ifm electronic

Original Programming Manual
CabinetController

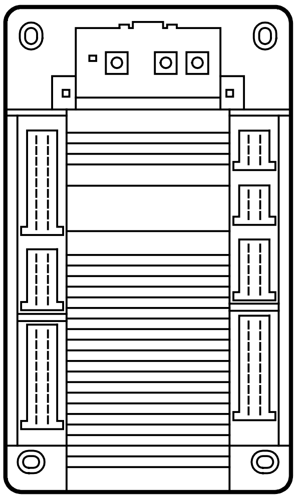
ecomat100

CR0303

Runtime system v05

CODESYS® v2.3

English



7390669_07_UK   2015-07-31

Contents

1 About this manual

Contents

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] > CR0303 > [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] > CR0303 > [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] > CR0303 > [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] > CR0303 > [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 (CR030n)

9181

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
2011-04-14	CR0303 several corrections:	- device has an own hydraulic library - some system flags do not exist - IEC addresses of in- and outputs - configuration of the inputs - set the status LED in the application program
2011-05-24	CR0303: memory FBs FRAMREAD, FRAMWRITE	permitted values of the parameters SRC, DST corrected
2012-01-09	Memory modules FRAMREAD, FRAMWRITE	Swapped parameters SRC, DST in the table "Permissible values"
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"
2014-07-18	CR0303: Error flag	Wrong: ERROR_A_INx Correct: ERROR_Ix
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
2015-01-13	Structure of documentation for error codes, system flags	• error flags: now only in the annex, chapter *System flags* • CAN / CANopen errors and error handling: now only in the system manual "Know-How" • error codes, EMCY codes: now in the annex, chapter *Error tables*
2015-03-10	Available memory	Description improved
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

Contents

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

Contents

975

3.1 Information about the device

19966

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

- CabinetController: CR0303

3.2 Hardware description

Contents

14081

3.2.1 Hardware structure

15332

Conditions

19971

The device does not start until sufficient voltage is applied to the supply connection VBBS.
A voltage > 10 V is deemed sufficient.
Permissible operating voltage → data sheet

Prinziple block diagram

19969

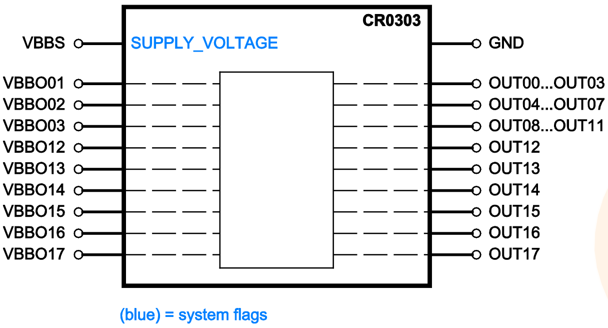


Figure: Block diagram of the supply

Available memory

13736

FLASH-Speicher

9901

FLASH memory (non-volatile, slow memory) overall existing in the device	1 MByte

Thereof the following memory areas are reserved for ...

maximum size of the application program	576 kByte
data other than the application program user can write data such as files, bitmaps, fonts	176 kByte
data other than the application program read data with *FLASHREAD* (→ page 179) or write data with *FLASHWRITE* (→ page 180) (files: 128 bytes less for header)	16 kByte

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

SRAM

6754

SRAM (volatile, fast memory) overall existing in the device SRAM indicates here all kinds of volatile and fast memories.	256 kByte

Thereof the following memory areas are reserved for ...

data reserved by the application program	80 kByte

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

FRAM

12342

FRAM (non-volatile, fast memory) overall existing in the device FRAM indicates here all kinds of non-volatile and fast memories.	2 kByte

Thereof the following memory areas are reserved for ...

variables in the application program, declared as VAR_RETAIN	256 Byte
remanent memory freely available to the user Access is made via *FRAMREAD* (→ page 182) and *FRAMWRITE* (→ page 183)	1536 Byte

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

3.2.2 Monitoring concept

19973

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

- the controller switches off completely
  > the program stops
  > the outputs become currentless and change to logic "0"
  > the status LED goes out

Monitoring and securing mechanisms

Contents

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.

3.2.3 Inputs (technology)

Contents

14090

Analogue inputs

2426

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

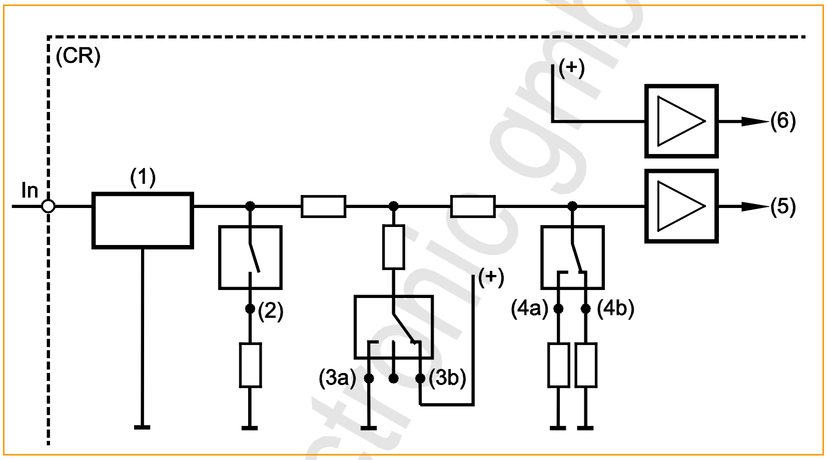
- current input 0...20 mA
- voltage input 0...10 V
- voltage input 0...32 V

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

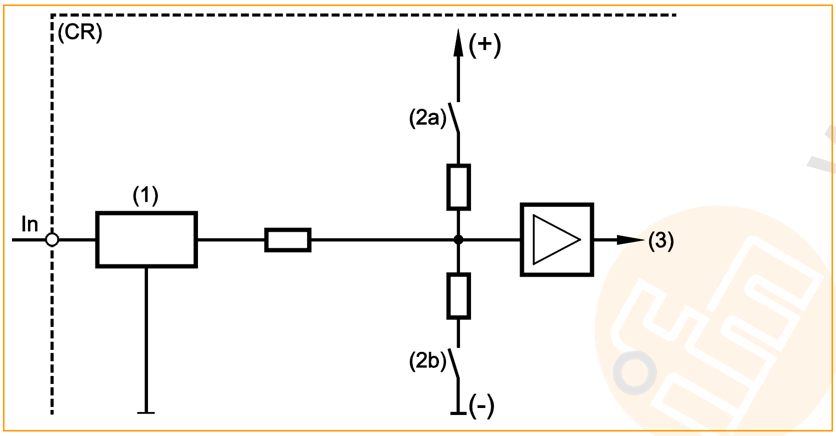
Digital inputs

1015
7345

The binary input can be operated in following modes:

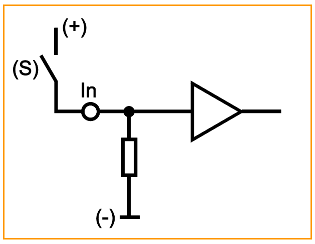
- binary input plus switching (BL) for positive sensor signal
- binary input minus switching (BH) for negative sensor signal

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



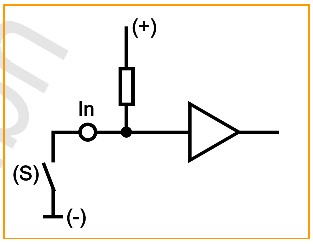
In = pin binary-input n
(CR) = device
(1) = input filter
(2a) = input minus switching
(2b) = input plus switching
(3) = voltage

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 (IN00...07)

19976

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 (with/without diagnosis)

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

► Configuration of each input is made via the application program:
- configuration byte INxx_MODE

Sensors with diagnostic capabilities to NAMUR can be evaluated.

Input group I1 (IN08...11 / FRQ00...03)

19979

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 (with/without diagnosis)
- fast input for e.g. incremental encoders and frequency or interval measurement

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

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

FAST_COUNT (→ page 134)	Counter block for fast input pulses
FREQUENCY (→ page 135)	Measures the frequency of the signal arriving at the selected channel
INC_ENCODER (→ page 136)	Up/down counter function for the evaluation of encoders
PERIOD (→ page 138)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 140)	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 142)	Reads a pair of channels with fast inputs and compares the phase position of the signals

Input group I2 (IN12...15)

19982

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
- binary input minus switching (BH) for negative sensor signal

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

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 198)

► Configuration of each input is made via the application program:
- configuration byte INxx_MODE

Input group I3 (A_IN16...23)

19984

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 ‰ von 32 V
- binary input plus switching (BL) for positive sensor signal (with/without diagnosis)

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

All inputs show the same behaviour concerning function and diagnosis.

► Configuration of each input is made via the application program:
  - configuration byte A_INxx_MODE
  - FB *INPUT_ANALOG* (→ page 127) > input MODE

> If the analogue inputs are configured for current measurement, the device switches to the safe voltage measurement range (0...32V DC) and the corresponding error bit in the flag byte ERROR_A_INx is set when the final value (> 23 mA) is exceeded. When the value is again below the limit value, the input automatically switches back to the current measurement range.

3.2.4 Outputs (technology)

Contents

14093

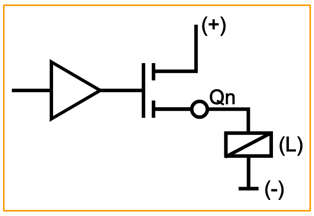
Binary outputs

19986

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

- binary output, plus switching (BH), short-circuit proof, overload protected

15451



Qn = pin output n
(L) = load

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

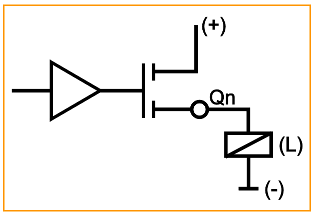
PWM outputs

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 Q0Q1 (OUT00...07)

19988

These outputs are a group of multifunction channels.

These outputs provide several function options (each output separately configurable):
- binary output, plus switching (BH), short-circuit proof, overload protected
- analogue output with Pulse Width Modulation (PWM)

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

► Configuration of each output is made via the application program:
PWM output: → FB *PWM1000* (→ page 148)

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

Output group Q2 (OUT08...11)

19990

These outputs are a group of channels with a single specified function.

These outputs have the following fixed setting:
- binary output, plus switching (BH), short-circuit proof, overload protected

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

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

Output group Q3 (OUT12...17)

19992

These outputs are a group of channels with a single specified function.

These outputs have the following fixed setting:

- binary output, plus switching (BH) with diagnostic function and protection, short-circuit proof, overload protected

→ chapter *Possible operating modes inputs/outputs* (→ page 203)

Switching current	≤ 10 A
Excessive current warning ➩ System flag WARNING_OVERCURRENT_OUTxx	10...16.5 A (typically 12 A)
Excessive current switch-off	13...21.5 A (typically 16 A)

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

Diagnosis: binary outputs (via current measurement)

19398
19396

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

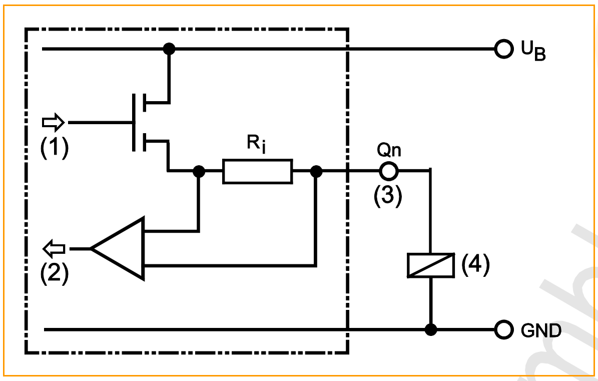


Figure: principle block diagram

(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 current 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.

3.2.5 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

3.2.6 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

3.2.7 Status LED

1437

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

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

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

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

LED_MODE	flashing frequency from the data structure "LED_MODES" allowed: LED_2HZ, LED_1HZ, LED_05HZ, LED_0HZ (permanently)

> ! If the flashing mode is changed by the application program, the above-mentioned table (default setting) is no longer valid.

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 194)):

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

Contents

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 CAN interfaces

Contents

14101

Connections and data → data sheet

CAN: interfaces and protocols

19523
14587

The devices are equipped with several CAN interfaces depending on the hardware design. Basically, all interfaces can be used with the following functions independently of each other:

- Layer 2: CAN at level 2 (→ chapter *Function elements: CAN layer 2* (→ page 63))
- CANopen master (→ chapter *Function elements: CANopen master* (→ page 79))
- CANopen slave (→ chapter *Function elements: CANopen slave* (→ page 89))
- CANopen network variables (via CODESYS)
- SAE J1939 (for drive management, → chapter *Function elements: SAE J1939* (→ page 102))
- bus load detection
- error frame counter
- download interface
- 100 % bus load without package loss

11796

The following CAN interfaces and CAN protocols are available in this **ecomatmobile** device:

CAN interface	**CAN 1**	**CAN 2**	**CAN 3**	**CAN 4**
Default download ID	ID 127	ID 126	ID 125	ID 124
CAN protocols	CAN Layer 2	CAN Layer 2	Interface do not exist	Interface do not exist
	CANopen	---		
	SAE J1939	SAE J1939		

Standard baud rate = 125 kBit/s

3.4 Software description

Contents

14107

3.4.1 Software modules for the device

Contents

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 yes	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 46)

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_CR0303_Vxxyyzz.LIB`	Device-specific library Must always be contained in the application program!
`ifm_CR0303_CANopenMaster_Vxxyyzz.LIB`	(optional) if CAN interface 1 of the device is to be operated as a CANopen master
`ifm_CR0303_CANopenSlave_Vxxyyzz.LIB`	(optional) if CAN interface1 of the device is to be operated as a CANopen slave
`ifm_CAN1_EXT_Vxxyyzz.LIB`	(optional) if CAN interface 1 of the device is to operate on 29 bits

`ifm_CR0303_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 CR0303* (→ page 57)

3.4.2 Programming notes for CODESYS projects

Contents

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] > CR0303 > [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 **ecomatmobile** numerous functions are available which enable use of the devices in a wide range of applications.

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

NOTICE

Risk that the 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.

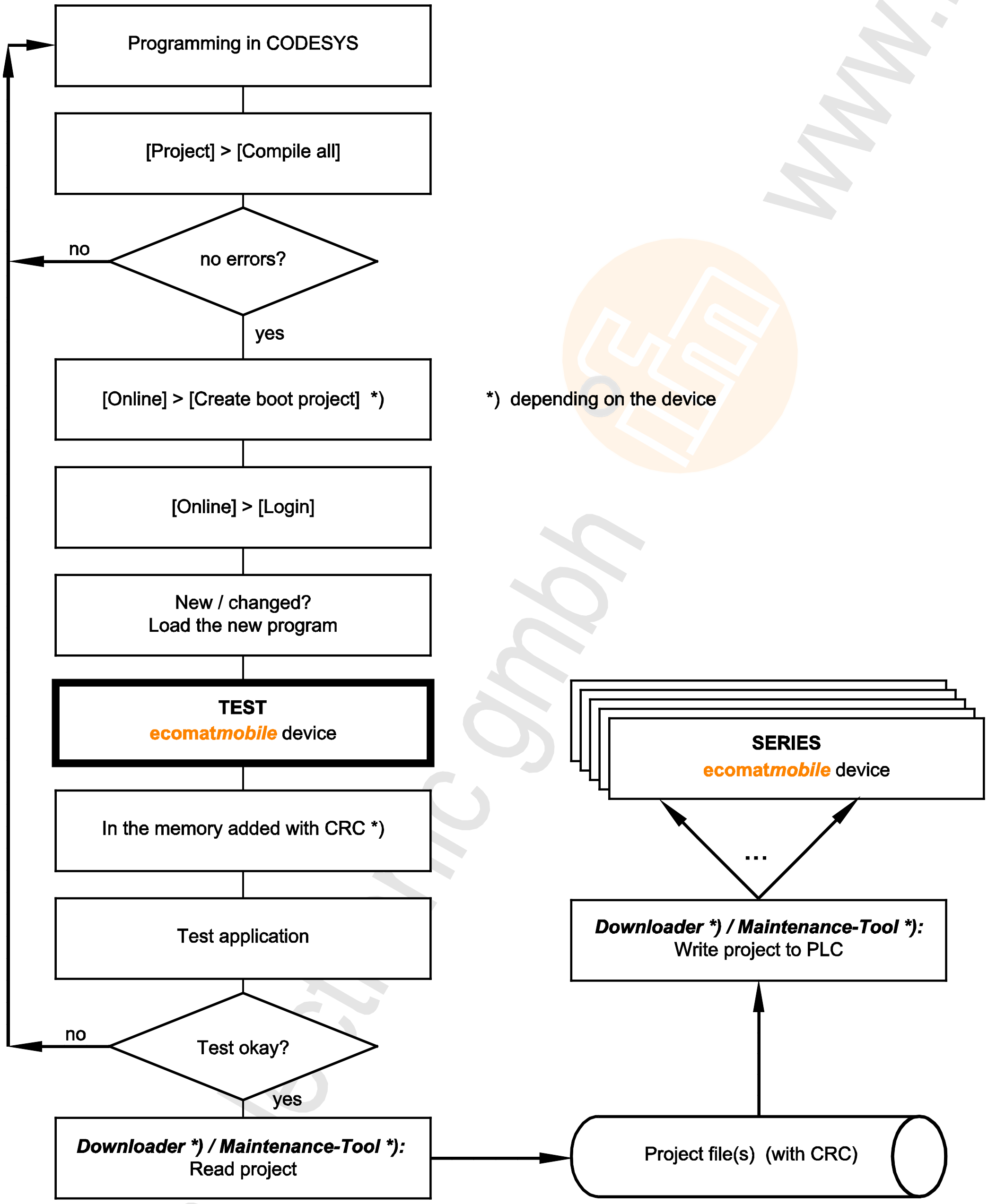
► If necessary, the cycle time must be optimised by restructuring the software and the system set-up.

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.


Programming in CODESYS
[Project] > [Compile all]
no
no errors?
yes
[Online] > [Create boot project] *)
*) depending on the device
[Online] > [Login]
New / changed?
Load the new program
TEST
ecomatmobile device
In the memory added with CRC *)
Test application
no
Test okay?
yes
Downloader *) / Maintenance-Tool *):
Read project
Project file(s) (with CRC)
Downloader *) / Maintenance-Tool *):
Write project to PLC
...
SERIES
ecomatmobile device

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

Contents

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

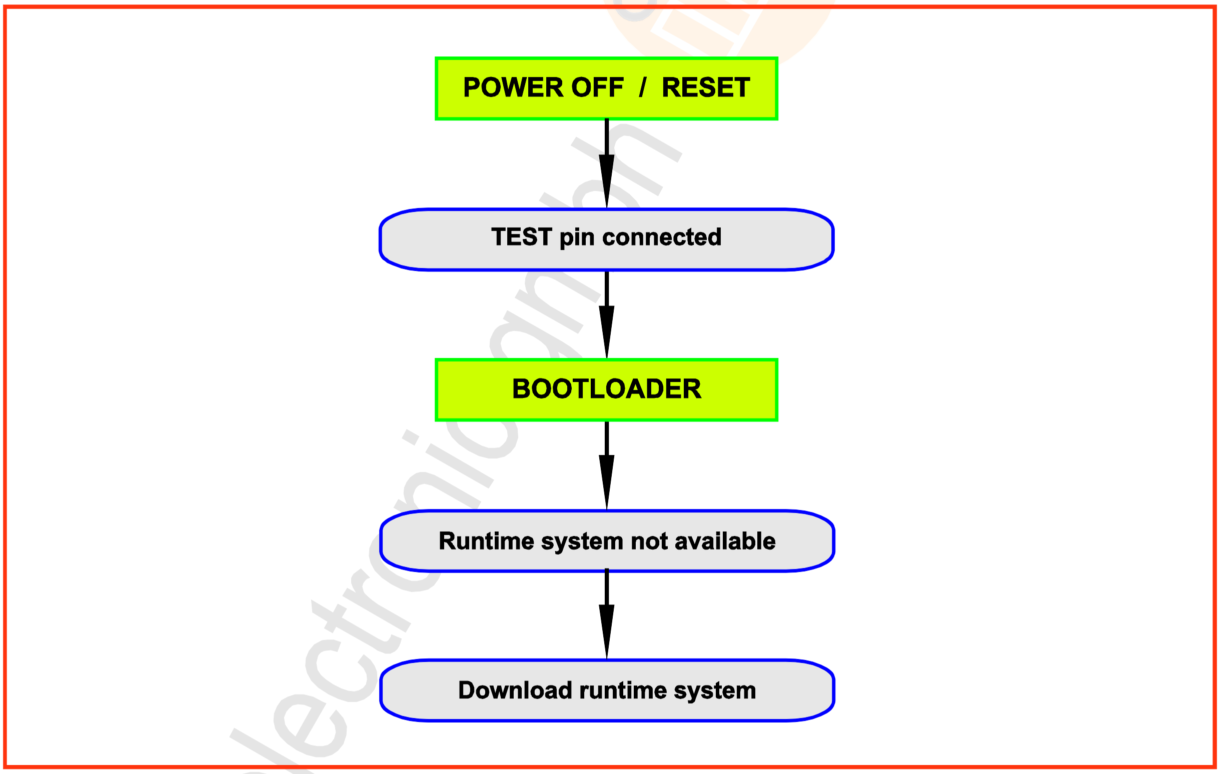


Figure: operating states (here: runtime system is not available)

Operating states: application program is not available

19218

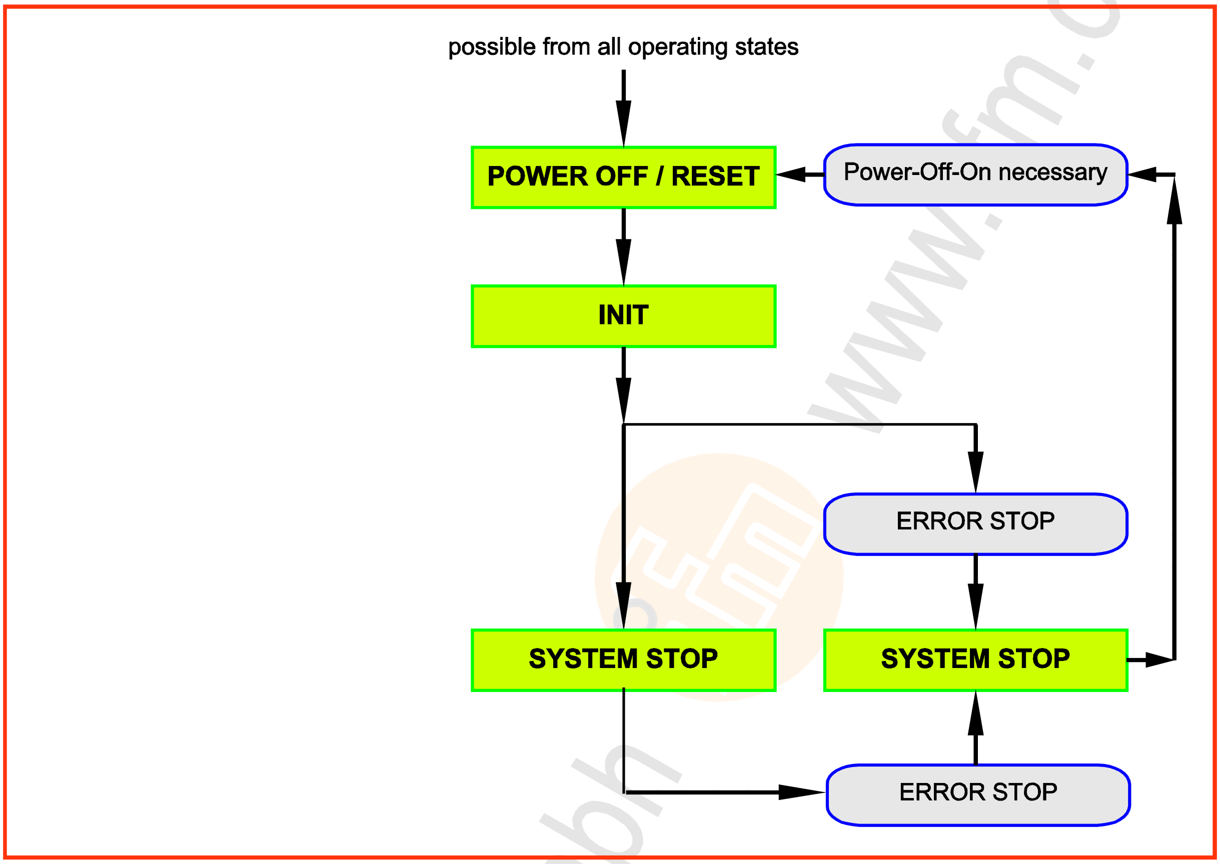


Figure: operating states (here: application program is not available)

Operating states: application program is available

19219

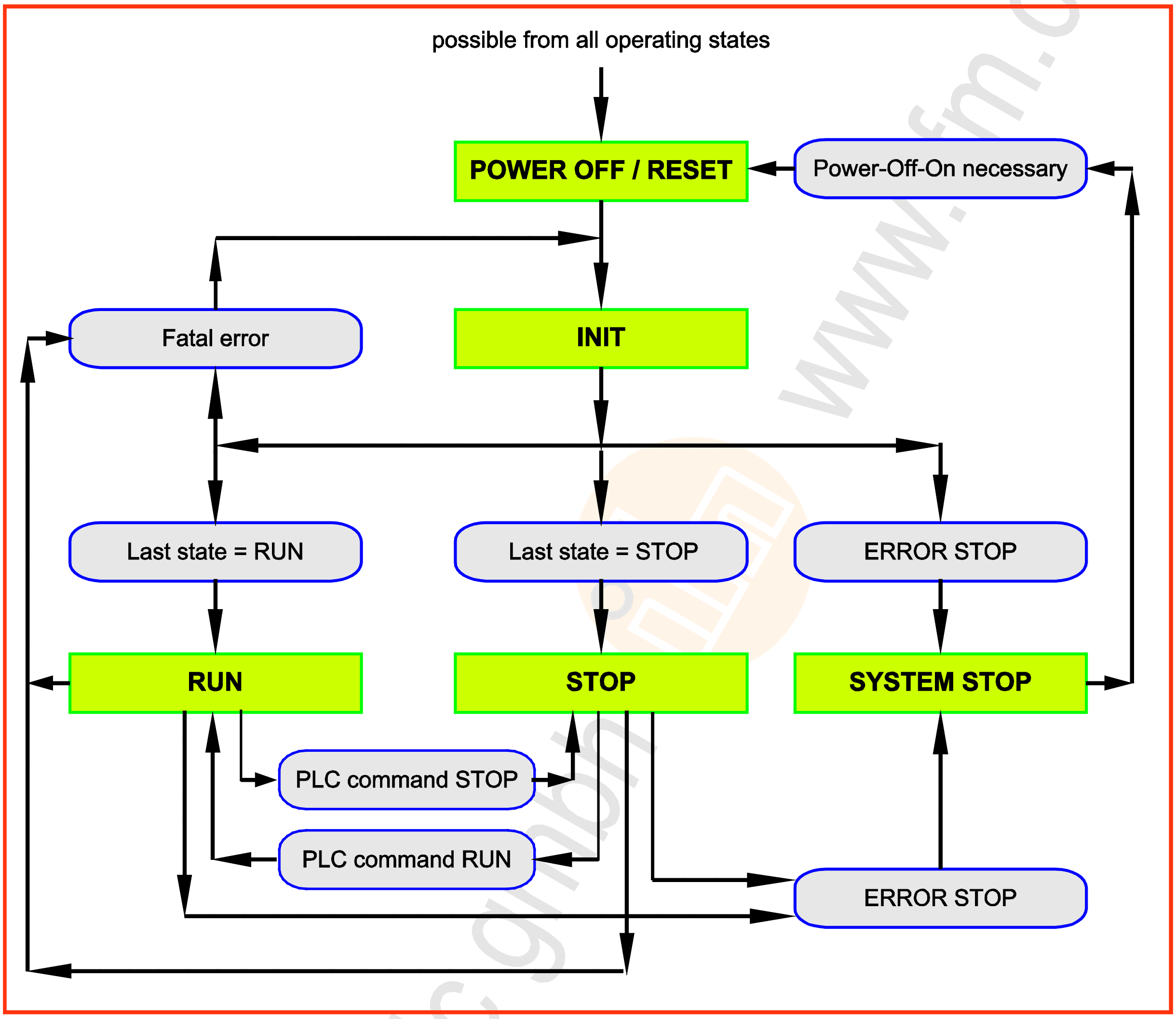


Figure: operating states (here: application program is available)

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 40))

> 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 40))

> 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 114)

DEBUG mode

1086

If the input DEBUG of *SET_DEBUG* (→ page 189) 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 40)) 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

1480
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.

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

1490

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

If the maximum watchdog time (100...200 ms) is exceeded:
> the device performs a reset and reboots

4 Configurations

Contents

1016

The device configurations described in the corresponding installation instructions or in the *Annex* (→ page 194) 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 Set up the runtime system

Contents

14091

4.1.1 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_CR0303_Vxxyyzz.H86`),
- PLC configuration (`ifm_CR0303_Vxx.CFG`),
- device library (`ifm_CR0303_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. "`CR0303`") 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).

4.1.2 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'.

4.1.3 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 188)	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 190))

4.2 Set up the programming system

Contents

3968

4.2.1 Set up the programming system manually

Contents

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_CR0303_Vxxyyzz.LIB` manually!

2689

❗ NOTE

The software versions suitable for the selected target must always be used:
- runtime system (`ifm_CR0303_Vxxyyzz.H86`),
- PLC configuration (`ifm_CR0303_Vxx.CFG`),
- device library (`ifm_CR0303_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. "`CR0303`") 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**.

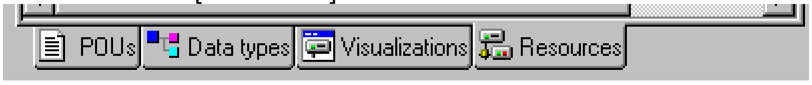
Activate the PLC configuration (e.g. CR0033)

15824

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

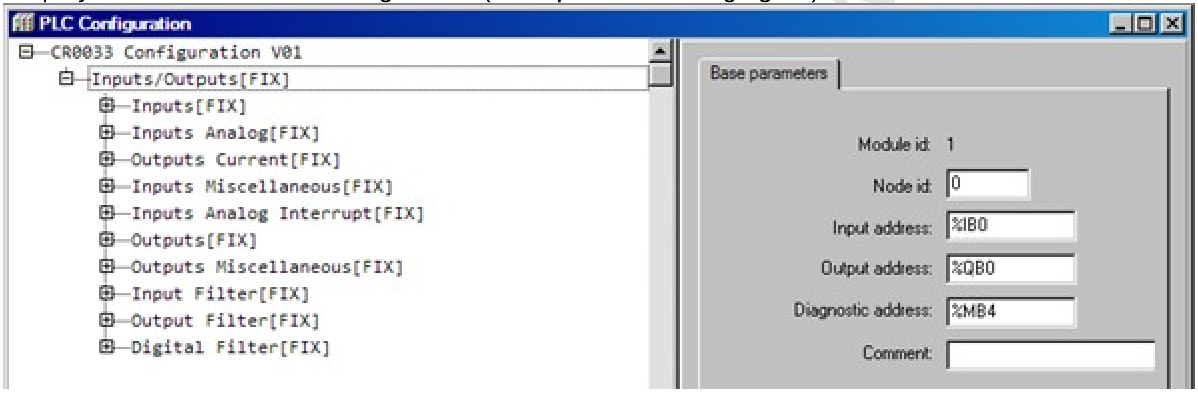
► The menu item [PLC Configuration] is reached via the tab [Resources].
Double-click on [PLC Configuration] to open the corresponding window.

► 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):

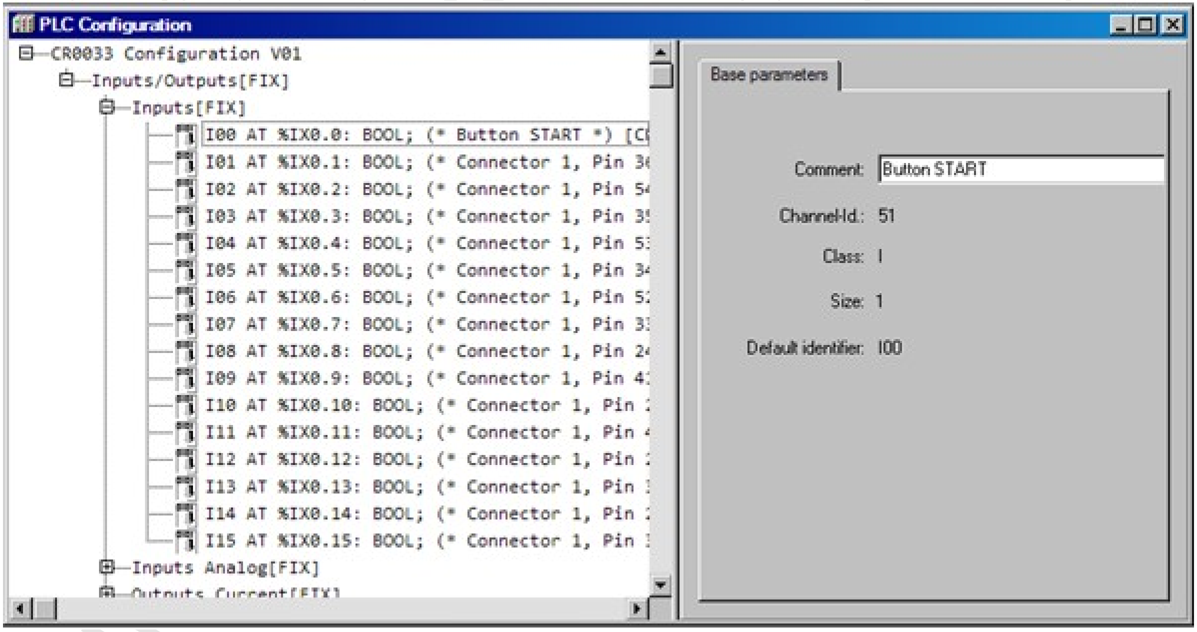


```
⊟—CR0033 Configuration V01
   ⊟—Inputs/Outputs[FIX]
      ⊞—Inputs[FIX]
      ⊞—Inputs Analog[FIX]
      ⊞—Outputs Current[FIX]
      ⊞—Inputs Miscellaneous[FIX]
      ⊞—Inputs Analog Interrupt[FIX]
      ⊞—Outputs[FIX]
      ⊞—Outputs Miscellaneous[FIX]
      ⊞—Input Filter[FIX]
      ⊞—Output Filter[FIX]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

4.2.2 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 ***ecomatmobile*** 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.

4.3 Function configuration in general

3971

4.3.1 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.

4.3.2 System variables

2252
13519
15576

All system variables (→ chapter *System flags* (→ page 194)) 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

4.4 Function configuration of the inputs and outputs

Contents

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 (`ifm_CR0303_Vxxyyzz.LIB`) for the configuration of the inputs and outputs.
  For details → *Possible operating modes inputs/outputs* (→ page 203).

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 Configure inputs

Contents

3973

Valid operating modes → chapter *Possible operating modes inputs/outputs* (→ page 203)

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

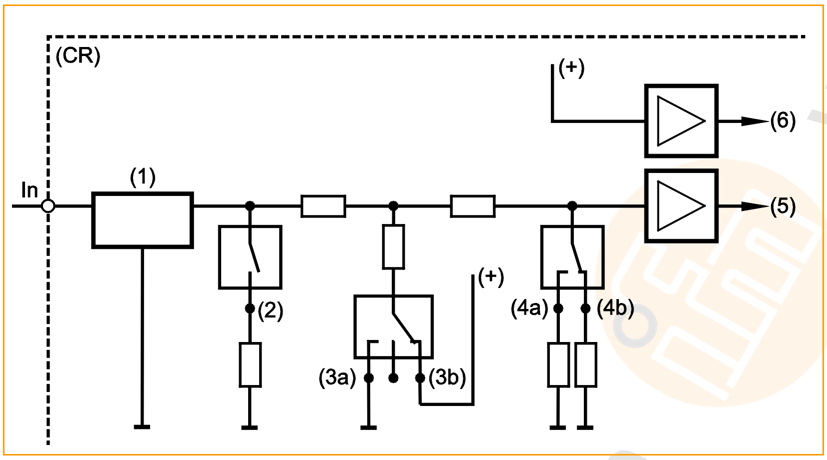
Analogue inputs: configuration and diagnosis

19998

- ► Configuration of each input is made via the application program:
  • configuration byte A_INxx_MODE
  → chapter *Possible operating modes inputs/outputs* (→ page 203)
- ► If the analogue inputs are configured for current measurement, the device switches to the safe voltage measurement range (0...32V DC) and the corresponding error bit in the flag byte ERROR_A_INx is set when the final value (> 23 mA) is exceeded. When the value is again below the limit value, the input automatically switches back to the current measurement range.

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

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

Binary inputs: configuration and diagnosis

20001

- ► Configuration of each input is made via the application program:
  • configuration byte INxx_MODE
  → chapter *Possible operating modes inputs/outputs* (→ page 203)

Fast inputs

19996

The devices dispose of fast counting/pulse inputs for an input frequency up to 30 kHz (→ data sheet). Appropriate function blocks are e.g.:

FAST_COUNT (→ page 134)	Counter block for fast input pulses
FREQUENCY (→ page 135)	Measures the frequency of the signal arriving at the selected channel
INC_ENCODER (→ page 136)	Up/down counter function for the evaluation of encoders
PERIOD (→ page 138)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 140)	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 142)	Reads a pair of channels with fast inputs and compares the phase position of the signals

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

Use as binary inputs

3804

The permissible high input frequencies also ensure the detection of faulty signals, e.g. bouncing contacts of mechanical switches.

► If required, suppress the faulty signals in the application program!

4.4.3 Configure outputs

Contents

3976

Valid operating modes → chapter *Possible operating modes inputs/outputs* (→ page 203)

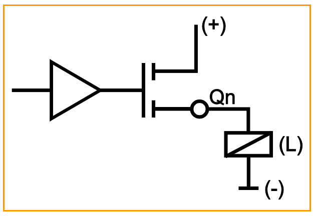
Binary outputs: configuration and diagnosis

20004

These outputs have the following fixed setting:

- binary output, plus switching (BH), short-circuit proof, overload protected

15451



Qn = pin output n
(L) = load

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

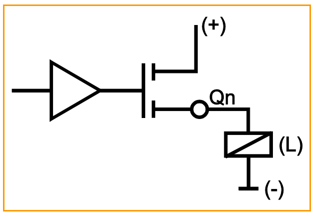
► Remedy:
- Reset the output logic in the application program!
- Remove the fault!
- Reset the outputs depending on the situation.

PWM outputs

14705

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

15414

⚠ WARNING

Property damage or bodily injury possible due to malfunctions!

For outputs in PWM mode:
- there are no diagnostic functions
- the overload protection OUT_OVERLOAD_PROTECTION is NOT active

9980

NOTE

PWM outputs must NOT be operated in parallel, e.g. in order to increase the max. output current. The outputs do not operate synchronously.

Otherwise the entire load current could flow through only one output. The current measurement would no longer function.

Availability of PWM

20006

Device	Number of available PWM outputs	of which current-controlled (PWMi)	PWM frequency [Hz]
CabinetController: CR0303	8	---	20...250

FBs for PWM functions

20007

The following function blocks are available for the PWM function of the outputs:

PWM (→ page 144)	Initialises and configures a PWM-capable output channel Definition of the PWM frequency via RELOAD
PWM1000 (→ page 148)	Initialises and configures a PWM-capable output channel the mark-to-space ratio can be indicated in steps of 1 ‰

4.5 Variables

Contents

3130

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

4.5.1 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.

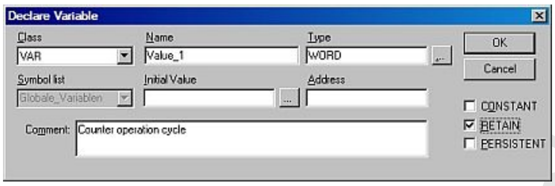
14166

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).



Class: VAR
Name: Value_1
Type: WORD
Symbol list: Globale_Variablen
Initial Value:
Address:
Comment: Counter operation cycle
□ CONSTANT
☑ RETAIN
□ PERSISTENT

4.5.2 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

Contents

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 CR0303

Contents

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_CR0303_V05yyzz.LIB

18572

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

Function element	Short description
CAN1_BAUDRATE (→ page 64)	Sets the transmission rate for the bus participant on CAN interface 1
CAN1_DOWNLOADID (→ page 65)	Sets the download identifier for CAN interface 1
CAN2 (→ page 71)	Initialises CAN interface 2 Set the mode and baud rate
CANx_ERRORHANDLER (→ page 72)	Executes a "manual" bus recovery on CAN interface x x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
CANx_EXT_RECEIVE_ALL (→ page 73)	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 74)	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 76)	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 98)	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 100)	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 78)	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)
CHECK_DATA (→ page 186)	Generates a checksum (CRC) for a configurable memory area and checks the data of the memory area for undesired changes
DELAY (→ page 164)	Delays the output of the input value by the time T (dead-time element)
FAST_COUNT (→ page 134)	Counter block for fast input pulses
FLASHREAD (→ page 179)	Transfers different data types directly from the flash memory to the RAM
FLASHWRITE (→ page 180)	Writes different data types directly into the flash memory
FRAMREAD (→ page 182)	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 183)	Writes different data types directly into the FRAM memory FRAM indicates here all kinds of non-volatile and fast memories.
FREQUENCY (→ page 135)	Measures the frequency of the signal arriving at the selected channel
GET_IDENTITY (→ page 188)	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 190))
GLR (→ page 165)	The synchro controller is a controller with PID characteristics
INC_ENCODER (→ page 136)	Up/down counter function for the evaluation of encoders
INPUT_ANALOG (→ page 127)	Current and voltage measurement on the analogue input channel
INPUT_CURRENT (→ page 128)	Current measurement on the analogue input channel
INPUT_VOLTAGE (→ page 129)	Voltage measurement on the analogue input channel
MEMCPY (→ page 184)	Writes and reads different data types directly in the memory
NORM (→ page 131)	Normalises a value [WORD] within defined limits to a value with new limits
PERIOD (→ page 138)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel

Function element	Short description
PERIOD_RATIO (→ page 140)	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 142)	Reads a pair of channels with fast inputs and compares the phase position of the signals
PID1 (→ page 167)	PID controller
PID2 (→ page 169)	PID controller
PT1 (→ page 171)	Controlled system with first-order delay
PWM (→ page 144)	Initialises and configures a PWM-capable output channel Definition of the PWM frequency via RELOAD
PWM1000 (→ page 148)	Initialises and configures a PWM-capable output channel the mark-to-space ratio can be indicated in steps of 1 ‰
SERIAL_PENDING (→ page 115)	Determines the number of data bytes stored in the serial receive buffer
SERIAL_RX (→ page 116)	Reads a received data byte from the serial receive buffer at each call
SERIAL_SETUP (→ page 117)	Initialises the serial RS232 interface
SERIAL_TX (→ page 118)	Transmits one data byte via the serial RS232 interface
SET_DEBUG (→ page 189)	organises the DEBUG mode or the monitoring mode (depending on the TEST input)
SET_IDENTITY (→ page 190)	Sets an application-specific program identification
SET_INTERRUPT_I (→ page 121)	Conditional execution of a program part after an interrupt request via a defined input channel
SET_INTERRUPT_XMS (→ page 124)	Conditional execution of a program part at an interval of x milliseconds
SET_PASSWORD (→ page 191)	Sets a user password for access control to program and memory upload
SOFTRESET (→ page 173)	leads to a complete reboot of the device
TIMER_READ (→ page 175)	Reads out the current system time in [ms] Max. value = 49d 17h 2min 47s 295ms
TIMER_READ_US (→ page 176)	Reads out the current system time in [µs] Max. value = 1h 11min 34s 967ms 295µs

5.1.2 Library ifm_CR0303_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 80)	Handles the device-specific error status of the CANopen master on CAN interface x x = 1 = number of the CAN interface
CANx_MASTER_SEND_EMERGENCY (→ page 81)	Sends application-specific error status of the CANopen master on CAN interface x x = 1 = number of the CAN interface
CANx_MASTER_STATUS (→ page 83)	Status indication on CAN interface x of the device used as CANopen master x = 1 = number of the CAN interface

5.1.3 Library ifm_CR0303_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 90)	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 = number of the CAN interface
CANx_SLAVE_NODEID (→ page 91)	Enables setting of the node ID of a CANopen slave on CAN interface x at runtime of the application program x = 1 = number of the CAN interface
CANx_SLAVE_SEND_EMERGENCY (→ page 92)	Sends application-specific error status of the CANopen slave on CAN interface x x = 1 = number of the CAN interface
CANx_SLAVE_STATUS (→ page 94)	Shows the status of the device used as CANopen slave on CAN interface x x = 1 = number of the CAN interface

5.1.4 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 66)	Initialises CAN interface 1 also for the exended mode Set the mode and baud rate
CAN1_EXT_ERRORHANDLER (→ page 67)	Executes a "manual" bus recovery on CAN interface 1
CAN1_EXT_RECEIVE (→ page 68)	CAN interface 1: Configures a data receive object and reads out the receive buffer of the data object
CANx_EXT_RECEIVE_ALL (→ page 73)	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 70)	Transfers a CAN data object (message) to CAN interface 1 for transmission at each call

5.1.5 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 103)	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 104)	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 106)	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 108)	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 110)	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 112)	CAN interface x: sends individual messages or message blocks x = 1...n = number of the CAN interface (depending on the device, → Data sheet)

5.1.6 Library ifm_hydraulic_CR0303_V01yyzz.LIB

18716

This library contains the function blocks for hydraulic controls.

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

Function element	Short description
JOYSTICK_0 (→ page 151)	Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0... 1000
JOYSTICK_1 (→ page 154)	Scales signals [INT] from a joystick D standardised to 0... 1000
JOYSTICK_2 (→ page 158)	Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation
NORM_HYDRAULIC (→ page 161)	Normalises a value [DINT] within defined limits to a value with new limits

5.2 ifm function elements for the device CR0303

Contents

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

Contents

13754

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

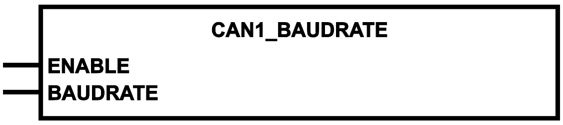
CAN1_BAUDRATE

651

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

654

CAN1_BAUDRATE sets the transmission rate for the bus participant.

► To do so, the corresponding value in kbits/s is entered at the input BAUDRATE.

NOTICE

Please note for CR250n, CR0301, CR0302 and CS0015:

The EEPROM memory module may be destroyed by the permanent use of this unit!

► Only carry out the unit **once** during initialisation in the first program cycle!

► Afterwards block the unit again with ENABLE = FALSE!

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

ExtendedController: In the slave module, the new baud rate will become effective after voltage OFF/ON.

Parameters of the inputs

655

Parameter	Data type	Description
ENABLE	BOOL	TRUE (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

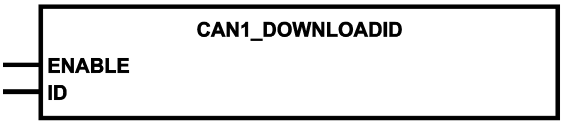
645

= CAN1 download ID

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

648

CAN1_DOWNLOADID sets the download identifier for the first CAN interface.

Using the FB the communication identifier for the program download and for debugging can be set. The new value is entered when the input ENABLE is set to TRUE. The new download ID will become effective after voltage OFF/ON or after a soft reset.

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

NOTICE

Please note for CR250n, CR0301, CR0302 and CS0015:

The EEPROM memory module may be destroyed by the permanent use of this unit!

► Only carry out the unit **once** during initialisation in the first program cycle!

► Afterwards block the unit again with ENABLE = FALSE!

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)

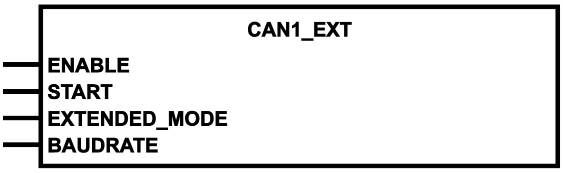
CAN1_EXT

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

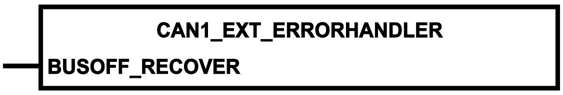
CAN1_EXT_ERRORHANDLER

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

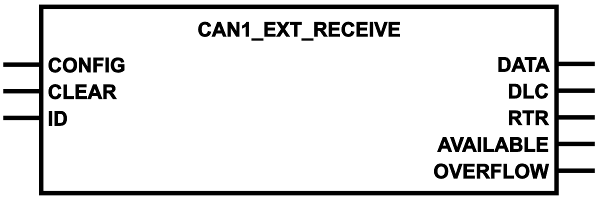
CAN1_EXT_RECEIVE

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 66).

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

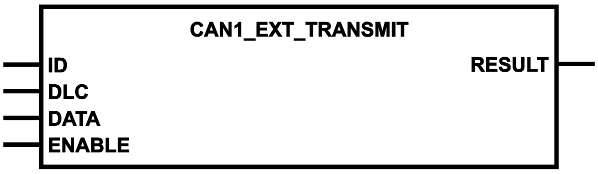
CAN1_EXT_TRANSMIT

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 66).

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

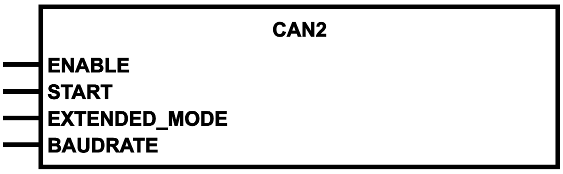
CAN2

639

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

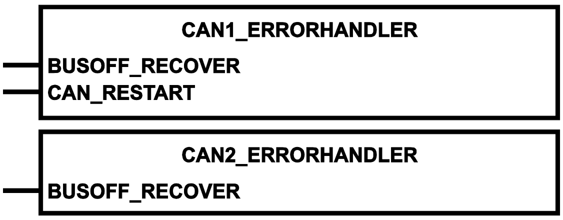
633

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_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

636

Error routine for monitoring the CAN interfaces

CANx_ERRORHANDLER monitors the CAN interfaces and evaluates the CAN errors. If a certain number of transmission errors occurs, the CAN participant becomes error passive. If the error frequency decreases, the participant becomes error active again (= normal condition).

If a participant already is error passive and still transmission errors occur, it is disconnected from the bus (= bus off) and the error bit CANx_BUSOFF is set. Returning to the bus is only possible if the "bus off" condition has been removed (signal BUSOFF_RECOVER).

The input CAN_RESTART is used for rectifying other CAN errors. The CAN interface is reinitialised.

Afterwards, the error bit must be reset in the application program.

The procedures for the restart of the interfaces are different:

- For CAN interface 1 or devices with only one CAN interface:
  set the input CAN_RESTART = TRUE (only 1 cycle)
- For CAN interface 2:
  set the input START = TRUE (only 1 cycle) in *CAN2* (→ page 71)

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

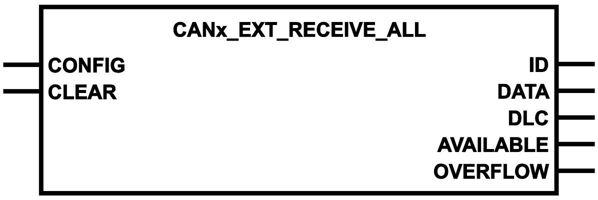
4183

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

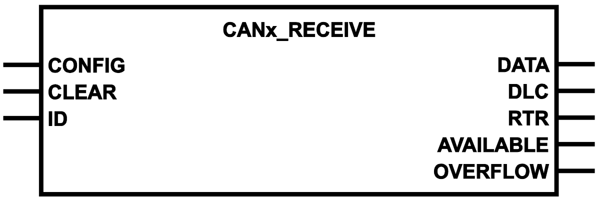
627

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_CR0303_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* (→ page 71).

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

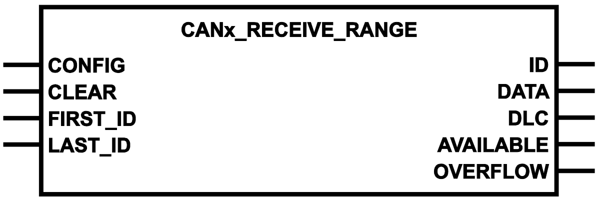
4179

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_CR0303_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 77).

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 74) 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!

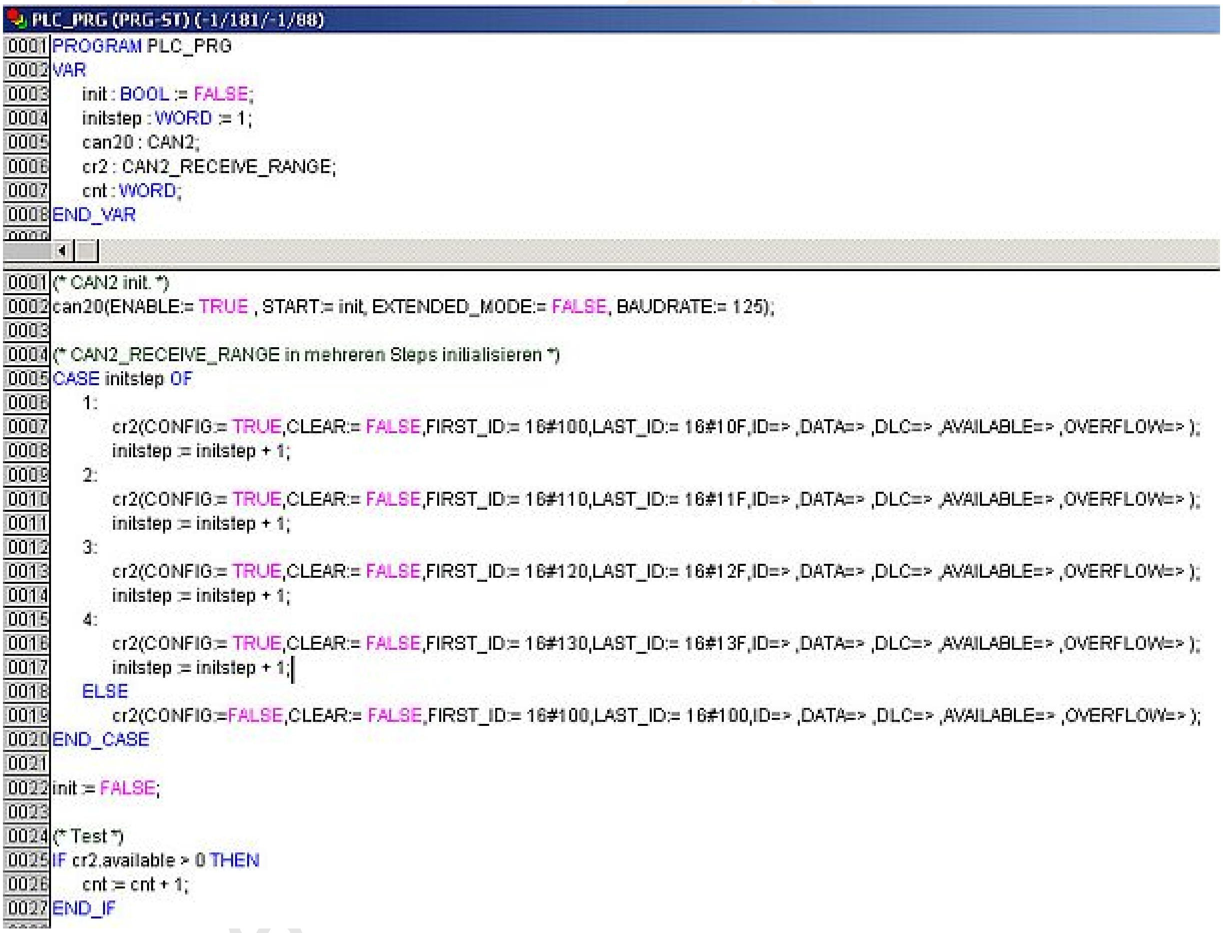
Parameters of the outputs

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



```
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

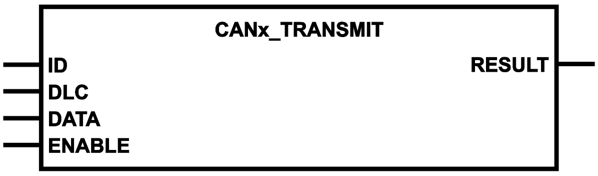
609

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_CR0303_Vxxyyzz.LIB`

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* (→ page 71).

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

Contents

1870

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

CANx_MASTER_EMCY_HANDLER

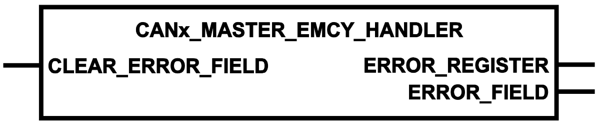
13192

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_CR0303_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 81).

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

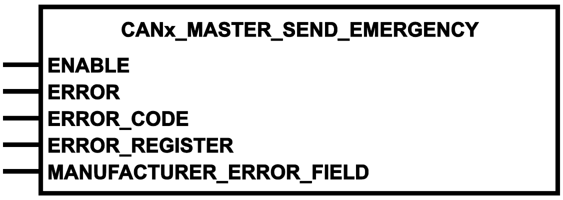
13195

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_CR0303_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.

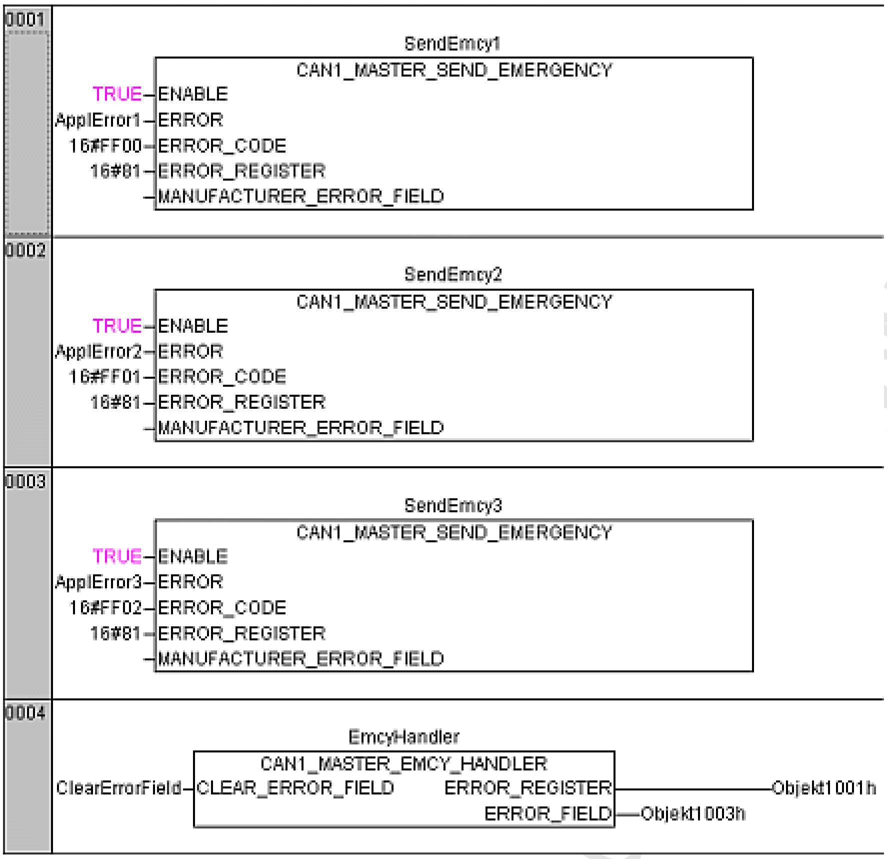
> ❗ If application-specific error messages are to be stored in the object directory, *CANx_MASTER_EMCY_HANDLER* (→ page 80) must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY.

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

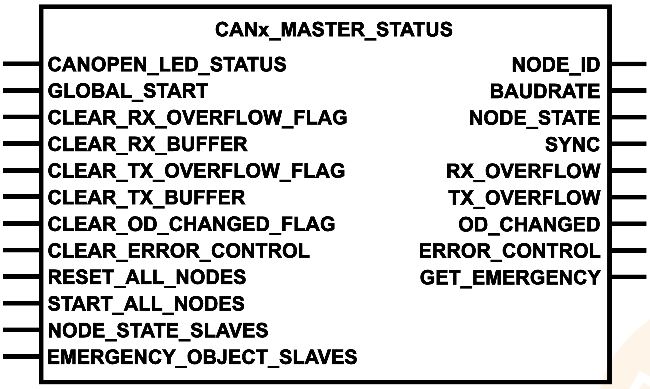
2021

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_CR0303_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 87)
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 88)

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) ℹ → chapter *Access to the structures at runtime of the application* (→ page 88)
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_CR0303_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_CR0303_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

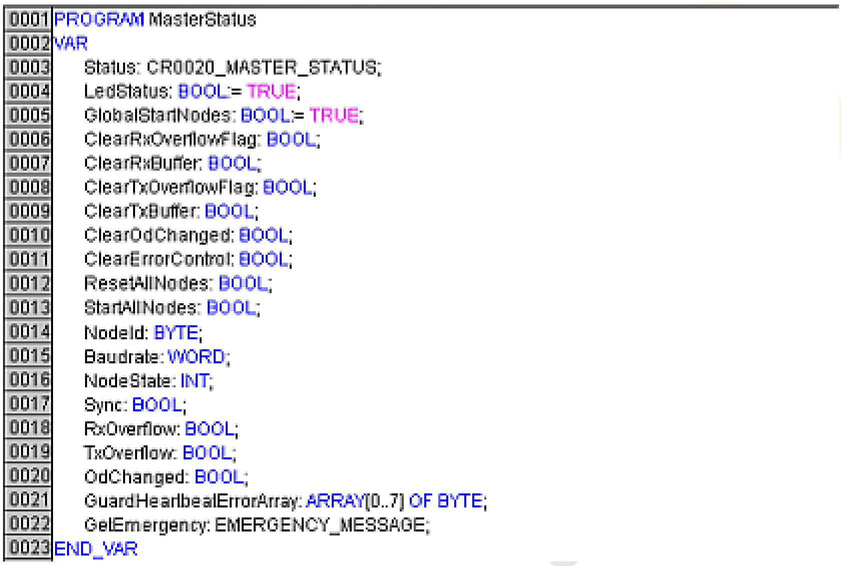
Slave information

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.

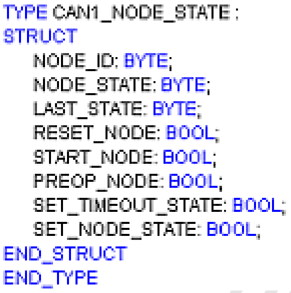
> ! The numbers of the array elements do **not** correspond to the node ID. The identifier can be read from the corresponding structure under NODE_ID.



```
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

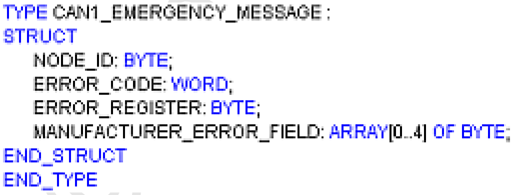
2034



```
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

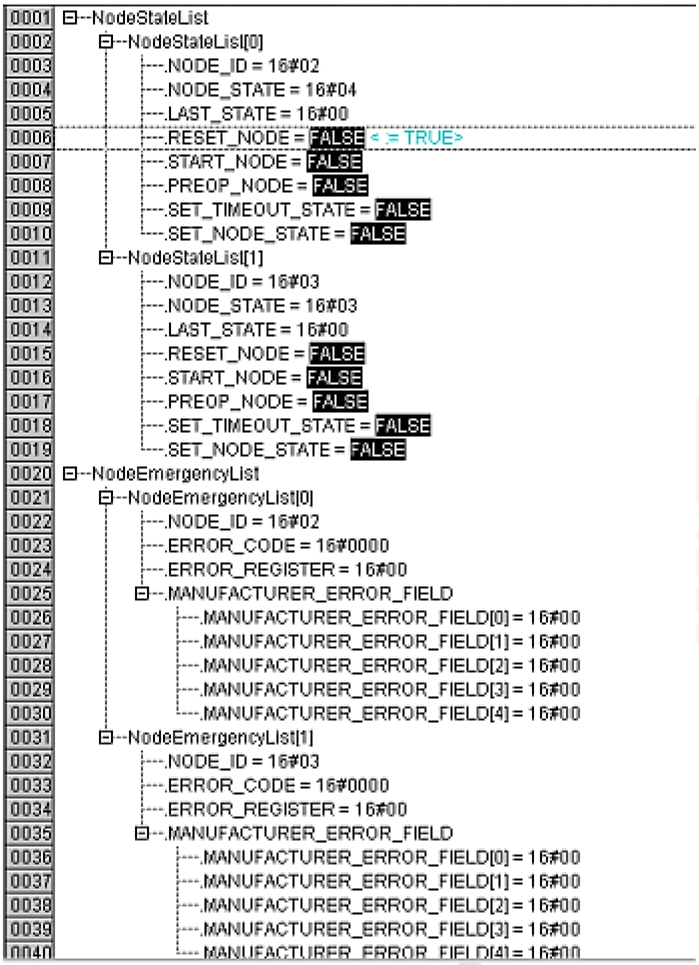


```
TYPE CAN1_EMERGENCY_MESSAGE :
STRUCT
    NODE_ID: BYTE;
    ERROR_CODE: WORD;
    ERROR_REGISTER: BYTE;
    MANUFACTURER_ERROR_FIELD: ARRAY[0..4] OF BYTE;
END_STRUCT
END_TYPE
```

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

Contents

1874

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

CANx_SLAVE_EMCY_HANDLER

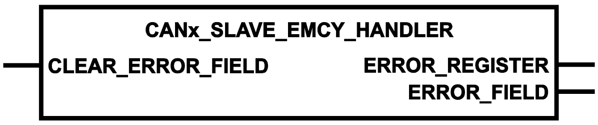
13199

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_CR0303_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 92) 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

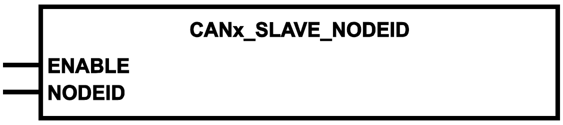
= CANx Slave Node-ID

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_CR0303_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

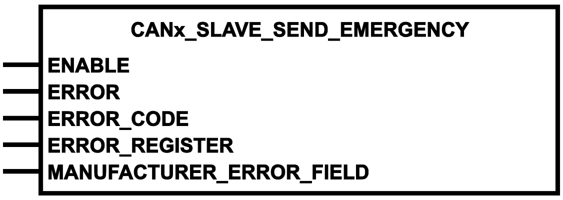
13205

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_CR0303_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.

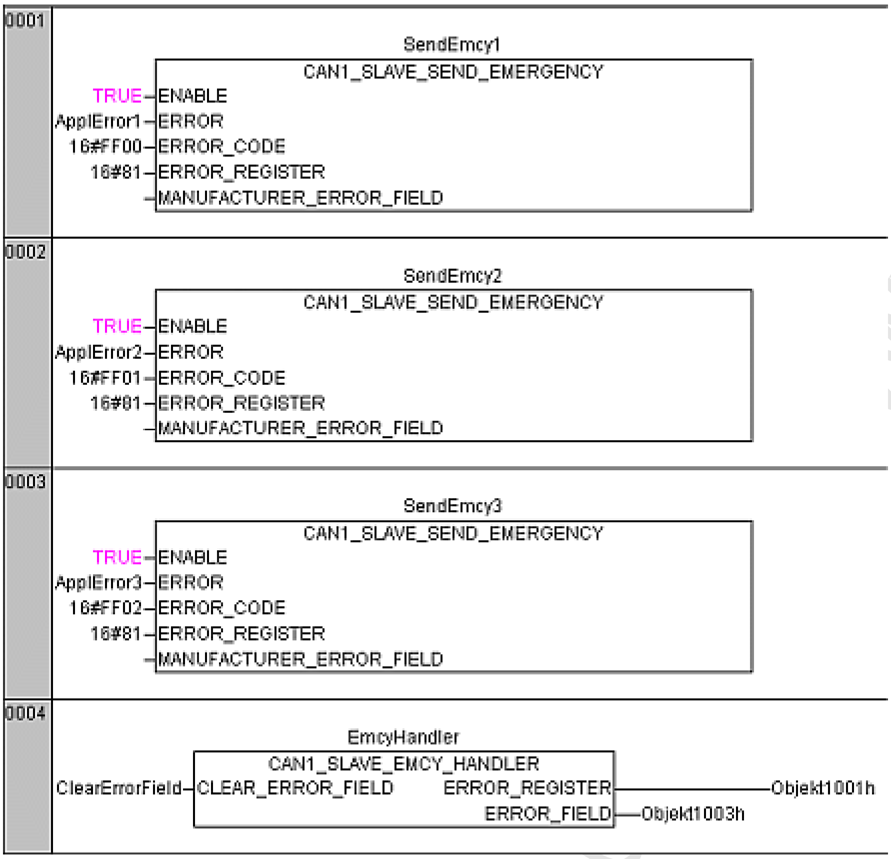
Parameters of the inputs

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

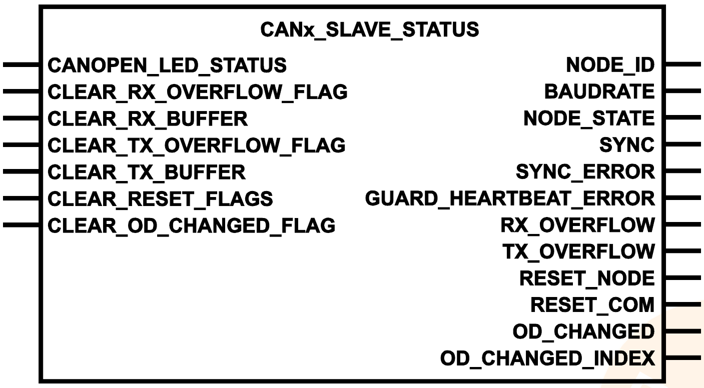
2063

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_CR0303_CANopenSlave_Vxxyyzz.LIB`

Symbol in CODESYS:



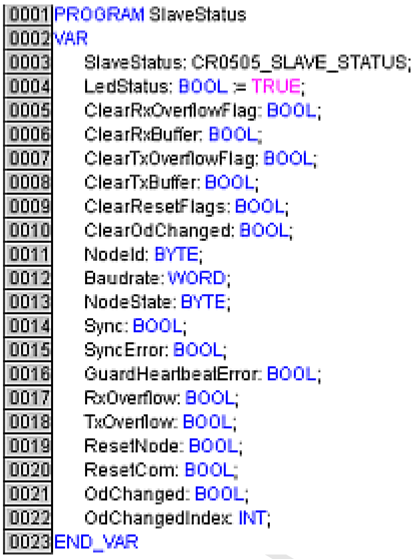
Description

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

Contents

2071

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

CANx_SDO_READ

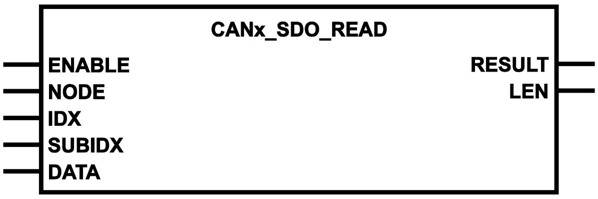
621

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_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

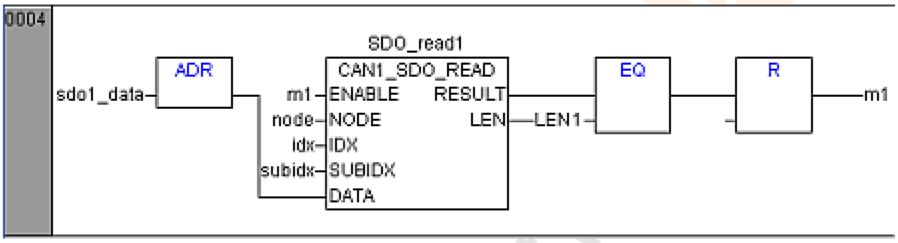
624

CANx_SDO_READ reads the →*SDO* (→ page 217) 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

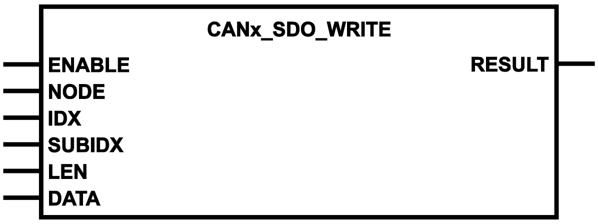
615

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_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

618

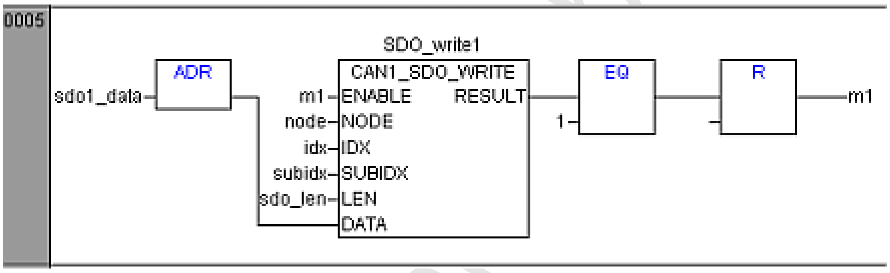
CANx_SDO_WRITE writes the →*SDO* (→ page 217) with the specified indexes to the node.

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

Contents

2273

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

J1939_x

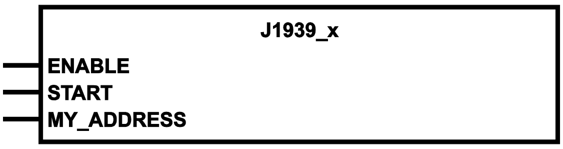
9375

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 66)!

J1939 communication via the 2nd CAN interface:

▶ First initialise the interface via *CAN2* (→ page 71)!

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

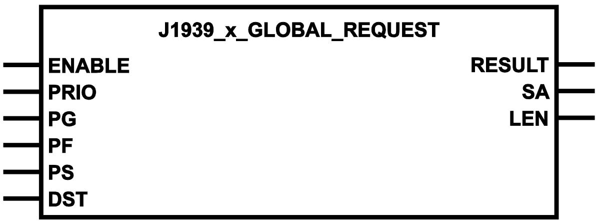
4315

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

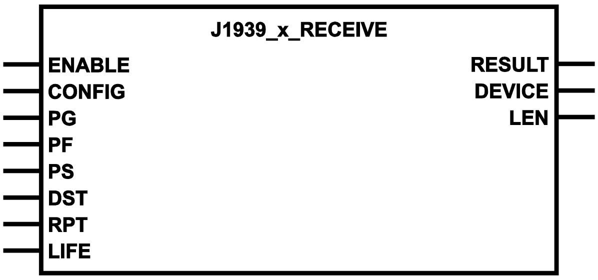
9393

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		Description
dec	hex	
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

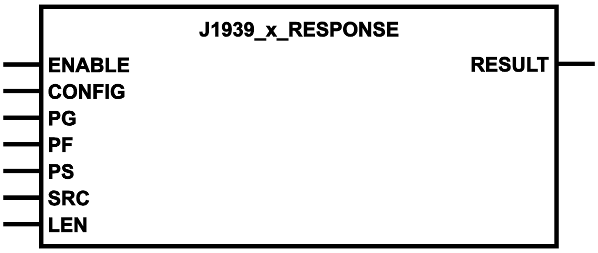
9399

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

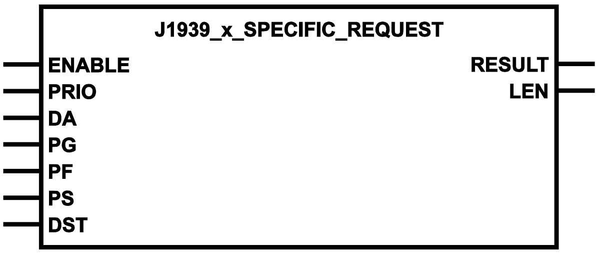
8884

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

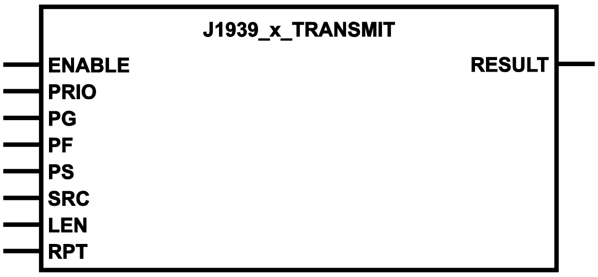
4322

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.

> ℹ **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.

ℹ 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

Contents

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.

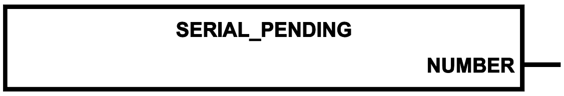
SERIAL_PENDING

314

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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 116) 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

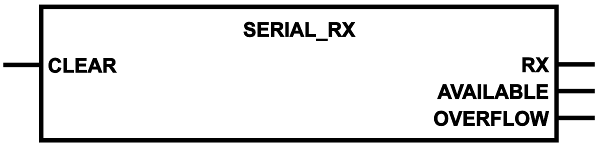
SERIAL_RX

308

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

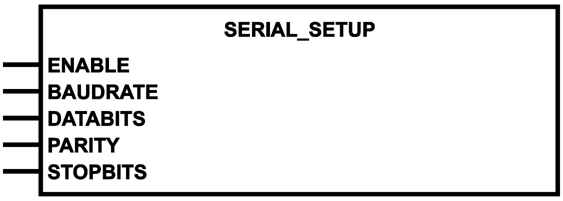
SERIAL_SETUP

302

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

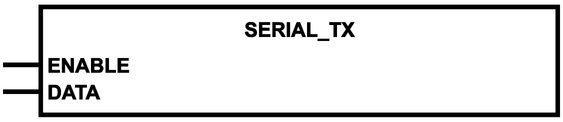
SERIAL_TX

296

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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 Function elements: Optimising the PLC cycle

Contents

8609

Here we show you functions to optimise the PLC cycle.

Function elements: processing interrupts

Contents

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.

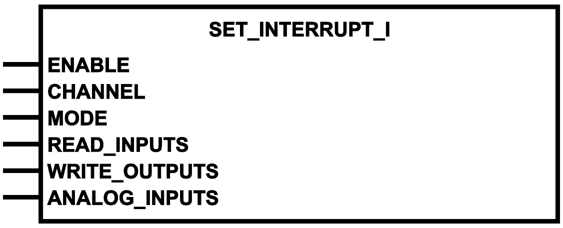
SET_INTERRUPT_I

2381

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

20089

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 the interrupt input (0...3) 0...3 for the inputs IN08...IN11
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 8...11 before calling the program and write to the input flags IN08...IN11 FALSE: only read the channel indicated under CHANNEL and write to the corresponding input flag INnn
WRITE_OUTPUTS	BOOL	TRUE: write the current values of the output flags Q00...Q03 to the outputs after completion of the program sequence FALSE: do not write outputs
ANALOG_INPUTS	BOOL	TRUE: read inputs 16...23 and write the unfiltered, uncalibrated analogue values to the flags ANALOG_IRQ16...23 FALSE: do not write the flags A_IN16...23

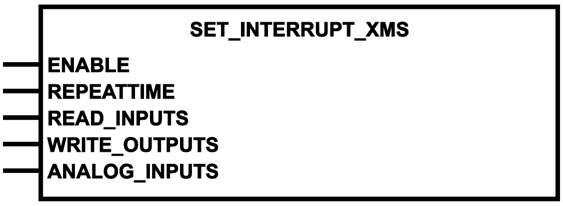
SET_INTERRUPT_XMS

272

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

20095

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 8...11 before calling the program and write to the input flags IN08...IN11 FALSE: only read the channel indicated under CHANNEL and write to the corresponding input flag INnn
WRITE_OUTPUTS	BOOL	TRUE: write the current values of the output flags Q00...Q03 to the outputs after completion of the program sequence FALSE: do not write outputs
ANALOG_INPUTS	BOOL	TRUE: read inputs 16...23 and write the unfiltered, uncalibrated analogue values to the flags ANALOG_IRQ16...23 FALSE: do not write the flags A_IN16...23

5.2.8 Function elements: processing input values

Contents

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.

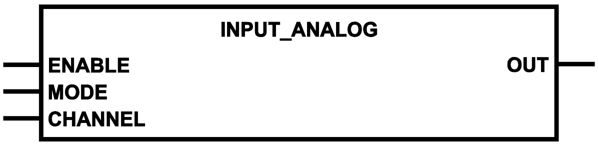
INPUT_ANALOG

519

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

20323

Parameter	Data type	Description		
ENABLE	BOOL	TRUE: FALSE:	execute this function element unit is not executed > Function block inputs are not active > Function block outputs are not specified	
MODE	BYTE	IN_DIGITAL_H IN_CURRENT IN_VOLTAGE10 IN_VOLTAGE30 IN_RATIO	Digital input Current input Voltage input Voltage input ratiometric analogue input	 0/20... 000 0...10 000 mV 0...30 000 mV
INPUT_CHANNEL	BYTE	Number of the input channel permissible = 16...23		

Parameters of the outputs

524

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

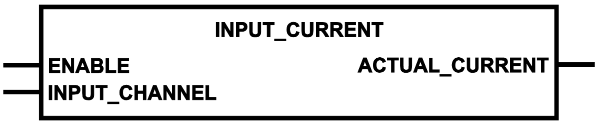
INPUT_CURRENT

513

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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 127) should be used.

Parameters of the inputs

20325

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 the input channel permissible = 16...23

Parameters of the outputs

518

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

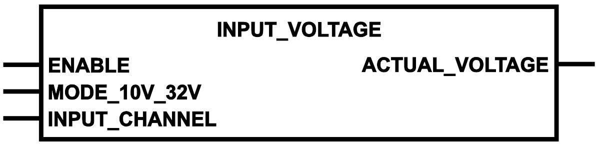
INPUT_VOLTAGE

507

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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 127) should be used.

Parameters of the inputs

20327

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 the input channel permissible = 16...23

Parameters of the outputs

512

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

5.2.9 Function elements: adapting analogue values

Contents

1603

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

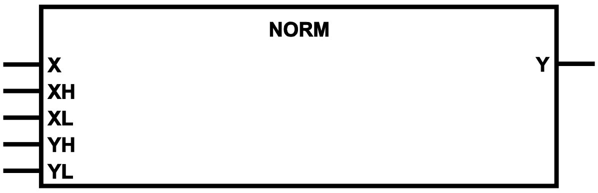
NORM

401

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

5.2.10 Function elements: counter functions for frequency and period measurement

Contents

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...30 000 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.

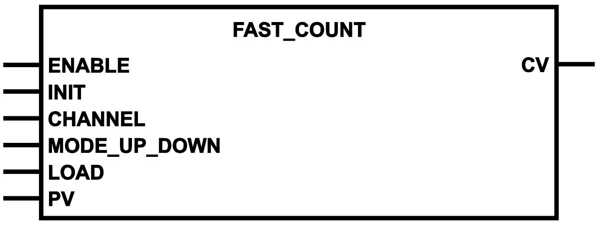
FAST_COUNT

20430

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

17812

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 IN08...IN11
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

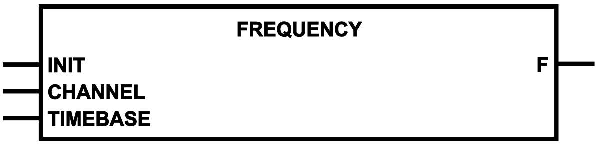
FREQUENCY

20604

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

17814

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 IN08...IN11
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

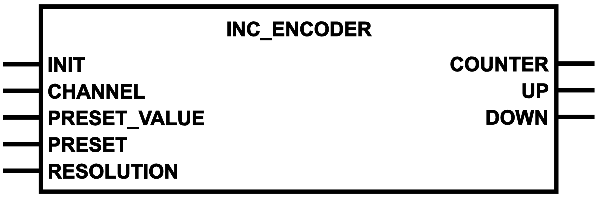
20432

= Incremental Encoder

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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)

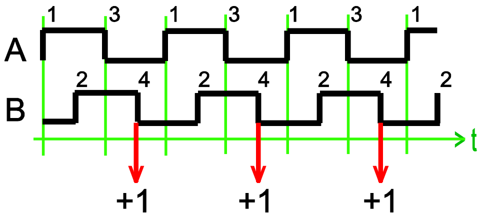
Set preset value:
1. Enter value in PRESET_VALUE
2. Set PRESET to TRUE for one cycle
3. Reset PRESET to FALSE

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.

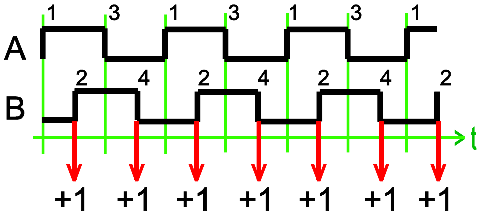
The outputs UP and DOWN indicate the current counting direction of the counter. The outputs are TRUE if the counter has counted in the corresponding direction in the preceding program cycle. If the counter stops, the direction output in the following program cycle is also reset.

Do **not** use this function block on one input together with one of the following function blocks!
- *FAST_COUNT* (→ page 134)
- *FREQUENCY* (→ page 135)
- *PERIOD* (→ page 138)
- *PERIOD_RATIO* (→ page 140)
- *PHASE* (→ page 142)

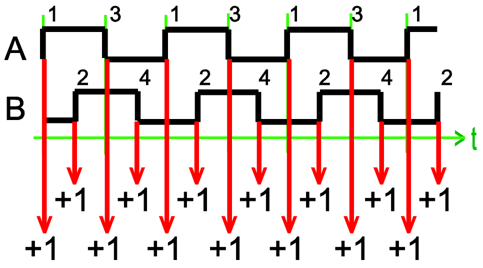
On input RESOLUTION the resolution of the encoder can be evaluated in multiples:
1 = normal resolution (identical with the resolution of the encoder),
2 = double evaluation of the resolution,
4 = 4-fold evaluation of the resolution.
All other values on this input mean normal resolution.



RESOLUTION = 1
In the case of normal resolution only the falling edge of the B-signal is evaluated.



RESOLUTION = 2
In the case of double resolution the falling and the rising edges of the B-signal are evaluated.



RESOLUTION = 4
In the case of 4-fold resolution the falling and the rising edges of the A-signal and the B-signal are evaluated.

Parameters of the inputs

17822

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 (0/1): 0 = channel pair 0 = inputs IN08 + IN09 1 = channel pair 1 = inputs IN10 + IN11
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

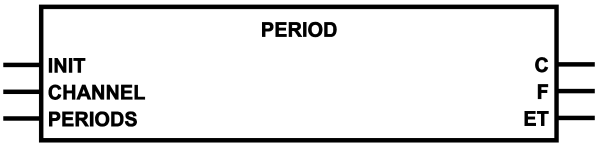
PERIOD

20606

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

17818

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 IN08...IN11
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)

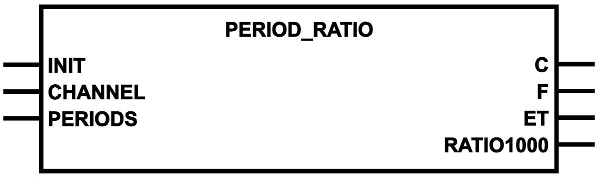
PERIOD_RATIO

20441

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

17820

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 IN08...IN11
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

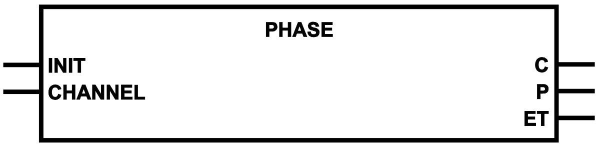
PHASE

20443

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

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

528

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 IN08 + IN09 1 = channel pair 1 = inputs IN10 + IN11

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

5.2.11 Function elements: PWM functions

Contents

13758

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

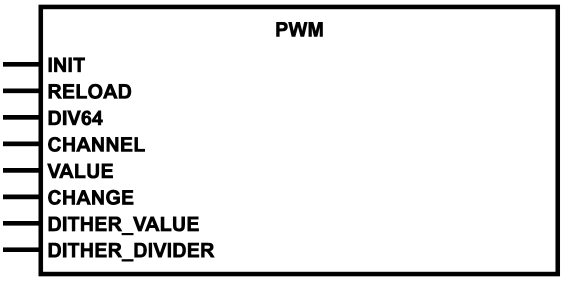
PWM

20457

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

13768

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 0...7.
>
> For these channels, PWM and *PWM1000* (→ page 148) 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 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

20612

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 146))
DIV64	BOOL	CPU cycle / 64
CHANNEL	BYTE	Number of the PWM output channel (0...7) 0...7 for the outputs OUT00...OUT07

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

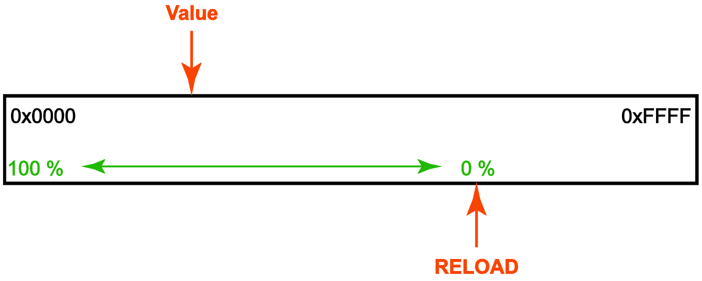


Figure: RELOAD value for the PWM channels 0...3

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 171). 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 **ecomat*mobile*** 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.

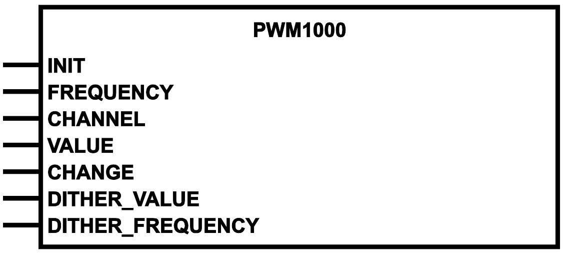
PWM1000

20465

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

7304

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 0...7.
>
> For these channels, *PWM* (→ page 144) 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 for example using the **ifm** module 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

17879

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...7) 0...7 for the outputs OUT00...OUT07
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.

5.2.12 Functions of the library ifm_HYDRAULIC_CR0303

9986

The library `ifm_HYDRAULIC_CR0303_Vxxyyzz.Lib` contains the following FBs:

JOYSTICK_0 (→ page 151)	Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0... 1000
JOYSTICK_1 (→ page 154)	Scales signals [INT] from a joystick D standardised to 0... 1000
JOYSTICK_2 (→ page 158)	Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation
NORM_HYDRAULIC (→ page 161)	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

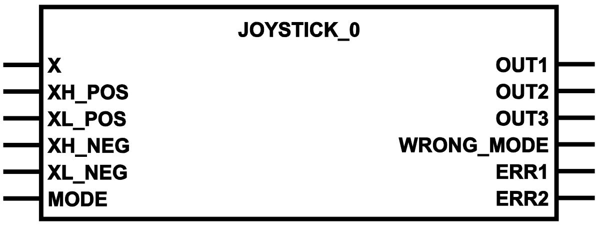
These function blocks are automatically called and configured by the function blocks of the hydraulics library.

JOYSTICK_0

13225

Unit type = function block (FB)
Unit is contained in the library `ifm_hydraulic_CR0303_Vxxyyzz.Lib`

Symbol in CODESYS:



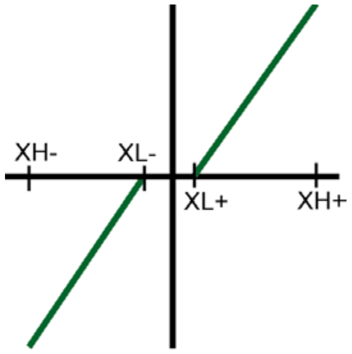
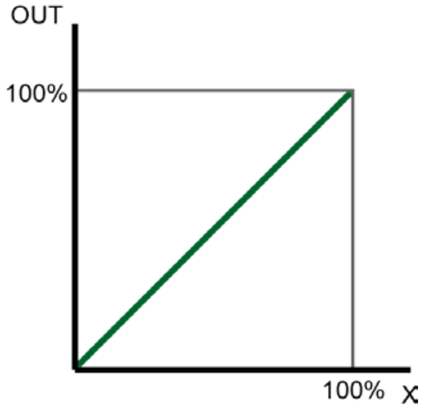
Description

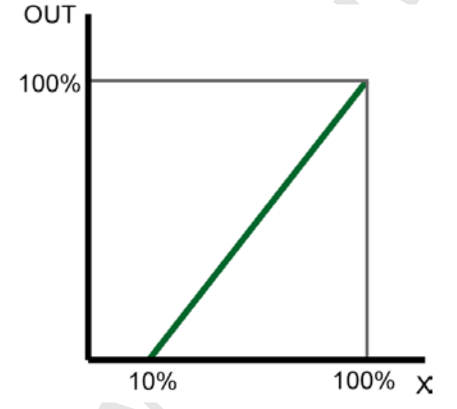
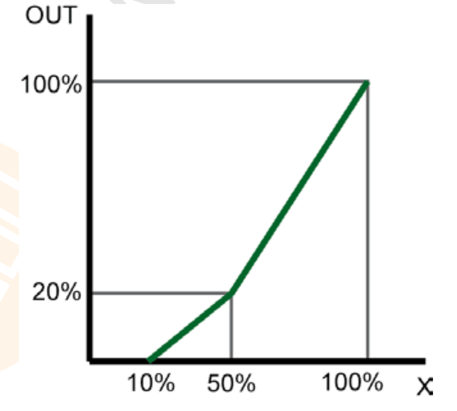
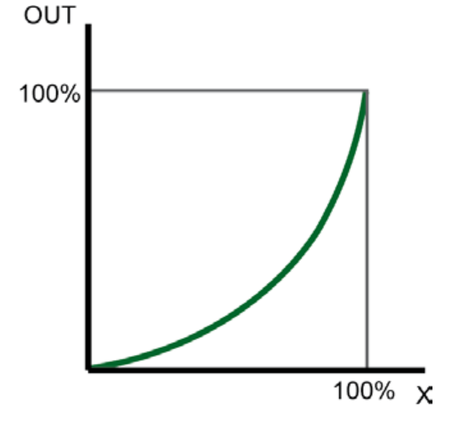
432

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:

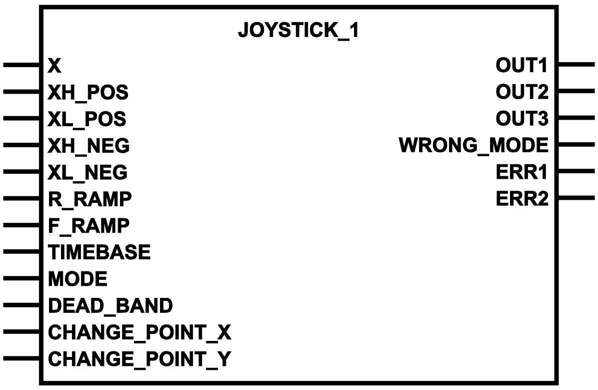
Value dec | hex		Description
0	00	no error
1	01	Error in array: wrong sequence
2	02	Error: Input value IN is not contained in value range of array
4	04	Error: invalid number N for array

JOYSTICK_1

13226

Unit type = function block (FB)
Unit is contained in the library `ifm_hydraulic_CR0303_Vxxyyzz.Lib`

Symbol in CODESYS:

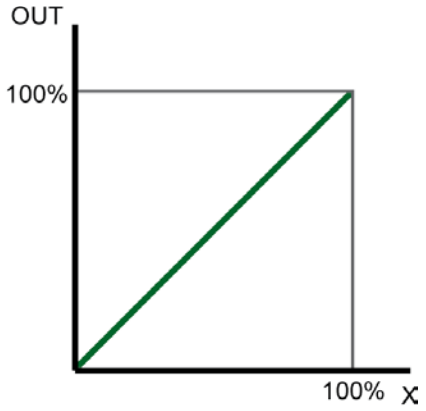
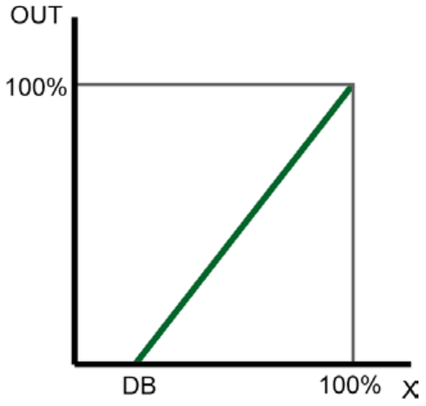


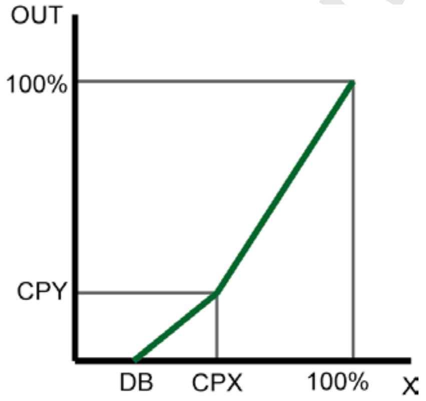
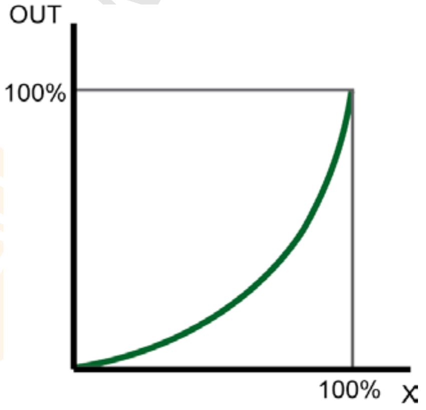
Description

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:

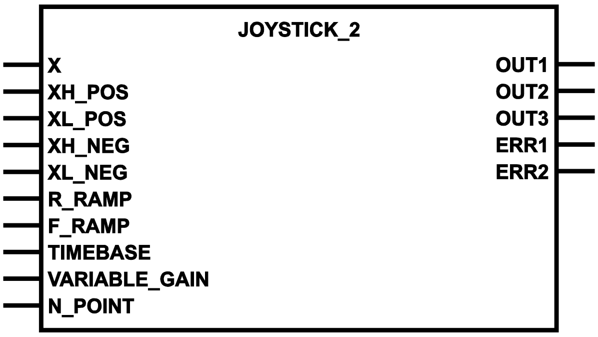
Value dec | hex		Description
0	00	no error
1	01	Error in array: wrong sequence
2	02	Error: Input value IN is not contained in value range of array
4	04	Error: invalid number N for array

JOYSTICK_2

13229

Unit type = function block (FB)
Unit is contained in the library `ifm_hydraulic_CR0303_Vxxyyzz.Lib`

Symbol in CODESYS:

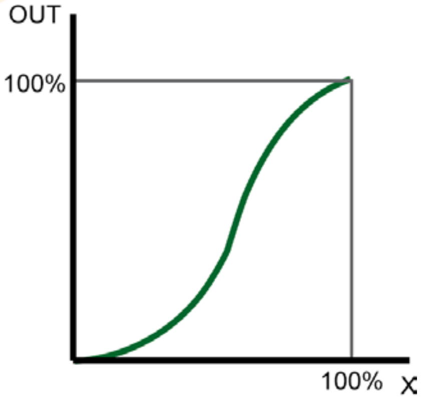


Description

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

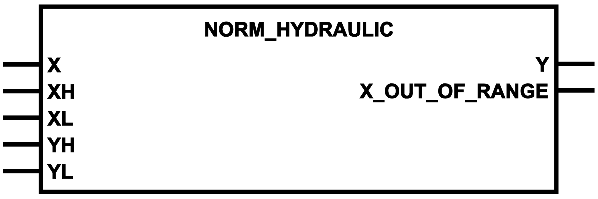
NORM_HYDRAULIC

13233

Unit type = function block (FB)

Unit is contained in the library `ifm_hydraulic_CR0303_Vxxyyzz.Lib`

Symbol in CODESYS:



Description

397

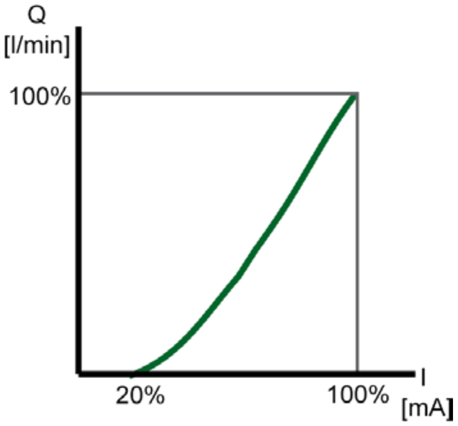
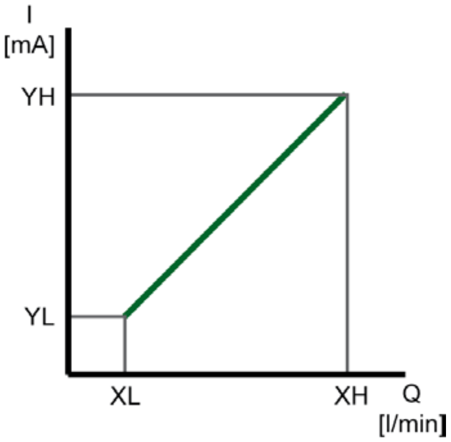
NORM_HYDRAULIC standardises input values with fixed limits to values with new limits.

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.

5.2.13 Function elements: controllers

Contents

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 171) (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.

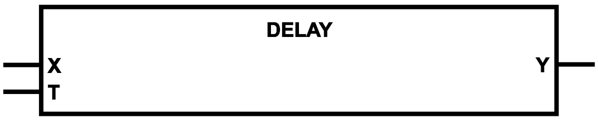
DELAY

585

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

588

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

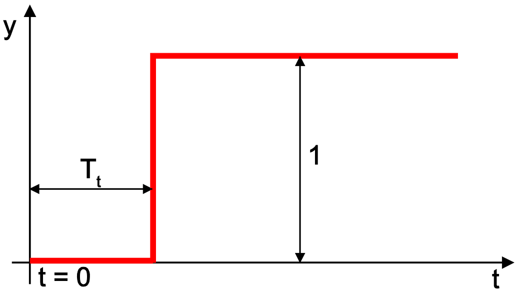


Figure: Time characteristics of DELAY

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

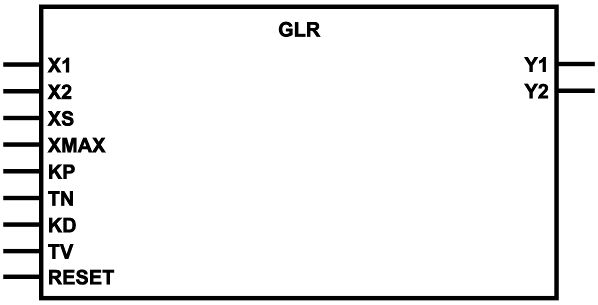
GLR

531

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

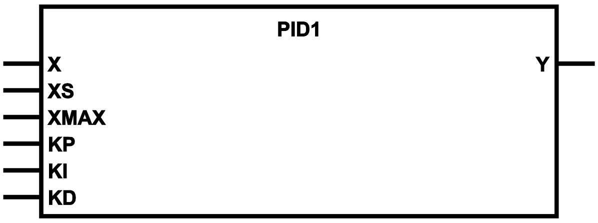
PID1

351

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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:

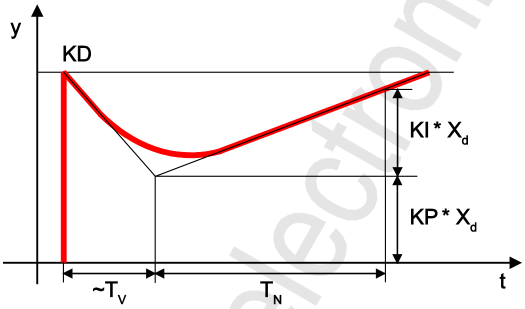


Figure: Typical step response of a PID controller

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.

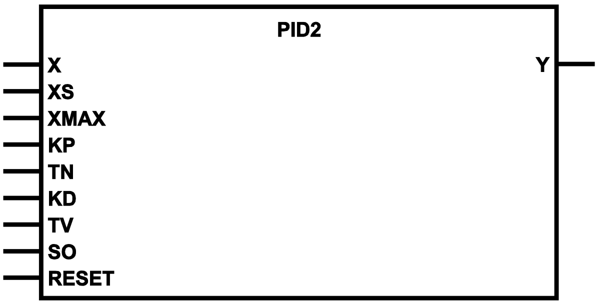
PID2

9167

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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.

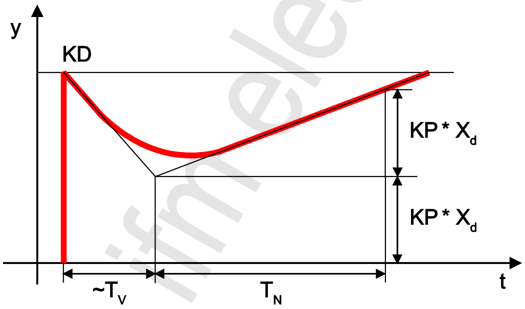


Figure: Typical step response of a PID controller

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.

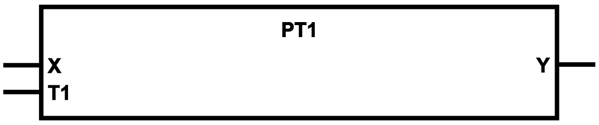
PT1

338

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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):

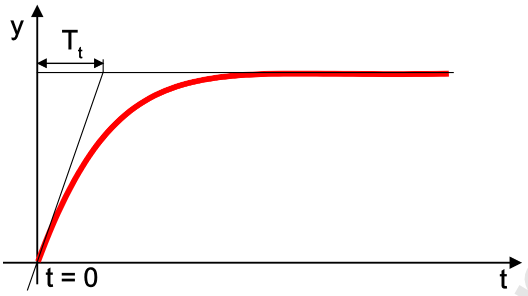


Figure: Time characteristics of PT1

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

5.2.14 Function elements: software reset

Contents

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_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:

SOFTRESET
ENABLE

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

5.2.15 Function elements: measuring / setting of time

Contents

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.

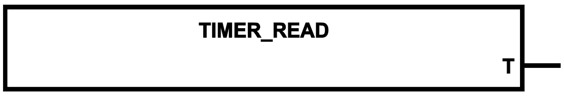
TIMER_READ

236

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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]

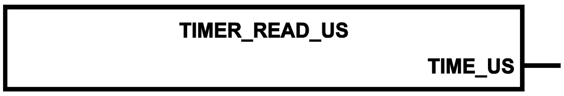
TIMER_READ_US

657

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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.

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]

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

Contents

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

Manual data storage

Contents

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 14)) the programmer can find out which memory area is available.

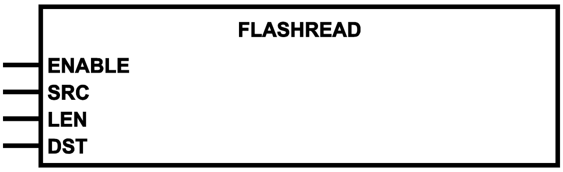
FLASHREAD

561

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

20045

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...16 383 = 0x0000...0x3FFF
LEN	WORD	number of data bytes permissible = 0...16 383 = 0x0000...0x3FFF
DST	DWORD	start address of the destination variable ❗ Determine the address by means of the operator ADR and assigne it to the FB!

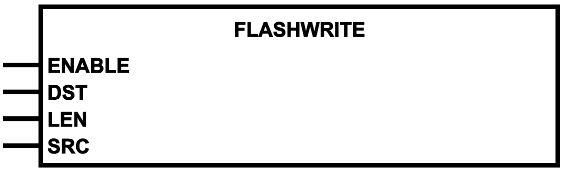
FLASHWRITE

555

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

20049

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	DWORD	relative start address in the memory memory acces only word by word permissible values: 0, 2, 4, 6, 8, ...
LEN	WORD	number of data bytes permissible = 0...16 383 = 0x0000...0x3FFF
SRC	DWORD	start address of the source variables ❗ Determine the address by means of the operator ADR and assigne it to the FB!

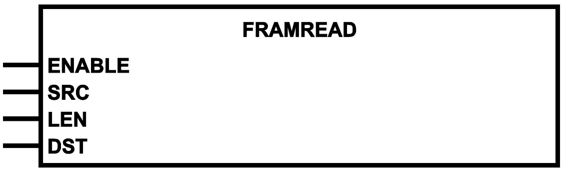
FRAMREAD

549

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

20065

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	INT	relative source start address in the memory permissible = 0...2 047 = 0x0000...0x07FF
LEN	INT	number of data bytes permissible = 0...128 = 0x0000...0x0080
DST	DINT	Start address of the target variables ❗ Determine the address by means of the operator ADR and assigne it to the FB!

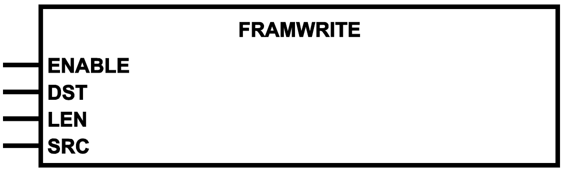
FRAMWRITE

543

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

20073

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	INT	relative destination start address in memory permissible = 512...2 047 = 0x0200...0x07FF
LEN	INT	number of data bytes permissible = 0...128 = 0x0000...0x0080
SRC	DINT	start address of the source variables ❗ Determine the address by means of the operator ADR and assigne it to the FB!

MEMCPY

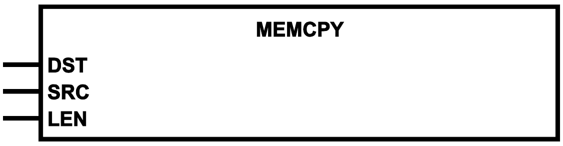
409

= memory copy

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

5.2.17 Function elements: data access and data check

Contents

1598

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

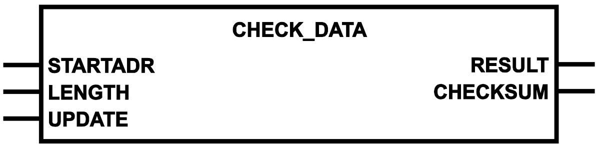
CHECK_DATA

603

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

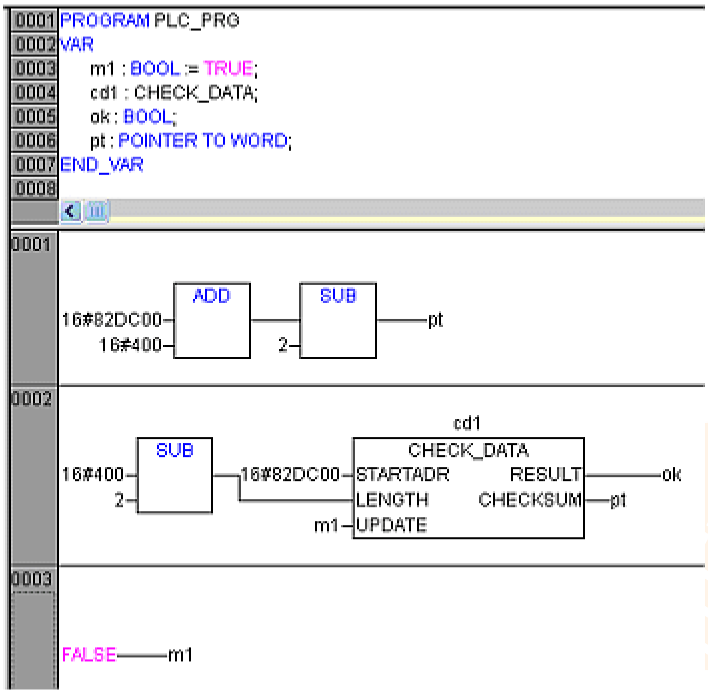
608

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.

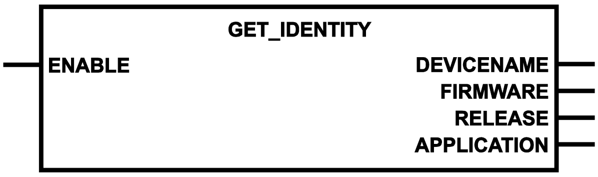
GET_IDENTITY

2212

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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 190))

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"

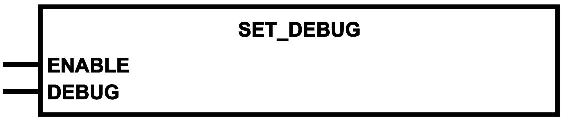
SET_DEBUG

290

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



Description

293

SET_DEBUG handles the DEBUG mode without active test input (→ chapter *TEST mode* (→ page 40)).

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

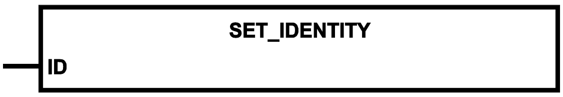
SET_IDENTITY

284

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_Vxxyyzz.LIB`

Symbol in CODESYS:



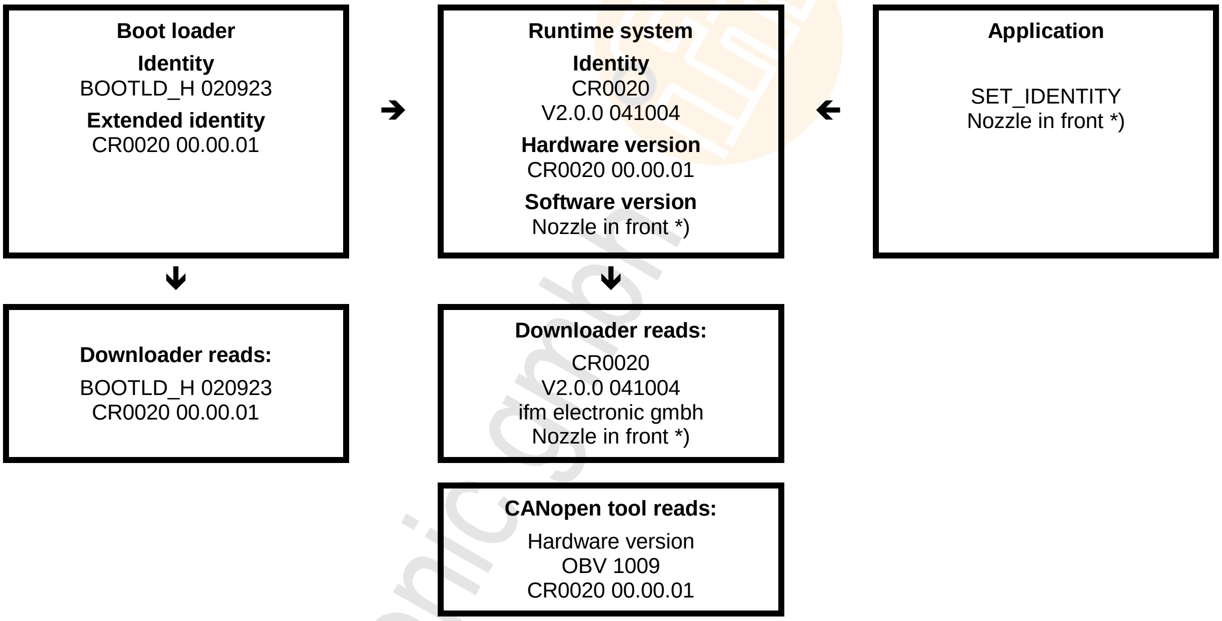
Description

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):



Boot loader

Identity
BOOTLD_H 020923

Extended identity
CR0020 00.00.01

Runtime system

Identity
CR0020
V2.0.0 041004

Hardware version
CR0020 00.00.01

Software version
Nozzle in front *)

Application

SET_IDENTITY
Nozzle in front *)

Downloader reads:
BOOTLD_H 020923
CR0020 00.00.01

Downloader reads:
CR0020
V2.0.0 041004
ifm electronic gmbh
Nozzle in front *)

CANopen tool reads:
Hardware version
OBV 1009
CR0020 00.00.01

*) 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

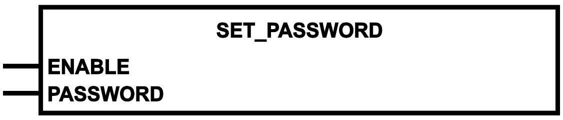
SET_PASSWORD

266

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0303_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

Contents

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 36)

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

19653

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,

ℹ Complete list of the device-specific error codes and diagnostic messages
→ chapter *System flags* (→ page 194).

6.4 Reaction in case of a system error

19654

❗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 sets the flag bit ERROR.
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.

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.

6.5 CAN / CANopen: errors and error handling

19604

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

7 Annex

Contents

1664

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

7.1 System flags

Contents

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

18830

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

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

7.1.2 System flags: error flags

18831

System flags (symbol name)	Type	Description
ERROR	BOOL	TRUE: safe state assumed all outputs = OFF output relays = OFF (e.g. fatal error / Error-Stop) FALSE: no serious error occurred

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_OVERCURRENT_OUTxx xx = 12...17	BOOL	Error: over-current on output OUTxx
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_TEMPERATURE	BOOL	Temperature error TRUE: Value out of range > general error FALSE: Value OK
WARNING_OVERCURRENT_OUTxx xx = 12...17	BOOL	Warning: over-current on output OUTxx

7.1.3 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)

7.1.4 System flags: voltages

833

System flags (symbol name)	Type	Description
REF_VOLTAGE	WORD	Value • 0.1 = voltage on the reference voltage output in [V]
REFERENCE_VOLTAGE_5	BOOL	Reference voltage output with 5 V activated
REFERENCE_VOLTAGE_10	BOOL	Reference voltage output with 10 V activated
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.
SERIAL_BAUDRATE	WORD	Baud rate of the RS232 interface
SUPPLY_VOLTAGE	WORD	Value • 0.1 = supply voltage on VBBS in [V]
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

7.1.5 System flags: 24 inputs and 18 outputs

18834

System flags (symbol name)	Type	Description
A_INxx xx = 00...15	WORD	Diagnosis value of input INxx: filtered A/D converter raw value (10 bits) without calibration or standardisation
A_INxx xx = 16...23	WORD	Analogue input xx: filtered A/D converter raw value (10 bits) without calibration or standardisation
A_INxx_MODE xx = 16...23	BYTE	Operating mode of the analogue input xx → chapter *Possible operating modes inputs/outputs* (→ page 203)
CURRxx xx = 12...17	WORD	PWM output xx: filtered A/D converter raw values (12 bits) of the current measurement without calibration or standardisation
INxx xx = 00...23	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
INxx_MODE xx = 00...15	BYTE	Operating mode of the input INxx → chapter *Possible operating modes inputs/outputs* (→ page 203)
OUTxx xx = 00...17	BOOL	Status on binary output xx: TRUE: output activated FALSE: output deactivated

7.2 Address assignment and I/O operating modes

Contents

1656

→ also data sheet

7.2.1 Address assignment inputs / outputs

Contents

2371

Inputs: address assignment

20099

Abbreviations →chapter *Note on wiring* (→ page 25)
Operating modes of the inputs/outputs →chapter *Possible operating modes inputs/outputs* (→ page 203)

IEC address	Symbolic address	Configuratiaon with ...	Note
%IX0.0	IN00	IN00_MODE	binary input
%IX0.1	IN01	IN01_MODE	binary input
%IX0.2	IN02	IN02_MODE	binary input
%IX0.3	IN03	IN03_MODE	binary input
%IX0.4	IN04	IN04_MODE	binary input
%IX0.5	IN05	IN05_MODE	binary input
%IX0.6	IN06	IN06_MODE	binary input
%IX0.7	IN07	IN07_MODE	binary input
%IX0.8	IN08	IN08_MODE	binary input
%IX0.9	IN09	IN09_MODE	binary input
%IX0.10	IN10	IN10_MODE	binary input
%IX0.11	IN11	IN11_MODE	binary input
%IX0.12	IN12	IN12_MODE	binary input
%IX0.13	IN13	IN13_MODE	binary input
%IX0.14	IN14	IN14_MODE	binary input
%IX0.15	IN15	IN15_MODE	binary input
%IX0.16	IN16	A_IN16_MODE	binary input
%IX0.17	IN17	A_IN17_MODE	binary input
%IX0.18	IN18	A_IN18_MODE	binary input
%IX0.19	IN19	A_IN19_MODE	binary input
%IX0.20	IN20	A_IN20_MODE	binary input
%IX0.21	IN21	A_IN21_MODE	binary input
%IX0.22	IN22	A_IN22_MODE	binary input
%IX0.23	IN23	A_IN23_MODE	binary input
%IW2	A_IN16	FB	analogue input
%IW3	A_IN17	FB	analogue input
%IW4	A_IN18	FB	analogue input
%IW5	A_IN19	FB	analogue input
%IW6	A_IN20	FB	analogue input
%IW7	A_IN21	FB	analogue input
%IW8	A_IN22	FB	analogue input
%IW9	A_IN23	FB	analogue input
%IW10	A_IN00	IN00_MODE	diagnostics of IN00
%IW11	A_IN01	IN01_MODE	diagnostics of IN01
%IW12	A_IN02	IN02_MODE	diagnostics of IN02
%IW13	A_IN03	IN03_MODE	diagnostics of IN03
%IW14	A_IN04	IN04_MODE	diagnostics of IN04

IEC address	Symbolic address	Configuratiaon with ...	Note
%IW15	A_IN05	IN05_MODE	diagnostics of IN05
%IW16	A_IN06	IN06_MODE	diagnostics of IN06
%IW17	A_IN07	IN07_MODE	diagnostics of IN07
%IW18	A_IN08	IN08_MODE	diagnostics of IN08
%IW19	A_IN09	IN09_MODE	diagnostics of IN09
%IW20	A_IN10	IN10_MODE	diagnostics of IN10
%IW21	A_IN11	IN11_MODE	diagnostics of IN11
%IW22	A_IN12	IN12_MODE	diagnostics of IN12
%IW23	A_IN13	IN13_MODE	diagnostics of IN13
%IW24	A_IN14	IN14_MODE	diagnostics of IN14
%IW25	A_IN15	IN15_MODE	diagnostics of IN15
%IW26	CURR12		current measurement on OUT12
%IW27	CURR13		current measurement on OUT13
%IW28	CURR14		current measurement on OUT14
%IW29	CURR15		current measurement on OUT15
%IW30	CURR16		current measurement on OUT16
%IW31	CURR17		current measurement on OUT17
%IW37	SUPPIY_VOLTAGE		
%IW38	REFERENCE_VOLTAGE		

Outputs: address assignment

20100

Abbreviations →chapter *Note on wiring* (→ page 25)
Operating modes of the inputs/outputs →chapter *Possible operating modes inputs/outputs* (→ page 203)

IEC address	Symbolic address	Configuratiaon with ...	Note
%QX0.0	OUT00		binary output PWM output
%QX0.1	OUT01		binary output PWM output
%QX0.2	OUT02		binary output PWM output
%QX0.3	OUT03		binary output PWM output
%QX0.4	OUT04		binary output PWM output
%QX0.5	OUT05		binary output PWM output
%QX0.6	OUT06		binary output PWM output
%QX0.7	OUT07		binary output PWM output
%QX0.8	OUT08		binary output
%QX0.9	OUT09		binary output
%QX0.10	OUT10		binary output
%QX0.11	OUT11		binary output
%QX0.12	OUT12		binary output
%QX0.13	OUT13		binary output
%QX0.14	OUT14		binary output
%QX0.15	OUT15		binary output
%QX0.16	OUT16		binary output
%QX0.17	OUT17		binary output
%QB3	IN00_MODE		configuration for IN00
%QB4	IN01_MODE		configuration for IN01
%QB5	IN02_MODE		configuration for IN02
%QB6	IN03_MODE		configuration for IN03
%QB7	IN04_MODE		configuration for IN04
%QB8	IN05_MODE		configuration for IN05
%QB9	IN06_MODE		configuration for IN06
%QB10	IN07_MODE		configuration for IN07
%QB11	IN08_MODE		configuration for IN08
%QB12	IN09_MODE		configuration for IN09
%QB13	IN10_MODE		configuration for IN10
%QB14	IN11_MODE		configuration for IN11
%QB15	IN12_MODE		configuration for IN12
%QB16	IN13_MODE		configuration for IN13

IEC address	Symbolic address	Configuratiaon with ...	Note
%QB17	IN14_MODE		configuration for IN14
%QB18	IN15_MODE		configuration for IN15
%QB19	A_IN16_MODE		configuration for A_IN16
%QB20	A_IN17_MODE		configuration for A_IN17
%QB21	A_IN18_MODE		configuration for A_IN18
%QB22	A_IN19_MODE		configuration for A_IN19
%QB23	A_IN20_MODE		configuration for A_IN20
%QB24	A_IN21_MODE		configuration for A_IN21
%QB25	A_IN22_MODE		configuration for A_IN22
%QB26	A_IN23_MODE		configuration for A_IN23
%QB27	REFERENCE_VOLTAGE_5		
%QB28	REFERENCE_VOLTAGE_10		

7.2.2 Possible operating modes inputs/outputs

Contents

2386

Inputs: operating modes

20103

Possible configuration combinations (where permissible) are created by adding the configuration values.

🟩 = this configuration value is default

Inputs	Possible operating mode		Set with ...	FB input	Value	
					dec	hex
IN00...07	IN_DIGITAL_H	plus	INxx_MODE	--	1	01
	IN_DIAGNOSTIC	bei IN_DIGITAL_H	INxx_MODE	--	64	40
IN08...11	IN_DIGITAL_H	plus	INxx_MODE	--	1	01
	IN_DIAGNOSTIC	if IN_DIGITAL_H	INxx_MODE	--	64	40
	IN_FAST	for Interrupt-FBs	INxx_MODE	--	128	80
	frequency measurement	0...30 000 Hz	FB FREQUENCY			
	period duration measurement	0.1...5 000 Hz	FB PERIOD			
	period duration and ratio measurement	0.1...5 000 Hz	FB PERIOD_RATIO			
	counter	0...50 Hz	FB FAST_COUNT			
	detect encoder	0...30 000 Hz	FB INC_ENCODER			
IN12...15	IN_DIGITAL_H	plus	INxx_MODE	--	1	01
	IN_DIGITAL_L	minus	INxx_MODE	--	2	02
	IN_DIAGNOSTIC	if IN_DIGITAL_H	INxx_MODE	--	64	40
IN16...23	IN_DIGITAL_H	plus	A_INxx_MODE	--	1	01
A_IN16...23	IN_CURRENT	0...20 000 µA	A_INxx_MODE	--	4	04
A_IN16...23	IN_VOLTAGE10	0...10 000 mV	A_INxx_MODE	--	8	08
A_IN16...23	IN_VOLTAGE32	0...32 000 mV	A_INxx_MODE	--	16	10
A_IN16...23	IN_RATIO32	0...1 000 ‰	A_INxx_MODE	--	32	20
IN16...23	IN_DIAGNOSTIC	if IN_DIGITAL_H	A_INxx_MODE	--	64	40

Set operating modes with the following function block:

FAST_COUNT (→ page 134)	Counter block for fast input pulses
FREQUENCY (→ page 135)	Measures the frequency of the signal arriving at the selected channel
INC_ENCODER (→ page 136)	Up/down counter function for the evaluation of encoders
PERIOD (→ page 138)	Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
PERIOD_RATIO (→ page 140)	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 [‰].

Outputs: operating modes

20104

🟩 = this configuration value is default

Outputs	Possible operating mode		Set with ...	FB input	Value	
					dec	hex
OUT00...07	binary output	plus switching	--	--	--	--
	analogue output with pulse-width modulation		PWM1000			
OUT08...11	binary output	plus switching	--	--	--	--
OUT12...17	binary output with diagnosis	plus switching	--	--	--	--

Set operating modes with the following function block:

PWM1000 (→ page 148)	Initialises and configures a PWM-capable output channel the mark-to-space ratio can be indicated in steps of 1 ‰

Outputs: permitted operating modes

20105

Operating mode		OUT00	OUT01	OUT02	OUT03	OUT04	OUT05	OUT06	OUT07
OUT_DIGITAL_H	plus	X	X	X	X	X	X	X	X
OUT_CURRENT_RANGE	4 A	X	X	X	X	X	X	X	X
PWM		X	X	X	X	X	X	X	X
OUT_OVERLOAD_PROTECTION		X	X	X	X	X	X	X	X

Operating mode		OUT08	OUT09	OUT10	OUT11	--	--	--	--
OUT_DIGITAL_H	plus	X	X	X	X	--	--	--	--
OUT_CURRENT_RANGE	4 A	X	X	X	X	--	--	--	--
OUT_OVERLOAD_PROTECTION		X	X	X	X	--	--	--	--

Operating mode		OUT12	OUT13	OUT14	OUT15	OUT16	OUT17	--	--
OUT_DIGITAL_H	plus	X	X	X	X	X	X	--	--
OUT_CURRENT_RANGE	10 A	X	X	X	X	X	X	--	--
OUT_DIAGNOSTIC		X	X	X	X	X	X	--	--
OUT_OVERLOAD_PROTECTION		X	X	X	X	X	X	--	--

7.3 Error tables

Contents

19606

7.3.1 Error flags

19608

→ chapter *System flags* (→ page 194)

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

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

2671

The following EMCY messages are sent automatically, if the FB *CANx_MASTER_EMCY_HANDLER* (→ page 80) 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					Diagnosis analogue current inputs
00	31	05						Terminal voltage VBBO/VBBS
00	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 • 10^{38}	3.402823466 • 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 211)

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**anagemen**t** = (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

11 ifm weltweit • ifm worldwide • ifm à l’échelle internationale

Version: 2015-03-06

8310

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.