fall when "many" encoders are evaluated.

If the load is too high the cycle time can get unacceptably long (→ Limitations and programming notes (→ page [107](#))).

Via PRESET_VALUE the counter can be set to a preset value. The value is adopted if PRESET is set to TRUE. Afterwards, PRESET must be set to FALSE again for the counter to become active again.

The current counter value is available at the output COUNTER. The outputs UP and DOWN indicate the current counting direction of the counter. The outputs are TRUE if the counter has counted in the corresponding direction in the preceding program cycle. If the counter stops, the direction output in the following program cycle is also reset.

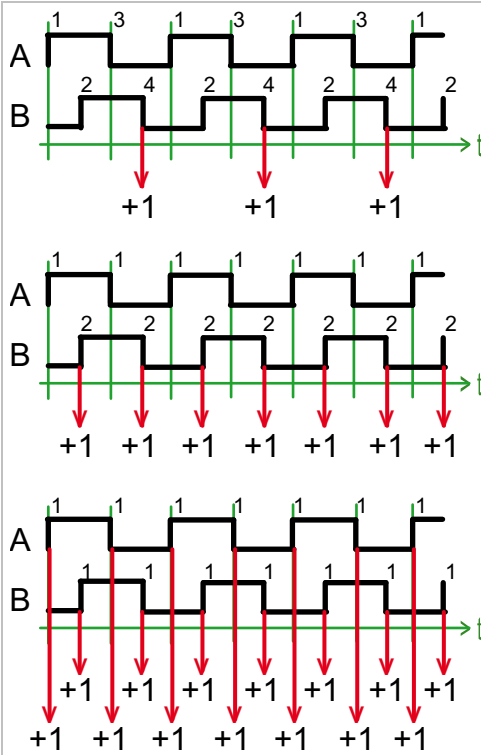
On input RESOLUTION the resolution of the encoder can be evaluated in multiples:

1 = normal resolution (identical with the resolution of the encoder),

2 = double evaluation of the resolution,

4 = 4-fold evaluation of the resolution.

All other values on this input mean normal resolution.



RESOLUTION = 1

In the case of normal resolution only the falling edge of the B-signal is evaluated.

RESOLUTION = 2

In the case of double resolution the falling and the rising edges of the B-signal are evaluated.

RESOLUTION = 4

In the case of 4-fold resolution the falling and the rising edges of the A-signal and the B-signal are evaluated.

Parameters of the inputs

4332
529

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the input channel pair (0...3): 0 = channel pair 0 = inputs 0 + 1 1 = channel pair 1 = inputs 2 + 3 2 = channel pair 2 = inputs 4 + 5 3 = channel pair 3 = inputs 6 + 7
PRESET_VALUE	DINT	preset value of the counter
PRESET	BOOL	TRUE (only 1 cycle): preset value is adopted FALSE: counter active
RESOLUTION	BYTE	factor of the encoder resolution (1, 2, 4): 1 = normal resolution 2 = double resolution 4 = 4-fold resolution all other values count as "1"

Parameters of the outputs

530

Parameter	Data type	Description
COUNTER	DINT	current counter value
UP	BOOL	TRUE: counter counts upwards FALSE: counter stands still
DOWN	BOOL	TRUE: counter counts downwards FALSE: counter stands still

FAST_COUNT (FB)

567

Contained in the library:

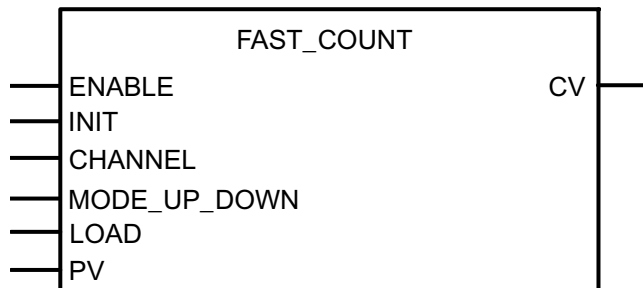
ifm_CRnnnn_Vxxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

570

FAST_COUNT operates as counter block for fast input pulses.

This FB detects fast pulses at the FRQ input channels 0...3. With the FRQ input channel 0 FAST_COUNT operates like the block CTU. Maximum input frequency → data sheet.

! NOTE

For the **ecomatmobile** controllers channel 0 can only be used as up counter. The channels 1...3 can be used as up and down counters.

Parameters of the inputs

571

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed starting from the start value FALSE: unit is not executed
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
MODE_UP_DOWN	BOOL	TRUE: counter counts downwards. FALSE: counter counts upwards
LOAD	BOOL	TRUE: start value PV being loaded FALSE: start value "0" being loaded
PV	WORD	start value (preset value)

! NOTE

After setting the parameter ENABLE the counter counts as from the indicated start value.
The counter does NOT continue from the value which was valid at the last deactivation of ENABLE.

Parameters of the outputs

572

Parameter	Data type	Description
CV	WORD	output value of the counter

10.4 PWM functions

Contents

Availability of PWM.....	272
PWM signal processing.....	273
Current control with PWM	285
Hydraulic control in PWMi	292

2303

In this chapter you will find out more about the pulse width modulation in the **ifm** device.

10.4.1 Availability of PWM

8472

PWM is available in the following devices:

	Number of available PWM outputs	of which current-controlled (PWMi)	PWM frequency [Hz]
BasicController: CR0401, CR0402	8	0	20...250
BasicController: CR0403	12	2	20...250
CabinetController: CR0301	4	0	25...250
CabinetController: CR0302, CR0303	8	0	25...250
ClassicController: CR0020	12	8	25...250
ClassicController: CR0505	8	8	25...250
ClassicController: CR0032	16	16	2...2000
ExtendedController: CR0200	24	16	25...250
ExtendedController: CR0232	32	32	2...2000
PCB controller: CS0015	8	0	25...250
SafetyController: CR7020, CR7021	12	8	25...250
SafetyController: CR7505, CR0506	8	8	25...250
ExtendedSafetyController: CR7200, CR7201	24	16	25...250
SmartController: CR25nn	4	4	25...250
PDM360smart: CR1071	4	0	25...250

10.4.2 PWM signal processing

Contents

PWM functions and their parameters.....	274
---	-----

1526

The abbreviation PWM stands for **pulse width modulation**. It is mainly used to trigger proportional valves (PWM valves) for mobile and robust controller applications. Also, with an additional component (accessory) for a PWM output the pulse-width modulated output signal can be converted into an analogue output voltage.

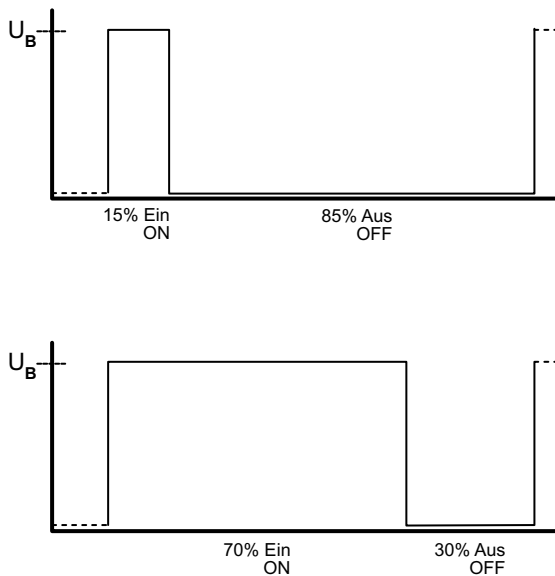


Figure: PWM principle

The PWM output signal is a pulsed signal between GND and supply voltage. Within a defined period (PWM frequency) the mark-to-space ratio is then varied. Depending on the mark-to-space ratio, the connected load determines the corresponding RMS current.

The PWM function of the **ecomatmobile** controller is a hardware function provided by the processor. To use the integrated PWM outputs of the controller, they must be initialised in the application program and parameterised corresponding to the requested output signal.

PWM functions and their parameters

Contents

PWM / PWM1000.....	274
PWM frequency.....	274
PWM channels 0...3	275
Calculation of the RELOAD value.....	275
Calculation examples RELOAD value.....	276
PWM channels 4...7 / 8...11	277
PWM dither.....	278
Ramp function	278
PWM (FB).....	279
PWM100 (FB).....	281
PWM1000 (FB).....	283

1527

PWM / PWM1000

1528

Depending on the application and the requested resolution, PWM or PWM1000 can be selected for the application programming. High accuracy and thus resolution is required when using the control functions. This is why the more technical PWM FB is used in this case.

If the implementation is to be kept simple and if there are no high requirements on the accuracy, PWM1000 (→ page [283](#)) can be used. For this FB the PWM frequency can be directly entered in [Hz] and the mark-to-space ratio in steps of 1 ‰.

PWM frequency

1529

Depending on the valve type, a corresponding PWM frequency is required. For the PWM function the PWM frequency is transmitted via the reload value (PWM (→ page [279](#))) or directly as a numerical value in [Hz] (PWM1000 (→ page [283](#))). Depending on the controller, the PWM outputs differ in their operating principle but the effect is the same.

The PWM frequency is implemented by means of an internally running counter, derived from the CPU pulse. This counter is started with the initialisation of the PWM. Depending on the PWM output group (0...3 and / or 4...7 or 4...11), it counts from FFFF₁₆ backwards or from 0000₁₆ forwards. If a transmitted comparison value (VALUE) is reached, the output is set. In case of an overflow of the counter (change of the counter reading from 0000₁₆ to FFFF₁₆ or from FFFF₁₆ to 0000₁₆), the output is reset and the operation restarts.

If this internal counter shall not operate between 0000₁₆ and FFFF₁₆, another preset value (RELOAD) can be transmitted for the internal counter. In doing so, the PWM frequency increases. The comparison value must be within the now specified range.

PWM channels 0...3

1530

These 4 PWM channels allow the most flexibility for the parameter setting. The PWM channels 0...3 are available in all **ecomatmobile** controller versions; depending on the type they feature a current control or not.

For each channel an own PWM frequency (RELOAD value) can be set. There is a free choice between PWM (→ page 279) and PWM1000 (→ page 283).

Calculation of the RELOAD value

1531

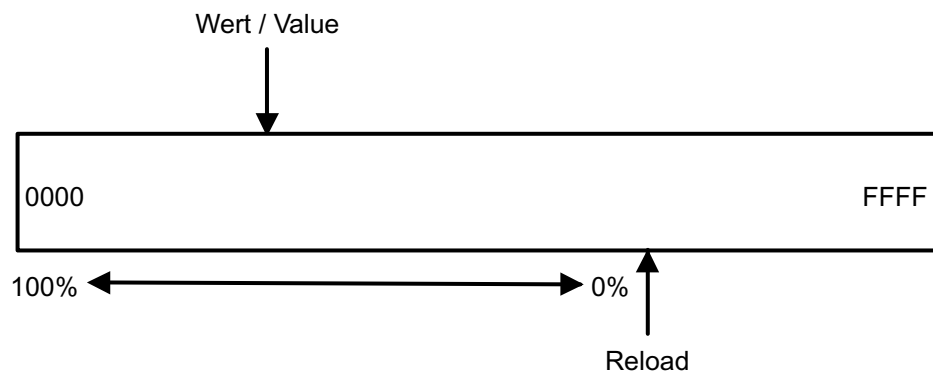


Figure: RELOAD value for the PWM channels 0...3

The RELOAD value of the internal PWM counter is calculated on the basis of the parameter DIV64 and the CPU frequency as follows:

	ClassicController ExtendedController SafetyController CabinetController (CR0303)	SmartController CabinetController (CR0301/CR0302) PCB controller
DIV64 = 0	RELOAD = 20 MHz / f_{PWM}	RELOAD = 10 MHz / f_{PWM}
DIV64 = 1	RELOAD = 312.5 kHz / f_{PWM}	RELOAD = 156.25 kHz / f_{PWM}

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1). In case of frequencies below 305 Hz respectively 152 Hz (according to the controller), DIV64 must be set to "1" to ensure that the RELOAD value is not greater than FFFF₁₆.

Calculation examples RELOAD value

1532

ClassicController ExtendedController SafetyController CabinetController (CR0303)	SmartController CabinetController (CR0301/CR0302) PCB controller
<p>The PWM frequency shall be 400 Hz.</p> <p>20 MHz</p> <p>_____ = 50 000₁₀ = C350₁₆ = RELOAD</p> <p>400 Hz</p> <p>Thus the permissible range of the PWM value is the range from 0000₁₆ to C350₁₆.</p> <p>The comparison value at which the output switches must then be between 0000₁₆ and C350₁₆.</p>	<p>The PWM frequency shall be 200 Hz.</p> <p>10 MHz</p> <p>_____ = 50 000₁₀ = C350₁₆ = RELOAD</p> <p>200 Hz</p> <p>Thus the permissible range of the PWM value is the range from 0000₁₆ to C350₁₆.</p> <p>The comparison value at which the output switches must then be between 0000₁₆ and C350₁₆.</p>

This results in the following mark-to-space ratios:

Mark-to-space ratio	Switch-on time	Value for mark-to-space ratio
Minimum	0 %	C350 ₁₆
Maximum	100 %	0000 ₁₆

Between minimum and maximum triggering 50 000 intermediate values (PWM values) are possible.

PWM channels 4...7 / 8...11

1533

These 4/8 PWM channels can only be set to one common PWM frequency. For programming, PWM and PWM1000 must not be mixed.

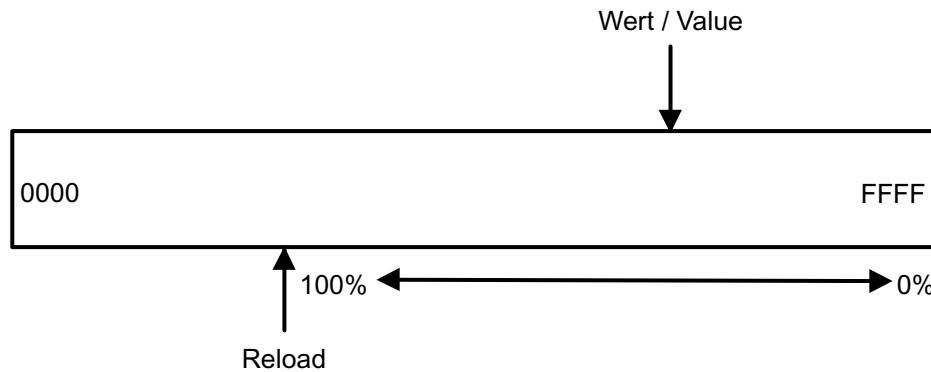


Figure: RELOAD value for PWM channels 4...7 / 8...11

The RELOAD value of the internal PWM counter is calculated (for all **ecomatmobile** controllers) on the basis of the parameters DIV64 and the CPU frequency as follows:

DIV64 = 0	$\text{RELOAD} = 10\,000_{16} - (2.5\text{ MHz} / f_{\text{PWM}})$
DIV64 = 1	$\text{RELOAD} = 10\,000_{16} - (312.5\text{ kHz} / f_{\text{PWM}})$

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1). In case of PWM frequencies below 39 Hz, DIV64 must be set to "1" to ensure that the RELOAD value is not smaller than 0000_{16} .

Example:

The PWM frequency shall be 200 Hz.

2.5 MHz

$$\frac{2.5\text{ MHz}}{200\text{ Hz}} = 12\,500_{10} = 30D4_{16}$$

200 Hz

$$\text{RELOAD value} = 10\,000_{16} - 30D4_{16} = CF2C_{16}$$

Thus the permissible range of the PWM value is the range from $CF2C_{16}$ to $FFFF_{16}$.

The comparison value at which the output switches must then be between $CF2C_{16}$ and $FFFF_{16}$.

NOTE

The PWM frequency is the same for all PWM outputs (4...7 or 4...11).

PWM and PWM1000 must not be mixed.

This results in the following mark-to-space ratios:

Mark-to-space ratio	Switch-on time	Value for mark-to-space ratio
Minimum	0 %	FFFF ₁₆
Maximum	100 %	CF2C ₁₆

Between minimum and maximum triggering 12 500 intermediate values (PWM values) are possible.

NOTE

for ClassicController and ExtendedController applies:

If the PWM outputs 4...7 are used (regardless of whether current-controlled or via one of the PWM FBs) the same frequency and the corresponding reload value have to be set for the outputs 8...11. This means that the same FBs have to be used for these outputs.

PWM dither

1534

For certain hydraulic valve types a so-called dither frequency must additionally be superimposed on the PWM frequency. If valves were triggered over a longer period by a constant PWM value, they could block due to the high system temperatures.

To prevent this, the PWM value is increased or reduced on the basis of the dither frequency by a defined value (DITHER_VALUE). As a consequence a vibration with the dither frequency and the amplitude DITHER_VALUE is superimposed on the constant PWM value. The dither frequency is indicated as the ratio (divider, DITHER_DIVIDER * 2) of the PWM frequency.

Ramp function

1535

In order to prevent abrupt changes from one PWM value to the next, e.g. from 15 % ON to 70 % ON (→ figure in PWM signal processing (→ page 273)), it is possible to delay the increase by using PT1. The ramp function used for PWM is based on the CoDeSys library UTIL.LIB. This allows a smooth start e.g. for hydraulic systems.

NOTE

When installing the **ecomatmobile** CD "Software, Tools and Documentation", projects with examples have been stored in the program directory of your PC:

...\\ifm electronic\\CoDeSys V...\\Projects\\DEMO_PLC_CDV... (for controllers) or
...\\ifm electronic\\CoDeSys V...\\Projects\\DEMO_PDM_CDV... (for PDMs).

There you also find projects with examples regarding this subject. It is strongly recommended to follow the shown procedure.

→ chapter ifm demo programs (→ page 75)

NOTE

The PWM function of the controller is a hardware function provided by the processor. The PWM function remains set until a hardware reset (switching on and off the supply voltage) has been carried out at the controller.

PWM (FB)

320

Contained in the library:

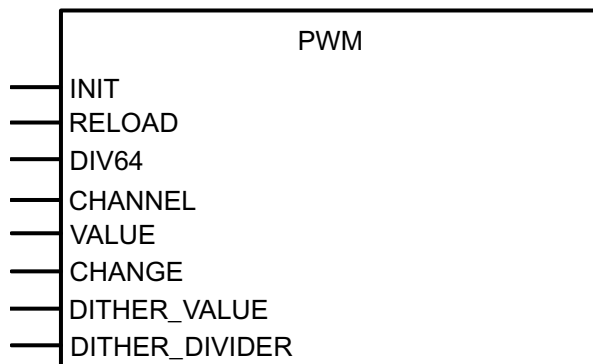
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

323

PWM is used for initialisation and parameter setting of the PWM outputs.

PWM has a more technical background. Due to their structure, PWM values can be very finely graded. So, this FB is suitable for use in controllers.

PWM is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the parameter RELOAD is also assigned.

NOTE

The value RELOAD must be identical for the channels 4...7 (for the ClassicController or ExtendedController: 4...11).

For these channels, PWM and PWM1000 (→ page [283](#)) must not be mixed.

The PWM frequency (and so the RELOAD value) is internally limited to 5 kHz.

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1).

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:

- via OUTPUT_CURRENT (→ page [291](#)) *)
- or for example using the **ifm** unit EC2049 (series element for current measurement).

*) Available for the following devices:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- SmartController: CR25nn

PWM_Dither is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the DIVIDER for the determination of the dither frequency and the VALUE are assigned.

Info

The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.

Parameters of the inputs

324

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
RELOAD	WORD	Value for the determination of the PWM frequency (→ chapter Calculation of the RELOAD value (→ page 275))
DIV64	BOOL	CPU cycle / 64
CHANNEL	BYTE	current PWM channel / output
VALUE	WORD	current PWM value
CHANGE	BOOL	TRUE: new PWM value is adopted FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	WORD	amplitude of the dither value (→ chapter PWM dither (→ page 278))
DITHER_DIVIDER	WORD	dither frequency = PWM frequency / DIVIDER * 2

PWM100 (FB)

332

IMPORTANT: New **ecomatmobile** controllers only support PWM1000 (→ page [283](#)).

Contained in the library:

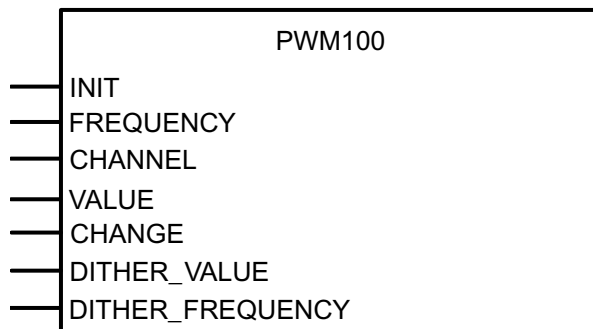
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7200, CR7505
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

335

PWM100 handles the initialisation and parameter setting of the PWM outputs.

The FB enables a simple application of the PWM FB in the **ecomatmobile** controller. The PWM frequency can be directly indicated in [Hz] and the mark-to-space ratio in steps of 1 %. This FB is **not** suited for use in controllers, due to the relatively coarse grading.

The FB is called once for each channel in the initialisation of the application program. For this, the input INIT must be set to TRUE. During initialisation, the parameter FREQUENCY is also assigned.

NOTE

The value FREQUENCY must be identical for the channels 4...7 (for the ClassicController or ExtendedController: 4...11).

For these channels, PWM (→ page [279](#)) and PWM100 must not be mixed.

The PWM frequency is limited to 5 kHz internally.

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:

- via OUTPUT_CURRENT (→ page [291](#)) *)
- or for example using the **ifm** unit EC2049 (series element for current measurement).

*) Available for the following devices:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- SmartController: CR25nn

DITHER is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the value FREQUENCY for determining the dither frequency and the dither value (VALUE) are transmitted.

Info

The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.

Parameters of the inputs

336

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
FREQUENCY	WORD	PWM frequency in [Hz]
CHANNEL	BYTE	current PWM channel / output
VALUE	BYTE	current PWM value
CHANGE	BOOL	TRUE: new PWM value is adopted FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	BYTE	amplitude of the dither value in [%]
DITHER_FREQUENCY	WORD	dither frequency in [Hz]

PWM1000 (FB)

326

Contained in the library:

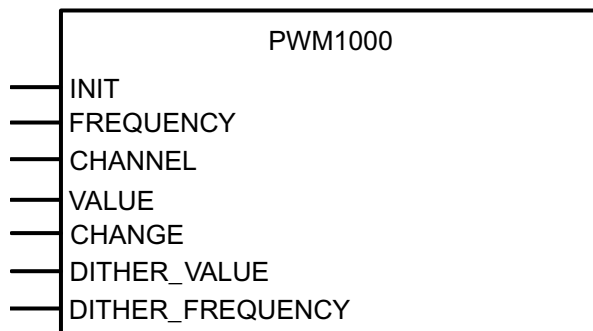
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

329

PWM1000 handles the initialisation and parameter setting of the PWM outputs.

The FB enables a simple use of the PWM FB in the **ecomatmobile** device. The PWM frequency can be directly indicated in [Hz] and the mark-to-space ratio in steps of 1 ‰.

The FB is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the parameter FREQUENCY is also assigned.

! NOTE

The value FREQUENCY must be identical for the channels 4...7 (for the ClassicController or ExtendedController: 4...11).

For these channels, [PWM](#) (See "PWM (FB)" → page [279](#)) and PWM1000 must not be mixed.

The PWM frequency is limited to 5 kHz internally.

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:

- via OUTPUT_CURRENT (→ page [291](#)) *)
- or for example using the **ifm** module EC2049 (series element for current measurement).

*) Available for the following devices:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- SmartController: CR25nn

DITHER is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the value FREQUENCY for determining the dither frequency and the dither value (VALUE) are transmitted.

Info

The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.

Parameters of the inputs

330

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
FREQUENCY	WORD	PWM frequency in [Hz]
CHANNEL	BYTE	current PWM channel / output
VALUE	WORD	current PWM value
CHANGE	BOOL	TRUE: new PWM value is adopted FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	WORD	amplitude of the dither value in [%]
DITHER_FREQUENCY	WORD	dither frequency in [Hz]

10.4.3 Current control with PWM

Contents

Overload protection	285
Current measurement with PWM channels	286
OUTPUT_CURRENT_CONTROL (FB)	287
OCC_TASK (FB)	289
OUTPUT_CURRENT (FB)	291

1550

This device of the **ecomatmobile** controller family can measure the actually flowing current on certain outputs and use the signal for further processing. For this purpose **ifm electronic** provides the user with some functions.

Overload protection

8525

In principle, the current-controlled outputs are protected against short circuit.

NOTICE

In the event of overload, in which the currents are limited by cable lengths and cross sections to for example between 8 A and 20 A, the measuring resistors (shunts) are thermally overloaded.

- The operating mode OUT_OVERLOAD_PROTECTION should always be selected for these outputs in the application program.
 - > With currents > 4.1 A the respective output is switched off automatically.
 - > If the output is no longer overloaded, the output is automatically switched on again.
- OUT_OVERLOAD_PROTECTION is not active in the PWM mode (without current control)!

Current measurement with PWM channels

1551

Current measurement of the coil current can be carried out via the current measurement channels integrated in the **ecomatmobile** controller. This allows for example that the current can be re-adjusted if the coil heats up. Thus the hydraulic conditions in the system remain the same.

NOTICE

Overload protection with ClassicController and ExtendedController:

In principle, the current-controlled outputs are protected against short circuit. In the event of overload, in which the currents are limited by cable lengths and cross sections to for example between 8 A and 20 A, the measuring resistors (shunts) are thermally overloaded.

- ▶ Since the maximum permissible current cannot always be preset, the operating mode `OUT_OVERLOAD_PROTECTION` should always be selected for the outputs in the application program. With currents > 4.1 A the respective output is switched off automatically.
- > If the output is no longer overloaded, the output is automatically switched on again.

`OUT_OVERLOAD_PROTECTION` is not active in the PWM mode (without current control)!

! NOTE

The following applies to ClassicController and ExtendedController:

The current-control `OCC_TASK` (→ page [289](#)) and `OUTPUT_CURRENT_CONTROL` (→ page [287](#)) are based on PWM (→ page [279](#)). If the current control functions are used, only the FB PWM may be used for channels 8...11. The `RELOAD` value corresponding to the frequency must be calculated.

OUTPUT_CURRENT_CONTROL (FB)

376

Contained in the library:

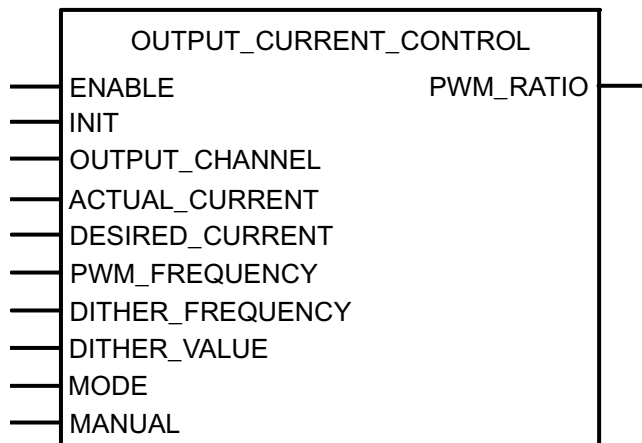
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

379

OUTPUT_CURRENT_CONTROL operates as current controller for the PWM outputs.

The controller is designed as an adaptive controller so that it is self-optimising. If this self-optimising performance is not desired, a value > 0 can be transmitted via the input MANUAL; the self-optimising performance is then deactivated. The numerical value represents a compensation value, which has an influence on the integral and differential components of the controller. To determine the best settings of the controller in the MANUAL mode, the value 50 is suitable. Depending on the requested controller characteristics the value can then be incremented step-by-step (controller becomes more sensitive / faster) or decremented (controller becomes less sensitive / slower).

If the input MANUAL is set to 0, the controller is always self-optimising. The performance of the controlled system is permanently monitored and the updated compensation values are automatically and permanently stored in each cycle. Changes in the controlled system are immediately recognised and corrected.

NOTE

To obtain a stable output value OUTPUT_CURRENT_CONTROL should be called cyclically at regular intervals.

If a precise cycle time (5 ms) is required: use OCC_TASK (→ page [289](#)).

OUTPUT_CURRENT_CONTROL is based on PWM (→ page [279](#)).

If OUTPUT_CURRENT_CONTROL is used for the outputs 4...7, only the PWM FB may be used there if the PWM outputs 8...11 are used simultaneously.

Parameters of the inputs

380

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
OUTPUT_CHANNEL	BYTE	PWM output channel (0...x: values depend on the device)
ACTUAL_CURRENT	WORD	actual current of the PWM output in [mA]; OUTPUT_CURRENT (→ page 291) must be called. The output value of OUTPUT_CURRENT is supplied to the input of ACTUAL CURRENT.
DESIRED_CURRENT	WORD	desired current value in [mA]
PWM_FREQUENCY	WORD	permissible PWM frequency for the load connected to the output
DITHER_FREQUENCY	WORD	dither frequency in [Hz]
DITHER_VALUE	BYTE	amplitude of the dither value in [%]
MODE	BYTE	controller characteristics: 0 = very slow increase, no overshoot 1 = slow increase, no overshoot 2 = minimum overshoot 3 = moderate overshoot permissible
MANUAL	BYTE	If value > 0, the self-optimising performance of the controller is overwritten (typ. value: 50).

Parameters of the outputs

381

Parameter	Data type	Description
PWM_RATIO	BYTE	for monitoring purposes: display PWM pulse ratio 0...100%

OCC_TASK (FB)

388

Contained in the library:

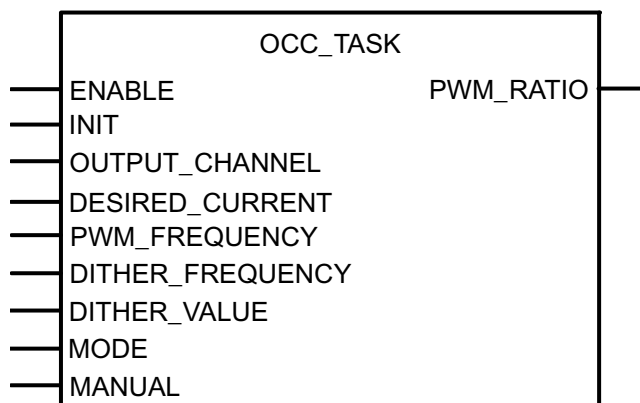
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices (NOT for SafetyController):

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

391

OCC_TASK operates as current controller for the PWM outputs.

The controller is designed as an adaptive controller so that it is self-optimising. If the self-optimising performance is not desired, a value > 0 can be transmitted via the input MANUAL (the self-optimising performance is deactivated). The numerical value represents a compensation value, which has an influence on the integral and differential components of the controller. To determine the best settings of the controller in the MANUAL mode, the value 50 is suitable. Depending on the requested controller characteristics the value can then be incremented step-by-step (controller becomes more sensitive / faster) or decremented (controller becomes less sensitive / slower).

If the input MANUAL is set to 0, the controller is always self-optimising. The performance of the controlled system is permanently monitored and the updated compensation values are automatically and permanently stored in each cycle. Changes in the controlled system are immediately recognised and corrected.

! NOTE

OCC_TASK operates with a fixed cycle time of 5 ms. No actual values need to be entered because these are detected internally by the FB.

OCC_TASK is based on PWM (→ page [279](#)).

If OUTPUT_CURRENT_CONTROL (→ page [287](#)) is used for the outputs 4...7, only the PWM FB may be used there if the PWM outputs 8...11 are used simultaneously.

Parameters of the inputs

392

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
OUTPUT_CHANNEL	BYTE	PWM output channel (0...x: values depend on the device)
DESIRED_CURRENT	WORD	desired current value in [mA]
PWM_FREQUENCY	WORD	permissible PWM frequency for the load connected to the output
DITHER_FREQUENCY	WORD	dither frequency in [Hz]
DITHER_VALUE	BYTE	amplitude of the dither value in [%]
MODE	BYTE	controller characteristics: 0 = very slow increase, no overshoot 1 = slow increase, no overshoot 2 = minimum overshoot 3 = moderate overshoot permissible
MANUAL	BYTE	If value > 0, the self-optimising performance of the controller is overwritten (typ. value: 50).

Parameters of the outputs

393

Parameter	Data type	Description
PWM_RATIO	BYTE	for monitoring purposes: display PWM pulse ratio 0...100 %

OUTPUT_CURRENT (FB)

382

Contained in the library:

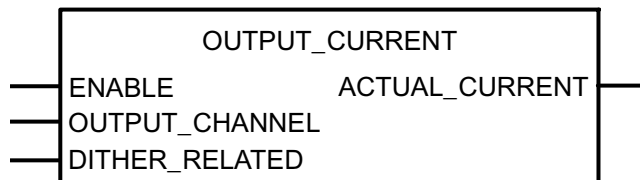
ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

385

OUTPUT_CURRENT handles the current measurement in conjunction with an active PWM channel.

The FB provides the current output current if the outputs are used as PWM outputs. The current measurement is carried out in the device, i.e. no external measuring resistors are required.

Parameters of the inputs

386

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
OUTPUT_CHANNEL	BYTE	PWM output channel (0...x: values depend on the device)
DITHER_RELATED	BOOL	averages out the current of... TRUE: one dither period FALSE: one PWM period

Parameters of the outputs

387

Parameter	Data type	Description
ACTUAL_CURRENT	WORD	output current in [mA]

10.4.4 Hydraulic control in PWMi

Contents

The purpose of this library? – An introduction	292
What does a PWM output do?	293
What is the dither?	294
Functions of the library ifm_hydraulic_16bitOS05	297

1559

ifm electronic offers the user special functions to control hydraulic systems as a special field of current regulation with PWM.

The purpose of this library? – An introduction

1560

Thanks to the FBs of this library you can fulfil the following tasks:

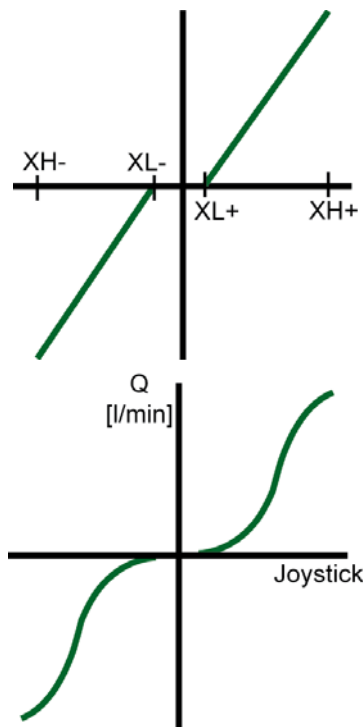
Standardise the output signals of a joystick

1561

It is not always intended that the whole movement area of the joy stick influences the movement of the machine.

Often the area around the neutral position of the joy stick is to be spared because the joy stick does not reliably supply 0 V in this neutral position.

Here in this figure the area between XL- and XL+ is to be spared.



The FBs of this library enable you to adapt the characteristic curve of your joy stick according to your requirements – on request even freely configurable:

Control hydraulic valves with current-controlled outputs

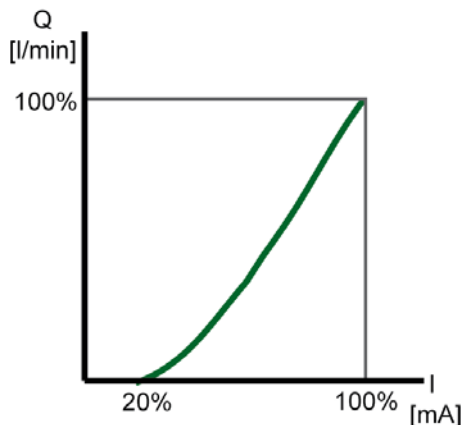
1562

As a rule hydraulic valves do not have a completely linear characteristic:

Typical characteristic curve of a hydraulic valve:

The oil flow starts at approx. 20 % of the coil current. The initial oil flow is not linear.

This has to be taken into account for the calculation of the preset values for the coil current. The FBs of this library support you here.



What does a PWM output do?

1563

PWM stands for "pulse width modulation" which means the following principle:

In general, digital outputs provide a fixed output voltage as soon as they are switched on. The value of the output voltage *cannot* be changed here. The PWM outputs, however, split the voltage into a quick sequence of many square-wave pulse trains. The pulse duration [switched on] / pulse duration [switched off] ratio determines the effective value of the requested output voltage. This is referred to as the switch-on time in [%].

Info

In the following sketches the current profiles are shown as a stylised straight line. In reality the current flows to an e-function.

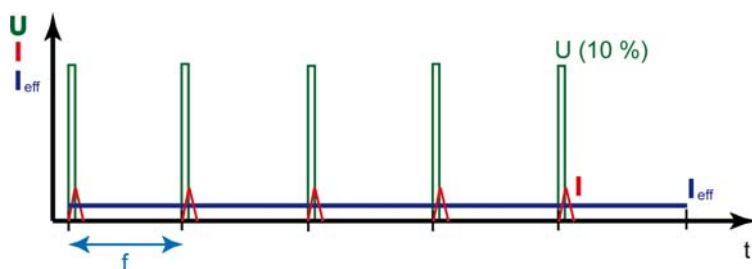


Figure: The profile of the PWM voltage U and the coil current I at 10 % switch-on time:
The effective coil current I_{eff} is also 10 %

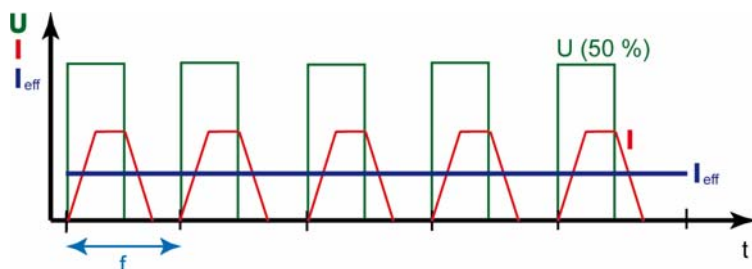


Figure: The profile of the PWM voltage U and the coil current I at 50 % switch-on time:
The effective coil current I_{eff} is also 50 %

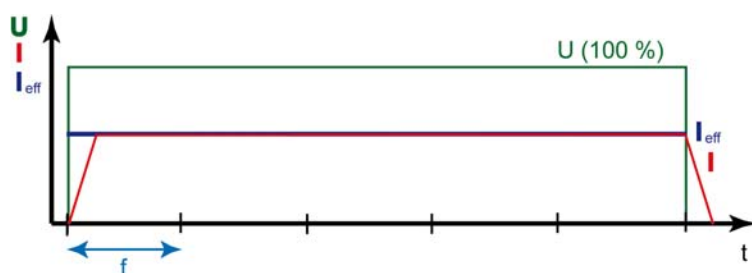


Figure: The profile of the PWM voltage U and the coil current I at 100 % switch-on time:
The effective coil current I_{eff} is also 100 %

What is the dither?

1564

If a proportional hydraulic valve is controlled, its piston does not move right away and at first not proportional to the coil current. Due to this "slip stick effect" – a kind of "break-away torque" – the valve needs a slightly higher current at first to generate the power it needs to move the piston from its off position. The same also happens for each other change in the position of the valve piston. This effect is reflected in a jerking movement, especially at very low manipulating speeds.

Technology solves this problem by having the valve piston move slightly back and forth (dither). The piston is continuously vibrating and cannot "stick". Also a small change in position is now performed without any delay, a "running start" so to speak.

Advantage: The hydraulic cylinder controlled in that way can be moved more sensitively.

Disadvantage: The valve becomes measurably hotter with dither than without because the valve coil is now working continuously.

That means that the "golden means" has to be found.

When is a dither useful?

1565

When the PWM output provides a pulse frequency that is small enough (standard value: up to 250 Hz) so that the valve piston continuously moves at a minimum stroke, an additional dither is not required (→ next figure):

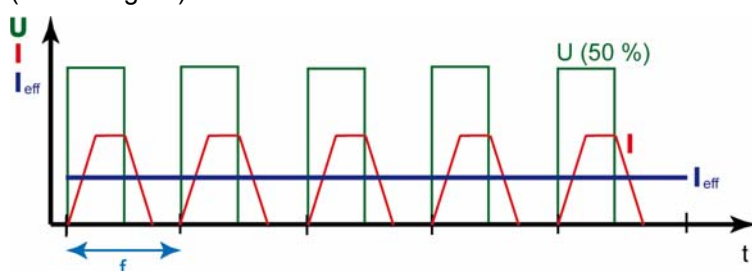


Figure: Balanced PWM signal; no dither required.

At a higher PWM frequency (standard value 250 Hz up to 1 kHz) the remaining movement of the valve piston is so short or so slow that this effectively results in a standstill so that the valve piston can again get stuck in its current position (and will do so!) (→ next figures):

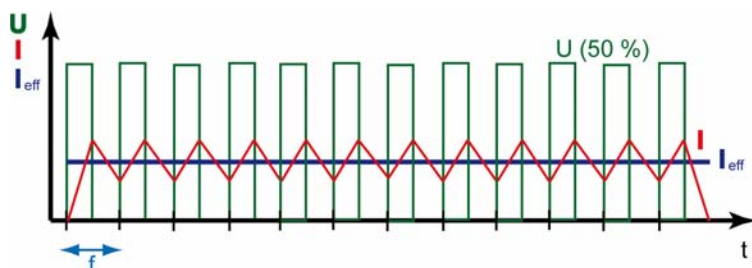


Figure: A high frequency of the PWM signal results in an almost direct current in the coil. The valve piston does not move enough any longer. With each signal change the valve piston has to overcome the break-away torque again.

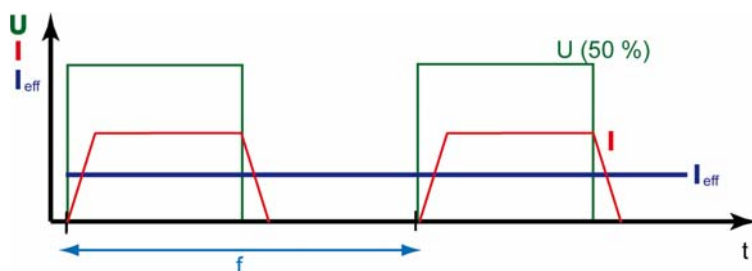


Figure: Too low frequencies of the PWM signal only allow rare, jerking movements of the valve piston. Each pulse moves the valve piston again from its off position; every time the valve piston has to overcome the break-away torque again.

NOTE

With a switch-on time below 10 % and above 90 % the dither does not have any measurable effect any longer. In such cases it makes sense and it is necessary to superimpose the PWM signal with a dither signal.

Dither frequency and amplitude

1566

The mark/space ratio (the switch-on time) of the PWM output signal is switched with the dither frequency. The dither amplitude determines the difference of the switch-on times in the two dither half-waves.

NOTE

The dither frequency must be an integer part of the PWM frequency. Otherwise the hydraulic system would not work evenly but it would oscillate.

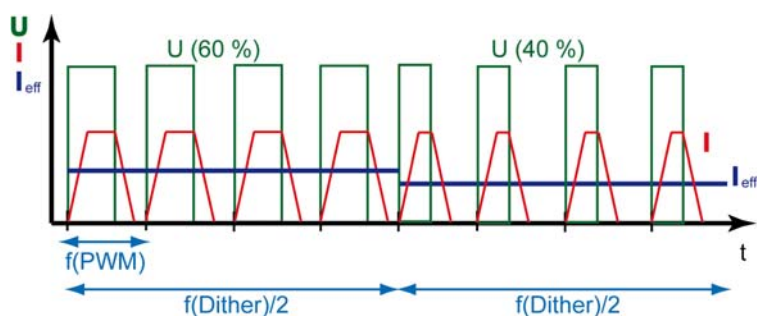
Example Dither

1567

The dither frequency is 1/8 of the PWM frequency.

The dither amplitude is 10 %.

With the switch-on time of 50 % in the figure, the actual switch-on time for 4 pulses is 60 % and for the next 4 pulses it is 40 % which means an average of 50 % switch-on time. The resulting effective coil current will be 50 % of the maximum coil current.



The result is that the valve piston always oscillates around its off position to be ready to take a new position with the next signal change without having to overcome the break-away torque before.

Functions of the library ifm_hydraulic_16bitOS05

Contents

CONTROL_OCC (FB).....	298
JOYSTICK_0 (FB).....	301
JOYSTICK_1 (FB).....	304
JOYSTICK_2 (FB).....	308
NORM_HYDRAULIC (FB)	310

6248

The library `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib` contains the following FBs:

- CONTROL_OCC (→ page [298](#)) *)
This FB uses OUTPUT_CURRENT_CONTROL (→ page [287](#)) and OUTPUT_CURRENT (→ page [291](#)) from the library `ifm_CRnnnn_Vxxyyzz.LIB`.
- JOYSTICK_0 (→ page [301](#))
- JOYSTICK_1 (→ page [304](#))
- JOYSTICK_2 (→ page [308](#))
- NORM_HYDRAULIC (→ page [310](#))

* OCC stands for **O**utput **C**urrent **C**ontrol.

The following FBs are needed from the library `UTIL.Lib` (in the CoDeSys package):

- RAMP_INT
- CHARCURVE

These FBs are automatically activated by the FBs of `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib` and configured.

The following packages are needed from the library `ifm_CRnnnn_Vxxyyzz.LIB`:

- OUTPUT_CURRENT (→ page [291](#))
- OUTPUT_CURRENT_CONTROL (→ page [287](#))
- OCC_TASK (→ page [289](#))

These FBs (→ chapter PWM signal processing (→ page [273](#))) are automatically activated and configured by the FBs of `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib`.

CONTROL_OCC (FB)

6245

Contained in the library:

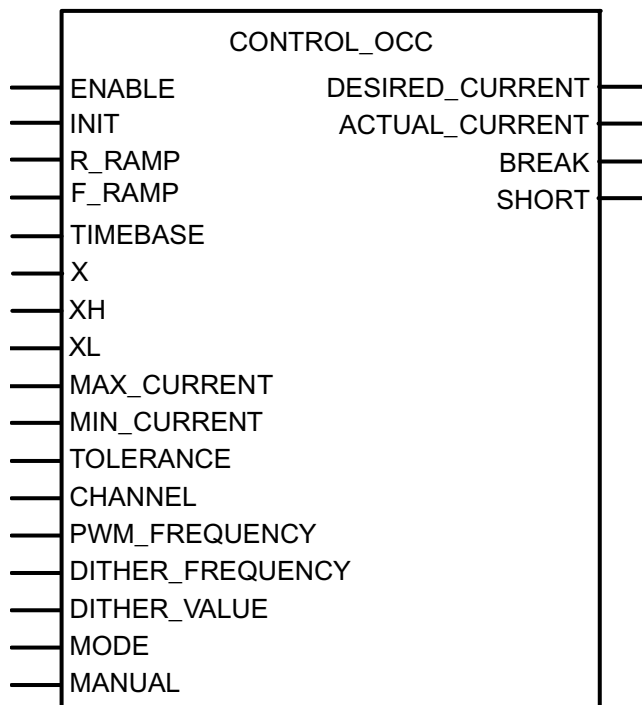
ifm_HYDRAULIC_16bitOS05_Vxxyyzz.Lib

Available for the following devices:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

600

CONTROL_OCC scales the input value X to a specified current range.

Each instance of the FB is called once in each PLC cycle. The FB uses OUTPUT_CURRENT_CONTROL (→ page 287) and OUTPUT_CURRENT (→ page 291) from the library ifm_CRnnnn_Vxxyyzz.LIB. The controller is designed as an adaptive controller so that it is self-optimising.

If this self-optimising performance is not desired, a value > 0 can be transferred via the input MANUAL: → the self-optimising performance is deactivated.

The numerical value in MANUAL represents a compensation value, which has an influence on the integral and differential components of the controller. To determine the best settings of the controller in the MANUAL mode, the value 50 is suitable.

Increase the value MANUAL: → controller becomes more sensitive / faster

Decrease the value MANUAL: → controller becomes less sensitive / slower

If the input MANUAL is set to "0", the controller is always self-optimising. The performance of the controlled system is permanently monitored and the updated compensation values are automatically and permanently stored in each cycle. Changes in the controlled system are immediately recognised and corrected.

Info

Input X of CONTROL_OCC should be supplied by the output of the JOYSTICK FBs.

Parameters of the inputs

6247

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
R_RAMP	INT	rising edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE]. 0 = without ramp
F_RAMP	INT	falling edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE]. 0 = without ramp
TIMEBASE	TIME	reference for rising / falling edge of the ramp: t#0s = rising / falling edge in [increments/PLC cycle] else = rising / falling edge in [increments/TIMEBASE]
X	WORD	input value in [increments] standardised by NORM_HYDRAULIC
XH	WORD	max. input value in [increments]
XL	WORD	min. input value in [increments]
MAX_CURRENT	WORD	max. valve current in [mA]
MIN_CURRENT	WORD	min. valve current in [mA]
TOLERANCE	BYTE	tolerance for min. valve current in [increments]. when the tolerance is exceeded, jump to MIN_CURRENT is effected
CHANNEL	BYTE	0...x = PWM output channel (values depend on the device)
PWM_FREQUENCY	WORD	PWM frequency for the connected valve in [Hz]
DITHER_FREQUENCY	WORD	dither frequency in [Hz]
DITHER_VALUE	BYTE	amplitude of the dither value in [%] of MAX_CURRENT
MODE	BYTE	controller characteristics: 0 = very slow increase, no overshoot 1 = slow increase, no overshoot 2 = minimum overshoot 3 = moderate overshoot permissible
MANUAL	BYTE	value = 0: the controller operates in a self-optimising way value > 0: the self-optimising performance of the controller is overwritten (typical: 50)

Parameters of the outputs

602

Parameter	Data type	Description
DESIRED_CURRENT	WORD	desired current value in [mA] for OCC (for monitoring purposes)
ACTUAL_CURRENT	WORD	actual current on the PWM output in [mA] (for monitoring purposes)
BREAK	BOOL	error: wire to the valve interrupted
SHORT	BOOL	error: short-circuit in the wire to the valve

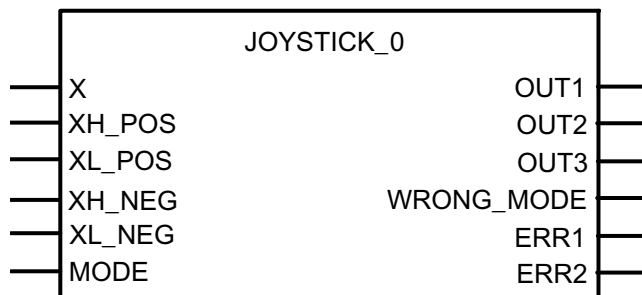
JOYSTICK_0 (FB)

6250

Contained in the library:

ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	ifm_hydraulic_32bit_Vxxyyzz.Lib
<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 • SmartController: CR25nn 	<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0032 • ExtendedController: CR0232

Symbol in CoDeSys:



Description

432

JOYSTICK_0 scales signals from a joystick to clearly defined characteristic curves, standardised to 0...1000.

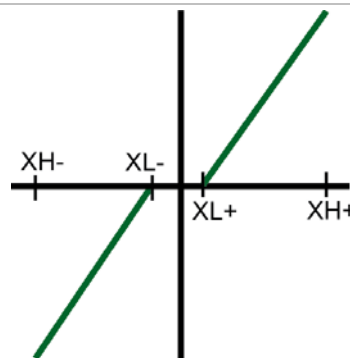
For this FB the characteristic curve values are specified (→ figures):

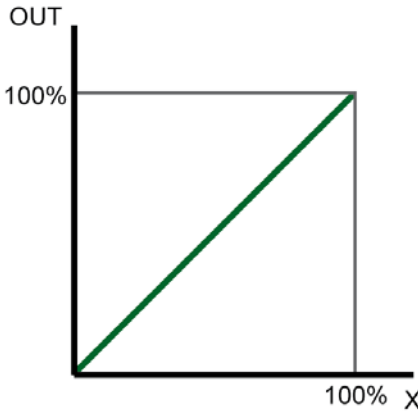
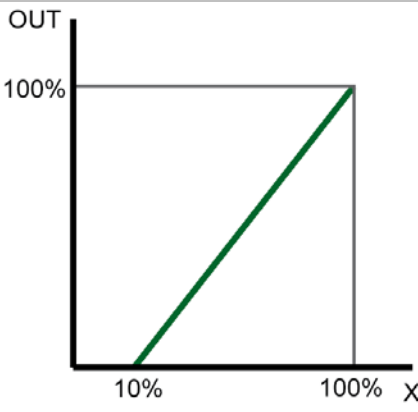
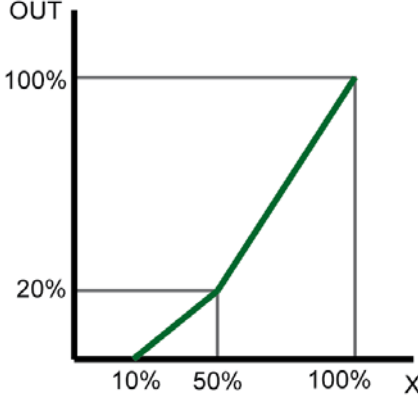
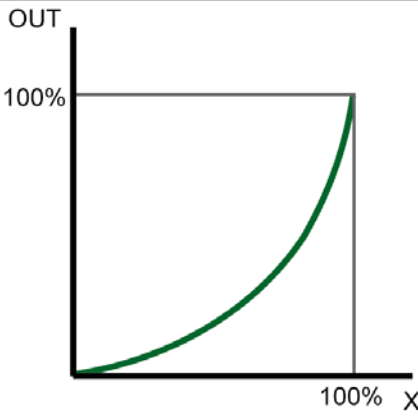
- Rising edge of the ramp = 5 increments/PLC cycle
- Falling edge of the ramp = no edge

The parameters XL_POS (XL+), XH_POS (XH+), XL_NEG (XL-) and XH_NEG (XH-) are used to evaluate the joystick movements only in the requested area.

The values for the positive and negative area may be different.

The values for XL_NEG and XH_NEG are negative here.



<p>Mode 0: characteristic curve linear for the range XL to XH</p>	
<p>Mode 1: Characteristic curve linear with dead band Values fixed to: Dead band: 0...10% of 1000 increments</p>	
<p>Mode 2: 2-step linear characteristic curve with dead band Values fixed to: Dead band: 0...10% of 1000 increments Step: X = 50 % of 1000 increments Y = 20 % of 1000 increments</p>	
<p>Characteristic curve mode 3: Curve rising (line is fixed)</p>	

Parameters of the inputs

433

Parameter	Data type	Description
X	INT	preset value input in [increments]
XH_POS	INT	max. preset value positive direction in [increments] (negative values also permissible)
XL_POS	INT	min. preset value positive direction in [increments] (negative values also permissible)
XH_NEG	INT	max. preset value negative direction in [increments] (negative values also permissible)
XL_NEG	INT	min. preset value negative direction in [increments] (negative values also permissible)
MODE	BYTE	mode selection characteristic curve: 0 = linear (0 0 – 1000 1000) 1 = linear with dead band (0 0 – 100 0 – 1000 1000) 2 = 2-step linear with dead band (0 0 – 100 0 – 500 200 – 1000 1000) 3 = curve rising

Parameters of the outputs

6252

Parameter	Data type	Description
OUT1	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve left
OUT2	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve right
OUT3	INT	standardised output value pairs of values 0 to 10 [increments] e.g. for valve on output module (e.g. CR2011 or CR2031)
WRONG_MODE	BOOL	error: invalid mode
ERR1	BYTE	error code for rising edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array
ERR2	BYTE	error code for falling edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array

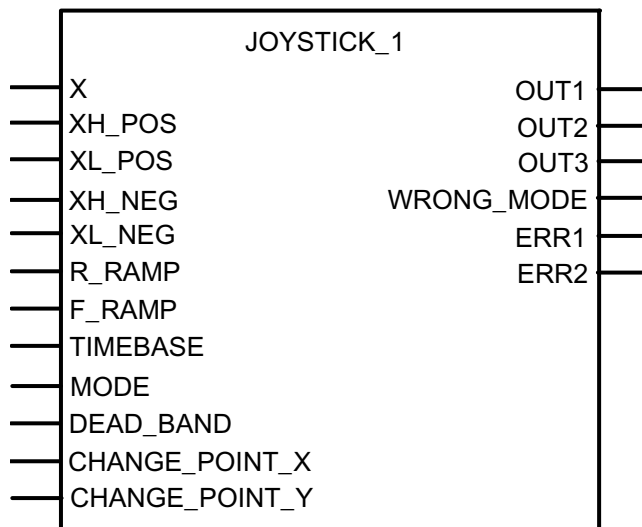
JOYSTICK_1 (FB)

6255

Contained in the library:

ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	ifm_hydraulic_32bit_Vxxyyzz.Lib
<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 • SmartController: CR25nn 	<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0032 • ExtendedController: CR0232

Symbol in CoDeSys:

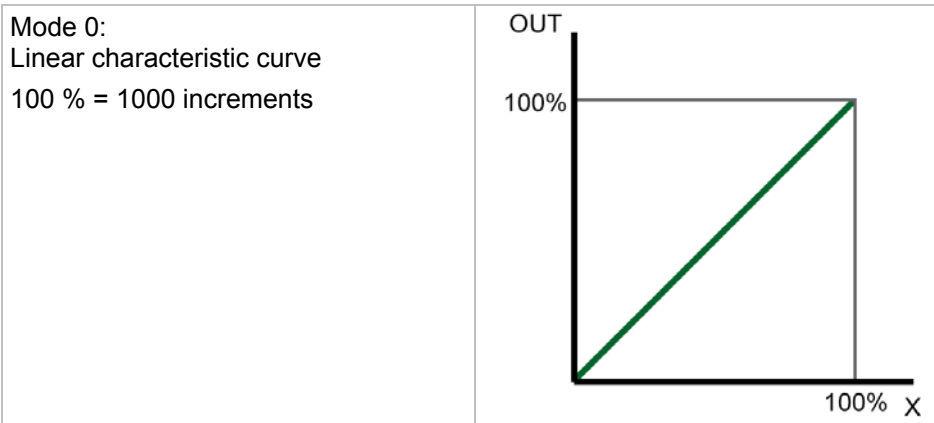


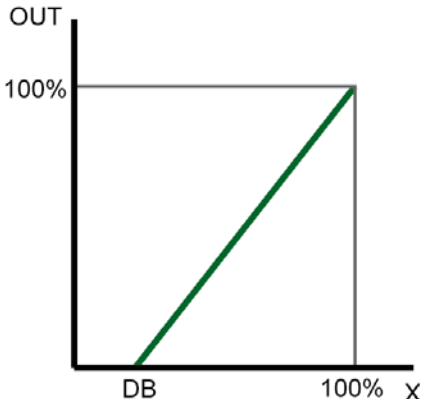
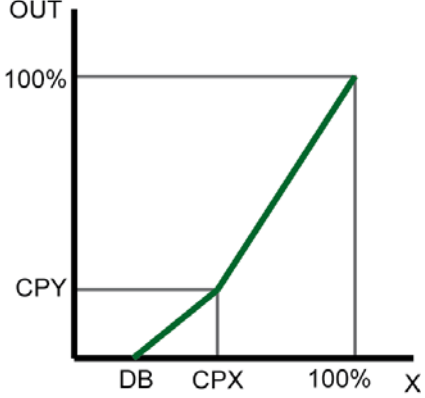
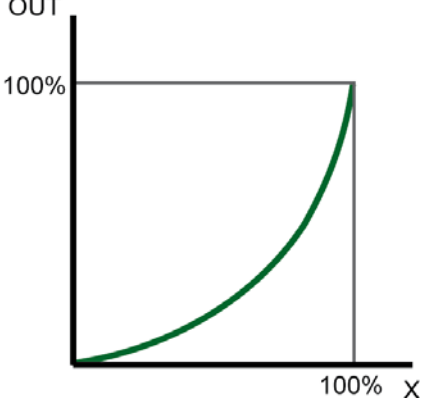
Description

425

JOYSTICK_1 scales signals from a joystick to configurable characteristic curves, standardised to 0...1000.

For this FB the characteristic curve values can be configured (→ figures):



<p>Mode 1: Characteristic curve linear with dead band</p> <p>Value for the dead band (DB) can be set in % of 1000 increments</p> <p>100 % = 1000 increments</p> <p>DB = Dead_Band</p>	
<p>Mode 2: 2-step linear characteristic curve with dead band</p> <p>Values can be configured to:</p> <p>Dead band: 0...DB in % of 1000 increments</p> <p>Step: X = CPX in % of 1000 increments Y = CPY in % of 1000 increments</p> <p>100 % = 1000 increments</p> <p>DB = Dead_Band CPX = Change_Point_X CPY = Change_Point_Y</p>	
<p>Characteristic curve mode 3: Curve rising (line is fixed)</p>	

Parameters of the inputs

6256

Parameter	Data type	Description
X	INT	preset value input in [increments]
XH_POS	INT	max. preset value positive direction in [increments] (negative values also permissible)
XL_POS	INT	min. preset value positive direction in [increments] (negative values also permissible)
XH_NEG	INT	max. preset value negative direction in [increments] (negative values also permissible)
XL_NEG	INT	min. preset value negative direction in [increments] (negative values also permissible)
R_RAMP	INT	rising edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
F_RAMP	INT	falling edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
TIMEBASE	TIME	reference for rising / falling edge of the ramp: t#0s = rising / falling edge in [increments/PLC cycle] else = rising / falling edge in [increments/TIMEBASE]
MODE	BYTE	mode selection characteristic curve: 0 = linear (0 0 – 1000 1000) 1 = linear with dead band (DB) (0 0 – DB... 0 – 1000 1000) 2 = 2-step linear with dead band (DB) (0 0 – DB 0 – CPX CPY – 1000 1000) 3 = curve rising
DEAD_BAND	BYTE	adjustable dead band (DB) in [% of 1000 increments]
CHANGE_POINT_X	BYTE	for mode 2: ramp step, value for X in [% of 1000 increments]
CHANGE_POINT_Y	BYTE	for mode 2: ramp step, value for Y in [% of 1000 increments]

Parameters of the outputs

6252

Parameter	Data type	Description
OUT1	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve left
OUT2	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve right
OUT3	INT	standardised output value pairs of values 0 to 10 [increments] e.g. for valve on output module (e.g. CR2011 or CR2031)
WRONG_MODE	BOOL	error: invalid mode
ERR1	BYTE	error code for rising edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array
ERR2	BYTE	error code for falling edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array

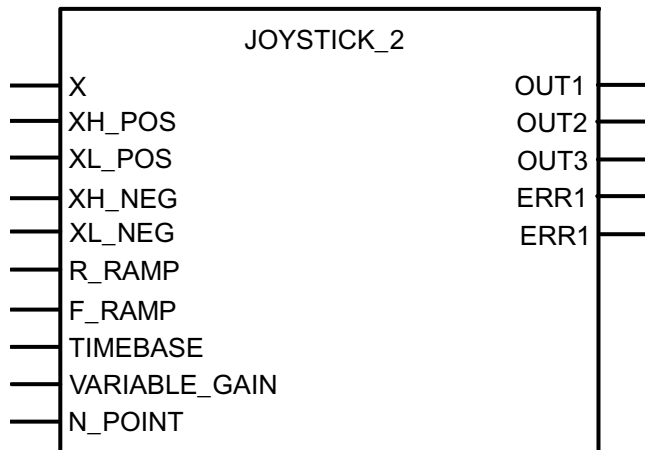
JOYSTICK_2 (FB)

6258

Contained in the library:

ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	ifm_hydraulic_32bit_Vxxyyzz.Lib
<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 • SmartController: CR25nn 	<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0032 • ExtendedController: CR0232

Symbol in CoDeSys:

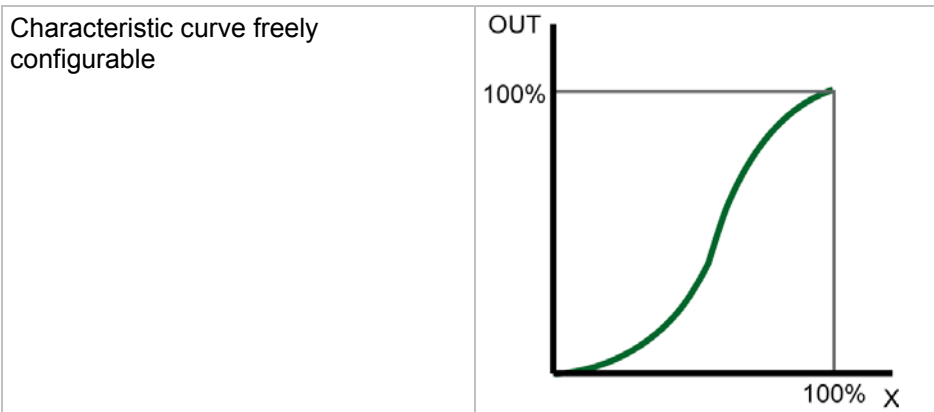


Description

418

JOYSTICK_2 scales the signals from a joystick to a configurable characteristic curve. Free selection of the standardisation.

For this FB, the characteristic curve is freely configurable (→ figure):



Parameters of the inputs

6261

Parameter	Data type	Description
X	INT	preset value input in [increments]
XH_POS	INT	max. preset value positive direction in [increments] (negative values also permissible)
XL_POS	INT	min. preset value positive direction in [increments] (negative values also permissible)
XH_NEG	INT	max. preset value negative direction in [increments] (negative values also permissible)
XL_NEG	INT	min. preset value negative direction in [increments] (negative values also permissible)
R_RAMP	INT	rising edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
F_RAMP	INT	falling edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
TIMEBASE	TIME	reference for rising and falling edge of the ramp: t#0s = rising / falling edge in [increments/PLC cycle] else = rising / falling edge in [increments/TIMEBASE]
VARIABLE_GAIN	ARRAY [0..10] OF POINT	pairs of values describing the curve the first pairs of values indicated in N_POINT are used. N = 2...11 example: 9 pairs of values declared as variable VALUES: VALUES: ARRAY[0..10] OF POINT := (X:=0,Y:=0),(X:=200,Y:=0), (X:=300,Y:=50), (X:=400,Y:=100), (X:=700,Y:=500), (X:=1000,Y:=900), (X:=1100,Y:=950), (X:=1200,Y:=1000), (X:=1400,Y:=1050); There may be blanks between the values.
N_POINT	BYTE	number of points (pairs of values in VARIABLE_GAIN) by which the curve characteristic is defined. N = 2...11

Parameters of the outputs

420

Parameter	Data type	Description
OUT1	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve left
OUT2	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve right
OUT3	INT	standardised output value pairs of values 0 to 10 [increments] e.g. for valve on output module (e.g. CR2011 or CR2031)
ERR1	BYTE	error code for rising edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array
ERR2	BYTE	error code for falling edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array

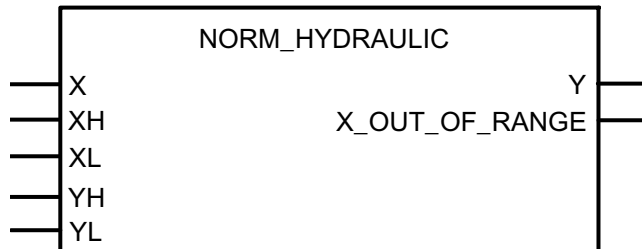
NORM_HYDRAULIC (FB)

394

Contained in the library:

ifm_hydraulic_16bitOS04_Vxxyyzz.Lib ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	ifm_hydraulic_32bit_Vxxyyzz.Lib
<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 • SmartController: CR25nn 	<p>Available for the following devices:</p> <ul style="list-style-type: none"> • ClassicController: CR0032 • ExtendedController: CR0232

Symbol in CoDeSys:



Description

397

NORM_HYDRAULIC standardises input values with fixed limits to values with new limits.

Please note: This FB corresponds to the 3S FB NORM_DINT from the CoDeSys library `UTIL.Lib`.

The FB standardises a value of type DINT within the limits of XH and XL to an output value within the limits of YH and YL.

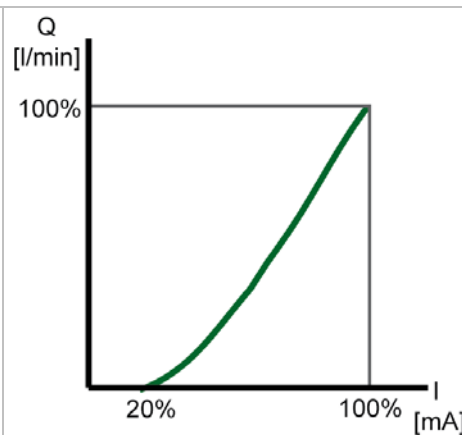
Due to rounding errors deviations from the standardised value of 1 may occur. If the limits (XH/XL or YH/YL) are indicated in inversed form, standardisation is also inverted.

If X outside the limits XL...XH, the error message `X_OUT_OF_RANGE = TRUE`.

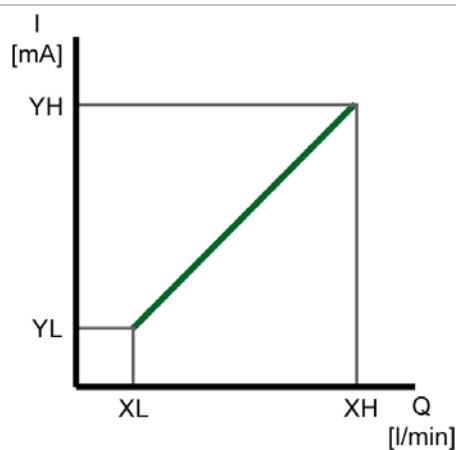
Typical characteristic curve of a hydraulic valve:

The oil flow will not start before 20% of the coil current has been reached.

At first the oil flow is not linear.



Characteristics of the FB



Parameters of the inputs

398

Parameter	Data type	Description
X	DINT	desired value input
XH	DINT	max. input value [increments]
XL	DINT	min. input value [increments]
YH	DINT	max. output value [increments], e.g.: valve current [mA] / flow [l/min]
YL	DINT	min. output value [increments], e.g.: valve current [mA] / flow [l/min]

Parameters of the outputs

399

Parameter	Data type	Description
Y	DINT	standardised output value
X_OUT_OF_RANGE	BOOL	error: X is beyond the limits XH and XL

Example: NORM_HYDRAULIC

400

Parameter	Case 1	Case 2	Case 3
Upper limit value input XH	100	100	2000
Lower limit value input XL	0	0	0
Upper limit value output YH	2000	0	100
Lower limit value output YL	0	2000	0
Non standardised value X	20	20	20
Standardised value Y	400	1600	1

Case 1:

Input with relatively coarse resolution.

Output with high resolution.

1 X increment results in 20 Y increments.

Case 2:

Input with relatively coarse resolution.

Output with high resolution.

1 X increment results in 20 Y increments.

Output signal is inverted as compared to the input signal.

Case 3:

Input with high resolution.

Output with relatively coarse resolution.

20 X increments result in 1 Y increment.

10.5 Controller functions

Contents

General.....	313
Setting rule for a controller.....	315
Functions for controllers.....	316

1622

10.5.1 General

1623

Controlling is a process during which the unit to be controlled (control variable x) is continuously detected and compared with the reference variable w . Depending on the result of this comparison, the control variable is influenced for adaptation to the reference variable.

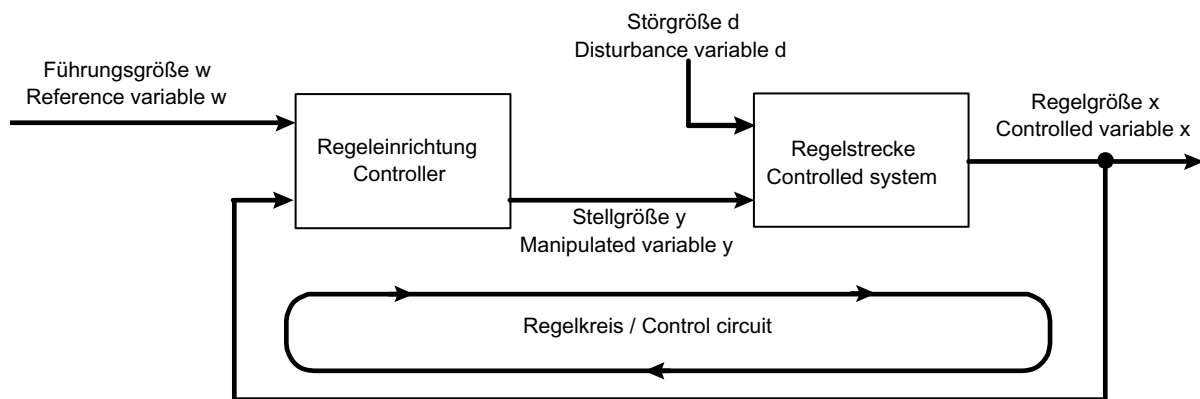


Figure: Principle of controlling

The selection of a suitable control device and its optimum setting require exact indication of the steady-state behaviour and the dynamic behaviour of the controlled system. In most cases these characteristic values can only be determined by experiments and can hardly be influenced.

Three types of controlled systems can be distinguished:

Self-regulating process

1624

For a self-regulating process the control variable x goes towards a new final value after a certain manipulated variable (steady state). The decisive factor for these controlled systems is the amplification (steady-state transfer factor K_S). The smaller the amplification, the better the system can be controlled. These controlled systems are referred to as P systems (P = proportional).

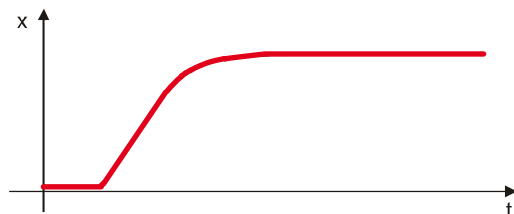


Figure: P controller = self-regulating process

Controlled system without inherent regulation

1625

Controlled systems with an amplifying factor towards infinity are referred to as controlled systems without inherent regulation. This is usually due to an integrating performance. The consequence is that the control variable increases constantly after the manipulated variable has been changed or by the influence of an interfering factor. Due to this behaviour it never reaches a final value. These controlled systems are referred to as I systems (I = integral).

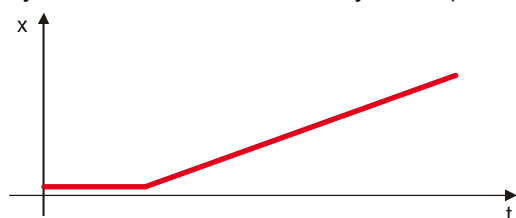


Figure: I controller = controlled system without inherent regulation

Controlled system with delay

1626

Most controlled systems correspond to series systems of P systems (systems with compensation) and one or several T1 systems (systems with inertia). A controlled system of the 1st order is for example made up of the series connection of a throttle point and a subsequent memory.

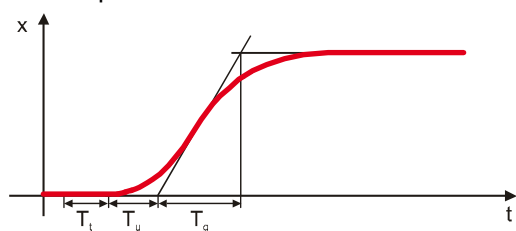


Figure: PT system = controlled system with delay

For controlled systems with dead time the control variable does not react to a change of the control variable before the dead time T_t has elapsed. The dead time T_t or the sum of $T_t + T_u$ relates to the controllability of the system. The controllability of a system is the better, the greater the ratio T_g/T_u .

The controllers which are integrated in the library are a summary of the preceding basic functions. It depends on the respective controlled system which functions are used and how they are combined.

10.5.2 Setting rule for a controller

1627

For controlled systems, whose time constants are unknown the setting procedure to Ziegler and Nickols in a closed control loop is of advantage.

Setting control

1628

At the beginning the controlling system is operated as a purely P-controlling system. In this respect the derivative time T_V is set to 0 and the reset time T_N to a very high value (ideally to ∞) for a slow system. For a fast controlled system a small T_N should be selected.

Afterwards the gain K_P is increased until the control deviation and the adjustment deviation perform steady oscillation at a constant amplitude at $K_P = K_{P_{critical}}$. Then the stability limit has been reached.

Then the time period $T_{critical}$ of the steady oscillation has to be determined.

Add a differential component only if necessary.

T_V should be approx. 2...10 times smaller than T_N

K_P should be equal to K_D .

Idealised setting of the controlled system:

Control unit	$K_P = K_D$	T_N	T_V
P	$2.0 * K_{P_{critical}}$	—	—
PI	$2.2 * K_{P_{critical}}$	$0.83 * T_{critical}$	—
PID	$1.7 * K_{P_{critical}}$	$0.50 * T_{critical}$	$0.125 * T_{critical}$

NOTE

For this setting process it has to be noted that the controlled system is not harmed by the oscillation generated. For sensitive controlled systems K_P must only be increased to a value at which no oscillation occurs.

Damping of overshoot

1629

To dampen overshoot PT1 (→ page [319](#)) (low pass) can be used. In this respect the preset value XS is damped by the PT1 link before it is supplied to the controller function.

The setting variable T1 should be approx. 4...5 times greater than T_N (of the PID or GLR controller).

10.5.3 Functions for controllers

Contents

DELAY (FB).....	317
PT1 (FB).....	319
PID1 (FB)	320
PID2 (FB)	322
GLR (FB).....	325

1634

The section below describes in detail the units that are provided for set-up by software controllers in the **ecomatmobile** device. The units can also be used as basis for the development of your own control functions.

DELAY (FB)

585

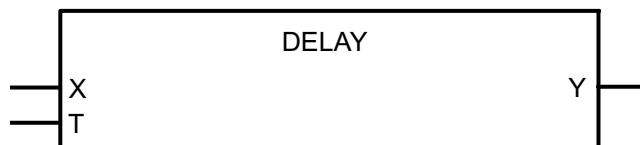
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

588

DELAY delays the output of the input value by the time T (dead-time element).

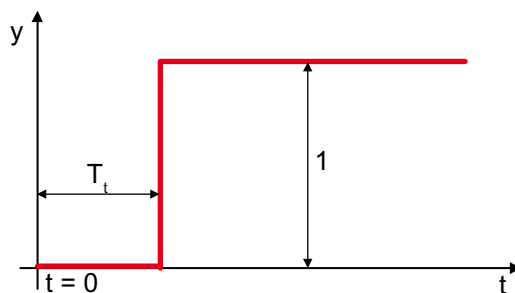


Figure: Time characteristics of DELAY

❗ NOTE

To ensure that the FB works correctly, it must be called in each cycle.

Parameters of the inputs

589

Parameter	Data type	Description
X	WORD	input value
T	TIME	time delay (dead time)

Parameters of the outputs

590

Parameter	Data type	Description
Y	WORD	input value, delayed by the time T

PT1 (FB)

338

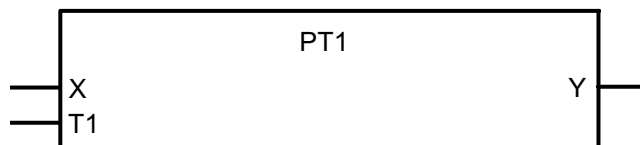
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

341

PT1 handles a controlled system with a first-order time delay.

This FB is a proportional controlled system with a time delay. It is for example used for generating ramps when using the PWM FBs.

The output variable Y of the low-pass filter has the following time characteristics (unit step):

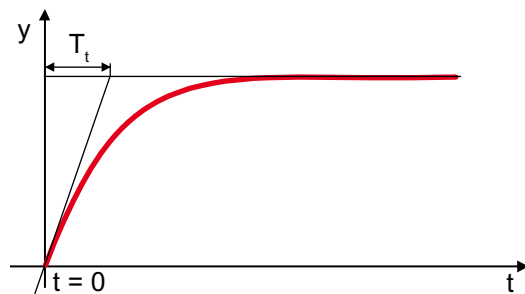


Figure: Time characteristics of PT1

Parameters of the inputs

342

Parameter	Data type	Description
X	INT	input value
T1	TIME	delay time (time constant)

Parameters of the outputs

343

Parameter	Data type	Description
Y	INT	output variable

PID1 (FB)

351

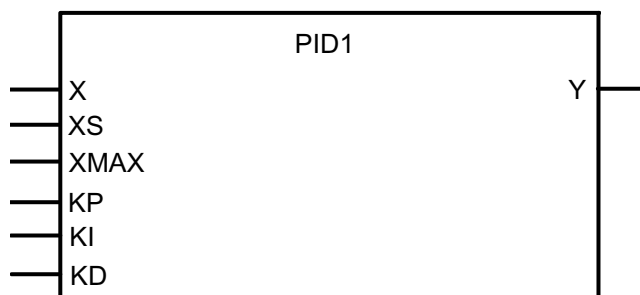
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

354

PID1 handles a PID controller.

The change of the manipulated variable of a PID controller has a **proportional**, **integral** and **differential** component. The manipulated variable changes first by an amount which depends on the rate of change of the input value (D component). After the end of the derivative action time the manipulated variable returns to the value corresponding to the proportional range and changes in accordance with the reset time.

NOTE

The manipulated variable Y is already standardised to the PWM FB (RELOAD value = 65,535). Note the reverse logic:

65,535 = minimum value

0 = maximum value.

Note that the input values KI and KD depend on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

If $X > X_S$, the manipulated variable is increased.

If $X < X_S$, the manipulated variable is reduced.

The manipulated variable Y has the following time characteristics:

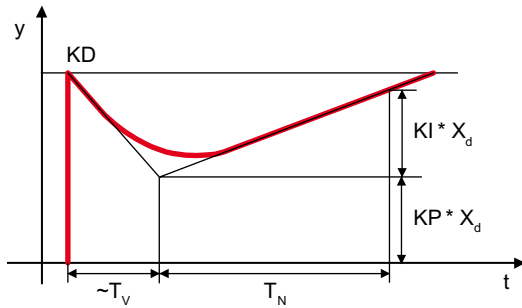


Figure: Typical step response of a PID controller

Parameters of the inputs

355

Parameter	Data type	Description
X	WORD	actual value
XS	WORD	desired value
XMAX	WORD	maximum value of the target value
KP	BYTE	constant of the proportional component
KI	BYTE	integral value
KD	BYTE	proportional component of the differential component

Parameters of the outputs

356

Parameter	Data type	Description
Y	WORD	manipulated variable

Recommended settings

357

KP = 50
KI = 30
KD = 5

With the values indicated above the controller operates very quickly and in a stable way. The controller does not fluctuate with this setting.

- To optimise the controller, the values can be gradually changed afterwards.

PID2 (FB)

9167

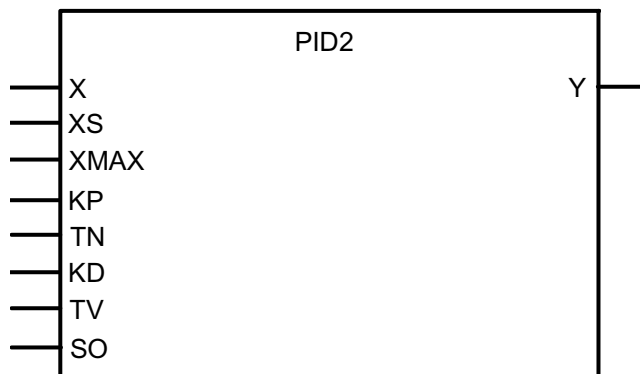
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

347

PID2 handles a PID controller with self optimisation.

The change of the manipulated variable of a PID controller has a **proportional**, **integral** and **differential** component. The manipulated variable changes first by an amount which depends on the rate of change of the input value (D component). After the end of the derivative action time TV the manipulated variable returns to the value corresponding to the proportional component and changes in accordance with the reset time TN.

The values entered at the inputs KP and KD are internally divided by 10. So, a finer grading can be obtained (e.g.: KP = 17, which corresponds to 1.7).

NOTE

The manipulated variable Y is already standardised to the PWM FB (RELOAD value = 65,535). Note the reverse logic:

65,535 = minimum value

0 = maximum value.

Note that the input value KD depends on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

If $X > X_S$, the manipulated variable is increased.

If $X < X_S$, the manipulated variable is reduced.

A reference variable is internally added to the manipulated variable.

$$Y = Y + 65,536 - (XS / XMAX * 65,536).$$

The manipulated variable Y has the following time characteristics.

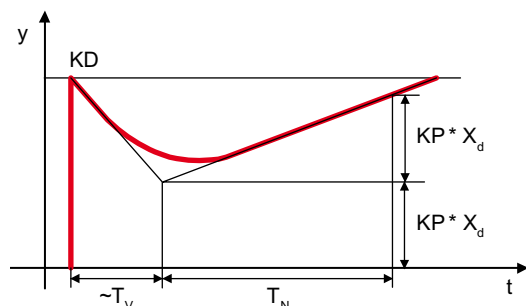


Figure: Typical step response of a PID controller

Parameters of the inputs

348

Parameter	Data type	Description
X	WORD	actual value
XS	WORD	desired value
XMAX	WORD	maximum value of the desired value
KP	BYTE	constant of the proportional component (/10)
TN	TIME	reset time (integral component)
KD	BYTE	proportional component of the differential component (/10)
TV	TIME	derivative action time (differential component)
SO	BOOL	self optimisation

Parameters of the outputs

349

Parameter	Data type	Description
Y	WORD	manipulated variable

Recommended setting

9127
350

- ▶ Select TN according to the time characteristics of the system:
fast system = small TN
slow system = large TN
- ▶ Slowly increment KP gradually, up to a value at which still definitely no fluctuation will occur.
- ▶ Readjust TN if necessary.
- ▶ Add differential component only if necessary:
Select a TV value approx. 2...10 times smaller than TN.
Select a KD value more or less similar to KP.

Note that the maximum control deviation is + 127. For good control characteristics this range should not be exceeded, but it should be exploited to the best possible extent.

Function input SO (self-optimisation) clearly improves the control performance. A precondition for achieving the desired characteristics:

- The controller is operated with I component ($TN \geq 50$ ms)
- Parameters KP and especially TN are already well adjusted to the actual controlled system.
- The control range ($X - XS$) of ± 127 is utilised (if necessary, increase the control range by multiplying X, XS and XMAX).
- ▶ When you have finished setting the parameters, you can set SO = TRUE.
- > This will significantly improve the control performance, especially reducing overshoot.

GLR (FB)

531

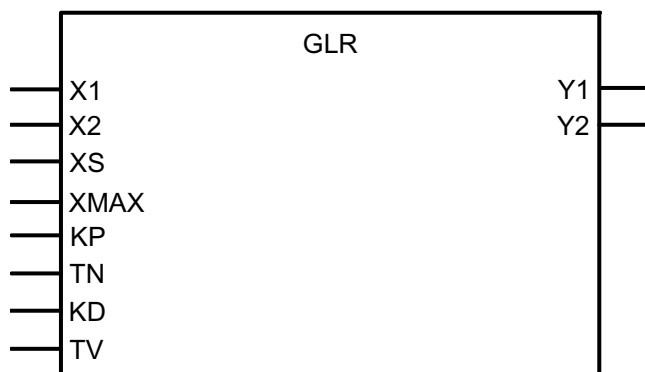
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

534

GLR handles a synchro controller.

The synchro controller is a controller with PID characteristics.

The values entered at the inputs KP and KD are internally divided by 10. So, a finer grading can be obtained (e.g.: KP = 17, which corresponds to 1.7).

The manipulated variable referred to the greater actual value is increased accordingly.

The manipulated variable referred to the smaller actual value corresponds to the reference variable.

Reference variable = $65\,536 - (XS / XMAX * 65\,536)$.

NOTE

The manipulated variables Y1 and Y2 are already standardised to the PWM FB (RELOAD value = 65 535). Note the reverse logic:

65 535 = minimum value

0 = maximum value.

Note that the input value KD depends on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

Parameters of the inputs

535

Parameter	Data type	Description
X1	WORD	actual value channel 1
X2	WORD	actual value channel 2
XS	WORD	desired value = reference variable
XMAX	WORD	maximum value of the desired value
KP	BYTE	constant of the proportional component (/10)
TN	TIME	reset time (integral component)
KD	BYTE	proportional component of the differential component (/10)
TV	TIME	derivative action time (differential component)

Parameters of the outputs

536

Parameter	Data type	Description
Y1	WORD	manipulated variable channel 1
Y2	WORD	manipulated variable channel 2

11 Communication via interfaces

Contents

Use of the serial interface	327
Communication via the internal SSC interface	334

8602

Here we show you functions to use for communication via interfaces.

11.1 Use of the serial interface

Contents

SERIAL_SETUP (FB)	328
SERIAL_TX (FB)	330
SERIAL_RX (FB)	331
SERIAL_PENDING (FB)	333

1600

! NOTE

In principle, the serial interface is not available for the user because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

The serial interface can be used in the application program by means of the following FBs.

11.1.1 SERIAL_SETUP (FB)

302

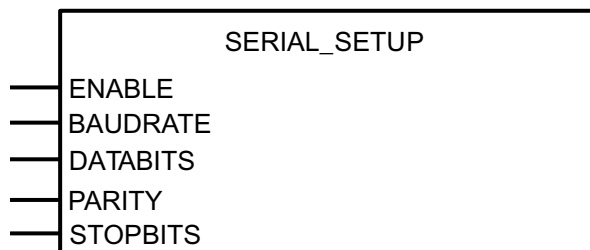
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

305

SERIAL_SETUP initialises the serial RS232 interface.

SERIAL_SETUP sets the serial interface to the indicated parameters. Using the input ENABLE, the FB is activated for one cycle.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

ATTENTION

The driver module of the serial interface can be damaged!

Disconnecting the serial interface while live can cause undefined states which damage the driver module.

► Do not disconnect the serial interface while live.

Parameters of the inputs

306

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only 1 cycle): interface is initialised FALSE: during further processing of the program
BAUDRATE	BYTE	baud rate (permissible values = 9 600, 19 200, 28 800, (57 600)) preset value → data sheet
DATABITS	BYTE	data bits (permissible values: 7 or 8) preset value = 8
PARITY	BYTE	parity (permissible values: 0=none, 1=even, 2=uneven) preset value = 0
STOPBITS	BYTE	stop bits (permissible values: 1 or 2) preset value = 1

11.1.2 SERIAL_TX (FB)

296

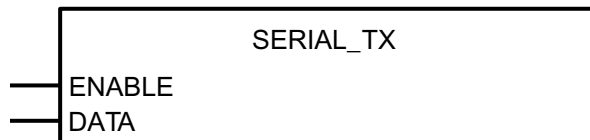
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

299

SERIAL_TX transmits one data byte via the serial RS232 interface.

Using the input ENABLE the transmission can be enabled or blocked.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

! NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

Parameters of the inputs

300

Parameter	Data type	Description
ENABLE	BOOL	TRUE: transmission enabled FALSE: transmission blocked
DATA	BYTE	byte to be transmitted

11.1.3 SERIAL_RX (FB)

308

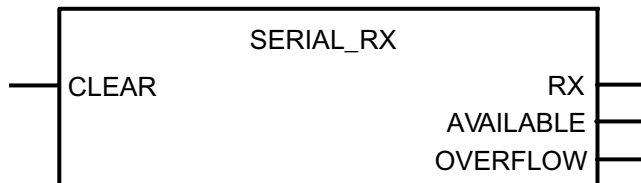
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

311

SERIAL_RX reads a received data byte from the serial receive buffer at each call.

Then, the value of AVAILABLE is decremented by 1.

If more than 1000 data bytes are received, the buffer overflows and data is lost. This is indicated by the bit OVERFLOW.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

Parameters of the inputs

312

Parameter	Data type	Description
CLEAR	BOOL	TRUE: receive buffer is deleted FALSE: this function is not executed

Parameters of the outputs

313

Parameter	Data type	Description
RX	BYTE	byte data received from the receive buffer
AVAILABLE	WORD	number of data bytes received 0 = no valid data available
OVERFLOW	BOOL	TRUE: overflow of the data buffer, loss of data!

Example:

3 bytes are received:

1st call of SERIAL_RX

1 valid value at output RX

→ AVAILABLE = 3

2nd call of SERIAL_RX

1 valid value at output RX

→ AVAILABLE = 2

3rd call of SERIAL_RX

1 valid value at output RX

→ AVAILABLE = 1

4th call of SERIAL_RX

invalid value at the output RX

→ AVAILABLE = 0

If AVAILABLE = 0, the FB can be skipped during processing of the program.

11.1.4 SERIAL_PENDING (FB)

314

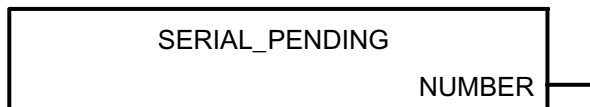
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

317

SERIAL_PENDING determines the number of data bytes stored in the serial receive buffer.

In contrast to SERIAL_RX (→ page [331](#)) the contents of the buffer remain unchanged after calling this FB.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

Parameters of the outputs

319

Parameter	Data type	Description
NUMBER	WORD	number of data bytes received

11.2 Communication via the internal SSC interface

Contents

SSC_RECEIVE (FB)	335
SSC_TRANSMIT (FB).....	337

1618

Available for the following devices:

- ExtendedController: CR0200
- ExtendedSafetyController: CR7200, CR7201

ExtendedControllers and ExtendedSafetyControllers are equipped with an internal SSC interface to enable a communication between the two controller halves. The following FBs serve as support.

11.2.1 SSC_RECEIVE (FB)

254

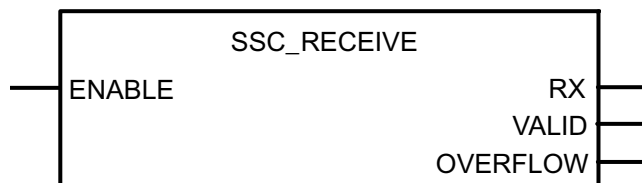
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- ExtendedController: CR0200
- SafetyController: CR7200, CR7201 (**POU not for safety signals!**)

Symbol in CoDeSys:



Description

257

SSC_RECEIVE handles the receipt of data via the internal SSC interface.

This FB is an internal communication function for the ExtendedController. At each call, it reads the transmitted data bytes (max. 16 messages with 20 bytes per control cycle) from the receive buffer. To do so, the input ENABLE must be set to TRUE. If new, valid data has been transmitted, the output VALID is set to TRUE for one cycle.

NOTE

The application programmer must immediately read the received data from the data array and ensure immediate further processing, because the data will be overwritten in the next cycle.

Only as many data messages as can be received by the other controller half may be transmitted via the SSC_TRANSMIT (→ page [337](#)). Otherwise there is the risk of losing data for example due to different cycle times.

An overflow of SSC_RECEIVE may occur if the receiver is the interface slave and if the interface master is running faster (error bit: OVERFLOW).

Parameters of the inputs

258

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active

Parameters of the outputs

259

Parameter	Data type	Description
RX	ARRAY[0...19] OF BYTE	data array [0..19]
VALID	BOOL	TRUE (only 1 cycle): new data has been transmitted
OVERFLOW	BOOL	TRUE: message received could not be entered in the receive buffer FALSE: transfer successfull

11.2.2 SSC_TRANSMIT (FB)

242

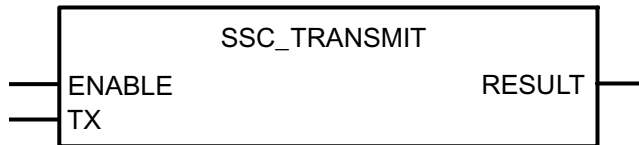
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- ExtendedController: CR0200
- SafetyController: CR7200, CR7201 (**POU not for safety signals!**)

Symbol in CoDeSys:



Description

245

SSC_TRANSMIT handles the transmission of data via the internal SSC interface.

The FB is an internal communication function for the ExtendedController. At each call it transmits the data contained in the data array TX. A maximum of 16 messages with 20 bytes each can be transmitted in one control cycle. For transmission, the input ENABLE must be set to TRUE.

! NOTE

The application programmer must immediately read the received data from the data array and ensure immediate further processing, because the data will be overwritten in the next cycle.

Only as many data messages as can be received by the other controller half may be sent via SSC_TRANSMIT (→ page [337](#)). Otherwise there is the risk of losing data for example due to different cycle times.

An overflow of SSC_TRANSMIT may occur if the transmitter is the interface slave and if the interface master is running slower (information bit: RESULT = FALSE).

Parameters of the inputs

246

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
TX	ARRAY[0...19] OF BYTE	data array [0...19]

Parameters of the outputs

247

Parameter	Data type	Description
RESULT	BOOL	TRUE: message was successfully transferred to the transmission buffer FAULT: transfer faulty

12 Managing the data

Contents

Software reset	338
Reading / writing the system time	340
Saving, reading and converting data in the memory.....	343
Data access and data check	351

8606

Here we show you functions how to read or manage data in the device.

12.1 Software reset

Contents

SOFTRESET (FB).....	339
---------------------	-----

1594

Using this FB the control can be restarted via an order in the application program.

12.1.1 SOFTRESET (FB)

260

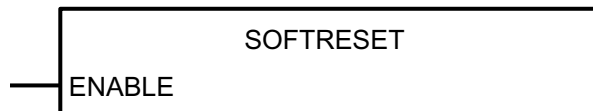
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

263

SOFTRESET leads to a complete reboot of the controller.

The FB can for example be used in conjunction with CANopen if a node reset is to be carried out. The behaviour of the controller after a SOFTRESET corresponds to that after switching the supply voltage off and on.

! NOTE

In case of active communication, the long reset period must be taken into account because otherwise guarding errors will be signalled.

Parameters of the inputs

264

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active

12.2 Reading / writing the system time

Contents

TIMER_READ (FB)	341
TIMER_READ_US (FB)	342

1601

The following FBs offered by **ifm electronic** allow you to read the continually running system time of the controller and to evaluate it in the application program, or to change the system time as needed.

12.2.1 TIMER_READ (FB)

236

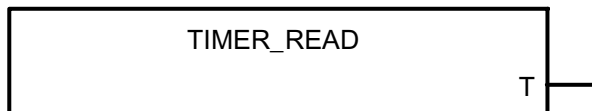
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

239

TIMER_READ reads the current system time.

When the supply voltage is applied, the controller generates a clock pulse which is counted upwards in a register. This register can be read using the FB call and can for example be used for time measurement.

! NOTE

The system timer goes up to $FFFF\ FFFF_{16}$ at the maximum (corresponds to about 49.7 days) and then starts again from 0.

Parameters of the outputs

241

Parameter	Data type	Description
T	TIME	current system time (resolution [ms])

12.2.2 TIMER_READ_US (FB)

657

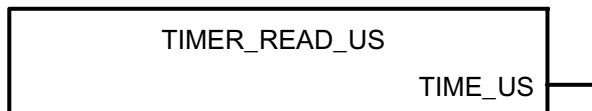
Contained in the library:

ifm_CRnnnn_Vxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

660

TIMER_READ_US reads the current system time in [μs].

When the supply voltage is applied, the device generates a clock pulse which is counted upwards in a register. This register can be read by means of the FB call and can for example be used for time measurement.



Info

The system timer runs up to the counter value 4 294 967 295 μs at the maximum and then starts again from 0.

4 294 967 295 μs = 71 582.8 min = 1 193 h = 49.7 d

Parameters of the outputs

662

Parameter	Data type	Description
TIME_US	DWORD	current system time (resolution [μs])

12.3 Saving, reading and converting data in the memory

Contents	
Automatic data backup	343
Manual data storage	344

1595

12.3.1 Automatic data backup

1596

The **ecomatmobile** devices allow to save data (BOOL, BYTE, WORD, DWORD) non-volatily (= saved in case of voltage failure) in the memory. If the supply voltage drops, the backup operation is automatically started. Therefore it is necessary that the data is filed as RETAIN variables.

The advantage of the automatic backup is that also in case of a sudden voltage drop or an interruption of the supply voltage, the storage operation is triggered and thus the current values of the data are saved (e.g. counter values).

If the supply voltage returns, the saved data is read from the memory via the operating system and written back in the flag area.

12.3.2 Manual data storage

Contents

MEMCPY (FB).....	345
FLASHWRITE (FB).....	346
FLASHREAD (FB).....	348
FRAMWRITE (FB).....	349
FRAMREAD (FB).....	350

1597

Besides the possibility to store the data automatically, user data can be stored manually, via FB calls, in integrated memories from where they can also be read.

Depending on the device the following memories are available:

- **EEPROM memory**

Available for the following devices:

- CabinetController: CR0301, CR0302
- PCB controller: CS0015
- SmartController: CR25nn

Slow writing and reading.

Limited writing and reading frequency.

Any memory area can be selected.

Storing data with E2WRITE.

Reading data with E2READ.

- **FRAM memory**

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- PDM360smart: CR1070, CR1071

Fast writing and reading.

Unlimited writing and reading frequency.

Any memory area can be selected.

Storing data with FRAMWRITE.

Reading data with FRAMREAD.

- **Flash memory**

For all devices.

Fast writing and reading.

Limited writing and reading frequency.

Really useful only for storing large data quantities.

Before anew writing, the memory contents must be deleted.

Storing data with FLASHWRITE.

Reading data with FLASHREAD.

Info

By means of the storage partitioning (→ data sheet or operating instructions) the programmer can find out which memory area is available.

MEMCPY (FB)

409

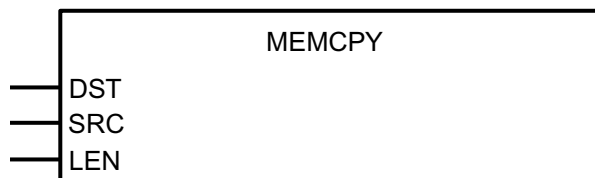
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

412

MEMCPY enables writing and reading different types of data directly in the memory.

The FB writes the contents of the address of SRC to the address DST. In doing so, as many bytes as indicated under LEN are transmitted. So it is also possible to transmit exactly one byte of a word file.

- The address must be determined by means of the operator ADR and assigned to the FB.

Parameters of the inputs

413

Parameter	Data type	Description
DST	DWORD	address of the target variables
SRC	DWORD	address of the source variables
LEN	WORD	number of data bytes

FLASHWRITE (FB)

555

Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

558

⚠ WARNING

Danger due to uncontrollable process operations!

The status of the inputs/outputs is "frozen" during execution of FLASHWRITE.


► Do not execute this FB when the machine is running!

FLASHWRITE enables writing of different data types directly into the flash memory.

The FB writes the contents of the address SRC into the flash memory. In doing so, as many bytes as indicated under LEN are transmitted.

► The address must be determined by means of the operator ADR and assigned to the FB.

An erasing operation must be carried out before the memory is written again. This is done by writing any content to the address "0".

 Info

Using this FB, large data volumes are to be stored during set-up, to which there is only read access in the process.

Parameters of the inputs

559

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
DST	INT	relative start address in the memory memory access only word-by-word; permissible values: 0, 2, 4, 6, 8, ...
LEN	INT	number of data bytes (max. 65 536 bytes)
SRC	DWORD	address of the source variables

FLASHREAD (FB)

561

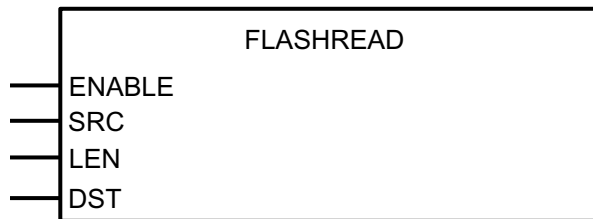
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

564

FLASHREAD enables reading of different types of data directly from the flash memory.

The FB reads the contents as from the address of SRC from the flash memory. In doing so, as many bytes as indicated under LEN are transmitted.

- The address must be determined by means of the operator ADR and assigned to the FB.

Parameters of the inputs

565

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
SRC	INT	relative start address in the memory
LEN	INT	number of data bytes (max. 65 536 bytes)
DST	DWORD	address of the target variables

FRAMWRITE (FB)

543

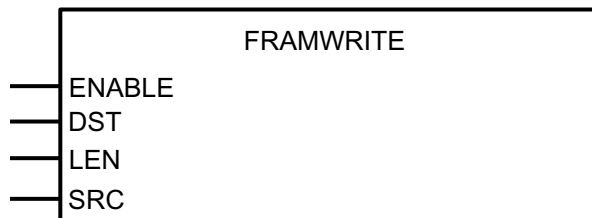
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

546

FRAMWRITE enables the quick writing of different data types directly into the FRAM memory.

The FB writes the contents of the address SRC to the non-volatile FRAM memory. In doing so, as many bytes as indicated under LEN are transmitted.

- The address must be determined by means of the operator ADR and assigned to the FB.

The FRAM memory can be written in several partial segments which are independent of each other. Monitoring of the memory segments must be carried out in the application program.

Parameters of the inputs

547

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
DST	INT	relative start address in the memory (0...3FFF ₁₆)
LEN	INT	number of data bytes CR0303, CR1070, CR1071: max. 128 bytes all other controllers: max. 16 384 bytes
SRC	DINT	address of the source variables

FRAMREAD (FB)

549

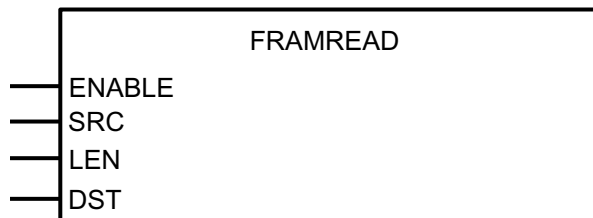
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

552

FRAMREAD enables quick reading of different data types directly from the FRAM memory.

The FB reads the contents as from the address of SRC from the FRAM memory. In doing so, as many bytes as indicated under LEN are transmitted.

► The address must be determined by means of the operator ADR and assigned to the FB.

The FRAM memory can be read in several independent partial segments. Monitoring of the memory segments must be carried out in the application program.

Parameters of the inputs

553

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
SRC	INT	relative start address in the memory (0...3FFF ₁₆)
LEN	INT	number of data bytes CR0303, CR1070, CR1071: max. 128 bytes all other controllers: max. 16 384 bytes
DST	DINT	address of the target variables

12.4 Data access and data check

Contents

SET_DEBUG (FB)	352
SET_IDENTITY (FB)	353
GET_IDENTITY (FB).....	355
SET_PASSWORD (FB)	357
CHECK_DATA (FB)	359

1598

The FBs described in this chapter control the data access and enable a data check.

12.4.1 SET_DEBUG (FB)

290

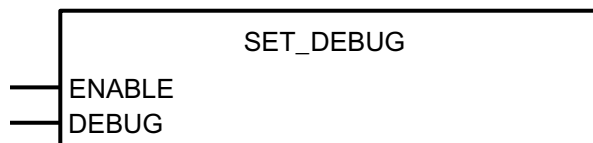
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

293

SET_DEBUG handles the DEBUG mode without active test input (→ chapter TEST mode (→ page [60](#))).

If the input DEBUG of the FB is set to TRUE, the programming system or the downloader, for example, can communicate with the device and execute system commands (e.g. for service functions via the GSM modem CANremote).

! NOTE

In this operating mode a software download is not possible because the test input is not connected to supply voltage.

Parameters of the inputs

294

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active
DEBUG	BOOL	TRUE: debugging via the interfaces possible FALSE: debugging via the interfaces not possible

12.4.2 SET_IDENTITY (FB)

284

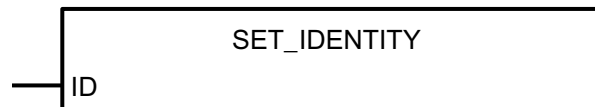
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



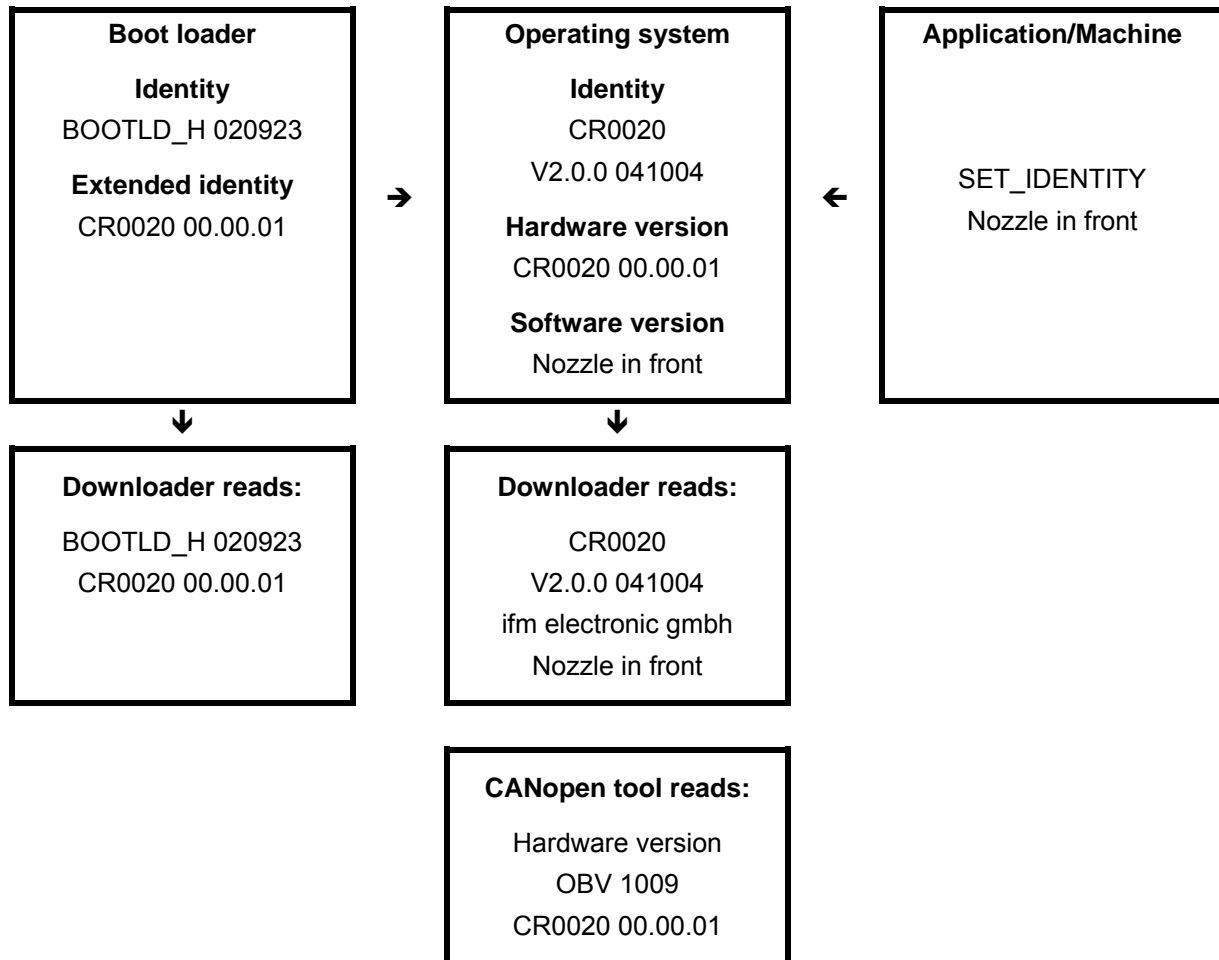
Description

287

SET_IDENTITY sets an application-specific program identification.

Using this FB, a program identification can be created by the application program. This identification (i.e. the software version) can be read via the software tool DOWNLOADER.EXE in order to identify the loaded program.

The following figure shows the correlations of the different identifications as indicated by the different software tools. (Example: ClassicController CR0020):



Parameters of the inputs

288

Parameter	Data type	Description
ID	STRING(80)	any string with a maximum length of 80 characters

12.4.3 GET_IDENTITY (FB)

2212

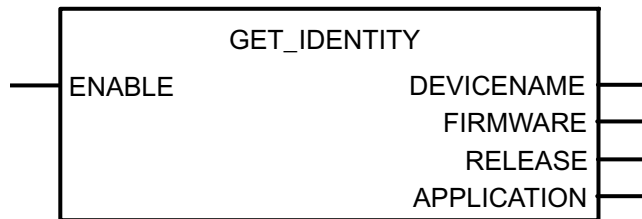
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2344

GET_IDENTITY reads the application-specific program identification stored in the controller.

With this FB the stored program identification can be read by the application program. The following information is available:

- Hardware name and version
e.g.: "CR0032 00.00.01"
- Name of the runtime system
e.g.: "CR0032"
- Version and build of the runtime system
e.g.: "V00.00.01 071128"
- Name of the application
e.g.: "Crane1704"

The name of the application can be changed with SET_IDENTITY (→ page [353](#)).

Parameters of the inputs

2609

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > FB in- and outputs are not active

Parameters of the outputs

2610

Parameter	Data type	Description
DEVICENAME	STRING(31)	hardware name and version as string of max. 31 characters e.g.: "CR0032 00.00.01"
FIRMWARE	STRING(31)	name of the runtime system as string of max. 31 characters e.g.: "CR0032"
RELEASE	STRING(31)	version and build of the runtime system as string of max. 31 characters e.g.: "V00.00.01 071128"
APPLICATION	STRING(79)	name of the application as string of max. 79 characters e.g.: "Crane1704"

12.4.4 SET_PASSWORD (FB)

266

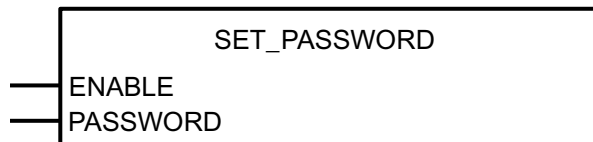
Contained in the library:

ifm_CRnnnn_Vxyyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

269

SET_PASSWORD sets a user password for the program and memory upload with the DOWNLOADER.

If the password is activated, reading of the application program or the data memory with the software tool DOWNLOADER is only possible if the correct password has been entered.

If an empty string (default condition) is assigned to the input PASSWORD, an upload of the application software or of the data memory is possible at any time.

ATTENTION

Please note for CR250n, CR0301, CR0302 and CS0015:

The EEPROM memory module may be destroyed by the permanent use of this unit!

- Only carry out the unit **once** during initialisation in the first program cycle!
- Afterwards block the unit again with ENABLE = FALSE!

NOTE

The password is reset when loading a new application program.

Parameters of the inputs

270

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only 1 cycle): ID set FALSE: unit is not executed
PASSWORD	STRING	password (maximum string length 16)

12.4.5 CHECK_DATA (FB)

603

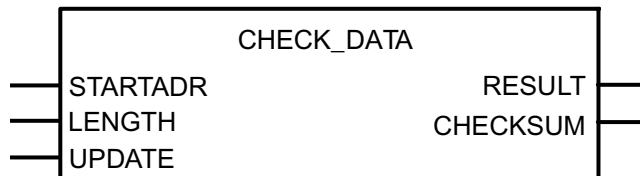
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

606

CHECK_DATA stores the data in the application data memory via a CRC code.

The FB serves for monitoring a range of the data memory (possible WORD addresses as from %MW0) for unintended changes to data in safety-critical applications. To do so, the FB determines a CRC checksum of the indicated data range.

- The address must be determined by means of the operator ADR and assigned to the FB.
- In addition, the number of data bytes LENGTH (length as from the STARTDR) must be indicated.

If the input UPDATE = FALSE and data in the memory are changed inadvertently, RESULT = FALSE. The result can then be used for further actions (e.g. deactivation of the outputs).

Data changes in the memory (e.g. by the application program or **ecomatmobile** device) are only permitted if the output UPDATE is set to TRUE. The value of the checksum is then recalculated. The output RESULT is permanently TRUE again.

! NOTE

This FB is a safety function. However, the controller does not automatically become a safety controller by using this FB. Only a tested and approved controller with a special operating system can be used as safety controller.

Parameters of the inputs

607

Parameter	Data type	Description
STARTADR	DINT	start address of the monitored data memory (WORD address as from %MW0)
LENGTH	WORD	length of the monitored data memory in [byte]
UPDATE	BOOL	TRUE: changes to data permissible FALSE: changes to data not permitted

Parameters of the outputs

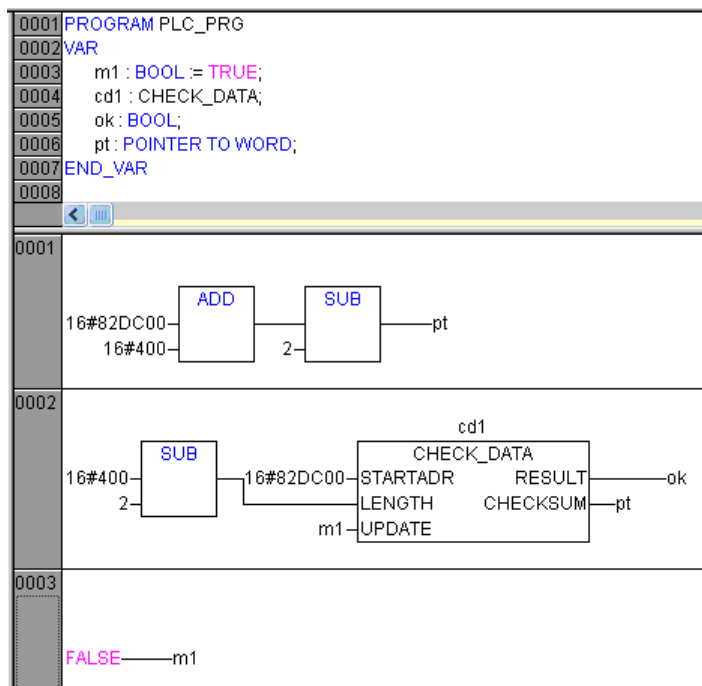
608

Parameter	Data type	Description
RESULT	BOOL	TRUE: CRC checksum ok FALSE: CRC checksum faulty (data modified)
CHECKSUM	WORD	result of the CRC checksum evaluation

Example: CHECK_DATA

4168

In the following example the program determines the checksum and stores it in the RAM via pointer pt:



NOTE: The method shown here is not suited for the flash memory.

13 Optimising the PLC cycle

Contents

Processing interrupts.....	361
----------------------------	-----

8609

Here we show you functions to optimise the PLC cycle.

13.1 Processing interrupts

Contents

SET_INTERRUPT_XMS (FB).....	362
SET_INTERRUPT_I (FB).....	365

1599

The PLC cyclically processes the stored application program in its full length. The cycle time can vary due to program branchings which depend e.g. on external events (= conditional jumps). This can have negative effects on certain functions.

By means of systematic interrupts of the cyclic program it is possible to call time-critical processes independently of the cycle in fixed time periods or in case of certain events.

Since interrupt functions are principally not permitted for SafetyControllers, they are thus not available.

13.1.1 SET_INTERRUPT_XMS (FB)

272

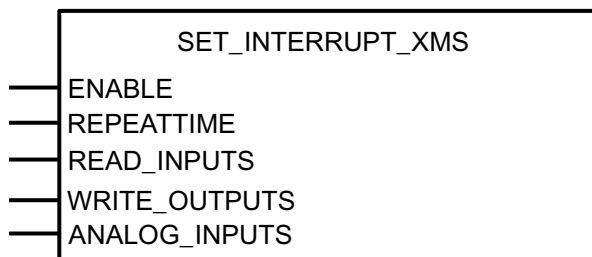
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

275

SET_INTERRUPT_XMS handles the execution of a program part at an interval of x ms.

In the conventional PLC the cycle time is decisive for real-time monitoring. So, the PLC is at a disadvantage as compared to customer-specific controllers. Even a "real-time operating system" does not change this fact when the whole application program runs in one single block which cannot be changed.

A possible solution would be to keep the cycle time as short as possible. This often leads to splitting the application up to several control cycles. This, however, makes programming complex and difficult.

Another possibility is to call a certain program part at fixed intervals (every x ms) independently of the control cycle.

The time-critical part of the application is integrated by the user in a block of the type PROGRAM (PRG). This block is declared as the interrupt routine by calling SET_INTERRUPT_XMS once (during initialisation). As a consequence, this program block is always processed after the REPEATTIME has elapsed (every x ms). If inputs and outputs are used in this program part, they are also read and written in the defined cycle. Reading and writing can be stopped via the FB inputs READ_INPUTS, WRITE_OUTPUTS and ANALOG_INPUTS.

So, in the program block all time-critical events can be processed by linking inputs or global variables and writing outputs. So, timers can be monitored more precisely than in a "normal cycle".

NOTE

To avoid that the program block called by interrupt is additionally called cyclically, it should be skipped in the cycle (with the exception of the initialisation call).

Several timer interrupt blocks can be active. The time requirement of the interrupt functions must be calculated so that all called functions can be executed. This in particular applies to calculations, floating point arithmetic or controller functions.

Please note: In case of a high CAN bus activity the set REPEATTIME may fluctuate.

NOTE

The uniqueness of the inputs and outputs in the cycle is affected by the interrupt routine. Therefore only part of the inputs and outputs is serviced. If initialised in the interrupt program, the following inputs and outputs will be read or written.

Inputs, digital:

%IX0.0...%IX0.7 (CRnn32)

%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)

%IX0.0, %IX0.8 (SmartController)

IN08...IN11 (CabinetController)

IN0...IN3 (PCB controller)

Inputs, analogue:

%IX0.0...%IX0.7 (CRnn32)

All channels (selection bit-coded) (all other controller)

Outputs, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)

%QX0.0, %QX0.8 (SmartController)

OUT00...OUT03 (CabinetController)

OUT0...OUT7 (PCB controller)

Global variants, too, are no longer unique if they are accessed simultaneously in the cycle and by the interrupt routine. This problem applies in particular to larger data types (e.g. DINT).

All other inputs and outputs are processed once in the cycle, as usual.

Parameters of the inputs

276

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only 1 cycle): changes to data allowed FALSE: changes to data not allowed (during processing of the program)
REPEATTIME	TIME	Time window during which the interrupt is triggered.
READ_INPUTS	BOOL	TRUE: inputs integrated into the routine are read (if necessary, set inputs to IN_FAST). FALSE: this function is not executed
WRITE_OUTPUTS	BOOL	TRUE: outputs integrated into the routine are written to. FALSE: this function is not executed
ANALOG_INPUTS	BYTE	TRUE: analogue inputs integrated into the routine are read and the raw value of the voltage is transferred to the system flags ANALOG_IRQxx FALSE: this function is not executed

13.1.2 SET_INTERRUPT_I (FB)

278

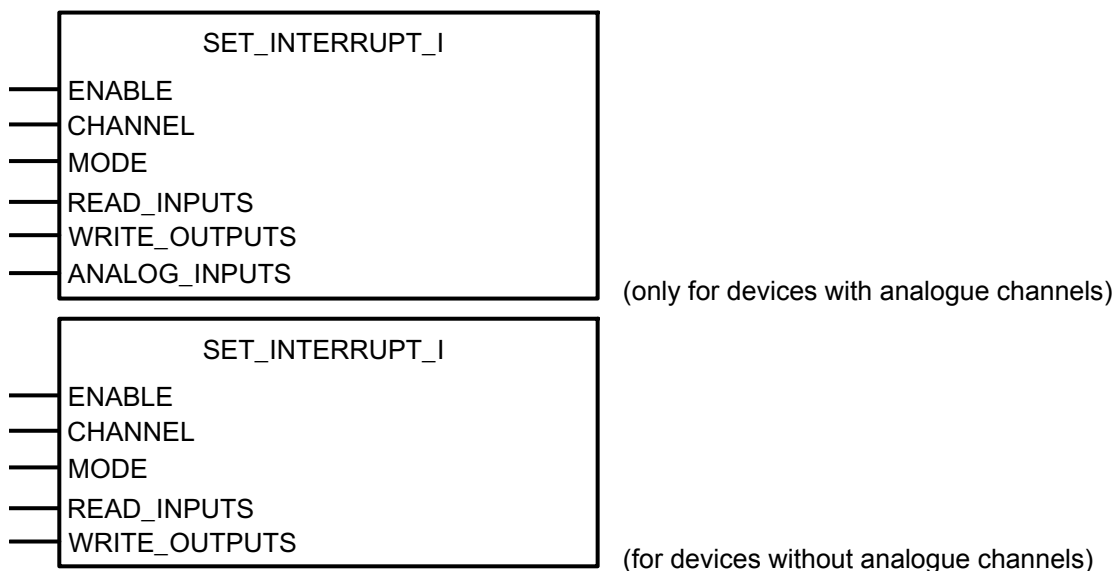
Contained in the library:

ifm_CRnnnn_Vxxyzz.LIB

Available for the following devices:

- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

281

SET_INTERRUPT_I handles the execution of a program part by an interrupt request via an input channel.

In the conventional PLC the cycle time is decisive for real-time monitoring. So the PLC is at a disadvantage as compared to customer-specific controllers. Even a "real-time operating system" does not change this fact when the whole application program runs in one single block which cannot be changed.

A possible solution would be to keep the cycle time as short as possible. This often leads to splitting the application up to several control cycles. This, however, makes programming complex and difficult.

Another possibility is to call a certain program part only upon request by an input pulse independently of the control cycle.

The time-critical part of the application is integrated by the user in a block of the type **PROGRAM (PRG)**. This block is declared as the interrupt routine by calling **SET_INTERRUPT_I** once (during initialisation). As a consequence, this program block will always be executed if an edge is detected on the input **CHANNEL**. If inputs and outputs are used in this program part, these are also read and written in the interrupt routine, triggered by the input edge. Reading and writing can be stopped via the FB inputs **READ_INPUTS**, **WRITE_OUTPUTS** and **ANALOG_INPUTS**.

365

So in the program block all time-critical events can be processed by linking inputs or global variables and writing outputs. So FBs can only be executed if actually called by an input signal.

❗ NOTE

The program block should be skipped in the cycle (except for the initialisation call) so that it is not cyclically called, too.

The input (CHANNEL) monitored for triggering the interrupt cannot be initialised and further processed in the interrupt routine.

The inputs must be in the operating mode IN_FAST, otherwise the interrupts cannot be read.

❗ NOTE

The uniqueness of the inputs and outputs in the cycle is affected by the interrupt routine. Therefore only part of the inputs and outputs is serviced. If initialised in the interrupt program, the following inputs and outputs will be read or written.

Inputs, digital:

%IX0.0...%IX0.7 (CRnn32)

%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)

%IX0.0, %IX0.8 (SmartController)

IN08...IN11 (CabinetController)

IN0...IN3 (PCB controller)

Inputs, analogue:

%IX0.0...%IX0.7 (CRnn32)

All channels (selection bit-coded) (all other controller)

Outputs, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)

%QX0.0, %QX0.8 (SmartController)

OUT00...OUT03 (CabinetController)

OUT0...OUT7 (PCB controller)

Global variants, too, are no longer unique if they are accessed simultaneously in the cycle and by the interrupt routine. This problem applies in particular to larger data types (e.g. DINT).

All other inputs and outputs are processed once in the cycle, as usual.

Parameters of the inputs

282

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only for 1 cycle): changes to data permissible FALSE: changes to data not permitted (during processing of the program)
CHANNEL	BYTE	interrupt input Classic/ExtendedController: 0 = %IX1.4 1 = %IX1.5 2 = %IX1.6 3 = %IX1.7 SmartController: 0 = %IX0.0 1 = %IX0.8 CabinetController: 0 = IN08 (etc.) 3 = IN11 CS0015: 0 = IN0 (etc.) 3 = IN3
MODE	BYTE	type of edge at the input CHANNEL which triggers the interrupt 1 = rising edge 2 = falling edge 3 = rising and falling edge
READ_INPUTS	BOOL	TRUE: inputs integrated into the routine are read (if necessary, set inputs to IN_FAST) FALSE: this function is not executed
WRITE_OUTPUTS	BOOL	TRUE: outputs integrated into the routine are written FALSE: this function is not executed
ANALOG_INPUTS	BYTE	(only for devices with analogue channels) selection of the inputs bit-coded: 0 ₁₀ = no input selected 1 ₁₀ = 1st analogue input selected (0000 0001 ₂) 2 ₁₀ = 2nd analogue input selected (0000 0010 ₂) ... 128 ₁₀ = 8th analogue input selected (1000 0000 ₂) A combination of the inputs is possible via an OR operation of the values. Example: Select 1st and 3rd analogue input: (0000 0001 ₂) OR (0000 0100 ₂) = (0000 0101 ₂) = 5 ₁₀

14 Annex

Contents

Address assignment and I/O operating modes	368
System flags	374
Overview of the files and libraries used	376
Troubleshooting	382

1664

Additionally to the indications in the data sheets you find summary tables in the annex.

14.1 Address assignment and I/O operating modes

Contents

Addresses / I/O variables	368
Possible operating modes inputs / outputs	370
Address assignment inputs / outputs	372

1656

→ also data sheet

14.1.1 Addresses / I/O variables

3922

NOTE: If CR7505+CR7506: Only the ports 0...2 are available

Port	IEC address 1)	I/O variable or Configuration variable	Description
0	%IB0	I0	Input byte 0 (%IX0.00...%IX0.07)
0	%QB4	I00_MODE	Configuration byte for %IX0.00
0	%QB5	I01_MODE	Configuration byte for %IX0.01
0	%QB6	I02_MODE	Configuration byte for %IX0.02
0	%QB7	I03_MODE	Configuration byte for %IX0.03
0	%QB8	I04_MODE	Configuration byte for %IX0.04
0	%QB9	I05_MODE	Configuration byte for %IX0.05
0	%QB10	I06_MODE	Configuration byte for %IX0.06
0	%QB11	I07_MODE	Configuration byte for %IX0.07
0	Flag byte*)	ERROR_I0	Error byte inputs port 0 (%IX0.00...%IX0.07)
1	%IB1	I1	Input byte 1 (%IX0.08...%IX0.15)
1	%QB16	I14_MODE	Configuration byte for %IX0.12
1	%QB17	I15_MODE	Configuration byte for %IX0.13
1	%QB18	I16_MODE	Configuration byte for %IX0.14
1	%QB19	I17_MODE	Configuration byte for %IX0.15
1	Flag byte*)	ERROR_I1	Error byte inputs port 1 (%IX0.08...%IX0.15)
1	%QB0	Q1Q2	Output byte 0 (%QX0.00...%QX0.07)
1	%QB40	Q10_MODE	Configuration byte for %QX0.00
1	%QB41	Q11_MODE	Configuration byte for %QX0.01
1	%QB42	Q12_MODE	Configuration byte for %QX0.02
1	%QB43	Q13_MODE	Configuration byte for %QX0.03
1	Flag byte*)	ERROR_SHORT_Q1Q2	Error byte ports 1+2 short circuit (%QX0.00...%QX0.07)
1	Flag byte*)	ERROR_BREAK_Q1Q2	Error byte ports 1+2 interruption (%QX0.00...%QX0.07)

Port	IEC address ¹⁾	I/O variable or Configuration variable	Description
2	%IB2	I2	Input byte 2 (%IX1.00...%IX1.07)
2	%QB44	Q20_MODE	Configuration byte for %QX0.04
2	%QB45	Q21_MODE	Configuration byte for %QX0.05
2	%QB46	Q22_MODE	Configuration byte for %QX0.06
2	%QB47	Q23_MODE	Configuration byte for %QX0.07
2	%QB20	I24_MODE	Configuration byte for %IX1.04
2	%QB21	I25_MODE	Configuration byte for %IX1.05
2	%QB22	I26_MODE	Configuration byte for %IX1.06
2	%QB23	I27_MODE	Configuration byte for %IX1.07
2	Flag byte*)	ERROR_I2	Error byte inputs port 2 (%IX1.00...%IX1.07)
3	%IB3	I3	Input byte 3 (%IX1.08...%IX1.15)
3	%QB24	I30_MODE	Configuration byte for %IX1.08
3	%QB25	I31_MODE	Configuration byte for %IX1.09
3	%QB26	I32_MODE	Configuration byte for %IX1.10
3	%QB27	I33_MODE	Configuration byte for %IX1.11
3	%QB28	I34_MODE	Configuration byte for %IX1.12
3	%QB29	I35_MODE	Configuration byte for %IX1.13
3	%QB30	I36_MODE	Configuration byte for %IX1.14
3	%QB31	I37_MODE	Configuration byte for %IX1.15
3	Flag byte*)	ERROR_I3	Error byte inputs port 3 (%IX1.08...%IX1.15)
3	%QB1	Q3	Output byte 1 (%QX0.0...%QX0.7)
3	%QB48	Q30_MODE	Configuration byte for %QX0.08
3	%QB59	Q31_MODE	Configuration byte for %QX0.09
3	%QB50	Q32_MODE	Configuration byte for %QX0.10
3	%QB51	Q33_MODE	Configuration byte for %QX0.11
3	%QB52	Q34_MODE	Configuration byte for %QX0.12
3	%QB53	Q35_MODE	Configuration byte for %QX0.13
3	%QB54	Q36_MODE	Configuration byte for %QX0.14
3	%QB55	Q37_MODE	Configuration byte for %QX0.15
3	Flag byte*)	ERROR_SHORT_Q3	Error byte port 3 short circuit (%QX0.08...%QX0.15)
3	Flag byte*)	ERROR_BREAK_Q3	Error byte port 3 interruption (%QX0.08...%QX0.15)
4	%IB4	I4	Input byte 4 (%IX2.00...%IX2.07)
4	%QB32	I40_MODE	Configuration byte for %IX2.00
4	%QB33	I41_MODE	Configuration byte for %IX2.01
4	%QB34	I42_MODE	Configuration byte for %IX2.02
4	%QB35	I43_MODE	Configuration byte for %IX2.03
4	%QB36	I44_MODE	Configuration byte for %IX2.04
4	%QB37	I45_MODE	Configuration byte for %IX2.05
4	%QB38	I46_MODE	Configuration byte for %IX2.06
4	%QB39	I47_MODE	Configuration byte for %IX2.07
4	Flag byte*)	ERROR_I4	Error byte inputs port 4 (%IX2.00...%IX2.07)
4	%QB2	Q4	Output byte 2 (%QX1.00...%QX1.07)
4	%QB56	Q40_MODE	Configuration byte for %QX1.00
4	%QB57	Q41_MODE	Configuration byte for %QX1.01
4	%QB58	Q42_MODE	Configuration byte for %QX1.02
4	%QB59	Q43_MODE	Configuration byte for %QX1.03
4	%QB60	Q44_MODE	Configuration byte for %QX1.04

Port	IEC address ¹⁾	I/O variable or Configuration variable	Description
4	%QB61	Q45_MODE	Configuration byte for %QX1.05
4	%QB62	Q46_MODE	Configuration byte for %QX1.06
4	%QB63	Q47_MODE	Configuration byte for %QX1.07
4	Flag byte*)	ERROR_SHORT_Q4	Error byte port 4 short circuit (%QX1.00...%QX1.07)
4	Flag byte*)	ERROR_BREAK_Q4	Error byte port 4 interruption (%QX1.00...%QX1.07)

¹⁾ IEC addresses of the configuration parameters.

*) IEC addresses can vary according to the control configuration.

For the ExtendedController when used in master/slave operation the following applies for the ports 5...9:

indicated IEC address of the configuration parameters = %IB or %QB **plus 64**,

indicated I/O or configuration variable = NAME_**E**.

14.1.2 Possible operating modes inputs / outputs

3924

❗ NOTE

The input/output operating modes are set best via the **ifm** templates. **For safety signals manual configurations of the inputs and outputs are not allowed!**

When the **ecomatmobile** CD "Software, Tools and Documentation" is installed, projects with templates have been stored in the program directory of your PC:

...\ifm electronic\CoDeSys V...\Projects\Template_CDV...

- ▶ Open the requested template in CoDeSys via:
[File] > [New from template...]
- > CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page [65](#))

NOTE: If CR7505+CR7506: only the ports 0...2 are available

Inputs	Operating mode	Config. value	Outputs	Operating mode	Config. value
I00...I07	IN_NOMODE	0			
	IN_DIGITAL_H (plus)	1 (default)			
	IN_CURRENT	4			
	IN_VOLTAGE10	8			
	IN_VOLTAGE30	16 (default)			
	IN_RATIO	32			
	IN_SAFETY (DIAGNOSTIC)	64			
I10...I13	IN_NOMODE	0 (default)	Q10...Q13	OUT_NOMODE	0
	IN_DIGITAL_H (plus)	1		OUT_DIGITAL_H	1 (default)
				OUT_CURRENT	4
				OUT_DIAGNOSTIC	64
				OUT_OVERLOAD_PROTECTION	128
I14...I17	IN_NOMODE	0			
	IN_DIGITAL_H (plus)	1 (default)			
	IN_SAFETY (DIAGNOSTIC)	64			
	IN_FAST	128			
			Q20...Q23	OUT_NOMODE	0
				OUT_DIGITAL_H	1 (default)
				OUT_CURRENT	4
				OUT_SAFETY	32
				OUT_DIAGNOSTIC	64
				OUT_OVERLOAD_PROTECTION	128
I24...I27	IN_NOMODE	0			
	IN_DIGITAL_H (plus)	1 (default)			
	IN_DIGITAL_L (minus)	2			
	IN_DIAGNOSTIC	64			
	IN_FAST	128			
I30...I37	IN_NOMODE	0	Q30...Q37	OUT_NOMODE	0
	IN_DIGITAL_H (plus)	1 (default)		OUT_DIGITAL_H	1 (default)
	IN_DIGITAL_L (minus)	2		OUT_DIAGNOSTIC	64
	IN_DIAGNOSTIC	64			
			Q40, Q43, Q44, Q47	OUT_NOMODE	0
				OUT_DIGITAL_H	1 (default)
				OUT_SAFETY	32
				OUT_DIAGNOSTIC	64
			Q41, Q42, Q45, Q46	OUT_NOMODE	0
				OUT_DIGITAL_H	1 (default)
				OUT_DIGITAL_L	2
				OUT_SAFETY	32
				OUT_DIAGNOSTIC	64

Possible configuration combinations (where permissible) are created by adding the values.

14.1.3 Address assignment inputs / outputs

3923

NOTE: If CR7505+CR7506: only the ports 0...2 are available

Port	IEC address	I/O variable	Configuration variable	Default value	Possible configuration
0	%IX0.00 %IW3	I00	I00_MODE	17	L digital analogue U/I / safety
0	%IX0.01 %IW4	I01	I01_MODE	17	L digital analogue U/I / safety
0	%IX0.02 %IW5	I02	I02_MODE	17	L digital analogue U/I / safety
0	%IX0.03 %IW6	I03	I03_MODE	17	L digital analogue U/I / safety
0	%IX0.04 %IW7	I04	I04_MODE	17	L digital analogue U/I / safety
0	%IX0.05 %IW8	I05	I05_MODE	17	L digital analogue U/I / safety
0	%IX0.06 %IW9	I06	I06_MODE	17	L digital analogue U/I / safety
0	%IX0.07 %IW10	I07	I07_MODE	17	L digital analogue U/I / safety
1	%IX0.08 %QX0.00	I10 Q10	- Q10_MODE	- 1	Off / L digital Off / H digital / PWM / PWMi
1	%IX0.09 %QX0.01	I11 Q11	- Q11_MODE	- 1	Off / L digital Off / H digital / PWM / PWMi
1	%IX0.10 %QX0.02	I12 Q12	- Q12_MODE	- 1	Off / L digital Off / H digital / PWM / PWMi
1	%IX0.11 %QX0.03	I13 Q13	- Q13_MODE	- 1	Off / L digital Off / H digital / PWM / PWMi
1	%IX0.12	I14	I14_MODE	1	L digital / FRQ0 / safety
1	%IX0.13	I15	I15_MODE	1	L digital / FRQ1 / safety
1	%IX0.14	I16	I16_MODE	1	L digital / FRQ2 / safety
1	%IX0.15	I17	I17_MODE	1	L digital / FRQ3 / safety
2	%QX0.04	Q20	Q20_MODE	1	Off / H digital / PWM / PWMi / safety
2	%QX0.05	Q21	Q21_MODE	1	Off / H digital / PWM / PWMi / safety
2	%QX0.06	Q22	Q22_MODE	1	Off / H digital / PWM / PWMi / safety
2	%QX0.07	Q23	Q23_MODE	1	Off / H digital / PWM / PWMi / safety
2	%IX1.04	I24	I24_MODE	1	L digital / H digital / CYL0 / safety
2	%IX1.05	I25	I25_MODE	1	L digital / H digital / CYL1 / safety
2	%IX1.06	I26	I26_MODE	1	L digital / H digital / CYL2 / safety
2	%IX1.07	I27	I27_MODE	1	L digital / H digital / CYL3 / safety
3	%IX1.08 %QX0.08	I30 Q30	I30_MODE Q30_MODE	0 1	Off / L digital / H digital Off / H digital
3	%IX1.09 %QX0.09	I31 Q31	I31_MODE Q31_MODE	0 1	Off / L digital / H digital Off / only H digital
3	%IX1.10 %QX0.10	I32 Q32	I32_MODE Q32_MODE	0 1	Off / L digital / H digital Off / only H digital
3	%IX1.11 %QX0.11	I33 Q33	I33_MODE Q33_MODE	0 1	Off / L digital / H digital Off / H digital
3	%IX1.12 %QX0.12	I34 Q34	I34_MODE Q34_MODE	0 1	Off / L digital / H digital Off / H digital
3	%IX1.13 %QX0.13	I35 Q35	I35_MODE Q35_MODE	0 1	Off / L digital / H digital Off / H digital

Port	IEC address	I/O variable	Configuration variable	Default value	Possible configuration
3	%IX1.14 %QX0.14	I36 Q36	I36_MODE Q36_MODE	0 1	Off / L digital / H digital Off / H digital
3	%IX1.15 %QX0.15	I37 Q37	I37_MODE Q37_MODE	0 1	Off / L digital / H digital Off / H digital
4	%QX1.00	Q40	Q40_MODE	1	Off / H digital / PWM / safety
4	%QX1.01	Q41	Q41_MODE	1	Off / H digital / L digital / H link / safety
4	%QX1.02	Q42	Q42_MODE	1	Off / H digital / L digital / H link / safety
4	%QX1.03	Q43	Q43_MODE	1	Off / H digital / PWM / safety
4	%QX1.04	Q44	Q44_MODE	1	Off / H digital / PWM / safety
4	%QX1.05	Q45	Q45_MODE	1	Off / H digital / L digital / H link / safety
4	%QX1.06	Q46	Q46_MODE	1	Off / H digital / L digital / H link / safety
4	%QX1.07	Q47	Q47_MODE	1	Off / H digital / PWM / safety

For the ExtendedController when used in master/slave operation the following applies for ports 5...9:
indicated IEC address = %IX or %QX **plus 32.00**,
indicated IEC address of the configuration parameters = %IB or %QB **plus 64**,
indicated I/O or configuration variable = NAME_**E**.

PWM description → chapter PWM signal processing (→ page [273](#))

PWM_i description → chapter Current control with PWM (→ page [285](#))

FRQ/CYL description → chapter Counter functions for frequency and period measurement
(→ page [258](#))

H link description → chapter "Motor control using the H link"

Info

The default value 17 of the configuration bytes I00_MODE to I07_MODE is composed as below
(→ chapter Possible operating modes inputs / outputs (→ page [370](#))):

IN_DIGITAL_H (1) AND IN_VOLTAGE30 (16) => 17.

The other values "IN_VOLTAGE10" etc. can only be set in the operating mode "Analogue Input".

14.2 System flags

3920

(→ chapter Error messages (→ page [114](#)))

System flags	Type	Description
CANx_BAUDRATE	WORD	CAN interface x: Baud rate set
CANx_BUSOFF	BOOL	CAN interface x: Error "CAN-Bus off"
CANx_ERRORCOUNTER_RX ¹⁾	BYTE	CAN interface x: Error counter receiver
CANx_ERRORCOUNTER_TX ¹⁾	BYTE	CAN interface x: Error counter transmitter
CANx_LASTERROR ¹⁾	BYTE	CAN interface x: Error number of the last CAN transmission: 0 = no error ≠0 → CAN specification → LEC
CANx_WARNING	BOOL	CAN interface x: Warning level reached (≥ 96)
CLAMP_15	BOOL	Monitoring terminal 15
DOWNLOADID	WORD	Currently set download identifier
ERROR ²⁾	BOOL	Set ERROR bit / switch off relay
ERROR_ADDRESS	BOOL	Addressing error
ERROR_ANALOG	BOOL	Error in analogue conversion
ERROR_BREAK_Qx	BYTE	Wire break error on the output group x
ERROR_CAN_SAFETY	BOOL	SCT, SRVT and data error
ERROR_CO_CPU	BOOL	Error in the co-processor
ERROR_CPU	BOOL	CPU error
ERROR_DATA	BOOL	System data faulty
ERROR_INSTRUCTION_TIME	BOOL	Error in processing time
ERROR_IO	BOOL	Group error wire break, short circuit, cross fault
ERROR_Ix	BYTE	Periphery fault on input group x
ERROR_MEMORY	BOOL	Memory error
ERROR_OUTPUTBLANKING	BOOL	Cross fault on one of the safety outputs
ERROR_POWER	BOOL	Undervoltage/overvoltage error on pin 23
ERROR_RELAIS	BOOL	Error relay control with failure of VBB ₀
ERROR_SHORT_Qx	BYTE	Short circuit error on the output group x
ERROR_TEMPERATUR	BOOL	Excessive-temperature error (>85°C)
ERROR_TIME_BASE	BOOL	Error internal system time
ERROR_VBBR	BOOL	Supply voltage error VBB _R
LED	WORD	LED colour for "active" (= on)
LED_MODE	WORD	Flashing frequency from the data structure "LED_MODES"
LED_X	WORD	LED colour for "pause" (= out)
RELAIS	BOOL	Monitoring relay for VBB _R
RELAY_CLAMP_15	BOOL	Relay terminal 15 (pin 5)
SERIAL_MODE	BOOL	Switch on serial communication
SERIALBAUDRATE	WORD	Baud rate of the RS232 interface
SUPPLY_VOLTAGE	WORD	Supply voltage on VBBs in [mV]
TEST	BOOL	Release programming mode

CANx designates the number of the CAN interface (CAN 1...x, depending on the device).

Ix or Qx designates the number of the input or output group (word 0...x, depending on the unit).

¹⁾ Access to these flags requires detailed knowledge of the CAN controller and is normally not required.

²⁾ By setting the ERROR system flag the ERROR output (terminal 13) is set to FALSE. In the "error-free state" the ERROR output is TRUE (negative logic).

! NOTE

For programming you should use only symbol names since the corresponding flag addresses could change when the controller configuration is extended.

! NOTE

ExtendedController: Symbol names extended by an "_E" (e.g. ERRORPOWER_E) designate system addresses in the slave module of the ExtendedControllers. They have the same functions as the symbol names in the master module.

14.3 Overview of the files and libraries used

Contents

General overview	376
What are the individual files and libraries used for?	378

2711

(as on 02 June 2010)

Depending on the unit and the desired function, different libraries and files are used. Some are automatically loaded, others must be inserted or loaded by the programmer.

Installation of the files and libraries in the device:

Factory setting: the device contains only the boot loader.

- ▶ Load the operating system (*.H86 or *.HEX)
- ▶ Create the project (*.PRO) in the PC: enter the target (*.TRG)
- ▶ Additionally depending on device and target:
Define the PLC configuration (*.CFG)
- > CoDeSys integrates the files belonging to the target into the project:
*.TRG, *.CFG, *.CHM, *.INI, *.LIB
- ▶ If required, add further libraries to the project (*.LIB).

Certain libraries automatically integrate further libraries into the project.

Some FBs in **ifm** libraries (ifm_*.LIB) e.g. are based on FBs in CoDeSys libraries (3S_*.LIB).

14.3.1 General overview

2712

File name	Description and memory location ³⁾
ifm_CRnnnn_Vxyyzz.CFG ¹⁾ ifm_CRnnnn_Vxx.CFG ²⁾	PLC configuration per device only 1 device-specific file includes: IEC and symbolic addresses of the inputs and outputs, the flag bytes as well as the memory allocation ...\CoDeSys V*\Targets\ifm\ifm_CRnnnncfg\Vxyyzz
CAA-*.CHM	Online help per device only 1 device-specific file includes: online help for this device ...\CoDeSys V*\Targets\ifm\Help\... (language)
ifm_CRnnnn_Vxyyzz.H86 ifm_CRnnnn_Vxyyzz.HEX	Operating system / runtime system (must be loaded into the controller / monitor when used for the first time) per device only 1 device-specific file ...\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn
ifm_Browser_CRnnnn.INI	CoDeSys browser commands (CoDeSys needs the file for starting the project) per device only 1 device-specific file includes: commands for the browser in CoDeSys ...\CoDeSys V*\Targets\ifm
ifm_Errors_CRnnnn.INI	CoDeSys error file (CoDeSys needs the file for starting the project) per device only 1 device-specific file includes: device-specific error messages from CoDeSys ...\CoDeSys V*\Targets\ifm
ifm_CRnnnn_Vxx.TRG	Target file per device only 1 device-specific file includes: hardware description for CoDeSys, e.g.: memory, file locations ...\CoDeSys V*\Targets\ifm

File name	Description and memory location ³⁾
ifm_*_Vxxyzz.LIB	General libraries per device several files are possible ...\CoDeSys V*\Targets\ifm\Library
ifm_CRnnnn_Vxxyzz.LIB	Device-specific library per device only 1 device-specific file includes: POU's of this device ...\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn
ifm_CRnnnn_*_Vxxyzz.LIB	Device-specific libraries per device several files are possible → following tables ...\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn

Legend:

*	any signs
CRnnnn	article number of the controller / monitor
V*	CoDeSys version
Vxx	version number of the ifm software
yy	release number of the ifm software
zz	patch number of the ifm software

¹⁾ valid for CRnn32 target version up to V01, all other devices up to V04

²⁾ valid for CRnn32 target version from V02 onwards, CR040n target version from V01 onwards, all other devices from V05 onwards

³⁾ memory location of the files:

System drive (C: / D:) \ program folder\ ifm electronic

NOTE

The software versions suitable for the selected target must always be used:

- operating system (CRnnnn_Vxxyzz.H86 / CRnnnn_Vxxyzz.HEX)
- PLC configuration (CRnnnn_Vxx.CFG)
- device library (ifm_CRnnnn_Vxxyzz.LIB)
- and the further files (→ chapter Overview of the files and libraries used (→ page [376](#)))

CRnnnn	device article number
Vxx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "CR0032") and the software version number "xx" (e.g. "02") must always have the same value! Otherwise the device goes to the STOP mode.

The values for "yy" (release number) and "zz" (patch number) do **not** have to match.

IMPORTANT: the following files must also be loaded:

- the internal libraries (created in IEC 1131) required for the project,
- the configuration files (*.CFG)
- and the target files (*.TRG).

14.3.2 What are the individual files and libraries used for?

Contents

Files for the operating system / runtime system	378
Target file	378
PLC configuration file	378
ifm device libraries.....	379
ifm CANopen libraries master / slave.....	379
CoDeSys CANopen libraries.....	379
Specific ifm libraries	380

2713

The following overview shows which files/libraries can and may be used with which unit. It may be possible that files/libraries which are not indicated in this list can only be used under certain conditions or the functionality has not yet been tested.

Files for the operating system / runtime system

2714

File name	Function	Available for:
ifm_CRnnnn_Vxyyzz.H86 ifm_CRnnnn_Vxyyzz.HEX	operating system / runtime system	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn
ifm_Browser_CRnnnn.INI	CoDeSys browser commands	all ecomatmobile controllers PDM: CR10nn
ifm_Errors_CRnnnn.INI	CoDeSys error file	all ecomatmobile controllers PDM: CR10nn

Target file

2715

File name	Function	Available for:
ifm_CRnnnn_Vxx.TRG	Target file	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn

PLC configuration file

2716

File name	Function	Available for:
ifm_CRnnnn_Vxyyzz.CFG	PLC configuration	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn

ifm device libraries

2717

File name	Function	Available for:
ifm_CRnnnn_Vxyyzz.LIB	device-specific library	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn
ifm_CR0200_MSTR_Vxyyzz.LIB	library without extended functions	ExtendedController: CR0200
ifm_CR0200_SMALL_Vxyyzz.LIB	library without extended functions, reduced functions	ExtendedController: CR0200

ifm CANopen libraries master / slave

2718

These libraries are based on the CoDeSys libraries (3S CANopen POU's) and make them available to the user in a simple way.

File name	Function	Available for:
ifm_CRnnnn_CANopenMaster_Vxyyzz.LIB	CANopen master emergency and status handler	all ecomatmobile controllers *) PDM: CR10nn *)
ifm_CRnnnn_CANopenSlave_Vxyyzz.LIB	CANopen slave emergency and status handler	all ecomatmobile controllers *) PDM: CR10nn *)
ifm_CANx_SDO_Vxyyzz.LIB	CANopen SDO read and SDO write	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
ifm_CANopen_small_Vxyyzz.LIB	CANopen POU's in the CAN stack	BasicController: CR0403 BasicDisplay: CR0451
ifm_CANopen_large_Vxyyzz.LIB	CANopen POU's in the CAN stack	PDM360NG: CR108n

*) but NOT for...

- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360NG: CR108n

CoDeSys CANopen libraries

2719

For the following devices these libraries are NOT useable:

- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360NG: CR108n

File name	Function	Available for:
3S_CanDrvOptTable.LIB ¹⁾ 3S_CanDrvOptTableEx.LIB ²⁾	CANopen driver	all ecomatmobile controllers PDM360smart: CR1070, CR1071
3S_CanDrv.LIB ³⁾		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
3S_CANopenDeviceOptTable.LIB ¹⁾ 3S_CANopenDeviceOptTableEx.LIB ²⁾	CANopen slave driver	all ecomatmobile controllers PDM360smart: CR1070, CR1071
3S_CANopenDevice.LIB ³⁾		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
3S_CANopenManagerOptTable.LIB ¹⁾ 3S_CANopenManagerOptTableEx.LIB ²⁾	CANopen network manager	all ecomatmobile controllers PDM360smart: CR1070, CR1071

File name	Function	Available for:
3S_CANOpenManager.LIB ³⁾		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
3S_CANOpenMasterOptTable.LIB ¹⁾ 3S_CANOpenMasterOptTableEx.LIB ²⁾	CANopen master	all ecomatmobile controllers PDM360smart: CR1070, CR1071
3S_CANOpenMaster.LIB ³⁾		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
3S_CANOpenNetVarOptTable.LIB ¹⁾ 3S_CANOpenNetVarOptTableEx.LIB ²⁾	Driver for network variables	all ecomatmobile controllers PDM360smart: CR1070, CR1071
3S_CANOpenNetVar.LIB ³⁾		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056

¹⁾ valid for CRnn32 target version up to V01, all other devices up to V04

²⁾ valid for CRnn32 target version from V02 onwards, all other devices from V05 onwards

³⁾ For the following devices: This library is without function used as placeholder:

- BasicController: CR040n
- BasicDisplay: CR0451

Specific ifm libraries

2720

File name	Function	Available for:
ifm_RawCAN_small_Vxxyzz.LIB	CANopen POUs in the CAN stack based on Layer 2	BasicController: CR0401, CR0402
ifm_RawCAN_large_Vxxyzz.LIB	CANopen POUs in the CAN stack based on Layer 2	BasicController: CR0403 BasicDisplay: CR0451 PDM360NG: CR108n
ifm_J1939_small_Vxxyzz.LIB	J1939 communication POUs in the CAN stack	BasicController: CR0401, CR0402
ifm_J1939_large_Vxxyzz.LIB	J1939 communication POUs in the CAN stack	BasicController: CR0403 BasicDisplay: CR0451 PDM360NG: CR108n
NetVarClib.LIB	additional driver for network variables	BasicController: CR040n BasicDisplay: CR0451 PDM360NG: CR108n
ifm_J1939_Vxxyzz.LIB	J1939 communication POUs	up to target V04: CabinetController: CR0303 ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7200, CR7505 SmartController: CR2500
ifm_J1939_x_Vxxyzz.LIB	J1939 communication POUs	from target V05: CabinetController: CR0303 ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 SmartController: CR2500 PDM360smart: CR1070, CR1071
ifm_CRnnnn_J1939_Vxxyzz.LIB	J1939 communication POUs	ClassicController: CR0032 ExtendedController: CR0232

File name	Function	Available for:
ifm_PDM_J1939_Vxxyzz.LIB	J1939 communication POU's	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
ifm_CANx_LAYER2_Vxxyzz.LIB	CAN POU's on the basis of layer 2: CAN transmit, CAN receive	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
ifm_CAN1E_Vxxyzz.LIB	changes the CAN bus from 11 bits to 29 bits	up to target V04: PDM360smart: CR1070, CR1071
ifm_CAN1_EXT_Vxxyzz.LIB	changes the CAN bus from 11 bits to 29 bits	from target V05: CabinetController: CR030n ClassicController: CR0020, CR0505 ExtendedController: CR0200 PCB controller: CS0015 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 SmartController: CR25nn PDM360smart: CR1070, CR1071
ifm_CAMERA_O2M_Vxxyzz.LIB	camera POU's	PDM360: CR1051
CR2013AnalogConverter.LIB	analogue value conversion for I/O module CR2013	all <i>ecomatmobile</i> controllers PDM: CR10nn
ifm_Hydraulic_16bitOS04_Vxxyzz.LIB	hydraulic POU's for R360 controllers	up to target V04: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7200, CR7505 SmartController: CR25nn
ifm_Hydraulic_16bitOS05_Vxxyzz.LIB	hydraulic POU's for R360 controllers	from target V05: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 SmartController: CR25nn
ifm_Hydraulic_32bit_Vxxyzz.LIB	hydraulic POU's for R360 controllers	ClassicController: CR0032 ExtendedController: CR0232
ifm_SafetyIO_Vxxyzz.LIB	safety POU's	SafetyController: CR7nnn
ifm_PDM_Util_Vxxyzz.LIB	help POU's PDM	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
ifm_PDMsmart_Util_Vxxyzz.LIB	help POU's PDM	PDM360smart: CR1070, CR1071
ifm_PDM_Input_Vxxyzz.LIB	alternative input POU's PDM	PDM: CR10nn
ifm_PDM_Init_Vxxyzz.LIB	initialisation POU's PDM360smart	PDM360smart: CR1070, CR1071
ifm_PDM_File_Vxxyzz.LIB	file POU's PDM360	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
Instrumente_x.LIB	predefined display instruments	PDM: CR10nn
Symbols_x.LIB	predefined symbols	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
Segment_x.LIB	predefined 7-segment displays	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056

Further libraries on request.

14.4 Troubleshooting

6757

Error	Cause	Remedy
Safe inputs are non-safe without monitoring.	VBB _R relay is not switched on. IMPORTANT: The monitoring of the safe inputs is only carried out if the VBB _R relay is switched on.	► Always switch on the VBB _R relay when using safe inputs.

15 Glossary of Terms

A

Address

This is the "name" of the bus participant. All participants need a unique address so that the signals can be exchanged without problem.

Application software

Software specific to the application, implemented by the machine manufacturer, generally containing logic sequences, limits and expressions that control the appropriate inputs, outputs, calculations and decisions

Necessary to meet the specific (→SRP/CS) requirements.

→ Programming language, safety-related

Architecture

Specific configuration of hardware and software elements in a system.

B

Baud

Baud, abbrev.: Bd = unit for the data transmission speed. Do not confuse baud with "bits per second" (bps, bits/s). Baud indicates the number of changes of state (steps, cycles) per second over a transmission length. But it is not defined how many bits per step are transmitted. The name baud can be traced back to the French inventor J. M. Baudot whose code was used for telex machines.

1 MBd = 1024 x 1024 Bd = 1 048 576 Bd

Bus

Serial data transmission of several participants on the same cable.

C

CAN

CAN = **C**ontroller **A**rea **N**etwork

CAN is a priority controlled fieldbus system for larger data volumes. It is available in different variants, e.g. "CANopen" or "CAN in Automation" (CiA).

CAN stack

CAN stack = stack of tasks for CAN data communication.

Category (CAT)

Classification of the safety-related parts of a control system in respect of their resistance to faults and their subsequent behaviour in the fault condition. This safety is achieved by the structural arrangement of the parts, fault detection and/or by their reliability.

(→ EN 954).

CCF

Common **C**ause **F**ailure

Failures of different items, resulting from a common event, where these failures are not consequences of each other.

CiA

CiA = CAN in Automation e.V.

User and manufacturer organisation in Germany / Erlangen. Definition and control body for CAN and CAN-based network protocols.

Homepage → <http://www.can-cia.org>

CiA DS 304

DS = **D**raft **S**tandard

CAN device profile CANopen safety for safety-related communication.

CiA DS 401

DS = **D**raft **S**tandard

CAN device profile for digital and analogue I/O modules

CiA DS 402

DS = **D**raft **S**tandard

CAN device profile for drives

CiA DS 403

DS = **D**raft **S**tandard

CAN device profile for HMI

CiA DS 404

DS = **D**raft **S**tandard

CAN device profile for measurement and control technology

CiA DS 405

DS = **D**raft **S**tandard

Specification for interface to programmable controllers (IEC 61131-3)

CiA DS 406

DS = **D**raft **S**tandard

CAN device profile for encoders

CiA DS 407

DS = **D**raft **S**tandard

CAN application profile for local public transport

Clamp 15

In vehicles clamp 15 is the plus cable switched by the ignition lock.

COB-ID

COB = **C**ommunication **O**bject

ID = **I**dentifier

Via the COB-ID the participants distinguish the different messages to be exchanged.

CoDeSys

CoDeSys® is a registered trademark of 3S – Smart Software Solutions GmbH, Germany.

"CoDeSys for Automation Alliance" associates companies of the automation industry whose hardware devices are all programmed with the widely used IEC 61131-3 development tool CoDeSys®.

Homepage → <http://www.3s-software.com>

CRC

CRC = **C**yclic **R**edundancy **C**heck

CRC is a method of information technology to determine a test value for data, to detect faults during the transmission or duplication of data.

Prior to the transmission of a block of data, a CRC value is calculated. After the end of the transaction the CRC value is calculated again at the target location. Then, these two test values are compared.

Cycle time

This is the time for a cycle. The PLC program performs one complete run.

Depending on event-controlled branchings in the program this can take longer or shorter.

D

DC

Direct **C**urrent

Glossary of Terms

DC

Diagnostic Coverage

Diagnostic coverage is the measure of the effectiveness of diagnostics as the ratio between the failure rate of detected dangerous failures and the failure rate of total dangerous failures:

Formula: $DC = \text{failure rate detected dangerous failures} / \text{total dangerous failures}$

Designation	Range
none	$DC < 60 \%$
low	$60 \% \leq DC < 90 \%$
medium	$90 \% \leq DC < 99 \%$
high	$99 \% \leq DC$

Table: Diagnostic coverage DC

An accuracy of 5 % is assumed for the limit values shown in the table.

Diagnostic coverage can be determined for the whole safety-related system or for only parts of the safety-related system.

Demand rate r_d

The demand rate r_d is the frequency of demands to a safety-related reaction of an SRP/CS per time unit.

Diagnosis

During the diagnosis, the "state of health" of the device is checked. It is to be found out if and what faults are given in the device.

Depending on the device, the inputs and outputs can also be monitored for their correct function.

- wire break,
- short circuit,
- value outside range.

For diagnosis, configuration and log data can be used, created during the "normal" operation of the device.

The correct start of the system components is monitored during the initialisation and start phase. Errors are recorded in the log file.

For further diagnosis, self-tests can also be carried out.

Diagnostic coverage

Diagnostic Coverage

Diagnostic coverage is the measure of the effectiveness of diagnostics as the ratio between the failure rate of detected dangerous failures and the failure rate of total dangerous failures:

Formula: $DC = \text{failure rate detected dangerous failures} / \text{total dangerous failures}$

Designation	Range
none	$DC < 60 \%$
low	$60 \% \leq DC < 90 \%$
medium	$90 \% \leq DC < 99 \%$
high	$99 \% \leq DC$

Table: Diagnostic coverage DC

An accuracy of 5 % is assumed for the limit values shown in the table.

Diagnostic coverage can be determined for the whole safety-related system or for only parts of the safety-related system.

Dither

Dither is a component of the PWM signals to control hydraulic valves. It has shown for electromagnetic drives of hydraulic valves that it is much easier for controlling the valves if the control signal (PWM pulse) is superimposed by a certain frequency of the PWM frequency. This dither frequency must be an integer part of the PWM frequency.

→ chapter What is the dither? (→ page [294](#))

Diversity

In technology diversity is a strategy to increase failure safety.

The systems are designed redundantly, however different implementations are used intentionally and not any individual systems of the same design. It is assumed that systems of the same performance, however of different implementation, are sensitive or insensitive to different interference and will therefore not fail simultaneously.

The actual implementation may vary according to the application and the requested safety:

- use of components of several manufacturers,
- use of different protocols to control devices,

Glossary of Terms

- use of totally different technologies, for example an electrical and a pneumatic controller,
- use of different measuring methods (current, voltage),
- two channels with reverse value progression:
channel A: 0...100 %
channel B: 100...0 %

DRAM

DRAM = **D**ynamic **R**andom **A**ccess **M**emory
Technology for an electronic memory module with random access (Random Access Memory, RAM). The memory element is a capacitor which is either charged or discharged. It becomes accessible via a switching transistor and is either read or overwritten with new contents. The memory contents are volatile: the stored information is lost in case of lacking operating voltage or too late restart.

DTC

DTC = **D**iagnostic **T**rouble **C**ode = error code
Faults and errors will be managed and reported via assigned numbers – the DTCs.

E

ECU

- (1) **E**lectronic **C**ontrol **U**nit = control unit or microcontroller
(2) **E**ngine **C**ontrol **U**nit = control device of a motor

EDS-file

EDS = **E**lectronic **D**ata **S**heet, e.g. for:

- File for the object directory in the master
- CANopen device descriptions

Via EDS devices and programs can exchange their specifications and consider them in a simplified way.

Embedded software

System software, basic program in the device, virtually the operating system.

The firmware establishes the connection between the hardware of the device and the user software. This software is provided by the manufacturer of the controller as a part of the system and cannot be changed by the user.

EMCY

abbreviation for emergency

EMV

EMC = **E**lectro **M**agnetic **C**ompatibilty

According to the EC directive (2004/108/EEC) concerning electromagnetic compatibility (in short EMC directive) requirements are made for electrical and electronic apparatus, equipment, systems or components to operate satisfactorily in the existing electromagnetic environment. The devices must not interfere with their environment and must not be adversely influenced by external electromagnetic interference.

Ethernet

Ethernet is a widely used, manufacturer-independent technology which enables data transmission in the network at a speed of 10 or 100 million bits per second (Mbps). Ethernet belongs to the family of so-called "optimum data transmission" on a non exclusive transmission medium. The concept was developed in 1972 and specified as IEEE 802.3 in 1985.

EUC

EUC = "Equipment Under Control"

EUC is equipment, machinery, apparatus or plant used for manufacturing, process, transportation, medical or other activities (→ IEC 61508-4, section 3.2.3). Therefore, the EUC is the set of all equipment, machinery, apparatus or plant that gives rise to hazards for which the safety-related system is required.

If any reasonably foreseeable action or inaction leads to hazards with an intolerable risk arising from the EUC, then safety functions are necessary to achieve or maintain a safe state for the EUC. These safety functions are performed by one or more safety-related systems.

F

Failure

Failure is the termination of the ability of an item to perform a required function.

After a failure, the item has a fault. Failure is an event, fault is a state.

The concept as defined does not apply to items consisting of software only.

Failure, dangerous

A dangerous failure has the potential to put the SRP/SC in a hazardous or fail-to-function state. Whether or not the potential is realized can depend on the channel architecture of the system; in redundant systems a dangerous hardware failure is less likely to lead to the overall dangerous or fail-to-function state.

Failure, systematic

A systematic failure is a failure related in a deterministic way (not coincidental) to a certain cause. The systematic failure can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors.

Corrective maintenance without modification of the system will usually not eliminate the failure cause.

Fault

A fault is the state of an item characterized by the inability to perform the requested function, excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources.

A fault is often the result of a failure of the item itself, but may exist without prior failure.

In ISO 13849-1 "fault" means "random fault".

Fault tolerance time

The max. time it may take between the occurrence of a fault and the establishment of the safe state in the application without having to assume a danger for people.

The max. cycle time of the application program

(in the worst case 100 ms, → Watchdog (→ page [109](#))) and the possible delay and response times due to switching elements have to be considered.

The resulting total time must be smaller than the fault tolerance time of the application.

FiFo

FiFo (**F**irst **I**n, **F**irst **O**ut) = operation of the stack: the data package which was written into a stack at first will be read at first too. For every identifier there is such one buffer (as a queue) available.

Firmware

System software, basic program in the device, virtually the operating system.

The firmware establishes the connection between the hardware of the device and the user software. This software is provided by the manufacturer of the controller as a part of the system and cannot be changed by the user.

First fault occurrence time

Time until the first failure of a safety element.

The operating system verifies the controller by means of the internal monitoring and test routines within a period of max. 30 s.

This "test cycle time" must be smaller than the statistical first fault occurrence time for the application.

Flash memory

Flash ROM (or flash EPROM or flash memory) combines the advantages of semiconductor memory and hard disks. Just like every other semiconductor memory the flash memory does not require moving parts. And the data is maintained after switch-off, similar to a hard disk.

The flash ROM evolved from the EEPROM (**E**lectrical **E**rasable and **P**rogrammable **R**ead-**O**nly **M**emory). The storage function of data in the flash ROM is identical to the EEPROM. Similar to a hard disk, the data are however written and deleted blockwise in data blocks up to 64, 128, 256, 1024, ... bytes at the same time.

Glossary of Terms

Advantages of flash memories

- The stored data are maintained even if there is no supply voltage.
- Due to the absence of moving parts, flash is noiseless and insensitive to shocks and magnetic fields.
- In comparison to hard disks, flash memories have a very short access time. Read and write speed are virtually constant across the entire memory area.
- The memory size that can be obtained has no upper limit, due to the simple and space-saving arrangement of the storage cells.

Disadvantages of flash memories

- A storage cell can tolerate a limited number of write and delete processes:
 - Multi-level cells: typ. 10 000 cycles
 - Single level cells: typ. 100 000 cycles
- Given that a write process writes memory blocks of between 16 and 128 Kbytes at the same time, memory cells which require no change are used as well.

FMEA

FMEA = **F**ailure **M**ode and **E**ffects **A**nalysis

Method of reliability engineering, to find potential weak points. Within the framework of quality or security management, the FMEA is used preventively to prevent faults and increase the technical reliability.

FRAM

FRAM, or also FeRAM, means **F**erroelectric **R**andom **A**ccess **M**emory. The storage operation and erasing operation is carried out by a polarisation change in a ferroelectric layer.

Advantages of FRAM as compared to conventional read-only memories:

- compatible with common EEPROMs,
- no power supply for data preservation,
- write time approx. 100 ns,
- 10^{10} read and write cycles guaranteed.

Functional safety

Part of the overall safety referred to the →EUC and the EUC control system which depends on

the correct functioning of the electric or electronic safety-related system, safety-related systems of other technologies and external devices for risk reduction.

H

Harm

Physical injury or damage to health.

Hazard

Hazard is the potential source of harm.

A distinction is made between the source of the hazard, e.g.:

- mechanical hazard,
 - electrical hazard,
- or the nature of the potential harm, e.g.:
- electric shock hazard,
 - cutting hazard,
 - toxic hazard.

The hazard envisaged in this definition is either permanently present during the intended use of the machine, e.g.:

- motion of hazardous moving elements,
 - electric arc during a welding phase,
 - unhealthy posture,
 - noise emission,
 - high temperature,
- or the hazard may appear unexpectedly, e.g.:
- explosion,
 - crushing hazard as a consequence of an unintended/unexpected start-up,
 - ejection as a consequence of a breakage,
 - fall as a consequence of acceleration/deceleration.

Heartbeat

The participants regularly send short signals. In this way the other participants can verify if a participant has failed. No master is necessary.

HMI

HMI = **H**uman **M**achine **I**nterface

Glossary of Terms

I

ID

ID = **I**dentifier

Name to differentiate the devices / participants connected to a system or the message packets transmitted between the participants.

IEC user cycle

IEC user cycle = PLC cycle in the CoDeSys application program.

Instructions

Superordinate word for one of the following terms:

installation instructions, data sheet, user information, operating instructions, device manual, installation information, online help, system manual, programming manual, etc.

Intended use

Use of a product in accordance with the information provided in the instructions for use.

IP address

IP = **I**nternet **P**rotocol

The IP address is a number which is necessary to clearly identify an internet participant. For the sake of clarity the number is written in 4 decimal values, e.g. 127.215.205.156.

ISO 11898

Standard: "Road vehicles – Controller area network"

Part 1: "Data link layer and physical signalling"

Part 2: "High-speed medium access unit"

Part 3: "Low-speed, fault-tolerant, medium dependent interface"

Part 4: "Time-triggered communication"

Part 5: "High-speed medium access unit with low-power mode"

ISO 11992

Standard: "Interchange of digital information on electrical connections between towing and towed vehicles"

Part 1: "Physical and data-link layers"

Part 2: "Application layer for brakes and running gear"

Part 3: "Application layer for equipment other than brakes and running gear"

Part 4: "Diagnostics"

ISO 16845

Standard: "Road vehicles – Controller area network (CAN) – Conformance test plan"

L

LED

LED = **L**ight **E**mitting **D**iode

Light emitting diode, also called luminescent diode, an electronic element of high coloured luminosity at small volume with negligible power loss.

Life, mean

Mean Time To Failure (MTTF) or: mean life.

The $MTTF_d$ is the expectation of the mean time to dangerous failure.

Designation	Range
low	$3 \text{ years} \leq MTTF_d < 10 \text{ years}$
medium	$10 \text{ years} \leq MTTF_d < 30 \text{ years}$
high	$30 \text{ years} \leq MTTF_d \leq 100 \text{ years}$

Table: Mean time of each channel to the dangerous failure $MTTF_d$

Link

A link is a cross-reference to another part in the document or to an external document.

LSB

Least **S**ignificant **B**it/**B**yte

Glossary of Terms

M

MAC-ID

MAC = **M**anufacturer's **A**ddress **C**ode
= manufacturer's serial number

→ID = **I**dentifier

Every network card has a MAC address, a clearly defined worldwide unique numerical code, more or less a kind of serial number. Such a MAC address is a sequence of 6 hexadecimal numbers, e.g. "00-0C-6E-D0-02-3F".

Master

Handles the complete organisation on the bus. The master decides on the bus access time and polls the →slaves cyclically.

Mission time T_M

Mission time T_M is the period of time covering the intended use of an SRP/CS.

Misuse

The use of a product in a way not intended by the designer.

The manufacturer of the product has to warn against readily predictable misuse in his user information.

MMI

HMI = **H**uman **M**achine **I**nterface

→ HMI (→ page [388](#))

Monitoring

Safety function which ensures that a protective measure is initiated:

- if the ability of a component or an element to perform its function is diminished.
- if the process conditions are changed in such a way that the resulting risk increases.

MSB

Most **S**ignificant **B**it/**B**yte

MTBF

Mean **T**ime **B**etween **F**ailures (MTBF)

Is the expected value of the operating time between two consecutive failures of items that are maintained.

NOTE: For items that are NOT maintained the mean life →MTTF is the expected value (mean value) of the distribution of lives.

MTTF

Mean **T**ime **T**o **F**ailure (MTTF) or: mean life.

MTTF_d

Mean **T**ime **T**o **F**ailure (MTTF) or: mean life.

The MTTF_d is the expectation of the mean time to dangerous failure.

Designation	Range
low	3 years ≤ MTTF _d < 10 years
medium	10 years ≤ MTTF _d < 30 years
high	30 years ≤ MTTF _d ≤ 100 years

Table: Mean time of each channel to the dangerous failure MTTF_d

Muting

Muting is the temporary automatic suspension of a safety function(s) by the SRP/CS.

Example: The safety light curtain is bridged, if the closing tools have reached a finger-proof distance to each other. The operator can now approach the machine without any danger and guide the workpiece.

N

NMT

NMT = **N**etwork **M**anagement = (here: in the CAN bus)

The NMT master controls the operating states of the NMT slaves.

Node

This means a participant in the network.

Node Guarding

Network participant

Configurable cyclic monitoring of each slave configured accordingly. The master verifies if the slaves reply in time. The slaves verify if the master regularly sends requests. In this way failed network participants can be quickly identified and reported.

O

Obj / object

Term for data / messages which can be exchanged in the CANopen network.

Object directory

Contains all CANopen communication parameters of a device as well as device-specific parameters and data.

OBV

Contains all CANopen communication parameters of a device as well as device-specific parameters and data.

Operating system

Basic program in the device, establishes the connection between the hardware of the device and the user software.

Operational

Operating state of a CANopen participant. In this mode SDOs, NMT commands and PDOs can be transferred.

P

PC card

→PCMCIA card

PCMCIA card

PCMCIA = Personal Computer Memory Card International Association, a standard for

expansion cards of mobile computers.

Since the introduction of the cardbus standard in 1995 PCMCIA cards have also been called PC card.

PDM

PDM = **P**rocess and **D**ialogue **M**odule

Device for communication of the operator with the machine / plant.

PDO

PDO = **P**rocess **D**ata **O**bject

The time-critical process data is transferred by means of the "process data objects" (PDOs). The PDOs can be freely exchanged between the individual nodes (PDO linking). In addition it is defined whether data exchange is to be event-controlled (asynchronous) or synchronised. Depending on the type of data to be transferred the correct selection of the type of transmission can lead to considerable relief for the CAN bus.

These services are not confirmed by the protocol, i.e. it is not checked whether the message reaches the receiver. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

PDU

PDU = **P**rotocol **D**ata **U**nit

The PDU is an item of the CAN protocol SAE J1939. PDU indicates a part of the destination or source address.

Performance Level

Performance Level

According to ISO 13849-1, a specification (PL a...e) of safety-related parts of control systems to perform a safety function under foreseeable conditions.

→ Chapter Performance Level PL (→ page [14](#))

PES

Programmable Electronic System

A programmable electronic system is a system

...

- for control, protection or monitoring,

- dependent for its operation on one or more

Glossary of Terms

programmable electronic devices,
- including all elements of the system such as input and output devices.

PGN

PGN = Parameter Group Number
PGN = PDU format (PF) + PDU source (PS)
The parameter group number is an item of the CAN protocol SAE J1939. PGN collects the address parts PF and PS.

Pictogram

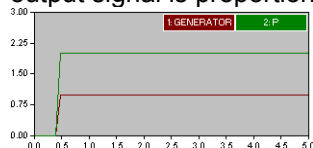
Pictograms are figurative symbols which convey information by a simplified graphic representation.

→ Chapter What do the symbols and formats stand for? (→ page [7](#))

PID controller

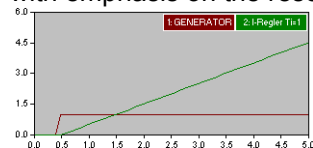
P = proportional part

The P controller exclusive consists of a proportional part of the amplification K_p . The output signal is proportional to the input signal.



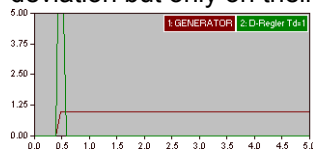
I = integral part

An I controller acts to the manipulating variable by phasing integration of the control deviation with emphasis on the reset time T_N .



D = differential part

The D controller doesn't react on the control deviation but only on their speed of change.



PL

Performance Level

According to ISO 13849-1, a specification (PL a...e) of safety-related parts of control systems to perform a safety function under

foreseeable conditions.

→ Chapter Performance Level PL (→ page [14](#))

PLC configuration

Part of the CoDeSys user interface.

- The programmer tells the programming system which hardware is to be programmed.
- > CoDeSys loads the corresponding libraries.
- > Reading and writing the peripheral states (inputs/outputs) is possible.

PLr

Using the "required performance level" PL_r the risk reduction for each safety function according to ISO 13849 is achieved.

For each selected safety function to be carried out by a SRP/CS, a PL_r shall be determined and documented. The determination of the PL_r is the result of the risk assessment and refers to the amount of the risk reduction.

Pre-Op

Pre-Op = Pre-operational mode

Operating status of a CANopen participant. After application of the supply voltage each participant automatically passes into this state. In the CANopen network only SDOs and NMT commands can be transferred in this mode but no process data.

prepared

Operating status of a CANopen participant. In this mode only NMT commands are transferred.

Process image

Process image is the status of the inputs and outputs the PLC operates with within one cycle.

- At the beginning of the cycle the PLC reads the conditions of all inputs into the process image. During the cycle the PLC cannot detect changes to the inputs.
- During the cycle the outputs are only

Glossary of Terms

changed virtually (in the process image).

- At the end of the cycle the PLC writes the virtual output states to the real outputs.

Programming language, safety-related

Only the following programming languages shall be used for safety-related applications:

- Limited variability language (LVL) that provides the capability of combining predefined, application-specific library functions.
In CoDeSys these are LD (ladder diagram) and FBD (function block diagram).
- Full variability language (FVL) provides the capability of implementing a wide variety of functions.
These include e.g. C, C++, Assembler. In CoDeSys it is ST (structured text).
- ▶ Structured text is recommended exclusively in separate, certified functions, usually in embedded software.
- ▶ In the "normal" application program only LD and FBD should be used. The following minimum requirements shall be met.

In general the following minimum requirements are made on the safety-related application software (SRASW):

- ▶ Modular and clear structure of the program. Consequence: simple testability.
- ▶ Functions are represented in a comprehensible manner:
 - for the operator on the screen (navigation)
 - readability of a subsequent print of the document.
- ▶ Use symbolic variables (no IEC addresses).
- ▶ Use meaningful variable names and comments.
- ▶ Use easy functions (no indirect addressing, no variable fields).
- ▶ Defensive programming.
- ▶ Easy extension or adaptation of the program possible.

Protective measure

Measure intended to achieve risk reduction, e.g.:

- fault-excluding design,

- safeguarding measures (guards),
- complementary protective measures (user information),
- personal protective equipment (helmet, protective goggles).

PWM

PWM = pulse width modulation

Via PWM a digital output (capability provided by the device) can provide an almost analogue voltage by means of regular fast pulses. The PWM output signal is a pulsed signal between GND and supply voltage.

Within a defined period (PWM frequency) the mark-to-space ratio is varied. Depending on the mark-to-space ratio, the connected load determines the corresponding RMS current.

→ chapter PWM signal processing

(→ page [273](#))

→ chapter What does a PWM output do?

(→ page [293](#))

R

Ratio

Measurements can also be performed ratiometrically. The input signal generates an output signal which is in a defined ratio to the input signal. This means that analogue input signals can be evaluated without additional reference voltage. A fluctuation of the supply voltage has no influence on this measured value.

→ Chapter Counter functions (→ page [258](#))

RAW-CAN

RAW-CAN means the pure CAN protocol which works without an additional communication protocol on the CAN bus (on ISO/OSI layer 2). The CAN protocol is international defined according to ISO 11898-1 and guarantees in ISO 16845 the interchangeability of CAN chips in addition.

Redundant

Redundancy is the presence of more than the necessary means so that a function unit performs a requested function or that data can represent information.

Several kinds of redundancy are distinguished:

Glossary of Terms

- Functional redundancy aims at designing safety-related systems in multiple ways in parallel so that in the event of a failure of one component the others ensure the task.
- In addition it is tried to separate redundant systems from each other with regard to space. Thus the risk that they are affected by a common interference is minimised.
- Finally, components from different manufacturers are sometimes used to avoid that a systematic fault causes all redundant systems to fail (diverse redundancy).

The software of redundant systems should differ in the following aspects:

- specification (different teams),
- specification language,
- programming (different teams),
- programming language,
- compiler.

Remanent

Remanent data is protected against data loss in case of power failure.

The operating system for example automatically copies the remanent data to a flash memory as soon as the voltage supply falls below a critical value. If the voltage supply is available again, the operating system loads the remanent data back to the RAM memory.

The data in the RAM memory of a controller, however, is volatile and normally lost in case of power failure.

Reset, manual

4021

The manual reset is an internal function within the SRP/CS used to restore manually one or more safety functions before re-starting a machine.

Residual risk

Risk remaining after protective measures have been taken. The residual risk has to be clearly warned against in operating instructions and on the machine.

Risk

Combination of the probability of occurrence of harm and the severity of that harm.

Risk analysis

Combination of ...

- the specification of the limits of the machine (intended use, time limits),
- hazard identification (intervention of people, operating status of the machine, foreseeable misuse) and
- the risk estimation (degree of injury, extent of damage, frequency and duration of the risk, probability of occurrence, possibility of avoiding the hazard or limiting the harm).

Risk assessment

Overall process comprising risk analysis and risk evaluation.

According to Machinery Directive 2006/42/EU the following applies: "The manufacturer of machinery or his authorised representative must ensure that a risk assessment is carried out in order to determine the health and safety requirements which apply to the machinery. The machinery must then be designed and constructed taking into account the results of the risk assessment." (→ Annex 1, General principles)

Risk evaluation

Judgement, on the basis of the risk analysis, of whether risk reduction objectives have been achieved.

ro

RO = read only for reading only

Unidirectional data transmission: Data can only be read and not changed.

RTC

RTC = Real Time Clock

Provides (batter-backed) the current date and time. Frequent use for the storage of error message protocols.

Glossary of Terms

rw

RW = read/ write

Bidirectional data transmission: Data can be read and also changed.

S

SAE J1939

The network protocol SAE J1939 describes the communication on a CAN bus in commercial vehicles for transmission of diagnosis data (e.g. motor speed, temperature) and control information.

→ CiA DS 402

Standard: "Recommended Practice for a Serial Control and Communications Vehicle Network"

Part 2: "Agricultural and Forestry Off-Road Machinery Control and Communication Network"

Part 3: "On Board Diagnostics Implementation Guide"

Part 5: "Marine Stern Drive and Inboard Spark-Ignition Engine On-Board Diagnostics Implementation Guide"

Part 11: "Physical Layer – 250 kBits/s, Shielded Twisted Pair"

Part 13: "Off-Board Diagnostic Connector"

Part 15: "Reduced Physical Layer, 250 kBits/s, Un-Shielded Twisted Pair (UTP)"

Part 21: "Data Link Layer"

Part 31: "Network Layer"

Part 71: "Vehicle Application Layer"

Part 73: "Application Layer – Diagnostics"

Part 81: "Network Management Protocol"

Safety function

Function of the machine whose failure can result in an immediate increase of the risk(s). The designer of such a machine therefore has to:

- safely prevent a failure of the safety function,
- reliably detect a failure of the safety function in time,
- bring the machine into a safe state in time in the event of a failure of the safety function.

Safety-standard types

The safety standards in the field of machines are structured as below:

Type-A standards (basic safety standards) giving basic concepts, principles for design, and general aspects that can be applied to all machinery. Examples: basic terminology, methodology (ISO 12100-1), technical principles (ISO 12100-2), risk assessment (ISO 14121), ...

Type-B standards (generic safety standards) dealing with one safety aspect or one type of safeguard that can be used across a wide range of machinery.

- Type-B1 standards on particular safety aspects. Examples: safety distances (EN 294), hand/arm speeds (EN 999), safety-related parts of control systems (ISO 13849), temperatures, noise, ...
- Type-B2 standards on safeguards. Examples: emergency stop circuits ((ISO 13850), two-hand controls, interlocking devices or electro-sensitive protective equipment (ISO 61496), ...

Type-C standards (machine safety standards) dealing with detailed safety requirements for a particular machine or group of machines.

SCT

In CANopen safety the **Safeguard Cycle Time** (SCT) monitors the correct function of the periodic transmission (data refresh) of the SRDOs. The data must have been repeated within the set time to be valid. Otherwise the receiving controller signals a fault and passes into the safe state (= outputs switched off).

SD card

An SD memory card (short for **Secure Digital Memory Card**) is a digital storage medium that operates to the principle of flash storage.

SDO

SDO = **S**ervice **D**ata **O**bject.

SDO is a specification for a manufacturer-dependent data structure for standardised data access. "Clients" ask for the requested data from "servers". The SDOs always consist of 8 bytes. Longer data

Glossary of Terms

packages are distributed to several messages.

Examples:

- Automatic configuration of all slaves via SDOs at the system start,
- reading error messages from the object directory.

Every SDO is monitored for a response and repeated if the slave does not respond within the monitoring time.

Self-test

Test program that actively tests components or devices. The program is started by the user and takes a certain time. The result is a test protocol (log file) which shows what was tested and if the result is positive or negative.

SIL

According to IEC 62061 the safety-integrity level SIL is a classification (SIL CL 1...4) of the safety integrity of the safety functions. It is used for the evaluation of electrical/electronic/programmable electronic (E/E/EP) systems with regard to the reliability of safety functions. The safety-related design principles that have to be adhered to so that the risk of a malfunction can be minimised result from the required level.

Slave

Passive participant on the bus, only replies on request of the →master. Slaves have a clearly defined and unique →address in the bus.

SRDO

Safe data is exchanged via SRDOs (**S**afety-**R**elated **D**ata **O**bjects). An SRDO always consists of two CAN messages with different identifiers:

- message 1 contains the original user data,
- message 2 contains the same data which are inverted bit by bit.

SRP/CS

Safety-**R**elated **P**art of a **C**ontrol **S**ystem

Part of a control system that responds to safety-related input signals and generates

safety-related output signals. The combined safety-related parts of a control system start at the point where the safety-related input signals are initiated (including, for example, the actuating cam and the roller of the position switch) and end at the output of the power control elements (including, for example, the main contacts of a contactor).

SRVT

The SRVT (**S**afety-**R**elated **O**bject **V**alidation **T**ime) ensures with CANopen safety that the time between the SRDO-message pairs is adhered to.

Only if the redundant, inverted message has been transmitted after the original message within the SRVT set are the transmitted data valid. Otherwise the receiving controller signals a fault and will pass into the safe state (= outputs switched off).

State, safe

The state of a machine is said to be safe when there is no more hazard formed by it. This is usually the case if all possible dangerous movements are switched off and cannot start again unexpectedly.

Symbols

Pictograms are figurative symbols which convey information by a simplified graphic representation.

→ Chapter What do the symbols and formats stand for? (→ page [7](#))

System variable

Variable to which access can be made via IEC address or symbol name from the PLC.

T

Target

The target indicates the target system where the PLC program is to run. The target contains the files (drivers and if available specific help files) required for programming and parameter setting.

Glossary of Terms

TCP

The **T**ransmission **C**ontrol **P**rotocol is part of the TCP/IP protocol family. Each TCP/IP data connection has a transmitter and a receiver. This principle is a connection-oriented data transmission. In the TCP/IP protocol family the TCP as the connection-oriented protocol assumes the task of data protection, data flow control and takes measures in the event of data loss.

(compare: →UDP)

Template

A template can be filled with content.
Here: A structure of pre-configured software elements as basis for an application program.

Test rate r_t

The test rate r_t is the frequency of the automatic tests to detect errors in an SRP/CS in time.

U

UDP

UDP (**U**ser **D**atagram **P**rotocol) is a minimal connectionless network protocol which belongs to the transport layer of the internet protocol family. The task of UDP is to ensure that data which is transmitted via the internet is passed to the right application.

At present network variables based on CAN and UDP are implemented. The values of the variables are automatically exchanged on the basis of broadcast messages. In UDP they are implemented as broadcast messages, in CAN as PDOs. These services are not confirmed by the protocol, i.e. it is not checked whether the message is received. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

Uptime, mean

Mean Time Between Failures (MTBF)

Is the expected value of the operating time between two consecutive failures of items that are maintained.

NOTE: For items that are NOT maintained the

mean life →MTTF is the expected value (mean value) of the distribution of lives.

Use, intended

Use of a product in accordance with the information provided in the instructions for use.

W

Watchdog

In general the term watchdog is used for a component of a system which watches the function of other components. If a possible malfunction is detected, this is either signalled or suitable program branchings are activated. The signal or branchings serve as a trigger for other co-operating system components to solve the problem.

wo

WO = write only

Unidirectional data transmission: Data can only be changed and not read.

16 Index

A distinction is made between the following errors:	245
About the ifm templates	66, 68
About this manual	7
Above-average stress	108
Access to the CAN device at runtime	197
Access to the OD entries by the application program	197
Access to the status of the CANopen master	187
Access to the structures at runtime of the application	212, 213, 215
Achievable safety class	45
Activating the PLC configuration	63
Adapting analogue values	255
Add and configure CANopen slaves	177, 196
Address	383
Address assignment and I/O operating modes ...	368
Address assignment inputs / outputs	372
Addresses / I/O variables	368
'Addresses' in CANopen	172
After application of the supply voltage	52
Analogue inputs	88
Annex	61, 368
Application program	56
Application software	383
Applications	36, 87, 258
Architecture	383
Archiving of documentation	12
Automatic configuration of slaves	180
Automatic data backup	343
Availability of PWM	272
Available memory (CR7nnn)	109
Baud	383
Boot up of the CANopen master	183
Boot up of the CANopen slaves	184
Bus	383
Bus cable length	123
Calculation examples RELOAD value	276
Calculation of the RELOAD value	275, 280

CAN	383
CAN bus level	122
CAN bus level according to ISO 11992-1	122
CAN device	170, 190, 223
CAN device configuration	191
CAN error	116
CAN errors	242
CAN errors and error handling	121, 215, 242
CAN for the drive engineering	154
CAN interfaces	120
CAN network variables	120, 170, 198
CAN stack	383
CAN units acc. to SAE J1939	120, 134, 141, 154
CAN_SAFETY_RECEIVE (FB)	33, 145, 147, 150, 234, 235, 236, 239
CAN_SAFETY_TRANSMIT (FB)	33, 143, 234, 235, 236, 237, 240
CAN1_BAUDRATE (FB)	129, 130, 197
CAN1_DOWNLOADID (FB)	132
CAN1_EXT (FB)	134, 136, 138, 197
CAN1_EXT_ERRORHANDLER (FB)	140
CAN1_EXT_RECEIVE (FB)	138
CAN1_EXT_TRANSMIT (FB)	136
CAN2 (FB)	129, 141, 143, 145, 152, 159
CAN-ID	125, 126, 128, 155
CANopen for safety-related communication	232
CANopen master	170, 173, 213
CANopen Safety in safety-related applications ..	29, 33, 103, 231
CANopen support by CoDeSys	170, 171
CANopen terms and implementation	172
CANx_ERRORHANDLER (FB)	152, 244
CANx_EXT_RECEIVE_ALL (FB)	150
CANx_MASTER_EMCI_HANDLER (FB)	69, 205, 207, 246
CANx_MASTER_SEND_EMERGENCY (FB)	205, 207, 246
CANx_MASTER_STATUS (FB)	69, 180, 182, 183, 184, 185, 186, 187, 189, 210, 213
CANx_RECEIVE (FB)	125, 128, 129, 145, 147
CANx_RECEIVE_RANGE (FB)	147

Index

CANx_SDO_READ (FB).....	174, 227, 245
CANx_SDO_WRITE (FB).....	174, 229
CANx_SLAVE_EMCY_HANDLER (FB)	69, 190, 197, 218, 220, 246
CANx_SLAVE_NODEID (FB).....	197, 217
CANx_SLAVE_SEND_EMERGENCY (FB).....	190, 197, 218, 220, 246
CANx_SLAVE_STATUS (FB).....	69, 197, 223
CANx_TRANSMIT (FB)	125, 128, 129, 143
Categories to ISO 13849	16, 19
Category (CAT)	383
CCF	383
Certification and distribution of the safety-related software.....	112
Change the PDO properties at runtime	197
Changes of the manual (S16).....	8
Changing the safety-relevant software after certification	113
Changing the standard mapping by the master configuration	196
CHECK_DATA (FB)	32, 359
CiA.....	383
CiA DS 304.....	383
CiA DS 401.....	384
CiA DS 402.....	384
CiA DS 403.....	384
CiA DS 404.....	384
CiA DS 405.....	384
CiA DS 406.....	384
CiA DS 407.....	384
Clamp 15.....	384
COB-ID.....	384
CoDeSys	384
CoDeSys CANopen libraries	379
Communication via interfaces.....	327
Communication via the internal SSC interface	334
Configuration of all correctly detected devices..	180
Configuration of CAN network variables	199
Configurations.....	15, 61
Configure inputs.....	79
Configure outputs	92
Connect e-stop	82
Connect terminal VBBO (5) to battery (not switched).....	49
Connect terminal VBBS (23) to the ignition switch	49
Control hydraulic valves with current-controlled outputs.....	293
CONTROL_OCC (FB).....	297, 298
Controlled system with delay.....	314
Controlled system without inherent regulation ..	314
Controller functions	313
Counter functions for frequency and period measurement	258, 373, 393
CPU frequency.....	107
CRC	384
Create a CANopen project	174
Creating application program.....	28, 111
Current control with PWM	285, 373
Current measurement with PWM channels.....	286
Cycle time.....	384
Cyclical transmission of the SYNC message.....	181
Damping of overshoot.....	315
Data access and data check	104, 351
Data reception	128
Data transmission.....	128
DC.....	384, 385
DEBUG mode.....	60
DELAY (FB)	317
Demand rate rd.....	385
Demo program for controller	75
Demo programs for PDM and BasicDisplay	76
Description of the CAN standard program units.....	129
Diagnosis	385
Diagnostic coverage.....	385
Differentiation from other CANopen libraries...	173
Digital and PWM outputs	92
Digital inputs.....	79
Digital outputs for safety functions.....	93
Digital safety inputs	80

Index

Dither	385	Fail-safe sensors and safety signal transmitters ...	30
Dither frequency and amplitude.....	295	Failure	387
Diversity.....	385	Failure, dangerous.....	387
DRAM	386	Failure, systematic	387
DTC	386	Fast inputs.....	86
ECU	386	Fast safety inputs	86
EDS-file	386	FAST_COUNT (FB)	86, 270
Embedded software.....	386	Fatal error.....	57
EMCY	386	Fatal errors	116
EMCY error code.....	246	Fault	387
EMV.....	386	Fault tolerance time	387
Error counter	243	FB, FUN, PRG in CoDeSys.....	110
Error message.....	242	Feedback in case of externally supplied outputs.....	54
Error messages	114, 374	FiFo.....	387
Ethernet.....	386	Files for the operating system / runtime system.....	378
EUC	386	Firmware.....	387
Example		First fault occurrence time	387
CANx_MASTER_SEND_EMERGENCY	209	Flash memory	387
CANx_MASTER_STATUS	212, 214	FLASHREAD (FB)	348
CANx_SLAVE_SEND_EMERGENCY.....	222	FLASHWRITE (FB).....	346
CHECK_DATA	360	FMEA	388
detailed message documentation	156	Folder structure in general	68
Initialisation of CANx_RECEIVE_RANGE		FRAM	388
in 4 cycles	147, 149	FRAMREAD (FB).....	350
NORM_HYDRAULIC	312	FRAMWRITE (FB).....	349
SAFE_ANALOG_OK.....	34, 35	FREQUENCY (FB).....	36, 86, 87, 258, 259, 261
SAFE_FREQUENCY_OK.....	36, 37, 87	Function configuration of the inputs and outputs.....	30, 78, 94
SAFE_INPUTS_OK	38, 40, 82	Functional safety	388
SAFETY_SWITCH.....	41, 43	Functionality of the CAN device library.....	190
short message documentation.....	157	Functions and features	44
Example 1	257	Functions for CANopen Safety.....	236
Example 2	257	Functions for controllers	316
Example Dither	296	Functions of the library ifm_hydraulic_16bitOS05	297
Example fail-safe sensor	85	Further ifm libraries for CANopen	226
Example of an object directory	191	General.....	313
Example process for response to a system error	118	General about CAN	119
Exchange of CAN data	121, 125	General information	11, 198
ExtendedSafetyController CR7200/CR7201	29		

Index

General information about CANopen with CoDeSys	171	INPUT_VOLTAGE (FB)	76, 253
General notes and explanations on CANopen Safety	231	Inputs for fail-safe inductive sensors	30, 83
General overview	376	Instructions	389
GET_IDENTITY (FB).....	355	Intended use	389
Global failsafe command GFC.....	234	IP address.....	389
GLR (FB).....	325	ISO 11898.....	389
Hardware structure.....	45, 48, 49	ISO 11992.....	389
Harm	388	ISO 16845.....	389
Hazard.....	388	J1939_x (FB)	159
Heartbeat.....	388	J1939_x_GLOBAL_REQUEST (FB)	168
Heartbeat from the master to the slaves	181	J1939_x_RECEIVE (FB).....	160
Hints.....	126	J1939_x_RESPONSE (FB)	164
Hints to wiring diagrams.....	101	J1939_x_SPECIFIC_REQUEST (FB)	166
HMI.....	388, 390	J1939_x_TRANSMIT (FB)	162
How is this manual structured?	8	JOYSTICK_0 (FB)	297, 301
Hydraulic control in PWMi.....	292	JOYSTICK_1 (FB)	297, 304
ID	389	JOYSTICK_2 (FB)	297, 308
Identifier.....	245	Latching	49, 117
Identifier acc. to SAE J1939	155	LED.....	389
IEC user cycle	389	Library for the CANopen master	204
If runtime system / application is running.....	52	Library for the CANopen slave.....	216
If the TEST pin is not active	53	Life, mean	389
ifm CANopen libraries master / slave	379	Limitations and programming notes	107, 268
ifm CANopen library	120, 170	Limits of the device	107
ifm demo programs	75, 278	Limits of the SafetyController	108
ifm device libraries.....	379	Link.....	389
Important!	9	Load the operating system	59
In-/output functions.....	250	LSB	389
INC_ENCODER (FB)	267	MAC-ID.....	390
Information concerning the device.....	44	Managing the data.....	338
Information concerning the software	46	Manual data storage	344
INIT state (Reset).....	57	Manufacturer specific information.....	248
Initialisation of the network with RESET_ALL_NODES	187	Master	390
Input group I0 (ANALOG0...7 or %IX0.0...%IX0.7)	90	Master at runtime	180, 212
Input group I1...I4 (%IX0.8...%IX2.7).....	91	Maximum program cycle time.....	56
INPUT_ANALOG (FB)	34, 90, 251, 253, 254	Measuring methods for fast inputs.....	87
INPUT_CURRENT (FB).....	254	MEMCPY (FB).....	345
		Mission time TM	390
		Misuse.....	390

Index

MMI.....	390	Operating system	56, 391
Monitoring	390	Operating system and software versions.....	32
Monitoring and securing mechanisms.....	52	Operational	391
Monitoring concept.....	48	Optimising the PLC cycle.....	361
Monitoring of the supply voltage VBBs	51	Output group Q1Q2 (%QX0.0...%QX0.7).....	96
MSB	390	Output group Q3 (%QX0.8...%QX0.15).....	98
MTBF.....	390	Output group Q4 (%QX1.0...%QX1.7).....	99
MTTF.....	390	OUTPUT_CURRENT (FB).....	95, 96, 280, 282, 284, 288, 291, 297, 298
MTTFd.....	390	OUTPUT_CURRENT_CONTROL (FB).....	286, 287, 289, 297, 298
Muting.....	390	Overload protection	285
Network states.....	183	Overview CANopen EMCY codes (C16).....	249
Network structure.....	121	Overview CANopen error codes.....	208, 221, 246, 247
NMT.....	390	Overview of the files and libraries used.....	32, 46, 59, 62, 376, 377
No operating system.....	58	Parameters of internal structures.....	213
Node.....	390	Participant, bus off.....	243, 244
Node Guarding.....	391	Participant, error active	243
Nodeguarding / heartbeat error	185	Participant, error passive.....	243
Nodeguarding with lifetime monitoring.....	181	Particularities for network variables	199, 203
NORM (FB).....	76, 256	PC card.....	391
NORM_HYDRAULIC (FB).....	297, 310	PCMCIA card	391
Note the cycle time!	110	PDM.....	391
Notes on devices with monitoring relay.....	117	PDO	391
Notes on safety-related applications	11, 231	PDU	391
Obj / object.....	391	Performance Level.....	391
Object 0x1001 (error register).....	248	Performance level PL.....	14, 391, 392
Object 0x1003 (error field)	246	PERIOD (FB)	36, 86, 87, 258, 259, 261
Object directory	391	PERIOD_RATIO (FB)	86, 263
OBV	391	PES	391
OCC_TASK (FB)	286, 288, 289, 297	PGN	392
One-time mechanisms.....	53	PHASE (FB)	86, 265
Operating mode master/master	104	Physical connection of CAN.....	121
Operating mode master/slave.....	105	Pictogram.....	392
Operating modes	60	PID controller	392
Operating modes of the ExtendedSafetyController	29, 78, 103	PID1 (FB)	320
Operating principle	83	PID2 (FB)	322
Operating principle of the delayed switch-off.....	49	PL.....	392
Operating principle of the monitoring concept	50	Plausibility check via the process	82
Operating states.....	57		
Operating states and operating system.....	57		

Index

PLC configuration.....	47, 392	Reset of all configured slaves on the bus at the system start	180
PLC configuration file	378	Reset, manual.....	394
PLr	392	Residual risk	394
Polling of the slave device type	180	Response to the system error	117
Possible operating modes inputs / outputs	78, 370, 373	Risk	394
Predefined identifiers for CANopen-Safety	235	Risk analysis	394
Pre-Op	392	Risk assessment	12, 13, 394
prepared	392	Risk assessment of a machine	11
Process image.....	392	Risk evaluation	394
Processing analogue input values.....	250	Risk graph to ISO 13849.....	18, 80, 86, 89, 93
Processing interrupts	361	ro.....	394
Processing of the SRDO in the SafetyController	234	RTC.....	394
Programming language, safety-related.....	393	Rules for documenting safety-related software....	28
Programming notes for CoDeSys projects	110	Rules for selecting the tools, libraries, languages	23
Programs and functions in the folders of the templates	69	Rules for subsequent program modifications.....	28
Protective measure	393	Rules for testing safety-related software.....	27
PT1 (FB)	315, 319	Rules on SRASW and non safety-related software in one component.....	25
PWM.....	393	Rules on the program structure	24
PWM (FB)	274, 275, 279, 281, 283, 286, 288, 289	Rules on the safety-related function blocks	25
PWM / PWM1000	274	Rules on the software implementation / coding ...	25
PWM channels 0...3	275	Rules on the specification of the SRASW	23
PWM channels 4...7 / 8...11	277	Rules on the use of data types	27
PWM dither.....	278, 280	Rules on the use of variables	25, 26
PWM frequency	274	Rules on the verification of safety-related software	28
PWM functions	272	RUN state.....	58
PWM functions and their parameters.....	274	rw	395
PWM signal processing	273, 278, 297, 373, 393	SAE J1939	395
PWM100 (FB)	76, 281	Safe state.....	29
PWM1000 (FB)	274, 275, 279, 281, 283	SAFE_ANALOG_OK (FB).....	33, 34, 35, 89, 251
Ramp function.....	278	SAFE_FREQUENCY_OK (FB).....	33, 36, 37, 86, 87, 259, 261
Ratio.....	393	SAFE_INPUTS_OK (FB)	33, 38, 80, 82
RAW-CAN	393	Safeguard cycle time SCT	233
Reading / writing the system time.....	340	Safety aspects.....	87
Reception of emergency messages.....	181	Safety concept.....	29, 45, 55
Recommended settings	321	Safety considerations	24, 31
Redundant	393	Safety for bus systems	20
Remanent	394		

Index

Safety function	395	SET_PASSWORD (FB)	357
Safety functions	33, 236	Setting control	315
Safety instructions	9	Setting of the node numbers and the baud rate of a CAN device	197
Safety outputs %QX0.04...0.07 (%QX32.04...32.07)	96	Setting rule for a controller	315
Safety outputs %QX1.00...1.07 (%QX33.00...33.07)	99	Settings in the global variable lists	200
Safety outputs for PWM and PWMi	95	Settings in the target settings	199
SAFETY_SWITCH (FB)	33, 41, 83, 85	Setup the target	47, 62, 170, 174
SafetyController	29	Signalling of device errors	246
Safety-related applications software (SRASW)	22, 31	SIL	396
Safety-related data objects SRDOs	233	Slave	396
Safety-related inputs and outputs	29	Slave information	214
Safety-related object validation time SRVT	233	Slight errors	114
Safety-related processing of the memory areas	53	SOFTRESET (FB)	339
Safety-related programming with CoDeSys to ISO 13849	21	Software reset	338
Safety-standard types	13, 395	Software structure	55
Save	112	Specific ifm libraries	380
Saving, reading and converting data in the memory	343	SRDO	396
SCT	395	SRP/CS	13, 396
SD card	395	SRVT	396
SDO	395	SSC_RECEIVE (FB)	104, 335
Self-regulating process	313	SSC_TRANSMIT (FB)	104, 335, 337
Self-test	396	Standard ISO 13849	13, 29, 45
SERIAL_MODE	60	Standardise the output signals of a joystick	292
SERIAL_PENDING (FB)	333	Start of all correctly configured slaves	181
SERIAL_RX (FB)	331, 333	Start the network	182, 183
SERIAL_SETUP (FB)	328	Starting the network with GLOBAL_START	186, 211, 224
SERIAL_TX (FB)	330	Starting the network with START_ALL_NODES	186, 211
Serious errors	114, 115	Start-up of the network without [Automatic startup]	186
Set up programming system	61	State, safe	396
Set up programming system manually	61	Status LED	58
Set up programming system via templates	65, 104, 106, 129, 174, 370	STOP state	57
SET_DEBUG (FB)	30, 60, 352	Structure Emergency_Message	215
SET_IDENTITY (FB)	353, 355	Structure node status	214
SET_INTERRUPT_I (FB)	365	Structure of an EMCY message	245
SET_INTERRUPT_XMS (FB)	76, 362	Structure of an error message	245
		Structure of the visualisations in the templates	71
		Summary CAN / CANopen	127

Index

Supplement project with further functions.....	67, 72	Use of the serial interface	60, 327
Symbols	396	Use, intended	397
System configuration	120	Using CAN	119
System description	44	Using ifm downloader.....	112
System flags.....	114, 374	Watchdog.....	397
System test	55	Watchdog behaviour	24, 31, 51, 109, 387
System variable.....	396	What are the individual files and libraries used for?	
Tab [Base settings].....	191	378
Tab [CAN parameters].....	175, 178, 213	What do the symbols and formats mean?	
Tab [CAN settings]	193	7, 392, 396
Tab [Default PDO mapping]	194	What does a PWM output do?	293, 393
Tab [Receive PDO-Mapping] and [Send		What does machine safety mean?	11
PDO-Mapping]	179	What is the dither?	294, 385
Tab [Service Data Objects]	179, 227, 229	What previous knowledge is required?	10
Target.....	396	When is a dither useful?.....	294
Target file.....	378	Wire cross-sections	124
TCP	397	wo	397
Technical about CANopen.....	171		
Technology of the safety-related control functions			
for PL or SIL.....	19		
Template	397		
Test basis for certification.....	44		
Test input	30		
TEST mode	30, 57, 58, 60, 352		
Test rate rt	397		
The object directory of the CANopen master			
.....	188, 198		
The purpose of this library? – An introduction ..	292		
TIMER_READ (FB).....	341		
TIMER_READ_US (FB).....	342		
Topology	119		
Transmit emergency messages via the application			
program.....	197		
Troubleshooting	382		
UDP	397		
Units for CANopen	204		
Units for SAE J1939	158		
Uptime, mean.....	397		
Use as digital inputs	87, 258		
Use in applications up to CAT 3 / PL d	31		
Use of analogue inputs for digital signals	89		