Programming Manual

# ExtendedController
# CR0232

Runtime system V01.00.03
CODESYS V2.3

English

7390673 / 09 03 / 2019

# Contents

1 About this manual 5

1.1 Copyright ....5
1.2 Overview: documentation modules for....5
1.3 What do the symbols and formats mean? ....6
1.4 How is this documentation structured? ....6
1.5 History of the instructions (CR0232) ....7

2 Safety instructions 9

2.1 Please note! ....9
2.2 What previous knowledge is required? ....10
2.3 Start-up behaviour of the controller....10
2.4 Notes: serial number ....10
2.5 Notes: TEST inputs ....11

3 System description 12

3.1 Information concerning the device ....12
3.2 Hardware description ....13
3.2.1 Hardware structure....14
3.2.2 Operating principle of the delayed switch-off....17
3.2.3 Relays: important notes!....18
3.2.4 Monitoring concept ....19
3.2.5 Inputs (technology) ....23
3.2.6 Outputs (technology) ....27
3.2.7 Note on wiring ....32
3.2.8 Safety instructions about Reed relays ....32
3.2.9 Feedback in case of externally supplied outputs ....33
3.2.10 Status LED ....34
3.3 Interface description....36
3.3.1 Serial interface ....36
3.3.2 USB interface ....36
3.3.3 CAN interfaces ....37
3.4 Software description ....38
3.4.1 Software modules for the device ....39
3.4.2 Programming notes for CODESYS projects....41
3.4.3 Operating states....45
3.4.4 Operating modes....49
3.4.5 Performance limits of the device ....50

4 Configurations 51

4.1 Set up the runtime system ....52
4.1.1 Reinstall the runtime system ....52
4.1.2 Update the runtime system....53
4.1.3 Verify the installation ....53
4.2 Set up the programming system....54
4.2.1 Set up the programming system manually ....55
4.2.2 Set up the programming system via templates....57
4.3 Function configuration in general....57
4.3.1 Configuration of the inputs and outputs (default setting) ....57
4.3.2 System variables....57
4.4 Function configuration of the inputs and outputs ....58
4.4.1 Configure inputs ....59
4.4.2 Configure outputs....62

4.5 Variables ....66
4.5.1 Retain variables....66
4.5.2 Network variables....67

## 5 ifm function elements 68

5.1 ifm libraries for the device CR0232....69
5.1.1 Library ifm_CR0232_V010003.LIB....69
5.1.2 Library ifm_CR0232_CANopenxMaster_Vxxyyzz.LIB....71
5.1.3 Library ifm_CR0232_CANopenxSlave_Vxxyyzz.LIB....72
5.1.4 Library ifm_CR0232_J1939_Vxxyyzz.LIB ....72
5.1.5 Library ifm_hydraulic_32bit_Vxxyyzz.LIB ....73
5.2 ifm function elements for the device CR0232 ....74
5.2.1 Function elements: CAN layer 2....75
5.2.2 Function elements: CANopen master....84
5.2.3 Function elements: CANopen slave ....94
5.2.4 Function elements: CANopen SDOs ....102
5.2.5 Function elements: SAE J1939 ....107
5.2.6 Function elements: serial interface....119
5.2.7 Function elements: Optimising the PLC cycle via processing interrupts ....124
5.2.8 Function elements: processing input values....129
5.2.9 Function elements: adapting analogue values ....132
5.2.10 Function elements: counter functions for frequency and period measurement ....137
5.2.11 Function elements: PWM functions ....152
5.2.12 Function elements: hydraulic control ....162
5.2.13 Function elements: controllers....177
5.2.14 Function elements: software reset....184
5.2.15 Function elements: measuring / setting of time ....186
5.2.16 Function elements: device temperature....189
5.2.17 Function elements: saving, reading and converting data in the memory....191
5.2.18 Function elements: data access and data check....203

## 6 Diagnosis and error handling 211

6.1 Diagnosis ....211
6.2 Fault ....211
6.3 Reaction in case of an error....212
6.4 Relay: important notes!....212
6.5 Response to system errors....212
6.6 CAN / CANopen: errors and error handling ....213

## 7 Appendix 214

7.1 System flags....215
7.1.1 System flags: CAN ....215
7.1.2 System flags: SAE-J1939....216
7.1.3 System flags: error flags (standard side)....217
7.1.4 System flags: error flags (extended side) ....218
7.1.5 System flags: status LED (standard side)....219
7.1.6 System flags: status LED (extended side)....219
7.1.7 System flags: voltages (standard side)....220
7.1.8 System flags: voltages (extended side) ....221
7.1.9 System flags: 16 inputs and 16 outputs (standard side)....221
7.1.10 System flags: 16 inputs and 32 outputs (extended side) ....223
7.2 Address assignment and I/O operating modes....224
7.2.1 Addresses / variables of the I/Os....225
7.2.2 Possible operating modes inputs/outputs....233
7.3 Error tables....240
7.3.1 Error flags....240
7.3.2 Errors: CAN / CANopen....240

8 Terms and abbreviations 242

9 Index 256

# 1 About this manual

**Content**

Copyright ....................5
Overview: documentation modules for ....................5
What do the symbols and formats mean? ....................6
How is this documentation structured? ....................6
History of the instructions (CR0232) ....................7

26077

## 1.1 Copyright

26002

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™
- EtherCAT® is a registered trade mark and patented technology, licensed by Beckhoff Automation GmbH, Germany
- IO-Link® (→ www.io-link.com) is the property of the →PROFIBUS Nutzerorganisation e.V., Germany
- ISOBUS is the property of the AEF – Agricultural Industry Electronics Foundation e.V., Deutschland (→ www.aef-online.org)
- Microsoft® is the property of the Microsoft Corporation, USA (→ www.microsoft.com)
- Modbus® is the property of the Schneider Electric SE, France (→ www.schneider-electric.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

28035

The documentation for this devices consists of the following modules:
(Downloads from ifm's website → www.ifm.com

| Document | Contents / Description |
|---|---|
| Data sheet | Technical data in a table |
| Installation instructions (are supplied with the device) | • Instructions for installation, electrical installation, and commissioning<br>• Technical data |
| Programming manual | • Functions of the setup menu of the device<br>• Creation of a CODESYS project with this device<br>• Target settings with CODESYS<br>• Programming of the device-internal PLC with CODESYS<br>• Description of the device-specific CODESYS function libraries |
| System manual "Know-How ecomatmobile" | Know-how about the following topics (examples):<br>• Overview Templates and demo programs<br>• CAN, CANopen<br>• Control outputs<br>• Visualisations<br>• Overview of the files and libraries |

## 1.3 What do the symbols and formats mean?

26329

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 note<br>Non-compliance can result in malfunction or interference |
| ℹ | Information<br>Supplementary note |
| ► ... | Request for action |
| > ... | Reaction, result |
| → ... | "see" |
| abc | Cross-reference |
| 123<br>0x123<br>0b010 | Decimal number<br>Hexadecimal number<br>Binary number |
| [...] | Designation of pushbuttons, buttons or indications |

## 1.4 How is this documentation structured?

204
26041

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 → Appendix.

In case of malfunctions or uncertainties please contact the manufacturer at:
Contact → www.ifm.com

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:
→ www.ifm.com

## 1.5 History of the instructions (CR0232)

9188

What has been changed in this manual? An overview:

| Date | Theme | Change |
|---|---|---|
| 2010-09-13 | configurations of Q16_MODE_E...Q31_MODE_E | default value 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 appendix |
| 2011-12-13 | INPUT_ANALOG | parameter MODE |
| 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 |
| 2014-06-24 | FB PID2 | Graphic corrected |
| 2014-06-30 | Name of the documentation | "System manual" renamed as "Programming manual" |
| 2014-07-04 | Device output ERROR (clamp 13) | Output is not available. Reference note removed. |
| 2014-07-31 | FB PHASE | Description of parameters of outputs C, ET corrected |
| 2014-07-31 | FB OUTPUT_CURRENT_CONTROL | If preset value = 0 mA >> control to 0 "within 100 ms" instead of "at once" |
| 2014-08-26 | Description of inputs, outputs | highside / lowside replaced by positive / negative switching |
| 2014-11-12 | Chapter "Outputs (technology)" | Section "Diagnostics of the binary outputs" supplemented or corrected |
| 2015-01-13 | Structure of documentation for error codes, system flags | • error flags:<br>now only in the appendix, chapter **System flags**<br>• CAN / CANopen errors and error handling:<br>now only in the system manual "Know-How"<br>• error codes, EMCY codes:<br>now in the appendix, 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 |
| 2015-07-27 | FB GET_IDENTITY | added with output SERIALNUMBER |
| 2015-07-27 | FB GET_IDENTITY_EIOS | new |
| 2015-09-22 | FB GET_IDENTITY_EIOS | corrected |
| 2015-09-22 | english manual | damaged images updated |
| 2015-10-22 | System flag bit SERIAL_MODE | Debugging of the application program via USB is not possible |
| 2016-04-27 | FBs for fast inputs | Note in case of higher frequencies added |
| 2017-01-13 | Software manual for CODESYS 2.3 | hint to download from the ifm homepage removed |
| 2017-06-02 | FRAM, MEMCPY, MEMSET:<br>Declaration for "remanent memory freely available to the user" | removed from manual, because value and start address depend from hardware and software of the device |
| 2017-06-02 | Fast inputs | Internal resistance of the signal source must be substantially lower than the input resistance of the used input |

| Date | Theme | Change |
|---|---|---|
| 2017-12-08 | Addresses and variables of the I/Os | Declaration of the input bytes and output bytes removed because invalid |
| 2018-07-09 | List of the ifm branch offices | removed |
| 2019-03 | Outputs Q16_E...Q23_E / Q31_E | Description added: Diagnosis: binary outputs (via voltage measurement) |

# 2 Safety instructions

**Content**

Please note! ....................9
What previous knowledge is required? ....................10
Start-up behaviour of the controller ....................10
Notes: serial number ....................10
Notes: TEST inputs ....................11

28333

## 2.1 Please note!

214
28588

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

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

► Observe the installation and wiring information as well as the functions and features of the devices!
→ supplied installation instructions or on **ifm's** homepage

► Please note the corrections and notes in the release notes for the existing hardware, software and documentation, available on the **ifm** website

Website → www.ifm.com

28588

**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?

28341

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

⚠ **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 the watchdog responded because the cycle time was too long
- Error elimination after an E-stop

To ensure safe controller behaviour:

► monitor the voltage supply in the application program.

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

► Additionally monitor actuators which can cause hazardous movements in the application program (feedback).

► 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.

## 2.4 Notes: serial number

28582

► In the user's production facility, draw a diagram of the controller network in the machine. Enter the serial number of each controller installed into the network diagram.

► Before downloading a software component, read out this serial number and check the network diagram to make sure that you are accessing the right controller.

## 2.5 Notes: TEST inputs

28581

- ► The TEST inputs of all the controllers in the machine should be wired individually and marked clearly so that they can be properly allocated to the controllers.
- ► During a service access only activate the TEST input of the controller to be accessed.

# 3 System description

**Content**

Information concerning the device ..........12
Hardware description..........13
Interface description ..........36
Software description ..........38

28392

## 3.1 Information concerning the device

6269

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

- ExtendedController: CR0232

## 3.2 Hardware description

**Content**

Hardware structure ....................................................................................................14
Operating principle of the delayed switch-off ......................................................................17
Relays: important notes!..............................................................................................18
Monitoring concept ...................................................................................................19
Inputs (technology) ...................................................................................................23
Outputs (technology) .................................................................................................27
Note on wiring.........................................................................................................32
Safety instructions about Reed relays..................................................................................32
Feedback in case of externally supplied outputs .......................................................................33
Status LED ............................................................................................................34

28381

## 3.2.1 Hardware structure

**Content**

Start conditions ........ 14
Relays ........ 14
Principle block diagram ........ 15
Available memory ........ 16

28382

### Start conditions

28418

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

- permissible operating voltage = 8...32 V
- start condition: VBBs > 10 V

### Relays

19663

The ExtendedController has 4 internal output relays:

- standard side: 2 relays each separate 8 outputs from the terminal voltage VBBx (x=o|r),
- extended side: 2 relays each separate 16 outputs from the terminal voltage VBBx (x=1|2|3|4).

Separation is effected upon power-off of the relay.

The relays are only activated under the following condition:

- the global bit ERROR = FALSE

AND

- the bit RELAIS_VBBx = TRUE

In an active condition the relay contacts connect the outputs to the terminal voltage VBBx.

! Activate the corresponding outputs no earlier than ≥ 45 ms after power-on of the relays!

## Principle block diagram

19664

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

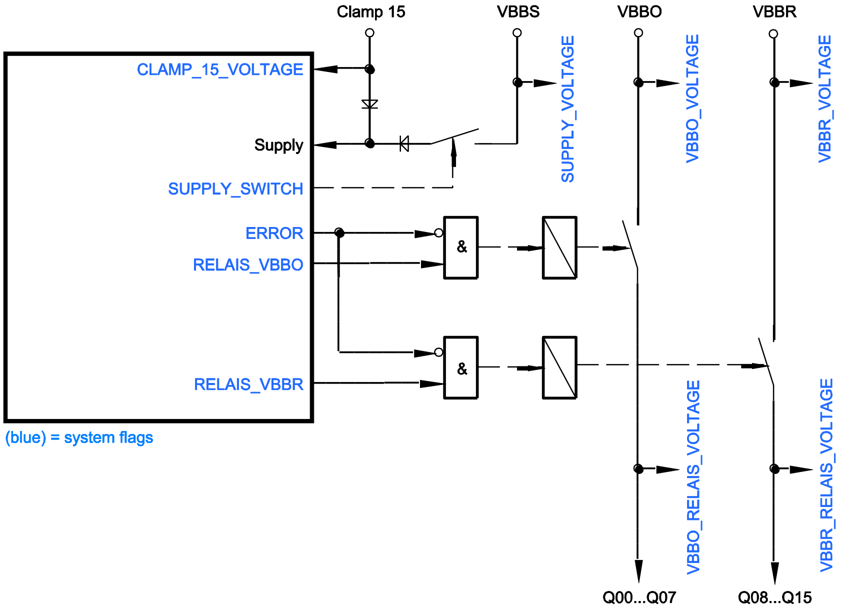


Figure: principle block diagram of supply and relays (standard side)

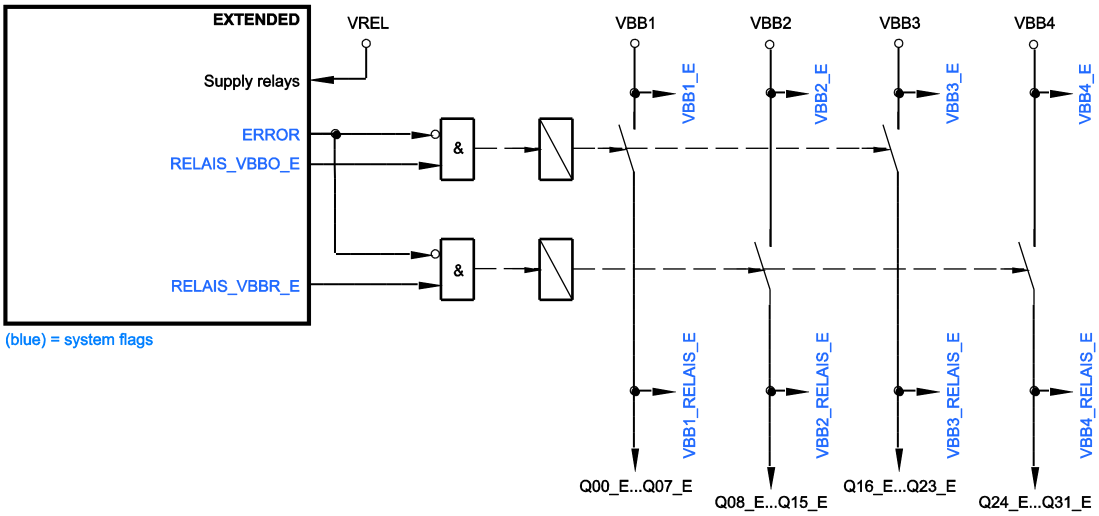


Figure: principle block diagram of supply and relays (extended side)

# Available memory

28798

## FLASH-Speicher

8136

| | |
|---|---|
| FLASH memory (non-volatile, slow memory)<br>overall existing in the device | 2 176 kByte |

Thereof the following memory areas are reserved for ...

| | |
|---|---|
| maximum size of the application program | 1 280 kByte |
| data other than the application program<br>user can write data such as files, bitmaps, fonts | 128 kByte |
| data other than the application program<br>read data with **FLASHREAD** (→ p. 196) or write data with **FLASHWRITE** (→ p. 197)<br>(files: 128 bytes less for header) | 64 kByte |

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

## SRAM

8360

| | |
|---|---|
| SRAM (volatile, fast memory)<br>overall existing in the device<br>SRAM indicates here all kinds of volatile and fast memories. | 2 216 kByte |

Thereof the following memory areas are reserved for ...

| | |
|---|---|
| data reserved by the application program | 192 kByte |

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

## FRAM

19547

| | |
|---|---|
| FRAM (non-volatile, fast memory)<br>overall existing in the device<br>FRAM indicates here all kinds of non-volatile and fast memories. | 128 kByte |

Thereof the following memory areas are reserved for ...

| | |
|---|---|
| variables in the application program, declared as VAR_RETAIN | 4 kByte |
| as remanent defined flags (from %MB0...)<br>▶ Set the end of the memory area by FB **MEMORY_RETAIN_PARAM** (→ p. 194)! | 4 kByte |

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

## 3.2.2 Operating principle of the delayed switch-off

28591

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

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

The ClassicControllers can be switched off via the program by means of a corresponding connection of the supply voltage inputs and the evaluation of the related system flags. The block diagram in the chapter **Hardware structure** (→ p. 14) shows the context of the individual current paths.

### Connect terminal VBB15 to the ignition switch

2418

The internal PLC electronics is initialised via the terminal VBB15 if at terminal VBBs supply voltage is applied.

These terminals VBB15 and VBBs are monitored internally. The applied terminal voltage VBB15 can be monitored via the system flag CLAMP_15_VOLTAGE. The applied terminal voltage VBBs can be monitored via the system flag SUPPLY_VOLTAGE.

### Latching

2419

Power-on of the controller:

- voltage is applied to VBB15 (clamp 15*) by means of the ignition switch.
- The system flag CLAMP_15_VOLTAGE recognises the voltage that has been applied and activates the system flag SUPPLY_SWITCH.
- SUPPLY_SWITCH activates the connection to the potential VBBs.

> The ignition switch is bypassed. Latching of the control voltage is established.

Power-off of the controller via clamp 15:

- The system flag CLAMP_15_VOLTAGE recognises the switching off of the supply voltage on terminal VBB15.

► Reset the system flag SUPPLY_SWITCH in the application program.

> Latching via VBBs is removed and the controller switches off completely.

*) In vehicles clamp 15 is the plus cable switched by the ignition lock.

### 3.2.3 Relays: important notes!

12976

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

**NOTICE**

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

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

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

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

## 3.2.4 Monitoring concept

**Content**

Monitoring of the supply voltages VBBx ....................19
Operating principle of the monitoring concept ....................21
Reference voltage output ....................22

28520

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

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

or:

- the runtime system deactivates the controller
  > the program stops
  > the outputs change to logic "0"
  > the status LED goes out

### Monitoring of the supply voltages VBBx

6752

In case of a fault we differentiate 2 scenarios:

#### Terminal voltage VBBx falls below the limit value of 5.25 V

15752

> The controller detects undervoltage. The outputs supplied by the terminal voltage VBBx are deactivated.

> If the terminal voltage recovers and returns to the normal range (> 10 V), the outputs are reactivated.

32395

⚠ **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.

### Terminal voltage VBBs falls below the limit value of 10 V

20638

> The controller continues to operate until the voltage has dropped so far that the internal voltages created from it also drop.

! Below 10 V no retain data is saved. → flag RETAIN_WARNING

> In case of a drop of the internal voltages the controller goes into reset.
> Execution of the runtime and application programs is interrupted.
> This happens irrespective of the current program step of the PLC.

> A restart of the controller is not carried out before the supply voltages are above the limit value again.

## Operating principle of the monitoring concept

2421

⚠ **WARNING**

Danger due to unintentional switch-off of all outputs!

If monitoring routines detect a system error:
> the device deactivates the energy for all outputs.

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

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

! Manual setting of a flag bit ERROR_VBB... has NO effects on the relays!

⚠ **WARNING**

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

► When creating the program, the programmer must ensure that no unintentional and dangerous start of machines or plant sections after a fault (e.g. e-stop) and the following fault elimination can occur!
⇨ Realise restart inhibit.

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

! If a watchdog error occurs, ...
> the program processing is interrupted automatically
> the outputs become currentless and go to logical "0"
> the controller is reset
> the controller then starts again as after power on.

## Reference voltage output

2250
13934

The reference voltage output is used to supply sensors with a stable voltage which is not subjected to the fluctuations of the supply voltage.

13402

**NOTICE**

Reference voltage output can get damaged!

► Do NOT apply any external voltage!

Via the binary system variables REFERENCE_VOLTAGE_5 or REFERENCE_VOLTAGE_10 the voltage is set on the reference voltage output [$V_{REF}$ OUT]:

| REFERENCE_VOLTAGE_10 | REFERENCE_VOLTAGE_5 | Reference voltage<br>[$V_{REF}$ OUT] |
|---|---|---|
| FALSE | FALSE | 0 V |
| FALSE | TRUE | 5 V |
| TRUE | FALSE | 10 V |
| TRUE | TRUE | 0 V |

► If reference voltage = 10 V selected:
supply the controller with min. 13 V!

► Voltage monitoring on the reference voltage output with system variable REF_VOLTAGE.

> If system variable ERROR = TRUE:
the reference voltage output is deactivated (output = 0 V).

## 3.2.5 Inputs (technology)

**Content**

Analogue inputs....23
Binary inputs....24
Input group I00...I15....25
Input group I00_E...I15_E....26

28353

### Analogue inputs

28803

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

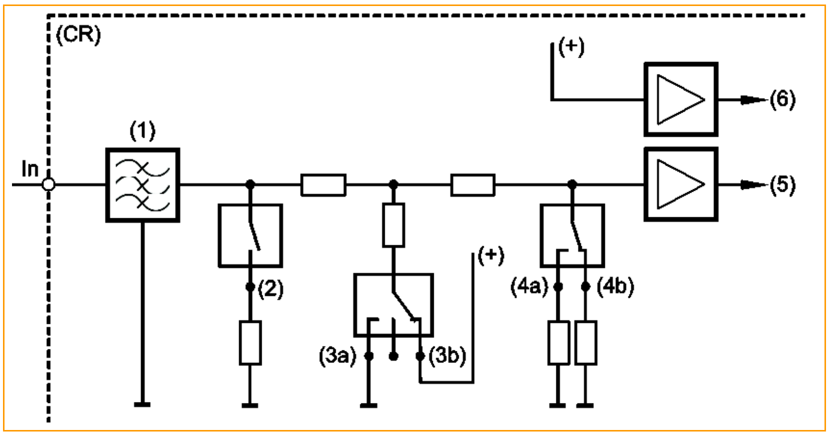
- current input 0...20 mA
- voltage input 0...10 V
- voltage input 0...32 V

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.

28803



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

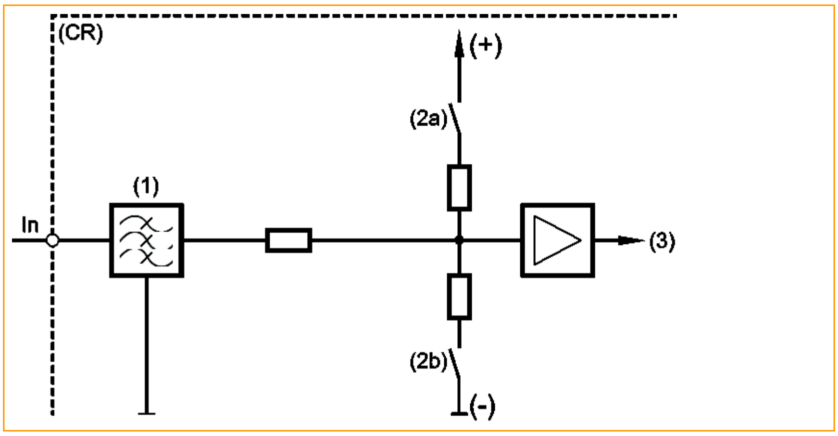
## Binary inputs

1015
28811

The binary input can be operated in following modes:

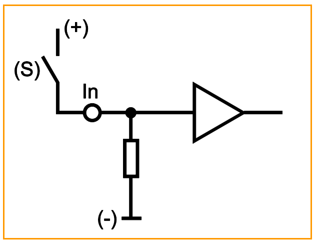
- binary input plus switching (BL) for positive sensor signal
- binary input minus switching (BH) for negative sensor signal

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

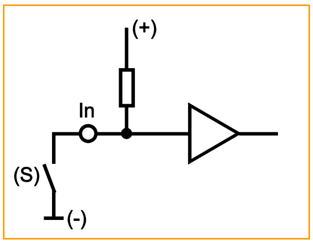


In = pin binary-input n
(CR) = device
(1) = input filter
(2a) = input minus switching
(2b) = input plus switching
(3) = voltage

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



In = pin binary input n
(S) = sensor



In = pin binary input n
(S) = sensor

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

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 I00...I15

20390

These inputs are a group of multifunction channels.

These inputs can be used as follows (each input separately configurable):
- analogue input 0...20 mA
- analogue input 0...10 V
- analogue input 0...32 V
- voltage measurement ratiometric 0...1000 ‰
- binary input minus switching (BH) for negative sensor signal
- binary input plus switching (BL) for positive sensor signal
- fast input for e.g. incremental encoders and frequency or interval measurement

→ chapter **Possible operating modes inputs/outputs** (→ p. 233)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

All inputs show the same behaviour concerning function and diagnosis.
Detailed description → chapter **Address assignment inputs / outputs**

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

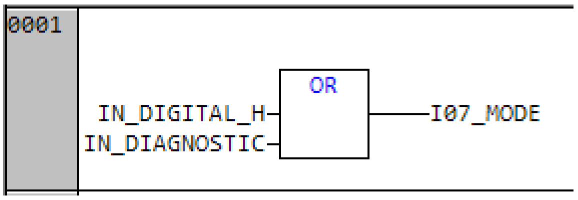
If the analogue inputs are configured for current measurement, the device switches to the safe voltage measurement range (0...32V DC) and the corresponding error bit in the flag byte ERROR_CURRENT_Ix is set when the final value (> 21.7 mA) is exceeded.
The device checks once a second if the current value is again below the limit value. When the value is again below the limit value, the input automatically switches back to the current measurement range.

► Configuration of each input is made via the application program:
- FB **INPUT_ANALOG** (→ p. 130) > input MODE
- Configuration byte Ixx_MODE
- Fast inputs with the following FBs:

| | |
|---|---|
| **FAST_COUNT** (→ p. 138) | Counter block for fast input pulses |
| **FREQUENCY** (→ p. 140) | Measures the frequency of the signal arriving at the selected channel |
| **FREQUENCY_PERIOD** (→ p. 142) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **INC_ENCODER** (→ p. 144) | Up/down counter function for the evaluation of encoders |
| **PERIOD** (→ p. 146) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **PHASE** (→ p. 150) | Reads a pair of channels with fast inputs and compares the phase position of the signals |

33866

**Example with configuration byte Ixx_MODE:**
The assignment sets the selected input to the operating mode IN_DIGITAL_H with diagnosis:



33866

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

| System flags (symbol name) | Type | Description |
|---|---|---|
| ERROR_BREAK_Ix<br>(0...x, value depends on the device,<br>→ data sheet) | DWORD | input double word x: wire break error<br>or (resistance input): short to supply<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_SHORT_Ix<br>(0...x, value depends on the device,<br>→ data sheet) | DWORD | input double word x: short circuit error<br>only if input mode = IN_DIGITAL_H<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |

## Input group I00_E...I15_E

6887

In principle, the same statements as for the input group I00...I15 apply.
Deviations:

- The symbolic addresses of the inputs are Inn_*E*.
- The symbolic addresses of the configuration variables are Inn_MODE_*E*.
- The symbolic addresses of the filters are Inn_FILTER_*E*
- The symbolic addresses of the digital filters are Inn_DFILTER_*E*
- The symbolic addresses of the other flags also end with '_*E*'.

## 3.2.6 Outputs (technology)

**Content**

Binary outputs....27
PWM outputs....27
Output group Q0 (Q00...15)....28
Output group Q00_E...Q15_E....30
Output group Q16_E...Q31_E....31

28572

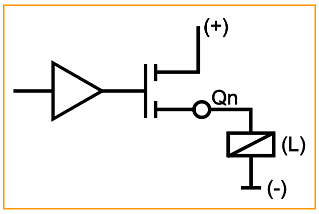
### Binary outputs

28815

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

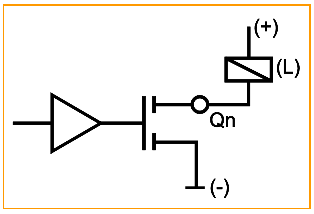
- binary output, plus switching (BH) with/without diagnostic function
- binary output minus switched (BL) without diagnostic function

28815



Qn = pin output n
(L) = load

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



Qn = pin output n
(L) = load

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

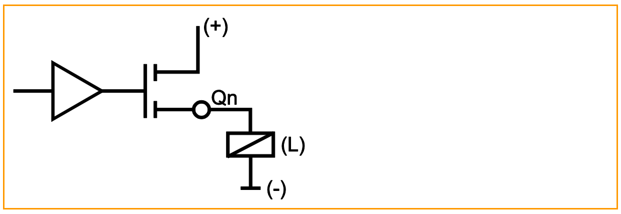
### PWM outputs

28530

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

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

28530



Qn = pin output n
(L) = load

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

## Output group Q0 (Q00...15)

2244

These outputs are a group of multifunction channels.

These outputs provide several function options (each output separately configurable):
- binary output, plus switching (BH), partly also minus switching (BL)
- analogue current-controlled output (PWMi)
- analogue output with pulse-width modulation (partly as H-bridge)

→ chapter **Possible operating modes inputs/outputs** (→ p. 233)

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

► Configuration of each output is made via the application program:
indicate the load currents → FB **OUTPUT_CURRENT** (→ p. 156)
PWM output: → FB **PWM1000** (→ p. 160)
control H-bridge → FB **OUTPUT_BRIDGE** (→ p. 153)

► Configure the current measuring range for outputs Q00…Q03 and Q08…Q11 (either 2 A or 4 A):
→ function block SET_OUTPUT_MODE > input CURRENT_RANGE

When using the H-bridge current control is not supported.

In case of a fault (e.g. short circuit) the outputs are switched off in 2 groups via the relay contacts.

33964

⚠ **WARNING**

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

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

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

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

When used as digital output, configuration is carried out for each output using the system variables Qxx_MODE. If the diagnosis is to be used, it must be activated in addition.

Wire break and short circuit of the output signal are (combined per output group) indicated separately via the system variables ERROR_BREAK_Qx or ERROR_SHORT_Qx. The individual output error bits can be masked in the application program, if necessary.

ⓘ **NOTE**

To protect the internal measuring resistors, OUT_OVERLOAD_PROTECTION should always be active (default setting). Depending on the selected current measuring range protection is given from 2.25 A or 4.5 A. The FB is **not** supported in the PWM mode and can be deactivated, if necessary.

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

Depending on the operating temperature, a short circuit might no longer reliably be detected when the short circuit current has reached a certain value since the output drivers are automatically and temporarily deactivated for protection against self-destruction.

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

## Diagnosis: binary outputs (via current measurement)

19398
28857

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

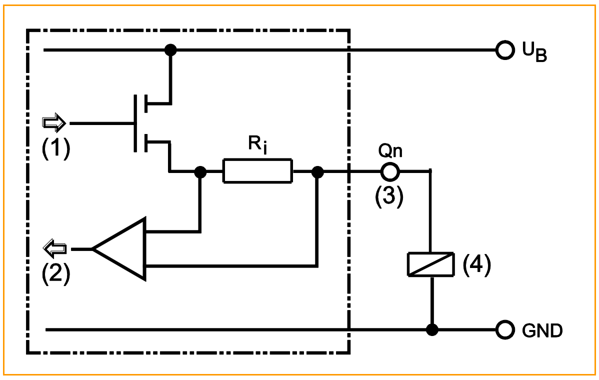


Figure: principle block diagram

(1) Output channel

(2) Read back channel for diagnostics

(3) Pin output n

(4) Load

### Diagnosis: overload (via current measurement)

19437
28851

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)

28854

Wire-break detection is done via the read back channel inside the output.

| | |
|---|---|
| Prerequisite for diagnosis: | output = TRUE |
| Diagnosis = wire break: | no current flows on the resistor Ri (no voltage drops).<br>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)

28850

Wire-break detection is done via the read back channel inside the output.

| | |
|---|---|
| Prerequisite for diagnosis: | output = TRUE |
| Diagnosis = short circuit against GND: | the supply voltage drops over the series resistor Ri |

## Output group Q00_E...Q15_E

6884

In principle, the same statements as for the first output group apply.
Deviations:

- The symbolic addresses of the outputs are Qnn_*E*.
- The symbolic addresses of the configuration variables are Qnn_MODE_*E*.
- The symbolic addresses of the other flags also end with '_*E*'.

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

### Diagnosis: binary outputs (via current measurement)

19398
28857

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

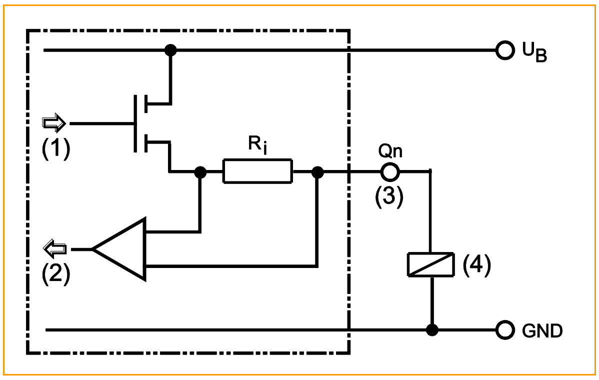


Figure: principle block diagram

(1) Output channel
(2) Read back channel for diagnostics
(3) Pin output n
(4) Load

### Diagnosis: overload (via current measurement)

19437
28851

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)

28854

Wire-break detection is done via the read back channel inside the output.

| | |
|---|---|
| Prerequisite for diagnosis: | output = TRUE |
| Diagnosis = wire break: | no current flows on the resistor Ri (no voltage drops).<br>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)

28850

Wire-break detection is done via the read back channel inside the output.

| | |
|---|---|
| Prerequisite for diagnosis: | output = TRUE |
| Diagnosis = short circuit against GND: | the supply voltage drops over the series resistor Ri |

## Output group Q16_E...Q31_E

10955

In principle, the same statements as for the first output group apply.
Deviations:

- The symbolic addresses of the outputs are Qnn_*E*.
- The symbolic addresses of the configuration variables are Qnn_MODE_*E*.
- The symbolic addresses of the other flags also end with '_*E*'.
- The outputs are rated up to max. 2 A (fixed).
- The outputs have the fixed configuration binary plus switching.
- There is no system variable Qnn_FILTER_*E*.
- Diagnosis: binary outputs (via voltage measurement)

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

## 3.2.7 Note on wiring

28579

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<br>Binary high side output: plus switching for positive output signal |
| BL | Binary low side input: plus switching for positive sensor signal<br>Binary low side output: minus switching for negative output signal |
| CYL | Input period measurement |
| ENC | Input encoder signals |
| FRQ | Frequency input |
| H bridge | Output with H-bridge function |
| PWM | **P**ulse-**w**idth **m**odulated signal |
| PWMi | PWM output with current measurement |
| IH | Pulse/counter input, high side: minus switching for negative sensor signal |
| IL | Pulse/counter input, low side: plus switching for positive sensor signal |
| R | Read back channel for one output |

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

## 3.2.8 Safety instructions about Reed relays

28354

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

28354

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

► **Remedy:** Install a series resistor for the Reed relay:
Series resistor = max. input voltage / permissible current in the Reed relay
**Example:** 32 V / 500 mA = 64 Ohm

► The series resistor must not exceed 5 % of the input resistance RE of the device input (→ data sheet). Otherwise, the signal will not be detected as TRUE.
**Example:**
RE = 3 000 Ohm
⇒ max. series resistor = 150 Ohm

## 3.2.9 Feedback in case of externally supplied outputs

28835

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

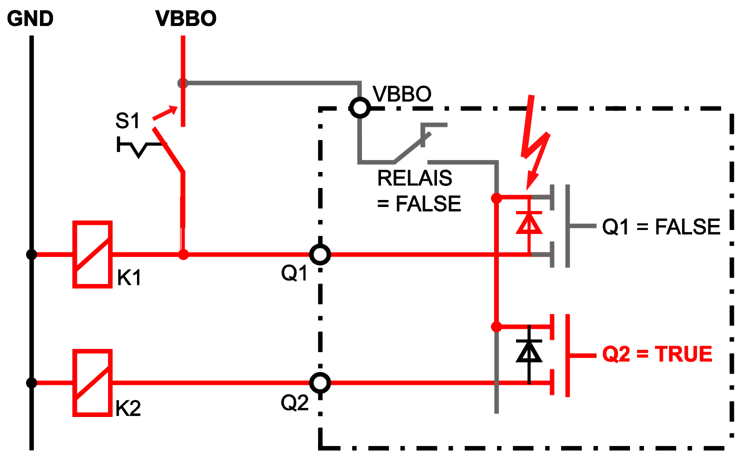
### NOTICE

Destruction of outputs if there is inadmissible feedback!

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

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

► Protect externally supplied outputs by means of blocking diodes!

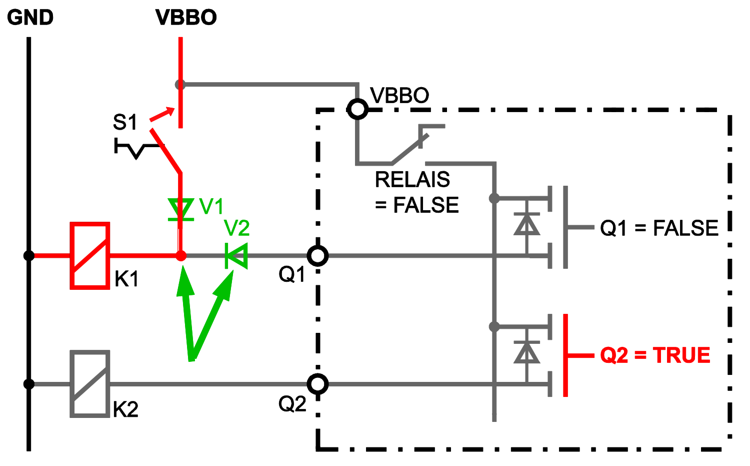


**Example:**

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

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

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



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

**Remedy:**

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

**Successful:**

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

## ⓘ NOTE

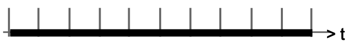
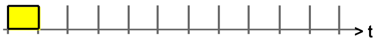
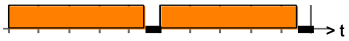
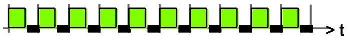
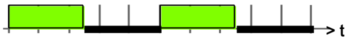
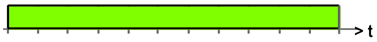
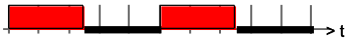
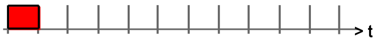
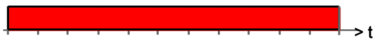
**Help for externally supplied outputs**

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

## 3.2.10 Status LED

20809

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

| LED colour | Display | Description |
|---|---|---|
| Off | Permanently off | No operating voltage |
| |  | |
| Yellow | Briefly on | Initialisation or reset checks |
| |  | (time frame = 200 ms) |
| Orange | Flashing with 0.2 Hz | TEST=FALSE: no runtime system loaded |
| |  | (time frame = 1 s) |
| Green | Flashing with 5 Hz | TEST=TRUE: no runtime system loaded |
| |  | (time frame = 200 ms) |
| Green | Flashing with 2 Hz | Application = RUN |
| |  | (time frame = 200 ms) |
| Green | Permanently on | Application = STOP |
| |  | |
| Red | Flashing with 2 Hz | Application = RUN with error |
| |  | (time frame = 200 ms) |
| Red | Briefly on | FATAL ERROR |
| |  | (time frame = 200 ms) |
| Red | Permanently on | TEST=TRUE: Application = STOP and FATAL ERROR<br>TEST=FALSE: ERROR STOP / SYSTEM STOP |
| |  | |

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

## Control the LED in the application program

13142

With this device the status LED can also be set by the application program. To do so, the following system variables are used (→ chapter **System flags** (→ p. 215)):

| System flags (symbol name) | Type | Description |
|---|---|---|
| LED | WORD | LED color for "LED switched on":<br>0x0000 = LED_GREEN (preset)<br>0x0001 = LED_BLUE<br>0x0002 = LED_RED<br>0x0003 = LED_WHITE<br>0x0004 = LED_BLACK<br>0x0005 = LED_MAGENTA<br>0x0006 = LED_CYAN<br>0x0007 = LED_YELLOW |
| LED_X | WORD | LED color for "LED switched off":<br>0x0000 = LED_GREEN<br>0x0001 = LED_BLUE<br>0x0002 = LED_RED<br>0x0003 = LED_WHITE<br>0x0004 = LED_BLACK (preset)<br>0x0005 = LED_MAGENTA<br>0x0006 = LED_CYAN<br>0x0007 = LED_YELLOW |
| LED_MODE | WORD | LED flashing frequency:<br>0x0000 = LED_2HZ (flashes at 2 Hz; preset)<br>0x0001 = LED_1HZ (flashes at 1 Hz)<br>0x0002 = LED_05HZ (flashes at 0.5 Hz)<br>0x0003 = LED_0HZ (lights permanently with value in LED) |

### ! 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

**Content**

Serial interface....36
USB interface ....36
CAN interfaces ....37

28337

### 3.3.1 Serial interface

28346

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

28346

**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 any of the CAN interfaces.

Connections and data → data sheet

### 3.3.2 USB interface

14100

This device features a USB interface for program download and debugging.

Connections and data → data sheet
Install the USB driver on the PC → installation instructions / operating instructions

Settings in CODESYS for [Online] > [Communication Parameters...] via USB:

| Device | Runtime system version | Parameter | Value |
|---|---|---|---|
| CR0032 | < V03.00.00 | Baud rate | 115200 |
| CR0032 | ≥ V03.00.01 | Baud rate | 4800...57600 |
| CR0033, CR0133 | ≤ V02.00.01 | Baud rate | 115200 |
| CR0033, CR0133 | ≥ V02.00.02 | Baud rate | 4800...57600 |
| CR0232, CR0233 | all | Baud rate | 115200 |
| CR0234, CR0235 | all | Baud rate | 4800...57600 |
| CR7n32 | ≤ V01.00.04 | Baud rate | 115200 |
| CR7n32 | ≥ V01.00.05 | Baud rate | 4800...57600 |
| CR0n3n, CR7n32 | all | Motorola byteorder | No |
| CR0n3n, CR7n32 | all | Flow Control | On |

## 3.3.3 CAN interfaces

**Content**

CAN: interfaces and protocols..................................................................37

28810

Connections and data → data sheet

### CAN: interfaces and protocols

13820
32633

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** (→ p. 75))
- CANopen master (→ chapter **Function elements: CANopen master** (→ p. 84))
- CANopen slave (→ chapter **Function elements: CANopen slave** (→ p. 94))
- CANopen network variables (via CODESYS)
- SAE J1939 (for drive management, → chapter **Function elements: SAE J1939** (→ p. 107))
- bus load detection
- error frame counter
- download interface
- 100 % bus load without package loss

11793

The following CAN interfaces and CAN protocols are available in this **ecomat*mobile*** 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 | CAN Layer 2 | CAN Layer 2 |
| | CANopen | CANopen | CANopen | CANopen |
| | SAE J1939 | SAE J1939 | SAE J1939 | SAE J1939 |

Standard baud rate = 125 Kbits/s

ℹ Which CANopen compatible interface works with which CANopen protocol is decided by the order in which you append the subelements in the PLC configuration:
CODESYS > [PLC Configuration] > [CR0232 Configuration Vxx] > [Append subelement] >
[CANopen master] or [CANopen slave]

## 3.4 Software description

**Content**

Software modules for the device ....................39
Programming notes for CODESYS projects....................41
Operating states ....................45
Operating modes....................49
Performance limits of the device ....................50

28396

### 3.4.1 Software modules for the device

**Content**

Bootloader ....................................................................................39
Runtime system ...............................................................................39
Application program .........................................................................40
Libraries ..........................................................................................40

28399

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,<br>MaintenanceTool |
| Runtime system *) | Upgrade yes<br>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** (→ p. 55)

Below we describe this software module:

## Bootloader

28807

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

28330

Basic program in the device, establishes the connection between the hardware of the device and the application program.
→ chapter **Software modules for the device** (→ p. 39)

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

## Application program

28795

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.

28795

⚠ **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

14117

**ifm electronic** offers several libraries (`*.LIB`) to match each device containing program modules for the application program. Examples:

| Library | Use |
|---|---|
| `ifm_CR0232_Vxxyyzz.LIB` | device-specific library<br>Must always be contained in the application program! |
| `ifm_CR0232_CANopenxMaster_Vxxyyzz.LIB`<br>x = 1...4 = number of the CAN interface | (optional)<br>if a CAN interface of the device is to be operated as a CANopen master |
| `ifm_CR0232_CANopenxSlave_Vxxyyzz.LIB`<br>x = 1...4 = number of the CAN interface | (optional)<br>if a CAN interface of the device is to be operated as a CANopen slave |
| `ifm_CR0232_J1939_Vxxyyzz.LIB` | (optional)<br>if a CAN interface of the device is to communicate with a Diesel engine |

Details: → chapter **ifm libraries for the device CR0232** (→ p. 69)

## 3.4.2 Programming notes for CODESYS projects

**Content**

FB, FUN, PRG in CODESYS ....................41
Calculations and conversions in the application program ....................42
Note the cycle time! ....................42
Creating application program ....................43
Save boot project ....................44
Using ifm downloader ....................44
Using ifm maintenance tool ....................44

28583

Here you receive tips how to program the device.

► See the notes in the CODESYS programming manual.

### FB, FUN, PRG in CODESYS

28833

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.

## Calculations and conversions in the application program

28189

### ⓘ NOTE

If the following elements are required in the application program:
- mathematical functions (e.g. ATAN),
- calculations,
- conversions (e.g. REAL_TO_BYTE),

then the following applies to the values at the inputs and outputs of the corresponding operators:

► Strictly observe the admissible value range in each individual case!

> Otherwise, this may cause an FPU error in the controller.

Examples:

28189

The value of the target format that can max. represented is exceeded.
Example:

REAL_TO_INT (12345678.3)

> INT is limited to -32768...+32767 (only integers)

28189

An existing real number is obviously in the value range of the target format.
In reality, however, the number is outside the target format (because of the internal representation of the real number).
Example:

DW := REAL_TO_DWORD (4294967295.0);

> The most accurate representation of 4294967295 in REAL is 4.294967296E9
> Therefore the value exceeds the max. permissible value of the target format by 1.
> DWORD is limited to 0...4294967295.

## Note the cycle time!

28578

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

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

### NOTICE

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

► When designing the application program the above-mentioned recommendations must be complied with and tested.

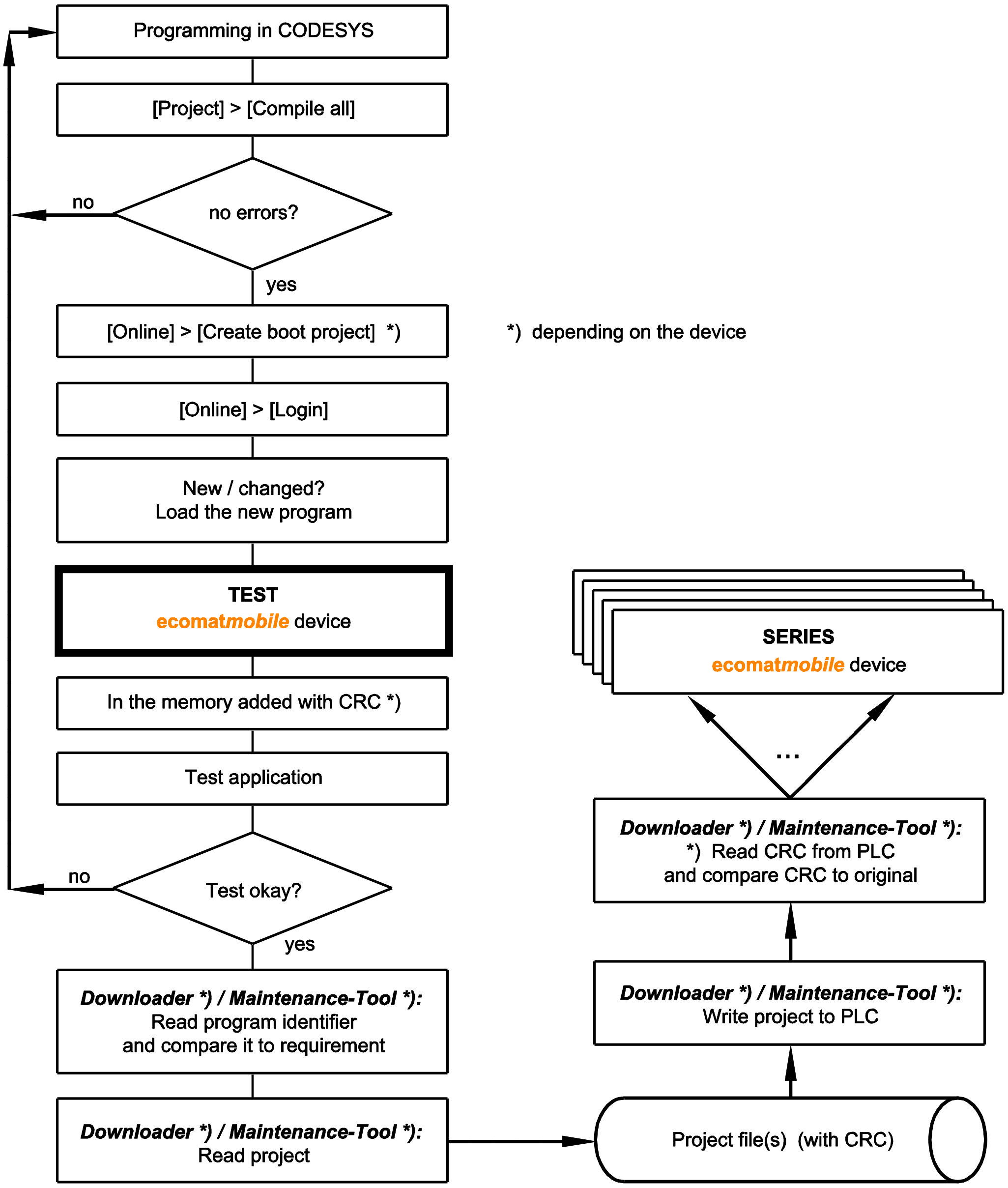
► If necessary, the cycle time must be optimised by restructuring the software and the system set-up.

## Creating application program

28845

The application program is generated by the CODESYS 2.3 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 2.3 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 errors?
no
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
Test okay?
no
yes
Downloader *) / Maintenance-Tool *):
Read program identifier
and compare it to requirement
Downloader *) / Maintenance-Tool *):
Read project
Project file(s) (with CRC)
Downloader *) / Maintenance-Tool *):
Write project to PLC
Downloader *) / Maintenance-Tool *):
*) Read CRC from PLC
and compare CRC to original
...
SERIES
ecomatmobile device

Graphics: Creation and distribution of the software

## Save boot project

28359

ⓘ 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

28403

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):
Homepage → www.ifm.com

## Using ifm maintenance tool

27717

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:
Homepage → www.ifm.com

## 3.4.3 Operating states

**Content**

Operating states: runtime system is not available ....45
Operating states: application program is not available ....46
Operating states: application program is available ....47
Bootloader state ....47
INIT state (Reset) ....48
STOP state ....48
RUN state ....48
SYSTEM STOP state ....48

28585

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

28558

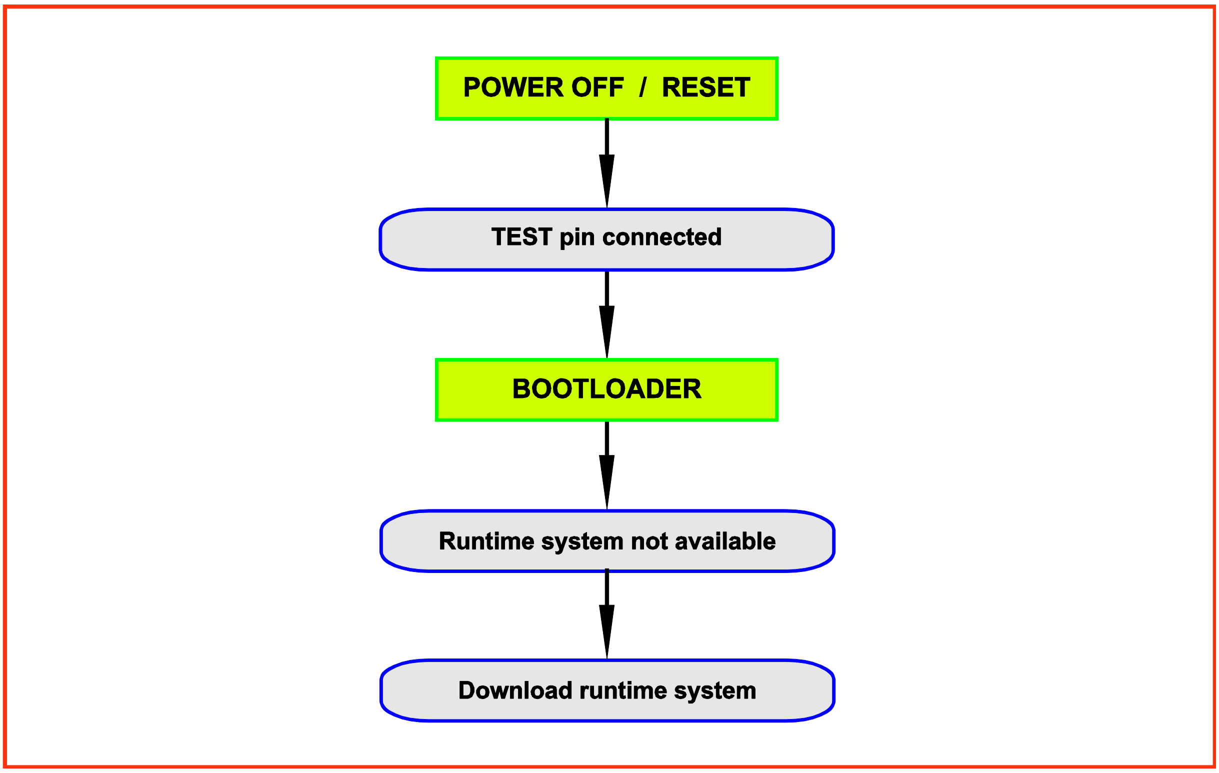


Figure: operating states (here: runtime system is not available)

## Operating states: application program is not available

28587

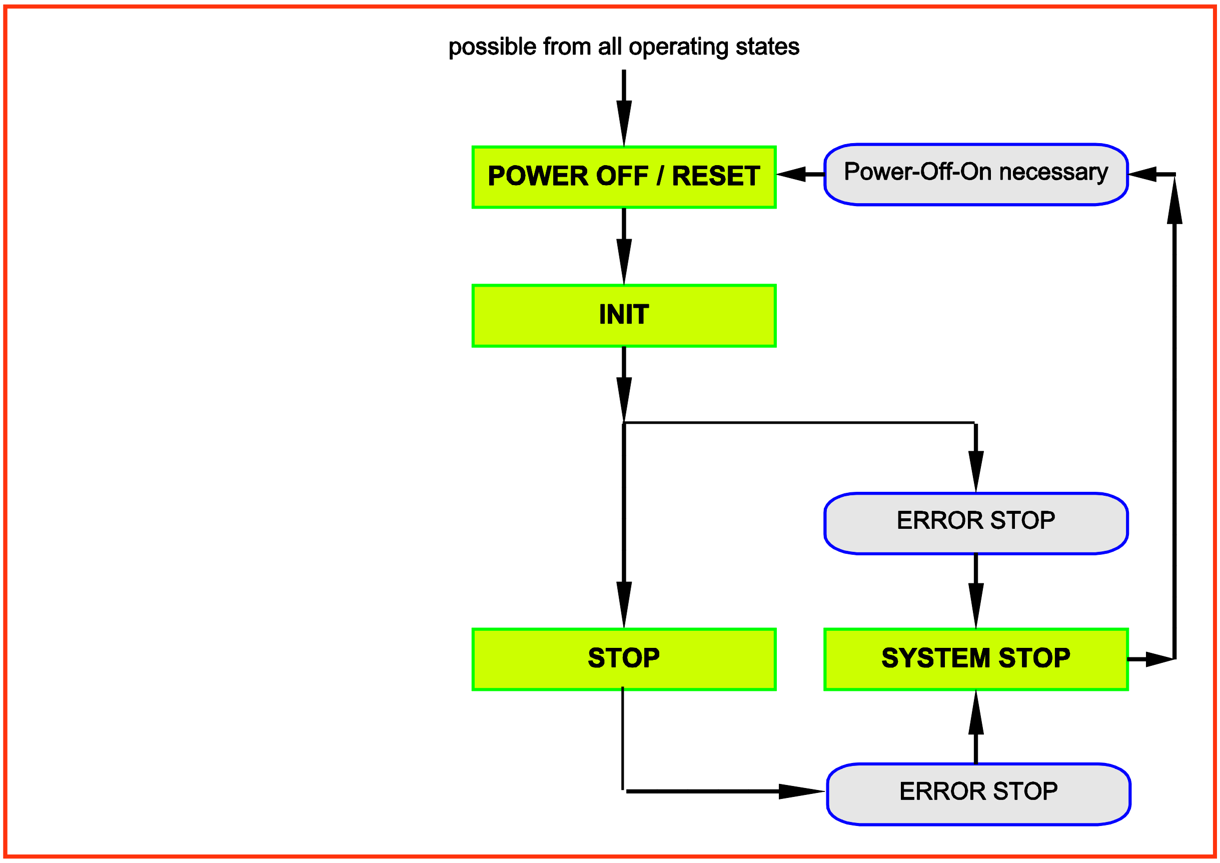


Figure: operating states (here: application program is not available)

## Operating states: application program is available

28586

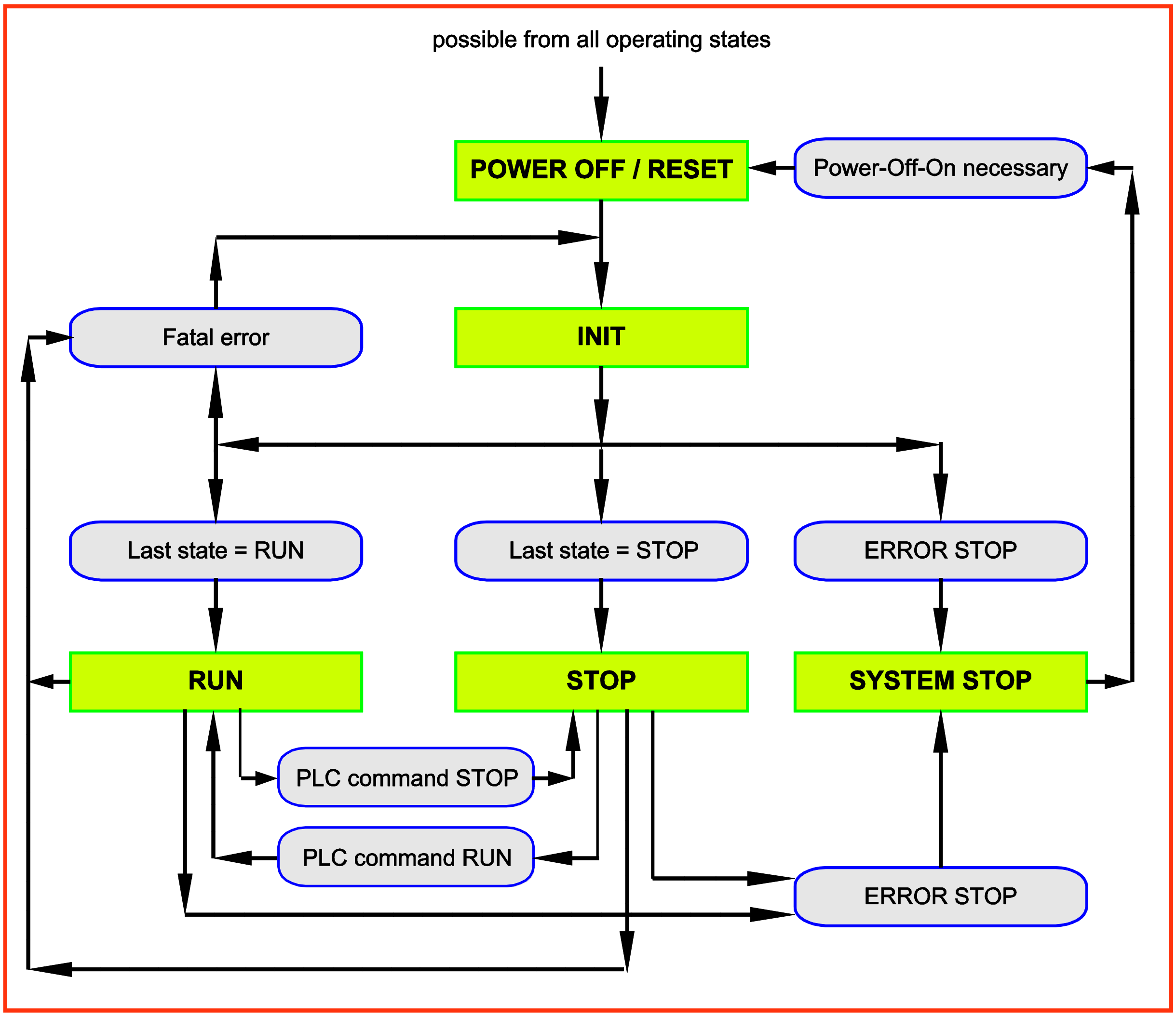


Figure: operating states (here: application program is available)

## Bootloader state

28806

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)

28313

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

28422

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** (→ p. 49)

> The LED lights green.

## RUN state

28335

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** (→ p. 49)

> The LED flashes green (2 Hz).

## SYSTEM STOP state

28385

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

28589

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

### TEST mode

28409

**NOTICE**

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

28409

**NOTE**

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

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

### Notes: TEST inputs

28581

► The TEST inputs of all the controllers in the machine should be wired individually and marked clearly so that they can be properly allocated to the controllers.

► During a service access only activate the TEST input of the controller to be accessed.

## SERIAL_MODE

2548

The serial interface is available for the exchange of data in the application. Debugging the application software is then only possible via all 4 CAN interfaces.

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** (→ p. 119)

## DEBUG mode

28844

If the input DEBUG of **SET_DEBUG** (→ p. 208) is set to TRUE, the programming system or the downloader, for example, can communicate with the controller and execute some special 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** (→ p. 49)) is not connected to supply voltage.

## 3.4.5 Performance limits of the device

28571

❗ Note the limits of the device! → Data sheet

## Watchdog behaviour

28352

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

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

This you can read in the flag LAST_RESET.

## CODESYS functions

2254

You should note the following limits:

- Up to 2 048 blocks (PB, FB...) are supported.
- Flags available for user → chapter **Available memory** (→ p. 16).
  Description of the retain flags → for the corresponding FBs.

# 4 Configurations

**Content**

Set up the runtime system ..... 52
Set up the programming system ..... 54
Function configuration in general ..... 57
Function configuration of the inputs and outputs ..... 58
Variables ..... 66

28785

The device configurations described in the corresponding installation instructions or in the **Appendix** (→ p. 214) 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

**Content**

Reinstall the runtime system ....................52
Update the runtime system....................53
Verify the installation ....................53

28320

### 4.1.1 Reinstall the runtime system

14092
28531

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**:
→ www.ifm.com

28531

**! NOTE**

The software versions suitable for the selected target must always be used:
- runtime system (`ifm_CR0232_Vxxyyzz.H86`),
- PLC configuration (`ifm_CR0232_Vxx.CFG`),
- device library (`ifm_CR0232_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. "`CR0232`") 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.

28531

! The following files must also be loaded:
- the internal libraries (created in IEC 1131) required for the project,
- the configuration files (`*.CFG`) and
- the target files (`*.TRG`).

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

The runtime system is transferred to the device using the separate program "**ifm** downloader".
The software can be downloaded from **ifm's** website, if necessary:
→ www.ifm.com

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

28404

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

28404

**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!

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

### 4.1.3 Verify the installation

14407
33893

► 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** (→ p. 206) | Reads the specific identifications stored in the device:<br>• hardware name and hardware version of the device<br>• name of the runtime system in the device<br>• version and revision no. of the runtime system in the device<br>• name of the application (has previously been saved by means of **SET_IDENTITY** (→ p. 209))<br>• serial number of the device |

► If the application detects an incorrect version of a runtime system:
bring all safety functions into the safe state.

## 4.2 Set up the programming system

**Content**

Set up the programming system manually ....................55
Set up the programming system via templates ....................57

28306

## 4.2.1 Set up the programming system manually

**Content**

Set up the target.....55
Activate the PLC configuration (e.g. CR0033) .....56

28307

### Set up the target

2687
28316

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_CR0232_Vxxyyzz.LIB` manually!

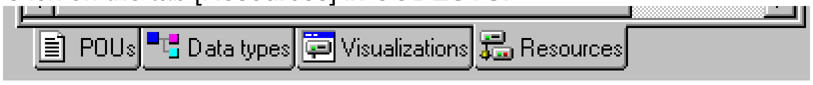
## Activate the PLC configuration (e.g. CR0033)

28916

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

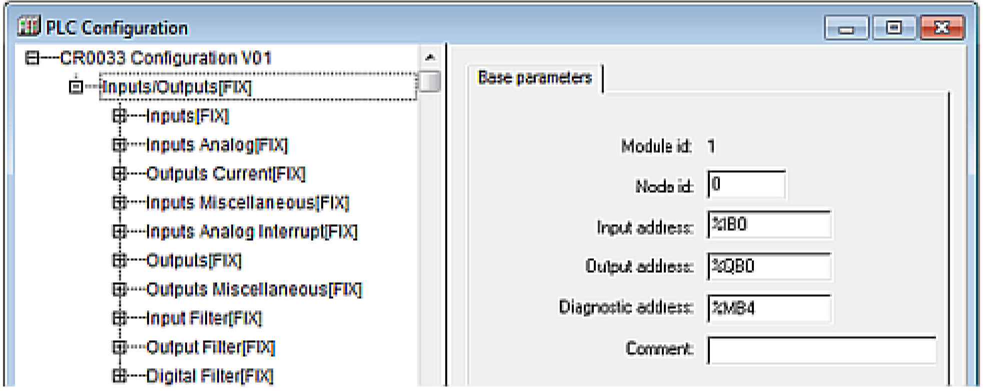
► The menu item [PLC Configuration] is reached via the tab [Resources].
Double-click on [PLC Configuration] to open the corresponding window.

► 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

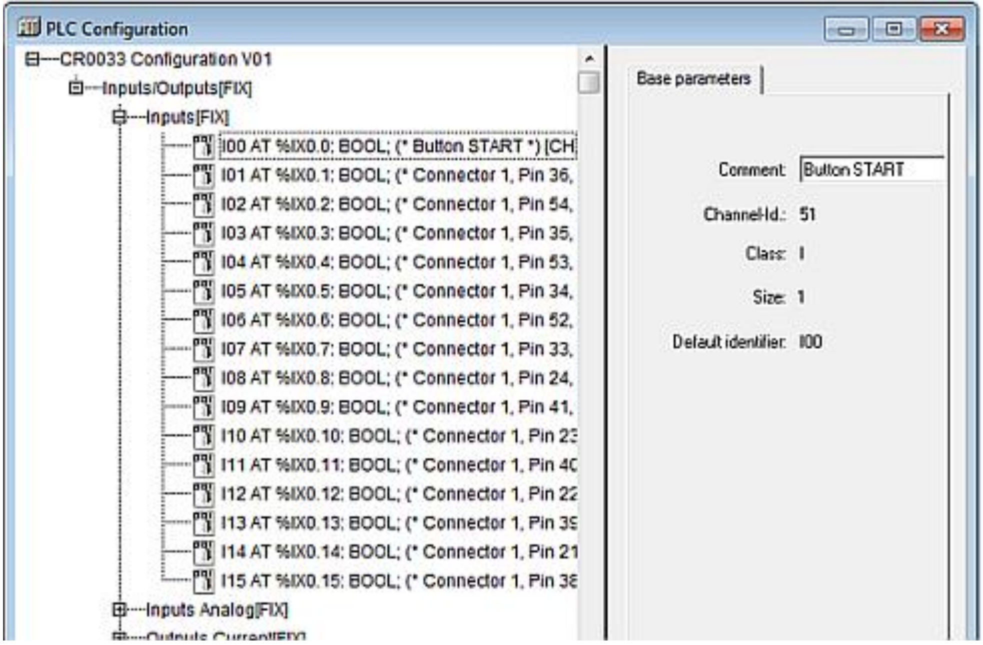
Output address: %QB0

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 *) [CH
            I01 AT %IX0.1: BOOL; (* Connector 1, Pin 36,
            I02 AT %IX0.2: BOOL; (* Connector 1, Pin 54,
            I03 AT %IX0.3: BOOL; (* Connector 1, Pin 35,
            I04 AT %IX0.4: BOOL; (* Connector 1, Pin 53,
            I05 AT %IX0.5: BOOL; (* Connector 1, Pin 34,
            I06 AT %IX0.6: BOOL; (* Connector 1, Pin 52,
            I07 AT %IX0.7: BOOL; (* Connector 1, Pin 33,
            I08 AT %IX0.8: BOOL; (* Connector 1, Pin 24,
            I09 AT %IX0.9: BOOL; (* Connector 1, Pin 41,
            I10 AT %IX0.10: BOOL; (* Connector 1, Pin 23
            I11 AT %IX0.11: BOOL; (* Connector 1, Pin 40
            I12 AT %IX0.12: BOOL; (* Connector 1, Pin 22
            I13 AT %IX0.13: BOOL; (* Connector 1, Pin 39
            I14 AT %IX0.14: BOOL; (* Connector 1, Pin 21
            I15 AT %IX0.15: BOOL; (* Connector 1, Pin 38
        ⊞---Inputs Analog[FIX]
        ⊞---Outputs Current[FIX]
```

Comment: Button START

ChannelId: 51

Class: I

Size: 1

Default identifier: I00

### 4.2.2 Set up the programming system via templates

28325

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

28325

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

► Open the requested template in CODESYS via:
[File] > [New from template…]

> CODESYS creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.

## 4.3 Function configuration in general

28350

### 4.3.1 Configuration of the inputs and outputs (default setting)

28792

- 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
28390

All system variables (→ chapter **System flags** (→ p. 215)) 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

**Content**

Configure inputs ....................................................................................................................59
Configure outputs ..................................................................................................................62

1812
32421

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_CR0232_Vxxyyzz.LIB`) for the configuration of the inputs and outputs.
  For details → **Possible operating modes inputs/outputs** (→ p. 233).

**Only ExtendedController:**
The designations of the inputs and outputs in the controller's second half are indicated by an appended _E.

### 4.4.1 Configure inputs

**Content**

Safety instructions about Reed relays....59
Fast inputs....60
Configure the software filters of the inputs....61
Configure the hardware filter....61

28784

Valid operating modes → chapter **Possible operating modes inputs/outputs** (→ p. 233)

#### Safety instructions about Reed relays

28354

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

28354

> ⓘ 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

## Fast inputs

2193

The devices dispose of fast counting/pulse inputs for an input frequency up to 30 kHz (→ data sheet).

The input resistance of the fast inputs switches automatically depending on the applied mode or function block:

| Input resistance | for mode / FB |
|---|---|
| 3.2 kohms | (standard) FAST_COUNT, FREQUENCY, INC_ENCODER, PERIOD and similar FBs |
| 50.7 kohms | input with fixed switching level 32 V |

23900

❗ The internal resistance $R_i$ of the signal source must be substantially lower than the input resistance $R_{input}$ of the used input (principle voltage alignment).

Otherwise the input signal of the fast input can be distort (low-pass characteristic).

14677

❗ If, for example, mechanical switches are connected to these inputs, there may be faulty signals in the controller due to contact bouncing.

▶ If necessary, filter these "false signals" using the filters Ixx_DFILTER.
(→ chapter **System flags** (→ p. 215)) (not available for all inputs)

Appropriate function blocks are e.g.:

| | |
|---|---|
| **FAST_COUNT** (→ p. 138) | Counter block for fast input pulses |
| **FREQUENCY** (→ p. 140) | Measures the frequency of the signal arriving at the selected channel |
| **FREQUENCY_PERIOD** (→ p. 142) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **INC_ENCODER** (→ p. 144) | Up/down counter function for the evaluation of encoders |
| **PERIOD** (→ p. 146) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **PERIOD_RATIO** (→ p. 148) | 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** (→ p. 150) | 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.

## Configure the software filters of the inputs

6883

A software filter that filters the measured input voltage on the analogue inputs can be configured via the system variables Ixx_FILTER. In case of a step response the filter behaves like a conventional low-pass filter, the limit frequency is set by the value entered in the system variable. Values of 0...8 are possible.

Table: limit frequency software low-pass filter on analogue input

| Ixx_FILTER | Filter frequency [Hz] | Signal rise time | Remarks |
|---|---|---|---|
| 0 | Filter deactivated | | |
| 1 | 390 | 1 ms | |
| 2 | 145 | 2.5 ms | |
| 3 | 68 | 5 ms | |
| 4 | 34 | 10 ms | Recommended, default setting |
| 5 | 17 | 21 ms | |
| 6 | 8 | 42 ms | |
| 7 | 4 | 84 ms | |
| 8 | 2 | 169 ms | |
| ≥ 9 | 34 | 10 ms | → Default setting |

12969

❗ After changing the filter setting, the value of this input or output is not output correctly at once. Only after the signal rise time (→ table) will the value be correct again.

ℹ The signal rise time is the time taken by a signal at the output of the filter to rise from 10 % to 90 % of the final value if an input step is applied. The signal fall time is the time taken by a signal to decrease from 90 % to 10 %.

## Configure the hardware filter

9154

A digital hardware filter can be configured on the fast counter and pulse inputs via the system variable Ixx_DFILTER. The value in µs (max. 100 000) indicates how long a binary level must be applied without interruption before it is adopted. Default = 0 µs.

❗ The level change of the input signal is delayed by the value set in the filter.

The filter has an effect on the detected signals only for the following function blocks:

| | |
|---|---|
| **FAST_COUNT** (→ p. 138) | Counter block for fast input pulses |
| **FREQUENCY** (→ p. 140) | Measures the frequency of the signal arriving at the selected channel |
| **FREQUENCY_PERIOD** (→ p. 142) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **INC_ENCODER** (→ p. 144) | Up/down counter function for the evaluation of encoders |
| **PERIOD** (→ p. 146) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **PERIOD_RATIO** (→ p. 148) | 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 [‰]. |

Digital filters are not available for all fast counter and pulse inputs.

## 4.4.2 Configure outputs

**Content**

Allowable configurations for Q00_MODE...Q15_MODE ....62
Allowable configurations for Q00_MODE_E...Q15_MODE_E ....62
Allowable configurations for Q16_MODE_E...Q31_MODE_E ....63
Configure the software filters of the outputs ....63
Binary and PWM outputs ....64

28788

Valid operating modes → chapter **Possible operating modes inputs/outputs** (→ p. 233)

### Allowable configurations for Q00_MODE...Q15_MODE

6903

| Overload | Diagnosis | -- | 4 A [1] | 2 A | -- | LS | HS | Config. value | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | [hex] | [dec] |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 09 | 9 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 11 | 17 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 49 | 73 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 51 | 81 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 89 | 137 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 91 | 145 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | C9 | 201 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | D1 | 209 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 | 2 |

🟩 = this configuration value is default

[1] only possible for outputs Q00...Q03 + Q08...Q11

### Allowable configurations for Q00_MODE_E...Q15_MODE_E

6904

| Overload | Diagnosis | -- | 4 A [1] | 2 A | -- | LS | HS | Config. value | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | [hex] | [dec] |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 09 | 9 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 11 | 17 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 49 | 73 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 51 | 81 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 89 | 137 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 91 | 145 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | C9 | 201 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | D1 | 209 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 02 | 2 |

🟩 = this configuration value is default

[1] only possible for outputs Q00_E...Q03_E + Q08_E...Q11_E

## Allowable configurations for Q16_MODE_E...Q31_MODE_E

17190

| Overload | Diagnosis | -- | 4 A [1] | 2 A | -- | LS | HS | Config. value | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | [hex] | [dec] |
| 0 | 0 | 0 | 0 | X (0) | 0 | 0 | 1 | 01 | 1 |
| 0 | 1 | 0 | 0 | X (0) | 0 | 0 | 1 | 41 | 65 |
| [1] | 0 | 0 | 0 | X (0) | 0 | 0 | 1 | 81 | 129 |

🟩 = this configuration value is default

[1] here not possible

## Configure the software filters of the outputs

6882

Via the system variables Qxx_FILTER a software filter which filters the measured current values can be configured.

- In case of a step response the filter behaves like a conventional low-pass filter, the limit frequency is set by the value entered in the system variable.
- During current measuring the filter setting affects the diagnosis time.

Table: Limit frequency software low-pass filter for the current measurement on the output

| Qxx_FILTER | Filter frequency [Hz] | Signal rise time | Remarks |
|---|---|---|---|
| 0 | Filter deactivated | | |
| 1 | 580 | 0.6 ms | |
| 2 | 220 | 1.6 ms | |
| 3 | 102 | 3.5 ms | |
| 4 | 51 | 7 ms | Recommended, default setting |
| 5 | 25 | 14 ms | |
| 6 | 12 | 28 ms | |
| 7 | 6 | 56 ms | |
| 8 | 3 | 112 ms | |
| ≥ 9 | 51 | 7 ms | → Default setting |

12969

❗ After changing the filter setting, the value of this input or output is not output correctly at once. Only after the signal rise time (→ table) will the value be correct again.

ℹ The signal rise time is the time taken by a signal at the output of the filter to rise from 10 % to 90 % of the final value if an input step is applied. The signal fall time is the time taken by a signal to decrease from 90 % to 10 %.

## Binary and PWM outputs

2423

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

- binary output, plus switching (BH) with/without diagnostic function
- binary output, plus switching (BH), partly also minus switching (BL)
- PWM output, plus switching (BH) without diagnostic function
- PWM output pair H-bridge without diagnostic function

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

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

❗ The medium current across a PWM signal can only be correctly determined via the FB OUTPUT_CURRENT if the current flowing in the switched-on state is within the measuring range.

33974

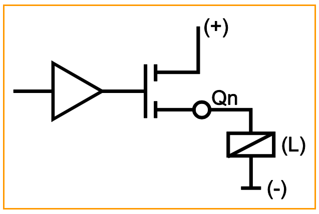
### ⚠ WARNING

Property damage or bodily injury possible due to malfunctions!

The following applies for outputs in PWM mode:

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

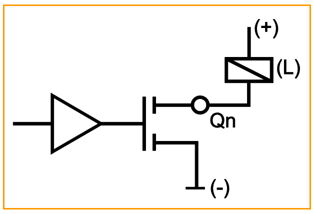
33974



Qn = pin output n

(L) = load

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



Qn = pin output n

(L) = load

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

33974

### ⚠ 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.

14931

### ❗ NOTE

► Do NOT reconfigure the outputs during operation!
It is not allowed to change from PWM output to binary output.

&gt; Otherwise the outputs may react unpredictably.

## Availability of PWM

12058

| Device | Number of available PWM outputs | of which current-controlled (PWMi) | PWM frequency [Hz] |
|---|---|---|---|
| CRn032, CR0033 | 16 | 16 | 20...250 |
| CRn232, CR0233 | 32 | 32 | 20...250 |

## Current control with PWM (= PWMi)

13829

Current measurement of the coil current can be carried out via the current measurement channels integrated in the controller. This allows for example that the current can be re-adjusted if the coil heats up. Thus the hydraulic relationships in the system remain the same.

In principle, the current-controlled outputs are protected against short circuit.

# 4.5 Variables

**Content**

Retain variables ........................................ 66
Network variables ........................................ 67

28318

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

28318

The device supports the following types of variables:

| **Variable** | **Declaration place** | Validity area | Memory behaviour |
|---|---|---|---|
| local | in the declaration part of the function element (POU) | Only valid in the function element (POU) where it was configured. | volatile |
| local retain | | | nonvolatile |
| global | In [Resources] > [Global Variables] > [Globale_Variables]: | Valid in all function elements of this CODESYS project. | volatile |
| global retain | | | nonvolatile |
| Network | In [Resources] > [Global Variables] > declaration list | Values are available to all CODESYS projects in the whole network if the variable is contained in its declaration lists. | volatile |
| Network retain | | | nonvolatile |

→ CODESYS programming manual

## 4.5.1 Retain variables

15454

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.

The contents of the retain variables are lost if the device is in the STOP state during power-off!

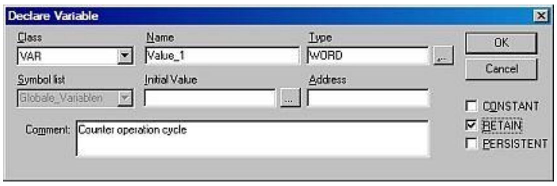
32425

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).



Declare Variable

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

28528

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

**Content**

ifm libraries for the device CR0232 ..........69
ifm function elements for the device CR0232..........74

28394

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 CR0232

**Content**

Library ifm_CR0232_V010003.LIB.....69
Library ifm_CR0232_CANopenxMaster_Vxxyyzz.LIB.....71
Library ifm_CR0232_CANopenxSlave_Vxxyyzz.LIB.....72
Library ifm_CR0232_J1939_Vxxyyzz.LIB.....72
Library ifm_hydraulic_32bit_Vxxyyzz.LIB.....73

28323

### 5.1.1 Library ifm_CR0232_V010003.LIB

18429

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

| Function element | Short description |
|---|---|
| **CANx** (→ p. 76) | Initialises CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_BAUDRATE** (→ p. 77) | Sets the transmission rate for the bus participant on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_BUSLOAD** (→ p. 78) | Determines the current bus load on CAN interface x and counts the occurred error frames<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_DOWNLOADID** (→ p. 79) | Sets the download identifier for CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_ERRORHANDLER** (→ p. 80) | Executes a "manual" bus recovery on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_RECEIVE** (→ p. 81) | CAN interface x: Configures a data receive object and reads out the receive buffer of the data object<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_SDO_READ** (→ p. 103) | CAN interface x: Reads the SDO with the indicated indices from the node<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_SDO_WRITE** (→ p. 105) | CAN interface x: writes the SDO with the indicated indices to the node<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_TRANSMIT** (→ p. 83) | Transfers a CAN data object (message) to the CAN interface x for transmission at each call<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CHECK_DATA** (→ p. 204) | Generates a checksum (CRC) for a configurable memory area and checks the data of the memory area for undesired changes |
| **DELAY** (→ p. 178) | Delays the output of the input value by the time T (dead-time element) |
| **FAST_COUNT** (→ p. 138) | Counter block for fast input pulses |
| FAST_COUNT_E | = **FAST_COUNT** (→ p. 138) for the extended side |
| **FLASHREAD** (→ p. 196) | transfers different data types directly from the flash memory to the RAM |
| **FLASHWRITE** (→ p. 197) | writes different data types directly into the flash memory |
| **FRAMREAD** (→ p. 199) | transfers different data types directly from the FRAM memory to the RAM<br>FRAM indicates here all kinds of non-volatile and fast memories. |
| **FRAMWRITE** (→ p. 200) | writes different data types directly into the FRAM memory<br>FRAM indicates here all kinds of non-volatile and fast memories. |
| **FREQUENCY** (→ p. 140) | Measures the frequency of the signal arriving at the selected channel |
| FREQUENCY_E | = **FREQUENCY** (→ p. 140) for the extended side |
| **FREQUENCY_PERIOD** (→ p. 142) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| FREQUENCY_PERIOD_E | = **FREQUENCY_PERIOD** (→ p. 142) for the extended side |

| Function element | Short description |
|---|---|
| **GET_IDENTITY** (→ p. 206) | Reads the specific identifications stored in the device:<br>• hardware name and hardware version of the device<br>• name of the runtime system in the device<br>• version and revision no. of the runtime system in the device<br>• name of the application (has previously been saved by means of **SET_IDENTITY** (→ p. 209))<br>• serial number of the device |
| **GET_IDENTITY_EIOS** (→ p. 207) | FB reads the specific identifications stored in the device for the extended side:<br>• name of the extended IO system (EIOS) in the device<br>• version and revision no. of the extended IO system (EIOS) in the device |
| **INC_ENCODER** (→ p. 144) | Up/down counter function for the evaluation of encoders |
| INC_ENCODER_E | = **INC_ENCODER** (→ p. 144) for the extended side |
| **INPUT_ANALOG** (→ p. 130) | analogue input channel: alternatively measurement of ...<br>• current<br>• voltage |
| INPUT_ANALOG_E | = **INPUT_ANALOG** (→ p. 130) for the extended side |
| **MEMCPY** (→ p. 201) | Writes and reads different data types directly in the memory |
| **MEMORY_RETAIN_PARAM** (→ p. 194) | Determines the remanent data behaviour for various events |
| **MEMSET** (→ p. 202) | Writes in a specified data area |
| **NORM** (→ p. 133) | Normalises a value [WORD] within defined limits to a value with new limits |
| **NORM_DINT** (→ p. 135) | Normalises a value [DINT] within defined limits to a value with new limits |
| **NORM_REAL** (→ p. 136) | Normalises a value [REAL] within defined limits to a value with new limits |
| **OUTPUT_BRIDGE** (→ p. 153) | H-bridge on a PWM channel pair |
| OUTPUT_BRIDGE_E | = **OUTPUT_BRIDGE** (→ p. 153) for the extended side |
| **OUTPUT_CURRENT** (→ p. 156) | Measures the current (average via dither period) on an output channel |
| OUTPUT_CURRENT_E | = **OUTPUT_CURRENT** (→ p. 156) for the extended side |
| **OUTPUT_CURRENT_CONTROL** (→ p. 157) | Current controller for a PWMi output channel |
| OUTPUT_CURRENT_CONTROL_E | = **OUTPUT_CURRENT_CONTROL** (→ p. 157) for the extended side |
| **PERIOD** (→ p. 146) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| PERIOD_E | = **PERIOD** (→ p. 146) for the extended side |
| **PERIOD_RATIO** (→ p. 148) | Measures the frequency and the cycle period (cycle time) in [µs] during the indicated periods at the indicated channel. In addition, the mark-to-space ratio is indicated in [‰]. |
| PERIOD_RATIO_E | = **PERIOD_RATIO** (→ p. 148) for the extended side |
| **PHASE** (→ p. 150) | Reads a pair of channels with fast inputs and compares the phase position of the signals |
| PHASE_E | = **PHASE** (→ p. 150) for the extended side |
| **PID1** (→ p. 179) | PID controller |
| **PID2** (→ p. 181) | PID controller |
| **PT1** (→ p. 183) | Controlled system with first-order delay |
| **PWM1000** (→ p. 160) | Initialises and configures a PWM-capable output channel<br>the mark-to-space ratio can be indicated in steps of 1 ‰ |
| PWM1000_E | = **PWM1000** (→ p. 160) for the extended side |
| **SERIAL_PENDING** (→ p. 120) | Determines the number of data bytes stored in the serial receive buffer |
| **SERIAL_RX** (→ p. 121) | Reads a received data byte from the serial receive buffer at each call |
| **SERIAL_SETUP** (→ p. 122) | Initialises the serial RS232 interface |
| **SERIAL_TX** (→ p. 123) | Transmits one data byte via the serial RS232 interface |
| **SET_DEBUG** (→ p. 208) | organises the DEBUG mode or the monitoring mode (depending on the TEST input) |
| **SET_IDENTITY** (→ p. 209) | Sets an application-specific program identification |

| Function element | Short description |
|---|---|
| **SET_INTERRUPT_I** (→ p. 125) | Conditional execution of a program part after an interrupt request via a defined input channel |
| **SET_INTERRUPT_XMS** (→ p. 127) | Conditional execution of a program part at an interval of x milliseconds |
| **SET_PASSWORD** (→ p. 210) | Sets a user password for access control to program and memory upload |
| **SOFTRESET** (→ p. 185) | leads to a complete reboot of the device |
| **TEMPERATURE** (→ p. 190) | Reads the current temperature in the device |
| **TIMER_READ** (→ p. 187) | Reads out the current system time in [ms]<br>Max. value = 49d 17h 2min 47s 295ms |
| **TIMER_READ_US** (→ p. 188) | Reads out the current system time in [µs]<br>Max. value = 1h 11min 34s 967ms 295µs |

## 5.1.2 Library ifm_CR0232_CANopenxMaster_Vxxyyzz.LIB

13707

x = 1...4 = number of the CAN interface

This library contains function blocks for operation of the device as a CANopen master.

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

| Function element | Short description |
|---|---|
| **CANx_MASTER_EMCY_HANDLER** (→ p. 85) | Handles the device-specific error status of the CANopen master on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_MASTER_SEND_EMERGENCY** (→ p. 86) | Sends application-specific error status of the CANopen master on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_MASTER_STATUS** (→ p. 88) | Status indication on CAN interface x of the device used as CANopen master<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |

## 5.1.3 Library ifm_CR0232_CANopenxSlave_Vxxyyzz.LIB

13709

x = 1...4 = number of the CAN interface

This library contains function blocks for operation of the device as a CANopen slave.

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

| Function element | Short description |
|---|---|
| **CANx_SLAVE_EMCY_HANDLER** (→ p. 95) | Handles the device-specific error status of the CANopen slave on CAN interface x:<br>• error register (index 0x1001) and<br>• error field (index 0x1003) of the CANopen object directory<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_SLAVE_NODEID** (→ p. 96) | Enables setting of the node ID of a CANopen slave on CAN interface x at runtime of the application program<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_SLAVE_SEND_EMERGENCY** (→ p. 97) | Sends application-specific error status of the CANopen slave on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_SLAVE_SET_PREOP** (→ p. 99) | Switches the operating mode of this CANopen slave from "OPERATIONAL" to "OPERATIONAL" on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **CANx_SLAVE_STATUS** (→ p. 100) | Shows the status of the device used as CANopen slave on CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |

## 5.1.4 Library ifm_CR0232_J1939_Vxxyyzz.LIB

13711

This library contains function blocks for engine control.

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

| Function element | Short description |
|---|---|
| **J1939_x** (→ p. 108) | CAN interface x: protocol handler for the communication profile SAE J1939<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **J1939_x_GLOBAL_REQUEST** (→ p. 109) | CAN interface x: handles global requesting and receipt of data from the J1939 network participants<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **J1939_x_RECEIVE** (→ p. 111) | CAN interface x: Receives a single message or a message block<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **J1939_x_RESPONSE** (→ p. 113) | CAN interface x: handles the automatic response to a request message<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **J1939_x_SPECIFIC_REQUEST** (→ p. 115) | CAN interface x: automatic requesting of individual messages from a specific J1939 network participant<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |
| **J1939_x_TRANSMIT** (→ p. 117) | CAN interface x: sends individual messages or message blocks<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet) |

### 5.1.5 Library ifm_hydraulic_32bit_Vxxyyzz.LIB

13729

This library contains function blocks for hydraulic controls.

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

| Function element | Short description |
|---|---|
| **CONTROL_OCC** (→ p. 163) | OCC = **O**utput **C**urrent **C**ontrol<br>Scales the input value [WORD] to an indicated current range |
| **JOYSTICK_0** (→ p. 165) | Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0... 1000 |
| **JOYSTICK_1** (→ p. 168) | Scales signals [INT] from a joystick D standardised to 0... 1000 |
| **JOYSTICK_2** (→ p. 172) | Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation |
| **NORM_HYDRAULIC** (→ p. 175) | Normalises a value [DINT] within defined limits to a value with new limits |

## 5.2 ifm function elements for the device CR0232

**Content**

Function elements: CAN layer 2 ........ 75
Function elements: CANopen master ........ 84
Function elements: CANopen slave ........ 94
Function elements: CANopen SDOs ........ 102
Function elements: SAE J1939 ........ 107
Function elements: serial interface ........ 119
Function elements: Optimising the PLC cycle via processing interrupts ........ 124
Function elements: processing input values ........ 129
Function elements: adapting analogue values ........ 132
Function elements: counter functions for frequency and period measurement ........ 137
Function elements: PWM functions ........ 152
Function elements: hydraulic control ........ 162
Function elements: controllers ........ 177
Function elements: software reset ........ 184
Function elements: measuring / setting of time ........ 186
Function elements: device temperature ........ 189
Function elements: saving, reading and converting data in the memory ........ 191
Function elements: data access and data check ........ 203

13988
28309

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

**Content**

CANx ....76
CANx_BAUDRATE ....77
CANx_BUSLOAD ....78
CANx_DOWNLOADID ....79
CANx_ERRORHANDLER ....80
CANx_RECEIVE ....81
CANx_TRANSMIT ....83

28402

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

## CANx

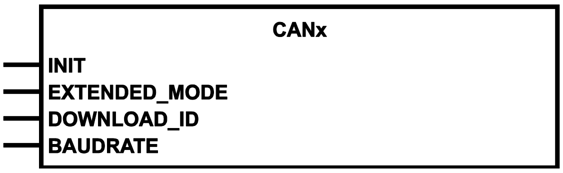
2159

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2162

CANx initialises the xth CAN interface
x = 1...n = number of the CAN interface (depending on the device, → data sheet).

The download ID must be different for every interface.

The baud rates of the individual CANx can be set to different values.

► The input INIT is only set for one cycle during reboot or restart of the interface!

A change of the download ID and/or baud rate only becomes valid after power off/on.

If the unit is not executed, the interface works with 11-bit identifiers.

### Parameters of the inputs

2163

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (in the 1st cycle):<br>Function block is initialised<br>FALSE: during further processing of the program |
| EXTENDED_MODE | BOOL := FALSE | TRUE: identifier of the CAN interface operates with 29 bits<br>FALSE: identifier of the CAN interface operates with 11 bits |
| DOWNLOAD_ID | BYTE | Download ID of CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet)<br>valid = 1...127<br>preset = 127 - (x-1) |
| BAUDRATE | WORD := 125 | Baud rate [kbits/s]<br>valid = 20, 50, 100, 125, 250, 500, 1000 |

## CANx_BAUDRATE

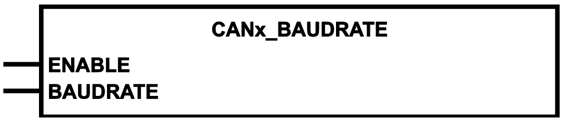
11834

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

11839

CANx_BAUDRATE sets the transmission rate for the bus participant.

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

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

### Parameters of the inputs

27644

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE (in the 1st cycle):<br>Adopt and activate parameters<br>else: this function is not executed |
| BAUDRATE | WORD := 125 | Baud rate [kbits/s]<br>valid = 20, 50, 100, 125, 250, 500, 1000 |

## CANx_BUSLOAD

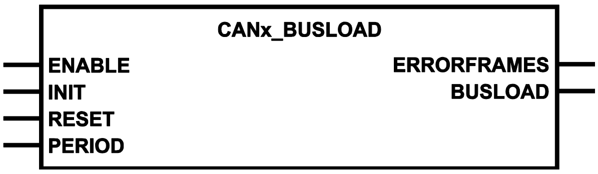
2178

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2180

Determines the current bus load on the CAN bus and counts the occurred error frames.

CANx_BUSLOAD determines the bus load via the number and length of the messages transferred via the CAN bus during the time indicated in PERIOD by taking the current baud rate into account. The value BUSLOAD is updated after the time indicated in PERIOD has elapsed.

If the bit RESET is permanently FALSE, the number of the error frames occurred since the last RESET is indicated.

## NOTE

If the communication on the CAN bus is carried out via the CANopen protocol, it is useful to set the value of PERIOD to the duration of the SYNC cycle.

The measurement period is not synchronised with the CANopen SYNC cycle.

### Parameters of the inputs

2181

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| INIT | BOOL | TRUE (only for 1 cycle):<br>configuration of the measurement duration PERIOD<br>FALSE: during further processing of the program |
| RESET | BOOL | TRUE: Set ERRORFRAME to "0"<br>FALSE: function element is not executed |
| PERIOD | WORD | Time in [ms] to determine the bus load<br>allowed = 20...1 000 ms |

### Parameters of the outputs

2182

| Parameter | Data type | Description |
|---|---|---|
| ERRORFRAMES | WORD | Number of error frames occurred on the CAN bus since the last reset |
| BUSLOAD | BYTE | Current bus load in [%] |

## CANx_DOWNLOADID

11841

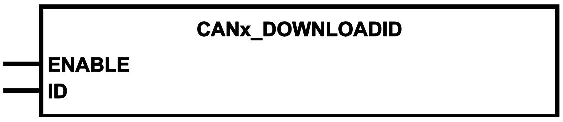
= CANx Download-ID

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

11846

CANx_DOWNLOADID sets the download identifier for the CAN interface x.

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

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

### Parameters of the inputs

27633

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE (in the 1st cycle):<br>Adopt and activate parameters<br>else: this function is not executed |
| ID | BYTE | Set download ID of CAN interface x<br>x = 1...n = number of the CAN interface (depending on the device, → data sheet)<br>allowed = 1...127<br>preset = 127 - (x-1) |

## CANx_ERRORHANDLER

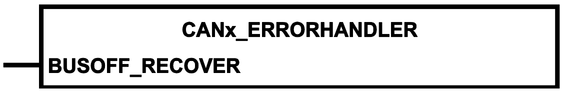
2174

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



CANx_ERRORHANDLER

BUSOFF_RECOVER

### Description

2329
13991

> ❗ If the automatic bus recover function is to be used (default setting) the function CANx_ERRORHANDLER must **not** be integrated and instanced in the program!

CANx_ERRORHANDLER executes a "manual" bus recovery on the CAN interface x.

► After a recognised CAN bus-off, call the function block for one cycle with BUSOFF_RECOVER = TRUE to make sure that the controller can send and receive on the CAN bus again.

► Then reset the error bit CANx_BUSOFF for this CAN interface in the application program.

> The CAN interface is operative again.

### Parameters of the inputs

27647

| Parameter | Data type | Description |
|---|---|---|
| BUSOFF_RECOVER | BOOL | TRUE (only 1 cycle):<br>> remedy 'bus off' status<br>> reboot of the CAN interfacex<br>FALSE: function element is not executed |

## CANx_RECEIVE

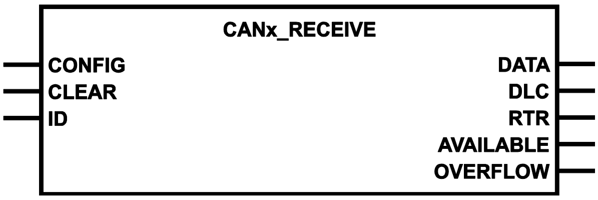
27450

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

13338

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 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.

► Depending on the CAN interface max. 256 instances are possible for the FB CANx_RECEIVE.

► In the Standard Mode all 2048 IDs can be used simultaneously
In the Extended Mode only 256 (any) IDs can be used simultaneously.

► Each ID (Standard or Extended) can be allotted to only one FB instance.
For multiple use of an ID: the last instance called.

► Set in FB CANx if CANx_RECEIVE should receive normal or extended frames.

> If CANx_RECEIVE is configured for the reception of a normal frame, the frame with this ID will not be transferred to a CANopen Stack ( if available).

> If an ID is set outside the permissible range (depending on the setting in CANx), the function block will not be executed.

► Evaluate the output AVAILABLE so that newly received data objects are read from the butter and processed in time.
Receive buffer: max. 16 software buffers per identifier.

> Each call of the FB decrements the byte AVAILABLE by 1.
If AVAILABLE = 0, there is no data in the buffer.

► Evaluate the output OVERFLOW to detect an overflow of the data buffer.
If OVERFLOW = TRUE, at least 1 data object has been lost.

## Parameters of the inputs

27658

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

## Parameters of the outputs

28454

| Parameter | Data type | Description |
|---|---|---|
| DATA | ARRAY [0..7] OF BYTE | received data, (1...8 bytes) |
| DLC | BYTE | Number of the bytes of the CAN telegram read from the receive buffer allowed: 0...8 |
| RTR | BOOL = FALSE | Received message was a **R**emote **T**ransmission **R**equest (wird hier nicht unterstützt) |
| AVAILABLE | BYTE | Number of the CAN telegrams received but not yet read from the receive buffer (before the FB is called).<br>Possible values = 0...16<br>0 = no valid data available |
| OVERFLOW | BOOL | TRUE: Overflow of the data buffer ⇨ loss of data!<br>FALSE: Data buffer is without data loss |

## CANx_TRANSMIT

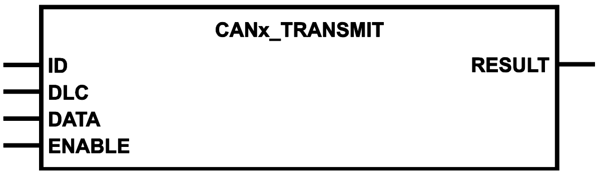
27812

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2166

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 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 1 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.

ℹ To put it simply, at 125 kbits/s one transmit order can be executed per 1 ms.

Several data objects with the same or with different ID 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.

Transmit buffer: max. 16 software buffers and 1 hardware buffer for all identifiers together.

### Parameters of the inputs

19813

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

### Parameters of the outputs

2168

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

### 5.2.2 Function elements: CANopen master

**Content**

CANx_MASTER_EMCY_HANDLER ..........85
CANx_MASTER_SEND_EMERGENCY ..........86
CANx_MASTER_STATUS ..........88

27841

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

## CANx_MASTER_EMCY_HANDLER

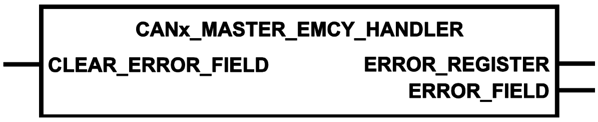
2006

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_CR0232_CANopenxMaster_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27782

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** (→ p. 86).

### Parameters of the inputs

27655

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

### Parameters of the outputs

27759

| 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)<br>ERROR_FIELD[0]: number of stored errors<br>ERROR_FIELD[1...5]: Stored errors, the most recent error is shown on index [1] |

## CANx_MASTER_SEND_EMERGENCY

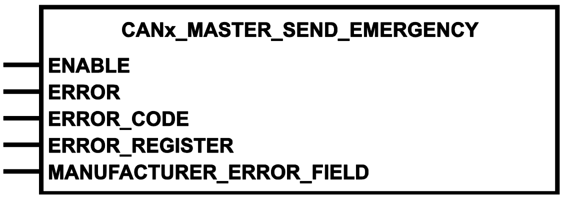
2012

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

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_CANopen**x**Master_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

27783

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** (→ p. 85) must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY.

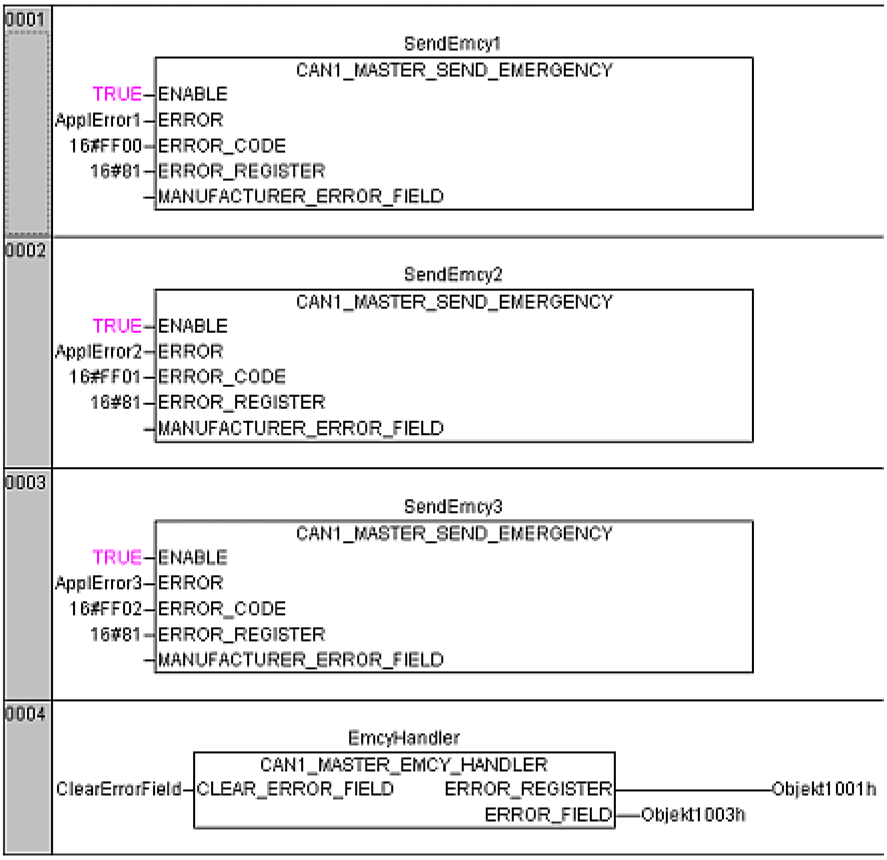
### Parameters of the inputs

27656

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> 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.<br>FALSE ⇨ TRUE (edge):<br>sends the next error code<br>if input was not TRUE in the last second<br>TRUE ⇨ FALSE (edge)<br>AND the fault is no longer indicated:<br>after a delay of approx. 1 s:<br>> zero error message is sent<br>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.<br>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.<br>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

27890



```
0001
                        SendEmcy1
            CAN1_MASTER_SEND_EMERGENCY
     TRUE-ENABLE
AppIError1-ERROR
  16#FF00-ERROR_CODE
    16#81-ERROR_REGISTER
         -MANUFACTURER_ERROR_FIELD

0002
                        SendEmcy2
            CAN1_MASTER_SEND_EMERGENCY
     TRUE-ENABLE
AppIError2-ERROR
  16#FF01-ERROR_CODE
    16#81-ERROR_REGISTER
         -MANUFACTURER_ERROR_FIELD

0003
                        SendEmcy3
            CAN1_MASTER_SEND_EMERGENCY
     TRUE-ENABLE
AppIError3-ERROR
  16#FF02-ERROR_CODE
    16#81-ERROR_REGISTER
         -MANUFACTURER_ERROR_FIELD

0004
                     EmcyHandler
               CAN1_MASTER_EMCY_HANDLER
ClearErrorField-CLEAR_ERROR_FIELD    ERROR_REGISTER-Objekt1001h
                                        ERROR_FIELD-Objekt1003h
```

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

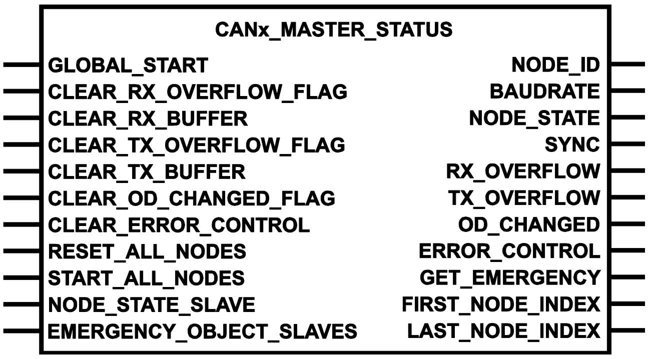
2692

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_CR0232_CANopenxMaster_Vxxyyzz.LIB`

### Symbol in CODESYS:



## Description

27780

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

19861

| Parameter | Data type | Description |
|---|---|---|
| GLOBAL_START | BOOL | TRUE: All connected network participants (slaves) are started simultaneously during network initialisation (⇨ state OPERATIONAL).<br>FALSE: The connected network participants are started one after the other. |
| CLEAR_RX_OVERFLOW_FLAG | BOOL | FALSE ⇨ TRUE (edge):<br>Clear error flag RX_OVERFLOW<br>else: this function is not executed |
| CLEAR_RX_BUFFER | BOOL | FALSE ⇨ TRUE (edge):<br>Delete data in the receive buffer<br>else: this function is not executed |
| CLEAR_TX_OVERFLOW_FLAG | BOOL | FALSE ⇨ TRUE (edge):<br>Clear error flag TX_OVERFLOW<br>else: this function is not executed |
| CLEAR_TX_BUFFER | BOOL | FALSE ⇨ TRUE (edge):<br>Delete data in the transmit buffer<br>else: this function is not executed |
| CLEAR_OD_CHANGED_FLAG | BOOL | FALSE ⇨ TRUE (edge):<br>Delete flag OD_CHANGED<br>else: this function is not executed |
| CLEAR_ERROR_CONTROL | BOOL | FALSE ⇨ TRUE (edge):<br>Delete the guard error list (ERROR_CONTROL)<br>else: this function is not executed |
| RESET_ALL_NODES | BOOL | FALSE ⇨ TRUE (edge):<br>All connected network participants (slaves) are reset via NMT command<br>else: this function is not executed |
| START_ALL_NODES | BOOL | FALSE ⇨ TRUE (edge):<br>All connected network participants (slaves) are started via NMT command<br>else: this function is not executed |
| NODE_STATE_SLAVES | DWORD | Pointer address to a array [0.. MAX_NODEINDEX] of CANx_NODE_STATE<br>The status information of the slaves in the CANopen network is to be written into this array. The behaviour of the slaves can be controlled by means of access to certainly values.<br>MAX_NODEINDEX is a constant, which is calculated by CODESYS during the compiling of the application.<br>❗ Determine the address by means of the operator ADR and assigne it to the POU!<br>Example code → chapter **Example: CANx_MASTER_STATUS** (→ p. 92) |
| EMERGENCY_OBJECT_SLAVES | DWORD | Pointer address to a array [0.. MAX_NODEINDEX] of CANx_EMERGENCY_MESSAGE<br>Shows the last error messages of all network nodes.<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |

## Parameters of the outputs

2696

| 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 | INT | Current status of CANopen master |
| SYNC | BOOL | SYNC signal of the CANopen master<br>TRUE: In the last cycle a SYNC signal was sent<br>FALSE: In the last cycle no SYNC signal was sent |
| RX_OVERFLOW | BOOL | TRUE: Error: receive buffer overflow<br>FALSE: no overflow |
| TX_OVERFLOW | BOOL | TRUE: Error: transmission buffer overflow<br>FALSE: no overflow |
| OD_CHANGED | BOOL | TRUE: Data in the object directory of the CANopen master have been changed<br>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) |
| GET_EMERGENCY | STRUCT CANx_EMERGENY_MESSAGE | At the output the data for the structure CANx_EMERGENCY_MESSAGE are available.<br>The last received EMCY message in the CANopen network is always displayed.<br>To obtain a list of all occurred errors, the array EmergencyObjectSlavesArray must be evaluated! |
| FIRST_NODE_INDEX | INT | Section where the node numbers of the nodes (slaves) connected to this CAN bus are located |
| LAST_NODE_INDEX | INT | |

## Internal structure parameters

2698

Here you can see the structures of the arrays used in this function block.

Using Controller CR0032 as an example, the following code fragments show the use of the function block CANx_MASTER_STATUS→ chapter **Example: CANx_MASTER_STATUS** (→ p. 92).

### Structure of CANx_EMERGENCY_MESSAGE

13996

The structure is defined by the global variables of the library `ifm_CR0232_CANopenMaster_Vxxyyzz.LIB`.

| Parameter | Data type | Description |
|---|---|---|
| NODE_ID | BYTE | Node ID of the participant the EMCY has been received from |
| ERROR_CODE | WORD | Error code indicating which error has occurred.<br>→ CANopen specification CiA Draft Standard 301 Version 4 |
| ERROR_REGISTER | BYTE | Value in the error register (index 0x1001/00) of the sending participant |
| MANUFACTURER_ERROR_FIELD | ARRAY [0..4] OF BYTE | Manufacturer-specific data field in EMCY message |

### Structure of CANx_NODE_STATE

13997

The structure is defined by the global variables of the library `ifm_CR0232_CANopenMaster_Vxxyyzz.LIB`.

| Parameter | Data type | Description |
|---|---|---|
| NODE_ID | BYTE | Node ID of the CANopen slave the status information and configuration flags in the structure belong to |
| NODE_STATE | BYTE | Current state of the CANopen slave seen from the perspective of the CANopen stack of the CANopen master |
| LAST_STATE | BYTE | The last known state of the CANopen slave<br>0 = receive bootup message from CANopen slave<br>4 = CANopen slave in PRE-OPERATIONAL state and is configured via SDO access<br>5 = CANopen slave in OPERATIONAL state<br>127 = CANopen slave in PRE-OPERATIONAL state |
| RESET_NODE | BOOL | Flag for manual reset of CANopen slave<br>(NMT command = `Reset_Node`) |
| START_NODE | BOOL | Flag for manual start of CANopen slave<br>(NMT command = `start`) |
| PREOP_NODE | BOOL | Flag to manually set the CANopen slave to the PRE-OPERATIONAL state<br>NMT command = `enter PRE-OPERATIONAL`) |
| SET_TIMEOUT_STATE | BOOL | Flag used to manually skip initialisation of a CANopen slave if the following applies:<br>• slave does not exist in network<br>• **and** slave is not configured as optional |
| SET_NODE_STATE | BOOL | Flag for manual initialisation of a CANopen slave<br>When accessing object 0x1000, the slave had identified itself as a device type other than the one indicated in the EDS file incorporated in the CODESYS configuration of the controller. |

## Example: CANx_MASTER_STATUS

27895

### Slave information

2699

To be able to access the information of the individual CANopen nodes, you must create an array for the corresponding structure. The structures are contained in the library. You can see them under [Data types] in the library manager.

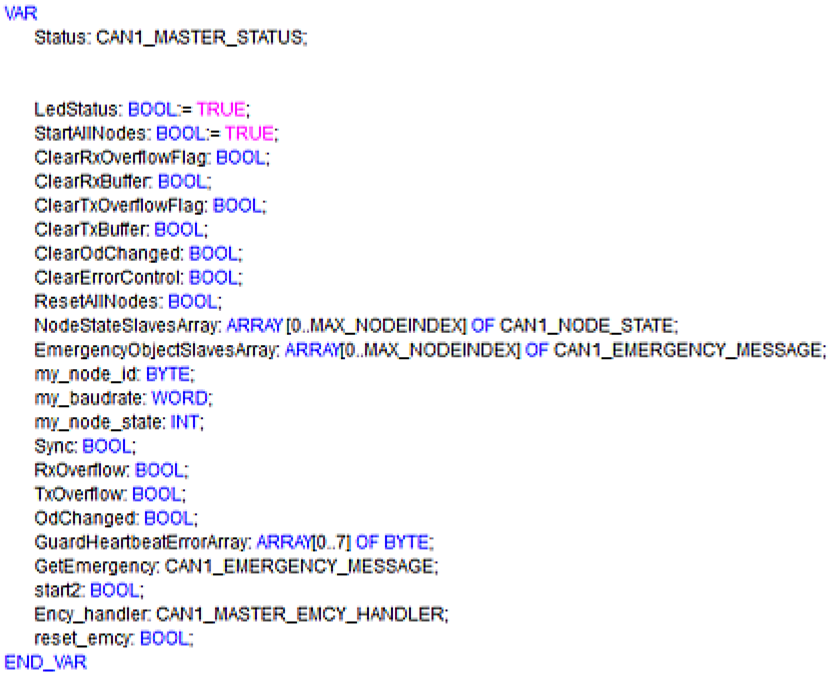
The number of the array elements is determined by the global variable MAX_NODEINDEX which is automatically generated by the CANopen stack. It contains the number of the slaves minus 1 indicated in the network configurator.

> 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 example to CAN1_MASTER_STATUS

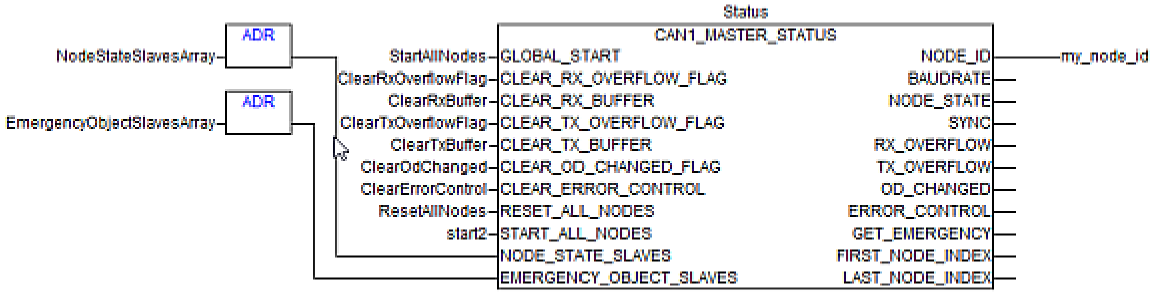
20651

Declaration of the variables:



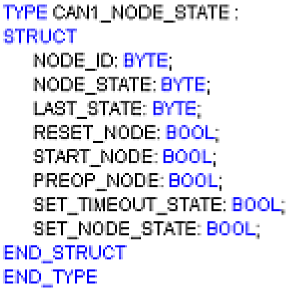
```
VAR
    Status: CAN1_MASTER_STATUS;


    LedStatus: BOOL:= TRUE;
    StartAllNodes: BOOL:= TRUE;
    ClearRxOverflowFlag: BOOL;
    ClearRxBuffer: BOOL;
    ClearTxOverflowFlag: BOOL;
    ClearTxBuffer: BOOL;
    ClearOdChanged: BOOL;
    ClearErrorControl: BOOL;
    ResetAllNodes: BOOL;
    NodeStateSlavesArray: ARRAY [0..MAX_NODEINDEX] OF CAN1_NODE_STATE;
    EmergencyObjectSlavesArray: ARRAY[0..MAX_NODEINDEX] OF CAN1_EMERGENCY_MESSAGE;
    my_node_id: BYTE;
    my_baudrate: WORD;
    my_node_state: INT;
    Sync: BOOL;
    RxOverflow: BOOL;
    TxOverflow: BOOL;
    OdChanged: BOOL;
    GuardHeartbeatErrorArray: ARRAY[0..7] OF BYTE;
    GetEmergency: CAN1_EMERGENCY_MESSAGE;
    start2: BOOL;
    Ency_handler: CAN1_MASTER_EMCY_HANDLER;
    reset_emcy: BOOL;
END_VAR
```

Example of the program:



## Structure node status

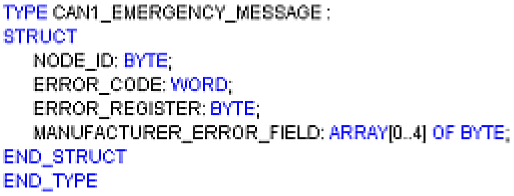
27713



```
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

27712



```
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
```

### 5.2.3 Function elements: CANopen slave

**Content**

CANx_SLAVE_EMCY_HANDLER ....95
CANx_SLAVE_NODEID ....96
CANx_SLAVE_SEND_EMERGENCY ....97
CANx_SLAVE_SET_PREOP ....99
CANx_SLAVE_STATUS ....100

27838

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

## CANx_SLAVE_EMCY_HANDLER

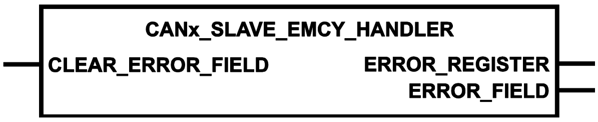
2050

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_CR0232_CANopenxSlave_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27786

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** (→ p. 97) call CANx_SLAVE_EMCY_HANDLER once!

### Parameters of the inputs

27659

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

### Parameters of the outputs

27749

| 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)<br>ERROR_FIELD[0]: number of stored errors<br>ERROR_FIELD[1...5]: Stored errors, the most recent error is shown on index [1] |

## CANx_SLAVE_NODEID

2044

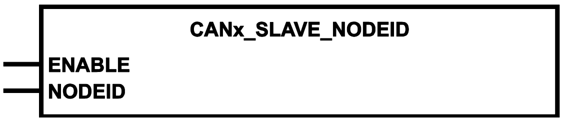
= CANx Slave Node-ID

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

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_CANopenxSlave_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

27781

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

27654

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | FALSE ⇨ TRUE (edge):<br>Adopt and activate parameters<br>else: this function is not executed |
| NODEID | BYTE | node ID = ID of the node<br>permissible values = 1...127 |

## CANx_SLAVE_SEND_EMERGENCY

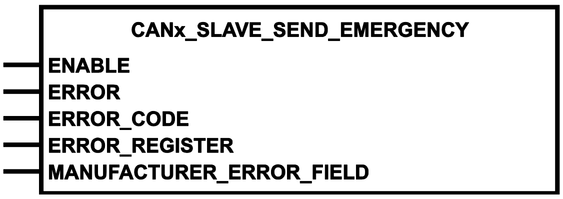
2056

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

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_CANopenxSlave_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

27787

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.

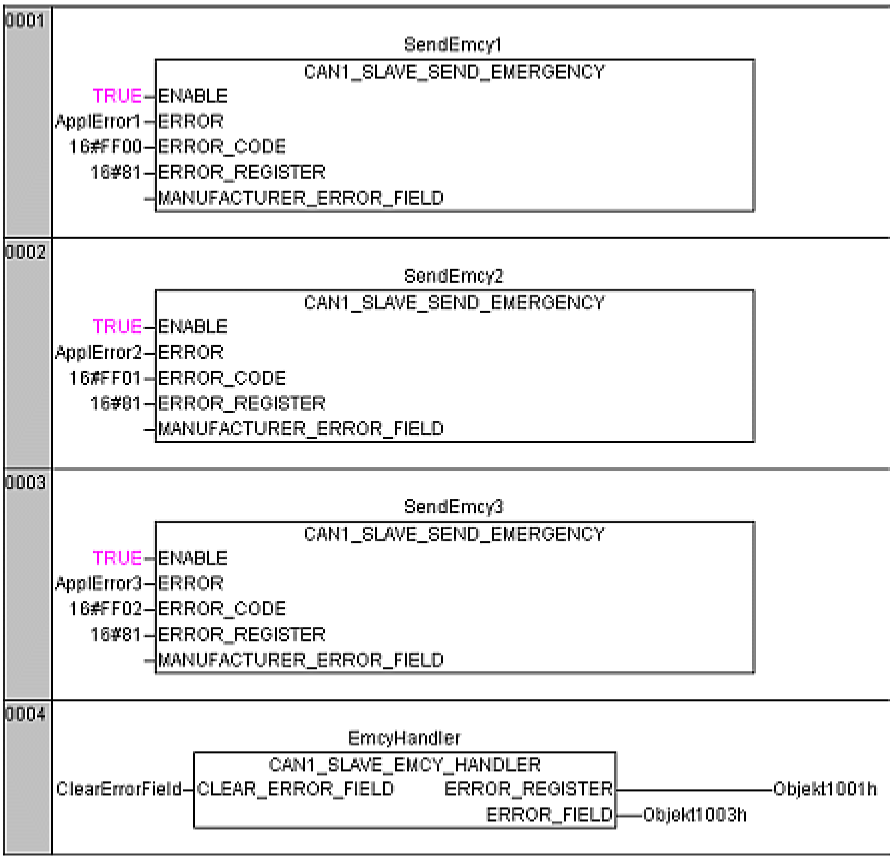
### Parameters of the inputs

27660

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> 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.<br>FALSE ⇨ TRUE (edge):<br>sends the next error code<br>if input was not TRUE in the last second<br>TRUE ⇨ FALSE (edge)<br>AND the fault is no longer indicated:<br>after a delay of approx. 1 s:<br>> zero error message is sent<br>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.<br>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.<br>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

27896



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_SET_PREOP

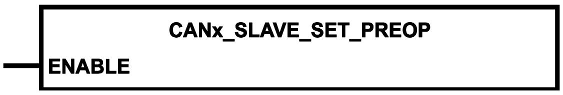
2700

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

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_CANopen**x**Slave_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

2703

CANx_SLAVE_SET_PREOP switches the operating mode of this CANopen slave from "OPERATIONAL" to "PRE-OPERATIONAL".

Normally, in case of a fault the controller switches as follows:

- a FATAL ERROR results in SOFT RESET of the controller
- an ERROR STOP results in a SYSTEM STOP

Under certain conditions it might be necessary that the application sets the operating mode of the device working as a slave to "PRE-OPERATIONAL". This is done via the FB described here.

### Parameters of the inputs

2704

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | FALSE ⇨ TRUE (edge):<br>Set slave to PRE-OPERATIONAL<br>else: this function is not executed |

## CANx_SLAVE_STATUS

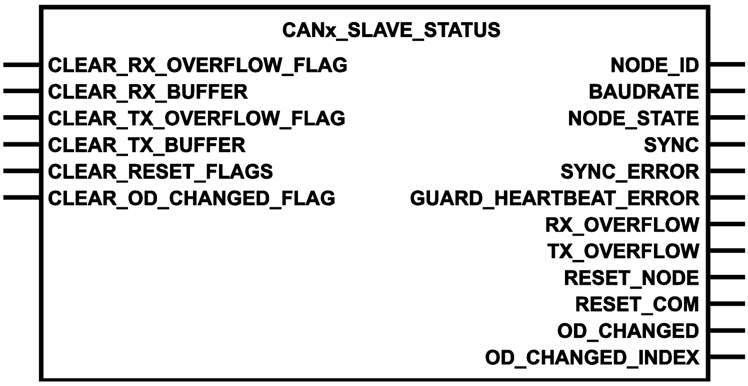
2706

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

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_CANopenxSlave_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

2707

CANx_SLAVE_STATUS shows the status of the device used as CANopen slave.

❗ We urgently recommend carrying out the evaluation of the network status via this function block.

### Parameters of the inputs

2708

| Parameter | Data type | Description |
|---|---|---|
| CLEAR_RX_OVERFLOW_FLAG | BOOL | FALSE ⇨ TRUE (edge):<br>Clear error flag RX_OVERFLOW<br>else: this function is not executed |
| CLEAR_RX_BUFFER | BOOL | FALSE ⇨ TRUE (edge):<br>Delete data in the receive buffer<br>else: this function is not executed |
| CLEAR_TX_OVERFLOW_FLAG | BOOL | FALSE ⇨ TRUE (edge):<br>Clear error flag TX_OVERFLOW<br>else: this function is not executed |
| CLEAR_TX_BUFFER | BOOL | FALSE ⇨ TRUE (edge):<br>Delete data in the transmit buffer<br>else: this function is not executed |
| CLEAR_RESET_FLAGS | BOOL | FALSE ⇨ TRUE (edge):<br>Clear flag RESET_NODE<br>Clear flag RESET_COM<br>else: this function is not executed |
| CLEAR_OD_CHANGED_FLAGS | BOOL | FALSE ⇨ TRUE (edge):<br>Clear flag OD_CHANGED<br>Clear flag OD_CHANGED-_INDEX<br>else: this function is not executed |

## Parameters of the outputs

27746

| 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<br>0 = Bootup message sent<br>4 = CANopen slave in PRE-OPERATIONAL state and is configured via SDO access<br>5 = CANopen slave in OPERATIONAL state<br>127 = CANopen slave in PRE-OPERATIONAL state |
| SYNC | BOOL | SYNC signal of the CANopen master<br>TRUE: In the last cycle a SYNC signal was received<br>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)<br>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<br>FALSE: no guarding or heartbeat error |
| RX_OVERFLOW | BOOL | TRUE: Error: receive buffer overflow<br>FALSE: no overflow |
| TX_OVERFLOW | BOOL | TRUE: Error: transmission buffer overflow<br>FALSE: no overflow |
| RESET_NODE | BOOL | TRUE: the CANopen stack of the slave was reset by the master<br>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<br>FALSE: the communication interface was not reset |
| OD_CHANGED | BOOL | TRUE: Data in the object directory of the CANopen master have been changed<br>FALSE: no data change |
| OD_CHANGED_INDEX | INT | Index of the object directory entry changed last |

## 5.2.4 Function elements: CANopen SDOs

**Content**

CANx_SDO_READ ....................................................................................................................103
CANx_SDO_WRITE..................................................................................................................105

28407

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

## CANx_SDO_READ

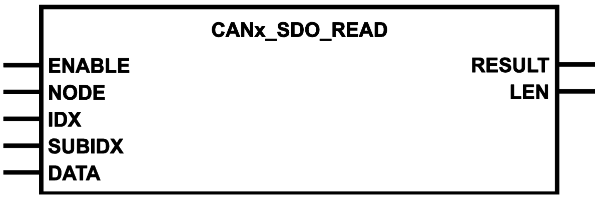
27448

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27791

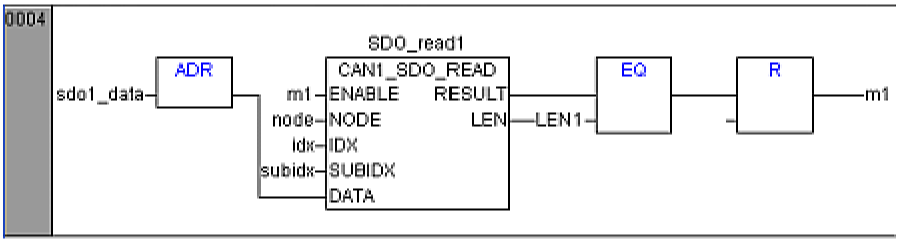
CANx_SDO_READ reads the →**SDO** (→ p. 253) with the indicated indexes from the node.

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.

> ! Danger of data loss!
> Allocate enough memory space for the requested SDO!
> Otherwise the data following behind will be overwritten.

### Example:



### Parameters of the inputs

27629

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| NODE | BYTE | ID of the node<br>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<br>valid length = 0...255<br>! Determine the address by means of the operator ADR and assigne it to the POU! |

## Parameters of the outputs

27761

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(possible messages → following table) |
| LEN | WORD | length of the entry in "number of bytes"<br>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<br>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

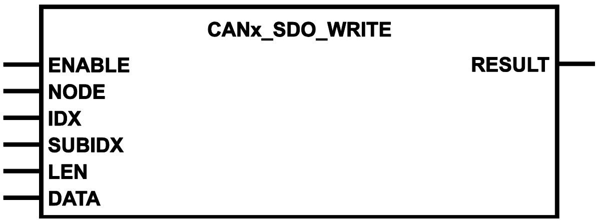
27807

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_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27790

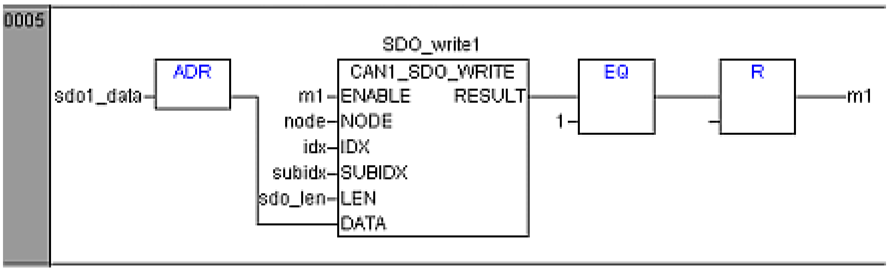
CANx_SDO_WRITE writes the →**SDO** (→ p. 253) with the specified indexes to the node.

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

27628

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| NODE | BYTE | ID of the node<br>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"<br>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<br>permissible length = 0...255<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |

## Parameters of the outputs

27760

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(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

**Content**

J1939_x ....................108
J1939_x_GLOBAL_REQUEST ....................109
J1939_x_RECEIVE ....................111
J1939_x_RESPONSE ....................113
J1939_x_SPECIFIC_REQUEST ....................115
J1939_x_TRANSMIT ....................117

28361

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

## J1939_x

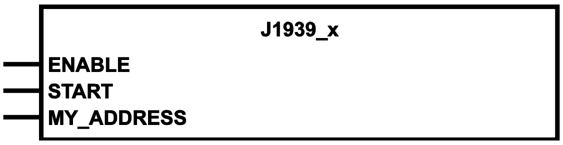
2274

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_CR0232_J1939_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2276

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

To handle the communication, the protocol handler must be called in each program cycle. To do so, the input ENABLE is set to TRUE.

⚠ Once set, ENABLE must remain 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

27639

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

## J1939_x_GLOBAL_REQUEST

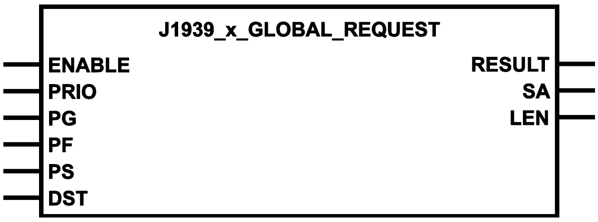
2282

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_CR0232_J1939_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27878

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]

27878

## 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 POU!

► 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

27638

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

### Info

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

## Parameters of the outputs

20789

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(possible messages → following table) |
| SA | BYTE | J1939 address of the answering device |
| LEN | WORD | number of received bytes |

Possible results for RESULT:

| Value<br>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) |

## J1939_x_RECEIVE

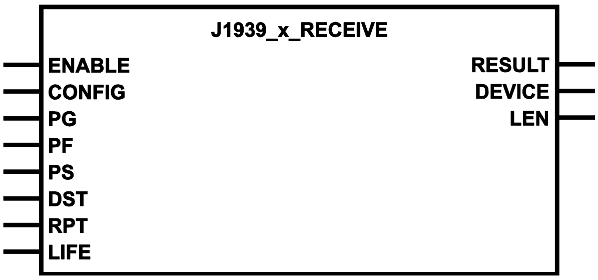
2278

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_CR0232_J1939_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27785

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.

27785

**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 POU!

❗ 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

27661

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

## Parameters of the outputs

27730

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(possible messages → following table) |
| DEVICE | BYTE | J1939 address of the sender |
| LEN | WORD | number of received bytes |

Possible results for RESULT:

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

## J1939_x_RESPONSE

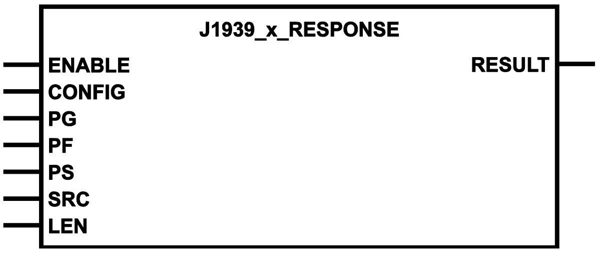
2280

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_CR0232_J1939_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27872

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 POU!

► In addition, the number of data bytes to be transmitted is assigned.

### Parameters of the inputs

27684

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

## Parameters of the outputs

27751

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(possible messages → following table) |

Possible results for RESULT:

| Value<br>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

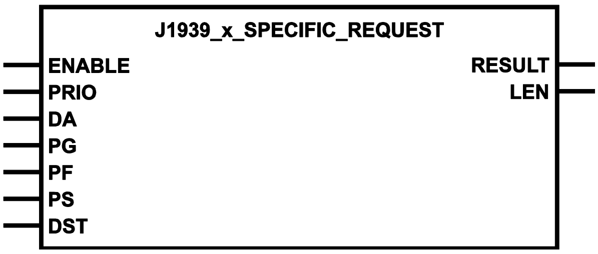
2281

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_CR0232_J1939_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27873

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]

27873

**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 POU!

► 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

27683

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

### Info

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

## Parameters of the outputs

27729

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(possible messages → following table) |
| LEN | WORD | number of received bytes |

Possible results for RESULT:

| Value<br>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

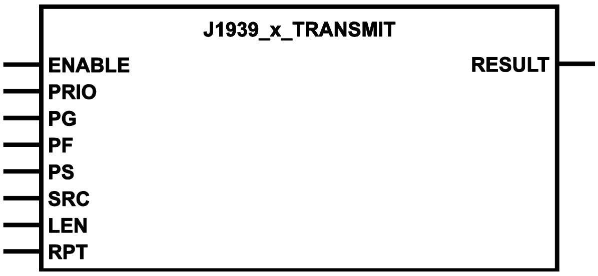
279

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_CR0232_J1939_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27875

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 POU!

► 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

27686

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

### Info

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

## Parameters of the outputs

27732

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BYTE | feedback of the function block<br>(possible messages → following table) |

Possible results for RESULT:

| Value<br>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

**Content**

SERIAL_PENDING ..... 120
SERIAL_RX ..... 121
SERIAL_SETUP ..... 122
SERIAL_TX ..... 123

13011

32570

**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 any of the CAN interfaces.

The function blocks listed below allow you to use the serial interface in the application program.

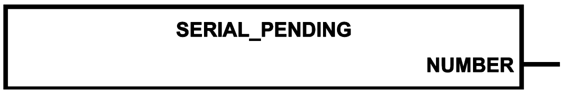
## SERIAL_PENDING

27711

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

12994

SERIAL_PENDING determines the number of data bytes stored in the serial receive buffer.

In contrast to SERIAL_RX the content of the buffer remains unchanged after this function has been called.

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!

32976

**! 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 any of the CAN interfaces.

### Parameters of the outputs

12996

| Parameter | Data type | Description |
|---|---|---|
| NUMBER | WORD | Number of data bytes received (1...1 000) |

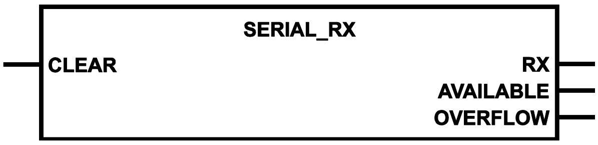
## SERIAL_RX

27722

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

12997

SERIAL_RX reads a received data byte from the serial receive buffer at each call.

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!

32837

**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 any of the CAN interfaces.

### Parameters of the inputs

27694

| Parameter | Data type | Description |
|---|---|---|
| CLEAR | BOOL | TRUE: delete receive buffer<br>FALSE: function element is not executed |

### Parameters of the outputs

12931

| Parameter | Data type | Description |
|---|---|---|
| Rx | BYTE | Byte data received from the receive buffer |
| AVAILABLE | WORD | Number of received bytes available in the receive buffer BEFORE the call of the function block:<br>0 = no data received<br>1...1 000 = number of bytes in the receive buffer |
| OVERFLOW | BOOL | TRUE: Overflow of the data buffer ⇨ loss of data!<br>FALSE: Data buffer is without data loss |

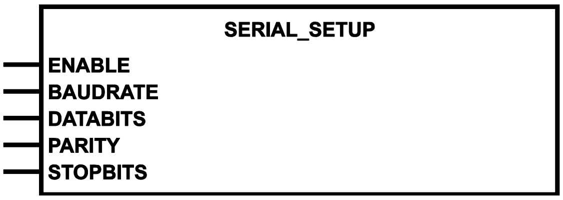
## SERIAL_SETUP

27723

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

13000

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 last preset value applies.

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.

32977

> **❗ 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 any of the CAN interfaces.

### Parameters of the inputs

13002

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE (only for 1 cycle):<br>Initialise interface<br>FALSE: during further processing of the program |
| BAUD RATE | DWORD | Baud rate<br>permissible values → data sheet<br>preset value → data sheet |
| DATABITS | BYTE := 8 | Number of data bits<br>allowed = 7 or 8 |
| PARITY | BYTE := 0 | Parity<br>allowed: 0=none, 1=even, 2=odd<br>❗ With parameter setting DATABITS = 7 and PARITY = 0: function block operates with PARITY = 1 |
| STOPBITS | BYTE := 1 | Number of stop bits<br>allowed = 1 or 2 |

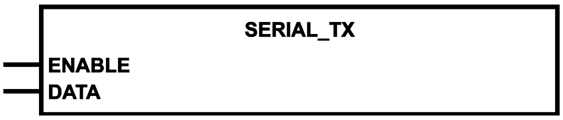
## SERIAL_TX

27720

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

13003

SERIAL_TX transmits one data byte via the serial RS232 interface.

The FiFo transmission memory contains 1 000 bytes.

Using the input ENABLE the transmission can be enabled or disabled.

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!

32838

**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 any of the CAN interfaces.

### Parameters of the inputs

28461

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

### 5.2.7 Function elements: Optimising the PLC cycle via processing interrupts

**Content**

SET_INTERRUPT_I ............................................................................................................125
SET_INTERRUPT_XMS ....................................................................................................127

20965
28098

Here we show you functions to optimise the PLC cycle.

28098

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.

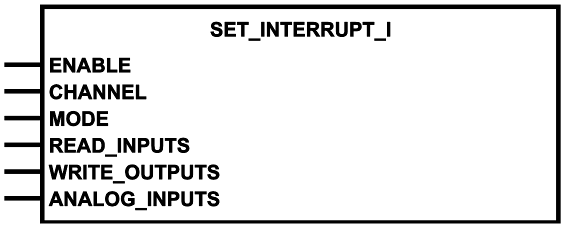
## SET_INTERRUPT_I

27727

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_Vxxyyzz.LIB

### Symbol in CODESYS:



### Description

19361
33947

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!

33947

**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.

19365

## 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: IN00...IN07
- Outputs: Q00...Q07

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

27646

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE (only for 1 cycle): initialisation of the function block<br>FALSE: unit is not executed |
| CHANNEL | BYTE | Number of interrupt input<br>0...7 for the inputs IN00...IN07 |
| MODE | BYTE | Type of edge at the input CHANNEL which triggers the interrupt<br>1 = rising edge (standard value)<br>2 = falling edge<br>3 = rising and falling edge<br>> 3 = standard value |
| READ_INPUTS | BOOL | TRUE: read the inputs 0...7 before calling the program and write into the input flags I00...I07<br>FALSE: only read the channel indicated under CHANNEL and write to the corresponding input flag Ixx |
| WRITE_OUTPUTS | BOOL | TRUE: write the current values of the output flags Q00...Q07 to the outputs after completion of the program sequence<br>FALSE: do not write outputs |
| ANALOG_INPUTS | BOOL | TRUE: read inputs 0...7 and write the unfiltered, uncalibrated analogue values to the flags ANALOG_IRQ00...07<br>FALSE: do not write flags ANALOG_IRQ00...07 |

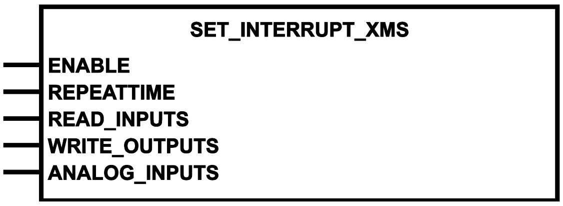
## SET_INTERRUPT_XMS

28322

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

28132

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.

28132

## 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

28486

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

### 5.2.8 Function elements: processing input values

**Content**

INPUT_ANALOG ....................................................................................................................130

1602
28368

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

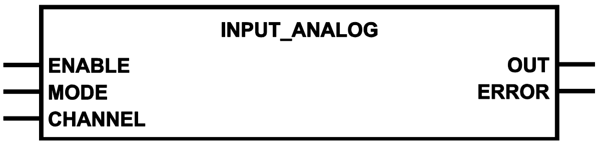
2245

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2361
12916

INPUT_ANALOG enables the following operating modes at the input channels.
Details → chapter **Possible operating modes inputs/outputs** (→ p. 233)

The function block provides the current analogue value at the selected analogue channel. The analogue values are provided as standardised values. At the same time the uncalibrated raw values are provided via the system flags ANALOGxx.

- ► For frequency and period measurements as well as counter functions: set MODE=1 (= IN_DIGITAL_H)!

The measurement and the output value results from the operating mode indicated via MODE:

12917

| MODE<br>dec | hex | Input operating mode | | FB Output<br>OUT | Unit |
|---|---|---|---|---|---|
| 0 | 00 | deactivated | | --- | --- |
| 1 | 01 | binary input, minus switching (BH) | IN_DIGITAL_H | 0 / 1 | --- |
| 2 | 02 | binary input, plus switching (BL) | IN_DIGITAL_L | 0 / 1 | --- |
| 4 | 04 | current input | IN_CURRENT | 0…20 000 | µA |
| 8 | 08 | voltage input | IN_VOLTAGE_10 | 0…10 000 | mV |
| 16 | 10 | voltage input | IN_VOLTAGE_30 | 0…32 000 | mV |
| 32 | 20 | voltage input, ratiometric | IN_RATIO | 0…1 000 | ‰ |
| 64 | 40 | diagnosis | IN_DIAGNOSIS | --- | --- |
| 128 | 80 | fast input | IN_FAST | 0 / 1 | --- |

18414

If inputI15 is not used:

- ► Configure input I15 as binary input!

## Parameters of the inputs

2362

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| MODE | BYTE | operating mode of the input channel: |
| | | 0 = 0x00 IN_NOMODE (off; preset is active) |
| | | 1 = 0x01 IN_DIGITAL_H preset |
| | | 2 = 0x02 IN_DIGITAL_L |
| | | 4 = 0x04 IN_CURRENT 0…20 000 µA |
| | | 8 = 0x08 IN_VOLTAGE10 0…10 000 mV |
| | | 16 = 0x10 IN_VOLTAGE30 0…32 000 mV |
| | | 32 = 0x20 IN_RATIO 0...1 000 ‰ |
| | | 64 = 0x40 IN_DIAGNOSTIC |
| | | 128 = 0x80 IN_FAST □ |
| CHANNEL | BYTE | Number of input channel<br>0...15 for the inputs I00...I15<br>For the function block xxx_**E** (if available) applies:<br>0...15 for the inputs I00_E...I15_E |

## Parameters of the outputs

2363

| Parameter | Data type | Description |
|---|---|---|
| OUT | WORD | Output value according to MODE<br>in case of an invalid setting: OUT = "0" |
| ERROR | BYTE | 00 = okay |
| | | 01 = over-current for IN_CURRENT |
| | | 02 = short circuit to VBB for IN_DIGITAL_H, OUT_DIAGNOSTIC |
| | | 03 = wire break for IN_DIGITAL_H, OUT_DIAGNOSTIC |

### 5.2.9 Function elements: adapting analogue values

**Content**

NORM.....133
NORM_DINT .....135
NORM_REAL .....136

27840

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

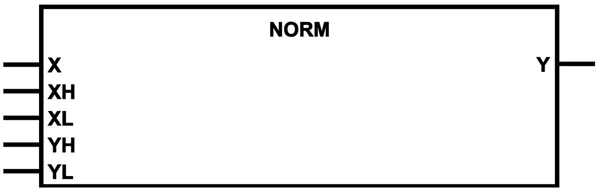
## NORM

27675

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27772

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

27679

| 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

27742

| Parameter | Data type | Description |
|---|---|---|
| Y | WORD | output value |

## Example: NORM (1)

27883

| | | |
|---|---|---|
| lower limit value input<br>upper limit value input<br>lower limit value output<br>upper limit value output | 0<br>100<br>0<br>2000 | XL<br>XH<br>YL<br>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)

27884

| | | |
|---|---|---|
| lower limit value input<br>upper limit value input<br>lower limit value output<br>upper limit value output | 2000<br>0<br>0<br>100 | XL<br>XH<br>YL<br>YH |

then the FB converts the input signal for example as follows:

| **from X =** | 1000 | 0 | 2000 | 1500 |
|---|---|---|---|---|
| | ↓ | ↓ | ↓ | ↓ |
| **to Y =** | 50 | 100 | 0 | 25 |

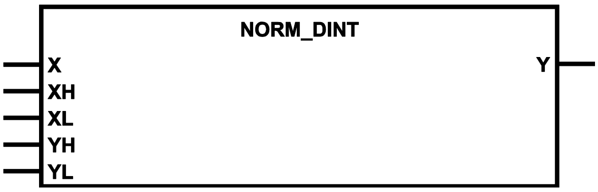
## NORM_DINT

2217

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2355

NORM_DINT normalises a value within defined limits to a value with new limits.

The FB normalises a value of type DINT 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.

► The result of the calculation (XH-XL)•(YH-YL) must remain in the value range of data type DINT (-2 147 483 648...2 147 483 647)!

> 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

2359

| Parameter | Data type | Description |
|---|---|---|
| X | DINT | current input value |
| XH | DINT | upper limit of input value range |
| XL | DINT | lower limit of input value range |
| YH | DINT | upper limit of output value range |
| YL | DINT | lower limit of output value range |

### Parameters of the outputs

2360

| Parameter | Data type | Description |
|---|---|---|
| Y | DINT | output value |

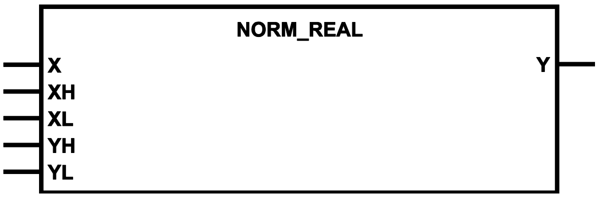
## NORM_REAL

2218

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2358

NORM_REAL normalises a value within defined limits to a value with new limits.

The FB normalises a value of type REAL 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.
>
> ► The result of the calculation (XH-XL)•(YH-YL) must remain in the value range of data type REAL (-3,402823466•$10^{38}$...3,402823466•$10^{38}$)!
>
> \> 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

2356

| Parameter | Data type | Description |
|---|---|---|
| X | REAL | Input value |
| XH | REAL | Upper limit of output value range |
| XL | REAL | Lower limit of the input value range |
| YH | REAL | Upper limit of the output value range |
| YL | REAL | Lower limit of output value range |

### Parameters of the outputs

2357

| Parameter | Data type | Description |
|---|---|---|
| Y | REAL | Output value |

### 5.2.10 Function elements: counter functions for frequency and period measurement

**Content**

FAST_COUNT ....138
FREQUENCY ....140
FREQUENCY_PERIOD ....142
INC_ENCODER ....144
PERIOD ....146
PERIOD_RATIO ....148
PHASE ....150

2322

Depending on the **ecomat*mobile*** device up to 16*) fast inputs are supported which can process input frequencies of up to 30 kHz. In addition to frequency measurement, the inputs can also be used for the evaluation of incremental encoders (counter function).
*) ExtendedController: up to 32 fast inputs

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 | 0.1...30 000 Hz | Measurement of the frequency on the indicated channel.<br>Measurement error is reduced in case of high frequencies |
| PERIOD | 0.1...5 000 Hz | Measurement of frequency and period duration (cycle time) on the indicated channel |
| PERIOD_RATIO | 0.1...5 000 Hz | Measurement of frequency and period duration (cycle time) as well as mark-to-space ratio [‰] on the indicated channel |
| FREQUENCY_PERIOD | 0.1...30 000 Hz | The FB combines the two FBs FREQUENCY and PERIOD or PERIOD_RATIO.<br>Automatic selection of the measuring method at 5 kHz |
| PHASE | 0.1...5 000 Hz | Reading of a channel pair and comparison of the phase position of the signals |
| INC_ENCODER | 0.1...30 000 Hz | Up/down counter function for the evaluation of encoders |
| FAST_COUNT | 0.1...30 000 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 input signal must be debounced, if necessary! → chapter **Configure the hardware filter** (→ p. 61)

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

## FAST_COUNT

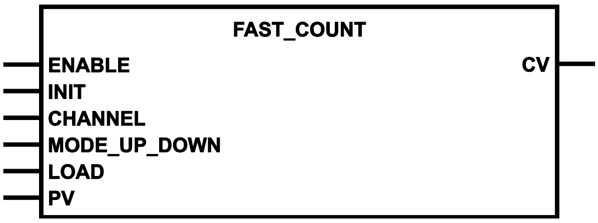
567

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

6830

FAST_COUNT operates as counter block for fast input pulses.

During ENABLE=TRUE the function block detects rising edges at the FRQ input channels.
Maximum input frequency → data sheet.

If resetting and newly setting ENABLE, the counter continues to count from the value that was valid at the last reset of ENABLE.

When setting INIT (rising edge) the count value CV is set to 0.
When resetting the parameter INIT the counter counts from 0.

! Do **not** use this function block on one input together with one of the following function blocks!
- **FREQUENCY** (→ p. 140)
- **FREQUENCY_PERIOD** (→ p. 142)
- **INC_ENCODER** (→ p. 144)
- **PERIOD** (→ p. 146)
- **PERIOD_RATIO** (→ p. 148)
- **PHASE** (→ p. 150)

33949

## ! NOTE

In case of higher frequencies (higher than those guaranteed by **ifm**) the following problems may occur:

- The switch-on and switch-off times of the outputs become more important.
- Undue heating of the components may occur.

The influences mentioned above depend on the components used in the individual case.
These possible influences cannot be exactly predicted.

## Parameters of the inputs

571

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> counter stopped |
| INIT | BOOL | FALSE ⇨ TRUE (edge):<br>unit is initialised<br>FALSE: during further processing of the program |
| CHANNEL | BYTE | number of the fast input channel<br>0...15 for the inputs I00...I15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the inputs I00_E...I15_E |
| MODE_UP_DOWN | BOOL | TRUE: counter counts downwards<br>FALSE: counter counts upwards |
| LOAD | BOOL | TRUE: start value PV is loaded in CV<br>FALSE: function element is not executed |
| PV | DWORD | Start value (preset value) for the counter |

## Parameters of the outputs

28430

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

## FREQUENCY

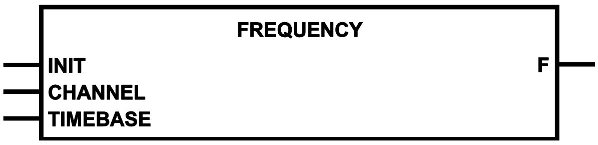
537

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2325
20675

FREQUENCY measures the frequency of the signal arriving at the selected CHANNEL. The FB evaluates the positive edge of the signal.

Depending on the TIMEBASE, frequency measurements can be carried out in a wide value range.
- high frequencies require a short timebase
- low frequencies require a longer timebase

Limit values:

| TIMEBASE | permissible, measurable frequency |
|---|---|
| 57 000 ms<br>(= maximum value) | 1 149 Hz |
| 2 184 ms | 30 000 Hz<br>(= maximum value) |

The longer the timebase for the frequency to be measured, the more precise the measured value determined.
Example of a frequency = 1 Hz:

| TIMEBASE [ms] | max. errors [%] | Measurement [Hz] |
|---|---|---|
| 1 000 | 100 | 0...2 |
| 10 000 | 10 | 0.9...1.1 |

The frequency is provided directly in [Hz].

33957

### NOTE

In case of higher frequencies (higher than those guaranteed by **ifm**) the following problems may occur:

- The switch-on and switch-off times of the outputs become more important.
- Undue heating of the components may occur.

The influences mentioned above depend on the components used in the individual case.
These possible influences cannot be exactly predicted.

7321

For frequency measuring: ensure that the function block does not receive more than 65 535 positive edges within the value of TIMEBASE!
Otherwise, the internal counting register may overflow and lead to incorrect results.

❗Do **not** use this function block on one input together with one of the following function blocks!
- **FAST_COUNT** (→ p. 138)
- **FREQUENCY_PERIOD** (→ p. 142)
- **INC_ENCODER** (→ p. 144)
- **PERIOD** (→ p. 146)
- **PERIOD_RATIO** (→ p. 148)
- **PHASE** (→ p. 150)

## Parameters of the inputs

2599

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (only for 1 cycle):<br>Function block and interface are initialised<br>FALSE: measurement in process<br>or:measurement begins if previously INIT=TRUE |
| CHANNEL | BYTE | number of the fast input channel<br>0...15 for the inputs I00...I15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the inputs I00_E...I15_E |
| TIMEBASE | TIME | Time basis for frequency measurement (max. 57 s) |

## Parameters of the outputs

28425

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

## FREQUENCY_PERIOD

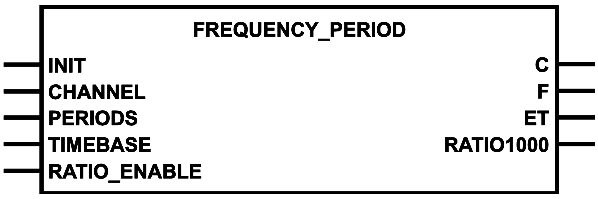
2206

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2335
20676

FREQUENCY_PERIOD measures the frequency and period duration (cycle time) in [µs] on the indicated channel (allowed for all inputs). Maximum input frequency → data sheet.

The FB combines PERIOD/PERIOD_RATIO and FREQUENCY in a common function. The measuring method is automatically selected at approx. 5 kHz:

- below 5.2 kHz the FB behaves like PERIOD or PERIOD_RATIO
- above 5.5 kHz the FB behaves like FREQUENCY.

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.

For an input frequency > 5 kHz and an active FREQUENCY mode the ratio cannot be measured.

The maximum measuring range is approx. 15 min.

33948

## NOTE

In case of higher frequencies (higher than those guaranteed by **ifm**) the following problems may occur:

- The switch-on and switch-off times of the outputs become more important.
- Undue heating of the components may occur.

The influences mentioned above depend on the components used in the individual case.
These possible influences cannot be exactly predicted.

7321

For frequency measuring: ensure that the function block does not receive more than 65 535 positive edges within the value of TIMEBASE!
Otherwise, the internal counting register may overflow and lead to incorrect results.

## NOTE

Do **not** use this function block on one input together with one of the following function blocks!

- **FAST_COUNT** (→ p. 138)
- **FREQUENCY** (→ p. 140)
- **INC_ENCODER** (→ p. 144)
- **PERIOD** (→ p. 146)
- **PERIOD_RATIO** (→ p. 148)
- **PHASE** (→ p. 150)

## Parameters of the inputs

2336

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (only for 1 cycle):<br>Function block and interface are initialised<br>FALSE: measurement in process<br>or:measurement begins if previously INIT=TRUE |
| CHANNEL | BYTE | number of the fast input channel<br>0...15 for the inputs I00...I15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the inputs I00_E...I15_E |
| PERIODS | BYTE | Number of periods to be averaged (1...16)<br>0 : Outputs C and F are not updated<br>> 16 : is limited to 16 |
| TIMEBASE | TIME | Time basis for frequency measurement (max. 57 s) |
| RATIO_ENABLE | BOOL | TRUE: Ratio measurement provided to RATIO1000<br>FALSE: No ratio measurement provided |

## Parameters of the outputs

2337

| Parameter | Data type | Description |
|---|---|---|
| C | DWORD | Cycle time of the detected periods in [µs]<br>permissible = 33...10 000 000 = 0x21...0x989680 |
| F | REAL | frequency of the input signal in [Hz] |
| ET | TIME | for measuring the interval:<br>(can be used for very slow signals)<br>RATIO_ENABLE = TRUE:<br>time elapsed since the last change of edge on the input<br>RATIO_ENABLE = FALSE:<br>time elapsed since the last rising edge on the input<br>for other measurements:<br>ET = 0 |
| RATIO1000 | WORD | Mark-to-space ratio in [‰]<br>permissible = 1...999 = 0x1...0x3E7<br>Preconditions:<br>• for measuring the interval<br>• pulse duration ≥ 100 µs<br>• frequency < 5 kHz |

# INC_ENCODER

525

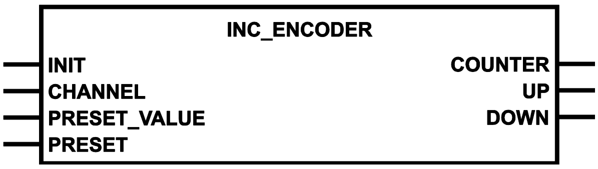
= Incremental Encoder

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_Vxxyyzz.LIB

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

### Symbol in CODESYS:



## Description

2602

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** (→ p. 138)
- **FREQUENCY** (→ p. 140)
- **FREQUENCY_PERIOD** (→ p. 142)
- **PERIOD** (→ p. 146)
- **PERIOD_RATIO** (→ p. 148)
- **PHASE** (→ p. 150)

## Parameters of the inputs

529

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (only for 1 cycle):<br>Function block is initialised<br>FALSE: during further processing of the program |
| CHANNEL | BYTE | Number of the input channel pair<br>0 = channel pair 0 = inputs I00 + I01<br>...<br>3 = channel pair 3 = inputs I06 + I07<br>ℹ For the function block xxx_**E** (if available) applies:<br>0 = channel pair 0 = inputs I00_E + I01_E<br>...<br>3 = channel pair 3 = inputs I06_E + I07_E |
| PRESET_VALUE | DINT | counter start value |
| PRESET | BOOL | FALSE ⇨ TRUE (edge):<br>PRESET_VALUE is loaded to COUNTER<br>TRUE: Counter ignores the input pulses<br>FALSE: Counter counts the input pulses |

## Parameters of the outputs

28429

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

## PERIOD

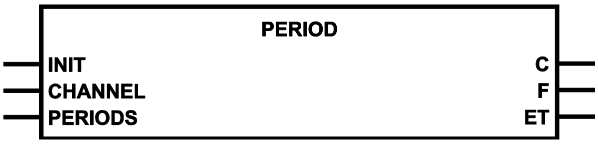
370

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2330
20677

PERIOD measures the frequency and period duration (cycle time) in [µs] on the indicated channel (allowed for all inputs). 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** (→ p. 140). To avoid this, PERIOD can be used. The cycle time is directly indicated in [µs].

The maximum measuring range is 10 seconds.

33956

## NOTE

In case of higher frequencies (higher than those guaranteed by **ifm**) the following problems may occur:

- The switch-on and switch-off times of the outputs become more important.
- Undue heating of the components may occur.

The influences mentioned above depend on the components used in the individual case.
These possible influences cannot be exactly predicted.

Do **not** use this function block on one input together with one of the following function blocks!

- **FAST_COUNT** (→ p. 138)
- **FREQUENCY** (→ p. 140)
- **FREQUENCY_PERIOD** (→ p. 142)
- **INC_ENCODER** (→ p. 144)
- **PERIOD_RATIO** (→ p. 148)
- **PHASE** (→ p. 150)

## Parameters of the inputs

2600

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | FALSE ⇨ TRUE (edge):<br>unit is initialised<br>FALSE: during further processing of the program |
| CHANNEL | BYTE | number of the fast input channel<br>0...15 for the inputs I00...I15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the inputs I00_E...I15_E |
| PERIODS | BYTE | Number of periods to be averaged (1...16)<br>0 : Outputs C and F are not updated<br>> 16 : is limited to 16 |

## Parameters of the outputs

28503

| Parameter | Data type | Description |
|---|---|---|
| C | DWORD | Cycle time of the detected periods in [µs]<br>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<br>(can be used for very slow signals) |

## PERIOD_RATIO

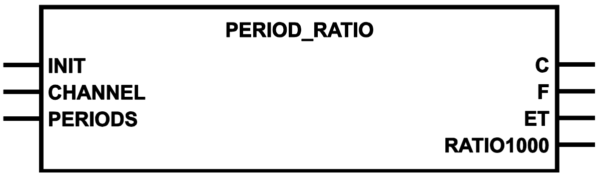
364

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2332
20678

PERIOD_RATIO measures the frequency and the period duration (cycle time) in [µs] over the indicated periods on the indicated channel (allowed for all inputs). 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** (→ p. 140). To avoid this, PERIOD_RATIO can be used. The cycle time is directly indicated in [µs].

The maximum measuring range is 10 seconds.

33952

## NOTE

In case of higher frequencies (higher than those guaranteed by **ifm**) the following problems may occur:

- The switch-on and switch-off times of the outputs become more important.
- Undue heating of the components may occur.

The influences mentioned above depend on the components used in the individual case.
These possible influences cannot be exactly predicted.

Do **not** use this function block on one input together with one of the following function blocks!

- **FAST_COUNT** (→ p. 138)
- **FREQUENCY** (→ p. 140)
- **FREQUENCY_PERIOD** (→ p. 142)
- **INC_ENCODER** (→ p. 144)
- **PERIOD** (→ p. 146)
- **PHASE** (→ p. 150)

## Parameters of the inputs

2601

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | FALSE ⇨ TRUE (edge):<br>unit is initialised<br>FALSE: during further processing of the program |
| CHANNEL | BYTE | number of the fast input channel<br>0...15 for the inputs I00...I15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the inputs I00_E...I15_E |
| PERIODS | BYTE | Number of periods to be averaged (1...16)<br>0 : Outputs C and F are not updated<br>> 16 : is limited to 16 |

## Parameters of the outputs

28502

| Parameter | Data type | Description |
|---|---|---|
| C | DWORD | Cycle time of the detected periods in [µs]<br>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 | Mark-to-space ratio in [‰]<br>permissible = 1...999 = 0x1...0x3E7<br>Preconditions:<br>• for measuring the interval<br>• pulse duration ≥ 100 µs<br>• frequency < 5 kHz |

## PHASE

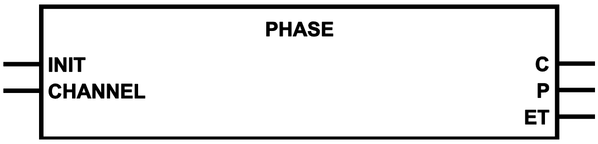
358

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2338
20679

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 (max. 10 seconds).

33843

## NOTE

In case of higher frequencies (higher than those guaranteed by **ifm**) the following problems may occur:

- The switch-on and switch-off times of the outputs become more important.
- Undue heating of the components may occur.

The influences mentioned above depend on the components used in the individual case.
These possible influences cannot be exactly predicted.

Do **not** use this function block on one input together with one of the following function blocks!

- **FAST_COUNT** (→ p. 138)
- **FREQUENCY** (→ p. 140)
- **FREQUENCY_PERIOD** (→ p. 142)
- **INC_ENCODER** (→ p. 144)
- **PERIOD** (→ p. 146)
- **PERIOD_RATIO** (→ p. 148)

## Parameters of the inputs

2339

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (only for 1 cycle):<br>Function block and interface are initialised<br>FALSE: during further processing of the program |
| CHANNEL | BYTE | number of the input channel pair x<br>0 = channel pair 0 = inputs I00 + I01<br>...<br>7 = channel pair 7 = inputs I14 + I15<br>(0...x, value depends on the device, → data sheet)<br>0 = channel pair 0 = inputs I00_E + I01_E<br>...<br>7 = channel pair 7 = inputs I14_E + I15_E |

## Parameters of the outputs

28473

| 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<br>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

**Content**

OUTPUT_BRIDGE ........................................................................................................................153
OUTPUT_CURRENT ..................................................................................................................156
OUTPUT_CURRENT_CONTROL .............................................................................................157
PWM1000 ..................................................................................................................................160

28362

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

## OUTPUT_BRIDGE

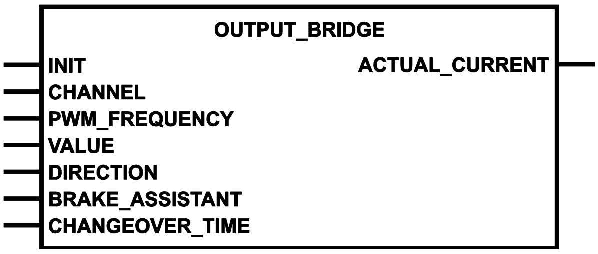
2198

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



### Description

2203

OUTPUT_BRIDGE controls the H-bridges on the PWM channels.

By means of this FB the outputs can easily be used as H-bridge. To do so, two successive output channels are combined to one bridge by means of a minus switching driver. If DIRECTION = FALSE, for the first output the plus switching driver is triggered via a PWM signal and the minus switching driver of the second output is switched.

## NOTE

When using the H-bridge current control is not supported.

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

The function OUT_OVERLOAD_PROTECTION is not active in this mode!
The flag in mode byte will be resetted by function OUTPUT_BRIDGE

If VALUE = 0, output will not be fully deactivated. In principle the output will be active during a timer tick of the PWM timer (typically approx. 50 µs).

► FB must be called in each cycle.

Assignment of the output channels usable as an H-bridge → data sheet.

15672

## NOTE

Is the measuring range for ACTUAL_CURRENT to be changed (to 4 A) at the function block OUTPUT_BRIDGE during operation?

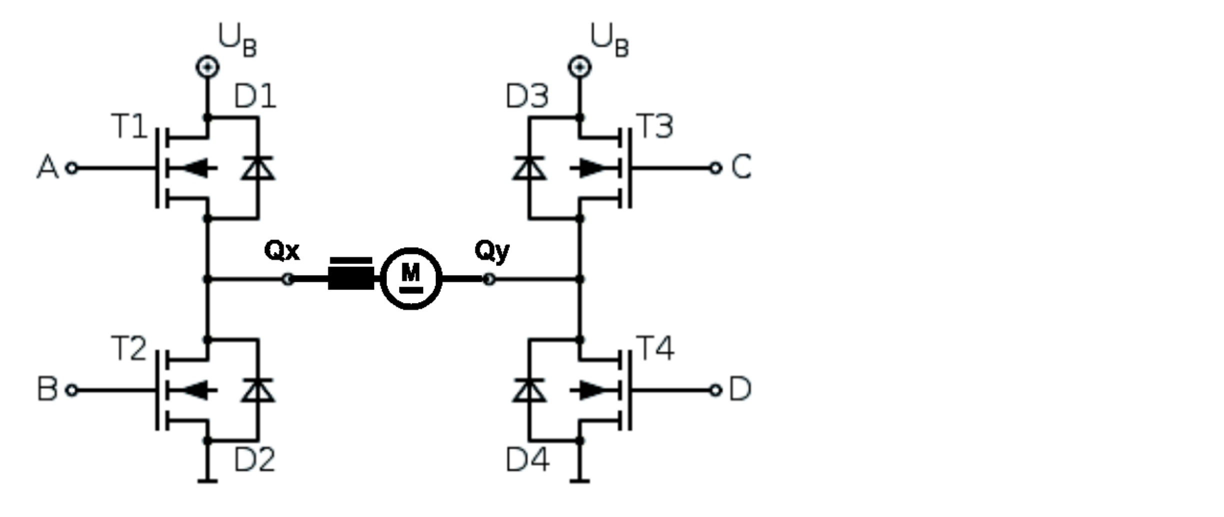
► For both related outputs, during the init phase, call the function block SET_OUTPUT_MODE **before** calling the function block OUTPUT_BRIDGE!
CURRENT_RANGE = 2 (for 4 A)

## Principle of the H-bridge

9990
16411

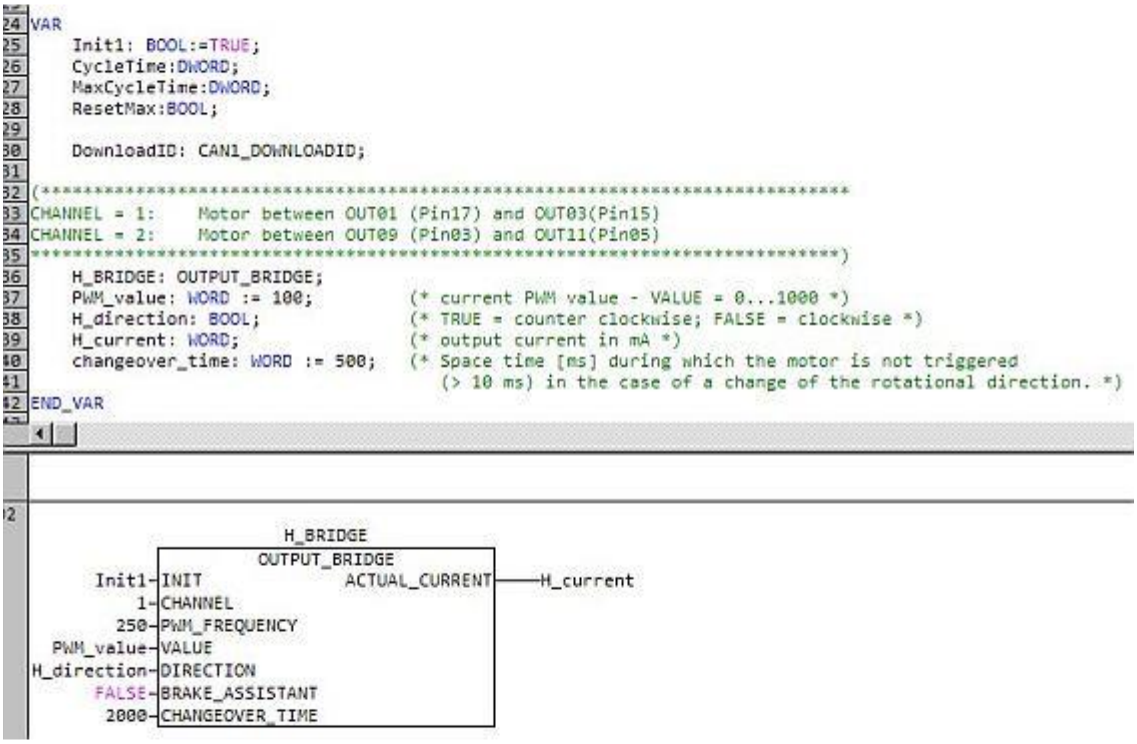
Here you can see how a h-bridge can be run via PWM outputs at a **ifm** controller.

**Basic circuit** of a H-bridge with PWM control:



T1 and T2 together are e.g. output Qx.
Also T3 and T4 belongs together, e.g. for Qy.
Therfore you only need two pins for connecting the DC motor.

**Program example:**



```
VAR
    Init1: BOOL:=TRUE;
    CycleTime:DWORD;
    MaxCycleTime:DWORD;
    ResetMax:BOOL;

    DownloadID: CAN1_DOWNLOADID;

(*****************************************************************************
CHANNEL = 1:    Motor between OUT01 (Pin17) and OUT03(Pin15)
CHANNEL = 2:    Motor between OUT09 (Pin03) and OUT11(Pin05)
*****************************************************************************)
    H_BRIDGE: OUTPUT_BRIDGE;
    PWM_value: WORD := 100;         (* current PWM value - VALUE = 0...1000 *)
    H_direction: BOOL;              (* TRUE = counter clockwise; FALSE = clockwise *)
    H_current: WORD;                (* output current in mA *)
    changeover_time: WORD := 500;   (* Space time [ms] during which the motor is not triggered
                                       (> 10 ms) in the case of a change of the rotational direction. *)
END_VAR
```

```
                      H_BRIDGE
                   OUTPUT_BRIDGE
      Init1-INIT          ACTUAL_CURRENT----H_current
          1-CHANNEL
        250-PWM_FREQUENCY
  PWM_value-VALUE
H_direction-DIRECTION
      FALSE-BRAKE_ASSISTANT
       2000-CHANGEOVER_TIME
```

## Parameters of the inputs

2204

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (only for 1 cycle):<br>Function block is initialised<br>FALSE: during further processing of the program |
| CHANNEL | BYTE | Name of output pair:<br>1 = bridge 1 at Q01 + Q03<br>2 = bridge 2 at Q09 + Q11<br>ℹ For the function block xxx_**E** (if available) applies:<br>1 = bridge 1 at Q01_E + Q03_E<br>2 = bridge 2 at Q09_E + Q11_E |
| PWM_FREQUENCY | WORD | PWM frequency [Hz] for load on output<br>> function block limited to value of 20...2 000 = 0x0014...0x07D0<br>Changes of the PWM frequencies during operation:<br>only permissible in the range 40...2 000 Hz. |
| VALUE | WORD | PWM value (mark-to-space ratio) in [‰]<br>allowed = 0...1 000 = 0x0000...0x03E8<br>Values > 1 000 are regarded as = 1 000 |
| DIRECTION | BOOL | Direction of rotation of the motor:<br>TRUE: Counter-clockwise (ccw):<br>Bridge 1: current flow Q01(_E) ⇦ Q03(_E)<br>Bridge 2: current flow Q09(_E) ⇦ Q11(_E)<br>FALSE: Clockwise (cw):<br>Bridge 1: current flow Q01(_E) ⇨ Q03(_E)<br>Bridge 2: current flow Q09(_E) ⇨ Q11(_E) |
| BRAKE_ASSISTANT | BOOL | TRUE: When changing the rotational direction:<br>the function block switches both outputs to ground to brake the motor<br>as long as CHANGEOVER_TIME is running.<br>FALSE: function element is not executed |
| CHANGEOVER_TIME | WORD | Space time in [ms] during which the motor is not triggered in the case of a change of the rotational direction<br>(≥ cycle time, at least 10 ms)<br>values < 10 ms work as = 10 ms |

## Parameters of the outputs

2205

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

## OUTPUT_CURRENT

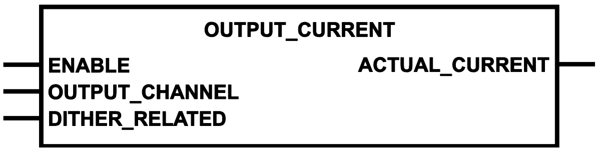
382

Unit type = function block (FB)

Unit is contained in the libraryifm_CR0232_Vxxyyzz.LIB

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

### Symbol in CODESYS:



### Description

28839

OUTPUT_CURRENT handles the current measurement in conjunction with an active PWM channel.

The FB provides the current output current if the outputs are used as PWM outputs or as plus switching. The current measurement is carried out in the device, i.e. no external measuring resistors are required.

### Parameters of the inputs

17894

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| OUTPUT_CHANNEL | BYTE | Number of the current-controlled output channel (0...15)<br>0...15 for the outputs Q00...Q15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the outputs Q00_E...Q15_E |
| DITHER_RELATED | BOOL | Current is determined as an average value via...<br>TRUE: one dither period<br>FALSE: one PWM period |

### Parameters of the outputs

28492

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

## OUTPUT_CURRENT_CONTROL

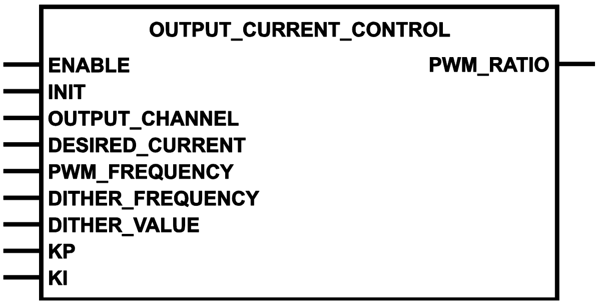
2196

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



## Description

2200

OUTPUT_CURRENT_CONTROL operates as current controller for the PWM outputs.

The setting parameters KI and KP represent the I-component and the P-component of the controller. It is recommended to set KI=50 and KP=50 as start values so as to determine the best setting of the controller. Depending on the requested controller behaviour the values can gradually be incremented (controller is stronger / faster) or decremented (controller is weaker / slower).

At the preset value DESIRED_CURRENT=0 the output is controlled down to 0 mA within about 100 ms with the setting parameters being ignored.

Depending on the controller hardware used, a different teach performance has to be noted.

If parameters are changed during running, than the following can occur:
- the control possibly can skip completely or
- the control can need a longer time to tune the given value.

▶ Therefore validate the measured current and restart the control if necessary.

**NOTE**

► When defining the parameter DITHER_VALUE make sure that the resulting PWM ratio in the operating range of the loop control remains between 0...1000 %:
  • PWM ratio + DITHER_VALUE < 1000 ‰ and
  • PWM ratio - DITHER_VALUE > 0 ‰.
Outside this permissible area, DITHER_VALUE is internally reduced to the maximum possible value temporarily, so that the average value of the PWM ratio corresponds to the required value.

> When the dither is activated, changes to PWM_FREQUENCY, DITHER_VALUE and DITHER_FREQUENCY become effective only when the current dither period has been completed.

► Change the parameters only during operation when INIT=FALSE. The new parameters will not be adopted before the current PWM period has elapsed.

► Changes of the PWM frequencies during operation:
only permissible in the range 40...2 000 Hz.

> If the current indicated in the parameter DESIRED_CURRENT cannot be reached because the PWM ratio is already at 100 %, this is indicated by the system variable ERROR_CONTROL_Qx.

> If KI = 0, there is no loop control.

> If during the loop control a PWM_RATIO = 0 results, the output is not deactivated completely. In principle the output will be active during a timer tick of the PWM timer (typically approx. 50 µs).

► The initialisation of the FB (INIT = TRUE) may only be carried out once per PLC cycle.

► Calling this FB with an output configured as B(L) is not permitted.

► An output defined as PWM output can no longer be used as binary output afterwards.

> If the flowing current in the switched-on condition exceeds the measuring range, control will no longer be possible because the AD converter is at the end of the measuring range and therefore provides wrong values (the max. value).

## Parameters of the inputs

32522

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> control continues with the latest valid parameters |
| INIT | BOOL | TRUE (only for 1 cycle):<br>Function block is initialised<br>FALSE: during further processing of the program |
| OUTPUT_CHANNEL | BYTE | Number of the current-controlled output channel (0...15)<br>0...15 for the outputs Q00...Q15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the outputs Q00_E...Q15_E |
| DESIRED_CURRENT | WORD | Desired current value of the output in [mA]<br>allowed = 0...2 000 / 0...4 000<br>(dependent on output and configuration) |
| PWM_FREQUENCY | WORD | PWM frequency [Hz] for load on output<br>> function block limited to value of 20...2 000 = 0x0014...0x07D0<br>Changes of the PWM frequencies during operation:<br>only permissible in the range 40...2 000 Hz. |
| DITHER_FREQUENCY | WORD | dither frequency in [Hz]<br>value range = 0...FREQUENCY / 2<br>FREQUENCY / DITHER_FREQUENCY must be even-numbered!<br>The FB increases all other values to the next matching value. |
| DITHER_VALUE | WORD | peak-to-peak value of the dither in [‰]<br>permissible values = 0...1 000 = 0000...03E8 |
| KP | BYTE | proportional component of the output signal |
| KI | BYTE | Integral component of the output signal<br>if KI = 0 no rule |

## Parameters of the outputs

2202

| Parameter | Data type | Description |
|---|---|---|
| PWM_RATIO | WORD | For monitoring purposes: display PWM pulse ratio 0...999 % |

## PWM1000

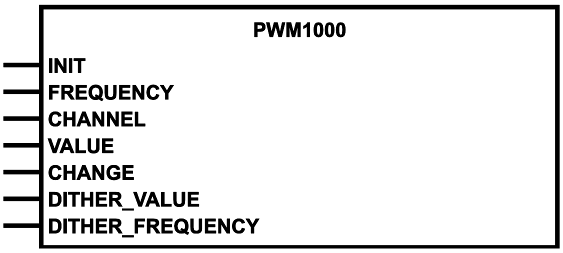
326

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

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

### Symbol in CODESYS:



## Description

2311

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

The FB enables a simple use of the PWM function in the device. For each channel an own PWM frequency, the mark-to-period ratio and the dither can be set.

The PWM frequency FREQUENCY can be directly indicated in [Hz] and the mark-to-period ratio VALUE in steps of 1 ‰.

❗ If VALUE = 0, output will not be fully deactivated. In principle the output will be active during a timer tick of the PWM timer (typically approx. 50 µs).

► When defining the parameter DITHER_VALUE make sure that the resulting PWM ratio in the operating range of the loop control remains between 0...1000 %:
  • PWM ratio + DITHER_VALUE < 1000 ‰ and
  • PWM ratio - DITHER_VALUE > 0 ‰.
  Outside this permissible area, DITHER_VALUE is internally reduced to the maximum possible value temporarily, so that the average value of the PWM ratio corresponds to the required value.

► Activate the function block permanently!

► Calling this FB with an output configured as B(L) is not permitted.

### ❗ NOTE

The function change of a channel defined as PWM function during operation is not possible. The PWM function remains set until a hardware reset is carried out on the controller ⇨ power off and on again.

For high PWM frequencies differences can occur between the set ratio and the ratio on the output due to the system.

► Change the parameters only during operation when INIT=FALSE. The new parameters will not be adopted before the current PWM period has elapsed.

► Changes of the PWM frequencies during operation:
  only permissible in the range 40...2 000 Hz.

**Changes during the runtime:**

Always when the input CHANGE is set to TRUE, the FB adopts the value ...
• 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

## Parameters of the inputs

2312

| Parameter | Data type | Description |
|---|---|---|
| INIT | BOOL | TRUE (only 1 cycle):<br>Function block is initialised<br>Adopting new value from FREQUENCY<br>FALSE: during further processing of the program |
| FREQUENCY | WORD | PWM frequency in [Hz]<br>> function block limited to value of 20...2 000 = 0x0014...0x07D0<br>Changes of the PWM frequencies during operation:<br>only permissible in the range 40...2 000 Hz. |
| CHANNEL | BYTE | Number of the PWM output channel<br>0...15 for the outputs Q00...Q15<br>ℹ For the function block xxx_**E** (if available) applies:<br>0...15 for the outputs Q00_E...Q15_E |
| VALUE | WORD | PWM value (mark-to-space ratio) in [‰]<br>allowed = 0...1 000 = 0x0000...0x03E8<br>Values > 1 000 are regarded as = 1 000 |
| CHANGE | BOOL | TRUE: adoption of the new value of ...<br>• FREQUENCY: after the current PWM period<br>• VALUE: after the current PWM period<br>• DITHER_VALUE: after the current dither period<br>• DITHER_FREQUENCY: after the current dither period<br>FALSE: the changed PWM value has no influence on the output |
| DITHER_VALUE | WORD | peak-to-peak value of the dither in [‰]<br>permissible values = 0...1 000 = 0000...03E8 |
| DITHER_FREQUENCY | WORD | dither frequency in [Hz]<br>value range = 0...FREQUENCY / 2<br>FREQUENCY / DITHER_FREQUENCY must be even-numbered!<br>The FB increases all other values to the next matching value. |

## 5.2.12 Function elements: hydraulic control

**Content**

CONTROL_OCC ....................163
JOYSTICK_0 ....................165
JOYSTICK_1 ....................168
JOYSTICK_2 ....................172
NORM_HYDRAULIC ....................175

13760

The library `ifm_HYDRAULIC_32bit_Vxxyyzz.Lib` contains the following function blocks:

| | |
|---|---|
| **CONTROL_OCC** (→ p. 163) | OCC = **O**utput **C**urrent **C**ontrol<br>Scales the input value [WORD] to an indicated current range |
| **JOYSTICK_0** (→ p. 165) | Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0... 1000 |
| **JOYSTICK_1** (→ p. 168) | Scales signals [INT] from a joystick D standardised to 0... 1000 |
| **JOYSTICK_2** (→ p. 172) | Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation |
| **NORM_HYDRAULIC** (→ p. 175) | Normalises a value [DINT] within defined limits to a value with new limits |

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

- RAMP_INT
- CHARCURVE

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

The following function blocks are needed from the library:`ifm_CR0232_Vxxyyzz.LIB`

| | |
|---|---|
| **OUTPUT_CURRENT** (→ p. 156) | Measures the current (average via dither period) on an output channel |
| **OUTPUT_CURRENT_CONTROL** (→ p. 157) | Current controller for a PWMi output channel |

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

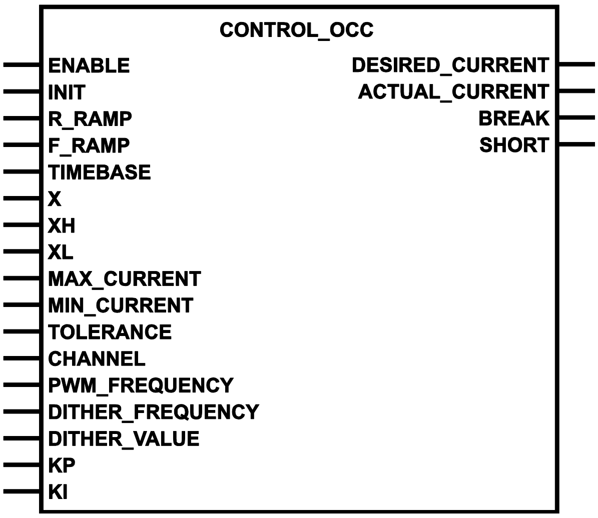
## CONTROL_OCC

2735

Unit type = function block (FB)

Unit is contained in the library`ifm_HYDRAULIC_32bit_Vxxyyzz.Lib`

### Symbol in CODESYS:



### Description

2737

CONTROL_OCC scales the input value X to a specified current range.

Each instance of the FB is called once in each PLC cycle.
This function block uses the following function blocks from the library:`ifm_CR0232_Vxxyyzz.LIB`

- **OUTPUT_CURRENT_CONTROL** (→ p. 157)
- **OUTPUT_CURRENT** (→ p. 156)

The controller controls on the basis of the cycle period of the PWM signal.

The setting parameters KI and KP represent the **i**ntegral and the **p**roportional component of the controller. It is recommended to set KI=50 and KP=50 as start values so as to determine the best setting of the controller.

► Increasing the values for KI and / or KP: ⇨ controller becomes more sensitive / faster
Decreasing the values for KI and / or KP: ⇨ controller becomes less sensitive / slower

> At the output DESIRED_CURRENT=0 the output is **immediately** switched to 0 mA and is **not** adjusted downward to 0 mA in accordance with the set parameters.

The controller has a fast compensation mechanism for voltage drops of the supply voltage. In addition to the controller behaviour of the controller and on the basis of the voltage drop, the ratio of the PWM is increased such that the controller reaches as quickly as possible the desired value.

ℹ The input X of CONTROL_OCC should be supplied by the output of the JOYSTICK FBs.

## Parameters of the inputs

2739

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| INIT | BOOL | FALSE ⇨ TRUE (edge):<br>unit is initialised<br>FALSE: during further processing of the program |
| R_RAMP | INT | Rising edge of the ramp in [increments/PLC cycle]<br>0 = no ramp |
| F_RAMP | INT | Falling edge of the ramp in [increments/PLC cycle]<br>0 = no ramp |
| TIMEBASE | TIME | Reference for rising and falling edge of the ramp:<br>t#0s = rising/falling edge in [increments/PLC cycle]<br>! Fast controllers have very short cycle times!<br>otherwise = rising/falling edge in [increments/TIMEBASE] |
| X | WORD | input value |
| XH | WORD | Upper limit of input value range [increments] |
| XL | WORD | Lower limit of input value range [increments] |
| MAX_CURRENT | WORD | Max. valve current in [mA] |
| MIN_CURRENT | WORD | Min. valve current in [mA] |
| TOLERANCE | BYTE | Tolerance for min. valve current in [increments]<br>When the tolerance is exceeded, jump to MIN_CURRENT is effected |
| CHANNEL | BYTE | Number of the current-controlled output channel<br>0...15 for the outputs Q00...Q15<br>ⓘ For the function block xxx_**E** (if available) applies:<br>0...15 for the outputs Q00_E...Q15_E |
| PWM_FREQUENCY | WORD | PWM frequency [Hz] for load on input |
| DITHER_FREQUENCY | WORD | dither frequency in [Hz]<br>value range = 0...FREQUENCY / 2<br>FREQUENCY / DITHER_FREQUENCY must be even-numbered!<br>The FB increases all other values to the next matching value. |
| DITHER_VALUE | BYTE | peak-to-peak value of the dither in [%]<br>permissible values = 0...100 = 0x00...0x64 |
| KP | BYTE | proportional component of the output signal |
| KI | BYTE | integral component of the output signal |

ⓘ For KP, KI applies: recommended start value = 50

## Parameters of the outputs

27757

| Parameter | Data type | Description |
|---|---|---|
| DESIRED_CURRENT | WORD | Desired current value in [mA] for OCC<br>(for monitoring purposes) |
| ACTUAL_CURRENT | WORD | Output current in [mA] |
| BREAK | BOOL | Error: cable interrupted at output |
| SHORT | BOOL | Error: short circuit in cable at output |

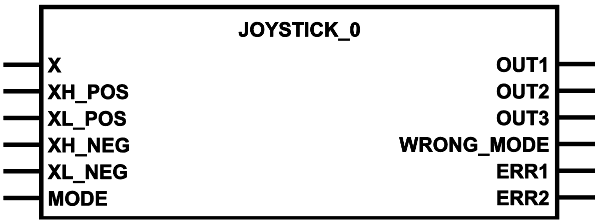
## JOYSTICK_0

6250

Unit type = function block (FB)

Unit is contained in the library`ifm_hydraulic_32bit_Vxxyyzz.Lib`

### Symbol in CODESYS:



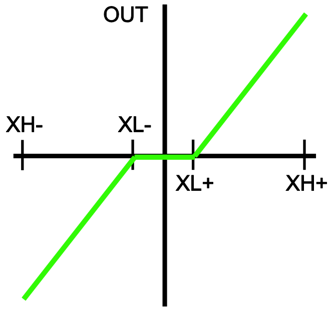
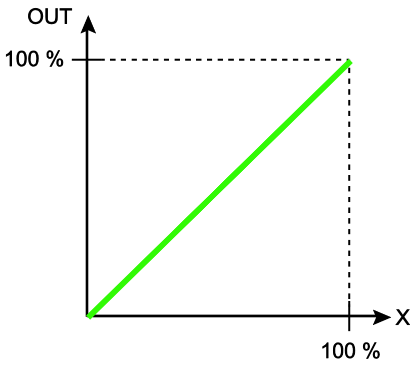
### Description

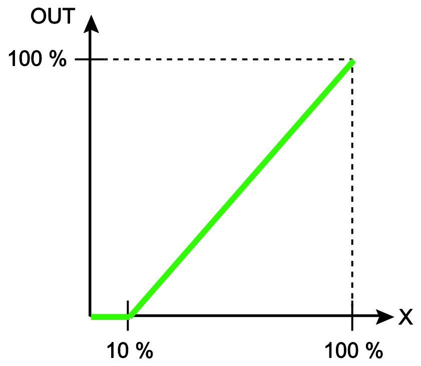
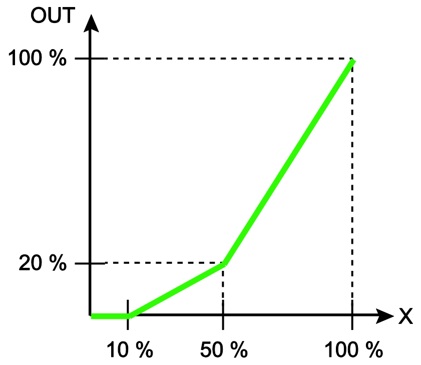
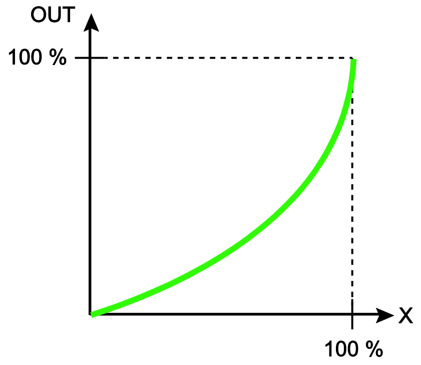
27776

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.<br>The values for the positive and negative area may be different.<br>The values for XL_NEG and XH_NEG are negative here. |  |
| Mode 0:<br>characteristic curve linear for the range XL to XH |  |

| | |
|---|---|
| Mode 1:<br>Characteristic curve linear with dead band<br>Values fixed to:<br>Dead band:<br>0…10% of 1000 increments |  |
| Mode 2:<br>2-step linear characteristic curve with dead band<br>Values fixed to:<br>Dead band:<br>0…10% of 1000 increments<br>Step:<br>X = 50 % of 1000 increments<br>Y = 20 % of 1000 increments |  |
| Characteristic curve mode 3:<br>Curve rising (line is fixed) |  |

## Parameters of the inputs

27685

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

## Parameters of the outputs

27750

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

Possible results for ERR1 and ERR2:

| Value<br>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 |

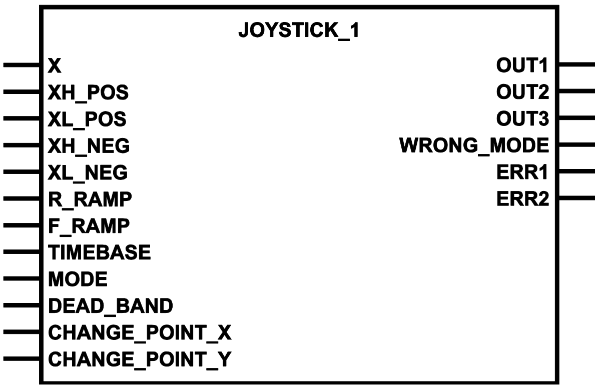
## JOYSTICK_1

6255

Unit type = function block (FB)

Unit is contained in the library`ifm_hydraulic_32bit_Vxxyyzz.Lib`

### Symbol in CODESYS:

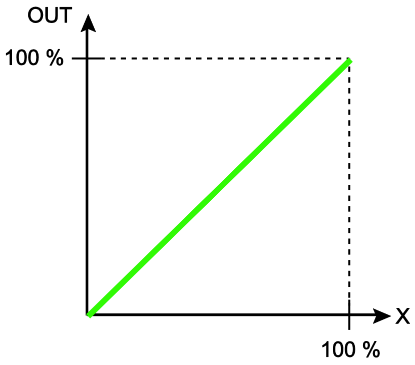
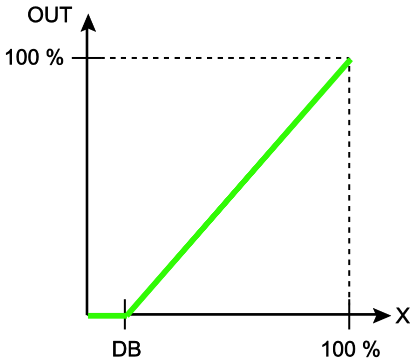


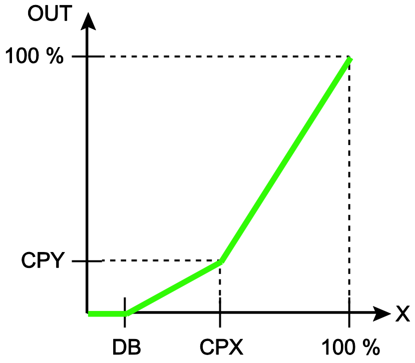
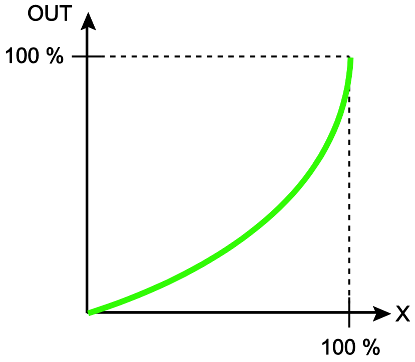
### Description

27775

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

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

## Parameters of the inputs

27650

| Parameter | Data type | Description |
|---|---|---|
| X | INT | Input value [increments] |
| XH_POS | INT | Max. preset value positive direction [increments]<br>(negative values also permissible) |
| XL_POS | INT | Min. preset value positive direction [increments]<br>(negative values also permissible) |
| XH_NEG | INT | Max. preset value negative direction [increments]<br>(negative values also permissible) |
| XL_NEG | INT | Min. preset value negative direction [increments]<br>(negative values also permissible) |
| R_RAMP | INT | Rising edge of the ramp in [increments/PLC cycle]<br>0 = no ramp |
| F_RAMP | INT | Falling edge of the ramp in [increments/PLC cycle]<br>0 = no ramp |
| TIMEBASE | TIME | Reference for rising and falling edge of the ramp:<br>t#0s = rising/falling edge in [increments/PLC cycle]<br>❗ Fast controllers have very short cycle times!<br>otherwise = rising/falling edge in [increments/TIMEBASE] |
| MODE | BYTE | Mode selection characteristic curve:<br>0 = linear<br>(X\|OUT = 0\|0 ... 1000\|1000)<br>1 = linear with dead band<br>(X\|OUT = 0\|0 ... DB\|0 ... 1000\|1000)<br>2 = 2-step linear with dead band<br>(X\|OUT = 0\|0 ... DB\|0 ... CPX\|CPY ... 1000\|1000)<br>3 = curve rising (line is fixed) |
| DEAD_BAND | BYTE | Adjustable dead band<br>in [% of 1000 increments] |
| CHANGE_POINT_X | BYTE | For mode 2: ramp step, value for X<br>in [% of 1000 increments] |
| CHANGE_POINT_Y | BYTE | For mode 2: ramp step, value for Y<br>in [% of 1000 increments] |

## Parameters of the outputs

27750

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

Possible results for ERR1 and ERR2:

| Value<br>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 |

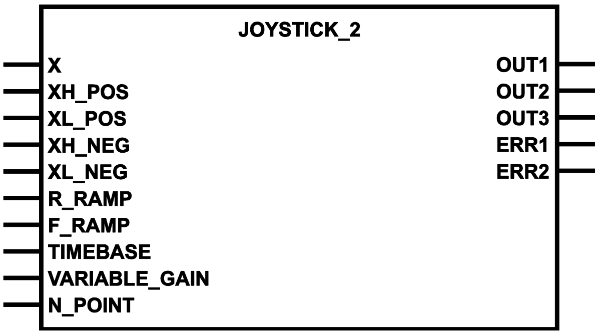
## JOYSTICK_2

6258

Unit type = function block (FB)

Unit is contained in the library`ifm_hydraulic_32bit_Vxxyyzz.Lib`

### Symbol in CODESYS:

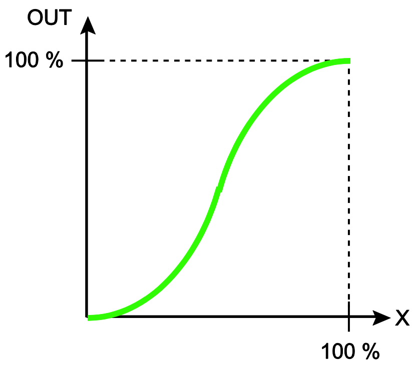


### Description

27778

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

27739

| Parameter | Data type | Description |
|---|---|---|
| X | INT | Input value [increments] |
| XH_POS | INT | Max. preset value positive direction [increments]<br>(negative values also permissible) |
| XL_POS | INT | Min. preset value positive direction [increments]<br>(negative values also permissible) |
| XH_NEG | INT | Max. preset value negative direction [increments]<br>(negative values also permissible) |
| XL_NEG | INT | Min. preset value negative direction [increments]<br>(negative values also permissible) |
| R_RAMP | INT | Rising edge of the ramp in [increments/PLC cycle]<br>0 = no ramp |
| F_RAMP | INT | Falling edge of the ramp in [increments/PLC cycle]<br>0 = no ramp |
| TIMEBASE | TIME | Reference for rising and falling edge of the ramp:<br>t#0s = rising/falling edge in [increments/PLC cycle]<br>❗ Fast controllers have very short cycle times!<br>otherwise = rising/falling edge in [increments/TIMEBASE] |
| VARIABLE_GAIN | ARRAY [0..10] OF POINT | Pairs of values describing the curve<br>The first pairs of values indicated in N_POINT are used. n = 2…11<br>**Example:** 9 pairs of values declared as variable VALUES:<br>`VALUES : ARRAY [0..10] OF POINT :=`<br>`(X:=0,Y:=0),(X:=200,Y:=0), (X:=300,Y:=50),`<br>`(X:=400,Y:=100), (X:=700,Y:=500),`<br>`(X:=1000,Y:=900), (X:=1100,Y:=950),`<br>`(X:=1200,Y:=1000), (X:=1400,Y:=1050);`<br>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

27731

| Parameter | Data type | Description |
|---|---|---|
| OUT1 | WORD | Standardised output value: 0…1000 increments<br>e.g. for valve left |
| OUT2 | WORD | Standardised output value: 0…1000 increments<br>e.g. for valve right |
| OUT3 | INT | Standardised output value -1000…0…1000 increments<br>e.g. for valve on output module (e.g. CR2011 or CR2031) |
| ERR1 | BYTE | Error code for rising edge<br>(referred to the internally used function blocks CHARCURVE and RAMP_INT from `util.lib`)<br>(possible messages → following table) |
| ERR2 | BYTE | Error code for falling edge<br>(referred to the internally used function blocks CHARCURVE and RAMP_INT from `util.lib`)<br>(possible messages → following table) |

Possible results for ERR1 and ERR2:

| Value<br>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 |

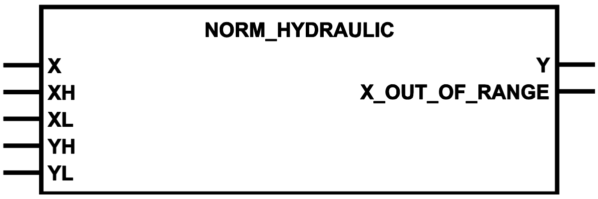
## NORM_HYDRAULIC

394

Unit type = function block (FB)

Unit is contained in the libraryifm_hydraulic_32bit_Vxxyyzz.Lib

### Symbol in CODESYS:



### Description

27771

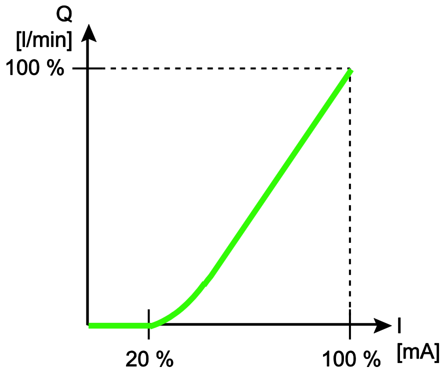
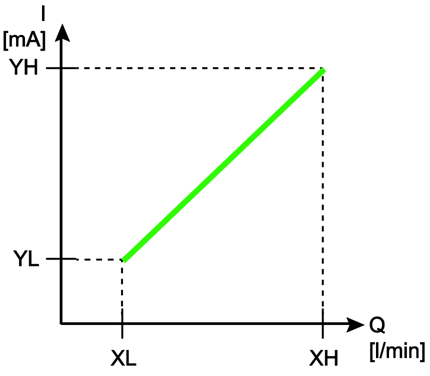
NORM_HYDRAULIC standardises input values with fixed limits to values with new limits.

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:<br>The oil flow will not start before 20% of the coil current has been reached.<br>At first the oil flow is not linear. |  |
| Characteristics of the function block |  |

## Parameters of the inputs

27682

| 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.:<br>valve current [mA] / flow [l/min] |
| YL | DINT | Min. output value [increments], e.g.:<br>valve current [mA], flow [l/min] |

## Parameters of the outputs

27741

| 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

27881

| 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

**Content**

Setting rule for a controller ....177
DELAY ....178
PID1 ....179
PID2 ....181
PT1 ....183

27839

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

28400

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

28405

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

27829

To dampen overshoot **PT1** (→ p. 183) (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.

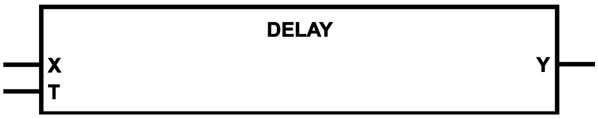
## DELAY

27826

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27770

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

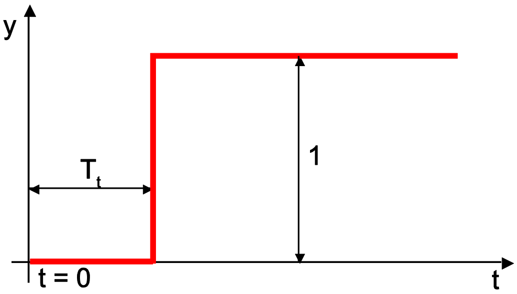


Figure: Time characteristics of DELAY

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

2615

| Parameter | Data type | Description |
|---|---|---|
| X | REAL | Input value |
| T | TIME | Delay time (dead time)<br>allowed: 0...100 • cycle time |

### Parameters of the outputs

2616

| Parameter | Data type | Description |
|---|---|---|
| Y | REAL | Input value, delayed by the time T |

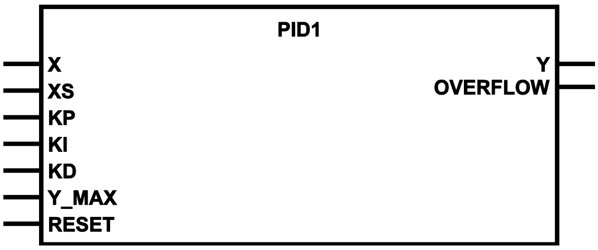
## PID1

19235

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

19237

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.

OVERFLOW = TRUE is signalled when the 'I' part reaches an internal limitation because a control deviation could not be corrected.
OVERFLOW remains TRUE as long as the limitation is active.

### Parameters of the inputs

19238

| Parameter | Data type | Description |
|---|---|---|
| X | REAL | Input value |
| XS | REAL | preset value |
| KP | REAL | Proportional component of the output signal<br>(only positive values permissible) |
| KI | REAL | Integral component of the output signal<br>(only positive values permissible) |
| KD | REAL | Differential component of the output signal<br>(only positive values permissible) |
| Y_MAX | REAL | maximum control value |
| RESET | BOOL | TRUE: reset the function element<br>FALSE: function element is not executed |

### Parameters of the outputs

19241

| Parameter | Data type | Description |
|---|---|---|
| Y | REAL | Output value |
| OVERFLOW | BOOL | TRUE: Overflow of the data buffer ⇨ loss of data!<br>FALSE: Data buffer is without data loss |

## Recommended settings

19242

- ► Start values:
  KP = 0
  KD = 0
- ► Adapt KI to the process.
- ► Then modify KP and KI gradually.

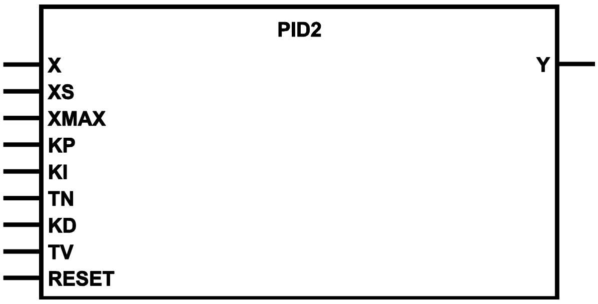
## PID2

344

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

6262

PID2 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 TV the manipulated variable returns to the value corresponding to the proportional component and changes in accordance with the reset time TN.

The manipulated variable Y is already standardised to **PWM1000** (→ p. 160).

**Rules:**

- Negative values for KP, KI and KD are not permitted.
- In case of TN = 0, the I value is not calculated
- In case of XS > XMAX, XS is limited to XMAX.
- In case of X > XMAX, Y is set to 0.
- 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.

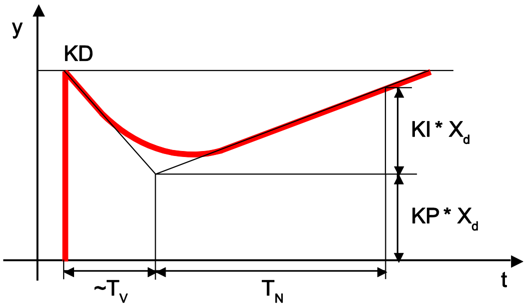


Figure: Typical step response of a PID controller

## Parameters of the inputs

12963

| Parameter | Data type | Description |
|---|---|---|
| X | WORD | input value |
| XS | WORD | preset value |
| XMAX | WORD | maximum preset value |
| KP | REAL | Proportional component of the output signal<br>(only positive values permissible) |
| KI | REAL | Integral component of the output signal<br>(only positive values permissible) |
| TN | TIME | integral action time (integral component) |
| KD | REAL | Differential component of the output signal<br>(only positive values permissible) |
| TV | TIME | derivative action time (differential component) |
| RESET | BOOL | TRUE: reset the function element<br>FALSE: function element is not executed |

## Parameters of the outputs

27743

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

## Recommended setting

27708

► 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.

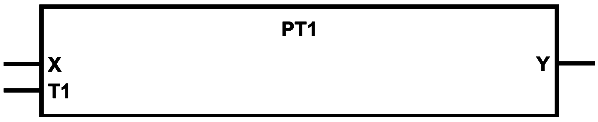
# PT1

28577

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



## Description

27819

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):

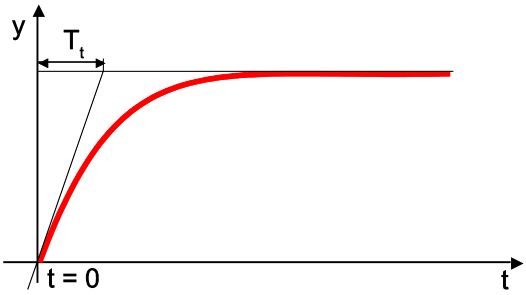


Figure: Time characteristics of PT1

## Parameters of the inputs

2618

| Parameter | Data type | Description |
|---|---|---|
| X | DINT | current input value |
| T1 | TIME | Delay time (time constant) |

## Parameters of the outputs

2619

| Parameter | Data type | Description |
|---|---|---|
| Y | DINT | output value |

## 5.2.14 Function elements: software reset

**Content**

SOFTREESET ....................................................................................................185

27843

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

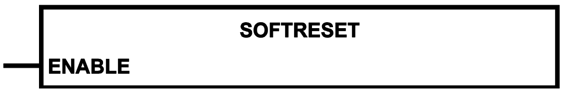
## SOFTRESET

27714

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

28134

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

27689

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

### 5.2.15 Function elements: measuring / setting of time

**Content**

TIMER_READ ....................................................................................................................187
TIMER_READ_US ..............................................................................................................188

28375

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

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

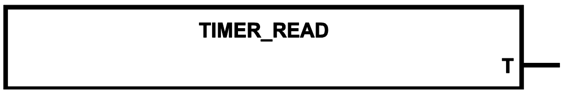
## TIMER_READ

27719

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27827

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

27740

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

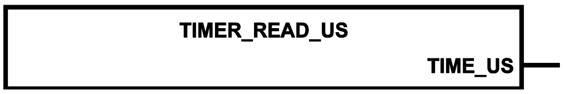
## TIMER_READ_US

27716

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

**Symbol in CODESYS:**



### Description

27793

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

27758

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

### 5.2.16 Function elements: device temperature

**Content**

TEMPERATURE ....................................................................................................................190

2364

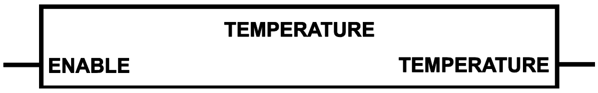
## TEMPERATURE

2216

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2365

TEMPERATURE reads the current temperature in the device.

The FB can be called cyclically and indicates the current device temperature (-40...125 °C) on its output.

### Parameters of the inputs

2366

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

### Parameters of the outputs

2367

| Parameter | Data type | Description |
|---|---|---|
| TEMPERATURE | INT | Current internal temperature of the device [°C] |

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

**Content**

Storage types for data backup ....191
File system....192
Automatic data backup....193
Manual data storage....195

28364

### Storage types for data backup

28421

The device provides the following memory types:

#### Flash memory

28829

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
- saving data with FLASHWRITE
- reading data with FLASHREAD

#### FRAM memory

28831

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
- saving data with FRAMWRITE
- reading data with FRAMREAD

## File system

2690

The file system coordinates the storage of the information in the memory. The size of the file system is 128 kbytes.

The file names of the data system are limited:
max. length for Controller: CR0n3n, CR7n3n: 15 characters
max. for all other units: 11 characters

**Behaviour of the file system in the** Controller: CR0n3n, CR7n3n**:**

- The controller always tries to write the file, even if the same file name already exists. The file might be saved several times. Only the current file is used. Via the download (see below) this multiple filing can be prevented.
- Individual files cannot be overwritten or deleted.
- The file system is completely deleted during each download (boot project download or RAM download). Then e.g. a symbol file or a project file (FBs in CODESYS) can be written.
- The file system is also deleted during a [Reset (Original)] (CODESYS function in the menu [Online]).

## Automatic data backup

**Content**

MEMORY_RETAIN_PARAM ........................................................................................................194

14168
32645

The **ecomat*mobile*** controllers allow to save data (BOOL, BYTE, WORD, DWORD) non-volatilely (= saved in case of voltage failure) in the memory. If the supply voltage drops, the backup operation is automatically started. Therefore it is necessary that the data is defined as RETAIN variables (→ CODESYS).

A distinction is made between variables declared as RETAIN and variables in the flag area which can be configured as a remanent block with **MEMORY_RETAIN_PARAM** (→ p. 194).
Details → chapter **Variables** (→ p. 66)

The advantage of the automatic backup is that also in case of a sudden voltage drop or an interruption of the supply voltage, the storage operation is triggered and thus the current values of the data are saved (e.g. counter values).

! If supply voltage < 8 V, retain data is no longer backed up!
In this case, flag RETAIN_WARNING = TRUE.

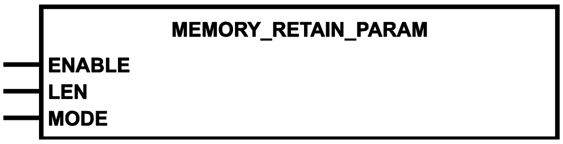
## MEMORY_RETAIN_PARAM

27673

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

28023

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

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

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

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

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

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

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

### Parameters of the inputs

27645

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

## Manual data storage

**Content**

FLASHREAD ... 196
FLASHWRITE ... 197
FRAMREAD ... 199
FRAMWRITE ... 200
MEMCPY ... 201
MEMSET ... 202

28519

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.

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

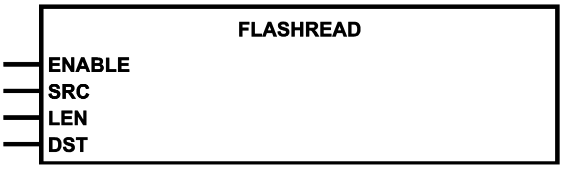
## FLASHREAD

27888

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27769

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 POU!

### Parameters of the inputs

2318

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| SRC | DWORD | relative start address in memory<br>allowed = 0...65 535 = 0x0...0x0000 FFFF<br>❗ If start address is outside the permissible range:<br>> no data transfer |
| LEN | DWORD | number of data bytes (max. 65 536 = 0x0001 0000)<br>❗ If the indicated number of bytes exceeded the flash memory space, the data would only be transmitted to the end of the flash memory space. |
| DST | DWORD | destination address<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |

►

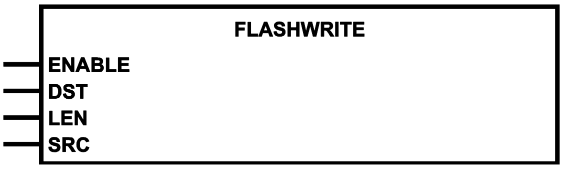
## FLASHWRITE

27885

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

19245

► Activate the TEST input to use the function block! Otherwise, a watchdog error occurs.

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

32442

**⚠ 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 once!

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

> 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 destination start address DST is outside the permissible range: no data transfer!

## Parameters of the inputs

2603

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| DST | DWORD | Relative start address in memory<br>allowed = 0...65 535 = 0x0...0x0000 FFFF<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |
| LEN | DWORD | number of data bytes (max. 65 536 = 0x0001 0000)<br>❗ If the indicated number of bytes exceeded the flash memory space, the data would only be transmitted to the end of the flash memory space. |
| SRC | DWORD | source address |

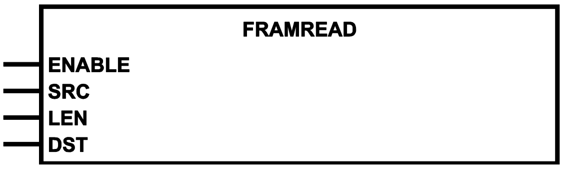
## FRAMREAD

27886

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

28176

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 POU!

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

### Parameters of the inputs

2606

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| SRC | DWORD | relative start address in memory<br>allowed = 0... 16 383 = 0x0000 0000...0x0000 3FFF |
| LEN | DWORD | number of data bytes<br>allowed = 0...16 384 = 0x0000 0000...0x0000 4000 |
| DST | DWORD | destination address<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |

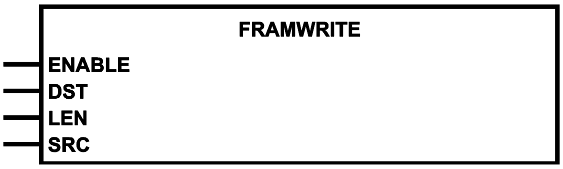
## FRAMWRITE

27863

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

28167

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 POU!

❗ 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

2605

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| DST | DWORD | Relative start address in memory<br>allowed = 0... 16 383 = 0x0...0x0000 3FFF |
| LEN | DWORD | number of data bytes<br>allowed = 0...16 384 = 0x0000 0000...0x0000 4000 |
| SRC | DWORD | start address in source memory<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |

## MEMCPY

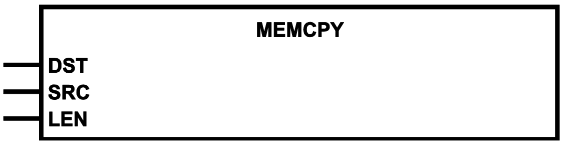
27672

= memory copy

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

15944
32818

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 POU!

> 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.

> If the memory area into which the data are to be copied is not entirely in a permissible memory area, the data will not be copied and a parameter error will be signalled.

| DST memory area | Device | Memory size |
|---|---|---|
| Application data | (all) | 192 Kbytes |

Tables "Available memory" → chapter **Available memory** (→ p. 16)

### Parameters of the inputs

27680

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

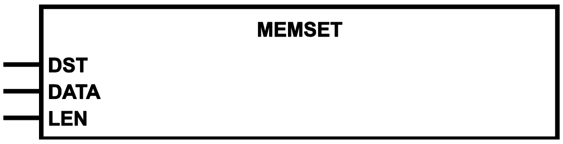
## MEMSET

2348

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

2350

MEMSET enables writing to a defined data area.

The FB writes the content of DATA into the memory as from the address of DST as many bytes as indicated under LEN.

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

> If the memory area into which the data are to be copied is not entirely in a permissible memory area, the data will not be copied and a parameter error will be signalled.

| DST memory area | Device | Memory size |
|---|---|---|
| Application data | (all) | 192 Kbytes |

### Parameters of the inputs

2351

| Parameter | Data type | Description |
|---|---|---|
| DST | DWORD | destination address<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |
| DATA | BYTE | Value to be written |
| LEN | WORD | number of data bytes to be overwritten with DATA |

## 5.2.18 Function elements: data access and data check

**Content**

CHECK_DATA ....................204
GET_IDENTITY ....................206
GET_IDENTITY_EIOS ....................207
SET_DEBUG ....................208
SET_IDENTITY ....................209
SET_PASSWORD ....................210

27844

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

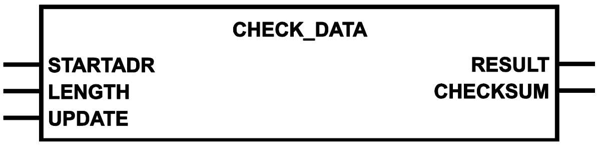
## CHECK_DATA

27809

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27768

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 POU!

► 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

2612

| Parameter | Data type | Description |
|---|---|---|
| STARTADR | DWORD | Start address of the monitored data memory<br>(WORD address as from %MW0)<br>❗ Determine the address by means of the operator ADR and assigne it to the POU! |
| LENGTH | DWORD | length of the monitored data memory in [byte] |
| UPDATE | BOOL | TRUE: Data was changed<br>> function block calculates new checksum<br>FALSE: Data was not changed<br>> function block checks memory area |

### Parameters of the outputs

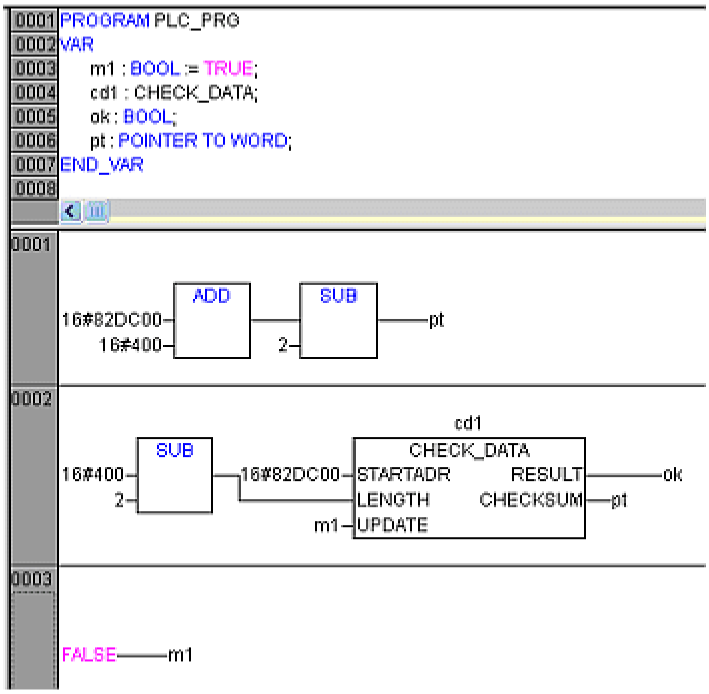
2613

| Parameter | Data type | Description |
|---|---|---|
| RESULT | BOOL | TRUE: CRC checksum OK:<br>intentional data change or no change<br>FALSE: CRC checksum faulty:<br>data was changed inadvertently |
| CHECKSUM | DWORD | Current CRC checksum |

## Example: CHECK_DATA

27893

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
```

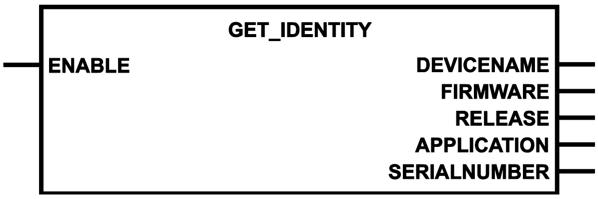
# GET_IDENTITY

19287

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0232_Vxxyyzz.LIB`
New output SERIALNUMBER exist in:
- CR0032 from RTS V02.01.06
- CR0033 from RTS V01.00.09
- CR0133 from RTS V01.00.09
- CR0232 from RTS V01.00.03
- CR0233 from RTS V01.00.09

### Symbol in CODESYS:



## Description

19288

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** (→ p. 209))
- serial number of the device

## Parameters of the inputs

28478

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

## Parameters of the outputs

19289

| Parameter | Data type | Description |
|---|---|---|
| DEVICENAME | STRING(31) | hardware name<br>as a string of max. 31 characters, e.g.: "CR0403" |
| FIRMWARE | STRING(31) | Name of the runtime system in the device<br>as character string of max. 31 characters<br>e.g.: "CR0403" |
| RELEASE | STRING(31) | software version<br>as a character string of max. 31 characters |
| APPLICATION | STRING(79) | Name of the application<br>as a string of max. 79 characters<br>e.g.: "Crane1704" |
| SERIALNUMBER | STRING(31) | Serial number of the device<br>as character string of max. 31 characters<br>e.g.: "12345678" |

## GET_IDENTITY_EIOS

19247

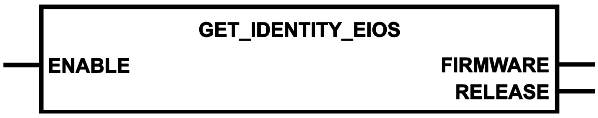
EIOS = Extended IO System = runtime system of the extended side

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`
Function block exist in:

- CR0232 from RTS V01.00.03
- CR0233 from RTS V01.00.09
- CR0234
- CR0235

### Symbol in CODESYS:



### Description

19249

GET_IDENTITY_EIOS reads the specific identifications for the extended side stored in the device:

- name of the extended IO system (EIOS) in the device
- version and revision no. of the extended IO system (EIOS) in the device

### Parameters of the inputs

19250

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

### Parameters of the outputs

19251

| Parameter | Data type | Description |
|---|---|---|
| FIRMWARE | STRING(31) | Name of the extended IO system (EIOS) in the device as character string of max. 31 characters |
| RELEASE | STRING(31) | software version of the extended IO system (EIOS) in the device as a character string of max. 31 characters |

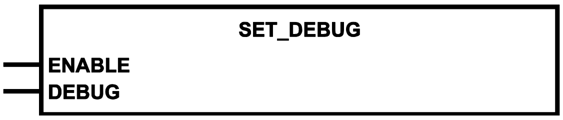
## SET_DEBUG

27721

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27815

SET_DEBUG handles the DEBUG mode without active test input (→ chapter **TEST mode** (→ p. 49)).

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 some special 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

27688

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

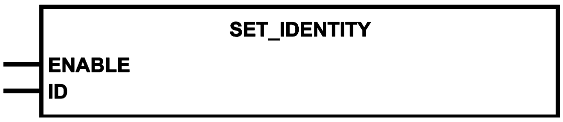
## SET_IDENTITY

11927

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



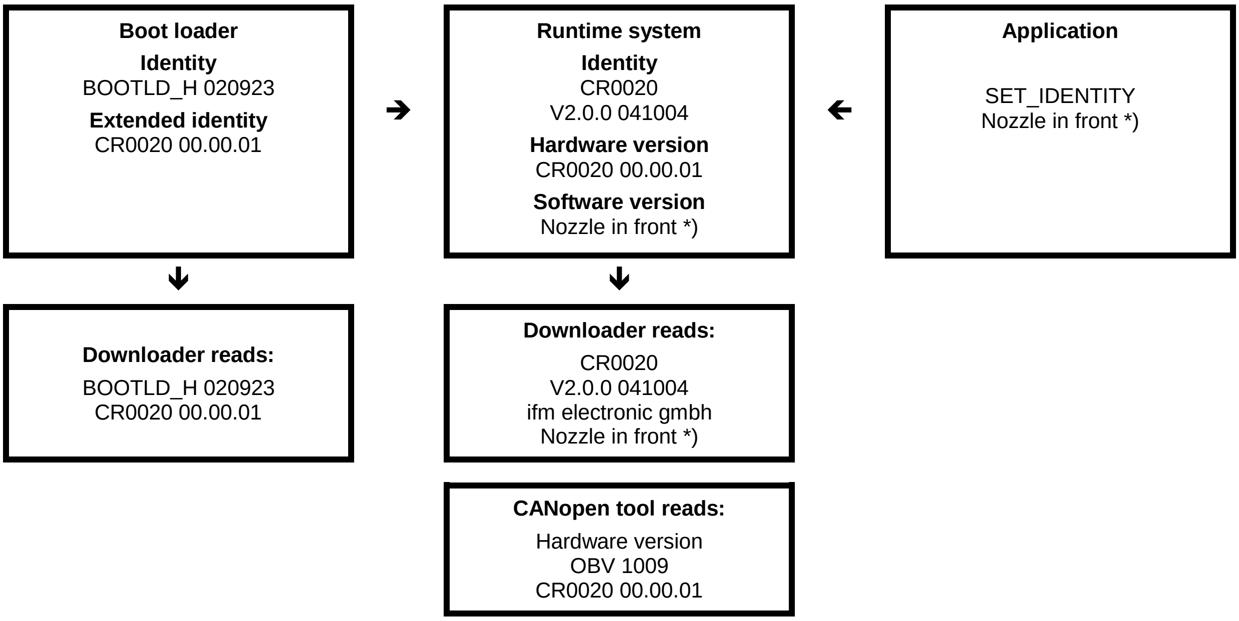
### Description

27814

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

11928

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| ID | STRING(79) | Any desired text with a maximum length of 79 characters |

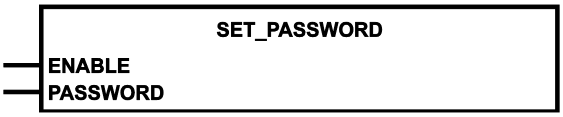
## SET_PASSWORD

27724

Unit type = function block (FB)

Unit is contained in the library`ifm_CR0232_Vxxyyzz.LIB`

### Symbol in CODESYS:



### Description

27816

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 as boot project.

### Parameters of the inputs

2353

| Parameter | Data type | Description |
|---|---|---|
| ENABLE | BOOL | FALSE ⇨ TRUE (edge):<br>Initialise block (only 1 cycle)<br>> Read block inputs<br>TRUE: execute this function element<br>FALSE: unit is not executed<br>> Function block inputs are not active<br>> Function block outputs are not specified |
| PASSWORD | STRING(16) | password<br>If PASSWORD = "", than access is possible without enter of a password |

# 6 Diagnosis and error handling

**Content**

Diagnosis ........................................................ 211
Fault ........................................................ 211
Reaction in case of an error ........................................................ 212
Relay: important notes! ........................................................ 212
Response to system errors ........................................................ 212
CAN / CANopen: errors and error handling ........................................................ 213

28855

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*** (→ p. 45)

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

## 6.1 Diagnosis

28856

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

28834

A fault is the state of an item characterized by the inability to perform the requested function, excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources.
A fault is often the result of a failure of the item itself, but may exist without prior failure.
In →ISO 13849-1 "fault" means "random fault".

## 6.3 Reaction in case of an error

19603
28543

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

> the operation LED lights red,
> the output relays switch off,
> the outputs protected by the relays are disconnected from power,
> the logic signal states of the outputs remain unchanged.

**ⓘ NOTE**

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

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

Complete list of the device-specific error codes and diagnostic messages
→ chapter **System flags** (→ p. 215).

## 6.4 Relay: important notes!

14034

**NOTICE**

Premature wear of the relay contacts possible.

► In normal operation, only switch the relays without load!
For this purpose, set all relevant outputs to FALSE via the application program!

## 6.5 Response to system errors

14033
32423

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

► Process the specific error flags and/or error codes in the application program!
An error description is provided via the error flag / error code.
This error flag / error code can be further processed if necessary.

After analysis and elimination of the error cause:

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

## 6.6 CAN / CANopen: errors and error handling

28808

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

# 7 Appendix

**Content**

System flags ....215
Address assignment and I/O operating modes ....224
Error tables ....240

28796

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

## 7.1 System flags

**Content**

System flags: CAN ....215
System flags: SAE-J1939....216
System flags: error flags (standard side)....217
System flags: error flags (extended side)....218
System flags: status LED (standard side) ....219
System flags: status LED (extended side) ....219
System flags: voltages (standard side) ....220
System flags: voltages (extended side)....221
System flags: 16 inputs and 16 outputs (standard side) ....221
System flags: 16 inputs and 32 outputs (extended side) ....223

28371

❗ 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

12820

| 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"<br>ℹ Reset of the error code alse resets the flag | |
| CANx_DOWNLOADID | BYTE | CAN interface x: set download identifier | |
| CANx_ERRORCOUNTER_RX | BYTE | CAN interface x: Error counter receiver<br>ℹ Reset of the flag is possible via write access | |
| CANx_ERRORCOUNTER_TX | BYTE | CAN interface x: error counter transmission<br>ℹ A reset of the flag is possible via write access | |
| CANx_LASTERROR | BYTE | CAN interface x:<br>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<br>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)<br>ℹ A reset of the flag is possible via write access | |

CANx stands for x = 1...4 = number of the CAN interface

## 7.1.2 System flags: SAE-J1939

12815

| System flags (symbol name) | Type | Description |
|---|---|---|
| J1939_RECEIVE_OVERWRITE | BOOL | Setting only applies to J1939 data that has not been transmitted via a J1939 transport protocol.<br>TRUE:<br>The old data is overwritten by the new data if the old data has not yet been read from the function block instance<br>FALSE:<br>New data is rejected as long as the old data has not been read from the function block instance<br>ℹ New data can arrive before the old data has been read out if the IEC cycle is longer than the refresh rate of the J1939 data |
| J1939_TASK | BOOL | Using J1939_TASK, the time requirement for sending J1939 messages is met.<br>If J1939 messages are to be sent with a repetition time ≤ 50 ms, the runtime system automatically sets J1939_TASK=TRUE.<br>For applications for which the time requirement is ≥ PLC cycle time:<br>▶ Reduce system load with J1939_TASK=FALSE!<br>TRUE: J1939 task is active (= initial value)<br>The task is called every 2 ms.<br>The J1939 stack sends its messages in the required time frame<br>FALSE: J1939 task is not active |

## 7.1.3 System flags: error flags (standard side)

12821

| System flags (symbol name) | Type | Description |
|---|---|---|
| ERROR | BOOL | TRUE = set group error message, switch off relay |
| ERROR_BREAK_Ix<br>(0...x, value depends on the device, → data sheet) | WORD | input word x: wire break error<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_BREAK_Qx<br>(0...x, value depends on the device, → data sheet) | WORD | output word x: wire break error<br>[Bit 0 for output 0] ... [bit z for output z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_CONTROL_Qx<br>(0...x, value depends on the device, → data sheet) | WORD | output word x:error current control<br>final value cannot be reached<br>[Bit 0 for output 0] ... [bit z for output z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_CURRENT_Ix<br>(0...x, value depends on the device, → data sheet) | WORD | input word x: over-current error<br>only if Ixx_MODE = IN_CURRENT<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_IO | BOOL | Group error message input / output error<br>TRUE: Error<br>FALSE: No error |
| ERROR_POWER | BOOL | Overvoltage error for VBBs / clamp 15:<br>TRUE: Value out of range<br>or: difference (VBB15 - VBBs) > 1 V<br>> general error<br>> application STOP<br>> outputs = inactive<br>> no communication<br>> message "Overvoltage clamp 15"<br>FALSE: Value OK |
| ERROR_SHORT_Ix<br>(0...x, value depends on the device, → data sheet) | WORD | input word x: short circuit error<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_SHORT_Qx<br>(0...x, value depends on the device, → data sheet) | WORD | output word x: short circuit error<br>[Bit 0 for output 0] ... [bit z for output z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_TEMPERATURE | BOOL | Temperature error<br>TRUE: Value out of range<br>> general error<br>FALSE: Value OK |
| ERROR_VBBx | BOOL | Supply voltage error on VBBx (x = o \| r):<br>TRUE: Value out of range<br>> general error<br>FALSE: Value OK |
| LAST_RESET | BYTE | Cause for the last reset:<br>00 = reset of the application<br>01 = power-on reset<br>02 = watchdog reset<br>03 = soft reset<br>04 = unknown cause |

### 7.1.4 System flags: error flags (extended side)

12823

| System flags (symbol name) | Type | Description |
|---|---|---|
| BOARD_LINK_ERROR | BOOL | The connection to the extended side is...<br>TRUE: interrupted<br>the extended side is offline<br>❗ If the connection is interrupted, no automatic reconnection will be possible. ▶ Restart device!<br>FALSE: OK |
| BOARD_LINK_WARNING | BOOL | The connection to the extended side is...<br>TRUE: disturbed but operational<br>FALSE: OK |
| ERROR_BREAK_Ix_E<br>(0...x, value depends on the device, → data sheet) | WORD | extended input word x: wire break error<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_BREAK_Q0_E | DWORD | first extended output double word: wire break error<br>[Bit 0 for output 0] ... [bit z for output z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_CONTROL_Qx_E<br>(0...x, value depends on the device, → data sheet) | WORD | extended output word x:error current control<br>final value cannot be reached<br>[Bit 0 for output 0] ... [bit z for output z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_CURRENT_Ix_E<br>(0...x, value depends on the device, → data sheet) | WORD | extended input word x: over-current error<br>only if Ixx_MODE_E = IN_CURRENT<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_IO_E | BOOL | Group error message input / output error extended side<br>TRUE: Error<br>FALSE: No error |
| ERROR_POWER_E | BOOL | Voltage error extended side:<br>TRUE: Value out of range<br>FALSE: Value OK |
| ERROR_SHORT_Ix_E<br>(0...x, value depends on the device, → data sheet) | WORD | extended input word x: short circuit error<br>[Bit 0 for input 0] ... [bit z for input z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_SHORT_Qx_E<br>(0...x, value depends on the device, → data sheet) | WORD | extended output word x: short circuit error<br>[Bit 0 for output 0] ... [bit z for output z] of this group<br>Bit = TRUE: error<br>Bit = FALSE: no error |
| ERROR_VBBx_E | BOOL | Supply voltage error on extended VBBx<br>x = 1 \| 2 \| 3 \| 4<br>TRUE: Value out of range<br>FALSE: Value OK |
| ERROR_VBBREL_E | BOOL | Supply voltage error at relay supply:<br>TRUE: Value out of range<br>FALSE: Value OK |

### 7.1.5 System flags: status LED (standard side)

28376

| System flags (symbol name) | Type | Description |
|---|---|---|
| LED | WORD | LED color for "LED switched on":<br>0x0000 = LED_GREEN (preset)<br>0x0001 = LED_BLUE<br>0x0002 = LED_RED<br>0x0003 = LED_WHITE<br>0x0004 = LED_BLACK<br>0x0005 = LED_MAGENTA<br>0x0006 = LED_CYAN<br>0x0007 = LED_YELLOW |
| LED_X | WORD | LED color for "LED switched off":<br>0x0000 = LED_GREEN<br>0x0001 = LED_BLUE<br>0x0002 = LED_RED<br>0x0003 = LED_WHITE<br>0x0004 = LED_BLACK (preset)<br>0x0005 = LED_MAGENTA<br>0x0006 = LED_CYAN<br>0x0007 = LED_YELLOW |
| LED_MODE | WORD | LED flashing frequency:<br>0x0000 = LED_2HZ (flashes at 2 Hz; preset)<br>0x0001 = LED_1HZ (flashes at 1 Hz)<br>0x0002 = LED_05HZ (flashes at 0.5 Hz)<br>0x0003 = LED_0HZ (lights permanently with value in LED) |

### 7.1.6 System flags: status LED (extended side)

12824

| System flags (symbol name) | Type | Description |
|---|---|---|
| LED_E | WORD | LED color for "LED switched on":<br>0x0000 = LED_GREEN (preset)<br>0x0001 = LED_BLUE<br>0x0002 = LED_RED<br>0x0003 = LED_WHITE<br>0x0004 = LED_BLACK<br>0x0005 = LED_MAGENTA<br>0x0006 = LED_CYAN<br>0x0007 = LED_YELLOW |
| LED_X_E | WORD | LED color for "LED switched off":<br>0x0000 = LED_GREEN<br>0x0001 = LED_BLUE<br>0x0002 = LED_RED<br>0x0003 = LED_WHITE<br>0x0004 = LED_BLACK (preset)<br>0x0005 = LED_MAGENTA<br>0x0006 = LED_CYAN<br>0x0007 = LED_YELLOW |
| LED_MODE_E | WORD | LED flashing frequency:<br>0x0000 = LED_2HZ (flashes at 2 Hz; preset)<br>0x0001 = LED_1HZ (flashes at 1 Hz)<br>0x0002 = LED_05HZ (flashes at 0.5 Hz)<br>0x0003 = LED_0HZ (lights permanently with value in LED_E) |

## 7.1.7 System flags: voltages (standard side)

12822

| System flags (symbol name) | Type | Description |
|---|---|---|
| CLAMP_15_VOLTAGE | WORD | voltage applied to clamp 15 in [mV] |
| REF_VOLTAGE | WORD | Voltage on reference voltage output in [mV] |
| REFERENCE_VOLTAGE_5 | BOOL | Reference voltage output with 5 V activated |
| REFERENCE_VOLTAGE_10 | BOOL | Reference voltage output with 10 V activated |
| RELAIS_VBBy<br>y = o \| r | BOOL | TRUE: relay for VBBy activated voltage is applied to output group x (x = 1 \| 2)<br>FALSE: relay for VBBy deactivated no voltage is applied to output group x |
| SERIAL_MODE | BOOL | Activate serial interface (RS232) for use in the application<br>TRUE:<br>The RS232 interface can be used in the application, but no longer for programming, debugging or monitoring of the device.<br>FALSE:<br>The RS232 interface cannot be used in the application. Programming, debugging or monitoring of the device is possible. |
| SUPPLY_SWITCH | BOOL | Bit for switching off the supply latching VBBs. Resetting the flag is only accepted by the runtime system if the voltage at clamp 15 < 4 V, otherwise the flag is activated again. Separation of VBBs is done before the next PLC cycle starts. Depending on the charging status of the internal capacitors it may take some time until the device switches off.<br>TRUE: Supply of the device via VBBs is active<br>FALSE: Supply of the device via VBBs is deactivated |
| SUPPLY_VOLTAGE | WORD | supply voltage at VBBs in [mV] |
| TEST | BOOL | TRUE: Test input is active:<br>• Programming mode is enabled<br>• Software download is possible<br>• Status of the application program can be queried<br>• Protection of stored software is not possible<br>FALSE: application is in operation |
| VBBx_RELAIS_VOLTAGE<br>x = o \| r | WORD | Supply voltage on VBBx to relay contact in [mV] |
| VBBx_VOLTAGE<br>x = o \| r | WORD | Supply voltage on VBBx in [mV] |

### 7.1.8 System flags: voltages (extended side)

12203

| System flags (symbol name) | Type | Description |
|---|---|---|
| RELAIS_VBBy_E<br>y = o \| r | BOOL | TRUE: relay for VBBy activated<br>VBBo → VBB1 + VBB3<br>VBBr → VBB2 + VBB4<br>voltage is applied to extended output group x<br>(x = 1 \| 2 \| 3 \| 4)<br>FALSE: relay for VBBy deactivated<br>no voltage is applied to output group x |
| VBBx_RELAIS_VOLTAGE_E<br>x = 1 \| 2 \| 3 \| 4 | WORD | Supply voltage on VBBx_E to relay contact in [mV] |
| VBB_RELAIS_VOLTAGE_E | WORD | Supply voltage for relay supply in [mV] |

### 7.1.9 System flags: 16 inputs and 16 outputs (standard side)

13119

| System flags (symbol name) | Type | Description |
|---|---|---|
| ANALOGxx<br>xx = 00...15 | WORD | Analogue input xx:<br>filtered A/D converter raw value (12 bits) without calibration or standardisation |
| ANALOG_IRQxx<br>xx = 00...07 | WORD | Analogue input xx:<br>unfiltered A/D converter raw value (12 bits)<br>without calibration or standardisation<br>Use in FB **SET_INTERRUPT_I** (→ p. 125) or **SET_INTERRUPT_XMS** (→ p. 127) |
| CURRENTxx<br>xx = 00...15 | WORD | PWM output xx:<br>filtered A/D converter raw values (12 bits) of the current measurement without calibration or standardisation |
| Ixx<br>xx = 00...15 | BOOL | Status on binary input xx<br>Condition: input is configured as binary input<br>(MODE = IN_DIGITAL_H or IN_DIGITAL_L)<br>TRUE: Voltage on binary input > 70 % of VBBS<br>FALSE: Voltage on binary input < 30 % of VBBS<br>or: not configured as binary input<br>or: wrong configuration |
| Ixx_DFILTER<br>xx = 00...11 | DWORD | Pulse input xx:<br>pulse duration in [µs] which is to be ignored as a glitch.<br>Acquisition of the input signal is delayed by the set time.<br>allowed = 0...100 000 µs<br>preset = 0 µs = no filter |
| Ixx_FILTER<br>xx = 00...15 | BYTE:=4 | Binary and analogue input xx:<br>limit frequency (or signal rise time) of the first-order software low-pass filter<br>0 = 0x00 = no filter<br>1 = 0x01 = 390 Hz (1 ms)<br>2 = 0x02 = 145 Hz (2.5 ms)<br>3 = 0x03 = 68 Hz (5 ms)<br>4 = 0x04 = 34 Hz (10 ms) (preset)<br>5 = 0x05 = 17 Hz (21 ms)<br>6 = 0x06 = 8 Hz (42 ms)<br>7 = 0x07 = 4 Hz (84 ms)<br>8 = 0x08 = 2 Hz (169 ms)<br>higher = → preset value |
| Ixx_MODE<br>xx = 00...15 | BYTE | Operating mode of the input Ixx<br>→ chapter **Possible operating modes inputs/outputs** (→ p. 233) |

| System flags (symbol name) | Type | Description |
|---|---|---|
| Qxx<br>xx = 00...15 | BOOL | Status on binary output xx:<br>Condition: output is configured as binary output<br>TRUE: output activated<br>FALSE: output deactivated (= initial value)<br>or: not configured as binary output |
| Qxx_FILTER<br>xx = 00...15 | BYTE | Output xx:<br>limit frequency of the first-order software low-pass filter for the current measurement<br>only if Qxx_MODE = OUT_DIGITAL_H<br>not if PWM mode<br>0 = 0x00 = no filter<br>1 = 0x01 = 580 Hz (0.6 ms)<br>2 = 0x02 = 220 Hz (1.6 ms)<br>3 = 0x03 = 102 Hz (3.5 ms)<br>4 = 0x04 = 51 Hz (7 ms) (preset)<br>5 = 0x05 = 25 Hz (14 ms)<br>6 = 0x06 = 12 Hz (28 ms)<br>7 = 0x07 = 6 Hz (56 ms)<br>8 = 0x08 = 3 Hz (112 ms)<br>higher = → preset value |
| Qxx_MODE<br>xx = 00...15 | BYTE | Operating mode of the output Qxx<br>→ chapter **Possible operating modes inputs/outputs** (→ p. 233) |

## 7.1.10 System flags: 16 inputs and 32 outputs (extended side)

13120

| System flags (symbol name) | Type | Description |
|---|---|---|
| ANALOGxx_E<br>xx = 00...15 | WORD | Extended analogue input xx:<br>filtered A/D converter raw value (12 bits) without calibration or standardisation |
| CURRENTxx_E<br>xx = 00...15 | WORD | Extended PWM output xx:<br>filtered A/D converter raw values (12 bits) of the current measurement without calibration or standardisation |
| Ixx_E<br>xx = 00...15 | BOOL | Status at extended binary input xx_E<br>Condition: input is configured as binary input<br>(MODE = IN_DIGITAL_H or IN_DIGITAL_L)<br>TRUE: Voltage on binary input > 70 % of VBBS<br>FALSE: Voltage on binary input < 30 % of VBBS<br>or: not configured as binary input<br>or: wrong configuration |
| Ixx_DFILTER_E<br>xx = 00...11 | DWORD | Extended pulse input xx:<br>pulse duration in [µs] which is to be ignored as a glitch.<br>Acquisition of the input signal is delayed by the set time.<br>allowed = 0...100 000 µs<br>preset = 0 µs = no filter |
| Ixx_FILTER_E<br>xx = 00...15 | BYTE:=4 | Extended binary and analogue input xx_E:<br>limit frequency (or signal rise time) of the first-order software low-pass filter<br>0 = 0x00 = no filter<br>1 = 0x01 = 390 Hz (1 ms)<br>2 = 0x02 = 145 Hz (2.5 ms)<br>3 = 0x03 = 68 Hz (5 ms)<br>4 = 0x04 = 34 Hz (10 ms) (preset)<br>5 = 0x05 = 17 Hz (21 ms)<br>6 = 0x06 = 8 Hz (42 ms)<br>7 = 0x07 = 4 Hz (84 ms)<br>8 = 0x08 = 2 Hz (169 ms)<br>higher = → preset value |
| Ixx_MODE_E<br>xx = 00...15 | BYTE | Operating mode of the extended input Ixx_E<br>→ chapter **Possible operating modes inputs/outputs** (→ p. 233) |
| Qxx_E<br>xx = 00...31 | BOOL | Status on extended binary input xx_E:<br>Condition: output is configured as binary output<br>TRUE: output activated<br>FALSE: output deactivated (= initial value)<br>or: not configured as binary output |
| Qxx_FILTER_E<br>xx = 00...15 | BYTE | Extended output xx:<br>limit frequency of the first-order software low-pass filter for the current measurement<br>only if Qxx_MODE_E = OUT_DIGITAL_H<br>not if PWM mode<br>0 = 0x00 = no filter<br>1 = 0x01 = 580 Hz (0.6 ms)<br>2 = 0x02 = 220 Hz (1.6 ms)<br>3 = 0x03 = 102 Hz (3.5 ms)<br>4 = 0x04 = 51 Hz (7 ms) (preset)<br>5 = 0x05 = 25 Hz (14 ms)<br>6 = 0x06 = 12 Hz (28 ms)<br>7 = 0x07 = 6 Hz (56 ms)<br>8 = 0x08 = 3 Hz (112 ms)<br>higher = → preset value |
| Qxx_MODE_E<br>xx = 00...31 | BYTE | Operating mode of the extended output Qxx_E<br>→ chapter **Possible operating modes inputs/outputs** (→ p. 233) |

## 7.2 Address assignment and I/O operating modes

**Content**

Addresses / variables of the I/Os ....................225
Possible operating modes inputs/outputs....................233

28801

→ also data sheet

## 7.2.1 Addresses / variables of the I/Os

**Content**

Inputs: addresses and variables (standard side) (16 inputs) ....................225
Inputs: addresses and variables (extended side) (16 inputs)....................226
Outputs: addresses and variables (standard side) (16 outputs) ....................228
Outputs: addresses and variables (extended side) (32 outputs)....................230

28802

### Inputs: addresses and variables (standard side) (16 inputs)

13352

Abbreviations →chapter **Note on wiring** (→ p. 32)
Operating modes of the inputs/outputs →chapter **Possible operating modes inputs/outputs** (→ p. 233)

| IEC address | I/O variable | Remark |
|---|---|---|
| %IX0.0 | I00 | Binary input channel 0 |
| %IX0.1 | I01 | Binary input channel 1 |
| %IX0.2 | I02 | Binary input channel 2 |
| %IX0.3 | I03 | Binary input channel 3 |
| %IX0.4 | I04 | Binary input channel 4 |
| %IX0.5 | I05 | Binary input channel 5 |
| %IX0.6 | I06 | Binary input channel 6 |
| %IX0.7 | I07 | Binary input channel 7 |
| %IX0.8 | I08 | Binary input channel 8 |
| %IX0.9 | I09 | Binary input channel 9 |
| %IX0.10 | I10 | Binary input channel 10 |
| %IX0.11 | I11 | Binary input channel 11 |
| %IX0.12 | I12 | Binary input channel 12 |
| %IX0.13 | I13 | Binary input channel 13 |
| %IX0.14 | I14 | Binary input channel 14 |
| %IX0.15 | I15 | Binary input channel 15 |
| | | |
| %IW2 | ANALOG00 | Analogue input channel 0 |
| %IW3 | ANALOG01 | Analogue input channel 1 |
| %IW4 | ANALOG02 | Analogue input channel 2 |
| %IW5 | ANALOG03 | Analogue input channel 3 |
| %IW6 | ANALOG04 | Analogue input channel 4 |
| %IW7 | ANALOG05 | Analogue input channel 5 |
| %IW8 | ANALOG06 | Analogue input channel 6 |
| %IW9 | ANALOG07 | Analogue input channel 7 |
| %IW10 | ANALOG08 | Analogue input channel 8 |
| %IW11 | ANALOG09 | Analogue input channel 9 |
| %IW12 | ANALOG10 | Analogue input channel 10 |
| %IW13 | ANALOG11 | Analogue input channel 11 |
| %IW14 | ANALOG12 | Analogue input channel 12 |
| %IW15 | ANALOG13 | Analogue input channel 13 |

| IEC address | I/O variable | Remark |
|---|---|---|
| %IW16 | ANALOG14 | Analogue input channel 14 |
| %IW17 | ANALOG15 | Analogue input channel 15 |
| | | |
| %IW18 | CURRENT00 | Output current (raw value) on Q00 |
| %IW19 | CURRENT01 | Output current (raw value) on Q01 |
| %IW20 | CURRENT02 | Output current (raw value) on Q02 |
| %IW21 | CURRENT03 | Output current (raw value) on Q03 |
| %IW22 | CURRENT04 | Output current (raw value) on Q04 |
| %IW23 | CURRENT05 | Output current (raw value) on Q05 |
| %IW24 | CURRENT06 | Output current (raw value) on Q06 |
| %IW25 | CURRENT07 | Output current (raw value) on Q07 |
| %IW26 | CURRENT08 | Output current (raw value) on Q08 |
| %IW27 | CURRENT09 | Output current (raw value) on Q09 |
| %IW28 | CURRENT10 | Output current (raw value) on Q10 |
| %IW29 | CURRENT11 | Output current (raw value) on Q11 |
| %IW30 | CURRENT12 | Output current (raw value) on Q12 |
| %IW31 | CURRENT13 | Output current (raw value) on Q13 |
| %IW32 | CURRENT14 | Output current (raw value) on Q14 |
| %IW33 | CURRENT15 | Output current (raw value) on Q15 |
| | | |
| %IW34 | SUPPLY_VOLTAGE | Supply voltage on VBBs in [mV] |
| %IW35 | CLAMP_15_VOLTAGE | Voltage clamp 15 |
| %IW36 | VBBO_VOLTAGE | Supply voltage on VBBo in [mV] |
| %IW37 | VBBR_VOLTAGE | Supply voltage on VBBr in [mV] |
| %IW38 | VBBO_RELAIS_VOLTAGE | Supply voltage VBBo to relay contact in [mV] |
| %IW39 | VBBR_RELAIS_VOLTAGE | Supply voltage VBBr to relay contact in [mV] |
| %IW40 | REF_VOLTAGE | Voltage on the reference output pin 51 |
| | | |
| %IW41 | ANALOG_IRQ00 | Interrupt to analogue input channel 0 |
| %IW42 | ANALOG_IRQ01 | Interrupt to analogue input channel 1 |
| %IW43 | ANALOG_IRQ02 | Interrupt to analogue input channel 2 |
| %IW44 | ANALOG_IRQ03 | Interrupt to analogue input channel 3 |
| %IW45 | ANALOG_IRQ04 | Interrupt to analogue input channel 4 |
| %IW46 | ANALOG_IRQ05 | Interrupt to analogue input channel 5 |
| %IW47 | ANALOG_IRQ06 | Interrupt to analogue input channel 6 |
| %IW48 | ANALOG_IRQ07 | Interrupt to analogue input channel 7 |
| | | |
| %MB7960 | ERROR_CURRENT_I0 | Error DWORD overcurrent |
| %MB7964 | ERROR_SHORT_I0 | Error DWORD short circuit |
| %MB7968 | ERROR_BREAK_I0 | Error DWORD wire break |

## Inputs: addresses and variables (extended side) (16 inputs)

12082

Abbreviations →chapter **Note on wiring** (→ p. 32)
Operating modes of the inputs/outputs →chapter **Possible operating modes inputs/outputs** (→ p. 233)

| IEC address | I/O variable | Remark |
|---|---|---|

| IEC address | I/O variable | Remark |
|---|---|---|
| %IX128.0 | I00_E | Binary input channel 0 |
| %IX128.1 | I01_E | Binary input channel 1 |
| %IX128.2 | I02_E | Binary input channel 2 |
| %IX128.3 | I03_E | Binary input channel 3 |
| %IX128.4 | I04_E | Binary input channel 4 |
| %IX128.5 | I05_E | Binary input channel 5 |
| %IX128.6 | I06_E | Binary input channel 6 |
| %IX128.7 | I07_E | Binary input channel 7 |
| %IX128.8 | I08_E | Binary input channel 8 |
| %IX128.9 | I09_E | Binary input channel 9 |
| %IX128.10 | I10_E | Binary input channel 10 |
| %IX128.11 | I11_E | Binary input channel 11 |
| %IX128.12 | I12_E | Binary input channel 12 |
| %IX128.13 | I13_E | Binary input channel 13 |
| %IX128.14 | I14_E | Binary input channel 14 |
| %IX128.15 | I15_E | Binary input channel 15 |
| | | |
| %IW130 | ANALOG00_E | Analogue input channel 0 |
| %IW131 | ANALOG01_E | Analogue input channel 1 |
| %IW132 | ANALOG02_E | Analogue input channel 2 |
| %IW133 | ANALOG03_E | Analogue input channel 3 |
| %IW134 | ANALOG04_E | Analogue input channel 4 |
| %IW135 | ANALOG05_E | Analogue input channel 5 |
| %IW136 | ANALOG06_E | Analogue input channel 6 |
| %IW137 | ANALOG07_E | Analogue input channel 7 |
| %IW138 | ANALOG08_E | Analogue input channel 8 |
| %IW139 | ANALOG09_E | Analogue input channel 9 |
| %IW140 | ANALOG10_E | Analogue input channel 10 |
| %IW141 | ANALOG11_E | Analogue input channel 11 |
| %IW142 | ANALOG12_E | Analogue input channel 12 |
| %IW143 | ANALOG13_E | Analogue input channel 13 |
| %IW144 | ANALOG14_E | Analogue input channel 14 |
| %IW145 | ANALOG15_E | Analogue input channel 15 |
| | | |
| %IW146 | CURRENT00_E | Output current (raw value) on Q00_E |
| %IW147 | CURRENT01_E | Output current (raw value) on Q01_E |
| %IW148 | CURRENT02_E | Output current (raw value) on Q02_E |
| %IW149 | CURRENT03_E | Output current (raw value) on Q03_E |
| %IW150 | CURRENT04_E | Output current (raw value) on Q04_E |
| %IW151 | CURRENT05_E | Output current (raw value) on Q05_E |
| %IW152 | CURRENT06_E | Output current (raw value) on Q06_E |
| %IW153 | CURRENT07_E | Output current (raw value) on Q07_E |
| %IW154 | CURRENT08_E | Output current (raw value) on Q08_E |

| IEC address | I/O variable | Remark |
|---|---|---|
| %IW155 | CURRENT09_E | Output current (raw value) on Q09_E |
| %IW156 | CURRENT10_E | Output current (raw value) on Q10_E |
| %IW157 | CURRENT11_E | Output current (raw value) on Q11_E |
| %IW158 | CURRENT12_E | Output current (raw value) on Q12_E |
| %IW159 | CURRENT13_E | Output current (raw value) on Q13_E |
| %IW160 | CURRENT14_E | Output current (raw value) on Q14_E |
| %IW161 | CURRENT15_E | Output current (raw value) on Q15_E |
| | | |
| %IW162 | VBB1_E | Supply voltage on VBB1 in [mV] |
| %IW163 | VBB2_E | Supply voltage on VBB2 in [mV] |
| %IW164 | VBB3_E | Supply voltage on VBB3 in [mV] |
| %IW165 | VBB4_E | Supply voltage on VBB4 in [mV] |
| %IW166 | VBB1_RELAIS_VOLTAGE | Supply voltage VBB1 to relay contact in [mV] |
| %IW167 | VBB2_RELAIS_VOLTAGE | Supply voltage VBB2 to relay contact in [mV] |
| %IW168 | VBB3_RELAIS_VOLTAGE | Supply voltage VBB3 to relay contact in [mV] |
| %IW169 | VBB4_RELAIS_VOLTAGE | Supply voltage VBB4 to relay contact in [mV] |
| %IW170 | VBB_RELAIS_VOLTAGE | Supply voltage on VBBrel in [mV] |
| | | |
| %MB8048 | ERROR_CURRENT_I0_E | Error DWORD overcurrent |
| %MB8052 | ERROR_SHORT_I0_E | Error DWORD short circuit |
| %MB8056 | ERROR_BREAK_I0_E | Error DWORD wire break |

## Outputs: addresses and variables (standard side) (16 outputs)

13354

Abbreviations →chapter **Note on wiring** (→ p. 32)
Operating modes of the inputs/outputs →chapter **Possible operating modes inputs/outputs** (→ p. 233)

| IEC address | I/O variable | Remark |
|---|---|---|
| %QX0.0 | Q00 | Binary output / PWM output channel 0 |
| %QX0.1 | Q01 | Binary output / PWM output channel 1 |
| %QX0.2 | Q02 | Binary output / PWM output channel 2 |
| %QX0.3 | Q03 | Binary output / PWM output channel 3 |
| %QX0.4 | Q04 | Binary output / PWM output channel 4 |
| %QX0.5 | Q05 | Binary output / PWM output channel 5 |
| %QX0.6 | Q06 | Binary output / PWM output channel 6 |
| %QX0.7 | Q07 | Binary output / PWM output channel 7 |
| %QX0.8 | Q08 | Binary output / PWM output channel 8 |
| %QX0.9 | Q09 | Binary output / PWM output channel 9 |
| %QX0.10 | Q10 | Binary output / PWM output channel 10 |
| %QX0.11 | Q11 | Binary output / PWM output channel 11 |
| %QX0.12 | Q12 | Binary output / PWM output channel 12 |
| %QX0.13 | Q13 | Binary output / PWM output channel 13 |
| %QX0.14 | Q14 | Binary output / PWM output channel 14 |
| %QX0.15 | Q15 | Binary output / PWM output channel 15 |
| | | |
| %QB2 | REFERENCE_VOLTAGE_5 | Activating the reference voltage output with 5 V |

| IEC address | I/O variable | Remark |
|---|---|---|
| %QB3 | REFERENCE_VOLTAGE_10 | Activating the reference voltage output with 10 V |
| | | |
| %QB68 | I00_FILTER | Filter byte for %IX0.0 / %IW2 |
| %QB69 | I01_FILTER | Filter byte for %IX0.1 / %IW3 |
| %QB70 | I02_FILTER | Filter byte for %IX0.2 / %IW4 |
| %QB71 | I03_FILTER | Filter byte for %IX0.3 / %IW5 |
| %QB72 | I04_FILTER | Filter byte for %IX0.4 / %IW6 |
| %QB73 | I05_FILTER | Filter byte for %IX0.5 / %IW7 |
| %QB74 | I06_FILTER | Filter byte for %IX0.6 / %IW8 |
| %QB75 | I07_FILTER | Filter byte for %IX0.7 / %IW9 |
| %QB76 | I08_FILTER | Filter byte for %IX0.8 / %IW2 |
| %QB77 | I09_FILTER | Filter byte for %IX0.9 / %IW3 |
| %QB78 | I10_FILTER | Filter byte for %IX0.10 / %IW4 |
| %QB79 | I11_FILTER | Filter byte for %IX0.11 / %IW5 |
| %QB80 | I12_FILTER | Filter byte for %IX0.12 / %IW6 |
| %QB81 | I13_FILTER | Filter byte for %IX0.13 / %IW7 |
| %QB82 | I14_FILTER | Filter byte for %IX0.14 / %IW8 |
| %QB83 | I15_FILTER | Filter byte for %IX0.15 / %IW9 |
| | | |
| %QB84 | Q00_FILTER | Filter byte for %QX0.0 |
| %QB85 | Q01_FILTER | Filter byte for %QX0.1 |
| %QB86 | Q02_FILTER | Filter byte for %QX0.2 |
| %QB87 | Q03_FILTER | Filter byte for %QX0.3 |
| %QB88 | Q04_FILTER | Filter byte for %QX0.4 |
| %QB89 | Q05_FILTER | Filter byte for %QX0.5 |
| %QB90 | Q06_FILTER | Filter byte for %QX0.6 |
| %QB91 | Q07_FILTER | Filter byte for %QX0.7 |
| %QB92 | Q08_FILTER | Filter byte for %QX0.8 |
| %QB93 | Q09_FILTER | Filter byte for %QX0.9 |
| %QB94 | Q10_FILTER | Filter byte for %QX0.10 |
| %QB95 | Q11_FILTER | Filter byte for %QX0.11 |
| %QB96 | Q12_FILTER | Filter byte for %QX0.12 |
| %QB97 | Q13_FILTER | Filter byte for %QX0.13 |
| %QB98 | Q14_FILTER | Filter byte for %QX0.14 |
| %QB99 | Q15_FILTER | Filter byte for %QX0.15 |
| | | |
| %QD25 | I00_DFILTER | Filter value counting/pulse input 0 |
| %QD26 | I01_DFILTER | Filter value counting/pulse input 1 |
| %QD27 | I02_DFILTER | Filter value counting/pulse input 2 |
| %QD28 | I03_DFILTER | Filter value counting/pulse input 3 |
| %QD29 | I04_DFILTER | Filter value counting/pulse input 4 |
| %QD30 | I05_DFILTER | Filter value counting/pulse input 5 |
| %QD31 | I06_DFILTER | Filter value counting/pulse input 6 |
| %QD32 | I07_DFILTER | Filter value counting/pulse input 7 |

| IEC address | I/O variable | Remark |
|---|---|---|
| %QD33 | I08_DFILTER | Filter value counting/pulse input 8 |
| %QD34 | I09_DFILTER | Filter value counting/pulse input 9 |
| %QD35 | I10_DFILTER | Filter value counting/pulse input 10 |
| %QD36 | I11_DFILTER | Filter value counting/pulse input 11 |
| %MB7948 | ERROR_SHORT_Q0 | Error DWORD short circuit |
| %MB7952 | ERROR_BREAK_Q0 | Error DWORD wire break |
| %MB7956 | ERROR_CONTROL_Q0 | Error DWORD current control |

## Outputs: addresses and variables (extended side) (32 outputs)

12084

Abbreviations →chapter **Note on wiring** (→ p. 32)
Operating modes of the inputs/outputs →chapter **Possible operating modes inputs/outputs** (→ p. 233)

| IEC address | I/O variable | Remark |
|---|---|---|
| %QX128.0 | Q00_E | Binary output / PWM output channel 0 |
| %QX128.1 | Q01_E | Binary output / PWM output channel 1 |
| %QX128.2 | Q02_E | Binary output / PWM output channel 2 |
| %QX128.3 | Q03_E | Binary output / PWM output channel 3 |
| %QX128.4 | Q04_E | Binary output / PWM output channel 4 |
| %QX128.5 | Q05_E | Binary output / PWM output channel 5 |
| %QX128.6 | Q06_E | Binary output / PWM output channel 6 |
| %QX128.7 | Q07_E | Binary output / PWM output channel 7 |
| %QX128.8 | Q08_E | Binary output / PWM output channel 8 |
| %QX128.9 | Q09_E | Binary output / PWM output channel 9 |
| %QX128.10 | Q10_E | Binary output / PWM output channel 10 |
| %QX128.11 | Q11_E | Binary output / PWM output channel 11 |
| %QX128.12 | Q12_E | Binary output / PWM output channel 12 |
| %QX128.13 | Q13_E | Binary output / PWM output channel 13 |
| %QX128.14 | Q14_E | Binary output / PWM output channel 14 |
| %QX128.15 | Q15_E | Binary output / PWM output channel 15 |
| %QX128.16 | Q16_E | Binary output / PWM output channel 16 |
| %QX128.17 | Q17_E | Binary output / PWM output channel 17 |
| %QX128.18 | Q18_E | Binary output / PWM output channel 18 |
| %QX128.19 | Q19_E | Binary output / PWM output channel 19 |
| %QX128.20 | Q20_E | Binary output / PWM output channel 20 |
| %QX128.21 | Q21_E | Binary output / PWM output channel 21 |
| %QX128.22 | Q22_E | Binary output / PWM output channel 22 |
| %QX128.23 | Q23_E | Binary output / PWM output channel 23 |
| %QX128.24 | Q24_E | Binary output / PWM output channel 24 |
| %QX128.25 | Q25_E | Binary output / PWM output channel 25 |
| %QX128.26 | Q26_E | Binary output / PWM output channel 26 |
| %QX128.27 | Q27_E | Binary output / PWM output channel 27 |
| %QX128.28 | Q28_E | Binary output / PWM output channel 28 |

| IEC address | I/O variable | Remark |
|---|---|---|
| %QX128.29 | Q29_E | Binary output / PWM output channel 29 |
| %QX128.30 | Q30_E | Binary output / PWM output channel 30 |
| %QX128.31 | Q31_E | Binary output / PWM output channel 31 |
| | | |
| %QB356 | I00_FILTER_E | Filter byte for %IX128.0 / %IW130 |
| %QB357 | I01_FILTER_E | Filter byte for %IX128.1 / %IW131 |
| %QB358 | I02_FILTER_E | Filter byte for %IX128.2 / %IW132 |
| %QB359 | I03_FILTER_E | Filter byte for %IX128.3 / %IW133 |
| %QB360 | I04_FILTER_E | Filter byte for %IX128.4 / %IW134 |
| %QB361 | I05_FILTER_E | Filter byte for %IX128.5 / %IW135 |
| %QB362 | I06_FILTER_E | Filter byte for %IX128.6 / %IW136 |
| %QB363 | I07_FILTER_E | Filter byte for %IX128.7 / %IW137 |
| %QB364 | I08_FILTER_E | Filter byte for %IX128.8 / %IW138 |
| %QB365 | I09_FILTER_E | Filter byte for %IX128.9 / %IW139 |
| %QB366 | I10_FILTER_E | Filter byte for %IX128.10 / %IW140 |
| %QB367 | I11_FILTER_E | Filter byte for %IX128.11 / %IW141 |
| %QB368 | I12_FILTER_E | Filter byte for %IX128.12 / %IW142 |
| %QB369 | I13_FILTER_E | Filter byte for %IX128.13 / %IW143 |
| %QB370 | I14_FILTER_E | Filter byte for %IX128.14 / %IW144 |
| %QB371 | I15_FILTER_E | Filter byte for %IX128.15 / %IW145 |
| | | |
| %QB372 | Q00_FILTER_E | Filter byte for %QX128.0 |
| %QB373 | Q01_FILTER_E | Filter byte for %QX128.1 |
| %QB374 | Q02_FILTER_E | Filter byte for %QX128.2 |
| %QB375 | Q03_FILTER_E | Filter byte for %QX128.3 |
| %QB376 | Q04_FILTER_E | Filter byte for %QX128.4 |
| %QB377 | Q05_FILTER_E | Filter byte for %QX128.5 |
| %QB378 | Q06_FILTER_E | Filter byte for %QX128.6 |
| %QB379 | Q07_FILTER_E | Filter byte for %QX128.7 |
| %QB380 | Q08_FILTER_E | Filter byte for %QX128.8 |
| %QB381 | Q09_FILTER_E | Filter byte for %QX128.9 |
| %QB382 | Q10_FILTER_E | Filter byte for %QX128.10 |
| %QB383 | Q11_FILTER_E | Filter byte for %QX128.11 |
| %QB384 | Q12_FILTER_E | Filter byte for %QX128.12 |
| %QB385 | Q13_FILTER_E | Filter byte for %QX128.13 |
| %QB386 | Q14_FILTER_E | Filter byte for %QX128.14 |
| %QB387 | Q15_FILTER_E | Filter byte for %QX128.15 |
| | | |
| %QD97 | I00_DFILTER_E | Filter value counting/pulse input 0 |
| %QD98 | I01_DFILTER_E | Filter value counting/pulse input 1 |
| %QD99 | I02_DFILTER_E | Filter value counting/pulse input 2 |
| %QD100 | I03_DFILTER_E | Filter value counting/pulse input 3 |
| %QD101 | I04_DFILTER_E | Filter value counting/pulse input 4 |
| %QD102 | I05_DFILTER_E | Filter value counting/pulse input 5 |

| IEC address | I/O variable | Remark |
|---|---|---|
| %QD103 | I06_DFILTER_E | Filter value counting/pulse input 6 |
| %QD104 | I07_DFILTER_E | Filter value counting/pulse input 7 |
| %QD105 | I08_DFILTER_E | Filter value counting/pulse input 8 |
| %QD106 | I09_DFILTER_E | Filter value counting/pulse input 9 |
| %QD107 | I10_DFILTER_E | Filter value counting/pulse input 10 |
| %QD108 | I11_DFILTER_E | Filter value counting/pulse input 11 |
| | | |
| %MB8036 | ERROR_SHORT_Q0_E | Error DWORD short circuit |
| %MB8040 | ERROR_BREAK_Q0_E | Error DWORD wire break |
| %MB8044 | ERROR_CONTROL_Q0_E | Error DWORD current control |

## 7.2.2 Possible operating modes inputs/outputs

**Content**

Inputs: operating modes (standard side) (16 inputs)....233
Inputs: operating modes (extended side) (16 inputs)....235
Outputs: operating modes (standard side) (16 outputs)....236
Outputs: operating modes (extended side) (32 outputs)....238

28584

## Inputs: operating modes (standard side) (16 inputs)

15548

🟩 = this configuration value is default

| Inputs | Possible operating mode | | Set with function block | Function block input | Value | |
|---|---|---|---|---|---|---|
| | | | | | dec | hex |
| I00...I15 | IN_NOMODE | off | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 0 | 00 |
| | IN_DIGITAL_H | plus | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 🟩 1 | 🟩 01 |
| | IN_DIGITAL_L | minus | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 2 | 02 |
| | IN_CURRENT | 0…20 000 µA | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 4 | 04 |
| | IN_VOLTAGE10 | 0…10 000 mV | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 8 | 08 |
| | IN_VOLTAGE30 | 0…32 000 mV | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 16 | 10 |
| | IN_RATIO | 0…1 000 ‰ | INPUT_ANALOG<br>SET_INPUT_MODE | MODE | 32 | 20 |
| | Diagnosis | for IN_DIGITAL_H | SET_INPUT_MODE | DIAGNOSTICS | TRUE | |
| | Frequency measurement<br>Period duration measurement<br>Phase measurement | 0…30 000 Hz | FREQUENCY<br>FREQUENCY_PERIOD<br>PHASE | | | |
| | Period duration measurement | 0.1...5 000 Hz | PERIOD | | | |
| | Period duration and ratio measurement | 0.1...5 000 Hz | PERIOD_RATIO | | | |
| | Counters | 0…30 000 Hz | FAST_COUNT | | | |
| I00...I07 | Detect encoder | 0…30 000 Hz<br>0...5 000 Hz | INC_ENCODER<br>INC_ENCODER_HR | | | |

Set operating modes with the following function block:

| | |
|---|---|
| **FAST_COUNT** (→ p. 138) | Counter block for fast input pulses |
| **FREQUENCY** (→ p. 140) | Measures the frequency of the signal arriving at the selected channel |
| **FREQUENCY_PERIOD** (→ p. 142) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **INC_ENCODER** (→ p. 144) | Up/down counter function for the evaluation of encoders |
| **INC_ENCODER_HR** | Up/down counter function for the high resolution evaluation of encoders |
| **INPUT_ANALOG** (→ p. 130) | analogue input channel: alternatively measurement of ...<br>• current<br>• voltage |

| | |
|---|---|
| **PERIOD** (→ p. 146) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |
| **PERIOD_RATIO** (→ p. 148) | 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** (→ p. 150) | Reads a pair of channels with fast inputs and compares the phase position of the signals |
| **SET_INPUT_MODE** | Assigns an operating mode to an input channel |

## Inputs: operating modes (extended side) (16 inputs)

19370

🟩 = this configuration value is default

| Inputs | Possible operating mode | | Set with function block | Function block input | Value | |
|---|---|---|---|---|---|---|
| | | | | | dec | hex |
| I00_E…I15_E | IN_NOMODE | off | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 0 | 00 |
| | IN_DIGITAL_H | plus | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 1 | 01 |
| | IN_DIGITAL_L | minus | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 2 | 02 |
| | IN_CURRENT | 0…20 000 µA | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 4 | 04 |
| | IN_VOLTAGE10 | 0…10 000 mV | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 8 | 08 |
| | IN_VOLTAGE30 | 0…32 000 mV | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 16 | 10 |
| | IN_RATIO | 0…1 000 ‰ | INPUT_ANALOG_E<br>SET_INPUT_MODE_E | MODE | 32 | 20 |
| | Diagnosis | for IN_DIGITAL_H | SET_INPUT_MODE_E | DIAGNOSTICS | TRUE | |
| | Frequency measurement<br>Interval measurement<br>Phase measurement | 0…30 000 Hz | FREQUENCY_E<br>FREQUENCY_PERIOD_E<br>PHASE_E | | | |
| | Interval measurement | 0,1...5 000 Hz | PERIOD_E | | | |
| | Period and ratio measurement | 0,1...5 000 Hz | PERIOD_RATIO_E | | | |
| | Counter | 0…30 000 Hz | FAST_COUNT_E | | | |
| I00_E...I07_E | Detect encoder | 0…30 000 Hz<br>0...5 000 Hz | INC_ENCODER_E<br>INC_ENCODER_HR_E | | | |

Set operating modes with the following function block:

| | |
|---|---|
| FAST_COUNT_E | = **FAST_COUNT** (→ p. 138) for the extended side |
| FREQUENCY_E | = **FREQUENCY** (→ p. 140) for the extended side |
| FREQUENCY_PERIOD_E | = **FREQUENCY_PERIOD** (→ p. 142) for the extended side |
| INC_ENCODER_E | = **INC_ENCODER** (→ p. 144) for the extended side |
| INC_ENCODER_HR_E | = **INC_ENCODER_HR** for the extended side |
| INPUT_ANALOG_E | = **INPUT_ANALOG** (→ p. 130) for the extended side |
| PERIOD_E | = **PERIOD** (→ p. 146) for the extended side |
| PERIOD_RATIO_E | = **PERIOD_RATIO** (→ p. 148) for the extended side |
| PHASE_E | = **PHASE** (→ p. 150) for the extended side |
| SET_INPUT_MODE_E | = **SET_INPUT_MODE** for the extended side |

## Outputs: operating modes (standard side) (16 outputs)

15523

🟩 = this configuration value is default

| Outputs | Possible operating mode | | Set with function block | Function block input | Value | |
|---|---|---|---|---|---|---|
| | | | | | dec | hex |
| Q00…Q15 | OUT_DIGITAL_H | plus | SET_OUTPUT_MODE | MODE | 1 | 0001 |
| | OUT_DIGITAL_L | minus | SET_OUTPUT_MODE | MODE | 2 | 0002 |
| | Diagnosis | for OUT_DIGITAL_H via current measurement | SET_OUTPUT_MODE | DIAGNOSTICS | TRUE | |
| | Overload protection | for OUT_DIGITAL_H with current measurement | SET_OUTPUT_MODE | PROTECTION | TRUE | |
| | Current measuring range | no current measurement | SET_OUTPUT_MODE | CURRENT_RANGE | 0 | 00 |
| | | 2 A / 3 A | SET_OUTPUT_MODE | CURRENT_RANGE | 1 | 01 |
| | | 4 A | SET_OUTPUT_MODE | CURRENT_RANGE | 2 | 02 |

Details → chapter **Outputs Q00...Q15: permitted operating modes** (→ p. 237)

Set operating modes with the following function block:

| | |
|---|---|
| **OUTPUT_BRIDGE** (→ p. 153) | H-bridge on a PWM channel pair |
| **OUTPUT_CURRENT_CONTROL** (→ p. 157) | Current controller for a PWMi output channel |
| **PWM1000** (→ p. 160) | Initialises and configures a PWM-capable output channel<br>the mark-to-space ratio can be indicated in steps of 1 ‰ |
| **SET_OUTPUT_MODE** | Sets the operating mode of the selected output channel |

## Outputs Q00...Q15: permitted operating modes

19296

| Operating mode | | Q00 | Q01 | Q02 | Q03 | Q04 | Q05 | Q06 | Q07 |
|---|---|---|---|---|---|---|---|---|---|
| OUT_DIGITAL_H | plus | X | X | X | X | X | X | X | X |
| OUT_DIGITAL_L | minus | -- | X | -- | X | -- | -- | -- | -- |
| Diagnosis | for OUT_DIGITAL_H via current measurement | X | X | X | X | X | X | X | X |
| Overload protection | for OUT_DIGITAL_H with current measurement | X | X | X | X | X | X | X | X |
| Current measuring range | 2 A | X | X | X | X | X | X | X | X |
| | 4 A | X | X | X | X | -- | -- | -- | -- |
| PWM | | X | X | X | X | X | X | X | X |
| PWMi | | X | X | X | X | X | X | X | X |
| H-bridge | | -- | X | -- | X | -- | -- | -- | -- |

| Operating mode | | Q08 | Q09 | Q10 | Q11 | Q12 | Q13 | Q14 | Q15 |
|---|---|---|---|---|---|---|---|---|---|
| OUT_DIGITAL_H | plus | X | X | X | X | X | X | X | X |
| OUT_DIGITAL_L | minus | -- | X | -- | X | -- | -- | -- | -- |
| Diagnosis | for OUT_DIGITAL_H via current measurement | X | X | X | X | X | X | X | X |
| Overload protection | for OUT_DIGITAL_H with current measurement | X | X | X | X | X | X | X | X |
| Current measuring range | 2 A | X | X | X | X | X | X | X | X |
| | 4 A | X | X | X | X | -- | -- | -- | -- |
| PWM | | X | X | X | X | X | X | X | X |
| PWMi | | X | X | X | X | X | X | X | X |
| H-bridge | | -- | X | -- | X | -- | -- | -- | -- |

## Outputs: operating modes (extended side) (32 outputs)

19297

🟩 = this configuration value is default

| Outputs | Possible operating mode | | Set with function block | Function block input | Value | |
|---|---|---|---|---|---|---|
| | | | | | dec | hex |
| Q00_E<br>...Q15_E | OUT_DIGITAL_H | plus | SET_OUTPUT_MODE_E | MODE | 1 | 0001 |
| | OUT_DIGITAL_L | minus | SET_OUTPUT_MODE_E | MODE | 2 | 0002 |
| | Diagnosis | for OUT_DIGITAL_H via current measurement | SET_OUTPUT_MODE_E | DIAGNOSTICS | TRUE | |
| | Overload protection | for OUT_DIGITAL_H with current measurement | SET_OUTPUT_MODE_E | PROTECTION | TRUE | |
| | Current measuring range | no current measurement | SET_OUTPUT_MODE_E | CURRENT_RANGE | 0 | 00 |
| | | 2 A | SET_OUTPUT_MODE_E | CURRENT_RANGE | 1 | 01 |
| | | 4 A | SET_OUTPUT_MODE_E | CURRENT_RANGE | 2 | 02 |
| Q16_E<br>...Q32_E | OUT_DIGITAL_H | plus | SET_OUTPUT_MODE_E | MODE | 1 | 0001 |
| | Diagnosis | for OUT_DIGITAL_H | SET_OUTPUT_MODE_E | DIAGNOSTICS | FALSE | |
| | Current measuring range | 2 A | SET_OUTPUT_MODE_E | CURRENT_RANGE | 1 | 01 |

Set operating modes with the following function block:

| | |
|---|---|
| OUTPUT_BRIDGE_E | = **OUTPUT_BRIDGE** (→ p. 153) for the extended side |
| OUTPUT_CURRENT_CONTROL_E | = **OUTPUT_CURRENT_CONTROL** (→ p. 157) for the extended side |
| PWM1000_E | = **PWM1000** (→ p. 160) for the extended side |
| SET_OUTPUT_MODE_E | = **SET_OUTPUT_MODE** for the extended side |

## Outputs Q00_E...Q31_E: permitted operating modes

19904

| Operating mode | | Q00_E | Q01_E | Q02_E | Q03_E | Q04_E | Q05_E | Q06_E | Q07_E |
|---|---|---|---|---|---|---|---|---|---|
| OUT_DIGITAL_H | plus | X | X | X | X | X | X | X | X |
| OUT_DIGITAL_L | minus | -- | X | -- | X | -- | -- | -- | -- |
| Diagnosis | for OUT_DIGITAL_H via current measurement | X | X | X | X | X | X | X | X |
| Overload protection | for OUT_DIGITAL_H with current measurement | X | X | X | X | X | X | X | X |
| Current measuring range | 2 A | X | X | X | X | X | X | X | X |
| | 4 A | X | X | X | X | -- | -- | -- | -- |
| PWM | | X | X | X | X | X | X | X | X |
| PWMi | | X | X | X | X | X | X | X | X |
| H-bridge | | -- | X | -- | X | -- | -- | -- | -- |

| Operating mode | | Q08_E | Q09_E | Q10_E | Q11_E | Q12_E | Q13_E | Q14_E | Q15_E |
|---|---|---|---|---|---|---|---|---|---|
| OUT_DIGITAL_H | plus | X | X | X | X | X | X | X | X |
| OUT_DIGITAL_L | minus | -- | X | -- | X | -- | -- | -- | -- |
| Diagnosis | for OUT_DIGITAL_H via current measurement | X | X | X | X | X | X | X | X |
| Overload protection | for OUT_DIGITAL_H with current measurement | X | X | X | X | X | X | X | X |
| Current measuring range | 2 A | X | X | X | X | X | X | X | X |
| | 4 A | X | X | X | X | -- | -- | -- | -- |
| PWM | | X | X | X | X | X | X | X | X |
| PWMi | | X | X | X | X | X | X | X | X |
| H-bridge | | -- | X | -- | X | -- | -- | -- | -- |

| Operating mode | | Q16_E | Q17_E | Q18_E | Q19_E | Q20_E | Q21_E | Q22_E | Q23_E |
|---|---|---|---|---|---|---|---|---|---|
| OUT_DIGITAL_H | plus | X | X | X | X | X | X | X | X |
| Diagnosis | for OUT_DIGITAL_H via voltage measurement | X | X | X | X | X | X | X | X |

| Operating mode | | Q24_E | Q25_E | Q26_E | Q27_E | Q28_E | Q29_E | Q30_E | Q31_E |
|---|---|---|---|---|---|---|---|---|---|
| OUT_DIGITAL_H | plus | X | X | X | X | X | X | X | X |
| Diagnosis | for OUT_DIGITAL_H via voltage measurement | X | X | X | X | X | X | X | X |

# 7.3 Error tables

**Content**

Error flags ....................................................................................................................240
Errors: CAN / CANopen.....................................................................................................240

28817

## 7.3.1 Error flags

28818

→ chapter **System flags** (→ p. 215)

## 7.3.2 Errors: CAN / CANopen

19610
28819

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

### EMCY codes: CANx

28825

The indications for CANx also apply to each of the CAN interfaces.

| EMCY code object 0x1003 | | Object 0x1001 | Manufactor specific information | | | | | |
|---|---|---|---|---|---|---|---|---|
| Byte 0 [hex] | Byte 1 [hex] | Byte 2 [hex] | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Description |
| 00 | 80 | 11 | -- | -- | -- | -- | -- | CANx monitoring SYNC error (only slave) |
| 00 | 81 | 11 | -- | -- | -- | -- | -- | CANx warning threshold (> 96) |
| 10 | 81 | 11 | -- | -- | -- | -- | -- | CANx receive buffer overrun |
| 11 | 81 | 11 | -- | -- | -- | -- | -- | CANx transmit buffer overrun |
| 30 | 81 | 11 | -- | -- | -- | -- | -- | CANx guard/heartbeat error (only slave) |

## EMCY codes: I/Os, system (standard side)

2668

The following EMCY messages are sent automatically in the following cases:
- as CANopen master: if **CANx_MASTER_EMCY_HANDLER** (→ p. 85) is called cyclically
- as CANopen slave: if **CANx_SLAVE_EMCY_HANDLER** (→ p. 95) 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 | I07...I00 | I15...I08 | | | | Inputs interruption |
| 08 | 21 | 03 | I07...I00 | I15...I08 | | | | Inputs short circuit |
| 10 | 21 | 03 | I07...I00 | I15...I08 | | | | Overcurrent 0…20 mA |
| 00 | 23 | 03 | Q07…Q00 | Q15…Q08 | | | | Outputs interruption |
| 08 | 23 | 03 | Q07…Q00 | Q15…Q08 | | | | Outputs short circuit |
| 00 | 31 | 05 | | | | | | Terminal voltage VBBs |
| 00 | 33 | 05 | | | | | | Terminal voltage VBBo |
| 08 | 33 | 05 | | | | | | Terminal voltage VBBr |
| 00 | 42 | 09 | | | | | | Excess temperature |

## EMCY codes: I/Os, system (extended side)

13095

The following EMCY messages are sent automatically in the following cases:
- as CANopen master: if **CANx_MASTER_EMCY_HANDLER** (→ p. 85) is called cyclically
- as CANopen slave: if **CANx_SLAVE_EMCY_HANDLER** (→ p. 95) is called cyclically

| EMCY code object 0x1003 | | Object 0x1001 | Manufactor specific information | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Byte 0 [hex]** | **Byte 1 [hex]** | **Byte 2 [hex]** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** | **Byte 7** | **Description** |
| 01 | 21 | 03 | I07_E ...I00_E | I15_E ...I08_E | | | | Inputs interruption |
| 09 | 21 | 03 | I07_E ...I00_E | I15_E ...I08_E | | | | Inputs short circuit |
| 11 | 21 | 03 | I07_E ...I00_E | I15_E ...I08_E | | | | Excess current 0…20 mA |
| 01 | 23 | 03 | Q07_E …Q00_E | Q15_E …Q08_E | Q23_E …Q16_E | Q31_E …Q24_E | | Outputs interruption |
| 09 | 23 | 03 | Q07_E …Q00_E | Q15_E …Q08_E | Q23_E …Q16_E | Q31_E …Q24_E | | Outputs short circuit |
| 10 | 33 | 05 | | | | | | Terminal voltage VBB1 |
| 11 | 33 | 05 | | | | | | Terminal voltage VBB2 |
| 12 | 33 | 05 | | | | | | Terminal voltage VBB3 |
| 13 | 33 | 05 | | | | | | Terminal voltage VBB4 |
| 18 | 33 | 05 | | | | | | Supply relays VBBrel |

# 8 Terms and abbreviations

## 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 | $-3.402823466 \bullet 10^{38}$ | $3.402823466 \bullet 10^{38}$ | 32 bits = 4 bytes |
| ULINT | 0 | 18 446 744 073 709 551 615 | 64 Bit = 8 Bytes |
| STRING | | | number of char. + 1 |

## DC

**D**irect **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

**D**ata **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

**L**east **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** (→ p. 247)

## 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

**M**ost **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 = protocol data unit.
The PDU is a term from the →CAN protocol →SAE J1939. It refers to a component of the target address (PDU format 1, connection-oriented) or the group extension (PDU format 2, message-oriented).

## PES

**P**rogrammable **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 = 6 zero bits + 1 bit reserved + 1 bit data page + 8 bit PDU Format (PF) + 8 PDU Specific (PS)
The parameter group number is a term from the →CAN protocol →SAE J1939.

## Pictogram

Pictograms are figurative symbols which convey information by a simplified graphic representation.
(→ chapter **What do the symbols and formats mean?** (→ p. 6))

## 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.
→ chapter **Software modules for the device** (→ p. 39)

### 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?** (→ p. 6))

### 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.

# 9 Index

## A

About this manual....5
Activate the PLC configuration (e.g. CR0033)....55
Address....238
Address assignment and I/O operating modes....222
Addresses / variables of the I/Os....223
Allowable configurations for Q00_MODE...Q15_MODE....62
Allowable configurations for Q00_MODE_E...Q15_MODE_E....62
Allowable configurations for Q16_MODE_E...Q31_MODE_E....63
Analogue inputs....24
Appendix....213
Application program....38
Application software....238
Architecture....238
Automatic data backup....191
Availability of PWM....64
Available memory....17

## B

Baud....238
Binary and PWM outputs....63
Binary inputs....24
Binary outputs....27
Boot loader....238
Bootloader....38
Bootloader state....46
Bus....238

## C

Calculations and conversions in the application program....40
CAN....238
- interfaces and protocols....36

CAN / CANopen
- errors and error handling....211

CAN interfaces....36
CAN stack....238
CANx....76
CANx_BAUDRATE....77
CANx_BUSLOAD....78
CANx_DOWNLOADID....80
CANx_ERRORHANDLER....81
CANx_MASTER_EMCY_HANDLER....86
CANx_MASTER_SEND_EMERGENCY....87
CANx_MASTER_STATUS....89
CANx_RECEIVE....82
CANx_SDO_READ....103
CANx_SDO_WRITE....105
CANx_SLAVE_EMCY_HANDLER....95
CANx_SLAVE_NODEID....96
CANx_SLAVE_SEND_EMERGENCY....97
CANx_SLAVE_SET_PREOP....99
CANx_SLAVE_STATUS....100
CANx_TRANSMIT....84
CHECK_DATA....202
CiA....238
CiA DS 304....239
CiA DS 401....239
CiA DS 402....239
CiA DS 403....239
CiA DS 404....239
CiA DS 405....239
CiA DS 406....239
CiA DS 407....239
Clamp 15....239
COB ID....239
CODESYS....239
CODESYS functions....49
Configuration of the inputs and outputs (default setting)....56
Configurations....51
Configure inputs....59
Configure outputs....62
Configure the hardware filter....60
Configure the software filters of the inputs....60
Configure the software filters of the outputs....63
Connect terminal VBB15 to the ignition switch....18
Control the LED in the application program....33
CONTROL_OCC....163
Copyright....5
Creating application program....41
CSV file....240
Current control with PWM (= PWMi)....64
Cycle time....240

## D

Damping of overshoot....175
Data type....240
DC....240
DEBUG mode....49
DELAY....176
Description.... 76, 77, 78, 80, 81, 82, 84, 86, 87, 89, 95, 96, 97, 99, 100, 103, 105, 108, 109, 111, 113, 115, 117, 120, 121, 122, 123, 125, 127, 130, 133, 135, 136, 138, 142, 144, 146, 148, 150, 153, 156, 157, 160, 163, 165, 168, 171, 173, 176, 177, 179, 181, 183, 185,186, 188, 192, 194, 195, 197, 198, 199, 200, 202, 204, 205, 206, 207, 208
Description....140
Diagnosis....210
- binary outputs (via current measurement)....28, 29
- overload (via current measurement)....29, 30
- short circuit (via current measurement)....29, 30
- wire break (via current measurement)....29, 30

Diagnosis....240
Diagnosis and error handling....210
Dither....241
DLC....241
DRAM....241
DTC....241

## E

ECU....241
EDS-file....241
Embedded software....241
EMC....241
EMCY....242
EMCY codes
- CANx....236
- I/Os, system (extended side)....237
- I/Os, system (standard side)....236

Error flags....236
Error tables....236

Errors
- CAN / CANopen ........ 236

Ethernet ........ 242
EUC ........ 242
Example
- CANx_MASTER_SEND_EMERGENCY ........ 88
- CANx_MASTER_STATUS ........ 92
- CANx_SLAVE_SEND_EMERGENCY ........ 98
- CHECK_DATA ........ 203
- NORM (1) ........ 134
- NORM (2) ........ 134
- NORM_HYDRAULIC ........ 174

## F

Fast inputs ........ 59
FAST_COUNT ........ 138
Fault ........ 210
FB, FUN, PRG in CODESYS ........ 40
Feedback in case of externally supplied outputs ........ 31
FiFo ........ 242
File system ........ 189
Flash memory ........ 189, 242
FLASHREAD ........ 194
FLASH-Speicher ........ 17
FLASHWRITE ........ 195
FRAM ........ 18, 242
FRAM memory ........ 189
FRAMREAD ........ 197
FRAMWRITE ........ 198
FREQUENCY ........ 140
FREQUENCY_PERIOD ........ 142
Function configuration in general ........ 56
Function configuration of the inputs and outputs ........ 58
Function elements
- adapting analogue values ........ 132
- CAN layer 2 ........ 75
- CANopen master ........ 85
- CANopen SDOs ........ 102
- CANopen slave ........ 94
- controllers ........ 175
- counter functions for frequency and period measurement ........ 137
- data access and data check ........ 201
- device temperature ........ 187
- hydraulic control ........ 162
- measuring / setting of time ........ 184
- Optimising the PLC cycle via processing interrupts ........ 124
- processing input values ........ 129
- PWM functions ........ 152
- SAE J1939 ........ 107
- saving, reading and converting data in the memory ........ 189
- serial interface ........ 119
- software reset ........ 182

## G

GET_IDENTITY ........ 204
GET_IDENTITY_EIOS ........ 205

## H

Hardware description ........ 15
Hardware structure ........ 16
Heartbeat ........ 243
History of the instructions (CR0232) ........ 7
HMI ........ 243
How is this documentation structured? ........ 6

## I

ID ........ 243
IEC 61131 ........ 243
IEC user cycle ........ 243
ifm function elements ........ 69
ifm function elements for the device CR0232 ........ 74
ifm libraries for the device CR0232 ........ 70
INC_ENCODER ........ 144
Information concerning the device ........ 14
INIT state (Reset) ........ 46
Input group I00...I15 ........ 25
Input group I00_E...I15_E ........ 26
INPUT_ANALOG ........ 130
Inputs
- addresses and variables (extended side) (16 inputs) ........ 224
- addresses and variables (standard side) (16 inputs) ........ 223
- operating modes (extended side) (16 inputs) ........ 232
- operating modes (standard side) (16 inputs) ........ 231

Inputs (technology) ........ 24
Instructions ........ 243
Intended use ........ 243
Interface description ........ 35
Internal structure parameters ........ 91
IP address ........ 243
ISO 11898 ........ 243
ISO 11992 ........ 244
ISO 16845 ........ 244

## J

J1939 ........ 244
J1939_x ........ 108
J1939_x_GLOBAL_REQUEST ........ 109
J1939_x_RECEIVE ........ 111
J1939_x_RESPONSE ........ 113
J1939_x_SPECIFIC_REQUEST ........ 115
J1939_x_TRANSMIT ........ 117
JOYSTICK_0 ........ 165
JOYSTICK_1 ........ 168
JOYSTICK_2 ........ 171

## L

Latching ........ 19
LED ........ 244
Libraries ........ 39
Library ifm_CR0232_CANopenxMaster_Vxxyyzz.LIB ........ 72
Library ifm_CR0232_CANopenxSlave_Vxxyyzz.LIB ........ 72
Library ifm_CR0232_J1939_Vxxyyzz.LIB ........ 73
Library ifm_CR0232_V010003.LIB ........ 70
Library ifm_hydraulic_32bit_Vxxyyzz.LIB ........ 73
Link ........ 244
LSB ........ 244

## M

MAC-ID ........ 244
Manual data storage ........ 193
Master ........ 244
MEMCPY ........ 199
MEMORY_RETAIN_PARAM ........ 192

- MEMSET 200
- Misuse 244
- MMI 244
- Monitoring concept 20
- Monitoring of the supply voltages VBBx 20
- MRAM 245
- MSB 245

## N

- Network variables 67
- NMT 245
- Node 245
- Node Guarding 245
- NORM 133
- NORM_DINT 135
- NORM_HYDRAULIC 173
- NORM_REAL 136
- Note on wiring 31
- Note the cycle time! 41
- Notes
  - serial number 11
  - TEST inputs 12, 48

## O

- Obj / object 245
- Object directory 245
- OBV 245
- OPC 245
- Operating modes 47
- Operating principle of the delayed switch-off 18
- Operating principle of the monitoring concept 22
- Operating states 44
  - application program is available 46
  - application program is not available 45
  - runtime system is not available 44
- Operational 245
- Output group Q0 (Q00...15) 27
- Output group Q00_E...Q15_E 29
- Output group Q16_E...Q31_E 30
- OUTPUT_BRIDGE 153
- OUTPUT_CURRENT 156
- OUTPUT_CURRENT_CONTROL 157
- Outputs
  - addresses and variables (extended side) (32 outputs) 228
  - addresses and variables (standard side) (16 outputs) 226
  - operating modes (extended side) (32 outputs) 234
  - operating modes (standard side) (16 outputs) 233
- Outputs (technology) 27
- Outputs Q00...Q15
  - permitted operating modes 233
- Outputs Q00_E...Q31_E
  - permitted operating modes 234
- Overview
  - documentation modules for 5

## P

- Parameters of the inputs 76, 77, 78, 80, 81, 82, 84, 86, 87, 89, 95, 96, 97, 99, 100, 103, 105, 108, 110, 112, 113, 115, 117, 121, 122, 123, 126, 128, 130, 133, 135, 136, 138, 141, 143, 144, 146, 149, 150, 154, 156, 158, 161, 163, 166, 169, 171, 173, 176, 177, 179, 181, 183, 188,192, 194, 195, 197, 198, 199, 200, 202, 204, 205, 206, 207, 208
- Parameters of the outputs 78, 83, 84, 86, 90, 95, 101, 104, 106, 110, 112, 114, 116, 118, 120, 121, 131, 133, 135, 136, 139, 141, 143, 145, 147, 149, 151, 155, 156, 159, 164, 167, 170, 172, 174, 176, 177, 180, 181, 185, 186, 188, 202, 204, 205
- PC card 246
- PCMCIA card 246
- PDM 246
- PDO 246
- PDU 246
- Performance limits of the device 49
- PERIOD 146
- PERIOD_RATIO 148
- PES 246
- PGN 246
- PHASE 150
- Pictogram 246
- PID controller 246
- PID1 177
- PID2 179
- PLC configuration 247
- Please note! 10
- Possible operating modes inputs/outputs 231
- Pre-Op 247
- Principle block diagram 17
- Principle of the H-bridge 153
- Process image 247
- Program example to CAN1_MASTER_STATUS 92
- Programming notes for CODESYS projects 40
- PT1 181
- PWM 247
- PWM outputs 27
- PWM1000 160

## R

- ratiometric 247
- RAW-CAN 247
- Reaction in case of an error 210
- Recommended setting 180
- Recommended settings 178
- Reference voltage output 22
- Reinstall the runtime system 52
- Relay
  - important notes! 211
- Relays 16
  - important notes! 19
- remanent 247
- Response to system errors 211
- Retain variables 66
- ro 247
- RTC 248
- RUN state 47
- Runtime system 38, 248
- rw 248

## S

- SAE J1939 248
- Safety instructions 10
- Safety instructions about Reed relays 31, 59
- Save boot project 42
- SD card 248
- SDO 248

Self-test.....248
Serial interface.....35
SERIAL_MODE.....49
SERIAL_PENDING.....120
SERIAL_RX.....121
SERIAL_SETUP.....122
SERIAL_TX.....123
Set up the programming system.....54
Set up the programming system manually.....55
Set up the programming system via templates.....56
Set up the runtime system.....52
Set up the target.....55
SET_DEBUG.....206
SET_IDENTITY.....207
SET_INTERRUPT_I.....125
SET_INTERRUPT_XMS.....127
SET_PASSWORD.....208
Setting control.....175
Setting rule for a controller.....175
Slave.....249
Slave information.....92
SOFTRESET.....183
Software description.....37
Software modules for the device.....38
SRAM.....18
Start conditions.....16
Start-up behaviour of the controller.....11
Status LED.....32
STOP state.....47
stopped.....249
Storage types for data backup.....189
Structure Emergency_Message.....93
Structure node status.....93
Structure of CANx_EMERGENCY_MESSAGE.....91
Structure of CANx_NODE_STATE.....91
Symbols.....249
System description.....14
System flags.....214
- 16 inputs and 16 outputs (standard side).....219
- 16 inputs and 32 outputs (extended side).....220
- CAN.....214
- error flags (extended side).....216
- error flags (standard side).....215
- SAE-J1939.....215
- status LED (extended side).....218
- status LED (standard side).....217
- voltages (extended side).....219
- voltages (standard side).....218

SYSTEM STOP state.....47
System variable.....249
System variables.....56

## T

Target.....249
TCP.....249
TEMPERATURE.....188
Template.....249
Terminal voltage VBBs falls below the limit value of 10 V.....20
Terminal voltage VBBx falls below the limit value of 5.25 V.....20
TEST mode.....47
TIMER_READ.....185
TIMER_READ_US.....186

## U

UDP.....249
Update the runtime system.....53
USB interface.....35
Use, intended.....249
Using ifm downloader.....43
Using ifm maintenance tool.....43

## V

Variables.....66
Verify the installation.....53

## W

Watchdog.....250
Watchdog behaviour.....49
What do the symbols and formats mean?.....6
What previous knowledge is required?.....11