Original Programming Manual
ExtendedController

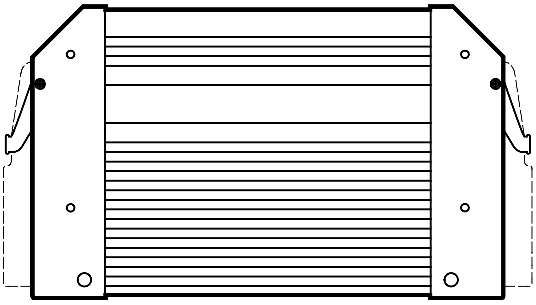
ecomat100

CR0200

Runtime system v06

CODESYS® v2.3

English



7391047_01_UK 2015-07-20

Contents

1 About this manual

Inhalt

202

1.1 Copyright

6088

© 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:

- AS-i is the property of the AS-International Association, (→ www.as-interface.net)
- CAN is the property of the CiA (CAN in Automation e.V.), Germany (→ www.can-cia.org)
- CODESYS™ is the property of the 3S – Smart Software Solutions GmbH, Germany (→ www.codesys.com)
- DeviceNet™ is the property of the ODVA™ (Open DeviceNet Vendor Association), USA (→ www.odva.org)
- EtherNet/IP® is the property of the →ODVA™
- IO-Link® (→ www.io-link.com) is the property of the →PROFIBUS Nutzerorganisation e.V., Germany
- Microsoft® is the property of the Microsoft Corporation, USA (→ www.microsoft.com)
- PROFIBUS® is the property of the PROFIBUS Nutzerorganisation e.V., Germany (→ www.profibus.com)
- PROFINET® is the property of the →PROFIBUS Nutzerorganisation e.V., Germany
- Windows® is the property of the →Microsoft Corporation, USA

1.2 Overview: documentation modules for ecomatmobile devices

17405

The documentation for **ecomat*mobile*** devices consists of the following modules:

1.	Data sheet
Contents	Technical data in a table
Source	→ www.ifm.com > select your country > [Data sheet search] > CR0200 > [Technical data in PDF format]

2.	Installation instructions / operating instructions
Contents	Instructions for installation, electrical installation, (commissioning*), technical data
Source	The instructions are supplied with the device They are also found on ifm's homepage: → www.ifm.com > select your country > [Data sheet search] > CR0200 > [Operating instructions]

3.	Programming manual + online help
Contents	Description of the configuration and the functions of the device software
Source	→ www.ifm.com > select your country > [Data sheet search] > CR0200 > [Operating instructions]

4.	System manual "Know-how ecomatmobile"
Contents	Know-how about the following topics: • Overview Templates and demo programs • CAN, CANopen • Control outputs • User flash memory • Visualisations • Overview of the files and libraries used
Source	→ www.ifm.com > select your country > [Data sheet search] > CR0200 > [Operating instructions]

*) The descriptions in brackets are only included in the instructions of certain devices.

1.3 CODESYS programming manual

17542

In the additional "Programming Manual for CODESYS V2.3" you obtain more details about the use of the programming system.
This manual can be downloaded free of charge from **ifm's** website:
→ www.ifm.com > Select your country > [Service] > [Download] > [Systems for mobile machines]
You also find manuals and online help for **ecomat*mobile*** at:
→ **ecomat*mobile*** DVD "Software, tools and documentation"

1.4 What do the symbols and formats mean?

203

The following symbols or pictograms illustrate the notes in our instructions:

⚠ WARNING

Death or serious irreversible injuries may result.

⚠ CAUTION

Slight reversible injuries may result.

NOTICE

Property damage is to be expected or may result.

❗	Important notes concerning malfunctions or disturbances
ℹ	Other remarks
► ...	Request for action
> ...	Reaction, result
→ ...	"see"
abc	Cross-reference
123 0x123 0b010	Decimal number Hexadecimal number Binary number
[...]	Designation of pushbuttons, buttons or indications

1.5 How is this documentation structured?

204
1508

This documentation is a combination of different types of manuals. It is for beginners and also a reference for advanced users. This document is addressed to the programmers of the applications.

How to use this manual:

- Refer to the table of contents to select a specific subject.
- Using the index you can also quickly find a term you are looking for.
- At the beginning of a chapter we will give you a brief overview of its contents.
- Abbreviations and technical terms → Annex.

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, indicate this number with the title and the language of this documentation. Thank you very much 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 country > [Data sheet search] > (Article no.) > [Operating instructions]

1.6 History of the instructions (CR0200)

9187

What has been changed in this manual? An overview:

Date	Theme	Change
2010-09-09	PID2 (FB)	parameters of the inputs corrected
2010-11-10	Terminating resistors	correction in topic 1244
2011-02-14	TIMER_READ_US (FB)	conversion of max. counter value corrected
2011-04-05	Memory POUs FRAMREAD, FRAMWRITE, FLASHREAD, FLASHWRITE	permitted values of the parameters SRC, LEN, DST
2011-04-13	CANopen overview	new: CANopen tables in the annex
2012-01-09	Memory modules FRAMREAD, FRAMWRITE	Swapped parameters SRC, DST in the table "Permissible values"
2012-07-16	Runtime system	upgrade to v06
2012-10-04	diverse	corrections
2013-06-24	various	new document structure
2014-04-28	Various function blocks	More precise description of the function block input CHANNEL
2014-06-30	Name of the documentation	"System manual" renamed as "Programming manual"
2014-07-31	FB PHASE	Description of parameters of outputs C, ET corrected
2014-08-26	Description of inputs, outputs	highside / lowside replaced by positive / negative switching
2015-03-10	Available memory	Description improved
2015-05-22	FBs INPUT_ANALOG, INPUT_CURRENT, INPUT_VOLTAGE	permissible input channels
2015-05-26	FB J1939_x_GLOBAL_REQUEST	More precise description
2015-06-10	Various function blocks	Description of the FB input CHANNEL corrected

2 Safety instructions

Inhalt

213

2.1 Please note!

214
11212

No characteristics are warranted with the information, notes and examples provided in this manual. With the drawings, representations and examples given no responsibility for the system is assumed and no application-specific particularities are taken into account.

- ► The manufacturer of the machine/equipment is responsible for ensuring the safety of the machine/equipment.
- ► Follow the national and international regulations of the country in which the machine/installation is to be placed on the market!

⚠ **WARNING**

Non-observance of these instructions can lead to property damage or bodily injury!
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 in this manual before working on and with this device.
- ► The acting person must be authorised to work on the machine/equipment.
- ► The acting person must have the qualifications and training required to perform this work.
- ► 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 number.) > [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 number.) > [Operating instructions]
- ► Please note the corrections and notes in the release notes for the existing documentation, available on the **ifm** website:
  → www.ifm.com > Select your country > [Data sheet search] > (article number.) > [Operating instructions]

5020

NOTICE

The driver module of the serial interface can be damaged!

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

- ► Do not disconnect or connect the serial interface while live.

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.

To program the PLC, the people should also be familiar with 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.

2.3 Start-up behaviour of the controller

6827
15233

11575

⚠ 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!
⇨ Realise restart inhibit!

▶ In case of an error, set the outputs concerned to FALSE in the program!

A restart can, for example, be caused by:
- voltage restoration after power failure
- reset after watchdog response because of too long a cycle time
- error elimination after an E-stop

To ensure a safe behaviour of the controller:

▶ monitor the voltage supply in the application program.

▶ In case of an error switch off all relevant outputs in the application program.

▶ Additionally monitor relay contacts which can cause hazardous movements in the application program (feedback).

▶ If necessary, ensure that welded relay contacts in the application project cannot trigger or continue hazardous movements.

▶ Additionally monitor relay contacts which can cause hazardous movements in the application program (feedback).

▶ If necessary, ensure that welded relay contacts in the application project cannot trigger or continue hazardous movements.

3 System description

Inhalt

975

3.1 Information about the device

19958

This manual describes of the **ecomat*mobile*** family for mobile machines of **ifm electronic gmbh**:

- ExtendedController: CR0200 as from production status AI
  Runtime system v06b

3.2 Hardware description

Inhalt

14081

3.2.1 Hardware structure

15332

Conditions

19658

The device does not start until sufficient voltage is applied to the supply connection VBBS (e.g. supply of the relays on the standard side) and to clamp 15.
In vehicles clamp 15 is the plus cable switched by the ignition lock.

Relays

19962

The ExtendedController has 4 internal output relays:

- Standard side: 2 relays each separate 12 outputs from the terminal voltage VBBx (x=O|R),
- Extended side: 2 relays each separate 12 outputs from the terminal voltage VBBx (x=O|R).

Separation is effected upon power-off of the relay.

The output relay (monitoring relay) is triggered by the microcontroller via two channels. To do so, one channel is triggered by an AND function of the watchdog signal (internal microcontroller monitoring) and the system flag bit RELAIS via a solid-state switch. The other channel is only triggered by the system flag bit ERROR via a solid-state switch. When actuated, the terminal voltage VBBx is applied to the outputs via the relay contact (not positively guided).

The clamp relay is activated via one channel via the system flag RELAIS_CLAMP_15 (→ graphics). RELAIS_CLAMP_15 is active after the start of the controller.
The clamp relay switches the VBBO voltage on the second output group.

The clamp relay ensures the internal supply of the device as long as VBBO continues to be applied even if VBBS is interrupted intentionally or unintentionally.

The clamp relay is submitted to the full control in the application program and can be switched via a set/reset command of the system flag RELAIS_CLAMP_15

Prinziple block diagram

19961

The following block diagram shows the dependence of the relays on the applied signals and the logic states of the system flags.

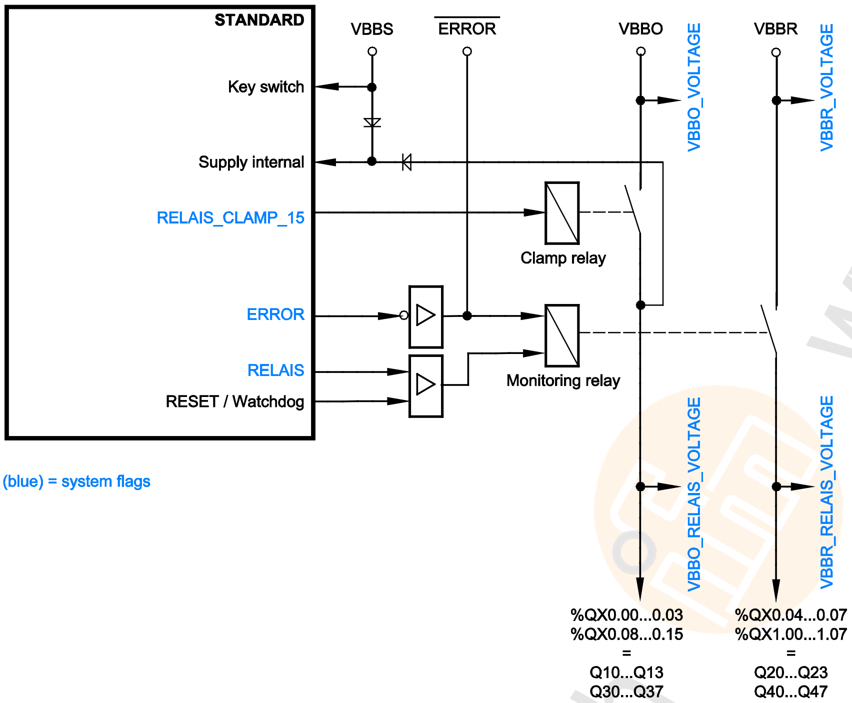
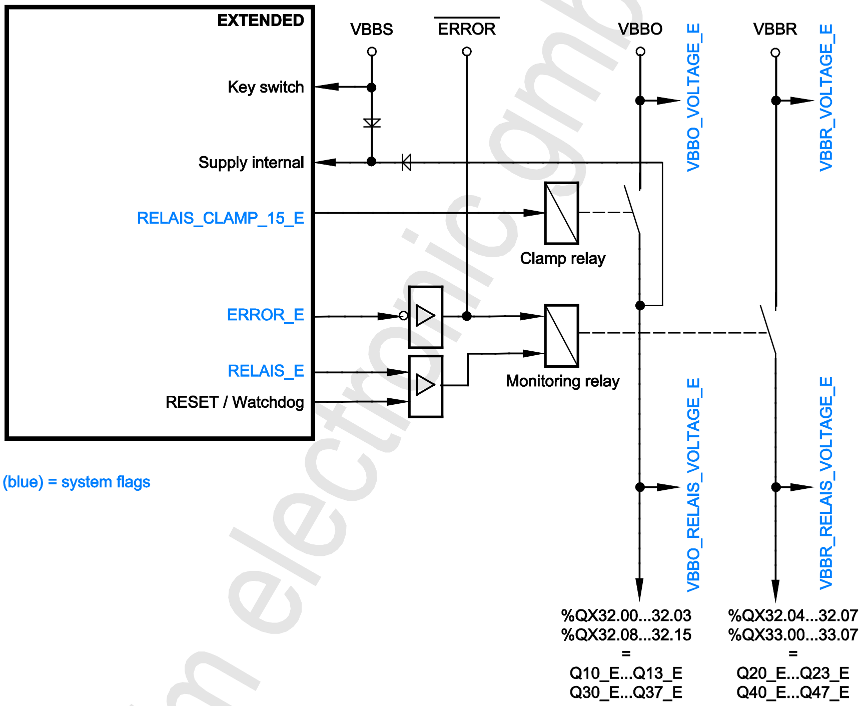


Figure: principle block diagram of supply and relays (standard side)



Grafik: principle block diagram of supply and relays (Extended-Seite)

NOTE

Bei beiden Steuerungshälften...

- das Potential VBBS (Klemme 23) mit dem Zündschalter verbinden und über den Zündschalter abschalten!
- die Systemmerker RELAY_CLAMP_15 und RELAY_CLAMP_15_E zurücksetzen, um den Controller vollständig abzuschalten!

Available memory

13736

FLASH-Speicher

15366

FLASH memory (non-volatile, slow memory) overall existing in the device	2 MByte

Thereof the following memory areas are reserved for ...

maximum size of the application program	704 kByte
data other than the application program user can write data such as files, bitmaps, fonts	1 MByte
data other than the application program read data with *FLASHREAD* (→ page 217) or write data with *FLASHWRITE* (→ page 218) (files: 128 bytes less for header)	64 kByte

The remaining rest of the memory is reserved for system internal purposes.

SRAM

15906

SRAM (volatile, fast memory) overall existing in the device SRAM indicates here all kinds of volatile and fast memories.	512 kByte

Thereof the following memory areas are reserved for ...

data reserved by the application program	160 kByte

The remaining rest of the memory is reserved for system internal purposes.

FRAM

8002

FRAM (non-volatile, fast memory) overall existing in the device FRAM indicates here all kinds of non-volatile and fast memories.	32 kByte

Thereof the following memory areas are reserved for ...

variables in the application program, declared as VAR_RETAIN	1 kByte
as remanent defined flags (from %MB0...) ▶ Set the end of the memory area by FB *MEMORY_RETAIN_PARAM* (→ page 215)!	256 Byte
remanent memory freely available to the user Access is made via *FRAMREAD* (→ page 219) and *FRAMWRITE* (→ page 220)	16 kByte

The remaining rest of the memory is reserved for system internal purposes.

3.2.2 Operating principle of the delayed switch-off

993

If the **ecomat*mobile*** 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 (runtime 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 structure* (→ page 13) 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 VBBO 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 VBBO **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.

3.2.3 Relays: important notes!

19480

Assignment relays – potentials: → data sheet
Max. total current per relay contact (= per output group): → data sheet

NOTICE

Risk of destruction of the relay contacts!
In an emergency situation, "sticking" relay contacts can no longer separate the outputs from the power supply!

Falls VBBS (clamp 15) and VBBO are separated from the power supply at the same time, but the potentials VBBx stay connected to it, then the relays can drop even before the outputs are deactivated by the system.

In this case the relays separate the outputs from the power supply **under load**. This significantly reduces the life cycle of the relays.

► If VBBx is permanently connected to the power supply:
- also connect VBBO permanently and
- switch off the outputs via the program with the help of VBBS (clamp 15).

3.2.4 Monitoring concept

Inhalt

991

The controller monitors the supply voltages and the system error flags. Depending on the status...

- the controller switches off the internal relays
  > the outputs are de-energised, but retain their logic state

or:

- the runtime system deactivates the controller
  > the program stops
  > the outputs change to logic "0"

Operating principle of the monitoring concept

997

During program processing the output 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 output 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 VBBR are disconnected directly by means of the output relay. So the outputs in the current path VBBO are only disconnected via the software.

11575

⚠ **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!
➪ Realise restart inhibit!

► In case of an error, set the outputs concerned to FALSE in the program!

❗ 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 VBBR

20109

Via the potential VBBR up to 12 outputs of the output group can be supplied. The terminal voltage is monitored:

ERROR_VBBR = TRUE	supply voltage is missed or too low
ERROR_VBBR = FALSE	supply voltage is in order

► This information is to be processed in the application program!

Monitoring and securing mechanisms

Inhalt

3926

For the these devices 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

> system data consistency

> CRC of the boot loader

> if exists and is started: CRC of the runtime system

> if exists and is started: CRC of the application program

> 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).

If the TEST pin is not active

3929

- Write protection for system data in FRAM [1], e.g.:
  - runtime system loaded,
  - calibration data.
  Implemented via hardware and software.
- Write protection for application program (in the flash memory)
- DEBUG mode

[1] FRAM indicates here all kinds of non-volatile and fast memories.

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.

3.2.5 Inputs (technology)

Inhalt

14090

Analogue inputs

2426

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

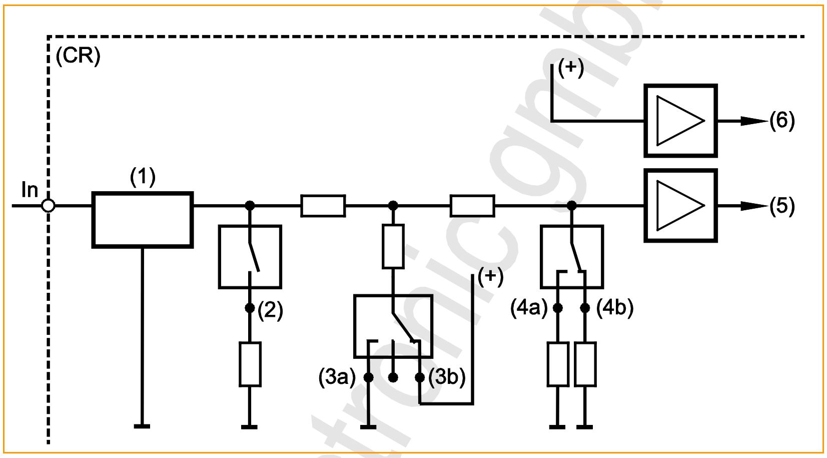
- current input 0...20 mA
- voltage input 0...10 V
- voltage input 0...32 V

The voltage measurement can also be carried out ratiometrically (0...1000 ‰, adjustable via function blocks). This means potentiometers or joysticks can be evaluated without additional reference voltage. A fluctuation of the supply voltage has no influence on this measured value.

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

> ❗ In case of ratiometric measurement the connected sensors should be supplied with VBBS of the device. So, faulty measurements caused by offset voltage are avoided.

8971



In = pin multifunction input n
(CR) = device
(1) = input filter
(2) = analogue current measuring
(3a) = binary-input plus switching
(3b) = binary-input minus switching
(4a) = analogue voltage measuring 0...10 V
(4b) = analogue voltage measuring 0...32 V
(5) = voltage
(6) = reference voltage

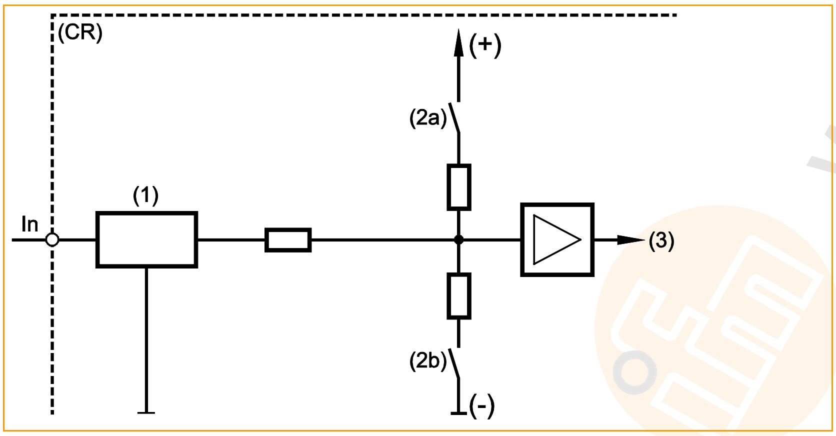
Figure: principle block diagram multifunction input

Digital inputs

1015
7345

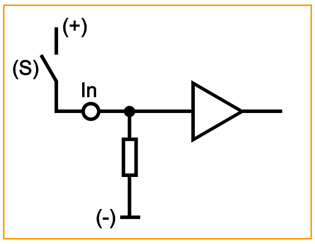
The binary input can be operated in following modes:
- binary input plus switching (BL) for positive sensor signal
- binary input minus switching (BH) for negative sensor signal

Depending on the device the binary inputs can configured differently. In addition to the protective mechanisms against interference, the binary 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



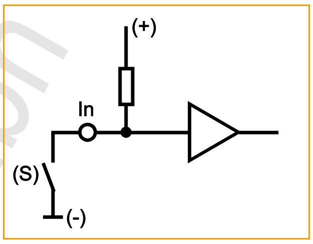
In = pin binary-input n
(CR) = device
(1) = input filter
(2a) = input minus switching
(2b) = input plus switching
(3) = voltage

Figure: basic circuit of binary input minus switching / plus switching for negative and positive sensor signals



In = pin binary input n
(S) = sensor

Basic circuit of binary input plus switching (BL)
for positive sensor signal:
Input = open ⇨ signal = low (GND)



In = pin binary input n
(S) = sensor

Basic circuit of binary input minus switching (BH)
for negative sensor signal:
Input = open ⇨ signal = high (supply)

For some of these inputs (→ data sheet) the potential can be selected to which it will be switched.

Input group I0 (I00...07 / ANALOG0...7)

20389

These inputs are a group of multifunction channels.

These inputs can be used as follows (each input separately configurable):

- analogue input 0...20 mA
- analogue input 0...10 V
- analogue input 0...32 V
- voltage measurement ratiometric 0...1000 ‰
- binary input plus switching (BL) for positive sensor signal (with/without diagnosis)

→ chapter *Possible operating modes inputs/outputs* (→ page 247)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

All inputs show the same behaviour concerning function and diagnosis.

ℹ Detailed description → chapter *Address assignment inputs / outputs* (→ page 240)

In the application program, the system variables ANALOG00...ANALOGxx can be used for customer-specific diagnostics.

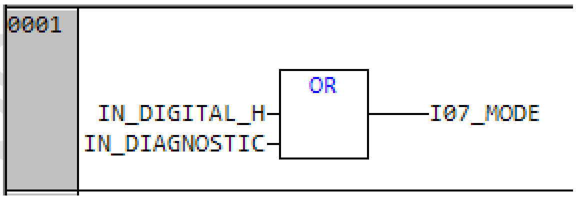
If the analogue inputs are configured for current measurement, the operating mode of the input switches to the safe voltage measurement range (0...30V DC) and the corresponding error bit in the flag byte ERROR_IO is set when the final value (> 21 mA) is exceeded. If the value is again below the limit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

► Configuration of each input is made via the application program:
- FB *INPUT_ANALOG* (→ page 151) > input MODE
- Configuration byte Ixx_MODE

15380

Example with configuration byte Ixx_MODE:

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



13956

> The result of the diagnostics is for example shown by the following system flags:

System flags (symbol name)	Type	Description
ERROR_BREAK_Ix (0...x, value depends on the device, → data sheet)	DWORD	input group x: wire break error or (resistance input): short to supply [Bit 0 for input 0] ... [bit z for input z] of this group Bit = TRUE: error Bit = FALSE: no error

ERROR_SHORT_Ix
(0...x, value depends on the device, → data sheet)
	DWORD	input group x: short circuit error
		[Bit 0 for input 0] ... [bit z for input z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

NAMUR diagnosis for binary signals of non-electronic switches:

► Equip the switch with an additional resistor connection!

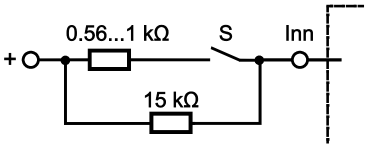


Figure: non-electronic switch S at input Inn

Input group I1 (I10...17 / FRQ0...3)

19487

Inputs I10...13

19490

These inputs are a group of multifunction channels.
These inputs can be used as follows (each input separately configurable):

- binary input plus switching (BL) for positive sensor signal
- Output (→ chapter *Outputs (technology)* (→ page 27))

→ chapter *Possible operating modes inputs/outputs* (→ page 247)

These inputs cannot be configured.

Inputs I14...17 / FRQ0...3

19497

These inputs are a group of multifunction channels.
These inputs can be used as follows (each input separately configurable):

- binary input plus switching (BL) for positive sensor signal
- fast input for e.g. incremental encoders and frequency or interval measurement

→ chapter *Possible operating modes inputs/outputs* (→ page 247)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

► Configuration of each input is made via the application program:
- Configuration byte Ixx_MODE
- Fast inputs with the following FBs:

FAST_COUNT (→ page 159)	Counter block for fast input pulses
FREQUENCY (→ page 160)	Measures the frequency of the signal arriving at the selected channel
INC_ENCODER (→ page 161)	Up/down counter function for the evaluation of encoders
PERIOD (→ page 164)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 166)	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 [‰].
PHASE (→ page 168)	Reads a pair of channels with fast inputs and compares the phase position of the signals

Input group I2 (I20...27)

19489

Inputs I20...23

19499

These inputs are a group of multifunction channels.
These inputs can be used as follows (each input separately configurable):

- binary input plus switching (BL) for positive sensor signal
- Output (→ chapter *Outputs (technology)* (→ page 27))

→ chapter *Possible operating modes inputs/outputs* (→ page 247)

These inputs cannot be configured.

Inputs I24...27 / CYL0...3

19500

These inputs are a group of multifunction channels.
These inputs can be used as follows (each input separately configurable):

- binary input plus switching (BL) for positive sensor signal
- fast input for e.g. incremental encoders and frequency or interval measurement

→ chapter *Possible operating modes inputs/outputs* (→ page 247)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

► Configuration of each input is made via the application program:
  - Configuration byte Ixx_MODE
  - Fast inputs with the following FBs:

FAST_COUNT (→ page 159)	Counter block for fast input pulses
FREQUENCY (→ page 160)	Measures the frequency of the signal arriving at the selected channel
INC_ENCODER (→ page 161)	Up/down counter function for the evaluation of encoders
PERIOD (→ page 164)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 166)	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 [‰].
PHASE (→ page 168)	Reads a pair of channels with fast inputs and compares the phase position of the signals

3.2.6 Outputs (technology)

Inhalt

14093

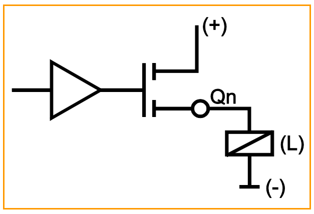
Binary outputs

14094

The following operating modes are possible for the device outputs (→ data sheet):

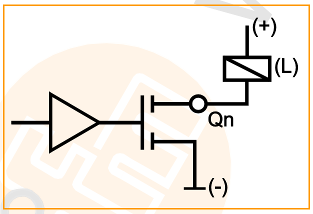
- binary output, plus switching (BH) with/without diagnostic function
- binary output minus switched (BL) without diagnostic function

15450



Qn = pin output n
(L) = load

Basic circuit of output plus switching (BH) for positive output signal



Qn = pin output n
(L) = load

Basic circuit of output minus switching (BL) for negative output signal

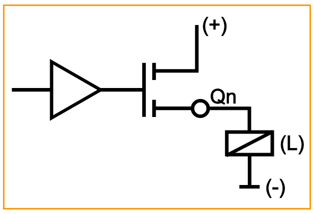
PWM outputs

14095

The following operating modes are possible for the device outputs (→ data sheet):

- PWM output, plus switching (BH) without diagnostic function

15451



Qn = pin output n
(L) = load

Basic circuit of output plus switching (BH) for positive output signal

Output group Q1Q2 (Q10...13 / Q20...23)

19507

These outputs are a group of multifunction channels.

These outputs provide several function options (each output separately configurable):
- binary output, plus switching (BH) with diagnostic function and protection
- analogue current-controlled output (PWMi)
- analogue output with Pulse Width Modulation (PWM)
- binary input (→ chapter *Inputs (technology)* (→ page 22))

→ chapter *Possible operating modes inputs/outputs* (→ page 247)

If the outputs are not used as PWM outputs, the diagnostics is carried out via the integrated current measurement channels which are also used for the current-controlled output functions.

► Configuration of each output is made via the application program:
indicate the load currents → FB *OUTPUT_CURRENT* (→ page 172)
PWM output: → FB *PWM1000* (→ page 181)
Configuration byte Qxx_MODE

13975

⚠ WARNING

Dangerous restart possible!
Risk of personal injury! Risk of material damage to the machine/plant!

If in case of a fault an output is switched off via the hardware, the logic state generated by the application program is not changed.

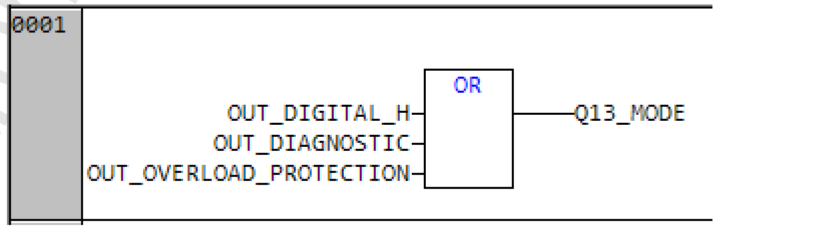
► Remedy:
- Reset the output logic in the application program!
- Remove the fault!
- Reset the outputs depending on the situation.

! The outputs in the PWM mode support no diagnostic functions.

When used as binary output, configuration is carried out using the system variables Q1x_MODE...Q2x_MODE. If the diagnostics 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 or 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 diagnostics. Overload protection is activated (preset).



! NOTE

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

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

The function OUT_OVERLOAD_PROTECTION is not supported in the pure PWM mode.

13976

! Depending on the ambient temperature a short circuit cannot be reliably detected from a certain short circuit current since the output drivers temporarily deactivate themselves for protection against destruction.

Diagnosis: binary outputs (via current measurement)

19398
19396

The diagnostics of these outputs is made via internal current measurement in the output:

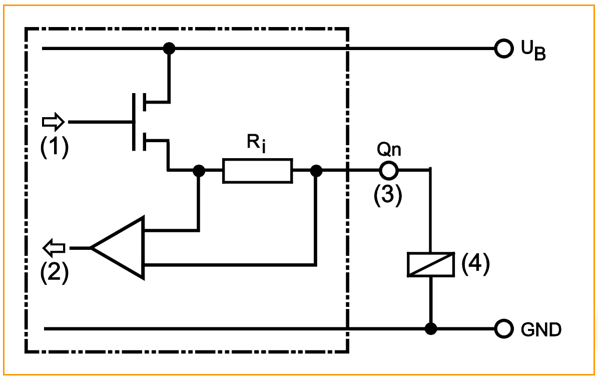


Figure: principle block diagram

(1) Output channel

(2) Read back channel for diagnostics

(3) Pin output n

(4) Load

Diagnosis: overload (via current measurement)

19437
15249

Overload can only be detected on an output with current measurement.

Overload is defined as ...
"a nominal maximum current of 12.5 %".

Diagnosis: wire break (via current measurement)

19400

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

Diagnosis: short circuit (via voltage measurement)

19401

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

3.2.7 Note on wiring

1426

The wiring diagrams (→ installation instructions of the devices, chapter "Wiring") describe the standard device configurations. The wiring diagram helps allocate the input and output channels to the IEC addresses and the device terminals.

The individual abbreviations have the following meaning:

A	Analogue input
BH	Binary high side input: minus switching for negative sensor signal Binary high side output: plus switching for positive output signal
BL	Binary low side input: plus switching for positive sensor signal Binary low side output: minus switching for negative output signal
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
PWMi	PWM output with current measurement
IH	Pulse/counter input, high side: minus switching for negative sensor signal
IL	Pulse/counter input, low side: plus switching for positive sensor signal
R	Read back channel for one output

Allocation of the input/output channels: → Catalogue, mounting instructions or data sheet

3.2.8 Safety instructions about Reed relays

7348

For use of non-electronic switches please note the following:

> ! 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

3.2.9 Feedback on bidirectional inputs/outputs

999

Some terminals of the controller can be configured as input or output (→ data sheet).

NOTICE

Destruction of outputs if there is inadmissible feedback!

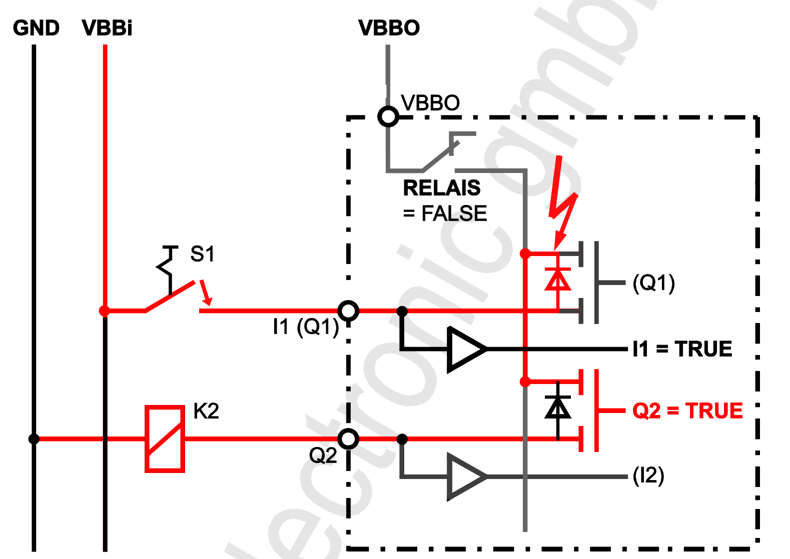
If a group of bidirectional inputs/outputs is operated at the same time with inputs and outputs, the potential VBB of this output group **must not** become potential-free.

The output group is potential free if e.g. ...
- RELAIS = FALSE or
- RELAIS_CLAMP_15 = FALSE.

This potential-free status has the consequence that the voltage is fed back via the protective diode of the output transistor if within an input/output group:
- an input (e.g. I1) = TRUE and
- an output of the same group (e.g. Q2) = TRUE.

> **Consequence:**
The load on the output (Q2) receives voltage via the protective diode of the input (I1). The protective diode and thus the output (Q1) via which the feedback current flows at that moment, can be destroyed.

► **Remedy:**
Operate an input/output group only as inputs OR only as outputs.
or:
Follow the note below.



Example:
The flag RELAIS switches off the VBBO potential of the output group.

The external switch S1 supplies the VBBi potential to input I1.

If output Q2 = TRUE (→ graphic), K2 will receive voltage via the protective diode Q1 despite RELAIS = FALSE (red lines). Due to overload this protective diode burns out and the output Q1 is destroyed!

Graphic: examples of inadmissible connection: danger of feedback!

NOTE

Help for mixed operated bidirectional inputs/outputs:

► Set the flag RELAIS and/or RELAIS_CLAMP_15 in the application program permanently to TRUE:
- TRUE ----- RELAIS
- TRUE ----- RELAIS_CLAMP_15

3.2.10 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).

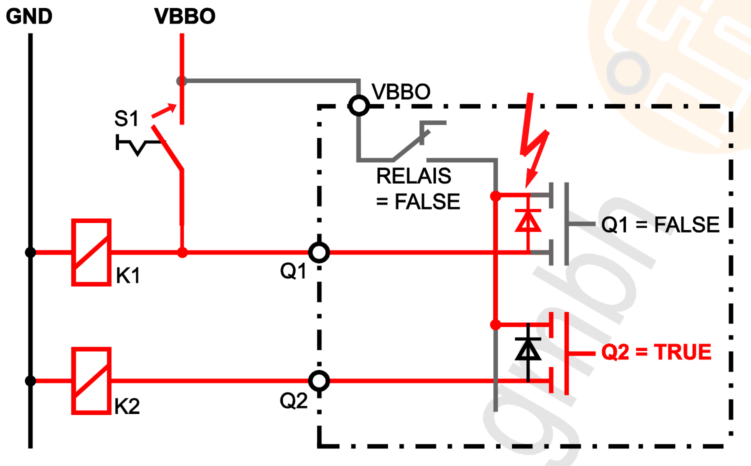
NOTICE

Destruction of outputs if there is inadmissible feedback!

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

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

► Protect externally supplied outputs by means of blocking diodes!

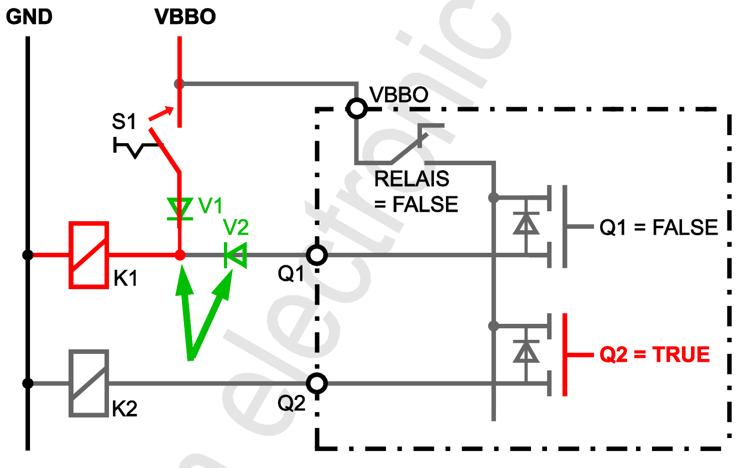


Example:

The flag RELAIS switches off the supply VBBO of the output group.

Without blocking diodes the external switch S1 feeds the supply VBBO via the internal protective diode (red) from output Q1 to the internal potential bar of the outputs.

If output Q2 = TRUE (→ graphic), K2 will receive voltage via the protective diode Q1 despite RELAIS = FALSE (red lines). Due to overload this protective diode burns out and the output Q1 is destroyed!



Graphic: example wiring with blocking diodes due to the danger of feedback

Remedy:

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

Successful:

If RELAIS = FALSE, K2 remains switched off, even if Q2 = TRUE.

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.

3.2.11 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
yellow	briefly on	initialisation or reset checks
green / black	5 Hz	no runtime system loaded
green / black	2 Hz	application RUN
green	permanently on	application STOP
red / black	2 Hz	application RUN with error
red	briefly on	fatal error
red	permanently on	fatal error (if input TEST = active) ERROR STOP / STYSTEM STOP

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

Control the LED in the application program

9989

For this device the status LED can also be set by the application program. To do so, the following system variables are used (→ *System flags* (→ page 232)):

LED	LED color for "active" (for "on")
LED_X	LED color for "Pause" (for "Off" or different colour)
---	Color constant from the data structure "LED_COLOR". Permissible entries: LED_GREEN LED_BLUE LED_RED LED_WHITE LED_MAGENTA LED_CYAN LED_YELLOW LED_ORANGE LED_BLACK (= LED off)
LED_MODE	Flashing frequency from the data structure "LED_MODES". Permissible entries: LED_2HZ LED_1HZ LED_05HZ (= 0.5 Hz) LED_0HZ (= constant)

NOTE

► Do NOT use the LED color RED in the application program.

> In case of an error the LED color RED is set by the runtime system.
BUT: If the colors and/or flashing modes are changed in the application program, the above table with the default setting is no longer valid.

3.3 Interface description

Inhalt

14098

3.3.1 Serial interface

14099

This device features a serial interface.

The serial interface can generally be used in combination with the following functions:
- program download
- debugging
- free use of the application

NOTE

The serial interface is not available to the user by default, 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=TRUE.
Debugging of the application program is then only possible via one of the 4 CAN interfaces or via USB.

Connections and data → data sheet

3.3.2 SSC interface

20533

This interface is only available in the following devices:
- ExtendedController: CR0200
- ExtendedSafetyController: CR7200, CR7201

The SSC interface is used for non-safe data exchange between the standard side and the extended side of the controller.

A maximum of 16 messages with 20 bytes each can be transmitted in one control cycle.
The following FBs are used *SSC_TRANSMIT* (→ page 142) and *SSC_RECEIVE* (→ page 139).

3.3.3 CAN interfaces

Inhalt

14101

Connections and data → data sheet

Diagnosis: wire break (via current measurement)

19435
19400

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

3.4 Software description

Inhalt

14107

3.4.1 Software modules for the device

Inhalt

14110

The software in this device communicates with the hardware as below:

software module	Can user change the module?	By means of what tool?
Application program with libraries	Yes	CODESYS, MaintenanceTool
Runtime system *)	Upgrade yes Downgrade no	MaintenanceTool
Bootloader	No	---
(Hardware)	No	---

*) The runtime system version number must correspond to the target version number in the CODESYS target system setting.
→ chapter *Set up the target* (→ page 55)

Below we describe this software module:

Bootloader

14111

On delivery **ecomat*mobile*** controllers only contain the boot loader.
The boot loader is a start program that allows to reload the runtime system and the application program on the device.
The boot loader contains basic routines...

- for communication between hardware modules,
- for reloading the operating system.

The boot loader is the first software module to be saved on the device.

Runtime system

14112

Basic program in the device, establishes the connection between the hardware of the device and the application program.

On delivery, there is normally no runtime system loaded in the controller (LED flashes green at 5 Hz). Only the bootloader is active in this operating mode. It provides the minimum functions for loading the runtime system, among others support of the interfaces (e.g. CAN).

Normally it is necessary to download the runtime system only once. Then, the application program can be loaded into the controller (also repeatedly) without affecting the runtime system.

The runtime 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**:
→ www.ifm.com > Select your country > [Service] > [Download]

Application program

14118

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.

8340

⚠ **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.

Libraries

19527

ifm electronic offers a series of libraries (`*.LIB`) suitable for each device, containing the program modules for the application program. Examples:

Library	Usage
`ifm_CR0200_Vxxyyzz.LIB`	Device-specific library Must always be contained in the application program!
`ifm_CAN1_EXT_Vxxyyzz.LIB`	(optional) if CAN interface 1 of the device is to operate on 29 bits
`ifm_CR0200_CANopenMaster_Vxxyyzz.LIB`	(optional) if CAN interface 1 of the device is to be operated as a CANopen master
`ifm_CR0200_CANopenSlave_Vxxyyzz.LIB`	(optional) if CAN interface1 of the device is to be operated as a CANopen slave

`ifm_CR0200_J1939_`**`x`**`_Vxxyyzz.LIB`
x = 1...2 = number of the CAN interface
	(optional)
	if a CAN interface of the device is to communicate with a motor control

→ chapter *ifm libraries for the device CR0200* (→ page 74)

3.4.2 Programming notes for CODESYS projects

Inhalt

7426

Here you receive tips how to program the device.

► See the notes in the CODESYS programming manual
→ www.ifm.com > select your country > [Data sheet search] > CR0200 > [Operating instructions]
→ **ecomat*mobile*** DVD "Software, tools and documentation".

FB, FUN, PRG in CODESYS

8473

In CODESYS we differentiate between the following types of function elements:

FB = function block

- An FB can have several inputs and several outputs.
- An FB may be called several times in a project.
- An instance must be declared for each call.
- Permitted: Call FB and FUN in FB.

FUN = function

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

PRG = program

- A PRG can have several inputs and several outputs.
- A PRG may only be called once in a project.
- Permitted: Call PRG, FB and FUN in PRG.

NOTE

Function blocks must NOT be called in functions!
Otherwise: During execution the application program will crash.

All function elements must NOT be called recursively, nor indirectly!

An IEC application must contain max. 8,000 function elements!

Background:

All variables of functions...

- are initialised when called and
- become invalid after return to the caller.

Function blocks have 2 calls:

- an initialisation call and
- the actual call to do something.

Consequently that means for the FB call in a function:

- every time there is an additional initialisation call and
- the data of the last call gets lost.

Note the cycle time!

8006

For the programmable devices from the controller family **ecomat*mobile*** 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 device 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.

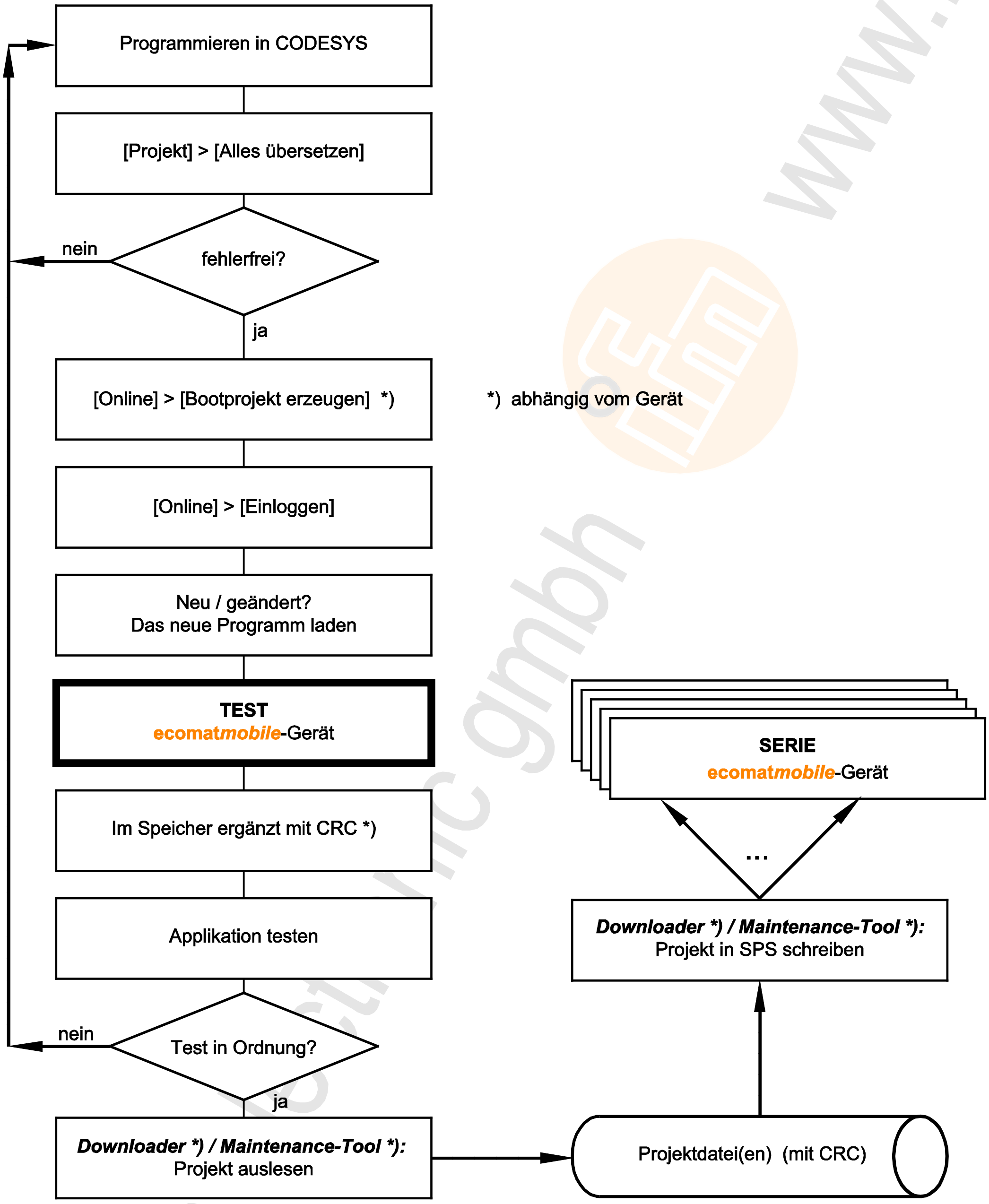
► If necessary, the cycle time must be optimised by restructuring the software and the system set-up.

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] > [Login] > load the new program.

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.



Graphics: Creation and distribution of the software

Save boot project

7430

ⓘ Always save the related boot project together with your application project in the device. Only then will the application program be available after a power failure in the device.

> **NOTE**
>
> Note: The boot project is slightly larger than the actual program.
>
> However: Saving the boot project in the device will fail if the boot project is larger than the available IEC code memory range. After power-on the boot project is deleted or invalid.

► CODESYS menu [Online] > [Create boot project]
This is necessary after each change!

> After a reboot, the device starts with the boot project last saved.

> If NO boot project was saved:
• The device remains in the STOP operation after reboot.
• The application program is not (no longer) available.
• The LED lights green.

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.

Here you will find the current **ifm** downloader (min. V06.18.26):
ecomat*mobile* DVD "Software, tools and documentation" under the tab 'R360 tools [D/E]'

Using ifm maintenance tool

8492

The **ifm** Maintenance Tool 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** Maintenance Tool. Advantage: A programming system with CODESYS licence is not required.

Here you will find the current **ifm** Maintenance Tool:
→ www.ifm.com > Select your country > [Service] > [Download] > [Systems for mobile machines]
→ **ecomat*mobile*** DVD "Software, tools and documentation" under the tab 'R360 tools [D/E]'

3.4.3 Operating states

Inhalt

14120

After power on the **ecomat*mobile*** device can be in one of five possible operating states:

- BOOTLOADER
- INIT
- STOP
- RUN
- SYSTEM STOP (after ERROR STOP)

Operating states: runtime system is not available

19217

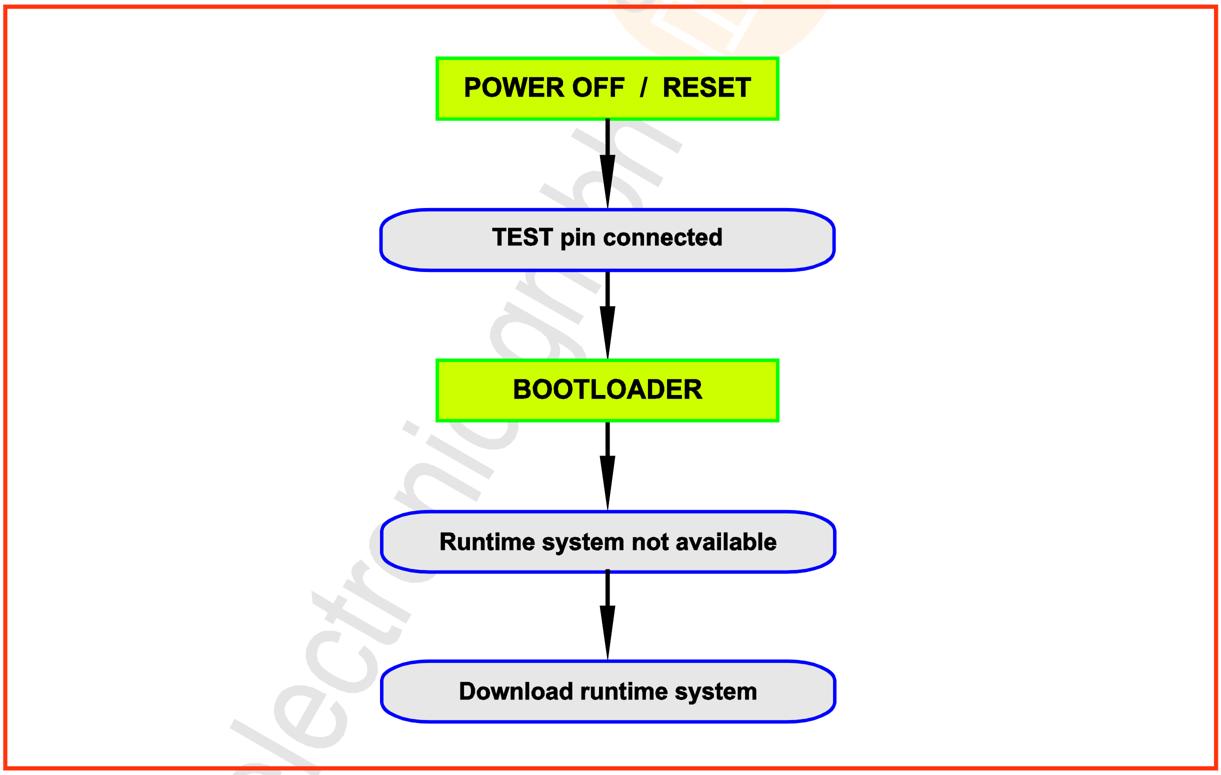


Figure: operating states (here: runtime system is not available)

Operating states: application program is not available

19218

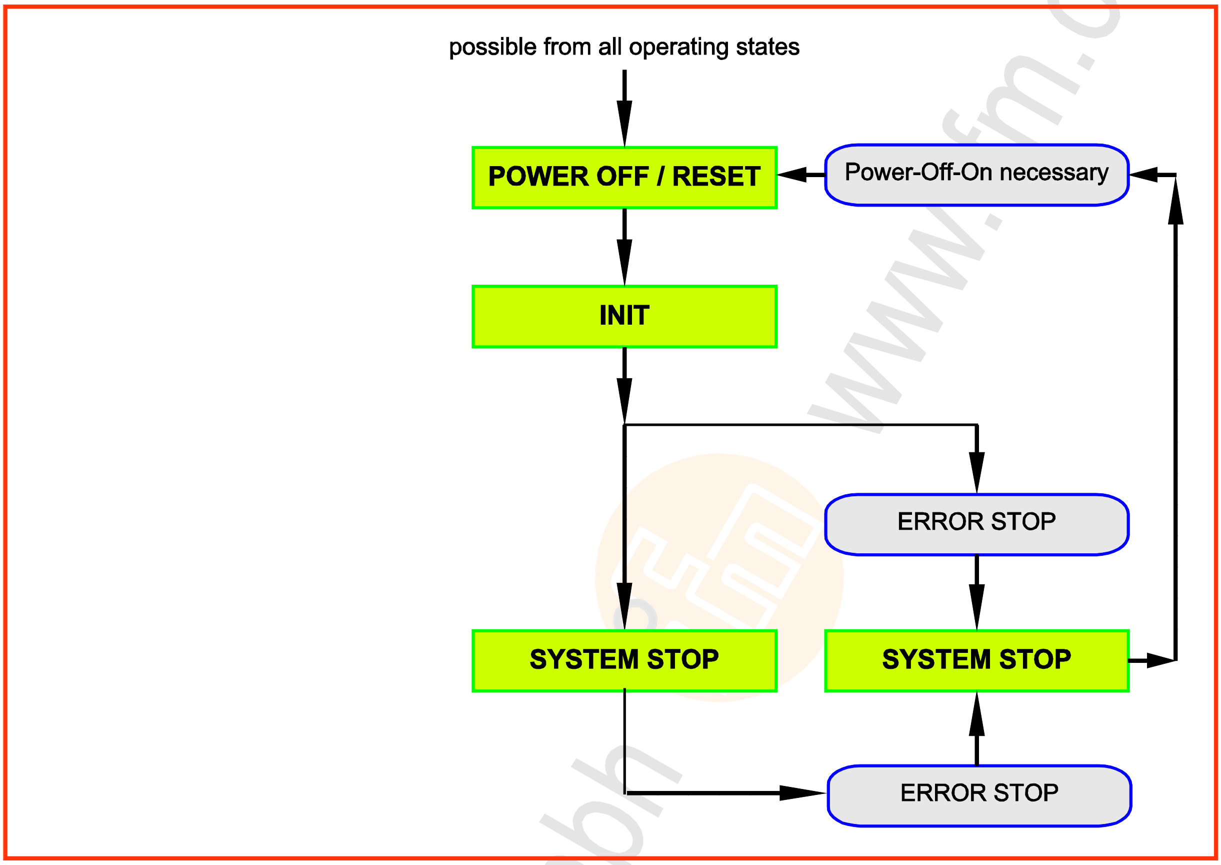


Figure: operating states (here: application program is not available)

Operating states: application program is available

19219

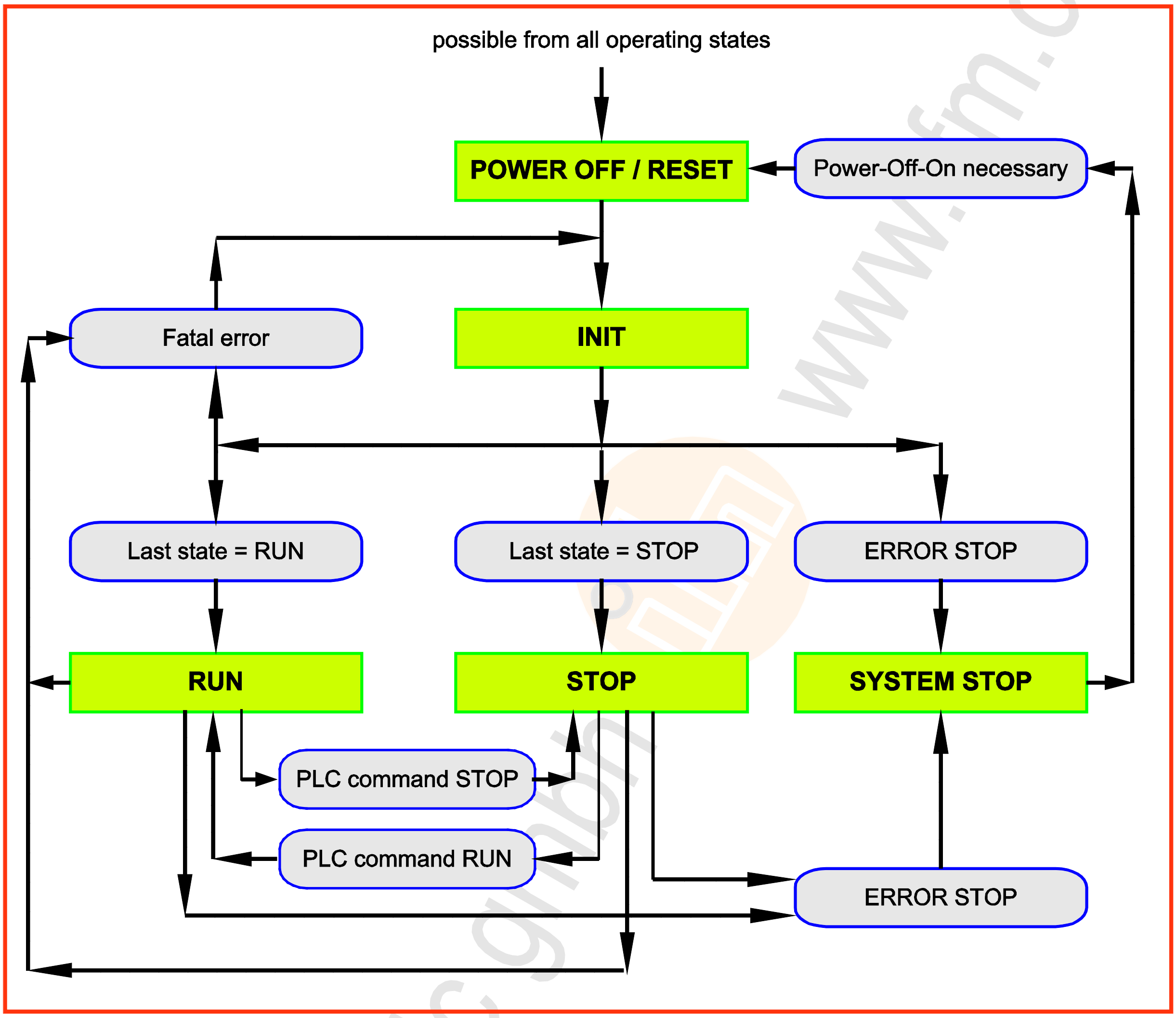


Figure: operating states (here: application program is available)

Bootloader state

1080

No runtime system was loaded. The **ecomat*mobile*** controller is in the boot loading state. Before loading the application software the runtime system must be downloaded.

> The LED flashes green (5 Hz).

INIT state (Reset)

1076

Premise: a valid runtime system is installed.

This state is passed through after every power on reset:

> The runtime 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
• STOP

STOP state

1078

This state is reached in the following cases:

- From the RESET state if:
  • no program is loaded or
  • the last state before the RESET state was the STOP state
- From the RUN state by the STOP command
  • only for the operating mode = Test (→ chapter *TEST mode* (→ page 47))

> The LED lights green.

RUN state

1077

This state is reached in the following cases:

- From the RESET state if:
  • the last state before the RESET state was the RUN state
- From the STOP state by the RUN command
  • only for the operating mode = Test (→ chapter *TEST mode* (→ page 47))

> The LED flashes green (2 Hz).

SYSTEM STOP state

19222

The **ecomat*mobile*** controller goes to this state if a non tolerable error (ERROR STOP) was found. This state can only be left by a power-off-on reset.

> The LED lights red.

3.4.4 Operating modes

1083

Independent of the operating states the **ecomat*mobile*** controller can be operated in different modes.

TEST mode

1084

NOTICE

Loss of the stored software possible!
In the test mode there is no protection of the stored runtime system and application software.

NOTE

▶ Connect the TEST connection to the supply voltage only AFTER you have connected the OPC client!

> Otherwise a fatal error will occur.

This operating mode is reached by applying supply voltage to the test input
(→ installation instructions > chapter "Technical data" > chapter "Wiring").

The **ecomat*mobile*** controller can now receive commands via one of the interfaces in the RUN or STOP mode and, for example, communicate with the programming system.

Only in the TEST mode the software can be downloaded to the controller.

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

Summary Test input is active:
- Programming mode is enabled
- Software download is possible
- Status of the application program can be queried
- Protection of stored software is not possible

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.

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 *Function elements: serial interface* (→ page 132)

DEBUG mode

1086

If the input DEBUG of *SET_DEBUG* (→ page 226) 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 *TEST mode* (→ page 47)) is not connected to supply voltage.

3.4.5 Performance limits of the device

7358

! Note the limits of the device! → Data sheet

Above-average stress

1488
5023

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

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:

Restrictions for the use of FBs

1489

Current controller	max. 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	If possible, do not use any other performance-affecting functions!

1509

NOTICE

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.

Watchdog behaviour

11786

In this device, a watchdog monitors the program runtime of the CODESYS application.

If the maximum watchdog time (approx. 100 ms) is exceeded:
> the device performs a reset and reboots.

This you can read in the flag LAST_RESET.

4 Configurations

Inhalt

1016

The device configurations described in the corresponding installation instructions or in the *Annex* (→ page 232) 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.

4.1 Operating modes of the ExtendedController

1413

The ExtendedController can be operated in two different ways

- two separate applications in the two controller halves
- only **one** controller halve (which halve can be freely defined) with a complete application program

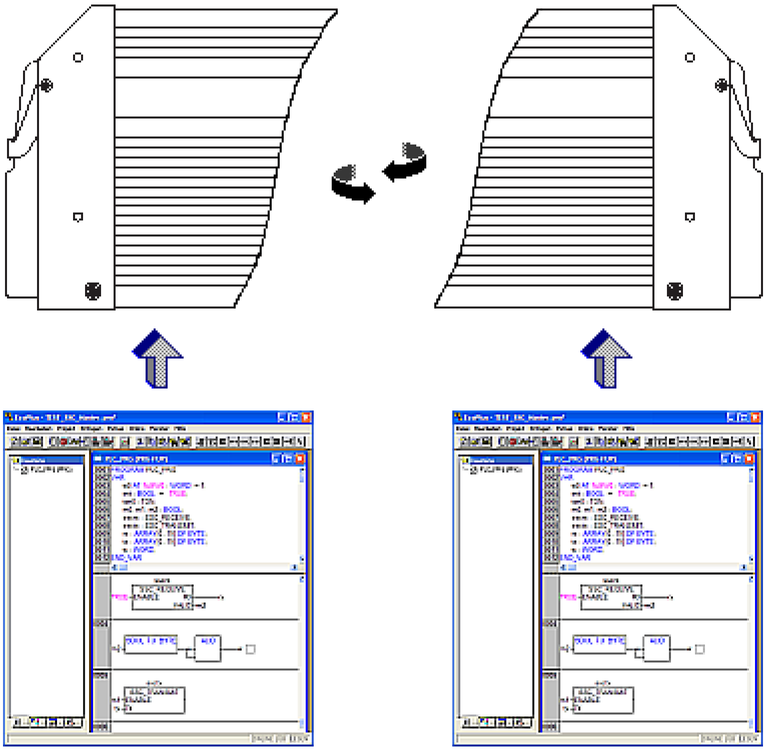
4.1.1 Variant 1: separated programs in both controller halves

1414

An independent application program operates in each controller half.

- Both application programs function completely asynchronously and independently of each other.
- Each controller half operates as master its side.
- The inputs and outputs are addressed in both controller halves via the same system variables and system functions.
- As an option, the internal interface is used for the data exchange between the two controller halves.

► Use the FBs *SSC_TRANSMIT* (→ page 142) and *SSC_RECEIVE* (→ page 139) in both application programs.



Graphics: Both controller halves used

4.1.2 Option 2: only one program for both controller halves

1415

An application program operates in any of the controller halves using both controller halves.

- The controller half with the application program operates as master on its side.
  The other controller half operates as slave.
  ► Use the FB *SSC_SET_MASTER* (→ page 140) in the application program (on the master side).
  > The FB assumes the function of initialising the slave.

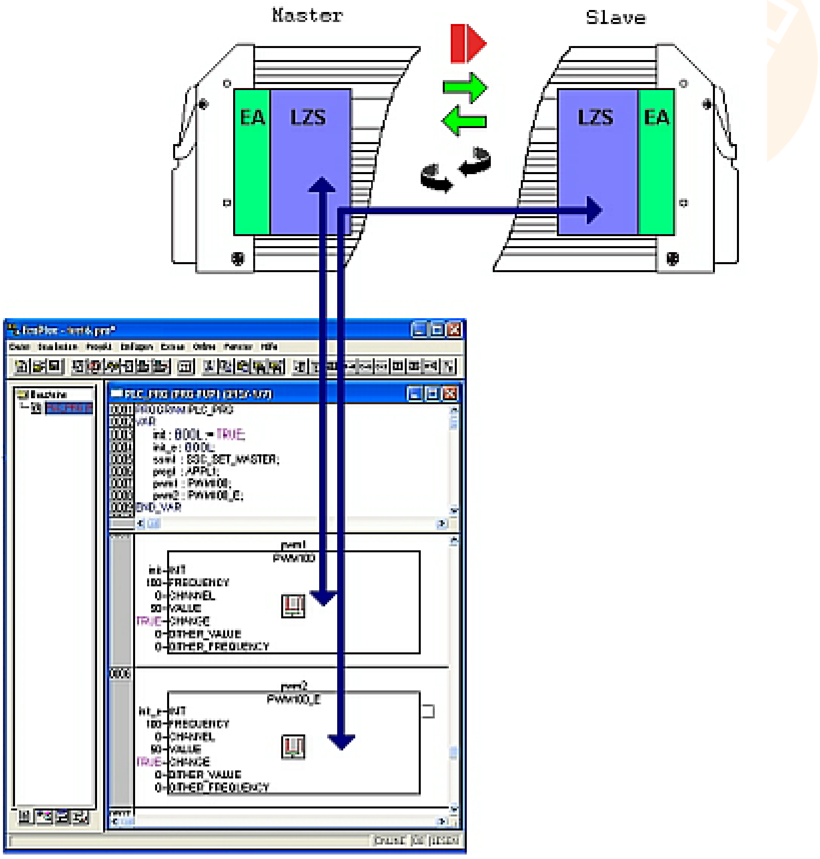
► Do NOT load ANY application program into the slave side of the controller.

> The master loads a small dummy program into the slave.
  ► Use the FB *DUMMY* (→ page 138) from the library `ifm_CR0200_DUMMY_Vxxyyzz.LIB` in the application program.

> The inputs and outputs of both controller halves are processed synchronously.

> For differentiation, the names of the variables and functions of the slave side are extended by an '_E' (for extended).

> The data exchange between the two halves of the controller is automatically carried out via the internal SSC interface.



```
PROGRAM PLC_PRG
VAR
    init : BOOL := TRUE;
    init_e : BOOL;
    saml  : SSC_SET_MASTER;
    progl  : APPL1;
    pwm1 : PWM100;
    pwm2 : PWM100_E;
END_VAR
```

Legend:

▶	Synchronisation master ⇨ slave
⇄	Exchange of input/output variables and system flags
↻	Exchange of messages of the application programs

4.2 Set up the runtime system

Inhalt

14091

4.2.1 Reinstall the runtime system

14092
2733

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

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

The runtime 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]

2689

NOTE

The software versions suitable for the selected target must always be used:
- runtime system (`ifm_CR0200_Vxxyyzz.H86`),
- PLC configuration (`ifm_CR0200_Vxx.CFG`),
- device library (`ifm_CR0200_Vxxyyzz.LIB`) and
- the further files.

V	version
xx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "`CR0200`") 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.

4368

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`).

It may happen that the target system cannot or only partly be programmed with your currently installed version of CODESYS. In such a case, please contact the technical support department of **ifm electronic gmbh**.

The runtime system is transferred to the device using the separate program "**ifm** downloader". (The downloader is on the **ecomat*mobile*** DVD "Software, tools and documentation" or can be downloaded from **ifm's** website, if necessary): → www.ifm.com > Select your country > [Service] > [Download].

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

4.2.2 Update the runtime system

13269

An older runtime system is already installed on the device. Now, you would like to update the runtime system on the device?

14158

NOTICE

Risk of data loss!

When deleting or updating the runtime system all data and programs on the device are deleted.

► Save all required data and programs before deleting or updating the runtime system!

3084

When the operating system software or the CODESYS runtime system is considerably improved, **ifm** releases a new version. The versions are numbered consecutively (V01, V02, V03, ...).

Please see the respective documentation for the new functions of the new software version. Note whether special requirements for the hardware version are specified in the documentation.

If you have a device with an older version and if the conditions for the hardware and your project are OK, you can update your device to the new software version.

For this operation, the same instructions apply as in the previous chapter 'Reinstall the runtime system'.

4.2.3 Verify the installation

14512

► After loading of the runtime system into the controller:
• check whether the runtime system was transmitted correctly!
• check whether the right runtime system is on the controller!

► 1st check:
use the **ifm** downloader or the maintenance tool to verify whether the correct version of the runtime system was loaded:
• read out the name, version and CRC of the runtime system in the device!
• Manually compare this information with the target data!

► 2nd check (optional):
verify in the application program whether the correct version of the runtime system was loaded:
• read out the name and version of the runtime system in the device!
• Compare this data with the specified values!
The following FB serves for reading the data:

GET_IDENTITY (→ page 225)	Reads the specific identifications stored in the device: • hardware name and hardware version of the device • name of the runtime system in the device • version and revision no. of the runtime system in the device • name of the application (has previously been saved by means of *SET_IDENTITY* (→ page 227))

4.3 Set up the programming system

Inhalt

3968

4.3.1 Set up the programming system manually

Inhalt

3963

Set up the target

2687
11379

When creating a new project in CODESYS the target file corresponding to the device must be loaded.

- ► Select the requested target file in the dialogue window [Target Settings] in the menu [Configuration].
- > The target file constitutes the interface to the hardware for the programming system.
- > At the same time, several important libraries and the PLC configuration are loaded when selecting the target.
- ► If necessary, in the window [Target settings] > tab [Network functionality] > activate [Support parameter manager] and / or activate [Support network variables].
- ► If necessary, remove the loaded (3S) libraries or complement them by further (ifm) libraries.
- ► Always complement the appropriate device library `ifm_CR0200_Vxxyyzz.LIB` manually!

2689

❗ NOTE

The software versions suitable for the selected target must always be used:
• runtime system (`ifm_CR0200_Vxxyyzz.H86`),
• PLC configuration (`ifm_CR0200_Vxx.CFG`),
• device library (`ifm_CR0200_Vxxyyzz.LIB`) and
• the further files.

V	version
xx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

The basic file name (e.g. "`CR0200`") 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.

4368

❗ 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`).

ℹ It may happen that the target system cannot or only partly be programmed with your currently installed version of CODESYS. In such a case, please contact the technical support department of **ifm electronic gmbh**.

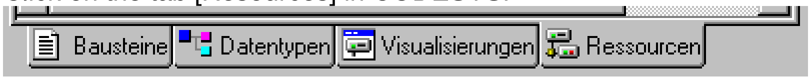
Activate the PLC configuration (e.g. CR0033)

15824

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

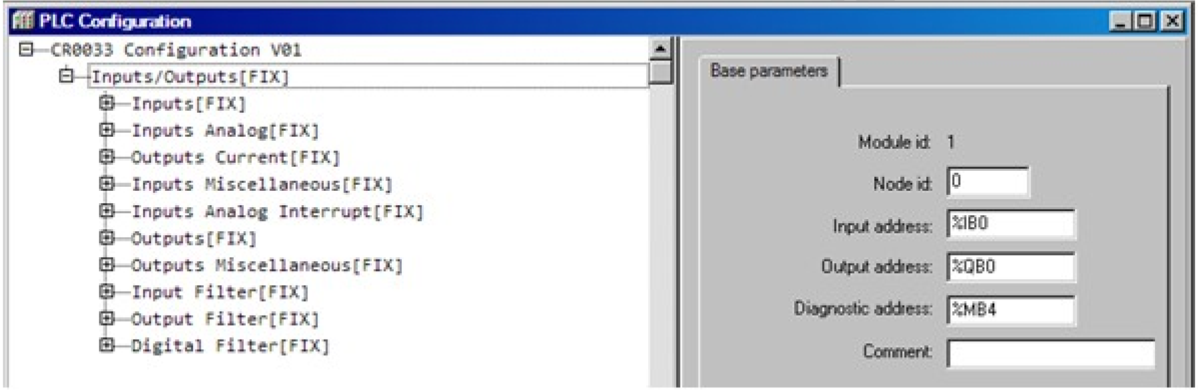
► The menu item [PLC Configuration] is reached via the tab [Resources].
Double-click on [PLC Configuration] to open the corresponding window.

► Click on the tab [Resources] in CODESYS:



► In the left column double-click on [PLC Configuration].

> Display of the current PLC configuration (example → following figure):

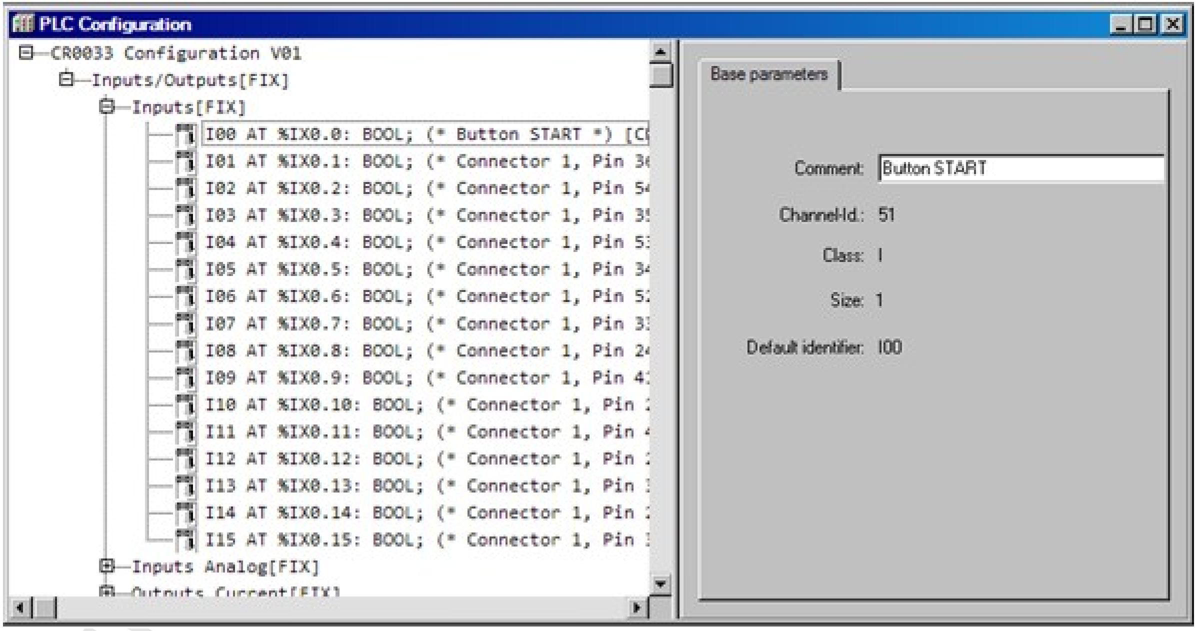


```
⊟—CR0033 Configuration V01
    ⊟—Inputs/Outputs[FIX]
        ⊞—Inputs[FIX]
        ⊞—Inputs Analog[FIX]
        ⊞—Outputs Current[FIX]
        ⊞—Inputs Miscellaneous[FIX]
        ⊞—Inputs Analog Interrupt[FIX]
        ⊞—Outputs[FIX]
        ⊞—Outputs Miscellaneous[FIX]
        ⊞—Input Filter[FIX]
        ⊞—Output Filter[FIX]
        ⊞—Digital Filter[FIX]
```

Module id: 1
Node id: 0
Input address: %IB0
Output address: %QB0
Diagnostic address: %MB4
Comment:

Based on the configuration the user can find the following in the program environment:

- 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 directly be designated symbolically (highly recommended!) in the window [PLC Configuration] (→ figure below) and are available in the whole project as [Global Variables].



```
⊟—CR0033 Configuration V01
    ⊟—Inputs/Outputs[FIX]
        ⊟—Inputs[FIX]
            I00 AT %IX0.0: BOOL; (* Button START *) [C
            I01 AT %IX0.1: BOOL; (* Connector 1, Pin 3
            I02 AT %IX0.2: BOOL; (* Connector 1, Pin 5
            I03 AT %IX0.3: BOOL; (* Connector 1, Pin 3
            I04 AT %IX0.4: BOOL; (* Connector 1, Pin 5
            I05 AT %IX0.5: BOOL; (* Connector 1, Pin 3
            I06 AT %IX0.6: BOOL; (* Connector 1, Pin 5
            I07 AT %IX0.7: BOOL; (* Connector 1, Pin 3
            I08 AT %IX0.8: BOOL; (* Connector 1, Pin 2
            I09 AT %IX0.9: BOOL; (* Connector 1, Pin 4
            I10 AT %IX0.10: BOOL; (* Connector 1, Pin
            I11 AT %IX0.11: BOOL; (* Connector 1, Pin
            I12 AT %IX0.12: BOOL; (* Connector 1, Pin
            I13 AT %IX0.13: BOOL; (* Connector 1, Pin
            I14 AT %IX0.14: BOOL; (* Connector 1, Pin
            I15 AT %IX0.15: BOOL; (* Connector 1, Pin
        ⊞—Inputs Analog[FIX]
        ⊞—Outputs Current[FIX]
```

Comment: Button START
Channel-Id.: 51
Class: I
Size: 1
Default identifier: I00

4.3.2 Set up the programming system via templates

13745

ifm offers ready-to-use templates (program templates), by means of which the programming system can be set up quickly, easily and completely.

970

ℹ When installing the **ecomat*mobile*** DVD "Software, tools and documentation", projects with templates have been stored in the program directory of your PC:
`…\ifm electronic\CoDeSys V…\Projects\Template_DVD_V…`

► 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.

4.4 Function configuration in general

3971

4.4.1 Configuration of the inputs and outputs (default setting)

2249

- All inputs and outputs are in the binary mode (plus switching!) when delivered.
- The diagnostic function is not active.
- The overload protection is active.

4.4.2 System variables

2252
13519
15576

All system variables (→ chapter *System flags* (→ page 232)) have defined addresses which cannot be shifted.

> To indicate and process a watchdog error or causes of a new start the system variable LAST_RESET is set.

> Indication of the selected I/O configuration via mode bytes

4.5 Function configuration of the inputs and outputs

Inhalt

20398
1394

For some devices of the **ecomat*mobile*** 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 (→ data sheet).

- 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 247).
- You find program blocks in the templates for each controller that are called during the 1st cycle after a restart of the controller. The networks programmed there are only used to assign a defined configuration to the input and outputs.

Only for the ExtendedController: The ExtendedController (or ExtendedSafetyController) is configured via the same system flags as the ClassicController (or SafetyController). If it is used in operating mode 2 (→ chapter *Operating modes of the ExtendedController* (→ page 49)), the designations of the inputs and outputs in the second controller are indicated by an appended "_E".

4.5.1 Configure inputs

Inhalt

20141
19567
3973

Valid operating modes → chapter *Possible operating modes inputs/outputs* (→ page 247)

- ► Configuration of each input is made via the application program:
  - FB *INPUT_ANALOG* (→ page 151) > input MODE
  - Configuration byte Ixx_MODE

20113

> For the extended side of the ExtendedControllers the FB name ends with '_E'.
> - The symbolic addresses of the inputs are Inn*_E*.
> - The symbolic addresses of the configuration variables are Inn*Mode_E*.
> - The symbolic addresses of the other flags also end with '*_E*'.

Safety instructions about Reed relays

7348

For use of non-electronic switches please note the following:

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

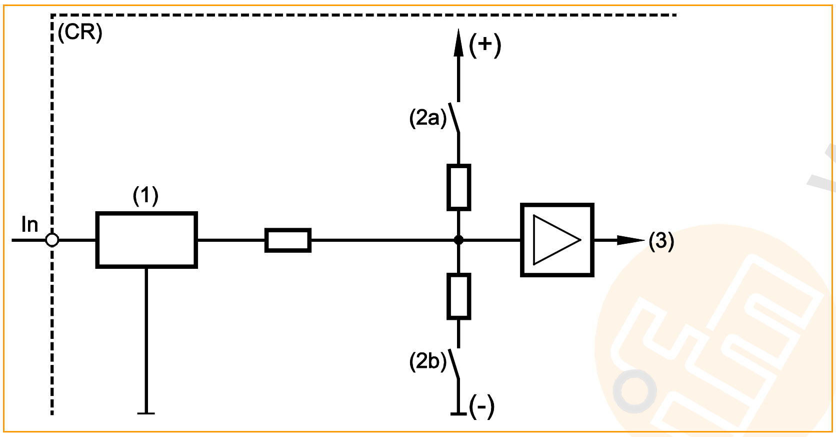
- ► **Remedy:** Install a series resistor for the Reed relay:
  Series resistor = max. input voltage / permissible current in the Reed relay
  Example: 32 V / 500 mA = 64 Ohm
- ► The series resistor must not exceed 5 % of the input resistance RE of the device input (→ data sheet). Otherwise, the signal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

Binary inputs

7345

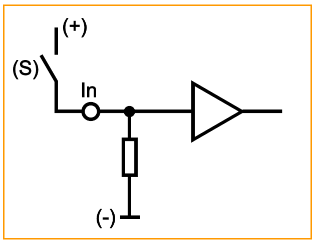
The binary input can be operated in following modes:
- binary input plus switching (BL) for positive sensor signal
- binary input minus switching (BH) for negative sensor signal

Depending on the device the binary inputs can configured differently. In addition to the protective mechanisms against interference, the binary 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



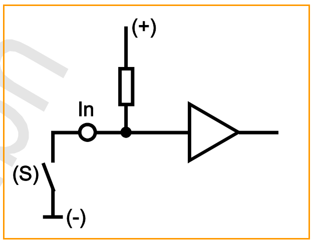
In = pin binary-input n
(CR) = device
(1) = input filter
(2a) = input minus switching
(2b) = input plus switching
(3) = voltage

Figure: basic circuit of binary input minus switching / plus switching for negative and positive sensor signals



In = pin binary input n
(S) = sensor

Basic circuit of binary input plus switching (BL)
for positive sensor signal:
Input = open ➩ signal = low (GND)



In = pin binary input n
(S) = sensor

Basic circuit of binary input minus switching (BH)
for negative sensor signal:
Input = open ➩ signal = high (supply)

Fast inputs

20142
1018

In addition, the **ecomat*mobile*** 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).

Appropriate function blocks are e.g.:

On FRQx inputs:

FAST_COUNT (→ page 159)	Counter block for fast input pulses
FREQUENCY (→ page 160)	Measures the frequency of the signal arriving at the selected channel

On CYLx inputs:

PERIOD (→ page 164)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 166)	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 [‰].
PHASE (→ page 168)	Reads a pair of channels with fast inputs and compares the phase position of the signals

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

20113

❗ 🛈 For the extended side of the ExtendedControllers the FB name ends with '_E'.
- The symbolic addresses of the inputs are Inn*_E*.
- The symbolic addresses of the configuration variables are Inn*Mode_E*.
- The symbolic addresses of the other flags also end with '*_E*'.

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...32 V

If in the operating mode "0...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.

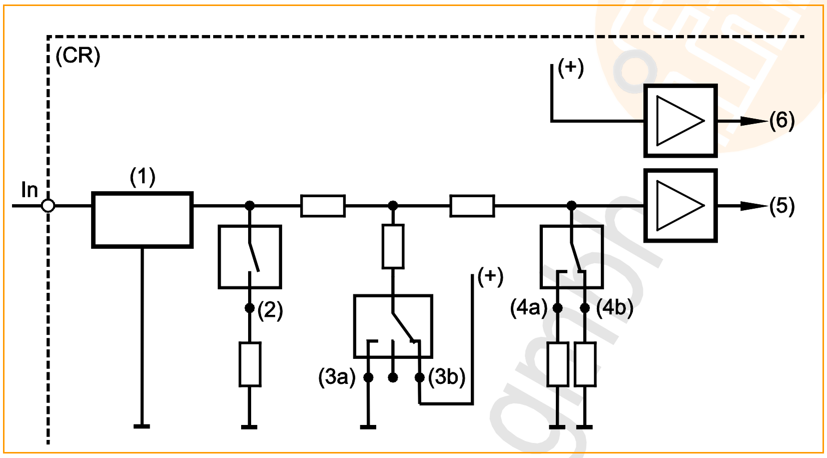
As an alternative, an analogue channel can also be evaluated binarily.

NOTE

In case of ratiometric measurement the connected sensors should be supplied with VBBS of the device. So, faulty measurements caused by offset voltage are avoided.

▶ Note the higher input resistances for binary evaluation.

8971



In = pin multifunction input n
(CR) = device
(1) = input filter
(2) = analogue current measuring
(3a) = binary-input plus switching
(3b) = binary-input minus switching
(4a) = analogue voltage measuring 0...10 V
(4b) = analogue voltage measuring 0...32 V
(5) = voltage
(6) = reference voltage

Figure: principle block diagram multifunction input

4.5.2 Configure outputs

Inhalt

20144
19568
3976

Valid operating modes → chapter *Possible operating modes inputs/outputs* (→ page 247)

► Configuration of each output is made via the application program:
indicate the load currents → FB *OUTPUT_CURRENT* (→ page 172)
PWM output: → FB *PWM1000* (→ page 181)
Configuration byte Qxx_MODE

20114

For the extended side of the ExtendedControllers the FB name ends with '_E'.
- The symbolic addresses of the outputs are Qnn*_E*.
- The symbolic addresses of the configuration variables are Qnn*Mode_E*.
- The symbolic addresses of the other flags also end with '*_E*'.

Binary and PWM outputs

1346

The following operating modes are possible for the device outputs (→ data sheet):

- binary output, plus switching (BH) with diagnostic function and protection
- analogue output with Pulse Width Modulation (PWM)
- PWM output pair H-bridge without diagnostic function

PWM outputs can be operated with and without current control function.

ℹ Current-controlled PWM outputs are mainly used for triggering proportional hydraulic functions.

14713

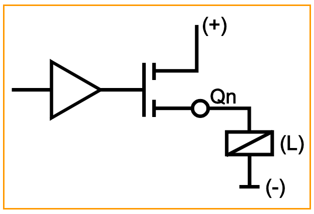
⚠ WARNING

Property damage or bodily injury possible due to malfunctions!

The following applies for outputs in PWM mode:

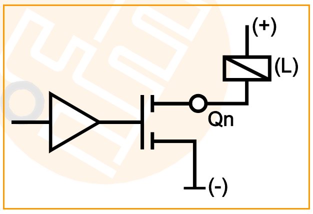
- there is no diagnostic function
- no ERROR flags are set
- the overload protection OUT_OVERLOAD_PROTECTION is NOT active

15450



Qn = pin output n
(L) = load

Basic circuit of output plus switching (BH)
for positive output signal



Qn = pin output n
(L) = load

Basic circuit of output minus switching (BL)
for negative output signal

13975

⚠ WARNING

Dangerous restart possible!
Risk of personal injury! Risk of material damage to the machine/plant!

If in case of a fault an output is switched off via the hardware, the logic state generated by the application program is not changed.

► Remedy:
- Reset the output logic in the application program!
- Remove the fault!
- Reset the outputs depending on the situation.

5035

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 runtime 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 VBBr via RELAY=FALSE (e.g. via ERROR=TRUE).

After fault elimination:

► Reset the error flag ERROR_..._Qx.

> The output relay re-enables the output group VBBr.

► New setting of the output or restart of the machine.

Output group Q1Q2 (Q10...13 / Q20...23)

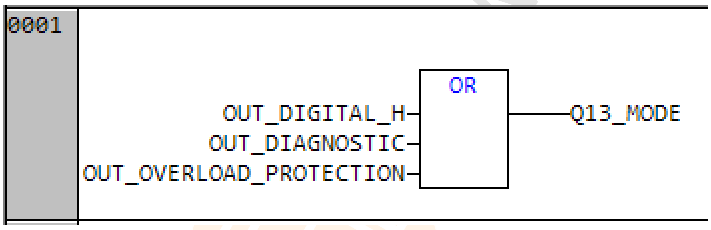
1378

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 172) 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).

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

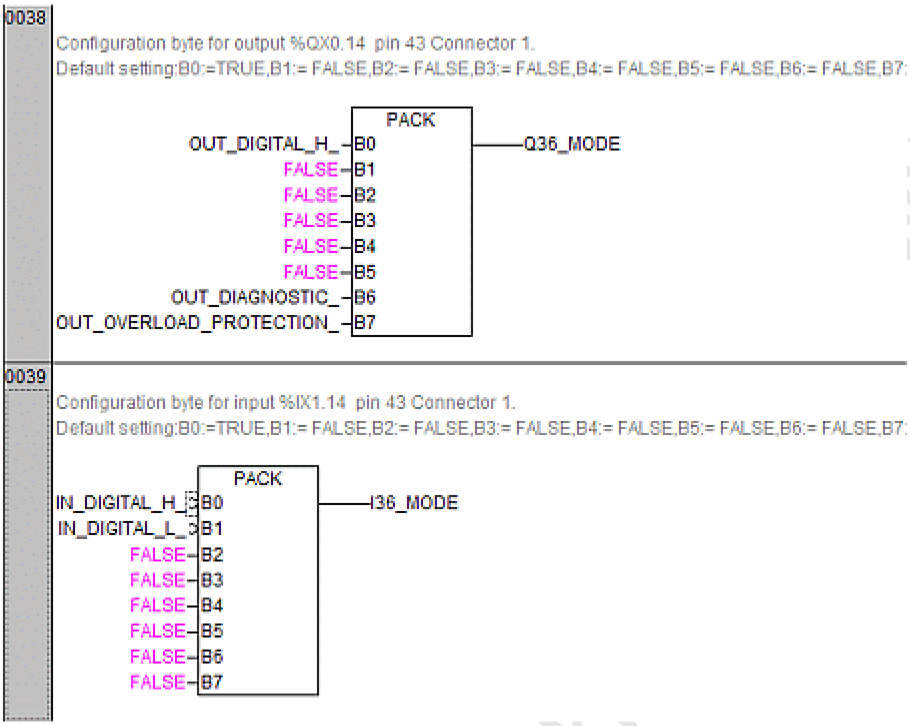
OUT_OVERLOAD_PROTECTION is not supported in the pure PWM mode.

Wire break and short circuit detection are active when ...
- the output is configured as "binary plus switching" (BH) AND
- the output is switched ON.

Output group Q3 (Q30...37)

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.



0038

Configuration byte for output %QX0.14 pin 43 Connector 1.
Default setting:B0:=TRUE,B1:= FALSE,B2:= FALSE,B3:= FALSE,B4:= FALSE,B5:= FALSE,B6:= FALSE,B7:

0039

Configuration byte for input %IX1.14 pin 43 Connector 1.
Default setting:B0:=TRUE,B1:= FALSE,B2:= FALSE,B3:= FALSE,B4:= FALSE,B5:= FALSE,B6:= FALSE,B7:

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.

❗ 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 (Q40...47)

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 Q41, Q42, Q45, Q46 can be switched to the mode OUT_DIGITAL_L in addition.

❗ 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 Q1Q2_E (Q10_E...13_E / Q20_E...23_E)

20145
20114

For the extended side of the ExtendedControllers the FB name ends with '_E'.
- The symbolic addresses of the outputs are Qnn_*E*.
- The symbolic addresses of the configuration variables are Qnn*Mode_E*.
- The symbolic addresses of the other flags also end with '_*E*'.

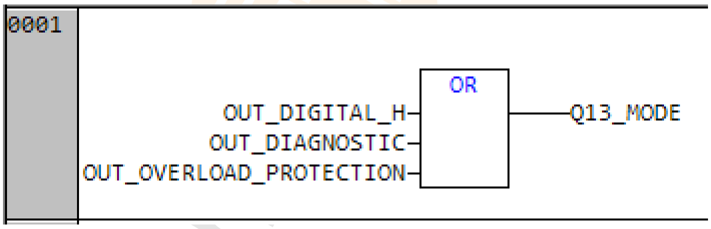
1378

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 172) 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).	 0001 OR OUT_DIGITAL_H OUT_DIAGNOSTIC OUT_OVERLOAD_PROTECTION Q13_MODE

NOTE

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

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 configured as "binary plus switching" (BH) AND
- the output is switched ON.

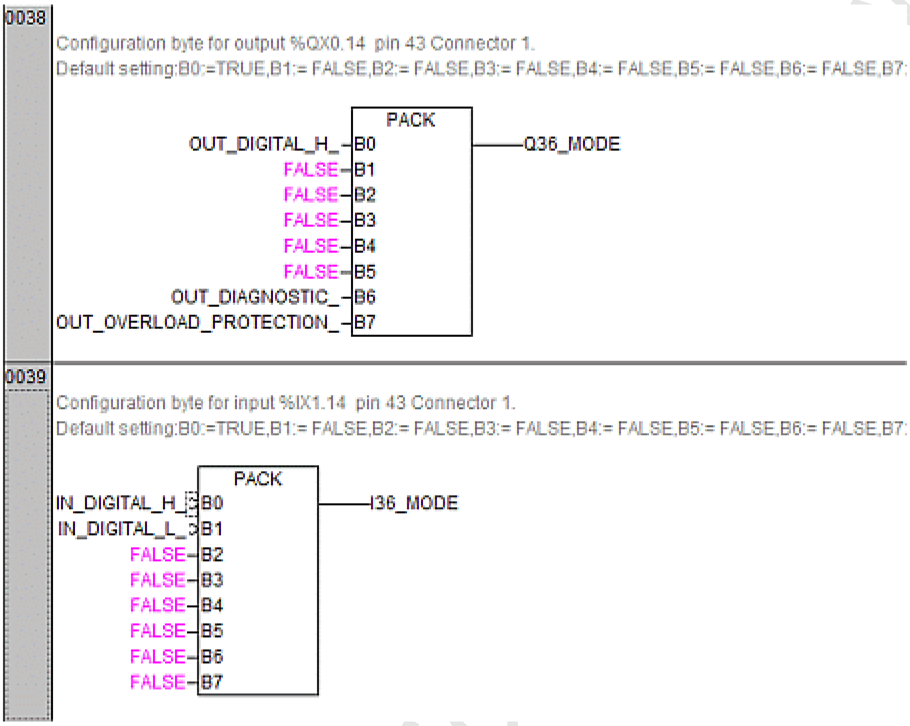
Output group Q3_E (Q30_E...37_E)

20146
20114

For the extended side of the ExtendedControllers the FB name ends with '_E'.
- The symbolic addresses of the outputs are Qnn_*E*.
- The symbolic addresses of the configuration variables are Qnn*Mode_E*.
- The symbolic addresses of the other flags also end with '_*E*'.

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.



0038

Configuration byte for output %QX0.14 pin 43 Connector 1.
Default setting:B0:=TRUE,B1:= FALSE,B2:= FALSE,B3:= FALSE,B4:= FALSE,B5:= FALSE,B6:= FALSE,B7:

0039

Configuration byte for input %IX1.14 pin 43 Connector 1.
Default setting:B0:=TRUE,B1:= FALSE,B2:= FALSE,B3:= FALSE,B4:= FALSE,B5:= FALSE,B6:= FALSE,B7:

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.

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_E (Q40_E...47_E)

20147
20114

! ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.
- The symbolic addresses of the outputs are Qnn*_E*.
- The symbolic addresses of the configuration variables are Qnn*Mode_E*.
- The symbolic addresses of the other flags also end with '*_E*'.

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 Q41, Q42, Q45, Q46 can be switched to the mode OUT_DIGITAL_L in addition.

! 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**.

4.6 Variables

Inhalt

3130

In this chapter you will learn more about how to handle variables.

4.6.1 Retain variables

3131

Variables declared as `RETAIN` generate remanent data. Retain variables keep the values saved in them when the device is switched on/off or when an online reset is made.

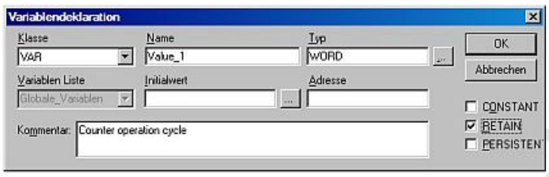
14166

Typical applications for retain variables are for example:
- operating hours which are counted up and retained while the machine is in operation,
- position values of incremental encoders,
- preset values entered in the monitor,
- machine parameters,

i.e. all variables whose values must not get lost when the device is switched off.

All variable types, also complex structures (e.g. timers), can be declared as retain.

► To do so, activate the control field [RETAIN] in the variable declaration (→ window).



Variablendeklaration

Klasse: VAR
Name: Value_1
Typ: WORD
Variablen Liste: Globale_Variablen
Initialwert:
Adresse:
Kommentar: Counter operation cycle
□ CONSTANT
☑ RETAIN
□ PERSISTENT

4.6.2 Network variables

9856

Global network variables are used for data exchange between controllers in the network. The values of global network variables are available to all CODESYS projects in the whole network if the variables are contained in their declaration lists.

► Integrate the following library/libraries into the CODESYS project:

- `3S_CANopenNetVar.lib`

5 ifm function elements

Inhalt

13586

All CODESYS function elements (FBs, PRGs, FUNs) are stored in libraries. Below you will find a list of all the **ifm** libraries you can use with this device.

This is followed by a description of the function elements, sorted by topic.

5.1 ifm libraries for the device CR0200

Inhalt

14235

Legend for ..._Vxxyyzz.LIB:

V	version
xx: 00...99	target version number
yy: 00...99	release number
zz: 00...99	patch number

Here you will find a list of the **ifm** function elements matching this device, sorted according to the CODESYS libraries.

5.1.1 Library ifm_CR0200_V06yyzz.LIB

20151

This is the device library. This **ifm** library contains the following function blocks:

Function element	Short description
ANALOG_RAW (→ page 150)	supplies non-standardised values of the analogue/digital converter for each individual input port
CAN2 (→ page 89)	Initialises CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CAN1_BAUDRATE (→ page 82)	Sets the transmission rate for the bus participant on CAN interface 1
CAN1_BAUDRATE_E	= *CAN1_BAUDRATE* (→ page 82) for the extended side
CAN1_DOWNLOADID (→ page 83)	Sets the download identifier for CAN interface 1
CAN1_DOWNLOADID_E	= *CAN1_DOWNLOADID* (→ page 83) for the extended side
CANx_ERRORHANDLER (→ page 90)	Executes a "manual" bus recovery on CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CAN1_EXT_ERRORHANDLER_E	= *CAN1_EXT_ERRORHANDLER* (→ page 85) for the extended side
CANx_EXT_RECEIVE_ALL (→ page 91)	CAN interface x: Configures all data receive objects and reads out the receive buffer of the data objects x = 2 = number of the CAN interface
CANx_RECEIVE (→ page 92)	CAN interface x: Configures a data receive object and reads out the receive buffer of the data object x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_RECEIVE_RANGE (→ page 94)	CAN interface x: Configures a sequence of data receive objects and reads out the receive buffer of the data objects x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_SDO_READ (→ page 116)	CAN interface x: Reads the SDO with the indicated indices from the node x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_SDO_WRITE (→ page 118)	CAN interface x: writes the SDO with the indicated indices to the node x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_TRANSMIT (→ page 96)	Transfers a CAN data object (message) to the CAN interface x for transmission at each call x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_TRANSMIT_E	= *CANx_TRANSMIT* (→ page 96) for the extended side x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CHECK_DATA (→ page 223)	Generates a checksum (CRC) for a configurable memory area and checks the data of the memory area for undesired changes
DELAY (→ page 200)	Delays the output of the input value by the time T (dead-time element)
FAST_COUNT (→ page 159)	Counter block for fast input pulses
FAST_COUNT_E	= *FAST_COUNT* (→ page 159) for the extended side
FLASHREAD (→ page 217)	Transfers different data types directly from the flash memory to the RAM
FLASHWRITE (→ page 218)	Writes different data types directly into the flash memory
FRAMREAD (→ page 219)	Transfers different data types directly from the FRAM memory to the RAM FRAM indicates here all kinds of non-volatile and fast memories.
FRAMWRITE (→ page 220)	Writes different data types directly into the FRAM memory FRAM indicates here all kinds of non-volatile and fast memories.
FREQUENCY (→ page 160)	Measures the frequency of the signal arriving at the selected channel
FREQUENCY_E	= *FREQUENCY* (→ page 160) for the extended side
GET_IDENTITY (→ page 225)	Reads the specific identifications stored in the device: • hardware name and hardware version of the device • name of the runtime system in the device • version and revision no. of the runtime system in the device • name of the application (has previously been saved by means of *SET_IDENTITY* (→ page 227))
GLR (→ page 201)	The synchro controller is a controller with PID characteristics

Function element	Short description
INC_ENCODER (→ page 161)	Up/down counter function for the evaluation of encoders
INC_ENCODER_E	= *INC_ENCODER* (→ page 161) for the extended side
INPUT_ANALOG (→ page 151)	Current and voltage measurement on the analogue input channel
INPUT_ANALOG_E	= *INPUT_ANALOG* (→ page 151) for the extended side
INPUT_CURRENT (→ page 153)	Current measurement on the analogue input channel
INPUT_CURRENT_E	= *INPUT_CURRENT* (→ page 153) for the extended side
INPUT_VOLTAGE (→ page 154)	Voltage measurement on the analogue input channel
INPUT_VOLTAGE_E	= *INPUT_VOLTAGE* (→ page 154) for the extended side
MEMCPY (→ page 221)	Writes and reads different data types directly in the memory
MEMORY_RETAIN_PARAM (→ page 215)	Determines the remanent data behaviour for various events
NORM (→ page 156)	Normalises a value [WORD] within defined limits to a value with new limits
OCC_TASK (→ page 170)	OCC = Output Current Control Current controller for a PWMi output channel Each instance of the function is called up in a cycle of 5 ms.
OUTPUT_CURRENT (→ page 172)	Measures the current (average via dither period) on an output channel
OUTPUT_CURRENT_E	= *OUTPUT_CURRENT* (→ page 172) for the extended side
OUTPUT_CURRENT_CONTROL (→ page 173)	Current controller for a PWMi output channel
PERIOD (→ page 164)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_E	= *PERIOD* (→ page 164) for the extended side
PERIOD_RATIO (→ page 166)	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 [‰].
PERIOD_RATIO_E	= *PERIOD_RATIO* (→ page 166) for the extended side
PHASE (→ page 168)	Reads a pair of channels with fast inputs and compares the phase position of the signals
PID1 (→ page 203)	PID controller
PID2 (→ page 205)	PID controller
PT1 (→ page 207)	Controlled system with first-order delay
PWM (→ page 175)	Initialises and configures a PWM-capable output channel Definition of the PWM frequency via RELOAD
PWM_E	= *PWM* (→ page 175) for the extended side
PWM100 (→ page 179)	Initialises and configures a PWM-capable output channel Indicate PWM frequency in [Hz] Indicate mark-to-space ratio in steps of 1 %
PWM100_E	= *PWM100* (→ page 179) for the extended side
PWM1000 (→ page 181)	Initialises and configures a PWM-capable output channel the mark-to-space ratio can be indicated in steps of 1 ‰
PWM1000_E	= *PWM1000* (→ page 181) for the extended side
SERIAL_PENDING (→ page 133)	Determines the number of data bytes stored in the serial receive buffer
SERIAL_PENDING_E	= *SERIAL_PENDING* (→ page 133) for the extended side
SERIAL_RX (→ page 134)	Reads a received data byte from the serial receive buffer at each call
SERIAL_RX_E	= *SERIAL_RX* (→ page 134) for the extended side
SERIAL_SETUP (→ page 135)	Initialises the serial RS232 interface
SERIAL_SETUP_E	= *SERIAL_SETUP* (→ page 135) for the extended side
SERIAL_TX (→ page 136)	Transmits one data byte via the serial RS232 interface
SERIAL_TX_E	= *SERIAL_TX* (→ page 136) for the extended side
SET_DEBUG (→ page 226)	organises the DEBUG mode or the monitoring mode (depending on the TEST input)

Function element	Short description
SET_IDENTITY (→ page 227)	Sets an application-specific program identification
SET_INTERRUPT_I (→ page 144)	Conditional execution of a program part after an interrupt request via a defined input channel
SET_INTERRUPT_XMS (→ page 147)	Conditional execution of a program part at an interval of x milliseconds
SET_PASSWORD (→ page 228)	Sets a user password for access control to program and memory upload
SOFTRESET (→ page 209)	leads to a complete reboot of the device
SSC_RECEIVE (→ page 139)	receives data via the internal SSC interface from the slave
SSC_SET_MASTER (→ page 140)	activates the master/slave operation
SSC_TRANSMIT (→ page 142)	transmits data to the slave via the internal SSC interface
TIMER_READ (→ page 211)	Reads out the current system time in [ms] Max. value = 49d 17h 2min 47s 295ms
TIMER_READ_US (→ page 212)	Reads out the current system time in [µs] Max. value = 1h 11min 34s 967ms 295µs

5.1.2 Library ifm_CR0200_DUMMY_vxxyyzz.LIB

20168

This **ifm** library contains the following function blocks:

Function element	Short description
DUMMY (→ page 138)	for master-slave operation of the ExtendedController: initialises the slave side

5.1.3 Library ifm_CR0200_CANopenMaster_V04yynn.LIB

18714

This library contains the function blocks for operation of the device as a CANopen master.
The library is only permissible for the 1st CAN interface.

x = 1 = number of the CAN interface

This **ifm** library contains the following function blocks:

Function element	Short description
CANx_MASTER_EMCY_HANDLER (→ page 98)	Handles the device-specific error status of the CANopen master on CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_MASTER_SEND_EMERGENCY (→ page 99)	Sends application-specific error status of the CANopen master on CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_MASTER_STATUS (→ page 101)	Status indication on CAN interface x of the device used as CANopen master x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

5.1.4 Library ifm_CR0200_CANopenSlave_V04yynn.LIB

18719

This library contains the function blocks for operation of the device as a CANopen slave.
The library is only permissible for the 1st CAN interface.

x = 1 = number of the CAN interface

This **ifm** library contains the following function blocks:

Function element	Short description
CANx_SLAVE_EMCY_HANDLER (→ page 108)	Handles the device-specific error status of the CANopen slave on CAN interface x: • error register (index 0x1001) and • error field (index 0x1003) of the CANopen object directory x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_SLAVE_NODEID (→ page 109)	Enables setting of the node ID of a CANopen slave on CAN interface x at runtime of the application program x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_SLAVE_SEND_EMERGENCY (→ page 110)	Sends application-specific error status of the CANopen slave on CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_SLAVE_STATUS (→ page 112)	Shows the status of the device used as CANopen slave on CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

5.1.5 Library ifm_CAN1_EXT_Vxxyyzz.LIB

18732

This library contains the complementary POUs for engine control on the 1st CAN interface.
The library is only permissible for the 1st CAN interface.

This **ifm** library contains the following function blocks:

Function element	Short description
CAN1_EXT (→ page 84)	Initialises CAN interface 1 also for the exended mode Set the mode and baud rate
CAN1_EXT_ERRORHANDLER (→ page 85)	Executes a "manual" bus recovery on CAN interface 1
CAN1_EXT_RECEIVE (→ page 86)	CAN interface 1: Configures a data receive object and reads out the receive buffer of the data object
CANx_EXT_RECEIVE_ALL (→ page 91)	CAN interface x: Configures all data receive objects and reads out the receive buffer of the data objects x = 1 = number of the CAN interface
CAN1_EXT_TRANSMIT (→ page 88)	Transfers a CAN data object (message) to CAN interface 1 for transmission at each call

5.1.6 Library ifm_J1939_x_Vxxyyzz.LIB

18722

This library contains the function blocks for engine control.
x = 1...2 = number of the CAN interface

This **ifm** library contains the following function blocks:

Function element	Short description
J1939_x (→ page 121)	CAN interface x: protocol handler for the communication profile SAE J1939 x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
J1939_x_GLOBAL_REQUEST (→ page 122)	CAN interface x: handles global requesting and receipt of data from the J1939 network participants x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
J1939_x_RECEIVE (→ page 124)	CAN interface x: Receives a single message or a message block x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
J1939_x_RESPONSE (→ page 126)	CAN interface x: handles the automatic response to a request message x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
J1939_x_SPECIFIC_REQUEST (→ page 128)	CAN interface x: automatic requesting of individual messages from a specific J1939 network participant x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
J1939_x_TRANSMIT (→ page 130)	CAN interface x: sends individual messages or message blocks x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

5.1.7 Library ifm_hydraulic_16bitOS05_Vxxyyzz.LIB

19535

This library contains function blocks for hydraulic controls.

This **ifm** library contains the following function blocks:

Function element	Short description
CONTROL_OCC (→ page 184)	OCC = Output Current Control Scales the input value [WORD] to an indicated current range
JOYSTICK_0 (→ page 187)	Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0... 1000
JOYSTICK_1 (→ page 190)	Scales signals [INT] from a joystick D standardised to 0... 1000
JOYSTICK_2 (→ page 194)	Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation
NORM_HYDRAULIC (→ page 197)	Normalises a value [DINT] within defined limits to a value with new limits

5.2 ifm function elements for the device CR0200

Inhalt

13988
3826

Here you will find the description of the **ifm** function elements suitable for this device, sorted by topic.

5.2.1 Function elements: CAN layer 2

Inhalt

13754

Here, the CAN function blocks (layer 2) for use in the application program are described.

CAN1_BAUDRATE

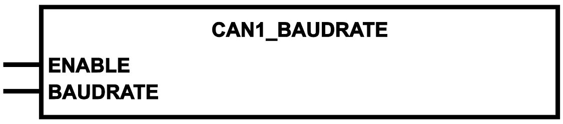
20180

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'.

Symbol in CODESYS:



Description

20181

CAN1_BAUDRATE sets the transmission rate for the bus participant.

The FB is used to set the transmission rate for the device. To do so, the corresponding value in Kbits/s is entered at the input BAUDRATE.

> **NOTE**
>
> The new baud rate will become effective on RESET (voltage OFF/ON or soft reset).
> 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 (in the 1st cycle): Adopt and activate parameters else: this function is not executed
BAUDRATE	WORD := 125	Baud rate [kbits/s] valid = 20, 50, 100, 125, 250, 500, 1000

CAN1_DOWNLOADID

20183

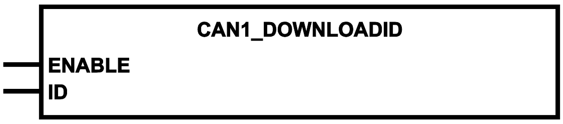
= CAN1 download ID

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

Symbol in CODESYS:



Description

20184

CAN1_DOWNLOADID sets the download identifier for the first CAN interface.

The FB can be used to set the communication identifier for program download and debugging. The new value is entered when the input ENABLE is set to TRUE.

❗ The new value will become effective on RESET (voltage OFF/ON or soft reset).

Parameters of the inputs

649

Parameter	Data type	Description
ENABLE	BOOL	TRUE (in the 1st cycle): Adopt and activate parameters else: this function is not executed

ID	BYTE	Set download ID of CAN interface x
		x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
		allowed = 1...127
		preset = 127 - (x-1)

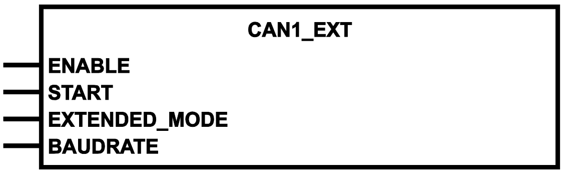
CAN1_EXT

4192

Unit type = function block (FB)

Unit is contained in the library `ifm_CAN1_EXT_Vxxyyzz.LIB`

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* 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.

> ! The FB must be executed **before** CAN1_EXT_... .

Parameters of the inputs

4334

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
START	BOOL	TRUE (in the 1st cycle): Start CAN protocol at CAN interface x FALSE: during further processing of the program
EXTENDED_MODE	BOOL := FALSE	TRUE: identifier of the CAN interface operates with 29 bits FALSE: identifier of the CAN interface operates with 11 bits
BAUDRATE	WORD := 125	Baud rate [Kbits/s] Permissible = 50, 100, 125, 250, 500, 800, 1000

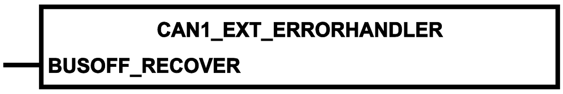
CAN1_EXT_ERRORHANDLER

4195

Unit type = function block (FB)

Unit is contained in the library `ifm_CAN1_EXT_Vxxyyzz.LIB`

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.

❗ 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	TRUE (only 1 cycle): > remedy 'bus off' status > reboot of the CAN interfacex FALSE: function element is not executed

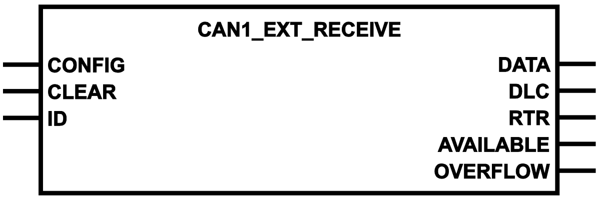
CAN1_EXT_RECEIVE

4302

Unit type = function block (FB)

Unit is contained in the library `ifm_CAN1_EXT_Vxxyyzz.LIB`

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.

> ❗ If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with *CAN1_EXT* (→ page 84).

Parameters of the inputs

2172

Parameter	Data type	Description
CONFIG	BOOL	TRUE (in the 1st cycle): configure data object FALSE: during further processing of the program
CLEAR	BOOL	TRUE: delete receive buffer FALSE: function element is not executed
ID	DWORD	Number of the data object identifier: normal frame (2^{11} IDs): 0...2 047 = 0x0000 0000...0x0000 07FF extended Frame (2^{29} IDs): 0...536 870 911 = 0x0000 0000...0x1FFF FFFF

Parameters of the outputs

19810

Parameter	Data type	Description
DATA	ARRAY [0..7] OF BYTE	received data, (1...8 bytes)
DLC	BYTE	Number of bytes received in the DATA array with RDO allowed: 0...8
RTR	BOOL = FALSE	Received message was a Remote Transmission Request (wird hier nicht unterstützt)
AVAILABLE	BYTE	Number of remaining data bytes allowed = 0...16 0 = no valid data available
OVERFLOW	BOOL	TRUE: Overflow of the data buffer ⇨ loss of data! FALSE: Data buffer is without data loss

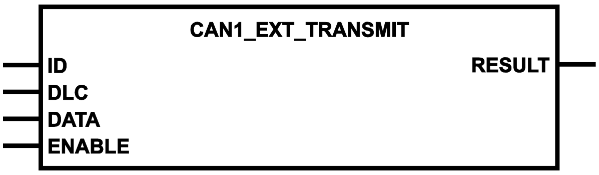
CAN1_EXT_TRANSMIT

4307

Unit type = function block (FB)

Unit is contained in the library `ifm_CAN1_EXT_Vxxyyzz.LIB`

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.

❗ If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with *CAN1_EXT* (→ page 84).

Parameters of the inputs

4380

Parameter	Data type	Description
ID	DWORD	Number of the data object identifier: normal frame (2^{11} IDs): 0...2 047 = 0x0000 0000...0x0000 07FF extended Frame (2^{29} IDs): 0...536 870 911 = 0x0000 0000...0x1FFF FFFF
DLC	BYTE	Number of bytes to be transmitted from the DATA array with RDO allowed: 0...8
DATA	ARRAY [0..7] OF BYTE	data to be sent (1...8 bytes)
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified

Parameters of the outputs

614

Parameter	Data type	Description
RESULT	BOOL	TRUE (only for 1 cycle): Function block accepted transmit order FALSE: Transmit order was not accepted

CAN2

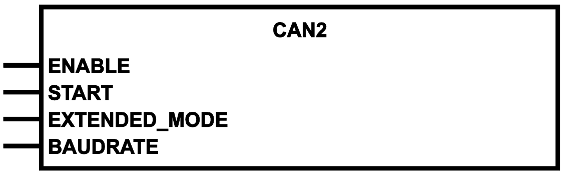
20186

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

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* and *Use of the CAN interface to ISO 11992*, among others, are available. The FBs to ISO 11992 are only available in the CR2501 on the 2nd CAN interface.

❗ The FB must be executed **before** CAN2... .

Parameters of the inputs

643

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
START	BOOL	TRUE (in the 1st cycle): Start CAN protocol at CAN interface x FALSE: during further processing of the program
EXTENDED_MODE	BOOL := FALSE	TRUE: identifier of the CAN interface operates with 29 bits FALSE: identifier of the CAN interface operates with 11 bits
BAUDRATE	WORD := 125	Baud rate [Kbits/s] Permissible = 50, 100, 125, 250, 500, 800, 1000

CANx_ERRORHANDLER

20187

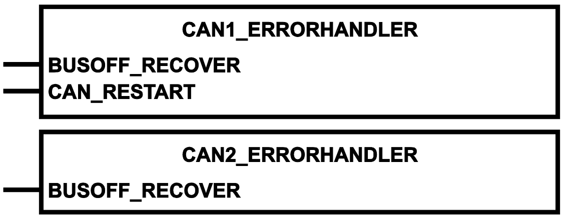
x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'.

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*

NOTE

In principle, CAN2 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 > reboot of the CAN interfacex FALSE: function element is not executed
CAN_RESTART	BOOL	TRUE (only 1 cycle): completely reinitialise CAN interface FALSE: function element is not executed

CANx_EXT_RECEIVE_ALL

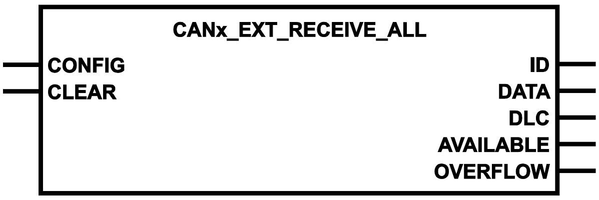
4183

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CANx_EXT_Vxxyyzz.LIB`

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 (in the 1st cycle): configure data object FALSE: during further processing of the program
CLEAR	BOOL	TRUE: delete receive buffer FALSE: function element is not executed

Parameters of the outputs

2292

Parameter	Data type	Description
ID	DWORD	Number of the data object identifier
DATA	ARRAY [0..7] OF BYTE	received data, (1...8 bytes)
DLC	BYTE	Number of bytes received in the DATA array with SRDO allowed: 0...8
AVAILABLE	BYTE	Number of remaining data bytes allowed = 0...16 0 = no valid data available
OVERFLOW	BOOL	TRUE: Overflow of the data buffer ➩ loss of data! FALSE: Data buffer is without data loss

CANx_RECEIVE

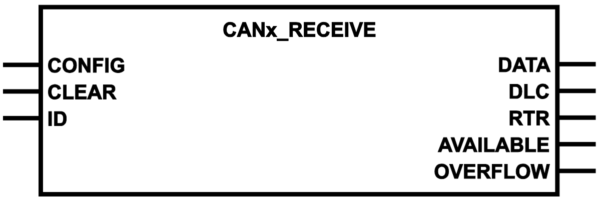
627

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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.

! If CAN2_RECEIVE is to be used, the second CAN interface must be initialised first using *CAN2*.

Parameters of the inputs

631

Parameter	Data type	Description
CONFIG	BOOL	TRUE (in the 1st cycle): configure data object FALSE: during further processing of the program
CLEAR	BOOL	TRUE: delete receive buffer FALSE: function element is not executed
ID	WORD	number of the data object identifier permissible values = 0...2 047

Parameters of the outputs

19810

Parameter	Data type	Description
DATA	ARRAY [0..7] OF BYTE	received data, (1...8 bytes)
DLC	BYTE	Number of bytes received in the DATA array with RDO allowed: 0...8
RTR	BOOL = FALSE	Received message was a Remote Transmission Request (wird hier nicht unterstützt)
AVAILABLE	BYTE	Number of remaining data bytes allowed = 0...16 0 = no valid data available
OVERFLOW	BOOL	TRUE: Overflow of the data buffer ⇨ loss of data! FALSE: Data buffer is without data loss

CANx_RECEIVE_RANGE

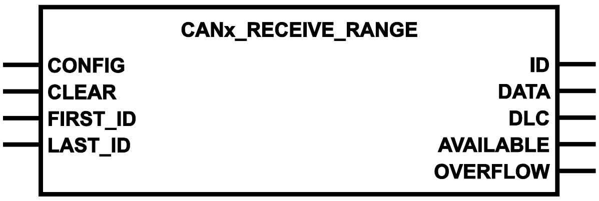
4179

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB` (xx ≥ 05)

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: Initialisation of CANx_RECEIVE_RANGE in 4 cycles* (→ page 95).

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 92) 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 (in the 1st cycle): configure data object FALSE: during further processing of the program
CLEAR	BOOL	TRUE: delete receive buffer FALSE: function element 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...536 870 911 (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...536 870 911 (2^{29}) LAST_ID has to be bigger than FIRST_ID!

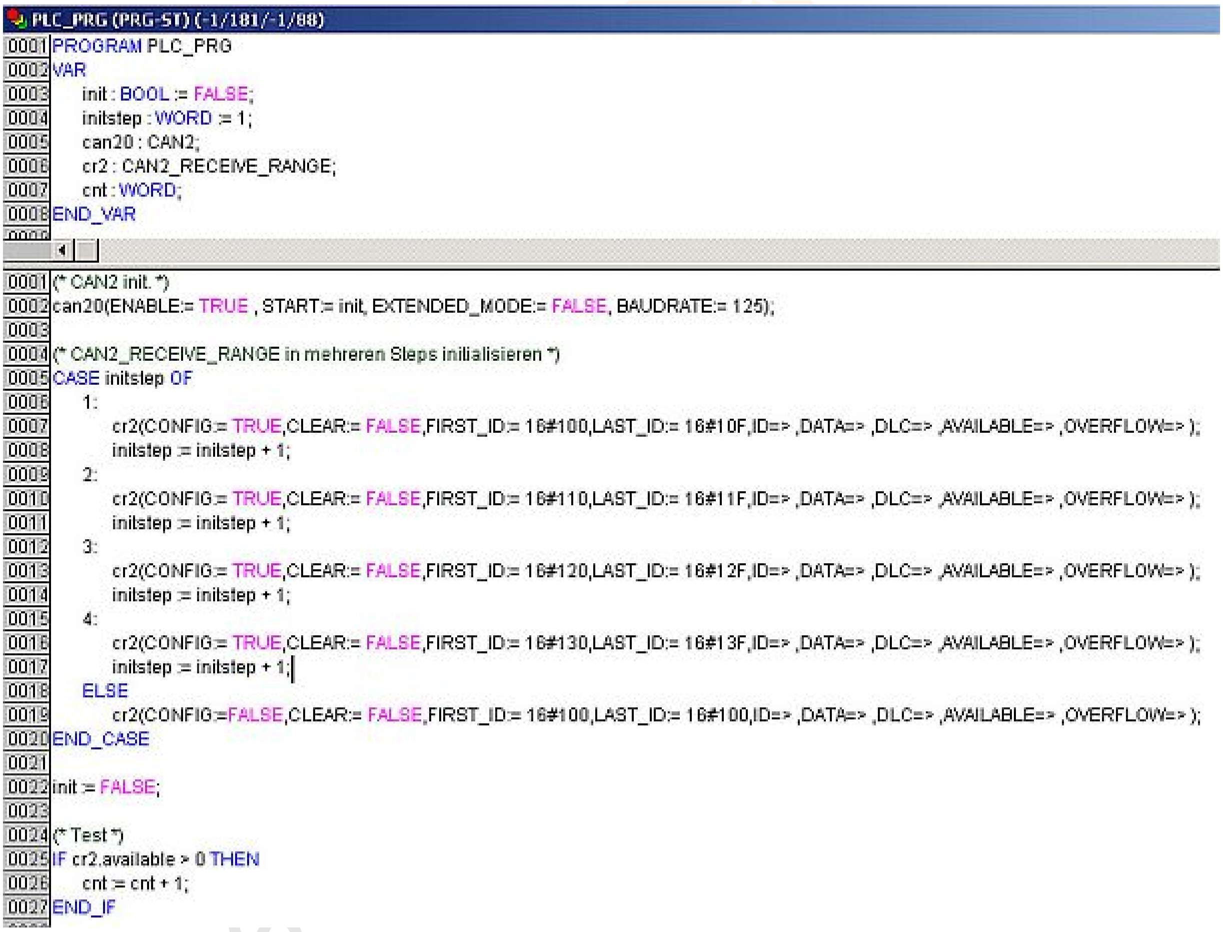
Parameters of the outputs

4381

Parameter	Data type	Description
ID	CAN1: WORD CAN2: DWORD	ID of the transmitted data object
DATA	ARRAY [0..7] OF BYTE	received data, (1...8 bytes)
DLC	BYTE	Number of bytes received in the DATA array with RDO allowed: 0...8
AVAILABLE	BYTE	Number of remaining data bytes allowed = 0...16 0 = no valid data available
OVERFLOW	BOOL	TRUE: Overflow of the data buffer ⇨ loss of data! FALSE: Data buffer is without data loss

Example: Initialisation of CANx_RECEIVE_RANGE in 4 cycles

2294



PLC_PRG (PRG-ST) (-1/181/-1/88)

```
PROGRAM PLC_PRG
VAR
    init : BOOL := FALSE;
    initstep : WORD := 1;
    can20 : CAN2;
    cr2 : CAN2_RECEIVE_RANGE;
    cnt : WORD;
END_VAR
```

```
(* CAN2 init. *)
can20(ENABLE:= TRUE , START:= init, EXTENDED_MODE:= FALSE, BAUDRATE:= 125);

(* CAN2_RECEIVE_RANGE in mehreren Steps initialisieren *)
CASE initstep OF
    1:
        cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#100,LAST_ID:= 16#10F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
        initstep := initstep + 1;
    2:
        cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#110,LAST_ID:= 16#11F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
        initstep := initstep + 1;
    3:
        cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#120,LAST_ID:= 16#12F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
        initstep := initstep + 1;
    4:
        cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#130,LAST_ID:= 16#13F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
        initstep := initstep + 1;
    ELSE
        cr2(CONFIG:=FALSE,CLEAR:= FALSE,FIRST_ID:= 16#100,LAST_ID:= 16#100,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
END_CASE

init := FALSE;

(* Test *)
IF cr2.available > 0 THEN
    cnt := cnt + 1;
END_IF
```

CANx_TRANSMIT

20190

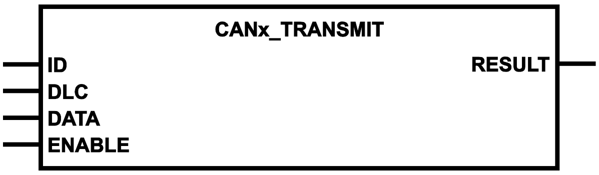
x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

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.

❗ If CAN2_TRANSMIT is to be used, the second CAN interface must be initialised first using *CAN2*.

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 DATA array with RDO allowed: 0...8
DATA	ARRAY [0..7] OF BYTE	data to be sent (1...8 bytes)
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified

Parameters of the outputs

614

Parameter	Data type	Description
RESULT	BOOL	TRUE (only for 1 cycle): Function block accepted transmit order FALSE: Transmit order was not accepted

5.2.2 Function elements: CANopen master

Inhalt

1870

ifm electronic provides a number of FBs for the CANopen master which will be explained below.

CANx_MASTER_EMCY_HANDLER

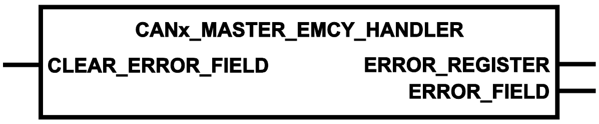
13192

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenMaster_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

2009

CANx_MASTER_EMCY_HANDLER manages 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.

The current values from the error register (index 0x1001/01) and error field (index 0x1003/0-5) of the CANopen object directory can be read via the FB.

> ! 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 99).

Parameters of the inputs

2010

Parameter	Data type	Description
CLEAR_ERROR_FIELD	BOOL	FALSE ⇨ TRUE (edge): • transmit content of ERROR_FIELD to function block output • delete content of ERROR_FIELD in object directory else: this function is not executed

Parameters of the outputs

2011

Parameter	Data type	Description
ERROR_REGISTER	BYTE	Shows content of OBV index 0x1001 (error register)
ERROR_FIELD	ARRAY [0..5] OF WORD	Shows the content of the OBV index 0x1003 (error field) ERROR_FIELD[0]: number of stored errors ERROR_FIELD[1...5]: Stored errors, the most recent error is shown on index [1]

CANx_MASTER_SEND_EMERGENCY

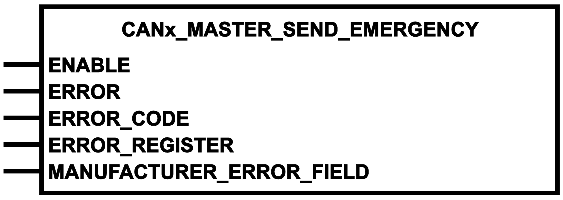
13195

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenMaster_Vxxyyzz.LIB`

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.

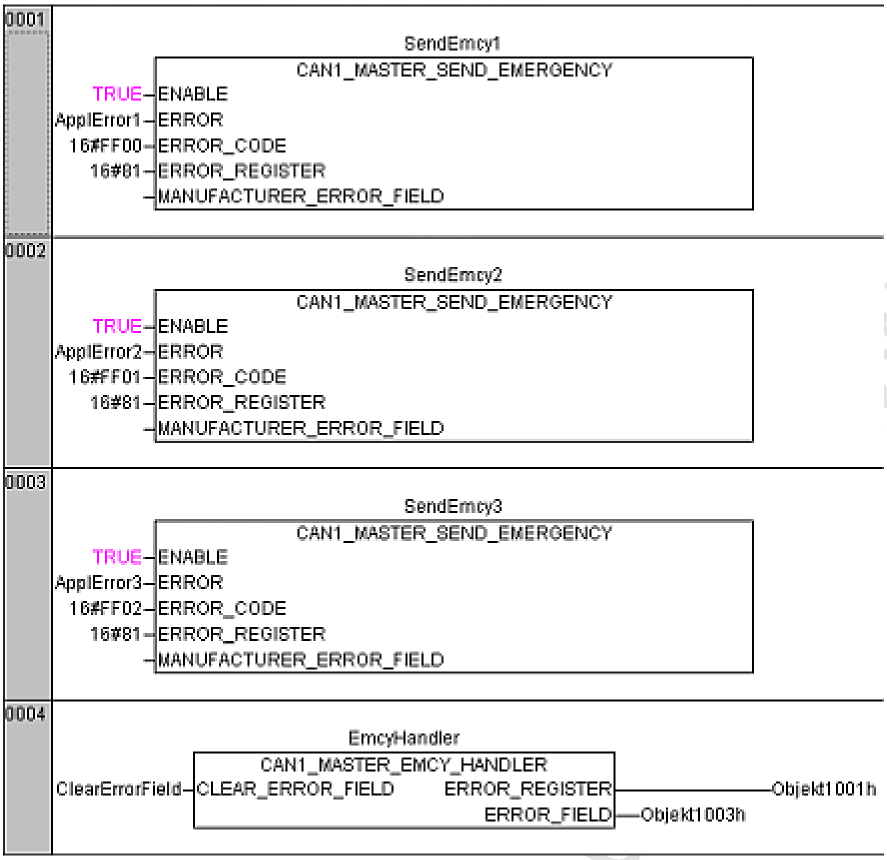
> ! If application-specific error messages are to be stored in the object directory, *CANx_MASTER_EMCY_HANDLER* (→ page 98) must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY.

Parameters of the inputs

2016

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
ERROR	BOOL	Using this input, the information whether the error associated to the configured error code is currently present is transmitted. FALSE ⇨ TRUE (edge): sends the next error code if input was not TRUE in the last second TRUE ⇨ FALSE (edge) AND the fault is no longer indicated: after a delay of approx. 1 s: > zero error message is sent else: this function is not executed
ERROR_CODE	WORD	The error code provides detailed information about the detected error. The values should be entered according to the CANopen specification.
ERROR_REGISTER	BYTE	ERROR_REGISTER indicates the error type. The value indicated here is linked by a bit-by-bit OR operation with all the other error messages that are currently active. The resulting value is written into the error register (index 1001_{16}/00) and transmitted with the EMCY message. 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



In this example 3 error messages will be generated subsequently:

1. ApplError1, Code = 0xFF00 in the error register 0x81
2. ApplError2, Code = 0xFF01 in the error register 0x81
3. ApplError3, Code = 0xFF02 in the error register 0x81

CAN1_MASTER_EMCY_HANDLER sends the error messages to the error register "Object 0x1001" in the error array "Object 0x1003".

CANx_MASTER_STATUS

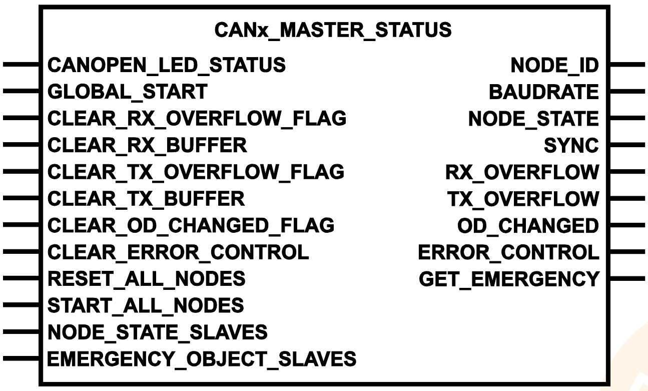
2021

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenMaster_Vxxyyzz.LIB`

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. Further possibilities:

- monitoring the network status
- monitoring the status of the connected slaves
- resetting or starting the slaves in the network.

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

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 (⇨ state OPERATIONAL). FALSE: The connected network participants are started one after the other.
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE ⇨ TRUE (edge): Clear error flag RX_OVERFLOW else: this function is not executed
CLEAR_RX_BUFFER	BOOL	FALSE ⇨ TRUE (edge): Delete data in the receive buffer else: this function is not executed
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE ⇨ TRUE (edge): Clear error flag TX_OVERFLOW else: this function is not executed
CLEAR_TX_BUFFER	BOOL	FALSE ⇨ TRUE (edge): Delete data in the transmit buffer else: this function is not executed
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE ⇨ TRUE (edge): Delete flag OD_CHANGED else: this function is not executed
CLEAR_ERROR_CONTROL	BOOL	FALSE ⇨ TRUE (edge): Delete the guard error list (ERROR_CONTROL) else: this function is not executed
RESET_ALL_NODES	BOOL	FALSE ⇨ TRUE (edge): All connected network participants (slaves) are reset via NMT command else: this function is not executed
START_ALL_NODES	BOOL	FALSE ⇨ TRUE (edge): All connected network participants (slaves) are started via NMT command else: this function is not executed
NODE_STATE_SLAVES	DWORD	Shows states of all network nodes. Example code → chapter *Example: CANx_MASTER_STATUS* (→ page 105)
EMERGENCY_OBJECT_SLAVES	DWORD	Shows the last error messages of all network nodes. ℹ → chapter *Access to the structures at runtime of the application* (→ page 106)

Parameters of the outputs

2029

Parameter	Data type	Description
NODE_ID	BYTE	current node ID of the CANopen master
BAUDRATE	WORD	current baudrate of the CANopen master in [kBaud]
NODE_STATE	INT	Current status of CANopen master
SYNC	BOOL	SYNC signal of the CANopen master TRUE: In the last cycle a SYNC signal was sent FALSE: In the last cycle no SYNC signal was sent
RX_OVERFLOW	BOOL	TRUE: Error: receive buffer overflow FALSE: no overflow
TX_OVERFLOW	BOOL	TRUE: Error: transmission buffer overflow FALSE: no overflow
OD_CHANGED	BOOL	TRUE: Data in the object directory of the CANopen master have been changed FALSE: no data change
ERROR_CONTROL	ARRAY [0..7] OF BYTE	The array contains the list (max. 8) of missing network nodes (guard or heartbeat error) → chapter *Access to the structures at runtime of the application* (→ page 106)
GET_EMERGENCY	STRUCT CANx_EMERGENY_MESSAGE	At the output the data for the structure CANx_EMERGENCY_MESSAGE are available. The last received EMCY message in the CANopen network is always displayed. To obtain a list of all occurred errors, the array "EMERGENCY_OBJECT_SLAVES" must be evaluated.
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]* of the master depending on the set time [Com. Cycle Period].
RX_OVERFLOW	BOOL	error flag "receive buffer overflow"

Parameters of internal structures

2030

Below are the structures of the arrays used in this FB.

Parameter	Data type	Description
CANx_EMERGENY_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 `ifm_CR0200_CANopenMaster_Vxxyyzz.LIB`.
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 `ifm_CR0200_CANopenMaster_Vxxyyzz.LIB`.

Using the controller CR0020 as an example the following code fragments show the use of the FB CANx_MASTER_STATUS.

Example: CANx_MASTER_STATUS

2031

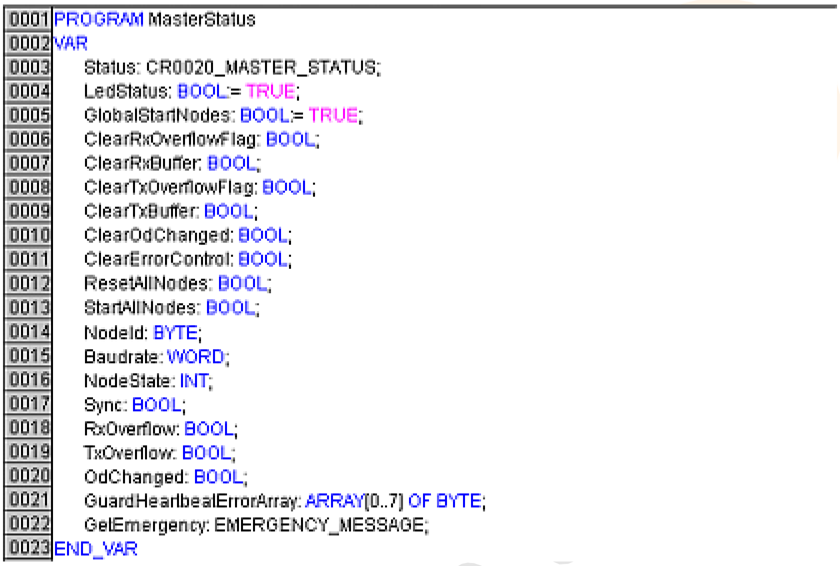
Slave information

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.

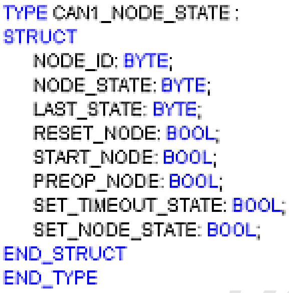
> ! The numbers of the array elements do **not** correspond to the node ID. The identifier can be read from the corresponding structure under NODE_ID.



```
PROGRAM MasterStatus
VAR
    Status: CR0020_MASTER_STATUS;
    LedStatus: BOOL:= TRUE;
    GlobalStartNodes: BOOL:= TRUE;
    ClearRxOverflowFlag: BOOL;
    ClearRxBuffer: BOOL;
    ClearTxOverflowFlag: BOOL;
    ClearTxBuffer: BOOL;
    ClearOdChanged: BOOL;
    ClearErrorControl: BOOL;
    ResetAllNodes: BOOL;
    StartAllNodes: BOOL;
    NodeId: BYTE;
    Baudrate: WORD;
    NodeState: INT;
    Sync: BOOL;
    RxOverflow: BOOL;
    TxOverflow: BOOL;
    OdChanged: BOOL;
    GuardHeartbeatErrorArray: ARRAY[0..7] OF BYTE;
    GetEmergency: EMERGENCY_MESSAGE;
END_VAR
```

Structure node status

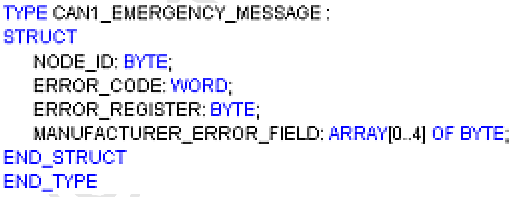
2034



```
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

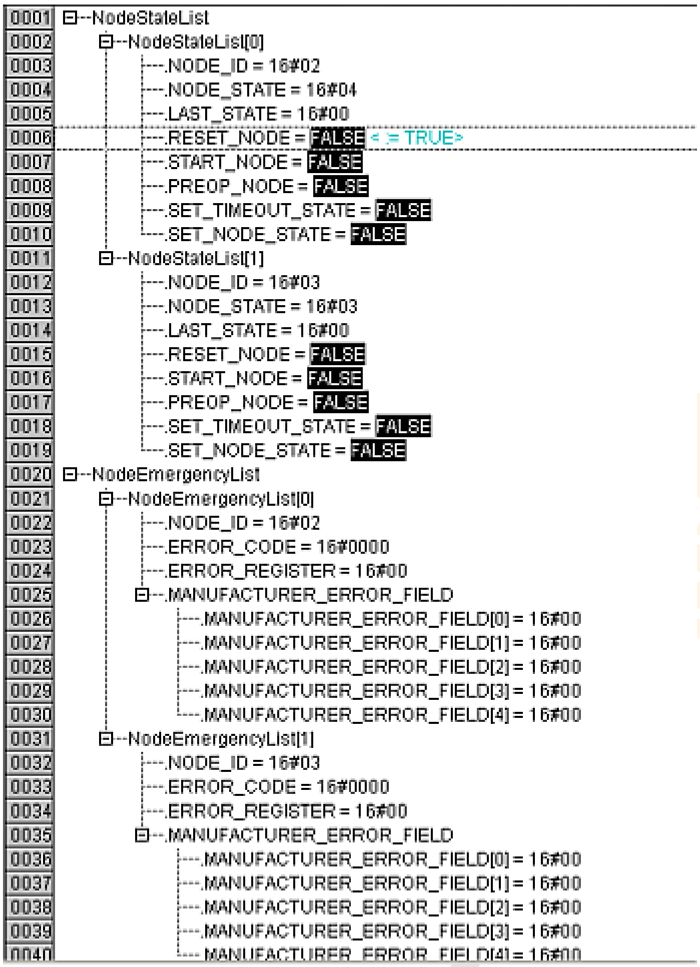


```
TYPE CAN1_EMERGENCY_MESSAGE :
STRUCT
    NODE_ID: BYTE;
    ERROR_CODE: WORD;
    ERROR_REGISTER: BYTE;
    MANUFACTURER_ERROR_FIELD: ARRAY[0..4] OF BYTE;
END_STRUCT
END_TYPE
```

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.



```
⊟--NodeStateList
    ⊟--NodeStateList[0]
    ¦   ----.NODE_ID = 16#02
    ¦   ----.NODE_STATE = 16#04
    ¦   ----.LAST_STATE = 16#00
    ¦   ----.RESET_NODE = FALSE <:= TRUE>
    ¦   ----.START_NODE = FALSE
    ¦   ----.PREOP_NODE = FALSE
    ¦   ----.SET_TIMEOUT_STATE = FALSE
    ¦   ----.SET_NODE_STATE = FALSE
    ⊟--NodeStateList[1]
        ----.NODE_ID = 16#03
        ----.NODE_STATE = 16#03
        ----.LAST_STATE = 16#00
        ----.RESET_NODE = FALSE
        ----.START_NODE = FALSE
        ----.PREOP_NODE = FALSE
        ----.SET_TIMEOUT_STATE = FALSE
        ----.SET_NODE_STATE = FALSE
⊟--NodeEmergencyList
    ⊟--NodeEmergencyList[0]
    ¦   ----.NODE_ID = 16#02
    ¦   ----.ERROR_CODE = 16#0000
    ¦   ----.ERROR_REGISTER = 16#00
    ¦   ⊟--.MANUFACTURER_ERROR_FIELD
    ¦       ----.MANUFACTURER_ERROR_FIELD[0] = 16#00
    ¦       ----.MANUFACTURER_ERROR_FIELD[1] = 16#00
    ¦       ----.MANUFACTURER_ERROR_FIELD[2] = 16#00
    ¦       ----.MANUFACTURER_ERROR_FIELD[3] = 16#00
    ¦       ----.MANUFACTURER_ERROR_FIELD[4] = 16#00
    ⊟--NodeEmergencyList[1]
        ----.NODE_ID = 16#03
        ----.ERROR_CODE = 16#0000
        ----.ERROR_REGISTER = 16#00
        ⊟--.MANUFACTURER_ERROR_FIELD
            ----.MANUFACTURER_ERROR_FIELD[0] = 16#00
            ----.MANUFACTURER_ERROR_FIELD[1] = 16#00
            ----.MANUFACTURER_ERROR_FIELD[2] = 16#00
            ----.MANUFACTURER_ERROR_FIELD[3] = 16#00
            ----.MANUFACTURER_ERROR_FIELD[4] = 16#00
```

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.

ℹ concerning the possible error codes → system manual "Know-How ecomat*mobile*"
→ chapter *CAN / CANopen: errors and error handling*.

5.2.3 Function elements: CANopen slave

Inhalt

1874

ifm electronic provides a number of FBs for the CANopen slave which will be explained below.

CANx_SLAVE_EMCY_HANDLER

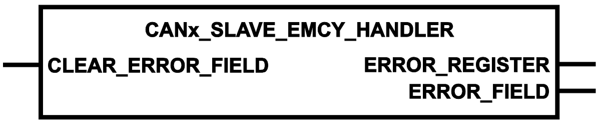
13199

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenSlave_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

2053

CANx_SLAVE_EMCY_HANDLER handles the device-specific error status of the CANopen slave:

- error register (index 0x1001) and
- error field (index 0x1003) of the CANopen object directory.

► Call the function block in the following cases:
- the error status is to be transmitted to the CAN network and
- the error messages of the application program are to be stored in the object directory.

> ! Do you want to store the error messages in the object directory?
>
> ► **After** (repeated) handling of *CANx_SLAVE_SEND_EMERGENCY* (→ page 110) call CANx_SLAVE_EMCY_HANDLER once!

Parameters of the inputs

2054

Parameter	Data type	Description
CLEAR_ERROR_FIELD	BOOL	FALSE ⇨ TRUE (edge): • transmit content of ERROR_FIELD to function block output • delete content of ERROR_FIELD in object directory else: this function is not executed

Parameters of the outputs

2055

Parameter	Data type	Description
ERROR_REGISTER	BYTE	Shows content of OBV index 0x1001 (error register)
ERROR_FIELD	ARRAY [0..5] OF WORD	Shows the content of the OBV index 0x1003 (error field) ERROR_FIELD[0]: number of stored errors ERROR_FIELD[1...5]: Stored errors, the most recent error is shown on index [1]

CANx_SLAVE_NODEID

13202

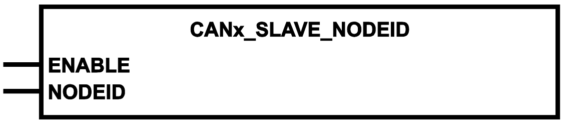
= CANx Slave Node-ID

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenSlave_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

2049

CANx_SLAVE_NODEID enables the setting of the node ID of a CANopen 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): Adopt and activate parameters else: this function is not executed
NODEID	BYTE	node ID = ID of the node permissible values = 0...127

CANx_SLAVE_SEND_EMERGENCY

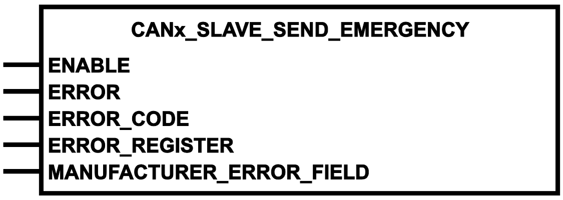
13205

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenSlave_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

2059

CANx_SLAVE_SEND_EMERGENCY transmits application-specific error states. These are error messages which are to be sent in addition to the device-internal error messages (e.g. short circuit on the output).

► Call the FB if the error status is to be transmitted to other devices in the network.

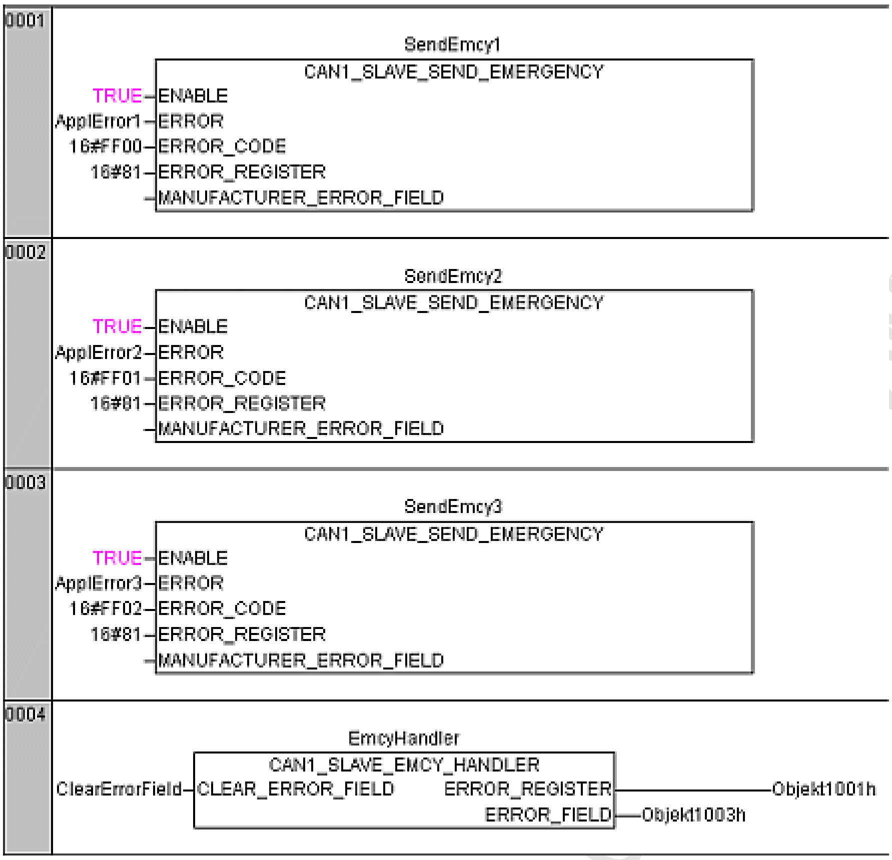
Parameters of the inputs

2060

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
ERROR	BOOL	Using this input, the information whether the error associated to the configured error code is currently present is transmitted. FALSE ⇨ TRUE (edge): sends the next error code if input was not TRUE in the last second TRUE ⇨ FALSE (edge) AND the fault is no longer indicated: after a delay of approx. 1 s: > zero error message is sent else: this function is not executed
ERROR_CODE	WORD	The error code provides detailed information about the detected error. The values should be entered according to the CANopen specification.
ERROR_REGISTER	BYTE	ERROR_REGISTER indicates the error type. The value indicated here is linked by a bit-by-bit OR operation with all the other error messages that are currently active. The resulting value is written into the error register (index 1001_{16}/00) and transmitted with the EMCY message. 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. ApplError1, Code = 0xFF00 in the error register 0x81
2. ApplError2, Code = 0xFF01 in the error register 0x81
3. ApplError3, Code = 0xFF02 in the error register 0x81

CAN1_SLAVE_EMCY_HANDLER sends the error messages to the error register "Object 0x1001" in the error array "Object 0x1003".

CANx_SLAVE_STATUS

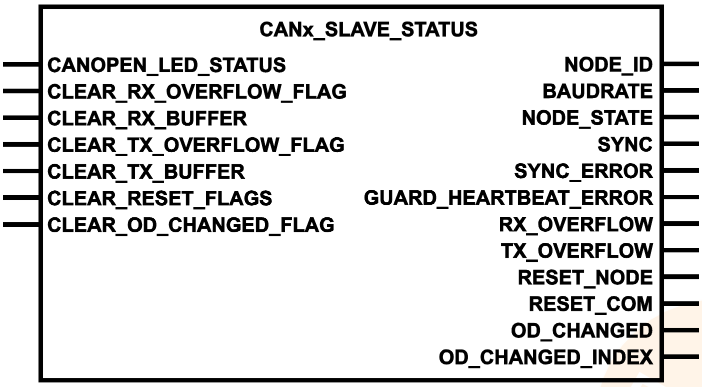
2063

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_CANopenSlave_Vxxyyzz.LIB`

Symbol in CODESYS:



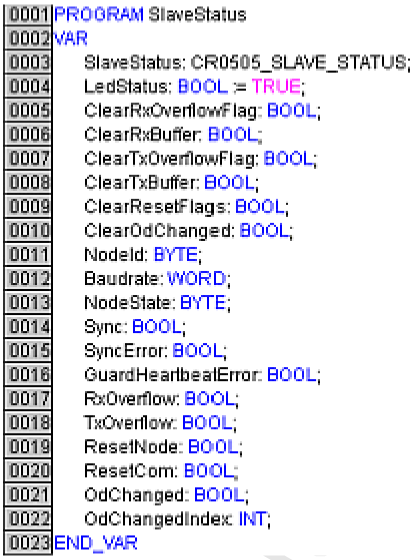
Description

2066

CANx_SLAVE_STATUS shows the status of the device used as CANopen slave. The FB simplifies the use of the CoDeSys CANopen slave libraries. We urgently recommend to carry out the evaluation of the network status via this FB.

At runtime you can then access the individual outputs of the block to obtain a status overview.

Example:



```
PROGRAM SlaveStatus
VAR
    SlaveStatus: CR0505_SLAVE_STATUS;
    LedStatus: BOOL := TRUE;
    ClearRxOverflowFlag: BOOL;
    ClearRxBuffer: BOOL;
    ClearTxOverflowFlag: BOOL;
    ClearTxBuffer: BOOL;
    ClearResetFlags: BOOL;
    ClearOdChanged: BOOL;
    NodeId: BYTE;
    Baudrate: WORD;
    NodeState: BYTE;
    Sync: BOOL;
    SyncError: BOOL;
    GuardHeartbeatError: BOOL;
    RxOverflow: BOOL;
    TxOverflow: BOOL;
    ResetNode: BOOL;
    ResetCom: BOOL;
    OdChanged: BOOL;
    OdChangedIndex: INT;
END_VAR
```

Parameters of the inputs

2067

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 (⇨ state OPERATIONAL). FALSE: The connected network participants are started one after the other.
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE ⇨ TRUE (edge): Clear error flag RX_OVERFLOW else: this function is not executed
CLEAR_RX_BUFFER	BOOL	FALSE ⇨ TRUE (edge): Delete data in the receive buffer else: this function is not executed
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE ⇨ TRUE (edge): Clear error flag TX_OVERFLOW else: this function is not executed
CLEAR_TX_BUFFER	BOOL	FALSE ⇨ TRUE (edge): Delete data in the transmit buffer else: this function is not executed
CLEAR_RESET_FLAGS	BOOL	FALSE ⇨ TRUE (edge): Clear flag RESET_NODE Clear flag RESET_COM else: this function is not executed
CLEAR_OD_CHANGED_FLAGS	BOOL	FALSE ⇨ TRUE (edge): Clear flag OD_CHANGED Clear flag OD_CHANGED-_INDEX else: this function is not executed

Parameters of the outputs

2068

Parameter	Data type	Description
NODE_ID	BYTE	current node ID of the CANopen slave
BAUDRATE	WORD	current baudrate of the CANopen node in [kBaud]
NODE_STATE	BYTE	Current status of CANopen slave 0 = Bootup message sent 4 = CANopen slave in PRE-OPERATIONAL state and is configured via SDO access 5 = CANopen slave in OPERATIONAL state 127 = CANopen slave in PRE-OPERATIONAL state
SYNC	BOOL	SYNC signal of the CANopen master TRUE: In the last cycle a SYNC signal was received FALSE: In the last cycle no SYNC signal was received
SYNC_ERROR	BOOL	TRUE: Error: the SYNC signal of the master was not received or received too late (after expiration of ComCyclePeriod) FALSE: no SYNC error
GUARD_HEARTBEAT_ERROR	BOOL	TRUE: Error: the guarding or heartbeat signal of the master was not received or received too late FALSE: no guarding or heartbeat error
RX_OVERFLOW	BOOL	TRUE: Error: receive buffer overflow FALSE: no overflow
TX_OVERFLOW	BOOL	TRUE: Error: transmission buffer overflow FALSE: no overflow
RESET_NODE	BOOL	TRUE: the CANopen stack of the slave was reset by the master FALSE: the CANopen stack of the slave was not reset
RESET_COM	BOOL	TRUE: the communication interface of the CAN stack was reset by the master FALSE: the communication interface was not reset
OD_CHANGED	BOOL	TRUE: Data in the object directory of the CANopen master have been changed FALSE: no data change
OD_CHANGED_INDEX	INT	Index of the object directory entry changed last

5.2.4 Function elements: CANopen SDOs

Inhalt

2071

Here you will find **ifm** function elements for CANopen handling of Service Data Objects (SDOs).

CANx_SDO_READ

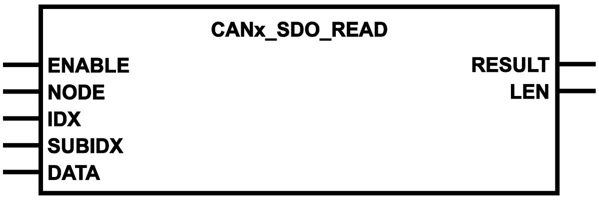
621

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

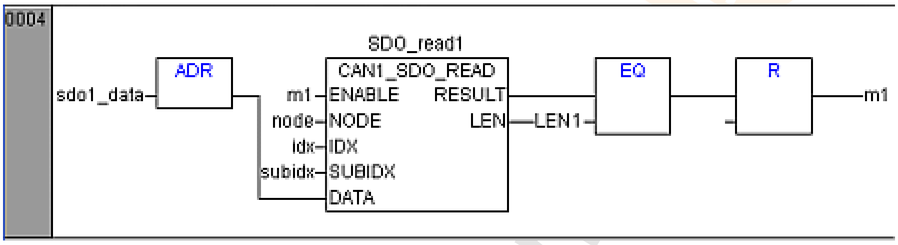
624

CANx_SDO_READ reads the →*SDO* (→ page 275) with the indicated indexes from the node.

Prerequisite: Node must be in the mode "PRE-OPERATIONAL" or "OPERATIONAL".

By means of these, the entries in the object directory can be read. So it is possible to selectively read the node parameters.

Example:



Parameters of the inputs

625

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
NODE	BYTE	ID of the node permissible values = 1...127 = 0x01...0x7F
IDX	WORD	index in object directory
SUBIDX	BYTE	sub-index referred to the index in the object directory
DATA	DWORD	Addresse of the receive data array valid length = 0...255 ❗ Determine the address by means of the operator ADR and assigne it to the FB!

Parameters of the outputs

626

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)
LEN	WORD	Length of the entry in "number of bytes" The value for LEN must not be greater than the size of the receive array. Otherwise any data is overwritten in the application.

Possible results for RESULT:

Value dec | hex		Description
0	00	FB is inactive
1	01	FB execution completed without error – data is valid
2	02	function block is active (action not yet completed)
3	03	Error, no data received during monitoring time

CANx_SDO_WRITE

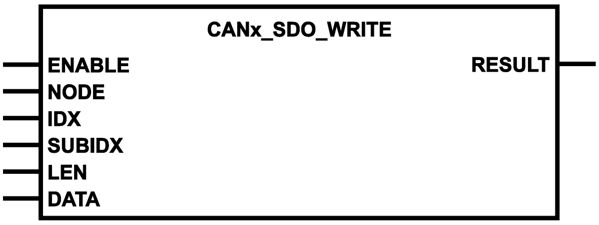
615

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

618

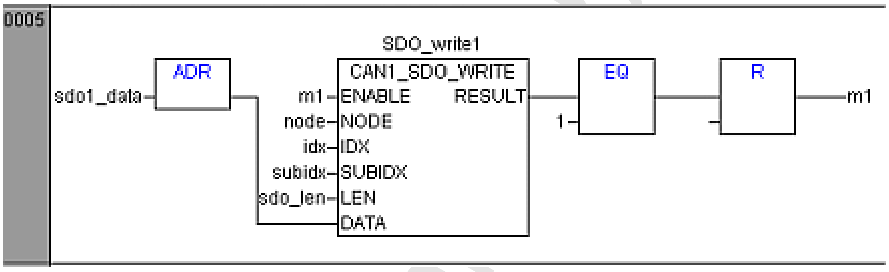
CANx_SDO_WRITE writes the →*SDO* (→ page 275) with the specified indexes to the node.

Prerequisite: the node must be in the state "PRE-OPERATIONAL" or "OPERATIONAL".

Using this FB, the entries can be written to the object directory. So it is possible to selectively set the node parameters.

> ❗ The value for LEN must be lower than the length of the transmit array. Otherwise, random data will be sent.

Example:



Parameters of the inputs

619

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
NODE	BYTE	ID of the node permissible values = 1...127 = 0x01...0x7F
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 not be greater than the size of the transmit array. Otherwise any data is sent.
DATA	DWORD	Address of the transmit data array permissible length = 0...255 ❗ Determine the address by means of the operator ADR and assigne it to the FB!

Parameters of the outputs

620

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)

Possible results for RESULT:

Value dec	hex	Description
0	00	FB is inactive
1	01	FB execution completed without error – data is valid
2	02	function block is active (action not yet completed)
3	03	Error, data cannot be transmitted

5.2.5 Function elements: SAE J1939

Inhalt

2273

For SAE J1939, **ifm electronic** provides a number of function elements which will be explained in the following.

J1939_x

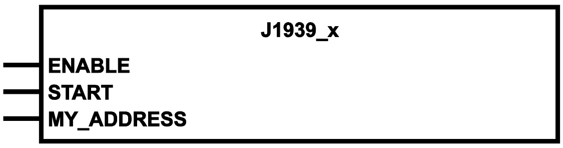
9375

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

4325

J1939_x serves as protocol handler for the communication profile SAE J1939.

4313

NOTE

(for RTS to v05 only)
J1939 communication via the 1st CAN interface:

- ► First initialise the interface via *CAN1_EXT* (→ page 84)!

J1939 communication via the 2nd CAN interface:

- ► First initialise the interface via *CAN2*!

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.

Parameters of the inputs

469

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
START	BOOL	TRUE (only for 1 cycle): Start J1939 protocol at CAN interface x FALSE: during further processing of the program
MY_ADDRESS	BYTE	J1939 address of the device

J1939_x_GLOBAL_REQUEST

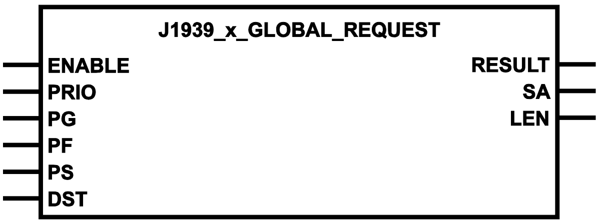
4315

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxxyyzz.LIB`

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 parameters PG, PF, PS and the address of the array DST in which the received data is stored are assigned to the FB.

ⓘ **Info**

PGN = [Page] + [PF] + [PS]
PDU = [PRIO] + [PGN] + [J1939 address] + [data]

13790

NOTICE

Risk of inadmissible overwriting of data!

- ► Create a receiver array with a size of 1 785 bytes.
  This is the maximum size of a J1939 message.
- ► Check the amount of received data:
  the value must not exceed the size of the array created to receive data!

- ► For every requested message use an own instance of the FB!
- ► To the destination address DST applies:
  ❗ Determine the address by means of the operator ADR and assigne it 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.
- RESULT = 2: the POU is waiting for data of the participants.
- RESULT = 1: data was received by a participant.
  The output LEN indicates how many data bytes have been received.
  Store / evaluate this new data immediately!
  When a new message is received, the data in the memory address DST is overwritten.
- RESULT = 0: no participant on the bus sends a reply within 1.25 seconds.
  The FB returns to the non-active state.
  Only now may ENABLE be set again to FALSE!
- ► For the reception of data from several participants at short intervals:
  call the POU several times in the same PLC cycle and evaluate it at once!

Parameters of the inputs

463

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
PRIO	BYTE	message priority (0…7)
PG	BYTE	Data page Value of defined PGN (Parameter Group Number) allowed = 0...1 (normally = 0)
PF	BYTE	PDU format byte Value of defined PGN (Parameter Group Number) PDU2 (global) = 240...255
PS	BYTE	PDU specific byte Value of defined PGN (Parameter Group Number) GE (Group Extension) = 0...255
DST	DWORD	destination address ❗ Determine the address by means of the operator ADR and assigne it to the FB!

Info

PGN = [Page] + [PF] + [PS]
PDU = [PRIO] + [PGN] + [J1939 address] + [data]

Parameters of the outputs

464

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)
SA	BYTE	J1939 address of the answering device
LEN	WORD	Number of received bytes

Possible results for RESULT:

Value dec	hex	Description
0	00	FB is inactive
1	01	FB execution completed without error – data is valid
2	02	function block is active (action not yet completed)
3	03	Error

J1939_x_RECEIVE

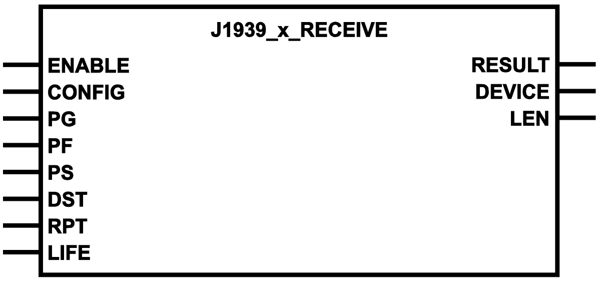
9393

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxxyyzz.LIB`

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.

❗ Once the following parameters have been configured they can no longer be modified in the running application program: PG, PF, PS, RPT, LIFE, DST.

13790

NOTICE

Risk of inadmissible overwriting of data!

- ► Create a receiver array with a size of 1 785 bytes.
  This is the maximum size of a J1939 message.
- ► Check the amount of received data:
  the value must not exceed the size of the array created to receive data!

- ► To the destination address DST applies:
  ❗ Determine the address by means of the operator ADR and assigne it to the FB!

❗ Once RPT has been set it can no longer be modified!

- ► 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).

❗ This block must also be used if the messages are requested using the FBs J1939_..._REQUEST.

Parameters of the inputs

457

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
CONFIG	BOOL	TRUE (in the 1st cycle): configure data object FALSE: during further processing of the program
PG	BYTE	Data page Value of defined PGN (Parameter Group Number) allowed = 0...1 (normally = 0)
PF	BYTE	PDU format byte Value of defined PGN (Parameter Group Number) PDU1 (specific) = 0...239 PDU2 (global) = 240...255
PS	BYTE	PDU specific byte Value of defined PGN (Parameter Group Number) If PF = PDU1 ➩ PS = DA (Destination Address) (DA = J1939 address of external device) If PF = PDU2 ➩ PS = GE (Group Extension)
DST	DWORD	destination address ❗ Determine the address by means of the operator ADR and assigne it to the FB!
RPT	TIME	Monitoring time Within this time window the messages must be received cyclically. > Otherwise, there will be an error message. RPT = T#0s ➩ no monitoring ❗ Once RPT has been set it can no longer be modified!
LIFE	BYTE	tolerated number of J1939 messages not received

Parameters of the outputs

458

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)
DEVICE	BYTE	J1939 address of the sender
LEN	WORD	Number of received bytes

Possible results for RESULT:

Value dec	hex	Description
0	00	FB is inactive
1	01	FB execution completed without error – data is valid
3	03	Error, no data received during monitoring time

J1939_x_RESPONSE

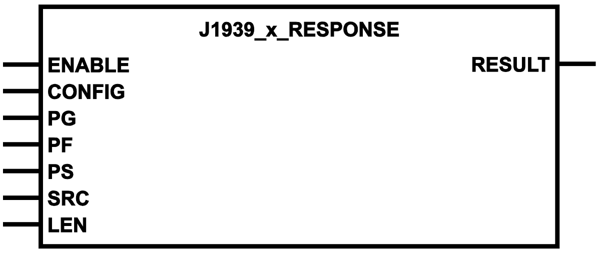
9399

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxxyyzz.LIB`

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.

- ► To the source address SRC applies:
  ❗ Determine the address by means of the operator ADR and assigne it 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
CONFIG	BOOL	TRUE (in the 1st cycle): configure data object FALSE: during further processing of the program
PG	BYTE	Data page Value of defined PGN (Parameter Group Number) allowed = 0...1 (normally = 0)
PF	BYTE	PDU format byte Value of defined PGN (Parameter Group Number) PDU1 (specific) = 0...239 PDU2 (global) = 240...255
PS	BYTE	PDU specific byte Value of defined PGN (Parameter Group Number) If PF = PDU1 ➪ PS = DA (Destination Address) (DA = J1939 address of external device) If PF = PDU2 ➪ PS = GE (Group Extension)
SRC	DWORD	Start address in source memory ❗ Determine the address by means of the operator ADR and assigne it to the FB!
LEN	WORD	number (≥ 1) of the data bytes to be transmitted

Parameters of the outputs

13993

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)

Possible results for RESULT:

Value dec | hex		Description
0	00	FB is inactive
1	01	Data transfer completed without errors
2	02	function block is active (action not yet completed)
3	03	Error, data cannot be transmitted

J1939_x_SPECIFIC_REQUEST

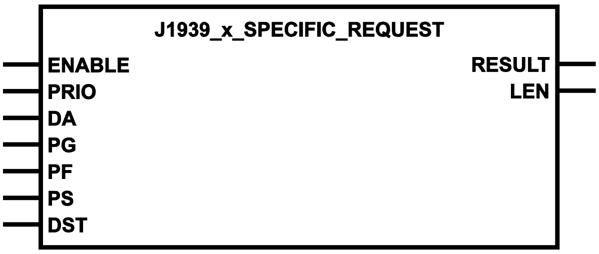
8884

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxxyyzz.LIB`

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.

Info

PGN = [Page] + [PF] + [PS]
PDU = [PRIO] + [PGN] + [J1939 address] + [data]

13790

NOTICE

Risk of inadmissible overwriting of data!

- ► Create a receiver array with a size of 1 785 bytes.
  This is the maximum size of a J1939 message.
- ► Check the amount of received data:
  the value must not exceed the size of the array created to receive data!

- ► To the destination address DST applies:
  ❗ Determine the address by means of the operator ADR and assigne it 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
PRIO	BYTE	message priority (0…7)
DA	BYTE	J1939 address of the requested device
PG	BYTE	Data page Value of defined PGN (Parameter Group Number) allowed = 0...1 (normally = 0)
PF	BYTE	PDU format byte Value of defined PGN (Parameter Group Number) PDU1 (specific) = 0...239 PDU2 (global) = 240...255
PS	BYTE	PDU specific byte Value of defined PGN (Parameter Group Number) If PF = PDU1 ⇨ PS = DA (Destination Address) (DA = J1939 address of external device) If PF = PDU2 ⇨ PS = GE (Group Extension)
DST	DWORD	destination address ! Determine the address by means of the operator ADR and assigne it to the FB!

Info

PGN = [Page] + [PF] + [PS]
PDU = [PRIO] + [PGN] + [J1939 address] + [data]

Parameters of the outputs

446

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)
LEN	WORD	Number of received bytes

Possible results for RESULT:

Value dec | hex		Description
0	00	FB is inactive
1	01	FB execution completed without error – data is valid
2	02	function block is active (action not yet completed)
3	03	Error

J1939_x_TRANSMIT

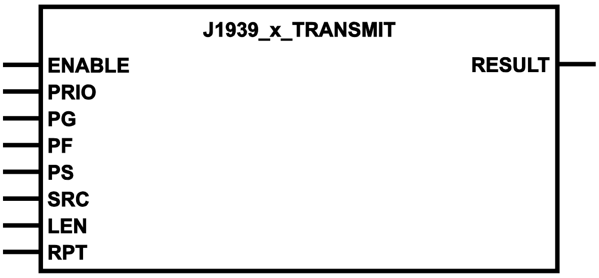
4322

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxxyyzz.LIB`

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.

> ⓘ **Info**
>
> PGN = [Page] + [PF] + [PS]
> PDU = [PRIO] + [PGN] + [J1939 address] + [data]

- ► To the source address SRC applies:
  ❗ Determine the address by means of the operator ADR and assigne it 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.

ⓘ 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
PRIO	BYTE	message priority (0…7)
PG	BYTE	Data page Value of defined PGN (Parameter Group Number) allowed = 0...1 (normally = 0)
PF	BYTE	PDU format byte Value of defined PGN (Parameter Group Number) PDU1 (specific) = 0...239 PDU2 (global) = 240...255
PS	BYTE	PDU specific byte Value of defined PGN (Parameter Group Number) If PF = PDU1 ➩ PS = DA (Destination Address) (DA = J1939 address of external device) If PF = PDU2 ➩ PS = GE (Group Extension)
SRC	DWORD	Start address in source memory ! Determine the address by means of the operator ADR and assigne it to the FB!
LEN	WORD	Number of data bytes to be transmitted allowed = 1...1 785 = 0x0001...0x06F9
RPT	TIME	Repeat time during which the data messages are to be transmitted cyclically RPT = T#0s ➩ sent only once

Info

PGN = [Page] + [PF] + [PS]
PDU = [PRIO] + [PGN] + [J1939 address] + [data]

Parameters of the outputs

440

Parameter	Data type	Description
RESULT	BYTE	feedback of the function block (possible messages → following table)

Possible results for RESULT:

Value dec	hex	Description
0	00	FB is inactive
1	01	FB execution completed without error – data is valid
2	02	function block is active (action not yet completed)
3	03	Error, data cannot be transmitted

5.2.6 Function elements: serial interface

Inhalt

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.

The serial interface can be used in the application program by means of the following FBs.

SERIAL_PENDING

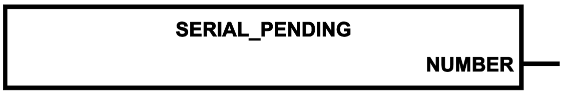
20200

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

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 134) 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.
To do so, set the system flag bit SERIAL_MODE=TRUE!

❗ 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.

Parameters of the outputs

319

Parameter	Data type	Description
NUMBER	WORD	Number of data bytes received

SERIAL_RX

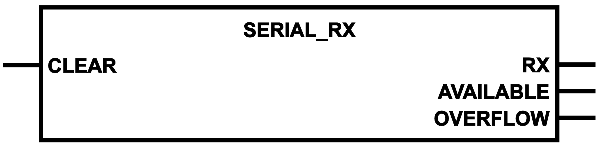
20201

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

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.

If 7-bit data transmission is used, the 8th bit contains the parity and must be suppressed by the user if necessary.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.
To do so, set the system flag bit SERIAL_MODE=TRUE!

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.

Parameters of the inputs

312

Parameter	Data type	Description
CLEAR	BOOL	TRUE: delete receive buffer FALSE: function element 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 remaining data bytes 0 = no valid data available
OVERFLOW	BOOL	TRUE: Overflow of the data buffer ⇨ loss of data! FALSE: Data buffer is without data loss

SERIAL_SETUP

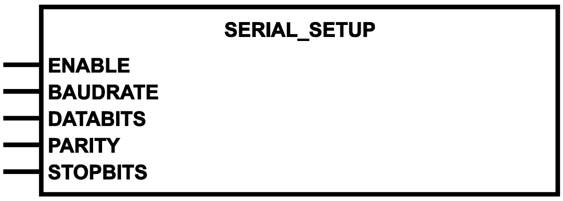
20202

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

Symbol in CODESYS:



Description

305

SERIAL_SETUP initialises the serial RS232 interface.

The function block does not necessarily need to be executed in order to be able to use the serial interface. Without function block call the default settings below apply.

Using ENABLE=TRUE for one cycle, the function block sets the serial interface to the indicated parameters. The changes made with the help of the function block are saved non-volatily.

> **❗ 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.

5020

> **NOTICE**
>
> The driver module of the serial interface can be damaged!
>
> Disconnecting or connecting the serial interface while live can cause undefined states which damage the driver module.
>
> ► Do not disconnect or connect the serial interface while live.

Parameters of the inputs

306

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only for 1 cycle): Initialise interface FALSE: during further processing of the program
BAUD RATE	WORD	Baud rate Permissible values → data sheet Preset value → data sheet
DATABITS	BYTE := 8	Number of data bits allowed = 7 or 8
PARITY	BYTE := 0	Parity allowed: 0=none, 1=even, 2=odd
STOPBITS	BYTE := 1	Number of stop bits allowed = 1 or 2

SERIAL_TX

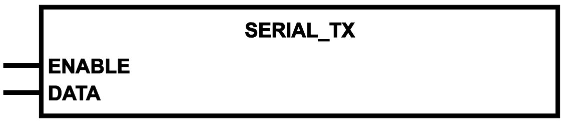
20203

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'.

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.
To do so, set the system flag bit SERIAL_MODE=TRUE!

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.

Parameters of the inputs

300

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
DATA	BYTE	value to be transmitted

5.2.7 POUs: Extended side (slave)

Inhalt

20220

- The controller half with the application program operates as master on its side.
  The other controller half operates as slave.
  ▶ Use the FB *SSC_SET_MASTER* (→ page 140) in the application program (on the master side).
  > The FB assumes the function of initialising the slave.

> The master loads a small dummy program into the slave.
▶ Use the FB *DUMMY* (→ page 138) from the library `ifm_CR0200_DUMMY_Vxxyyzz.LIB` in the application program.

→ chapter *Operating modes of the ExtendedController* (→ page 49)

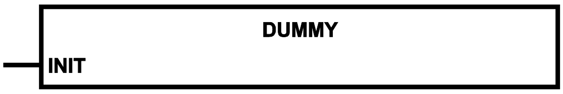
DUMMY

20171

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_DUMMY_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

20173

DUMMY initialises the slave side of the extended controller.
This FB is required for the master-slave operation of the ExtendedController.

Parameters of the inputs

20175

Parameter	Data type	Description
INIT	BOOL	TRUE (in the 1st cycle): initialise the slave of the ExtendedController FALSE: during further processing of the program

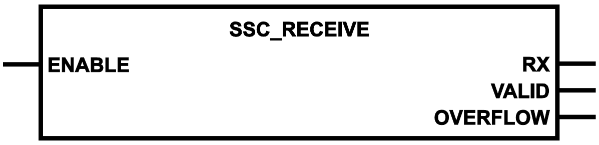
SSC_RECEIVE

254

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 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 142). 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified

Parameters of the outputs

259

Parameter	Data type	Description
RX	ARRAY [0..19] OF BYTE	data array [0..19]
VALID	BOOL	TRUE (only for 1 cycle): new valid data has been received FALSE: during further processing of the program
OVERFLOW	BOOL	TRUE: Overflow of the data buffer ⇨ loss of data! FALSE: Data buffer is without data loss

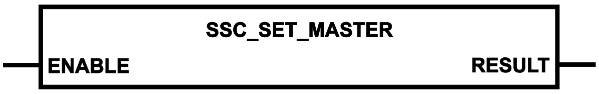
SSC_SET_MASTER

248

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

251

SSC_SET_MASTER activates the master/slave mode for the ExtendedController.

This FB is an internal communication function for the ExtendedController. It initialises the master/slave operating mode of the controller. The inputs and outputs are processed synchronously. The master loads a small dummy program into the slave. This program block of the slave library `ifm_CR0200_DUMMY_Vxxyyzz.LIB` must be inserted into the application program.

The names of the system variables and system functions of the 2nd input/output level are extended by an _E (for Extended) to distinguish them from those of the 1st input/output level. The data exchange between the two halves of the controller is automatically carried out via the internal SSC interface.

> **! NOTE**
>
> The programmer must read the received data from the data array and ensure 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. Otherwise there is the risk of losing data for example due to different cycle times.

Parameters of the inputs

252

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified

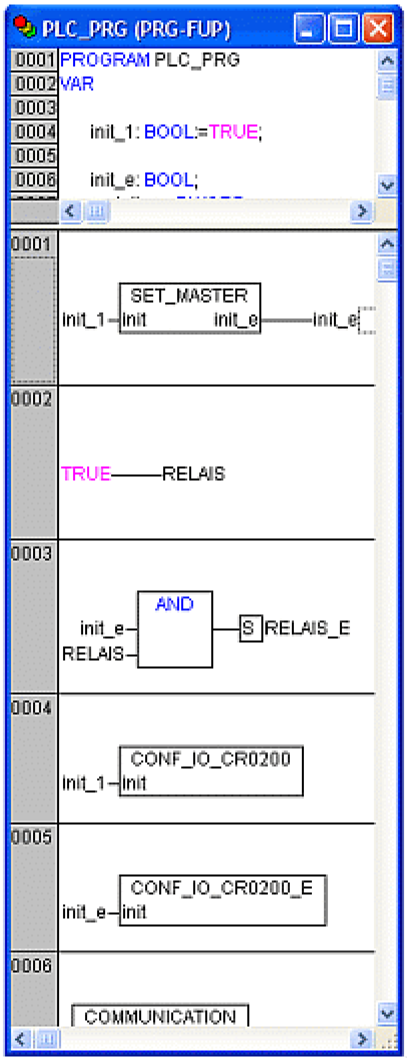
Parameters of the outputs

253

Parameter	Data type	Description
RESULT	BYTE	0 = not active (SSC_NO_WORKING) 1 = data transmission completed (SSC_DONE) 2 = unit active (data transmission) (SSC_BUSY) 3 = error, data cannot be sent (SSC_ERROR)

Example: SSC_SET_MASTER

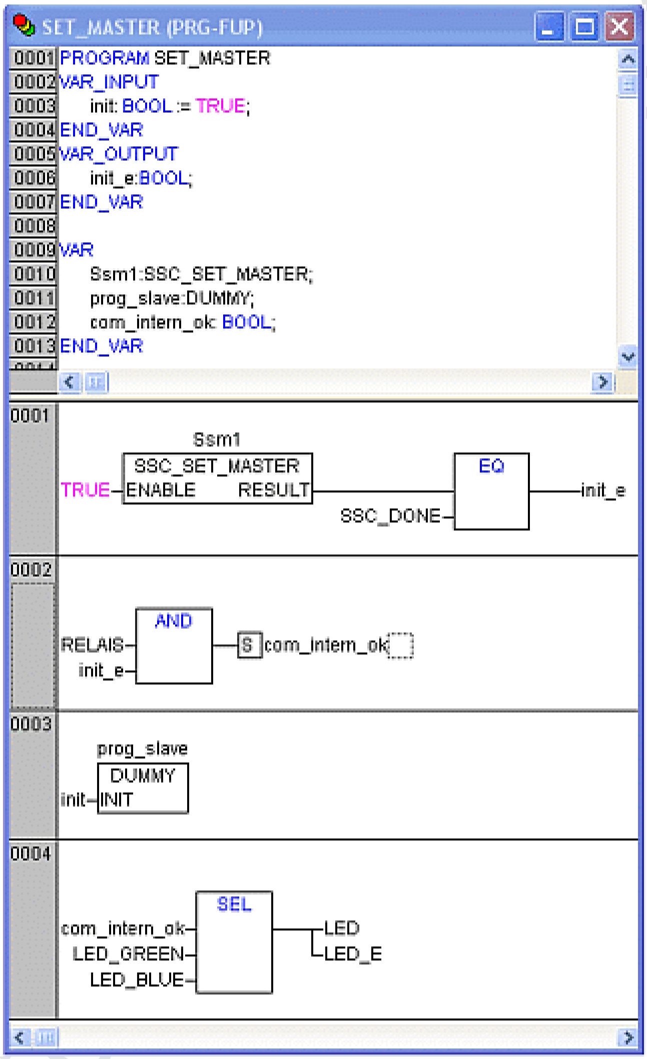
20224



```
PROGRAM PLC_PRG
VAR

    init_1: BOOL:=TRUE;

    init_e: BOOL;
```



```
PROGRAM SET_MASTER
VAR_INPUT
    init: BOOL := TRUE;
END_VAR
VAR_OUTPUT
    init_e:BOOL;
END_VAR

VAR
    Ssm1:SSC_SET_MASTER;
    prog_slave:DUMMY;
    com_intern_ok: BOOL;
END_VAR
```

The example shows the principal program structure with initialisation and integration of the dummy program (FB DUMMY). To adhere to the time flow during initialisation, the structure of the application program should follow the example.

! POU names which end with "_E" are executed in the slave module.

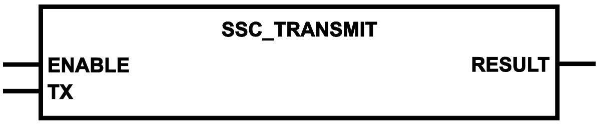
SSC_TRANSMIT

242

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 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. 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
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

5.2.8 Function elements: Optimising the PLC cycle

Inhalt

8609

Here we show you functions to optimise the PLC cycle.

Function elements: processing interrupts

Inhalt

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.

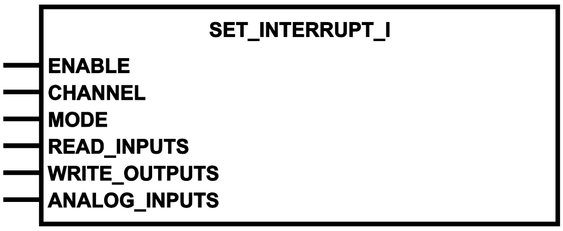
SET_INTERRUPT_I

2381

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

281
11573

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.

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 runtime of the main cycle plus the sum of the duration of all program parts called via interrupt must always be within the max. permissible cycle time!

The user is responsible for data consistency between the main program and the program parts running in the interrupt mode!

19866

Interrupt priorities:

- All program parts called via interrupt have the same priority of execution. Several simultaneous interrupts are processed sequentially in the order of their occurrence.
- If a further edge is detected on the same input during execution of the program part called via interrupt, the interrupt is listed for processing and the program is directly called again after completion. As an option, interfering multiple pulses can be filtered out by setting the glitch filter.
- The program running in the interrupt mode can be disrupted by interrupts with a higher priority (e.g. CAN).
- If several interrupts are present on the same channel, the last initialised FB (or the PRG) will be assigned the channel. The previously defined FB (or the PRG) is then no longer called and no longer provides data.

971

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 (Controller: CR0n3n, CR7n3n)
%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)
%IX0.0, %IX0.8 (SmartController: CR250n)
IN08...IN11 (CabinetController: CR030n)
IN0...IN3 (PCB controller: CS0015)

Inputs, analogue:

%IX0.0...%IX0.7 (Controller: CR0n3n, CR7n3n)
All channels (selection bit-coded) (all other controller)

Outputs, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)
%QX0.0, %QX0.8 (SafetyController: CR7nnn)
OUT00...OUT03 (CabinetController: CR030n)
OUT0...OUT7 (PCB controller: CS0015)

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

2383
282

Parameter	Data type	Description	
ENABLE	BOOL	TRUE (only for 1 cycle): initialisation of the function block FALSE: unit is not executed	
CHANNEL	BYTE	Number of interrupt input	
		• CabinetController: CR030n	0 = IN08 ... 3 = IN11
		• ClassicController: CR0020, CR0505 • ExtendedController: CR0200	0 = %IX1.4 ... 3 = %IX1.7
		• ClassicController: CR0032, CR0033 • ExtendedController: CR0232, CR0233	0 = IN00 ... 7 = IN07
		• SmartController: CR250n	0 = %IX0.0 1 = %IX0.8
		• PCB controller: CS0015 • PDM360smart: CR1071	0 = IN0 ... 3 = IN3
MODE	BYTE	Type of edge at the input CHANNEL which triggers the interrupt 1 = rising edge (standard value) 2 = falling edge 3 = rising and falling edge > 3 = standard value	
READ_INPUTS	BOOL	TRUE: read the inputs 0...7 before calling the program and write into the input flags I00...I07 FALSE: only read the channel indicated under CHANNEL and write to the corresponding input flag Ixx	
WRITE_OUTPUTS	BOOL	TRUE: write the current values of the output flags Q00...Q07 to the outputs after completion of the program sequence FALSE: do not write outputs	
ANALOG_INPUTS	BOOL	TRUE: read inputs 0...7 and write the unfiltered, uncalibrated analogue values to the flags ANALOG_IRQ00...07 FALSE: do not write flags ANALOG_IRQ00...07	

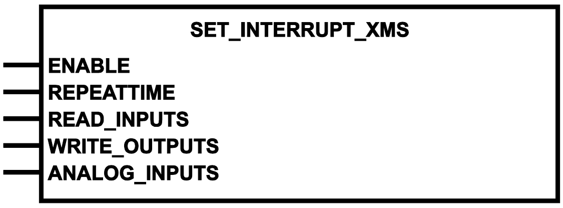
SET_INTERRUPT_XMS

272

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 xAnother 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.

The user is responsible for data consistency between the main program and the program parts running in the interrupt!

Please note: In case of a high CAN bus activity the set REPEATTIME may fluctuate.

971

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 (Controller: CR0n3n, CR7n3n)
%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)
%IX0.0, %IX0.8 (SmartController: CR250n)
IN08...IN11 (CabinetController: CR030n)
IN0...IN3 (PCB controller: CS0015)

Inputs, analogue:

%IX0.0...%IX0.7 (Controller: CR0n3n, CR7n3n)
All channels (selection bit-coded) (all other controller)

Outputs, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)
%QX0.0, %QX0.8 (SafetyController: CR7nnn)
OUT00...OUT03 (CabinetController: CR030n)
OUT0...OUT7 (PCB controller: CS0015)

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

2382

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only for 1 cycle): initialisation of the function block FALSE: unit is not executed
REPEATTIME	TIME	Duration in [ms] between end of program and reboot The duration between two calls is determined as the sum of REPEATTIME and runtime of the program called via interrupt.
READ_INPUTS	BOOL	TRUE: read the inputs 0...7 before calling the program and write into the input flags I00...I07 FALSE: no update of the inputs
WRITE_OUTPUTS	BOOL	TRUE: write the current values of the output flags Q00...Q07 to the outputs after completion of the program sequence FALSE: do not write outputs
ANALOG_INPUTS	BOOL	TRUE: read inputs 0...7 and write the unfiltered, uncalibrated analogue values to the flags ANALOG_IRQ00...07 FALSE: do not write flags ANALOG_IRQ00...07

5.2.9 Function elements: processing input values

Inhalt

1602
1302

In this chapter we show you **ifm** FBs which allow you to read and process the analogue or digital signals at the device input.

NOTE

The analogue raw values shown in the PLC configuration of CODESYS directly come from the ADC. They are not yet corrected!

Therefore different raw values can appear in the PLC configuration for identical devices.

Error correction and normalisation are only carried out by ifm function blocks. The function blocks provide the corrected value.

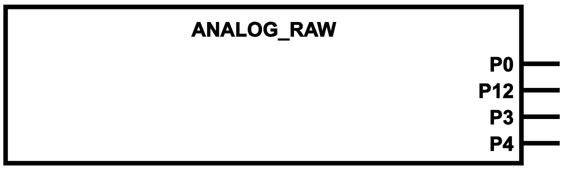
ANALOG_RAW

19589

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

9918

ANALOG_RAW provides the raw analogue signal of the inputs, without any filtering.

Parameters of the outputs

13148

Parameter	Data type	Description
P0	ARRAY [0..7] OF WORD	Raw input values of the analogue inputs, port 0: P0.0 for I00 ... P0.7 for I07
P12	ARRAY [0..7] OF WORD	Raw input values of the analogue inputs, ports 1+2: P12.0 for I14 ... P12.3 for I17 P12.4 for I24 ... P12.7 for I27
P3	ARRAY [0..7] OF WORD	Raw input values of the analogue inputs, port 3: P3.0 for I30 ... P3.7 for I37
P4	ARRAY [0..7] OF WORD	Raw input values of the analogue inputs, port 4: P4.0 for I40 ... P4.7 for I47

INPUT_ANALOG

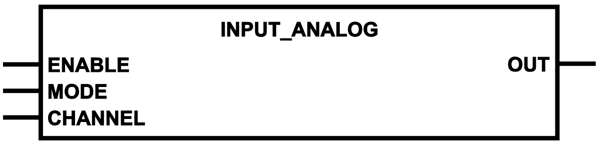
20208

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'.

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.

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_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.

> When using this FB you must set the system variable RELAIS *).
> Otherwise the internal reference voltages are missed for the current measurement.

*) Relay exists only in the following devices: CR0020, CRnn32, CRnn33, CR0200, CR0505, CR7nnn

Parameters of the inputs

523

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
MODE	BYTE	IN_DIGITAL_H Digital input IN_CURRENT Current input 0/20... 000 IN_VOLTAGE10 Voltage input 0...10 000 mV IN_VOLTAGE30 Voltage input 0...30 000 mV IN_RATIO ratiometric analogue input
INPUT_CHANNEL	BYTE	Number of input channel allowed = 0...7

Parameters of the outputs

524

Parameter	Data type	Description
OUT	WORD	Output value according to MODE in case of an invalid setting: OUT = "0"

INPUT_CURRENT

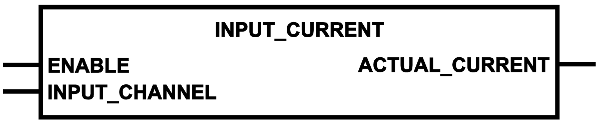
20209

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 151) should be used.

Parameters of the inputs

517

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
INPUT_CHANNEL	BYTE	Number of input channel allowed = 0...7

Parameters of the outputs

518

Parameter	Data type	Description
ACTUAL_CURRENT	WORD	input current in [µA]

INPUT_VOLTAGE

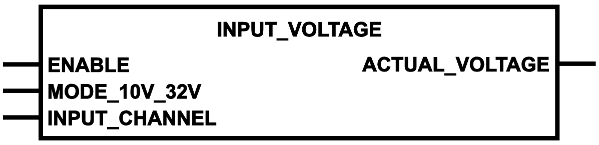
20210

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 151) should be used.

Parameters of the inputs

511

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
MODE_10V_32V	BOOL	TRUE: voltage range 0...32 V FALSE: voltage range 0...10 V
INPUT_CHANNEL	BYTE	Number of input channel allowed = 0...7

Parameters of the outputs

512

Parameter	Data type	Description
ACTUAL_VOLTAGE	WORD	input voltage in [mV]

5.2.10 Function elements: adapting analogue values

Inhalt

1603

If the values of analogue inputs or the results of analogue functions must be adapted, the following FBs will help you.

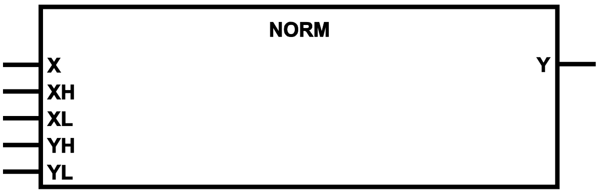
NORM

401

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 X.

> 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	input value
XH	WORD	Upper limit of input value range [increments]
XL	WORD	Lower limit of input value range [increments]
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	output value

Example: NORM (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: NORM (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

5.2.11 Function elements: counter functions for frequency and period measurement

Inhalt

18818

The controllers support up to 4 fast inputs which can process input frequencies of up to 30 kHz. In addition to frequency measurement, the FRQ inputs can also be used for the evaluation of incremental encoders (counter function).

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

The following FBs are available for easy evaluation:

Function element	Permissible values	Explanation
FREQUENCY	0...50 000 Hz	Measurement of the frequency on the indicated channel. Measurement error is reduced in case of high frequencies
PERIOD	0.1...30 000 Hz	Measurement of frequency and period duration (cycle time) on the indicated channel
PERIOD_RATIO	0...30 000 Hz	Measurement of frequency and period duration (cycle time) as well as mark-to-space ratio [‰] on the indicated channel
FREQUENCY_PERIOD	0...30 000 Hz	The FB combines the two FBs FREQUENCY and PERIOD or PERIOD_RATIO. Automatic selection of the measuring method at 5 kHz
PHASE	0...5 000 Hz	Reading of a channel pair and comparison of the phase position of the signals
INC_ENCODER	0...5 000 Hz	Up/down counter function for the evaluation of encoders
FAST_COUNT	0...5 000 Hz	Counting of fast pulses

! Important when using the fast inputs as "normal" digital inputs:

► The increased sensitivity to noise pulses must be taken into account (e.g. contact bouncing for mechanical contacts).

- The standard digital input can evaluate signals up to 50 Hz.

FAST_COUNT

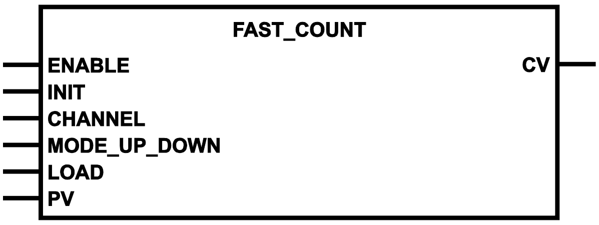
567

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ 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.

❗ Due to the technical design, for the **ecomat*mobile*** 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

10253

Parameter	**Data type**	**Description**
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > counter stopped
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	Number of the fast input channel (0...3) 0...3 for the inputs I14...I17 ℹ For the function block xxx_E (if available) applies: 0...3 for the inputs I14_E...I17_E
MODE_UP_DOWN	BOOL	TRUE: counter counts downwards FALSE: counter counts upwards
LOAD	BOOL	TRUE: start value PV is loaded in CV FALSE: function element is not executed
PV	DWORD	Start value (preset value) for the counter

Parameters of the outputs

572

Parameter	**Data type**	**Description**
CV	DWORD	current counter value Behaviour in case of overflow: • the counter stops at 0 when counting downwards • there is an overflow when counting upwards

FREQUENCY

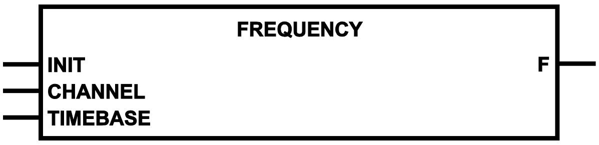
537

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

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].

> ! For FREQUENCY only the inputs FRQ0...FRQ3 can be used.

Parameters of the inputs

10258

Parameter	Data type	Description
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	Number of the fast input channel (0...3) 0...3 for the inputs I14...I17 ℹ For the function block xxx_E (if available) applies: 0...3 for the inputs I14_E...I17_E
TIMEBASE	TIME	Time basis for frequency measurement (max. 57 s)

8406

> ! 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

542

Parameter	Data type	Description
F	REAL	frequency of the input signal in [Hz]

INC_ENCODER

4187

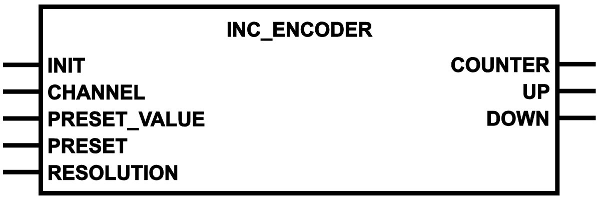
= Incremental Encoder

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'.

Symbol in CODESYS:



Description

4330

INC_ENCODER offers up/down counter functions for the evaluation of encoders.

Each input pair to be evaluated by means of the function block is formed by two frequency inputs.

Limit frequency = 30 kHz
max. number of units to be connected: 4 encoders (ExtendedController: max. 8 encoders)

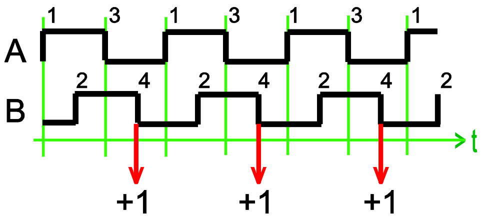
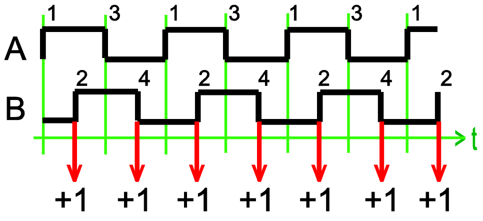
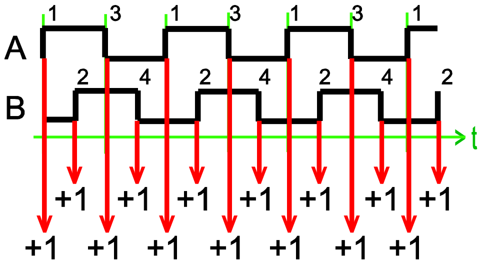
Set preset value:
1. Enter value in PRESET_VALUE
2. Set PRESET to TRUE for one cycle
3. Reset PRESET to FALSE

The function block counts the pulses at the inputs as long as INIT=FALSE and PRESET=FALSE.
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.

Do **not** use this function block on one input together with one of the following function blocks!
- *FAST_COUNT* (→ page 159)
- *FREQUENCY* (→ page 160)
- *PERIOD* (→ page 164)
- *PERIOD_RATIO* (→ page 166)
- *PHASE* (→ page 168)

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.
 A B 1 3 1 3 1 3 1 2 4 2 4 2 4 2 t +1 +1 +1 +1 +1 +1 +1	RESOLUTION = 2 In the case of double resolution the falling and the rising edges of the B-signal are evaluated.
 A B 1 3 1 3 1 3 1 2 4 2 4 2 4 2 t +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1	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

17806

Parameter	Data type	Description
INIT	BOOL	TRUE (only for 1 cycle): Function block is initialised FALSE: during further processing of the program
CHANNEL	BYTE	Number of the input channel pair (0...3) 0 = channel pair 0 = inputs I14 + I15 1 = channel pair 1 = inputs I16 + I17 2 = channel pair 2 = inputs I24 + I25 3 = channel pair 3 = inputs I26 + I27 ℹ For the function block xxx_E (if available) applies: 0 = channel pair 0 = inputs I14_E + I15_E 1 = channel pair 1 = inputs I16_E + I17_E 2 = channel pair 2 = inputs I24_E + I25_E 3 = channel pair 3 = inputs I26_E + I27_E
PRESET_VALUE	DINT	counter start value
PRESET	BOOL	FALSE ⇨ TRUE (edge): PRESET_VALUE is loaded to COUNTER TRUE: Counter ignores the input pulses FALSE: Counter counts the input pulses
RESOLUTION	BYTE	evaluation of the encoder resolution: 01 = counts for every fourth edge (= resolution of the encoder) 02 = counts for every second edge 04 = counts for every rising and falling edge All other values count as "01".

Parameters of the outputs

530

Parameter	Data type	Description
COUNTER	DINT	Current counter value
UP	BOOL	TRUE: counter counts upwards in the last cycle FALSE: counter counts not upwards in the last cycle
DOWN	BOOL	TRUE: counter counts downwards in the last cycle FALSE: counter counts not downwards in the last cycle

PERIOD

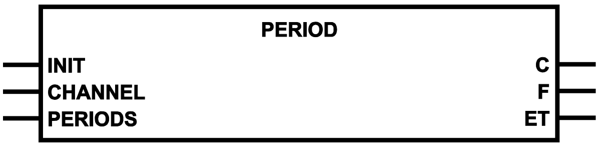
370

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

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.
For PDM360smart: CR1071: all inputs.

Frequencies < 0.5 Hz are no longer clearly indicated!

Parameters of the inputs

17808

Parameter	Data type	Description
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	Number of the fast input channel (0...3) 0...3 for the inputs I24...I27 ℹ For the function block xxx_E (if available) applies: 0...3 for the inputs I24_E...I27_E
PERIODS	BYTE	Number of periods to be compared

8406

❗ 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

375

Parameter	Data type	Description
C	DWORD	Cycle time of the detected periods in [µs] allowed = 200...10 000 000 = 0xC8...0x989680 (= 10 seconds)
F	REAL	frequency of the input signal in [Hz]
ET	TIME	time elapsed since the last rising edge on the input (can be used for very slow signals)

PERIOD_RATIO

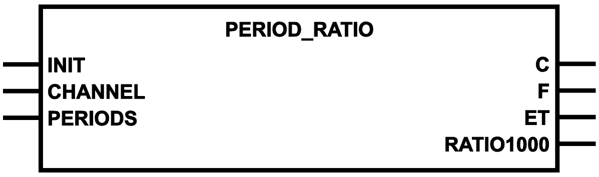
364

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ 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-period ratio is indicated in [‰]. 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-period 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.
For PDM360smart: CR1071: all inputs.

The output RATIO1000 provides the value 0 for a mark-to-period ratio of 100 % (input signal permanently at supply voltage).

Frequencies < 0.05 Hz are no longer clearly indicated!

Parameters of the inputs

10316

Parameter	Data type	Description
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	Number of the fast input channel (0...3) 0...3 for the inputs I24...I27 ℹ For the function block xxx_E (if available) applies: 0...3 for the inputs I24_E...I27_E
PERIODS	BYTE	Number of periods to be compared

8406

! 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

369

Parameter	Data type	Description
C	DWORD	Cycle time of the detected periods in [µs] allowed = 200...10 000 000 = 0xC8...0x989680 (= 10 seconds)
F	REAL	frequency of the input signal in [Hz]
ET	TIME	Time passed since the last change of state on the input (can be used in case of very slow signals)
RATIO1000	WORD	for measuring the interval: Mark-to-space ratio in [‰] Preconditions: • pulse duration ≥ 100 µs • frequency < 5 kHz for other measurements: RATIO1000 = 0

PHASE

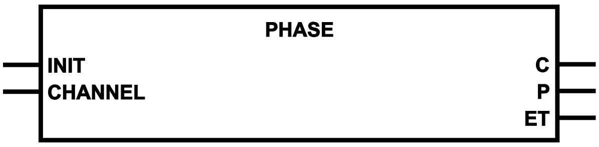
358

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ 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.

❗ For frequencies lower than 15 Hz a cycle period or phase shift of 0 is indicated.

Parameters of the inputs

17810

Parameter	Data type	Description
INIT	BOOL	FALSE ⇨ TRUE (edge): 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 I14 + I15 1 = channel pair 1 = inputs I16 + I17 ℹ For the function block xxx_E (if available) applies: 0 = channel pair 0 = inputs I14_E + I15_E 1 = channel pair 1 = inputs I16_E + I17_E

8406

❗ 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	period duration of the first input's signal of the channel pair in [µs]
P	INT	angle of the phase shaft valid measurement: 1...358 °
ET	TIME	Time elapsed since the last positive edge at the second pulse input of the channel pair

5.2.12 Function elements: PWM functions

Inhalt

13758

Here, you will find **ifm** function blocks that allow you to operate the outputs with Pulse-Width Modulation (PWM).

OCC_TASK

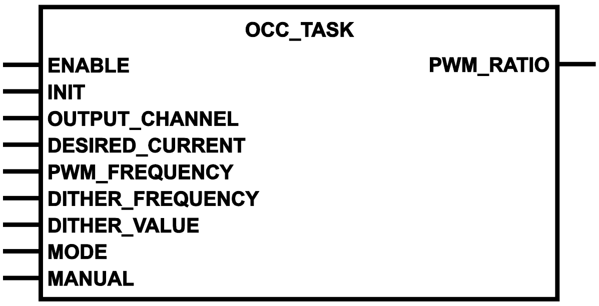
388

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 **i**ntegral and **d**ifferential 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 175).

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.

▶ When defining the parameter DITHER_VALUE make sure that the resulting PWM ratio in the operating range of the loop control remains between 0...100 %:
  - PWM ratio + DITHER_VALUE < 100 % and
  - PWM ratio - DITHER_VALUE > 0 %.

  Outside of this permissible range the current specified in the parameter DESIRED_CURRENT cannot be reached.

Parameters of the inputs

392

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
OUTPUT_CHANNEL	BYTE	Number of the current-controlled output channel (0...7) 0…3 for the outputs Q10...Q13 4...7 for the outputs Q20...Q23 ℹ For the function block xxx_E (if available) applies: 0…3 for the outputs Q10_E...Q13_E 4...7 for the outputs Q20_E...Q23_E
DESIRED_CURRENT	WORD	desired current value of the output in [mA]
PWM_FREQUENCY	WORD	PWM frequency [Hz] for load on input
DITHER_FREQUENCY	WORD	dither frequency in [Hz] value range = 0...FREQUENCY / 2 FREQUENCY / DITHER_FREQUENCY must be even-numbered! The FB increases all other values to the next matching value.
DITHER_VALUE	BYTE	peak-to-peak value of the dither in [%] permissible values = 0...100 = 0x00...0x64
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 closed-loop controller is overwritten (typical: 50)

Parameters of the outputs

393

Parameter	Data type	Description
PWM_RATIO	BYTE	for monitoring purposes: display PWM pulse ratio 0...100%

OUTPUT_CURRENT

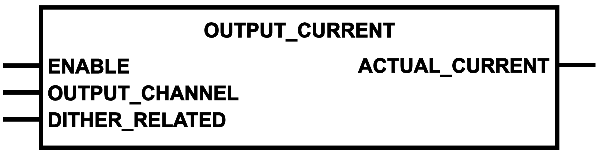
382

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'. (not for CR0133)

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 or as plus switching. 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
OUTPUT_CHANNEL	BYTE	Number of the current-controlled output channel (0...7) 0…3 for the outputs Q10...Q13 4...7 for the outputs Q20...Q23 ℹ For the function block xxx_E (if available) applies: 0…3 for the outputs Q10_E...Q13_E 4...7 for the outputs Q20_E...Q23_E
DITHER_RELATED	BOOL	Current is determined as an average value via... TRUE: one dither period FALSE: one PWM period

Parameters of the outputs

387

Parameter	Data type	Description
ACTUAL_CURRENT	WORD	Output current in [mA]

OUTPUT_CURRENT_CONTROL

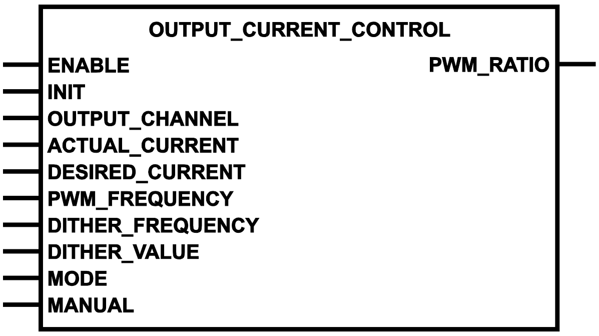
376

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ 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 **i**ntegral and **d**ifferential 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 170).

OUTPUT_CURRENT_CONTROL is based on *PWM* (→ page 175).

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.

► When defining the parameter DITHER_VALUE make sure that the resulting PWM ratio in the operating range of the loop control remains between 0...100 %:
  • PWM ratio + DITHER_VALUE < 100 % and
  • PWM ratio - DITHER_VALUE > 0 %.
  Outside of this permissible range the current specified in the parameter DESIRED_CURRENT cannot be reached.

Parameters of the inputs

380

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
OUTPUT_CHANNEL	BYTE	Number of the current-controlled output channel (0...7) 0…3 for the outputs Q10...Q13 4...7 for the outputs Q20...Q23 ℹ For the function block xxx_E (if available) applies: 0…3 for the outputs Q10_E...Q13_E 4...7 for the outputs Q20_E...Q23_E
ACTUAL_CURRENT	WORD	Actual current on the PWM output in [mA] ► Transfer the output value of *OUTPUT_CURRENT* (→ page 172) to the input ACTUAL_CURRENT.
DESIRED_CURRENT	WORD	desired current value of the output in [mA]
PWM_FREQUENCY	WORD	PWM frequency [Hz] for load on input
DITHER_FREQUENCY	WORD	dither frequency in [Hz] value range = 0...FREQUENCY / 2 FREQUENCY / DITHER_FREQUENCY must be even-numbered! The FB increases all other values to the next matching value.
DITHER_VALUE	BYTE	peak-to-peak value of the dither in [%] permissible values = 0...100 = 0x00...0x64
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 closed-loop controller is overwritten (typical: 50)

Parameters of the outputs

381

Parameter	Data type	Description
PWM_RATIO	BYTE	for monitoring purposes: display PWM pulse ratio 0...100%

PWM

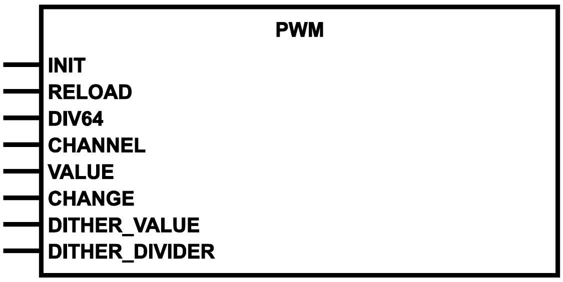
320

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'.

Symbol in CODESYS:



Description

20467

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...11.
>
> For these channels, PWM and *PWM1000* (→ page 181) 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 172)
- or for example using the **ifm** unit EC2049 (series element for current measurement).

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.

> 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	FALSE ⇨ TRUE (edge): 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 177))
DIV64	BOOL	CPU cycle / 64
CHANNEL	BYTE	Current PWM output channel x (0...x, value depends on the device, → data sheet)

VALUE	WORD	Current PWM value
		permissible = 0...RELOAD
		0 = switch-on time 100 %
		RELOAD = switch-on time 0 %

CHANGE	BOOL	TRUE: Adopting new value from ... • VALUE: after the current PMW period • DITHER_VALUE: after the current Dither period FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	WORD	peak-to-peak value of the dither in [‰] permissible values = 0...1 000 = 0000...03E8
DITHER_DIVIDER	WORD	Dither frequency = PWM frequency / DIVIDER * 2

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) or directly as a numerical value in [Hz] (PWM1000). 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 0xFFFF backwards or from 0x0000 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 0x0000 to 0xFFFF or from 0xFFFF to 0x0000), the output is reset and the operation restarts.

If this internal counter shall not operate between 0x0000 and 0xFFFF, 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.

Calculation of the RELOAD value

1531

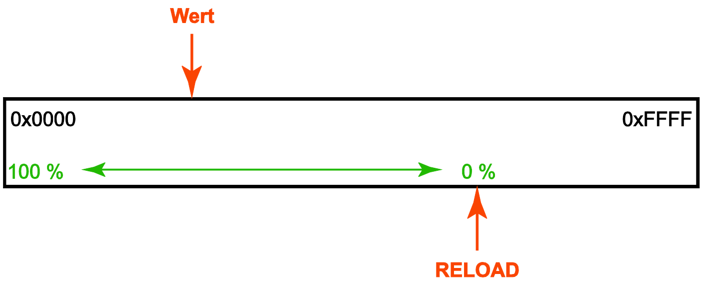


Figure: RELOAD value for the PWM channels 0...3

The RELOAD value of the internal PWM counter is calculated on the basis of the parameter DIV64 and the CPU frequency as follows:

	• CabinetController: CR0303 • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506	• CabinetController: CR0301, CR0302 • SmartController: CR250n • PCB controller: CS0015 • PDM360smart: CR1071
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 0xFFFF.

Calculation examples RELOAD value

1532

• CabinetController: CR0303 • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506		• CabinetController: CR0301, CR0302 • SmartController: CR250n • PCB controller: CS0015 • PDM360smart: CR1071
The PWM frequency shall be 400 Hz. $\frac{20\text{ MHz}}{400\text{ Hz}}$ = 50 000 = 0xC350 = RELOAD		The PWM frequency shall be 200 Hz. $\frac{10\text{ MHz}}{200\text{ Hz}}$ = 50 000 = 0xC350 = RELOAD
Thus the permissible range of the PWM value is the range from 0x0000...0xC350. The comparison value at which the output switches must then be between 0x0000 and 0xC350.		

This results in the following mark-to-space ratios:

Mark-to-space ratio	Switch-on time	Value for mark-to-space ratio
Minimum	0 %	50 000 = 0xC350
Maximum	100 %	0 = 0x0000

Between minimum and maximum triggering 50 000 intermediate values (PWM values) are possible.

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, it is possible to delay the increase by using *PT1* (→ page 207). The ramp function used for PWM is based on the CODESYS library `UTIL.LIB`. This allows a smooth start e.g. for hydraulic systems.

964

NOTE

When installing the **ecomatmobile** DVD "Software, tools and documentation", projects with examples have been stored in the program directory of your PC:
`…\ifm electronic\CoDeSys V…\Projects\DEMO_PLC_DVD_V…` (for controllers) or
`…\ifm electronic\CoDeSys V…\Projects\DEMO_PDM_DVD_V…` (for PDMs).

There you also find projects with examples regarding this subject. It is strongly recommended to follow the shown procedure.

The PWM function of the controller is a hardware function provided by the processor. The PWM function remains set until a hardware reset (power off and on) has been carried out on the controller.

PWM100

332

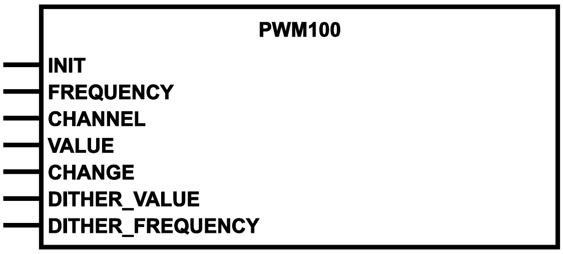
Unit type = function block (FB)

ℹ New **ecomatmobile** controllers only support *PWM1000* (→ page 181).

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

ℹ For the extended side of the ExtendedControllers the FB name ends with '_E'.

Symbol in CODESYS:



Description

20462

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...11.
>
> For these channels, *PWM* (→ page 175) 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 172)
- or for example using the **ifm** unit EC2049 (series element for current measurement).

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.

> ❗ The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.

Parameters of the inputs

20472

Parameter	Data type	Description
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
FREQUENCY	WORD	PWM frequency in [Hz] allowed = 20...250 = 0x0014...0x00FA
CHANNEL	BYTE	Number of the PWM output channel (0...15) 0…3 for the outputs Q10...Q13 4...7 for the outputs Q20...Q23 8...15 for the outputs Q40...Q47 ℹ For the function block xxx_E (if available) applies: 0…3 for the outputs Q10_E...Q13_E 4...7 for the outputs Q20_E...Q23_E • 8...15 for the outputs Q40_E...Q47_E
VALUE	BYTE	current PWM value
CHANGE	BOOL	TRUE: Adopting new value from ... • VALUE: after the current PMW period • DITHER_VALUE: after the current Dither period FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	BYTE	peak-to-peak value of the dither in [%] permissible values = 0...100 = 0x00...0x64
DITHER_FREQUENCY	WORD	dither frequency in [Hz] value range = 0...FREQUENCY / 2 FREQUENCY / DITHER_FREQUENCY must be even-numbered! The FB increases all other values to the next matching value.

PWM1000

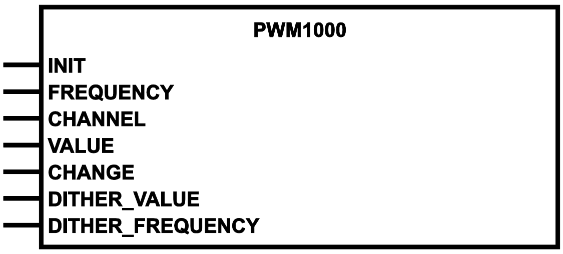
326

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

For the extended side of the ExtendedControllers the FB name ends with '_E'. (not for CR0133)

Symbol in CODESYS:



Description

20466

PWM1000 handles the initialisation and parameter setting of the PWM outputs.

The FB enables a simple use of the PWM FB in the **ecomat*mobile*** 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...11.

For these channels, *PWM* (→ page 175) 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 172)
- or for example using the **ifm** unit EC2049 (series element for current measurement).

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.

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	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
FREQUENCY	WORD	PWM frequency in [Hz] allowed = 20...250 = 0x0014...0x00FA
CHANNEL	BYTE	Number of the PWM output channel (0...15) 0…3 for the outputs Q10...Q13 4...7 for the outputs Q20...Q23 8...15 for the outputs Q40...Q47 ℹ For the function block xxx_E (if available) applies: 0…3 for the outputs Q10_E...Q13_E 4...7 for the outputs Q20_E...Q23_E • 8...15 for the outputs Q40_E...Q47_E
VALUE	WORD	PWM value (mark-to-space ratio) in [‰] allowed = 0...1 000 = 0x0000...0x03E8 Values > 1 000 are regarded as = 1 000
CHANGE	BOOL	TRUE: adoption of the new value of ... • FREQUENCY: after the current PWM period • VALUE: after the current PWM period • DITHER_VALUE: after the current dither period • DITHER_FREQUENCY: after the current dither period FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	WORD	peak-to-peak value of the dither in [‰] permissible values = 0...1 000 = 0000...03E8
DITHER_FREQUENCY	WORD	dither frequency in [Hz] value range = 0...FREQUENCY / 2 FREQUENCY / DITHER_FREQUENCY must be even-numbered! The FB increases all other values to the next matching value.

5.2.13 Function elements: hydraulic control

Inhalt

19540

The library `ifm_hydraulic_16bitS05_Vxxyyzz.LIB` contains the following function blocks:

CONTROL_OCC (→ page 184)	OCC = Output Current Control Scales the input value [WORD] to an indicated current range
JOYSTICK_0 (→ page 187)	Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0... 1000
JOYSTICK_1 (→ page 190)	Scales signals [INT] from a joystick D standardised to 0... 1000
JOYSTICK_2 (→ page 194)	Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation
NORM_HYDRAULIC (→ page 197)	Normalises a value [DINT] within defined limits to a value with new limits

The following function blocks are needed from the library `UTIL.Lib` (in the CODESYS package):

- RAMP_INT
- CHARCURVE

These function blocks are automatically called and configured by the function blocks of the hydraulics library.

The following function blocks are needed from the library:`ifm_CR0200_Vxxyyzz.LIB`

OUTPUT_CURRENT (→ page 172)	Measures the current (average via dither period) on an output channel
OUTPUT_CURRENT_CONTROL (→ page 173)	Current controller for a PWMi output channel

These function blocks are automatically called and configured by the function blocks of the hydraulics library.

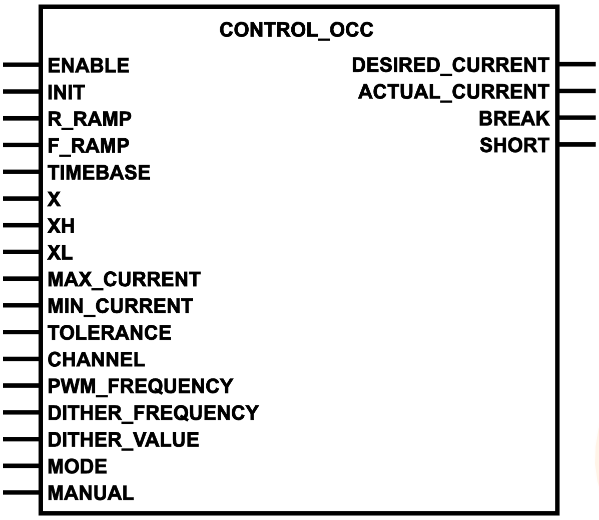
CONTROL_OCC

6245

Unit type = function block (FB)

Unit is contained in the library `ifm_HYDRAULIC_16bitOS05_Vxxyyzz.Lib`

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 173) and *OUTPUT_CURRENT* (→ page 172) from the library `ifm_CR0200_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 **i**ntegral and **d**ifferential 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.

ℹ The 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
INIT	BOOL	FALSE ⇨ TRUE (edge): unit is initialised FALSE: during further processing of the program
R_RAMP	INT	Rising edge of the ramp in [increments/PLC cycle] or iIncrements/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] ❗ Fast controllers have very short cycle times! otherwise = rising/falling edge in [increments/TIMEBASE]
X	WORD	input value
XH	WORD	Upper limit of input value range [increments]
XL	WORD	Lower limit of input value range [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	Number of the current-controlled output channel (0...7) 0...3 for the outputs Q10...Q13 4...7 for the outputs Q20...Q23
PWM_FREQUENCY	WORD	PWM frequency [Hz] for load on input
DITHER_FREQUENCY	WORD	dither frequency in [Hz] value range = 0...FREQUENCY / 2 FREQUENCY / DITHER_FREQUENCY must be even-numbered! The FB increases all other values to the next matching value.
DITHER_VALUE	BYTE	peak-to-peak value of the dither in [%] permissible values = 0...100 = 0x00...0x64
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 closed-loop 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	Output current in [mA]
BREAK	BOOL	Error: cable interrupted at output
SHORT	BOOL	Error: short circuit in cable at output

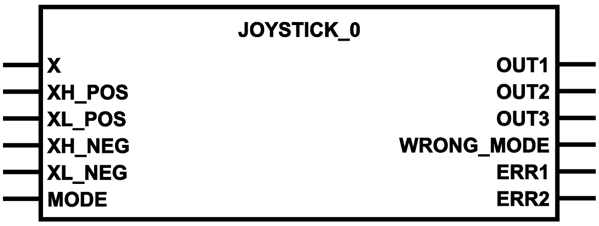
JOYSTICK_0

13224

Unit type = function block (FB)

Unit is contained in the library `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib`

Symbol in CODESYS:



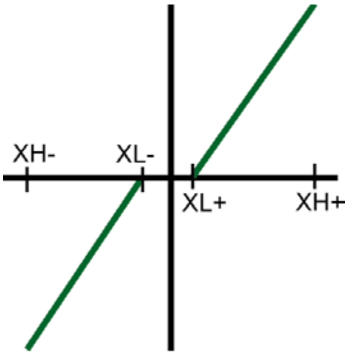
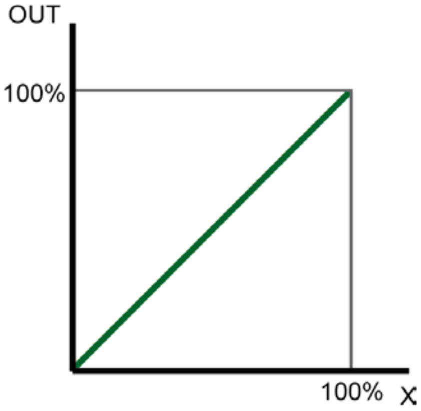
Description

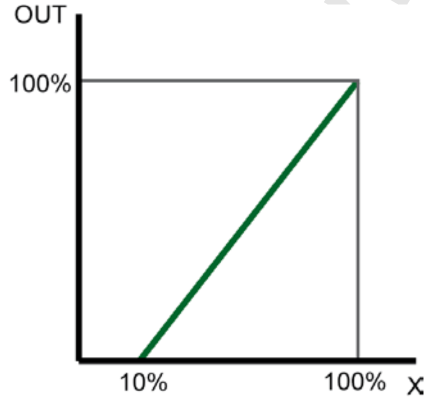
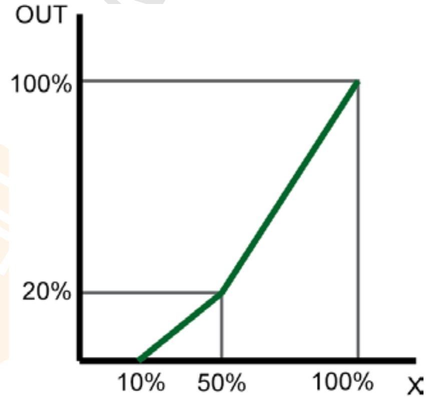
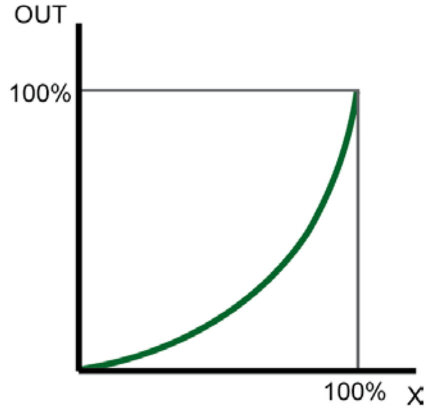
432

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
  ❗ Fast Controllers have a very short cycle time!
- Falling edge of the ramp = no ramp

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.	
Mode 0: characteristic curve linear for the range XL to XH	

Mode 1: Characteristic curve linear with dead band Values fixed to: Dead band: 0…10% of 1000 increments	
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	
Characteristic curve mode 3: Curve rising (line is fixed)	

Parameters of the inputs

433

Parameter	Data type	Description
X	INT	Input value [increments]
XH_POS	INT	Max. preset value positive direction [increments] (negative values also permissible)
XL_POS	INT	Min. preset value positive direction [increments] (negative values also permissible)
XH_NEG	INT	Max. preset value negative direction [increments] (negative values also permissible)
XL_NEG	INT	Min. preset value negative direction [increments] (negative values also permissible)
MODE	BYTE	Mode selection characteristic curve: 0 = linear (X|OUT = 0|0 ... 1000|1000) 1 = linear with dead band (X|OUT = 0|0 ... 100|0 ... 1000|1000) 2 = 2-step linear with dead band (X|OUT = 0|0 ... 100|0 ... 500|200 ... 1000|1000) 3 = curve rising (line is fixed)

Parameters of the outputs

6252

Parameter	Data type	Description
OUT1	WORD	Standardised output value: 0...1000 increments e.g. for valve left
OUT2	WORD	Standardised output value: 0...1000 increments e.g. for valve right
OUT3	INT	Standardised output value -1000...0...1000 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 (referred to the internally used function blocks CHARCURVE and RAMP_INT from util.lib) (possible messages → following table)
ERR2	BYTE	Error code for falling edge (referred to the internally used function blocks CHARCURVE and RAMP_INT from util.lib) (possible messages → following table)

Possible results for ERR1 and ERR2:

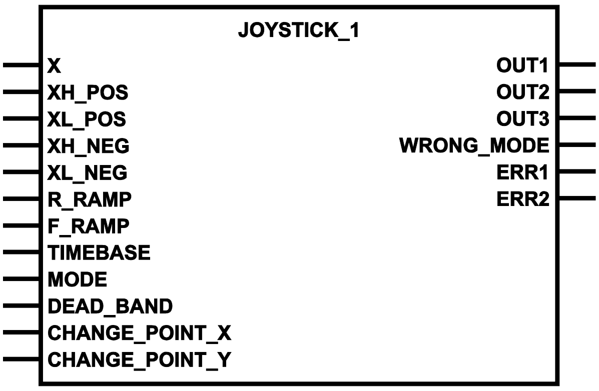
Value dec | hex		Description
0	00	no error
1	01	Error in array: wrong sequence
2	02	Error: Input value IN is not contained in value range of array
4	04	Error: invalid number N for array

JOYSTICK_1

13227

Unit type = function block (FB)
Unit is contained in the library `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib`

Symbol in CODESYS:

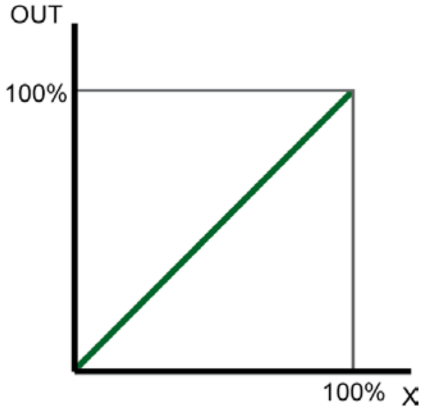
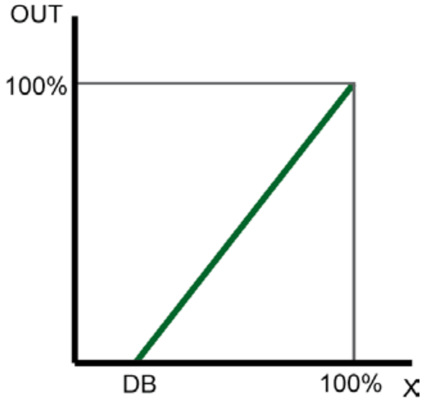


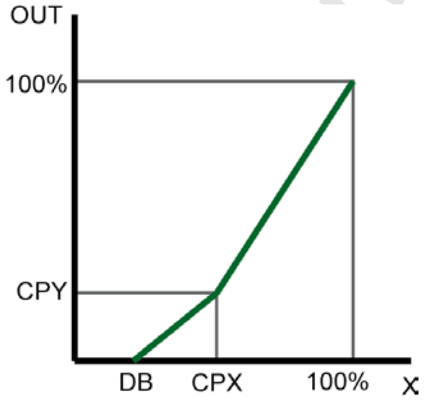
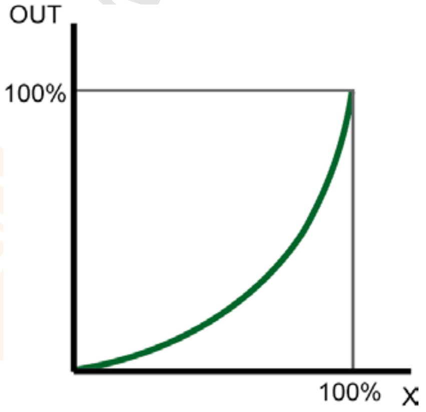
Description

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):

Mode 0: Linear characteristic curve 100 % = 1000 increments	
Mode 1: Characteristic curve linear with dead band Value for the dead band (DB) can be set in % of 1000 increments 100 % = 1000 increments DB = Dead_Band	

Mode 2: 2-step linear characteristic curve with dead band Values can be configured to: Dead band: 0…DB in % of 1000 increments Step: X = CPX in % of 1000 increments Y= CPY in % of 1000 increments 100 % = 1000 increments DB = Dead_Band CPX = Change_Point_X CPY = Change_Point_Y	
Characteristic curve mode 3: Curve rising (line is fixed)	

Parameters of the inputs

6256

Parameter	Data type	Description
X	INT	Input value [increments]
XH_POS	INT	Max. preset value positive direction [increments] (negative values also permissible)
XL_POS	INT	Min. preset value positive direction [increments] (negative values also permissible)
XH_NEG	INT	Max. preset value negative direction [increments] (negative values also permissible)
XL_NEG	INT	Min. preset value negative direction [increments] (negative values also permissible)
R_RAMP	INT	Rising edge of the ramp in [increments/PLC cycle] 0 = no ramp
F_RAMP	INT	Falling edge of the ramp in [increments/PLC cycle] 0 = no ramp
TIMEBASE	TIME	Reference for rising and falling edge of the ramp: t#0s = rising/falling edge in [increments/PLC cycle] ❗ Fast controllers have very short cycle times! otherwise = rising/falling edge in [increments/TIMEBASE]
MODE	BYTE	Mode selection characteristic curve: 0 = linear (X|OUT = 0|0 ... 1000|1000) 1 = linear with dead band (X|OUT = 0|0 ... DB|0 ... 1000|1000) 2 = 2-step linear with dead band (X|OUT = 0|0 ... DB|0 ... CPX|CPY ... 1000|1000) 3 = curve rising (line is fixed)
DEAD_BAND	BYTE	Adjustable dead band 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: 0…1000 increments e.g. for valve left
OUT2	WORD	Standardised output value: 0…1000 increments e.g. for valve right
OUT3	INT	Standardised output value -1000…0…1000 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 (referred to the internally used function blocks CHARCURVE and RAMP_INT from `util.lib`) (possible messages → following table)
ERR2	BYTE	Error code for falling edge (referred to the internally used function blocks CHARCURVE and RAMP_INT from `util.lib`) (possible messages → following table)

Possible results for ERR1 and ERR2:

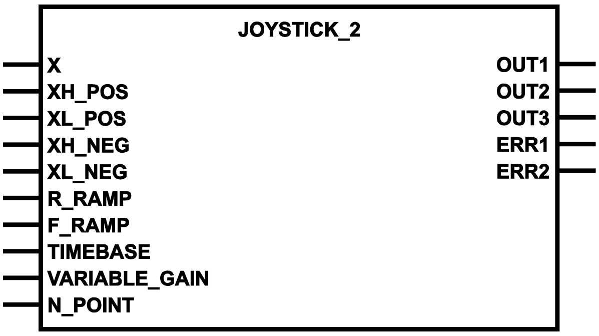
Value dec | hex		Description
0	00	no error
1	01	Error in array: wrong sequence
2	02	Error: Input value IN is not contained in value range of array
4	04	Error: invalid number N for array

JOYSTICK_2

13228

Unit type = function block (FB)
Unit is contained in the library `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib`

Symbol in CODESYS:

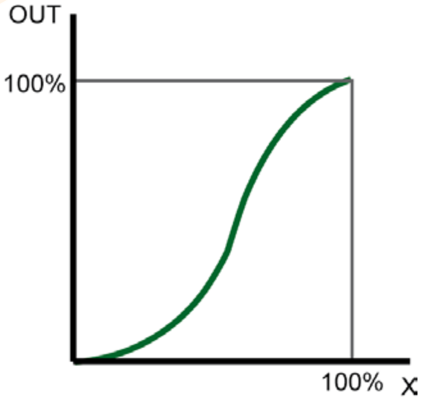


Description

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):

Characteristic curve freely configurable	

Parameters of the inputs

6261

Parameter	Data type	Description
X	INT	Input value [increments]
XH_POS	INT	Max. preset value positive direction [increments] (negative values also permissible)
XL_POS	INT	Min. preset value positive direction [increments] (negative values also permissible)
XH_NEG	INT	Max. preset value negative direction [increments] (negative values also permissible)
XL_NEG	INT	Min. preset value negative direction [increments] (negative values also permissible)
R_RAMP	INT	Rising edge of the ramp in [increments/PLC cycle] 0 = no ramp
F_RAMP	INT	Falling edge of the ramp in [increments/PLC cycle] 0 = no ramp
TIMEBASE	TIME	Reference for rising and falling edge of the ramp: t#0s = rising/falling edge in [increments/PLC cycle] ❗ Fast controllers have very short cycle times! otherwise = 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: 0…1000 increments e.g. for valve left
OUT2	WORD	Standardised output value: 0…1000 increments e.g. for valve right
OUT3	INT	Standardised output value -1000…0…1000 increments e.g. for valve on output module (e.g. CR2011 or CR2031)
ERR1	BYTE	Error code for rising edge (referred to the internally used function blocks CHARCURVE and RAMP_INT from `util.lib`) (possible messages → following table)
ERR2	BYTE	Error code for falling edge (referred to the internally used function blocks CHARCURVE and RAMP_INT from `util.lib`) (possible messages → following table)

Possible results for ERR1 and ERR2:

Value dec | hex		Description
0	00	no error
1	01	Error in array: wrong sequence
2	02	Error: Input value IN is not contained in value range of array
4	04	Error: invalid number N for array

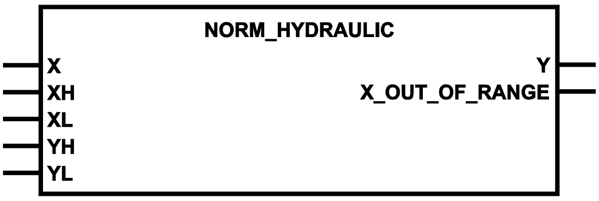
NORM_HYDRAULIC

13232

Unit type = function block (FB)

Unit is contained in the library `ifm_hydraulic_16bitOS05_Vxxyyzz.Lib`

Symbol in CODESYS:



Description

397

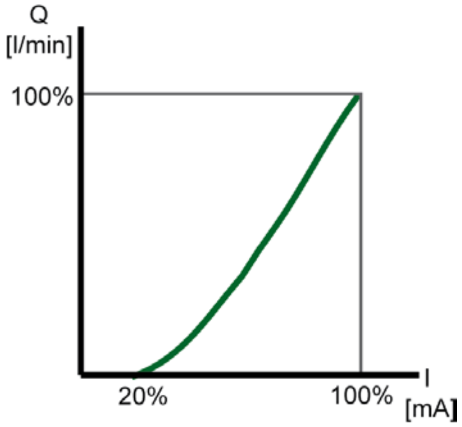
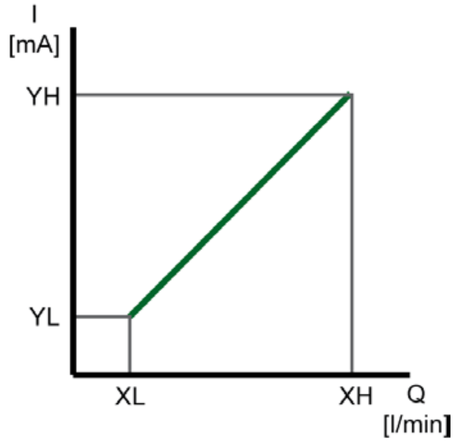
NORM_HYDRAULIC standardises input values with fixed limits to values with new limits.

ℹ This function block corresponds to NORM_DINT from the CODESYS library `UTIL.Lib`.

The function block standardises a value of type DINT, which is 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 is outside the limits of XL…XH, the error message will be 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 function block	

Parameters of the inputs

398

Parameter	Data type	Description
X	DINT	current input value
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	output value
X_OUT_OF_RANGE	BOOL	Error: X is beyond the limits of 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.

5.2.14 Function elements: controllers

Inhalt

1634

The section below describes in detail the units that are provided for set-up by software controllers in the **ecomat*mobile*** device. The units can also be used as basis for the development of your own control functions.

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 KP is increased until the control deviation and the adjustment deviation perform steady oscillation at a constant amplitude at KP = $KP_{critical}$. 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.

KP should be equal to KD.

Idealised setting of the controlled system:

Control unit	KP = KD	TN	TV
P	2.0 • $KP_{critical}$	—	—
PI	2.2 • $KP_{critical}$	0.83 • $T_{critical}$	—
PID	1.7 • $KP_{critical}$	0.50 • $T_{critical}$	0.125 • $T_{critical}$

❗ For this setting process it has to be noted that the controlled system is not harmed by the oscillation generated. For sensitive controlled systems KP must only be increased to a value at which no oscillation occurs.

Damping of overshoot

1629

To dampen overshoot *PT1* (→ page 207) (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 TN of the controller.

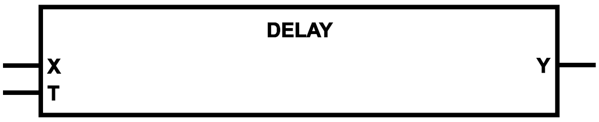
DELAY

585

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

588

DELAY delays the output of the input value by the time T (dead-time element).

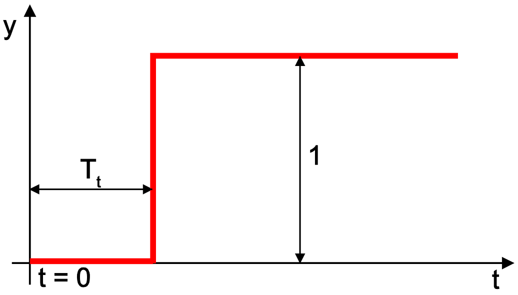


Figure: Time characteristics of DELAY

The dead time is influenced by the duration of the PLC cycle.
The dead time my not exceed 100 • PLC cycle time (memory limit!).
In case a longer delay is set, the resolution of the values at the output of the FB will be poorer, which may cause that short value changes will be lost.

❗ To ensure that the FB works correctly: FB must be called in each cycle.

Parameters of the inputs

589

Parameter	Data type	Description
X	WORD	input value
T	TIME	Delay time (dead time) allowed: 0...100 • cycle time

Parameters of the outputs

590

Parameter	Data type	Description
Y	WORD	input value, delayed by the time T

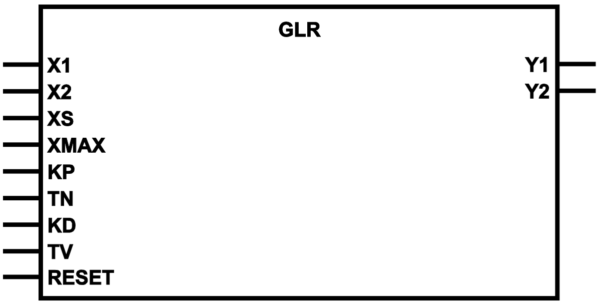
GLR

531

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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	preset value
XMAX	WORD	maximum preset value
KP	Byte	constant of the proportional component (/10) (positive values only!)
TN	TIME	integral action time (integral component)
KD	BYTE	differential component (/10) (positive values only!)
TV	TIME	derivative action time (differential component)
RESET	BOOL	TRUE: reset the function element FALSE: function element is not executed

Parameters of the outputs

536

Parameter	Data type	Description
Y1	WORD	manipulated variable channel 1
Y2	WORD	manipulated variable channel 2

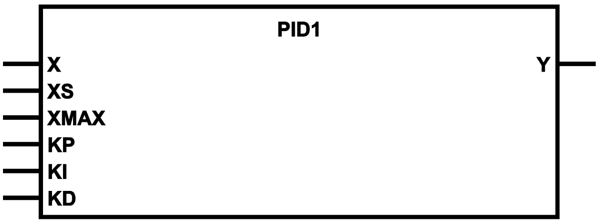
PID1

351

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

354

PID1 handles a PID controller.

The change of the manipulated variable of a PID controller has a **p**roportional, **i**ntegral and **d**ifferential 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 > XS, the manipulated variable is increased.
If X < XS, the manipulated variable is reduced.

The manipulated variable Y has the following time characteristics:

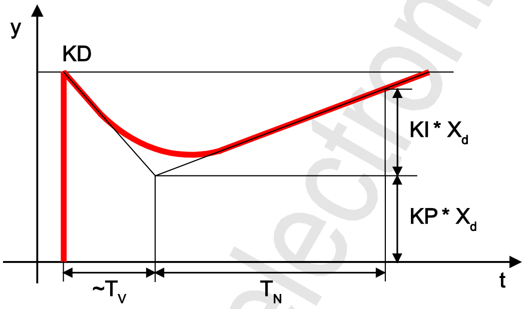


Figure: Typical step response of a PID controller

Parameters of the inputs

355

Parameter	Data type	Description
X	WORD	input value
XS	WORD	preset value
XMAX	WORD	maximum preset value
KP	BYTE	proportional component of the output signal
KI	BYTE	integral component of the output signal
KD	BYTE	differential component of the output signal

Parameters of the outputs

356

Parameter	Data type	Description
Y	WORD	Manipulated variable (0...1000 ‰)

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.

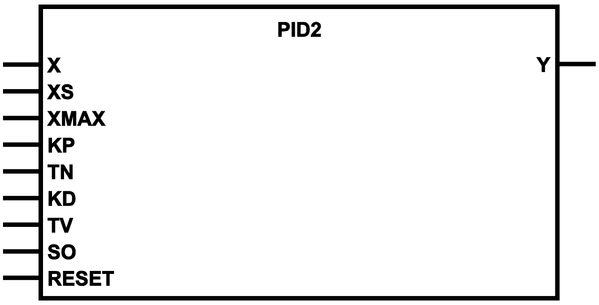
PID2

9167

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 **p**roportional, **i**ntegral and **d**ifferential 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 > XS, the manipulated variable is increased.
If X < XS, 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.

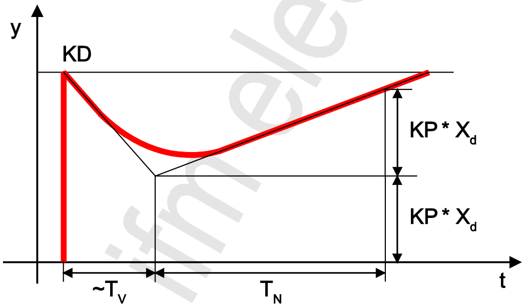


Figure: Typical step response of a PID controller

Parameters of the inputs

348

Parameter	Data type	Description
X	WORD	input value
XS	WORD	preset value
XMAX	WORD	maximum preset value
KP	Byte	constant of the proportional component (/10) (positive values only!)
TN	TIME	integral action time (integral component)
KD	BYTE	differential component (/10) (positive values only!)
TV	TIME	derivative action time (differential component)
SO	BOOL	TRUE: self-optimisation active FALSE: self-optimisation not active
RESET	BOOL	TRUE: reset the function element FALSE: function element is not executed

Parameters of the outputs

349

Parameter	Data type	Description
Y	WORD	Manipulated variable (0...1000 ‰)

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 **d**ifferential 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 ≥ 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.

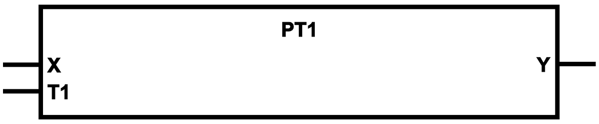
PT1

338

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 of the FB can become instable if T1 is shorter than the SPS cycle time.

The output variable Y of the low-pass filter has the following time characteristics (unit step):

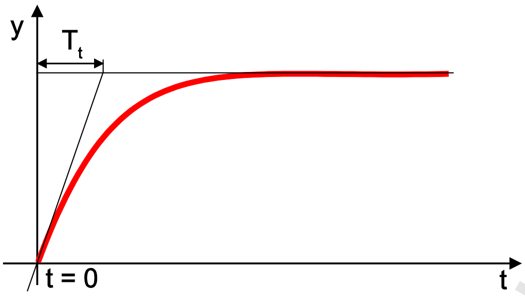


Figure: Time characteristics of PT1

Parameters of the inputs

342

Parameter	Data type	Description
X	INT	Input value [increments]
T1	TIME	Delay time (time constant)

Parameters of the outputs

343

Parameter	Data type	Description
Y	INT	output value

5.2.15 Function elements: software reset

Inhalt

1594

Using this FB the control can be restarted via an order in the application program.

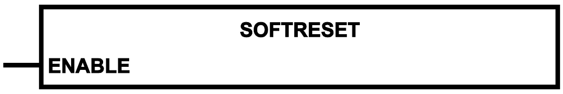
SOFTRESET

260

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

263

SOFTRESET leads to a complete reboot of the device.

The FB can for example be used in conjunction with CANopen if a node reset is to be carried out. FB SOFTRESET executes an immediate reboot of the controller. The current cycle is not completed.

Before reboot, the retain variables are stored.
The reboot is logged in the error memory.

> ❗ 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: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified

5.2.16 Function elements: measuring / setting of time

Inhalt

1601

Using the following function blocks of **ifm electronic** you can...

- measure time and evaluate it in the application program,
- change time values, if required.

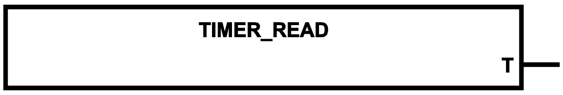
TIMER_READ

236

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

239

TIMER_READ reads the current system time.

When the supply voltage is applied, the device 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.

> ❗ The system timer goes up to 0xFFFF FFFF at the maximum (corresponds to 49d 17h 2min 47s 295ms) and then starts again from 0.

Parameters of the outputs

241

Parameter	Data type	Description
T	TIME	Current system time [ms]

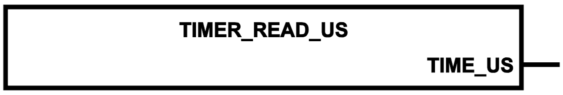
TIMER_READ_US

657

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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.

ℹ 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 = 1h 11min 34s 967ms 295µs

Parameters of the outputs

662

Parameter	Data type	Description
TIME_US	DWORD	current system time [µs]

5.2.17 Function elements: saving, reading and converting data in the memory

Inhalt

13795

Storage types for data backup

13805

The device provides the following memory types:

Flash memory

13803

Properties:

- non-volatile memory
- writing is relatively slow and only block by block
- before re-writing, memory content must be deleted
- fast reading
- limited writing and reading frequency
- really useful only for storing large data quantities
- secure data with FLASHWRITE
- read data with FLASHREAD

FRAM memory

13802

FRAM indicates here all kinds of non-volatile and fast memories.

Properties:

- fast writing and reading
- unlimited writing and reading frequency
- any memory area can be selected
- secure data with FRAMWRITE
- read data with FRAMREAD

Automatic saving of data

Inhalt

2347

The **ecomat*mobile*** controllers allow to save data (BOOL, BYTE, WORD, DWORD) non-volatilely (= 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 defined as RETAIN variables (→ CODESYS).

A distinction is made between variables declared as RETAIN and variables in the flag area which can be configured as a remanent block with *MEMORY_RETAIN_PARAM* (→ page 215).
Details → chapter *Variables* (→ page 73)

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).

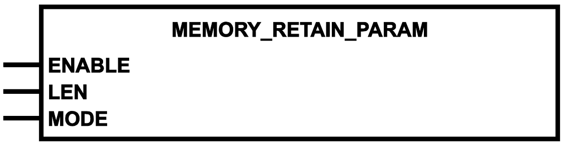
MEMORY_RETAIN_PARAM

2372

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

2374

MEMORY_RETAIN_PARAM determines the remanent data behaviour for various events. Variables declared as VAR_RETAIN in CODESYS have a remanent behaviour from the outset.

Remanent data keep their value (as the variables declared as VAR_RETAIN) after an uncontrolled termination as well as after normal switch off and on of the controller. After a restart the program continues to work with the stored values.

For groups of events that can be selected (with MODE), this function block determines how many (LEN) data bytes (from flag byte %MB0) shall have retain behaviour even if they have not been explicitly declared as VAR_RETAIN.

Event	MODE = 0	MODE = 1	MODE = 2	MODE = 3
Power OFF ⇨ ON	Data is newly initialised	Data is remanent	Data is remanent	Data is remanent
Soft reset	Data is newly initialised	Data is remanent	Data is remanent	Data is remanent
Cold reset	Data is newly initialised	Data is newly initialised	Data is remanent	Data is remanent
Reset default	Data is newly initialised	Data is newly initialised	Data is remanent	Data is remanent
Load application program	Data is newly initialised	Data is newly initialised	Data is remanent	Data is remanent
Load runtime system	Data is newly initialised	Data is newly initialised	Data is newly initialised	Data is remanent

If MODE = 0, only those data have retain behaviour as with MODE=1 which have been explicitly declared as VAR_RETAIN.

If the FB is never called, the flag bytes act according to MODE = 0. The flag bytes which are above the configured area act according to MODE = 0, too.

Once a configuration has been made, it remains on the device even if the application or the runtime system is reloaded.

Parameters of the inputs

2375

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
LEN	WORD	Number of data bytes from flag address %MB0 onwards to show remanent behaviour allowed = 0...4 096 = 0x0...0x1000 LEN > 4 096 will be corrected automatically to LEN = 4 096
MODE	BYTE	Events for which these variables shall have retain behaviour (0...3; → table above) For MODE > 3 the last valid setting will remain

Manual data storage

Inhalt

13801

Besides the possibility to store data automatically, user data can be stored manually, via function block calls, in integrated memories from where they can also be read.

ℹ By means of the storage partitioning (→ chapter *Available memory* (→ page 16)) the programmer can find out which memory area is available.

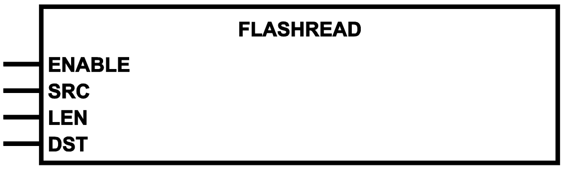
FLASHREAD

561

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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 contents are read completely during the cycle in which the FB is called up.

► Please make sure that the target memory area in the RAM is sufficient.

► To the destination address DST applies:
❗ Determine the address by means of the operator ADR and assigne it to the FB!

Parameters of the inputs

565

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
SRC	WORD	relative source start address in the memory permissible = 0...65 535 = 0x0000...0xFFFF
LEN	WORD	number of data bytes permissible = 0...65 535 = 0x0000...0xFFFF
DST	DWORD	start address of the destination variable ❗ Determine the address by means of the operator ADR and assigne it to the FB!

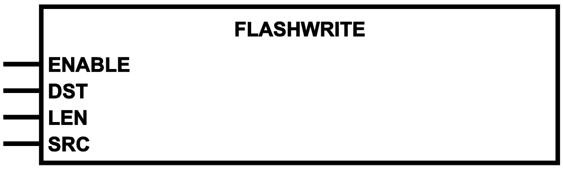
FLASHWRITE

555

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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.

Using this FB, large data volumes are to be stored during set-up, to which there is only read access in the process.

▶ If a page has already been written (even if only partly), the entire flash memory area needs to be deleted before new write access to this page. This is done by write access to the address 0.

▶ Never write to a page several times! Always delete everything first!
Otherwise, traps or watchdog errors occur.

▶ ❗ Do not delete the flash memory area more often than 100 times. Otherwise, the data consistency in other flash memory areas is no longer guaranteed.

▶ During each SPS cycle, FLASHWRITE may only be started <u>once</u>!

▶ To the destination address DST applies:
❗ Determine the address by means of the operator ADR and assigne it to the FB!

> 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.

❗ If start address SRC is outside the permissible range: no data transfer!

Parameters of the inputs

559

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
DST	WORD	relative destination start address in the memory permissible = 0...65 535 = 0x0000...0xFFFF
LEN	WORD	number of data bytes permissible = 0...65 535 = 0x0000...0xFFFF
SRC	DWORD	start address of the source variables ❗ Determine the address by means of the operator ADR and assigne it to the FB!

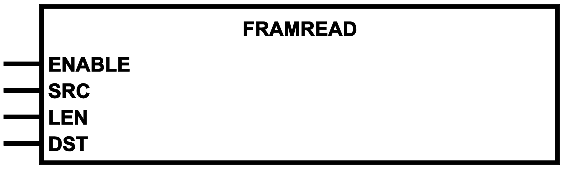
FRAMREAD

549

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

552

FRAMREAD enables quick reading of different data types directly from the FRAM memory [1]).

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.
If the FRAM memory area were to be exceeded by the indicated number of bytes, only the data up to the end of the FRAM memory area will be read.

► To the destination address DST applies:
❗ Determine the address by means of the operator ADR and assigne it to the FB!

[1]) FRAM indicates here all kinds of non-volatile and fast memories.

Parameters of the inputs

553

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
SRC	WORD	relative source start address in the memory zulässig = 0...1 023 = 0x0000...0x03FF
LEN	WORD	number of data bytes
DST	DWORD	start address of the destination variable ❗ Determine the address by means of the operator ADR and assigne it to the FB!

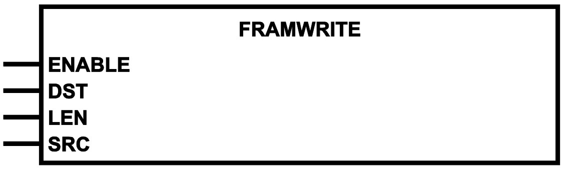
FRAMWRITE

543

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

546

FRAMWRITE enables the quick writing of different data types directly into the FRAM memory [1]).

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.
If the FRAM memory area were to be exceeded by the indicated number of bytes, only the data up to the end of the FRAM memory area will be written.

► To the source address SRC applies:
  ❗ Determine the address by means of the operator ADR and assigne it to the FB!

❗ If the target address DST is outside the permissible range: no data transfer!

[1]) FRAM indicates here all kinds of non-volatile and fast memories.

Parameters of the inputs

547

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
DST	WORD	relative destination start address in the memory permissible = 0...1 023 = 0x0000...0x03FF
LEN	WORD	number of data bytes
SRC	DWORD	start address of the source variables ❗ Determine the address by means of the operator ADR and assigne it to the FB!

MEMCPY

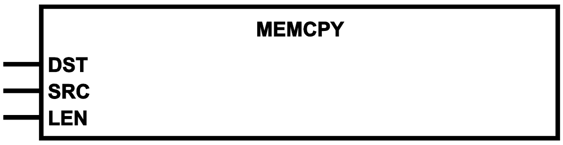
409

= memory copy

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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.

► To the addresses SRC and DST apply:
❗ Determine the address by means of the operator ADR and assigne it to the FB!

> 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 variable.

Parameters of the inputs

413

Parameter	Data type	Description
DST	DWORD	destination address ❗ Determine the address by means of the operator ADR and assigne it to the FB!
SRC	DWORD	Start address in source memory ❗ Determine the address by means of the operator ADR and assigne it to the FB!
LEN	WORD	number (≥ 1) of the data bytes to be transmitted

5.2.18 Function elements: data access and data check

Inhalt

1598

The FBs described in this chapter control the data access and enable a data check.

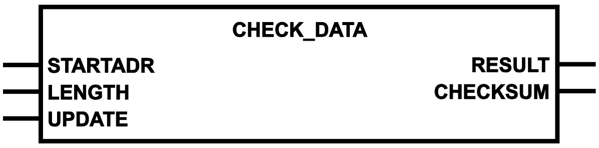
CHECK_DATA

603

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

606

CHECK_DATA generates a checksum (CRC) for a configurable memory area and checks the data of the memory area for undesired changes.

► Create a separate instance of the function block for each memory area to be monitored.

► ❗ Determine the address by means of the operator ADR and assigne it to the FB!

► In addition, indicate the number of data bytes LENGTH (length from the STARTADR).

Undesired change: Error!
If input UPDATE = FALSE and data in the memory is changed inadvertently, then RESULT = FALSE. The result can then be used for further actions (e.g. deactivation of the outputs).

Desired change:
Data changes in the memory (e.g. of the application program or **ecomat*mobile*** 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.

Parameters of the inputs

607

Parameter	Data type	Description
STARTADR	DINT	start address of the monitored data memory (WORD address as from %MW0) ❗ Determine the address by means of the operator ADR and assigne it to the FB!
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

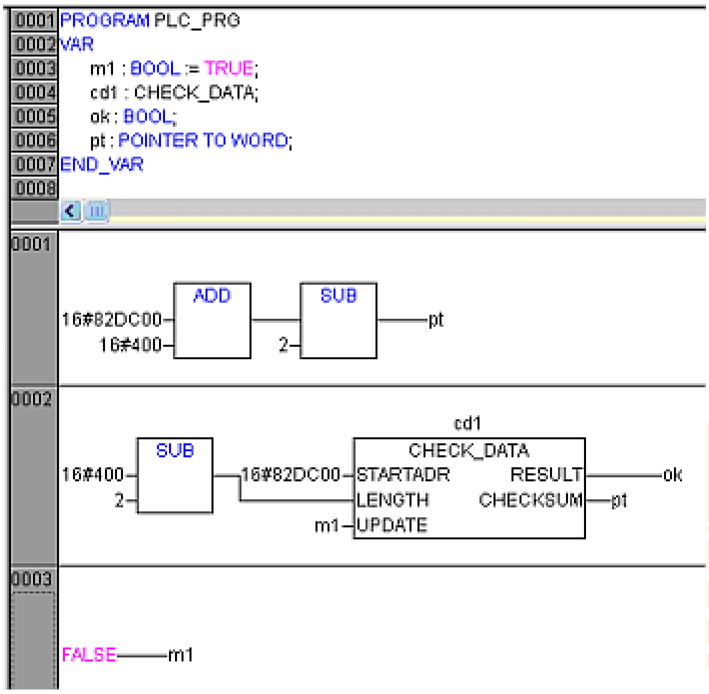
608

Parameter	Data type	Description
RESULT	BOOL	TRUE: CRC checksum ok FALSE: CRC checksum faulty (data modified)
CHECKSUM	DWORD	current CRC checksum

Example: CHECK_DATA

4168

In the following example the program determines the checksum and stores it in the RAM via pointer pt:



```
PROGRAM PLC_PRG
VAR
    m1 : BOOL := TRUE;
    cd1 : CHECK_DATA;
    ok : BOOL;
    pt : POINTER TO WORD;
END_VAR
```

❗ The method shown here is not suited for the flash memory.

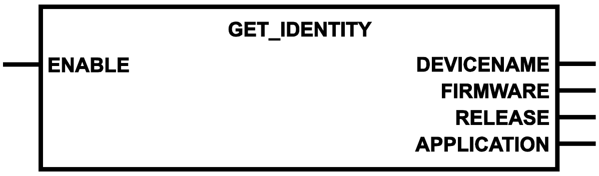
GET_IDENTITY

2212

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

2344

GET_IDENTITY reads the specific identifications stored in the device:

- hardware name and hardware version of the device
- name of the runtime system in the device
- version and revision no. of the runtime system in the device
- name of the application (has previously been saved by means of *SET_IDENTITY* (→ page 227))

Parameters of the inputs

2609

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified

Parameters of the outputs

2610

Parameter	Data type	Description
DEVICENAME	STRING(31)	hardware name as a string of max. 31 characters, e.g.: "CR0403"
FIRMWARE	STRING(31)	Name of the runtime system in the device as character string of max. 31 characters e.g.: "CR0403"
RELEASE	STRING(31)	software version as a character string of max. 31 characters
APPLICATION	STRING(79)	Name of the application as a string of max. 79 characters e.g.: "Crane1704"

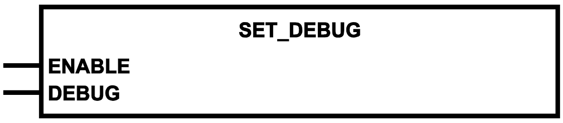
SET_DEBUG

290

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

293

SET_DEBUG handles the DEBUG mode without active test input (→ chapter *TEST mode* (→ page 47)).

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).

> ❗ In this operating mode a software download is not possible because the test input is not connected to supply voltage. Only read access is possible.

Parameters of the inputs

294

Parameter	Data type	Description
ENABLE	BOOL	TRUE: execute this function element FALSE: unit is not executed > Function block inputs are not active > Function block outputs are not specified
DEBUG	BOOL	TRUE: debugging via the interfaces possible FALSE: debugging via the interfaces not possible

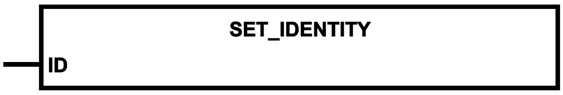
SET_IDENTITY

284

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

Symbol in CODESYS:



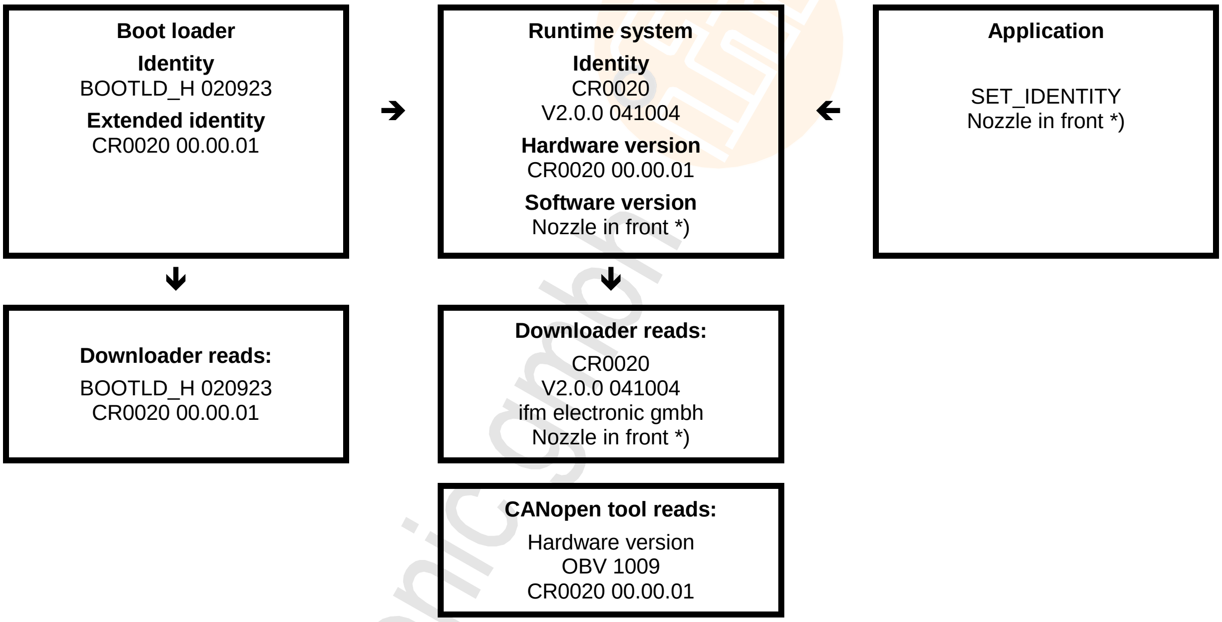
Description

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):



*) ℹ 'Nozzle in front' is substitutionally here for a customised text.

Parameters of the inputs

288

Parameter	Data type	Description
ID	STRING(80)	Any text with a maximum length of 80 characters

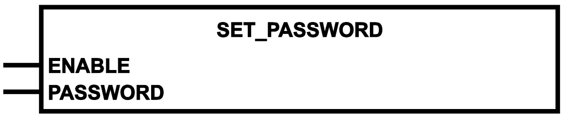
SET_PASSWORD

266

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0200_Vxxyyzz.LIB`

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.

A new password can be set only after resetting the previous password.

❗ The password is reset when loading a new application program.

Parameters of the inputs

270

Parameter	Data type	Description
ENABLE	BOOL	TRUE (nur 1 Zyklus lang): use new Parameter FALSE: unit is not executed
PASSWORD	STRING(16)	password If PASSWORD = "", than access is possible without enter of a password

6 Diagnosis and error handling

Inhalt

19598

The runtime-system (RTS) checks the device by internal error checks:

- during the boot phase (reset phase)
- during executing the application program

→ chapter ***Operating states*** (→ page 43)

In so doing a high operating reliability is provided, as much as possible.

6.1 Diagnosis

19601

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.

6.2 Fault

19602

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".

6.3 Reaction in case of an error

19603
12217

When errors are detected the system flag ERROR can also be set in the application program. Thus, in case of a fault, the controller reacts as follows:

> the operation LED lights red,

> the output relays switch off,

> the outputs protected by the relays are disconnected from power,

> the logic signal states of the outputs remain unchanged.

NOTE

If the outputs are switched off by the relays, the logic signal states remain unchanged.

► The programmer must evaluate the ERROR bit and thus also reset the output logic in case of a fault.

Complete list of the device-specific error codes and diagnostic messages
→ chapter *System flags* (→ page 232).

6.4 Relay: important notes!

1446

Using the logic function via the system flag RELAIS or RELAY_CLAMP_15 (→ chapter *Latching* (→ page 17)) 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!

6.5 Response to system errors

2258

> ❗The programmer has the sole responsibility for the safe processing of data in the application software.

► Process the specific error flags in the application program!
An error description is provided via the error flag.
These error flags can be further processed if necessary.

In case of serious errors the system flag bit ERROR can additionally be set.
At the same time, ERROR = TRUE leads to the following:

- set all relevant outputs to FALSE via the application program,
- the operation LED lights red,
- the ERROR output is set to FALSE and
- the output relays switch off.
- So the outputs protected via these relays are switched off.

After analysis and elimination of the error cause:

► as a general rule, reset all error flags via the application program.
Without explicit reset of the error flags the flags remain set with the corresponding effect on the application program.

6.6 CAN / CANopen: errors and error handling

19604

→ System manual "Know-How ecomat*mobile*"
→ chapter *CAN / CANopen: errors and error handling*

7 Annex

Inhalt

1664

Additionally to the indications in the data sheets you find summary tables in the annex.

7.1 System flags

Inhalt

12167

❗ The addresses of the system flags can change if the PLC configuration is extended.

- ► While programming only use the symbol names of the system flags!

→ System manual "Know-How ecomat*mobile*"
→ chapter *Error codes and diagnostic information*

7.1.1 System flags: CAN

19555

System flags (symbol name)	Type	Description	
CANx_BAUDRATE	WORD	CAN interface x: set baud rate in [kBaud]	
CANx_BUSOFF	BOOL	CAN interface x: Error "CAN-Bus off" ℹ Reset of the error code alse resets the flag	
CANx_LASTERROR	BYTE	CAN interface x: Error number of the last CAN transmission:	
		0 = no error	Initial value
		1 = stuff error	more than 5 identical bits in series on the bus
		2 = form error	received message had wrong format
		3 = ack error	sent message was not confirmed
		4 = bit1 error	a recessive bit was sent outside the arbitration area, but a dominant bit was read on the bus
		5 = bit0 error	it was tried to send a dominant bit, but a recessive level was read OR: a sequence of 11 recessive bits was read during bus-off recovery
		6 = CRC error	checksum of the received message was wrong
CANx_WARNING	BOOL	CAN interface x: warning threshold reached (> 96) ℹ A reset of the flag is possible via write access	
DOWNLOADID	WORD	CAN interface x: set download identifier	

x = 1...2 = number of the CAN interface

7.1.2 System flags: CAN (extended side)

20270

System flags (symbol name)	Type	Description	
CANx_BAUDRATE_E	WORD	Extended side CAN interface x: set baud rate in [kBaud]	
CANx_BUSOFF_E	BOOL	Extended side CAN interface x: Error "CAN-Bus off" ℹ Reset of the error code alse resets the flag	
CANx_LASTERROR_E	BYTE	Extended side CAN interface x: Error number of the last CAN transmission:	
		0 = no error	Initial value
		1 = stuff error	more than 5 identical bits in series on the bus
		2 = form error	received message had wrong format
		3 = ack error	sent message was not confirmed
		4 = bit1 error	a recessive bit was sent outside the arbitration area, but a dominant bit was read on the bus
		5 = bit0 error	it was tried to send a dominant bit, but a recessive level was read OR: a sequence of 11 recessive bits was read during bus-off recovery
		6 = CRC error	checksum of the received message was wrong
CANx_WARNING_E	BOOL	Extended side CAN interface x: warning threshold reached (> 96) ℹ A reset of the flag is possible via write access	

CANx stands for x = 1...2 = number of the CAN interface

7.1.3 System flags: SAE J1939

19556

System flags (symbol name)	Type	Description
J1939_TASK	BOOL	Using J1939_TASK, the time requirement for sending J1939 messages is met. If J1939 messages are to be sent with a repetition time ≤ 50 ms, the runtime system automatically sets J1939_TASK=TRUE. For applications for which the time requirement is ≥ PLC cycle time: ▶ Reduce system load with J1939_TASK=FALSE! TRUE: J1939 task is active (= initial value) The task is called every 2 ms. The J1939 stack sends its messages in the required time frame FALSE: J1939 task is not active

7.1.4 System flags: error flags (standard side)

19557

System flags (symbol name)	Type	Description
ERROR	BOOL	TRUE = set group error message, switch off relay

ERROR_BREAK_Qx
(0...x, value depends on the device, → data sheet)
	WORD	output group x: wire break error
		[Bit 0 for output 0] ... [bit z for output z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

ERROR_Ix
(0...x, value depends on the device, → data sheet)
	BYTE	input group x: periphery fault
		[Bit 0 for input 0] ... [bit z for input z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

ERROR_MEMORY	BOOL	memory error
ERROR_POWER	BOOL	Voltage error for VBBS / clamp 15: TRUE: Value out of range or: difference (VBB15 - VBBS) too great > general error FALSE: Value OK

ERROR_SHORT_Qx
(0...x, value depends on the device, → data sheet)
	WORD	output group x: short circuit error
		[Bit 0 for output 0] ... [bit z for output z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

ERROR_TEMPERATURE	BOOL	Temperature error TRUE: Value out of range > general error FALSE: Value OK
ERROR_VBBx	BOOL	Supply voltage error on VBBx (x = O | R): TRUE: Value out of range > general error FALSE: Value OK

7.1.5 System flags: error flags (extended side)

20262

System flags (symbol name)	Type	Description
ERROR_E	BOOL	Extended side: TRUE = set group error message, switch off relay

ERROR_BREAK_Qx_E
(0...x, value depends on the device, → data sheet)
	BYTE	Extended output group x_E: wire break error
		[Bit 0 for output 0] ... [bit z for output z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

ERROR_Ix_E
(0...x, value depends on the device, → data sheet)
	BYTE	Extended input group x_E: periphery fault
		[Bit 0 for input 0] ... [bit z for input z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

ERROR_MEMORY_E	BOOL	memory error extended side
ERROR_POWER_E	BOOL	Voltage error extended side: TRUE: Value out of range FALSE: Value OK

ERROR_SHORT_Qx_E
(0...x, value depends on the device, → data sheet)
	WORD	extended output group x_E: short circuit error
		[Bit 0 for output 0] ... [bit z for output z] of this group
		Bit = TRUE: error
		Bit = FALSE: no error

ERROR_TEMPERATURE_E	BOOL	Extended side temperature error TRUE: Value out of range > general error FALSE: Value OK
ERROR_VBBR_E	BOOL	Missing voltage on the terminal VBBR extended side

7.1.6 System flags: LED (standard side)

12817

System flags (symbol name)	Type	Description
LED	WORD	LED color for "LED switched on": 0x0000 = LED_GREEN (preset) 0x0001 = LED_BLUE 0x0002 = LED_RED 0x0003 = LED_WHITE 0x0004 = LED_BLACK 0x0005 = LED_MAGENTA 0x0006 = LED_CYAN 0x0007 = LED_YELLOW
LED_X	WORD	LED color for "LED switched off": 0x0000 = LED_GREEN 0x0001 = LED_BLUE 0x0002 = LED_RED 0x0003 = LED_WHITE 0x0004 = LED_BLACK (preset) 0x0005 = LED_MAGENTA 0x0006 = LED_CYAN 0x0007 = LED_YELLOW
LED_MODE	WORD	LED flashing frequency: 0x0000 = LED_2HZ (flashes at 2 Hz; preset) 0x0001 = LED_1HZ (flashes at 1 Hz) 0x0002 = LED_05HZ (flashes at 0.5 Hz) 0x0003 = LED_0HZ (lights permanently with value in LED) 0x0004 = LED_5HZ (flashes at 5 Hz)

7.1.7 System flags: LED (extended side)

20268

System flags (symbol name)	Type	Description
LED_E	WORD	LED color for "LED switched on": 0x0000 = LED_GREEN (preset) 0x0001 = LED_BLUE 0x0002 = LED_RED 0x0003 = LED_WHITE 0x0004 = LED_BLACK 0x0005 = LED_MAGENTA 0x0006 = LED_CYAN 0x0007 = LED_YELLOW

7.1.8 System flags: voltages (standard side)

19558

System flags (symbol name)	Type	Description
CLAMP_15	BOOL	Voltage monitoring on clamp 15
RELAIS	BOOL	TRUE: relay energised outputs are supplied with voltage FALSE: relay de-energised outputs are voltage-free
RELAIS_CLAMP_15	BOOL	Relay clamp 15 (pin 5)
SERIAL_BAUDRATE	WORD	Baud rate of the RS232 interface
SERIAL_MODE	BOOL	Activate serial interface (RS232) for use in the application TRUE: The RS232 interface can be used in the application, but no longer for programming, debugging or monitoring of the device. FALSE: The RS232 interface cannot be used in the application. Programming, debugging or monitoring of the device is possible.
SUPPLY_VOLTAGE	WORD	supply voltage at VBBS in [mV]
TEST	BOOL	TRUE: Test input is active: • Programming mode is enabled • Software download is possible • Status of the application program can be queried • Protection of stored software is not possible FALSE: application is in operation

7.1.9 System flags: voltages (extended side)

20269

System flags (symbol name)	Type	Description
CLAMP_15_E	BOOL	Voltage monitoring on extended side clamp 15
RELAIS_E	BOOL	TRUE: relay energised extended outputs are supplied with voltage FALSE: relay de-energised extended outputs are voltage-free
RELAIS_CLAMP_15_E	BOOL	Relay clamp 15 (extended side pin 5)
SERIAL_MODE_E	BOOL	Extended side: activate serial interface (RS232) for use in the application TRUE: The RS232 interface can be used in the application, but no longer for programming, debugging or monitoring of the device. FALSE: The RS232 interface cannot be used in the application. Programming, debugging or monitoring of the device is possible.
SSC_OK	BOOL	TRUE: communication via SSC interface is OK FALSE: communication via SSC interface is interrupted
SUPPLY_VOLTAGE_E	WORD	Extended side supply voltage at VBBS in [mV]
TEST_E	BOOL	TRUE: Extended side: Test input is active: • Programming mode is enabled • Software download is possible • Status of the application program can be queried • Protection of stored software is not possible FALSE: application is in operation

7.1.10 System flags: 16...40 inputs and 24...0 outputs (standard side)

19560

System flags (symbol name)	Type	Description
ANALOGx x = 0...7	WORD	Analogue input xx: filtered A/D converter raw value (12 bits) without calibration or standardisation
Ixx xx = 00...07 / 10...17 / 20...27 / 30...37 / 40...47	BOOL	Status on binary input xx Requirement: input is configured as binary input (MODE = IN_DIGITAL_H or IN_DIGITAL_L) TRUE: Voltage on binary input > 70 % of VBBs FALSE: Voltage on binary input < 30 % of VBBs or: not configured as binary input or: wrong configuration
Ixx_MODE xx = 00...07 / 14...17 / 24...27 / 30...37 / 40...47	BYTE	Operating mode of the input Ixx → chapter *Possible operating modes inputs/outputs* (→ page 247)
Qxx xx = 10...13 / 20...23 / 30...37 / 40...47	BOOL	Status on binary output xx: Requirement: output is configured as binary output TRUE: output activated FALSE: output deactivated (= initial value) or: not configured as binary output
Qxx_MODE xx = 10...13 / 20...23 / 30...37 / 40...47	BYTE	Operating mode of the output Qxx → chapter *Possible operating modes inputs/outputs* (→ page 247)

7.1.11 System flags: 16...40 inputs and 24...0 outputs (extended side)

20283

System flags (symbol name)	Type	Description
ANALOGx_E x = 0...7	WORD	Extended analogue input xx: filtered A/D converter raw value (12 bits) without calibration or standardisation
Ixx_E xx = 00...07 / 10...17 / 20...27 / 30...37 / 40...47	BOOL	Status at extended binary input xx_E Requirement: input is configured as binary input (MODE = IN_DIGITAL_H or IN_DIGITAL_L) TRUE: Voltage on binary input > 70 % of VBBS FALSE: Voltage on binary input < 30 % of VBBS or: not configured as binary input or: wrong configuration
IxxMode_E xx = 00...07 / 14...17 / 24...27 / 30...37 / 40...47	BYTE	Operating mode of the extended input Ixx_E → chapter *Possible operating modes inputs/outputs* (→ page 247)
Qxx_E xx = 10...13 / 20...23 / 30...37 / 40...47	BOOL	Status on extended binary input xx_E: Condition: output is configured as binary output TRUE: output activated FALSE: output deactivated (= initial value) or: not configured as binary output
QxxMode_E xx = 10...13 / 20...23 / 30...37 / 40...47	BYTE	Operating mode of the extended output Qxx_E → chapter *Possible operating modes inputs/outputs* (→ page 247)

7.2 Address assignment and I/O operating modes

Inhalt

1656

→ also data sheet

7.2.1 Address assignment inputs / outputs

Inhalt

2371

Inputs: Address assignment (standard side) (16...40 inputs)

19572

Abbreviations →chapter *Note on wiring* (→ page 30)
Operating modes of the inputs/outputs →chapter *Possible operating modes inputs/outputs* (→ page 247)

IEC address	Symbolic address
%IX0.00 %IW03	I00 ANALOG0
%IX0.01 %IW3	I01 ANALOG1
%IX0.02 %IW4	I02 ANALOG2
%IX0.03 %IW5	I03 ANALOG3
%IX0.04 %IW6	I04 ANALOG4
%IX0.05 %IW7	I05 ANALOG5
%IX0.06 %IW8	I06 ANALOG6
%IX0.07 %IW9	I07 ANALOG7
%IX0.08	I10
%IX0.09	I11
%IX0.10	I12
%IX0.11	I13
%IX0.12	I14
%IX0.13	I15
%IX0.14	I16
%IX0.15	I17
%IX1.00	I20
%IX1.01	I21
%IX1.02	I22
%IX1.03	I23
%IX1.04	I24
%IX1.05	I25
%IX1.06	I26
%IX1.07	I27
%IX1.08	I30
%IX1.09	I31
%IX1.10	I32
%IX1.11	I33
%IX1.12	I34
%IX1.13	I35
%IX1.14	I36
%IX1.15	I37

IEC address	Symbolic address
%IX2.00	I40
%IX2.01	I41
%IX2.02	I42
%IX2.03	I43
%IX2.04	I44
%IX2.05	I45
%IX2.06	I46
%IX2.07	I47

Inputs: address assignment (extended side) (16...40 inputs)

20282

Abbreviations →chapter *Note on wiring* (→ page 30)
Operating modes of the inputs/outputs →chapter *Possible operating modes inputs/outputs* (→ page 247)

IEC address	Symbolic address
%IX32.00 %IW35	I00_E ANALOG0_E
%IX32.01 %IW36	I01_E ANALOG1_E
%IX32.02 %IW37	I02_E ANALOG2_E
%IX32.03 %IW38	I03_E ANALOG3_E
%IX32.04 %IW39	I04_E ANALOG4_E
%IX32.05 %IW40	I05_E ANALOG5_E
%IX32.06 %IW41	I06_E ANALOG6_E
%IX32.07 %IW42	I07_E ANALOG7_E
%IX32.08	I10_E
%IX32.09	I11_E
%IX32.10	I12_E
%IX32.11	I13_E
%IX32.12	I14_E
%IX32.13	I15_E
%IX32.14	I16_E
%IX32.15	I17_E
%IX33.00	I20_E
%IX33.01	I21_E
%IX33.02	I22_E
%IX33.03	I23_E
%IX33.04	I24_E
%IX33.05	I25_E
%IX33.06	I26_E
%IX33.07	I27_E
%IX33.08	I30_E
%IX33.09	I31_E
%IX33.10	I32_E
%IX33.11	I33_E
%IX33.12	I34_E
%IX33.13	I35_E
%IX33.14	I36_E
%IX33.15	I37_E

IEC address	Symbolic address
%IX34.00	I40_E
%IX34.01	I41_E
%IX34.02	I42_E
%IX34.03	I43_E
%IX34.04	I44_E
%IX34.05	I45_E
%IX34.06	I46_E
%IX34.07	I47_E

Outputs: address assignment (standard side) (0...24 outputs)

19573

Abbreviations →chapter *Note on wiring* (→ page 30)
Operating modes of the inputs/outputs →chapter *Possible operating modes inputs/outputs* (→ page 247)

IEC address	Symbolic address
%QX0.00	Q10
%QX0.01	Q11
%QX0.02	Q12
%QX0.03	Q13
%QX0.04	Q20
%QX0.05	Q21
%QX0.06	Q22
%QX0.07	Q23
%QX0.08	Q30
%QX0.09	Q31
%QX0.10	Q32
%QX0.11	Q33
%QX0.12	Q34
%QX0.13	Q35
%QX0.14	Q36
%QX0.15	Q37
%QX1.00	Q40
%QX1.01	Q41
%QX1.02	Q42
%QX1.03	Q43
%QX1.04	Q44
%QX1.05	Q45
%QX1.06	Q46
%QX1.07	Q47

Outputs: address assignment (extended side) (0...24 outputs)

20296

Abbreviations →chapter *Note on wiring* (→ page 30)
Operating modes of the inputs/outputs →chapter *Possible operating modes inputs/outputs* (→ page 247)

IEC address	Symbolic address
%QX32.00	Q10_E
%QX32.01	Q11_E
%QX32.02	Q12_E
%QX32.03	Q13_E
%QX32.04	Q20_E
%QX32.05	Q21_E
%QX32.06	Q22_E
%QX32.07	Q23_E
%QX32.08	Q30_E
%QX32.09	Q31_E
%QX32.10	Q32_E
%QX32.11	Q33_E
%QX32.12	Q34_E
%QX32.13	Q35_E
%QX32.14	Q36_E
%QX32.15	Q37_E
%QX33.00	Q40_E
%QX33.01	Q41_E
%QX33.02	Q42_E
%QX33.03	Q43_E
%QX33.04	Q44_E
%QX33.05	Q45_E
%QX33.06	Q46_E
%QX33.07	Q47_E

7.2.2 Possible operating modes inputs/outputs

Inhalt

2386

Inputs: operating modes (standard side) (16...40 inputs)

19575

Possible configuration combinations (where permissible) are created by adding the configuration values.

🟩 = this configuration value is default

Inputs	Possible operating mode		Set with function block	FB input	Value	
					dec	hex
I00…I07	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
	IN_CURRENT	0…20 000 µA	INPUT_ANALOG	MODE	4	04
	IN_VOLTAGE10	0…10 000 mV	INPUT_ANALOG	MODE	8	08
	IN_VOLTAGE30	0…30 000 mV	INPUT_ANALOG	MODE	16	10
	IN_RATIO	0…1 000 ‰	INPUT_ANALOG	MODE	32	20
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG	MODE	64	40
I10…I13	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
I14…I17	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG	MODE	64	40
	IN_FAST	with IN_DIGITAL_H	internal		128	80
	IN_FAST	0…30 000 Hz	FREQUENCY PHASE	Frequency measurement		
		0.1…5 000 Hz	PERIOD	Period duration measurement		
		0.1…5 000 Hz	PERIOD_RATIO	Period duration and ratio measurement		
		0…30 000 Hz	FAST_COUNT	Counters		
		0…30 000 Hz	INC_ENCODER	Detect encoder values		
I20…I23	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
I24…I27	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
	IN_DIGITAL_L	minus	INPUT_ANALOG	MODE	2	02
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG	MODE	64	40
	IN_FAST	with IN_DIGITAL_H	internal		128	80
	IN_FAST	0…1 000 Hz	FREQUENCY PHASE	Frequency measurement		
		0.1…1 000 Hz	PERIOD	Period duration measurement		
		0.1…1 000 Hz	PERIOD_RATIO	Period duration and ratio measurement		
		0…1 000 Hz	FAST_COUNT	Counters		
		0…1 000 Hz	INC_ENCODER	Detect encoder		

Inputs	Possible operating mode		Set with function block	FB input	Value	
					dec	hex
I30…I37	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
	IN_DIGITAL_L	minus	INPUT_ANALOG	MODE	2	02
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG	MODE	64	40
I40…I47	IN_NOMODE	Off	INPUT_ANALOG	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG	MODE	1	01
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG	MODE	64	40

Set operating modes with the following function block:

FAST_COUNT (→ page 159)	Counter block for fast input pulses
FREQUENCY (→ page 160)	Measures the frequency of the signal arriving at the selected channel
INC_ENCODER (→ page 161)	Up/down counter function for the evaluation of encoders
INPUT_ANALOG (→ page 151)	Current and voltage measurement on the analogue input channel
PERIOD (→ page 164)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 166)	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 [‰].
PHASE (→ page 168)	Reads a pair of channels with fast inputs and compares the phase position of the signals

Inputs: operating modes (extended side) (16...40 inputs)

20297

Possible configuration combinations (where permissible) are created by adding the configuration values.

🟩 = this configuration value is default

Inputs	Possible operating mode		Set with FB	FB input	Value	
					dec	hex
I00_E...I07_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
	IN_CURRENT	0...20 000 µA	INPUT_ANALOG_E	MODE	4	04
	IN_VOLTAGE10	0...10 000 mV	INPUT_ANALOG_E	MODE	8	08
	IN_VOLTAGE30	0...30 000 mV	INPUT_ANALOG_E	MODE	16	10
	IN_RATIO	0...1 000 ‰	INPUT_ANALOG_E	MODE	32	20
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG_E	MODE	64	40
I10_E...I13_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
I14_E...I17_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG_E	MODE	64	40
	IN_FAST	with IN_DIGITAL_H	internal		128	80
	IN_FAST	0...30 000 Hz	FREQUENCY_E PHASE_E	Frequency measurement		
		0.1...5 000 Hz	PERIOD_E	Period duration measurement		
		0.1...5 000 Hz	PERIOD_RATIO_E	Period duration and ratio measurement		
		0...30 000 Hz	FAST_COUNT_E	Counters		
		0...30 000 Hz	INC_ENCODER_E	Detect encoder		
I20_E...I23_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
I24_E...I27_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
	IN_DIGITAL_L	minus	INPUT_ANALOG_E	MODE	2	02
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG_E	MODE	64	40
	IN_FAST	with IN_DIGITAL_H	internal		128	80
	IN_FAST	0...1 000 Hz	FREQUENCY_E PHASE_E	Frequency measurement		
		0.1...1 000 Hz	PERIOD_E	Period duration measurement		
		0.1...1 000 Hz	PERIOD_RATIO_E	Period duration and ratio measurement		
		0...1 000 Hz	FAST_COUNT_E	Counters		
		0...1 000 Hz	INC_ENCODER_E	Detect encoder		

Inputs	Possible operating mode		Set with FB	FB input	Value	
					dec	hex
I30_E...I37_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
	IN_DIGITAL_L	minus	INPUT_ANALOG_E	MODE	2	02
	IN_DIAGNOSTIC	with IN_DIGITAL_H	INPUT_ANALOG_E	MODE	64	40
I40_E...I47_E	IN_NOMODE	Off	INPUT_ANALOG_E	MODE	0	00
	IN_DIGITAL_H	plus	INPUT_ANALOG_E	MODE	1	01
	IN_DIAGNOSTIC	bei IN_DIGITAL_H	INPUT_ANALOG_E	MODE	64	40

Set operating modes with the following function block:

FAST_COUNT_E	= *FAST_COUNT* (→ page 159) for the extended side
FREQUENCY_E	= *FREQUENCY* (→ page 160) for the extended side
INC_ENCODER_E	= *INC_ENCODER* (→ page 161) for the extended side
INPUT_ANALOG_E	= *INPUT_ANALOG* (→ page 151) for the extended side
PERIOD_E	= *PERIOD* (→ page 164) for the extended side
PERIOD_RATIO_E	= *PERIOD_RATIO* (→ page 166) for the extended side
PHASE_E	= *PHASE* (→ page 168) for the extended side

Outputs: operating modes (standard side) (0...24 outputs)

19576

Possible configuration combinations (where permissible) are created by adding the configuration values.

🟩 = this configuration value is default

Outputs	Possible operating mode		set with	Value	
				dec	hex
Q10...Q13	OUT_NOMODE	Off	Qxx_MOD	0	00
	OUT_DIGITAL_H	plus	Qxx_MODE	1	01
	OUT_CURRENT		Qxx_MODE	4	04
	OUT_DIAGNOSTIC		Qxx_MODE	64	40
	OUT_OVERLOAD_PROTECTION		Qxx_MODE	128	80
Q20...Q23	OUT_NOMODE	Off	Qxx_MODE	0	00
	OUT_DIGITAL_H	plus	Qxx_MODE	1	01
	OUT_CURRENT		Qxx_MODE	4	04
	OUT_DIAGNOSTIC		Qxx_MODE	64	40
	OUT_OVERLOAD_PROTECTION		Qxx_MODE	128	80
Q30...Q37	OUT_NOMODE	Off	Qxx_MODE	0	00
	OUT_DIGITAL_H	plus	Qxx_MODE	1	01
	OUT_DIAGNOSTIC		Qxx_MODE	64	40
	OUT_OVERLOAD_PROTECTION		Qxx_MODE	128	80
Q40, Q43, Q44, Q47	OUT_NOMODE	Off	Qxx_MODE	0	00
	OUT_DIGITAL_H	plus	Qxx_MODE	1	01
	OUT_DIAGNOSTIC		Qxx_MODE	64	40
Q41, Q42, Q45, Q46	OUT_NOMODE	Off	Qxx_MODE	0	00
	OUT_DIGITAL_H	plus	Qxx_MODE	1	01
	OUT_DIGITAL_L	minus	Qxx_MODE	2	02
	OUT_DIAGNOSTIC		Qxx_MODE	64	40

Outputs: permitted operating modes

19577

Operating mode		Q10	Q11	Q12	Q13	Q20	Q21	Q22	Q23
OUT_NOMODE	Off	X	X	X	X	X	X	X	X
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_CURRENT		X	X	X	X	X	X	X	X
OUT_DIAGNOSTIC		X	X	X	X	X	X	X	X
OUT_OVERLOAD_PROTECTION		X	X	X	X	X	X	X	X
PWM		X	X	X	X	X	X	X	X

Operating mode		Q30	Q31	Q32	Q33	Q34	Q35	Q36	Q37
OUT_NOMODE	Off	X	X	X	X	X	X	X	X
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_DIAGNOSTIC		X	X	X	X	X	X	X	X

Operating mode		Q40	Q41	Q42	Q43	Q44	Q45	Q46	Q47
OUT_NOMODE	Off	X	X	X	X	X	X	X	X
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_DIGITAL_L	minus	--	X	X	--	--	X	X	--
OUT_DIAGNOSTIC		X	X	X	X	X	X	X	X
OUT_OVERLOAD_PROTECTION		X	X	X	X	X	X	X	X
PWM		X	--	--	X	X	--	--	X
H bridge		--	X	X	--	--	X	X	--

Outputs: operating modes (extended side) (0...24 outputs)

15550

Possible configuration combinations (where permissible) are created by adding the configuration values.

🟩 = this configuration value is default

Outputs	Possible operating mode		set with	Value	
				dec	hex
Q10_E...Q13_E	OUT_NOMODE	Off	QxxMode_E	0	00
	OUT_DIGITAL_H	plus	QxxMode_E	1	01
	OUT_CURRENT		QxxMode_E	4	04
	OUT_DIAGNOSTIC		QxxMode_E	64	40
	OUT_OVERLOAD_PROTECTION		QxxMode_E	128	80
Q20_E...Q23_E	OUT_NOMODE	Off	QxxMode_E	0	00
	OUT_DIGITAL_H	plus	QxxMode_E	1	01
	OUT_CURRENT		QxxMode_E	4	04
	OUT_DIAGNOSTIC		QxxMode_E	64	40
	OUT_OVERLOAD_PROTECTION		QxxMode_E	128	80
Q30_E...Q37_E	OUT_NOMODE	Off	QxxMode_E	0	00
	OUT_DIGITAL_H	plus	QxxMode_E	1	01
	OUT_DIAGNOSTIC		QxxMode_E	64	40
	OUT_OVERLOAD_PROTECTION		QxxMode_E	128	80
Q40_E, Q43_E, Q44_E, Q47_E	OUT_NOMODE	Off	QxxMode_E	0	00
	OUT_DIGITAL_H	plus	QxxMode_E	1	01
	OUT_DIAGNOSTIC		QxxMode_E	64	40
Q41_E, Q42_E, Q45_E, Q46_E	OUT_NOMODE	Off	QxxMode_E	0	00
	OUT_DIGITAL_H	plus	QxxMode_E	1	01
	OUT_DIGITAL_L	minus	QxxMode_E	2	02
	OUT_DIAGNOSTIC		QxxMode_E	64	40

Outputs: permitted operating modes

15542

Operating mode		Q10_E	Q11_E	Q12_E	Q13_E	Q20_E	Q21_E	Q22_E	Q23_E
OUT_NOMODE	Off	X	X	X	X	X	X	X	X
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_CURRENT		X	X	X	X	X	X	X	X
OUT_DIAGNOSTIC		X	X	X	X	X	X	X	X
OUT_OVERLOAD_PROTECTION		X	X	X	X	X	X	X	X
PWM		X	X	X	X	X	X	X	X

Operating mode		Q30_E	Q31_E	Q32_E	Q33_E	Q34_E	Q35_E	Q36_E	Q37_E
OUT_NOMODE	Off	X	X	X	X	X	X	X	X
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_DIAGNOSTIC		X	X	X	X	X	X	X	X

Operating mode		Q40_E	Q41_E	Q42_E	Q43_E	Q44_E	Q45_E	Q46_E	Q47_E
OUT_NOMODE	Off	X	X	X	X	X	X	X	X
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_DIGITAL_L	minus	--	X	X	--	--	X	X	--
OUT_DIAGNOSTIC		X	X	X	X	X	X	X	X
OUT_OVERLOAD_PROTECTION		X	X	X	X	X	X	X	X
PWM		X	--	--	X	X	--	--	X
H bridge		--	X	X	--	--	X	X	--

7.2.3 Addresses / variables of the I/Os

Inhalt

2376

Inputs: Addresses and variables (standard side) (16...40 inputs)

19579

IEC address	I/O variable	Note
%QB8	I00_MODE	Configuration byte for %IX0.00 (I00)
%QB9	I01_MODE	Configuration byte for %IX0.01 (I01)
%QB10	I02_MODE	Configuration byte for %IX0.02 (I02)
%QB11	I03_MODE	Configuration byte for %IX0.03 (I03)
%QB12	I04_MODE	Configuration byte for %IX0.04 (I04)
%QB13	I05_MODE	Configuration byte for %IX0.05 (I05)
%QB14	I06_MODE	Configuration byte for %IX0.06 (I06)
%QB15	I07_MODE	Configuration byte for %IX0.07 (I07)
%QB16	I14_MODE	Configuration byte for %IX0.14 (I14)
%QB17	I15_MODE	Configuration byte for %IX0.15 (I15)
%QB18	I16_MODE	Configuration byte for %IX0.16 (I16)
%QB19	I17_MODE	Configuration byte for %IX0.17 (I17)
%QB20	I24_MODE	Configuration byte for %IX1.04 (I24)
%QB21	I25_MODE	Configuration byte for %IX1.05 (I25)
%QB22	I26_MODE	Configuration byte for %IX1.06 (I26)
%QB23	I27_MODE	Configuration byte for %IX1.07 (I27)
%QB24	I30_MODE	Configuration byte for %IX1.08 (I30)
%QB25	I31_MODE	Configuration byte for %IX1.09 (I31)
%QB26	I32_MODE	Configuration byte for %IX1.10 (I32)
%QB27	I33_MODE	Configuration byte for %IX1.11 (I33)
%QB28	I34_MODE	Configuration byte for %IX1.12 (I34)
%QB29	I35_MODE	Configuration byte for %IX1.13 (I35)
%QB30	I36_MODE	Configuration byte for %IX1.14 (I36)
%QB31	I37_MODE	Configuration byte for %IX1.15 (I37)
%QB32	I40_MODE	Configuration byte for %IX2.00 (I40)
%Q33	I41_MODE	Configuration byte for %IX2.01 (I41)
%QB34	I42_MODE	Configuration byte für %IX2.02 (I42)
%QB35	I43_MODE	Configuration byte for %IX2.03 (I43)
%QB36	I44_MODE	Configuration byte for %IX2.04 (I44)
%QB37	I45_MODE	Configuration byte for %IX2.05 (I45)
%QB38	I46_MODE	Configuration byte for %IX2.06 (I46)

IEC address	I/O variable	Note
%QB39	I47_MODE	Configuration byte for %IX2.07 (I47)
%IW3	ANALOG0	Analogue value at I00
%IW4	ANALOG1	Analogue value at I01
%IW5	ANALOG2	Analogue value at I02
%IW6	ANALOG3	Analogue value at I03
%IW7	ANALOG4	Analogue value at I04
%IW8	ANALOG5	Analogue value at I05
%IW9	ANALOG6	Analogue value at I06
%IW10	ANALOG7	Analogue value at I07

Inputs: addresses and variables (extended side) (16...40 inputs)

20302

IEC address	I/O variable	Note
%QB72	I00Mode_E	Configuration byte for %IX32.00 (I00_E)
%QB73	I01Mode_E	Configuration byte for %IX32.01 (I01_E)
%QB74	I02Mode_E	Configuration byte for %IX32.02 (I02_E)
%QB75	I03Mode_E	Configuration byte for %IX32.03 (I03_E)
%QB76	I04Mode_E	Configuration byte for %IX32.04 (I04_E)
%QB77	I05Mode_E	Configuration byte for %IX32.05 (I05_E)
%QB78	I06Mode_E	Configuration byte for %IX32.06 (I06_E)
%QB79	I07Mode_E	Configuration byte for %IX32.07 (I07_E)
%QB80	I14Mode_E	Configuration byte for %IX32.14 (I14_E)
%QB81	I15Mode_E	Configuration byte for %IX32.15 (I15_E)
%QB82	I16Mode_E	Configuration byte for %IX32.16 (I16_E)
%QB83	I17Mode_E	Configuration byte for %IX32.17 (I17_E)
%QB84	I24Mode_E	Configuration byte for %IX33.04 (I24_E)
%QB85	I25Mode_E	Configuration byte for %IX33.05 (I25_E)
%QB86	I26Mode_E	Configuration byte for %IX33.06 (I26_E)
%QB87	I27Mode_E	Configuration byte for %IX33.07 (I27_E)
%QB88	I30Mode_E	Configuration byte for %IX33.08 (I30_E)
%QB89	I31Mode_E	Configuration byte for %IX33.09 (I31_E)
%QB90	I32Mode_E	Configuration byte for %IX33.10 (I32_E)
%QB91	I33Mode_E	Configuration byte for %IX33.11 (I33_E)
%QB92	I34Mode_E	Configuration byte for %IX33.12 (I34_E)
%QB93	I35Mode_E	Configuration byte for %IX33.13 (I35_E)
%QB94	I36Mode_E	Configuration byte for %IX33.14 (I36_E)
%QB95	I37Mode_E	Configuration byte for %IX33.15 (I37_E)
%QB96	I40Mode_E	Configuration byte for %IX34.00 (I40_E)
%QB97	I41Mode_E	Configuration byte for %IX34.01 (I41_E)
%QB98	I42Mode_E	Configuration byte for %IX34.02 (I42_E)
%QB99	I43Mode_E	Configuration byte for %IX34.03 (I43_E)
%QB100	I44Mode_E	Configuration byte for %IX34.04 (I44_E)
%QB101	I45Mode_E	Configuration byte for %IX34.05 (I45_E)
%QB102	I46Mode_E	Configuration byte for %IX34.06 (I46_E)
%QB103	I47Mode_E	Configuration byte for %IX34.07 (I47_E)
%IW35	ANALOG0_E	Analogue value at I00_E
%IW36	ANALOG1_E	Analogue value at I01_E
%IW37	ANALOG2_E	Analogue value at I02_E
%IW38	ANALOG3_E	Analogue value at I03_E
%IW39	ANALOG4_E	Analogue value at I04_E
%IW40	ANALOG5_E	Analogue value at I05_E

IEC address	I/O variable	Note
%IW41	ANALOG6_E	Analogue value at I06_E
%IW42	ANALOG7_E	Analogue value at I07_E

Outputs: Addresses and variables (standard side) (0...24 outputs)

19580

IEC address	I/O variable	Note
%QB40	Q10_MODE	Configuration byte for %QX0.00 (Q10)
%QB41	Q11_MODE	Configuration byte for %QX0.01 (Q11)
%QB42	Q12_MODE	Configuration byte for %QX0.02 (Q12)
%QB43	Q13_MODE	Configuration byte for %QX0.03 (Q13)
%QB44	Q20_MODE	Configuration byte for %QX0.04 (Q20)
%QB45	Q21_MODE	Configuration byte for %QX0.05 (Q21)
%QB46	Q22_MODE	Configuration byte for %QX0.06 (Q22)
%QB47	Q23_MODE	Configuration byte for %QX0.07 (Q23)
%QB48	Q30_MODE	Configuration byte for %QX0.08 (Q30)
%QB49	Q31_MODE	Configuration byte for %QX0.09 (Q31)
%QB50	Q32_MODE	Configuration byte for %QX0.10 (Q32)
%QB51	Q33_MODE	Configuration byte for %QX0.11 (Q33)
%QB52	Q34_MODE	Configuration byte for %QX0.12 (Q34)
%QB53	Q35_MODE	Configuration byte for %QX0.13 (Q35)
%QB54	Q36_MODE	Configuration byte for %QX0.14 (Q36)
%QB55	Q37_MODE	Configuration byte for %QX0.15 (Q37)
%QB56	Q40_MODE	Configuration byte for %QX1.00 (Q40)
%QB57	Q41_MODE	Configuration byte for %QX1.01 (Q41)
%QB58	Q42_MODE	Configuration byte for %QX1.02 (Q42)
%QB59	Q43_MODE	Configuration byte for %QX1.03 (Q43)
%QB60	Q44_MODE	Configuration byte for %QX1.04 (Q44)
%QB61	Q45_MODE	Configuration byte for %QX1.05 (Q45)
%QB62	Q46_MODE	Configuration byte for %QX1.06 (Q46)
%QB63	Q47_MODE	Configuration byte for %QX1.07 (Q47)

Outputs: addresses and variables (extended side) (0...24 outputs)

20303

IEC address	I/O variable	Note
%QB104	Q10Mode_E	Configuration byte for %QX32.00 (Q10_E)
%QB105	Q11Mode_E	Configuration byte for %QX32.01 (Q11_E)
%QB106	Q12Mode_E	Configuration byte for %QX32.02 (Q12_E)
%QB107	Q13Mode_E	Configuration byte for %QX32.03 (Q13_E)
%QB108	Q20Mode_E	Configuration byte for %QX32.04 (Q20_E)
%QB109	Q21Mode_E	Configuration byte for %QX32.05 (Q21_E)
%QB110	Q22Mode_E	Configuration byte for %QX32.06 (Q22_E)
%QB111	Q23Mode_E	Configuration byte for %QX32.07 (Q23_E)
%QB112	Q30Mode_E	Configuration byte for %QX32.08 (Q30_E)
%QB113	Q31Mode_E	Configuration byte for %QX32.09 (Q31_E)
%QB114	Q32Mode_E	Configuration byte for %QX32.10 (Q32_E)
%QB115	Q33Mode_E	Configuration byte for %QX32.11 (Q33_E)
%QB116	Q34Mode_E	Configuration byte for %QX32.12 (Q34_E)
%QB117	Q35Mode_E	Configuration byte for %QX32.13 (Q35_E)
%QB118	Q36Mode_E	Configuration byte for %QX32.14 (Q36_E)
%QB119	Q37Mode_E	Configuration byte for %QX32.15 (Q37_E)
%QB120	Q40Mode_E	Configuration byte for %QX33.00 (Q40_E)
%QB121	Q41Mode_E	Configuration byte for %QX33.01 (Q41_E)
%QB122	Q42Mode_E	Configuration byte for %QX33.02 (Q42_E)
%QB123	Q43Mode_E	Configuration byte for %QX33.03 (Q43_E)
%QB124	Q44Mode_E	Configuration byte for %QX33.04 (Q44_E)
%QB125	Q45Mode_E	Configuration byte for %QX33.05 (Q45_E)
%QB126	Q46Mode_E	Configuration byte for %QX33.06 (Q46_E)
%QB127	Q47Mode_E	Configuration byte for %QX33.07 (Q47_E)

7.3 Error tables

19606

7.3.1 Error flags

19608

→ chapter *System flags* (→ page 232)

7.3.2 Errors: CAN / CANopen

19610
19604

→ System manual "Know-How ecomat*mobile*"
→ chapter *CAN / CANopen: errors and error handling*

EMCY codes: CANx

13094

ℹ The indications for CANx also apply to each of the CAN interfaces.

EMCY code object 0x1003		Object 0x1001	Manufactor specific information					
Byte 0 [hex]	Byte 1 [hex]	Byte 2 [hex]	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Description
00	80	11	--	--	--	--	--	CANx monitoring SYNC error (only slave)
00	81	11	--	--	--	--	--	CANx warning threshold (> 96)
10	81	11	--	--	--	--	--	CANx receive buffer overrun
11	81	11	--	--	--	--	--	CANx transmit buffer overrun
30	81	11	--	--	--	--	--	CANx guard/heartbeat error (only slave)

EMCY codes: I/Os, system (standard side)

19552

The following EMCY messages are sent automatically, if the FB *CANx_MASTER_EMCY_HANDLER* (→ page 98) is called cyclically.

EMCY code object 0x1003		Object 0x1001	Manufactor specific information					
Byte 0 [hex]	Byte 1 [hex]	Byte 2 [hex]	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Description
00	21	03	I0	I1	I2	I3	I4	Input diagnostics
00	23	03	Q1Q2	Q3	Q4			Output diagnostics if interruption
02	23	03	Q1Q2	Q3	Q4			Output diagnostics if short circuit
00	31	05						Terminal voltage VBBO/VBBS
00	33	05						Output voltage VBBR
00	42	09						Excess temperature
00	61	11						Memory error

EMCY codes: I/Os, system (extended side)

20304

The following EMCY messages are sent automatically, if the FB *CANx_MASTER_EMCY_HANDLER* (→ page 98) is called cyclically.

EMCY code object 0x1003		Object 0x1001	Manufactor specific information					
Byte 0 [hex]	Byte 1 [hex]	Byte 2 [hex]	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Description
01	21	03	I0_E	I1_E	I2_E	I3_E	I4_E	Input diagnostics
01	23	03	Q1Q2_E	Q3_E	Q4_E			Output diagnostics if interruption
03	23	03	Q1Q2_E	Q3_E	Q4_E			Output diagnostics if short circuit
01	31	05						Terminal voltage VBBO/VBBS
01	33	05						Output voltage VBBR
01	42	09						Excess temperature
01	61	11						Memory error

8 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.

Architecture

Specific configuration of hardware and/or 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

Boot loader

On delivery **ecomat*mobile*** controllers only contain the boot loader.
The boot loader is a start program that allows to reload the runtime system and the application program on the device.
The boot loader contains basic routines...

- for communication between hardware modules,
- for reloading the operating system.

The boot loader is the first software module to be saved on the device.

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 large data volumes. There are several higher-level protocols that are based on CAN, e.g. 'CANopen' or 'J1939'.

CAN stack

CAN stack = software component that deals with processing CAN messages.

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 → www.can-cia.org

CiA DS 304

DS = **D**raft **S**tandard
CANopen device profile for safety communication

CiA DS 401

DS = **D**raft **S**tandard
CANopen device profile for binary and analogue I/O modules

CiA DS 402

DS = **D**raft **S**tandard
CANopen device profile for drives

CiA DS 403

DS = **D**raft **S**tandard
CANopen device profile for HMI

CiA DS 404

DS = **D**raft **S**tandard
CANopen device profile for measurement and control technology

CiA DS 405

DS = **D**raft **S**tandard
CANopen specification of the interface to programmable controllers (IEC 61131-3)

CiA DS 406

DS = **D**raft **S**tandard
CANopen device profile for encoders

CiA DS 407

DS = **D**raft **S**tandard
CANopen 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 **Ob**ject
ID = **Id**entifier
ID of a CANopen communication object
Corresponds to the identifier of the CAN message with which the communication project is sent via the CAN bus.

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 → www.codesys.com

CSV file

CSV = **C**omma **S**eparated **V**alues (also: **C**haracter **S**eparated **V**alues)
A CSV file is a text file for storing or exchanging simply structured data.
The file extension is `.csv`.

Example: Source table with numerical values:

value 1.0	value 1.1	value 1.2	value 1.3
value 2.0	value 2.1	value 2.2	value 2.3
value 3.0	value 3.1	value 3.2	value 3.3

This results in the following CSV file:

```
value 1.0;value 1.1;value 1.2;value 1.3
value 2.0;value 2.1;value 2.2;value 2.3
value 3.0;value 3.1;value 3.2;value 3.3
```

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

Data type

Depending on the data type, values of different sizes can be stored.

Data type	min. value	max. value	size in the memory
BOOL	FALSE	TRUE	8 bits = 1 byte
BYTE	0	255	8 bits = 1 byte
WORD	0	65 535	16 bits = 2 bytes
DWORD	0	4 294 967 295	32 bits = 4 bytes
SINT	-128	127	8 bits = 1 byte
USINT	0	255	8 bits = 1 byte
INT	-32 768	32 767	16 bits = 2 bytes
UINT	0	65 535	16 bits = 2 bytes
DINT	-2 147 483 648	2 147 483 647	32 bits = 4 bytes
UDINT	0	4 294 967 295	32 bits = 4 bytes
REAL	$-3.402823466 \cdot 10^{38}$	$3.402823466 \cdot 10^{38}$	32 bits = 4 bytes
ULINT	0	18 446 744 073 709 551 615	64 Bit = 8 Bytes
STRING			number of char. + 1

DC

Direct **C**urrent

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.

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.

DLC

Data **L**ength **C**ode = in CANopen the number of the data bytes in a message.
For →SDO: DLC = 8

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
In the protocol J1939 faults and errors well 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 engine

EDS-file

EDS = **E**lectronic **D**ata **S**heet, e.g. for:
- File for the object directory in the CANopen 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 →runtime system.
The firmware establishes the connection between the hardware of the device and the application program. The firmware is provided by the manufacturer of the controller as a part of the system and cannot be changed by the user.

EMC

EMC = **E**lectro **M**agnetic **C**ompatibility.
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.

EMCY

abbreviation for emergency
Message in the CANopen protocol with which errors are signalled.

Ethernet

Ethernet is a widely used, manufacturer-independent technology which enables data transmission in the network at a speed of 10...10 000 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 = **E**quipment **U**nder **C**ontrol.
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

FiFo

FIFO (**F**irst **I**n, **F**irst **O**ut) = Operating principle of the stack memory: The data packet that was written into the stack memory first, will also be read first. Each identifier has such a buffer (queue).

Flash memory

Flash ROM (or flash EPROM or flash memory) combines the advantages of semiconductor memory and hard disks. 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.

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.

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.

FRAM

FRAM, or also FeRAM, means **Fe**rroelectric **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:
- non-volatile,
- compatible with common EEPROMs, but:
- access time approx. 100 ns,
- nearly unlimited access cycles possible.

H

Heartbeat

The participants regularly send short signals. In this way the other participants can verify if a participant has failed.

HMI

HMI = **H**uman **M**achine **I**nterface

I

ID

ID = **Id**entifier

Name to differentiate the devices / participants connected to a system or the message packets transmitted between the participants.

IEC 61131

Standard: Basics of programmable logic controllers
- Part 1: General information
- Part 2: Production equipment requirements and tests
- Part 3: Programming languages
- Part 5: Communication
- Part 7: Fuzzy Control Programming

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

J

J1939

→ SAE J1939

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.

Link

A link is a cross-reference to another part in the document or to an external document.

LSB

Least **S**ignificant **B**it/Byte

M

MAC-ID

MAC = **M**anufacturer's **A**ddress **C**ode
= manufacturer's serial number.
→ID = **Id**entifier
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.

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* (→ page 269)

MRAM

MRAM = **M**agnetoresistive **R**andom **A**ccess **M**emory
The information is stored by means of magnetic storage elements. The property of certain materials is used to change their electrical resistance when exposed to magnetic fields.
Advantages of MRAM as compared to conventional RAM memories:

- non volatile (like FRAM), but:
- access time only approx. 35 ns,
- unlimited number of access cycles possible.

MSB

Most **S**ignificant **B**it/Byte

N

NMT

NMT = **N**etwork **M**anagement = (here: in the CANopen protocol).
The NMT master controls the operating states of the NMT slaves.

Node

This means a participant in the network.

Node Guarding

Node = here: network participant
Configurable cyclic monitoring of each →slave configured accordingly. The →master verfies 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.

OPC

OPC = **O**LE for **P**rocess **C**ontrol
Standardised software interface for manufacturer-independent communication in automation technology
OPC client (e.g. device for parameter setting or programming) automatically logs on to OPC server (e.g. automation device) when connected and communicates with it.

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.
According to the protocol, these services are unconfirmed data transmission: 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).

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.

PES

Programmable **E**lectronic **S**ystem ...
- for control, protection or monitoring,
- dependent for its operation on one or more programmable electronic devices,
- including all elements of the system such as input and output devices.

PGN

PGN = **P**arameter **G**roup **N**umber
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 mean?* (→ page 7))

PID controller

The PID controller (proportional–integral–derivative controller) consists of the following parts:
- P = proportional part
- I = integral part
- D = differential part (but not for the controller CR04nn, CR253n).

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 periphery states (inputs/outputs) is possible.

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.

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 changed virtually (in the process image).
- At the end of the cycle the PLC writes the virtual output states to the real outputs.

PWM

PWM = pulse width modulation
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.

R

ratiometric

Measurements can also be performed ratiometrically. If the output signal of a sensor is proportional to its suppy voltage then via ratiometric measurement (= measurement proportional to the supply) the influence of the supply's fluctuation can be reduced, in ideal case it can be eliminated.
→ analogue input

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 garantees in ISO 16845 the interchangeability of CAN chips in addition.

remanent

Remanent data is protected against data loss in case of power failure.
The →runtime 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 runtime 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.

ro

RO = read only for reading only
Unidirectional data transmission: Data can only be read and not changed.

RTC

RTC = **R**eal **T**ime **C**lock
Provides (batter-backed) the current date and time. Frequent use for the storage of error message protocols.

Runtime system

Basic program in the device, establishes the connection between the hardware of the device and the application program.

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.engine speed, temperature) and control information.
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

SD card

An SD memory card (short for **S**ecure **D**igital Memory Card) is a digital storage medium that operates to the principle of →flash storage.

SDO

SDO = **S**ervice **D**ata **O**bject.
The SDO is used for access to objects in the CANopen object directory. 'Clients' ask for the requested data from 'servers'. The SDOs always consist of 8 bytes.

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.

Slave

Passive participant on the bus, only replies on request of the →master. Slaves have a clearly defined and unique →address in the bus.

stopped

Operating status of a CANopen participant. In this mode only →NMT commands are transferred.

Symbols

Pictograms are figurative symbols which convey information by a simplified graphic representation.
(→ chapter *What do the symbols and formats mean?* (→ page 7))

System variable

Variable to which access can be made via IEC address or symbol name from the PLC.

T

Target

The target contains the hardware description of the target device for CODESYS, e.g.: inputs and outputs, memory, file locations.
Corresponds to an electronic data sheet.

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.

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.
According to the protocol, these services are unconfirmed data transmission: 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).

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.

9 Index

10 Notizen • Notes • Notes

Notizen • Notes • Notes

11 ifm weltweit • ifm worldwide • ifm à l’échelle internationale

8310

Version: 2015-03-06

www.ifm.com • Email: info@ifm.com

Service hotline: 0800 / 16 16 16 (only Germany, Mo-Fr 07.00...18.00 h)

ifm Niederlassungen • Sales offices • Agences

D	ifm electronic gmbh Vertrieb Deutschland Niederlassung Nord • 31135 Hildesheim • Tel. 0 51 21 / 76 67-0 Niederlassung West • 45128 Essen • Tel. 02 01 / 3 64 75 -0 Niederlassung Mitte-West • 58511 Lüdenscheid • Tel. 0 23 51 / 43 01-0 Niederlassung Süd-West • 64646 Heppenheim • Tel. 0 62 52 / 79 05-0 Niederlassung Baden-Württemberg • 73230 Kirchheim • Tel. 0 70 21 / 80 86-0 Niederlassung Bayern • 82178 Puchheim • Tel. 0 89 / 8 00 91-0 Niederlassung Ost • 07639 Tautenhain • Tel. 0 36 601 / 771-0 ifm electronic gmbh • Friedrichstraße 1 • 45128 Essen
A	ifm electronic gmbh • 1120 Wien • Tel. +43 16 17 45 00
OFF	ifm efector pty ltd. • Mulgrave Vic 3170 • Tel. +61 3 00 365 088
B, L	ifm electronic N.V. • 1731 Zellik • Tel. +32 2 / 4 81 02 20
BR	ifm electronic Ltda. • 03337-000, Sao Paulo SP • Tel. +55 11 / 2672-1730
CH	ifm electronic ag • 4 624 Härkingen • Tel. +41 62 / 388 80 30
CN	ifm electronic (Shanghai) Co. Ltd. • 201203 Shanghai • Tel. +86 21 / 3813 4800
CND	ifm efector Canada inc. • Oakville, Ontario L6K 3V3 • Tel. +1 800-441-8246
CZ	ifm electronic spol. s.r.o. • 25243 Průhonice • Tel. +420 267 990 211
DK	ifm electronic a/s • 2605 BROENDBY • Tel. +45 70 20 11 08
E	ifm electronic s.a. • 08820 El Prat de Llobregat • Tel. +34 93 479 30 80
F	ifm electronic s.a. • 93192 Noisy-le-Grand Cedex • Tél. +33 0820 22 30 01
FIN	ifm electronic oy • 00440 Helsinki • Tel . +358 75 329 5000
GB, IRL	ifm electronic Ltd. • Hampton, Middlesex TW12 2HD • Tel. +44 208 / 213-0000
GR	ifm electronic Monoprosopi E.P.E. • 15125 Amaroussio • Tel. +30 210 / 6180090
H	ifm electronic kft. • 9028 Györ • Tel. +36 96 / 518-397
I	ifm electronic s.a. • 20041 Agrate-Brianza (MI) • Tel. +39 039 / 68.99.982
IL	Astragal Ltd. • Azur 58001 • Tel. +972 3 -559 1660
IND	ifm electronic India Branch Office • Kolhapur, 416234 • Tel. +91 231-267 27 70
J	efector co., ltd. • Chiba-shi, Chiba 261-7118 • Tel. +81 043-299-2070
MAL	ifm electronic Pte. Ltd • 47100 Puchong Selangor • Tel. +603 8063 9522
MEX	ifm efector S. de R. L. de C. V. • Monterrey, N. L. 64630 • Tel. +52 81 8040-3535
N	Sivilingeniør J. F. Knudtzen A/S • 1396 Billingstad • Tel. +47 66 / 98 33 50
NL	ifm electronic b.v. • 3843 GA Harderwijk • Tel. +31 341 / 438 438
P	ifm electronic s.a. • 4410-136 São Félix da Marinha • Tel. +351 223 / 71 71 08
PL	ifm electronic Sp. z o.o. • 40-106 Katowice • Tel. +48 32-608 74 54
RA, ROU	ifm electronic s.r.l. • 1107 Buenos Aires • Tel. +54 11 / 5353 3436
ROK	ifm electronic Ltd. • 140-884 Seoul • Tel. +82 2 / 790 5610
RP	Gram Industrial, Inc. • 1770 Mantilupa City • Tel. +63 2 / 850 22 18
RUS	ifm electronic • 105318 Moscow • Tel. +7 495 921-44-14
S	ifm electronic a b • 41250 Göteborg • Tel. +46 31 / 750 23 00
SGP	ifm electronic Pte. Ltd. • Singapore 609 916 • Tel. +65 6562 8661/2/3
SK	ifm electronic s.r.o. • 835 54 Bratislava • Tel. +421 2 / 44 87 23 29
THA	SCM Allianze Co., Ltd. • Bangkok 10 400 • Tel. +66 02 615 4888
TR	ifm electronic Ltd. Sti. • 34381 Sisli/Istanbul • Tel. +90 212 / 210 50 80
UA	TOV ifm electronic • 02660 Kiev • Tel. +380 44 501 8543
USA	ifm efector inc. • Exton, PA 19341 • Tel. +1 610 / 5 24-2000
ZA	ifm electronic (Pty) Ltd. • 0157 Pretoria • Tel. +27 12 345 44 49

Technische Änderungen behalten wir uns ohne vorherige Ankündigung vor.
We reserve the right to make technical alterations without prior notice.
Nous nous réservons le droit de modifier les données techniques sans préavis.