ifm electronic

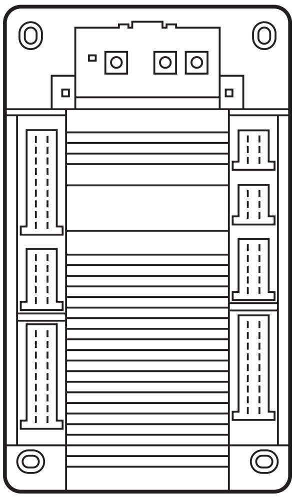
System Manual
CabinetController

ecomat100®
CR0303

CoDeSys® V2.3

Target V05

English



7390669 / 05   2011-05-25

Contents

1 About this manual 7

1.1 What do the symbols and formats mean? 7
1.2 How is this manual structured? 8
1.3 Changes of the manual (CA) 9

2 Safety instructions 10

2.1 Important! 10
2.2 What previous knowledge is required? 11

3 System description 12

3.1 Information concerning the device 12
3.2 Information concerning the software 12
3.3 PLC configuration 14

4 Operating states and operating system 15

4.1 Operating states 15
  4.1.1 INIT state (Reset) 15
  4.1.2 STOP state 15
  4.1.3 Fatal error 16
  4.1.4 RUN state 16
  4.1.5 No operating system 16
4.2 Status LED 16
  4.2.1 Setting the LED via application program 17
4.3 Load the operating system 18
4.4 Operating modes 19
  4.4.1 TEST mode 19
  4.4.2 SERIAL_MODE 19
  4.4.3 DEBUG mode 19

5 Configurations 20

5.1 Set up programming system 20
  5.1.1 Set up programming system manually 20
  5.1.2 Set up programming system via templates 24
  5.1.3 ifm demo programs 34
5.2 Function configuration of the inputs and outputs 38
  5.2.1 Configure inputs 38
  5.2.2 Configure outputs 43
5.3 Hints to wiring diagrams 45

6 Limitations and programming notes 47

6.1 Limits of the device 47
6.1.1 CPU frequency 47
6.1.2 Above-average stress....48
6.1.3 Watchdog behaviour....49
6.1.4 Available memory 50
6.2 Programming notes for CoDeSys projects 51
6.2.1 FB, FUN, PRG in CoDeSys....51
6.2.2 Note the cycle time! 52
6.2.3 Creating application program 52
6.2.4 Save....54
6.2.5 Using ifm downloader 54
6.2.6 Certification and distribution of the safety-related software....55
6.2.7 Changing the safety-relevant software after certification 55

7 Error codes and diagnostic information 56

7.1 Overview 56
7.2 Response to the system error....57
7.2.1 Notes on devices with monitoring relay....57
7.2.2 Example process for response to a system error....58

8 Using CAN 59

8.1 General about CAN....59
8.1.1 Topology 60
8.1.2 CAN interfaces....61
8.1.3 Available CAN interfaces and CAN protocols 61
8.1.4 System configuration 63
8.2 Physical connection of CAN 65
8.2.1 Network structure....65
8.2.2 CAN bus level 66
8.2.3 CAN bus level according to ISO 11992-1....66
8.2.4 Bus cable length 67
8.2.5 Wire cross-sections 68
8.3 Exchange of CAN data 69
8.3.1 Hints....70
8.3.2 Data reception 72
8.3.3 Data transmission....72
8.4 Description of the CAN standard program units....73
8.4.1 CAN1_BAUDRATE....75
8.4.2 CAN1_DOWNLOADID 77
8.4.3 CAN1_EXT 79
8.4.4 CAN1_EXT_TRANSMIT....81
8.4.5 CAN1_EXT_RECEIVE 83
CAN1_EXT_ERRORHANDLER 85
8.4.6 CAN2 86
8.4.7 CANx_TRANSMIT....88
8.4.8 CANx_RECEIVE....90
8.4.9 CANx_RECEIVE_RANGE....92
8.4.10 CANx_EXT_RECEIVE_ALL 95
8.4.11 CANx_ERRORHANDLER 97

8.5 CAN units acc. to SAE J1939 99
8.5.1 CAN for the drive engineering 100
8.5.2 Units for SAE J1939 104
8.6 ifm CANopen libraries 117
8.6.1 Technical about CANopen 118
8.6.2 Libraries for CANopen 156
8.7 CAN errors and error handling 182
8.7.1 CAN errors 182
8.7.2 Structure of an EMCY message 185
8.7.3 Overview CANopen error codes 187

9 In-/output functions 190

9.1 Processing input values 190
9.1.1 INPUT_ANALOG 191
9.1.2 INPUT_VOLTAGE 193
9.1.3 INPUT_CURRENT 194
9.2 Adapting analogue values 195
9.2.1 NORM 196
9.3 Counter functions for frequency and period measurement 198
9.3.1 Applications 198
9.3.2 Use as digital inputs 199
9.4 PWM functions 213
9.4.1 Availability of PWM 213
9.4.2 PWM signal processing 214
9.4.3 Hydraulic control in PWMi 227
9.5 Controller functions 245
9.5.1 General 245
9.5.2 Setting rule for a controller 247
9.5.3 Functions for controllers 248

10 Communication via interfaces 259

10.1 Use of the serial interface 259
10.1.1 SERIAL_SETUP 260
10.1.2 SERIAL_TX 262
10.1.3 SERIAL_RX 263
10.1.4 SERIAL_PENDING 265

11 Managing the data 266

11.1 Software reset 266
11.1.1 SOFTRESET 267
11.2 Reading / writing the system time 268
11.2.1 TIMER_READ 269
11.2.2 TIMER_READ_US 270
11.3 Saving, reading and converting data in the memory 271
11.3.1 Automatic data backup 271
11.3.2 Manual data storage 272
11.4 Data access and data check 282
11.4.1 SET_DEBUG 283
11.4.2 SET_IDENTITY 284
11.4.3 GET_IDENTITY 286
11.4.4 SET_PASSWORD 288
11.4.5 CHECK_DATA 290

12 Optimising the PLC cycle 292

12.1 Processing interrupts 292
  12.1.1 SET_INTERRUPT_XMS 293
  12.1.2 SET_INTERRUPT_I 296

13 Annex 299

13.1 Address assignment and I/O operating modes 299
  13.1.1 Address assignment inputs / outputs 300
  13.1.2 Addresses / variables of the I/Os 302
  13.1.3 Possible operating modes inputs / outputs 304
13.2 System flags 305
13.3 CANopen tables 306
  13.3.1 IDs (addresses) in CANopen 306
  13.3.2 Structure of CANopen messages 307
  13.3.3 Bootup messsage 311
  13.3.4 Network management (NMT) 312
  13.3.5 CANopen error code 316
13.4 Overview of the files and libraries used 319
  13.4.1 Installation of the files and libraries 319
  13.4.2 General overview 320
  13.4.3 What are the individual files and libraries used for? 322

14 Glossary of Terms 329

15 Index 344

1 About this manual

Contents

What do the symbols and formats mean? 7
How is this manual structured? 8
Changes of the manual (CA)....9

202

In the additional "Programming Manual for CoDeSys V2.3" you will obtain more details about the use of the programming system "CoDeSys for Automation Alliance". This manual can be downloaded free of charge from **ifm's** website:
a) → www.ifm.com > select your country > [Service] > [Download] > [Control systems]
b) → **ecomat*mobile*** DVD "Software, tools and documentation"

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

1.1 What do the symbols and formats mean?

203

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

⚠ DANGER

Death or serious irreversible injuries are to be expected.

⚠ WARNING

Death or serious irreversible injuries are possible.

⚠ CAUTION

Slight reversible injuries are possible.

NOTICE

Property damage is to be expected or possible.

❗ NOTE

Important notes on faults and errors.

ℹ Info

Further hints.

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

1.2 How is this manual structured?

204

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

How to use this documentation:

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

In case of malfunctions or uncertainties please contact the manufacturer at:
→ www.ifm.com > select your country > [Contact].

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

We reserve the right to make alterations which can result in a change of contents of the documentation. You can find the current version on **ifm's** website at:
→ www.ifm.com > select your country > [Service] > [Download] > [Control systems]

1.3 Changes of the manual (CA)

9181

What has been changed in this manual? An Overview:

Date	Theme	Change
2010-09-09	PID2 (FB)	parameters of the inputs corrected
2010-11-10	Terminating resistors	correction in topic 1244
2011-02-14	TIMER_READ_US (FB)	conversion of max. counter value corrected
2011-04-05	Memory POUs FRAMREAD, FRAMWRITE, FLASHREAD, FLASHWRITE	permitted values of the parameters SRC, LEN, DST
2011-04-13	CANopen overview	new: CANopen tables in the annex
2011-04-14	CR0303 several corrections:	- device has an own hydraulic library - some system flags do not exist - IEC addresses of in- and outputs - configuration of the inputs - set the status LED in the application program
2011-05-24	CR0303: memory POUs FRAMREAD, FRAMWRITE	permitted values of the parameters SRC, DST corrected

2 Safety instructions

Contents

Important! 10
What previous knowledge is required? 11

213

2.1 Important!

214

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

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

⚠ WARNING

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

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

ATTENTION

The driver module of the serial interface can be damaged!

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

- ► Do not disconnect the serial interface while live.

Start-up behaviour of the controller

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

A restart can, for example, be caused by:

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

2.2 What previous knowledge is required?

215

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

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

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

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

Adhere to the safety instructions.

3 System description

Contents

Information concerning the device 12
Information concerning the software 12
PLC configuration 14

975

3.1 Information concerning the device

1310

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

- CabinetController: CR0301, CR0302 (with restrictions)
- CabinetController: CR0303

3.2 Information concerning the software

2730

The device operates with CoDeSys, version 2.3.9.1 or higher.

In the "programming manual CoDeSys 2.3" you will find more details about how to use the programming system "CoDeSys for Automation Alliance". This manual can be downloaded free of charge from **ifm's** website at:
→ www.ifm.com > select your country > [Service] > [Download] > [Control systems]
→ **ecomat*mobile*** DVD "Software, tools and documentation"

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

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

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

! NOTE

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

- operating system (`CRnnnn_Vxxyyzz.H86` / `CRnnnn_Vxxyyzz.RESX`)
- PLC configuration (`CRnnnn_Vxx.CFG`)
- device library (`ifm_CRnnnn_Vxxyyzz.LIB`)
- and the further files (→ chapter Overview of the files and libraries used (→ page 319))

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

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

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

IMPORTANT: the following files must also be loaded:

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

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

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

⚠ WARNING

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

3.3 PLC configuration

1797

The control system **ecomat*mobile*** is a device concept for series use. This means that the devices can be configured in an optimum manner for the applications. If necessary, special functions and hardware solutions can be implemented. In addition, the current version of the **ecomat*mobile*** software can be downloaded from our website at: www.ifm.com.
→ Setup the target (→ page 21)

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

4 Operating states and operating system

Contents

Operating states 15
Status LED 16
Load the operating system 18
Operating modes 19

1074

4.1 Operating states

1075

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

4.1.1 INIT state (Reset)

1076

This state is passed through after every power on reset:

> The operating system is initialised.

> Various checks are carried out, e.g. waiting for correctly power supply voltage.

> This temporary state is replaced by the RUN or STOP state.

> The LED lights yellow.

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

- RUN
- FATAL ERROR
- STOP

4.1.2 STOP state

1078

This state is reached in the following cases:

- From the INIT state if no program is loaded
- From the RUN state if:
  - the STOP command is sent via the interface
  - AND: operating mode = Test (→ chapter TEST mode (→ page 19))

4.1.3 Fatal error

1079

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

> The LED lights red.

4.1.4 RUN state

1077

This state is reached in the following cases:

- From the INIT state (autostart)
- From the STOP state by the RUN command
  - only for the operating mode = Test (→ chapter TEST mode (→ page 19))

4.1.5 No operating system

1080

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

> The LED flashes green (quickly).

4.2 Status LED

1430

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

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

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

4.2.1 Setting the LED via application program

9989

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

LED	LED colour for "active" (= on)
LED_X	LED colour for "pause" (= off or other colour)
---	colour constant from the data structure "LED colour". Allowed: LED_GREEN LED_BLUE LED_RED LED_WHITE LED_MAGENTA LED_CYAN LED_YELLOW LED_ORANGE LED_BLACK (= LED off)
LED_MODE	flashing frequency from the data structure "LED_MODES". Allowed: LED_5HZ LED_2HZ LED_1HZ LED_05HZ (= 0.5 Hz) LED_0HZ (= constant)

NOTE

► In the application do NOT use the LED colour RED.

> In case of an error the LED colour RED is set by the operating system.
BUT: If the colours and/or flashing modes are changed by the application program, the above-mentioned table (default setting) is no longer valid.

4.3 Load the operating system

2733

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

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

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

The operating system is provided with this documentation on a separate data carrier. In addition, the current version can be downloaded from the website of **ifm electronic gmbh** at:
→ www.ifm.com > select your country > [Service] > [Download] > [Control systems]

NOTE

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

- operating system (`CRnnnn_Vxxyyzz.H86` / `CRnnnn_Vxxyyzz.RESX`)
- PLC configuration (`CRnnnn_Vxx.CFG`)
- device library (`ifm_CRnnnn_Vxxyyzz.LIB`)
- and the further files (→ chapter Overview of the files and libraries used (→ page 319))

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

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

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

IMPORTANT: the following files must also be loaded:

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

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

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

4.4 Operating modes

1083

Independent of the operating states the **ecomat*mobile*** controller can be operated in different modes. The corresponding control bits can be set and reset with the programming software CoDeSys (window: Global Variables) via the application software or in test mode (→ chapter TEST mode (→ page 19)).

4.4.1 TEST mode

1084

This operating mode is reached by applying a high level (supply voltage) to the test input (→ installation instructions, chapter "wiring"). The **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. Moreover the software can only be downloaded to the controller in this operating state.

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

NOTICE

Loss of the stored software possible!

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

4.4.2 SERIAL_MODE

1085

The serial interface is available for the exchange of data in the application. Debugging the application software is then only possible via the CAN interface.
For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

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

→ chapter Use of the serial interface (→ page 259)

4.4.3 DEBUG mode

1086

If the input DEBUG of SET_DEBUG (→ page 283) is set to TRUE, the programming system or the downloader, for example, can communicate with the controller and execute system commands (e.g. for service functions via the GSM modem CANremote).

In this operating mode a software download is not possible because the test input (→ chapter TEST mode (→ page 19)) is not connected to supply voltage.

5 Configurations

Contents

Set up programming system 20
Function configuration of the inputs and outputs 38
Hints to wiring diagrams 45

3615

5.1 Set up programming system

Contents

Set up programming system manually 20
Set up programming system via templates 24
ifm demo programs 34

3968

5.1.1 Set up programming system manually

Contents

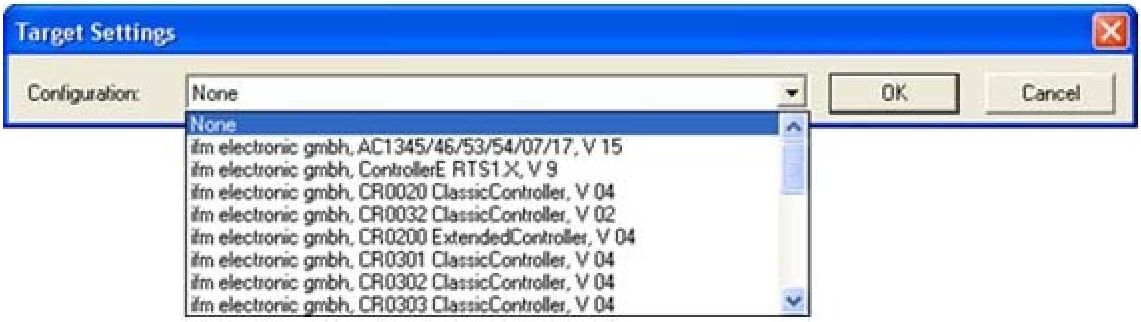
Setup the target 21
Activating the PLC configuration 22

3963

Setup the target

2687

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



Configuration: None

Target Settings
OK
Cancel
None
ifm electronic gmbh, AC1345/46/53/54/07/17, V 15
ifm electronic gmbh, ControllerE RTS1.X, V 9
ifm electronic gmbh, CR0020 ClassicController, V 04
ifm electronic gmbh, CR0032 ClassicController, V 02
ifm electronic gmbh, CR0200 ExtendedController, V 04
ifm electronic gmbh, CR0301 ClassicController, V 04
ifm electronic gmbh, CR0302 ClassicController, V 04
ifm electronic gmbh, CR0303 ClassicController, V 04

Figure: Target system settings

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

NOTE

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

- operating system (`CRnnnn_Vxxyyzz.H86` / `CRnnnn_Vxxyyzz.RESX`)
- PLC configuration (`CRnnnn_Vxx.CFG`)
- device library (`ifm_CRnnnn_Vxxyyzz.LIB`)
- and the further files (→ chapter Overview of the files and libraries used (→ page 319))

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

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

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

IMPORTANT: the following files must also be loaded:

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

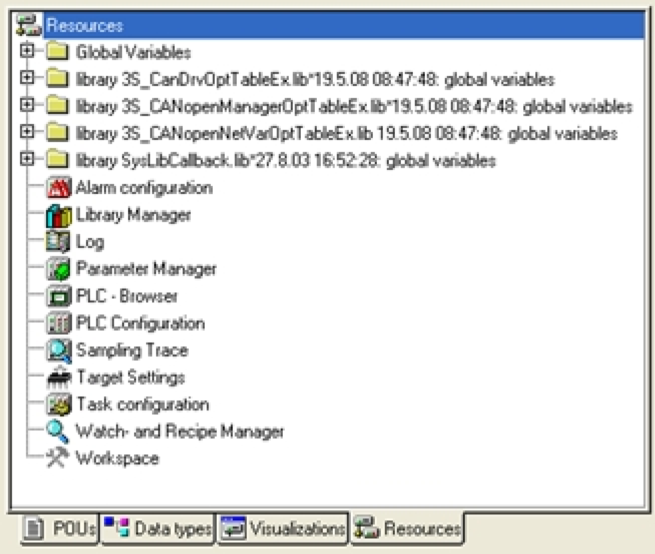
Activating the PLC configuration

2688

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

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

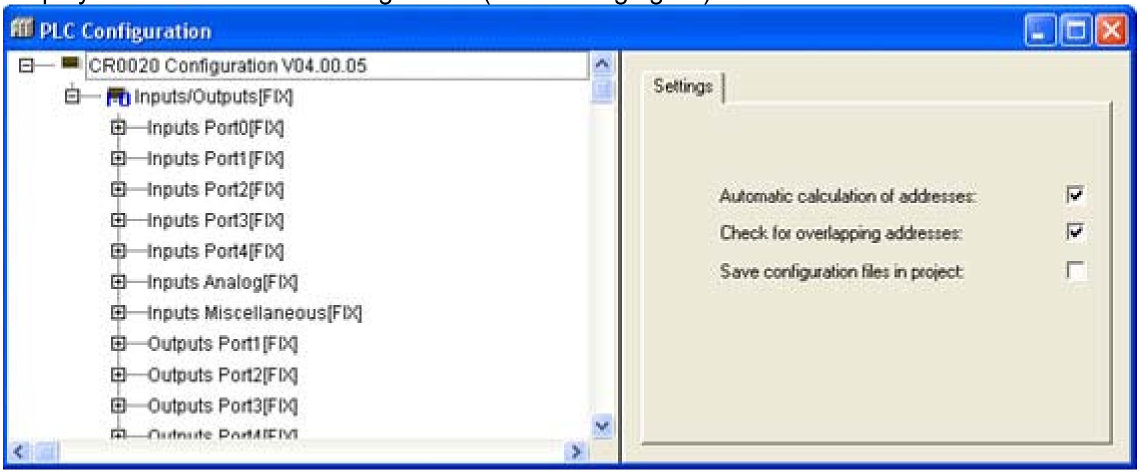
► Click on the tab [Resources] in CoDeSys:



Resources
- Global Variables
- library 3S_CanDrvOptTableEx.lib*19.5.08 08:47:48: global variables
- library 3S_CANopenManagerOptTableEx.lib*19.5.08 08:47:48: global variables
- library 3S_CANopenNetVarOptTableEx.lib 19.5.08 08:47:48: global variables
- library SysLibCallback.lib*27.8.03 16:52:28: global variables
- Alarm configuration
- Library Manager
- Log
- Parameter Manager
- PLC - Browser
- PLC Configuration
- Sampling Trace
- Target Settings
- Task configuration
- Watch- and Recipe Manager
- Workspace

► Double-click on [PLC Configuration] in the left column.

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



PLC Configuration

- CR0020 Configuration V04.00.05
  - Inputs/Outputs[FIX]
    - Inputs Port0[FIX]
    - Inputs Port1[FIX]
    - Inputs Port2[FIX]
    - Inputs Port3[FIX]
    - Inputs Port4[FIX]
    - Inputs Analog[FIX]
    - Inputs Miscellaneous[FIX]
    - Outputs Port1[FIX]
    - Outputs Port2[FIX]
    - Outputs Port3[FIX]
    - Outputs Port4[FIX]

Settings

Automatic calculation of addresses: ☑

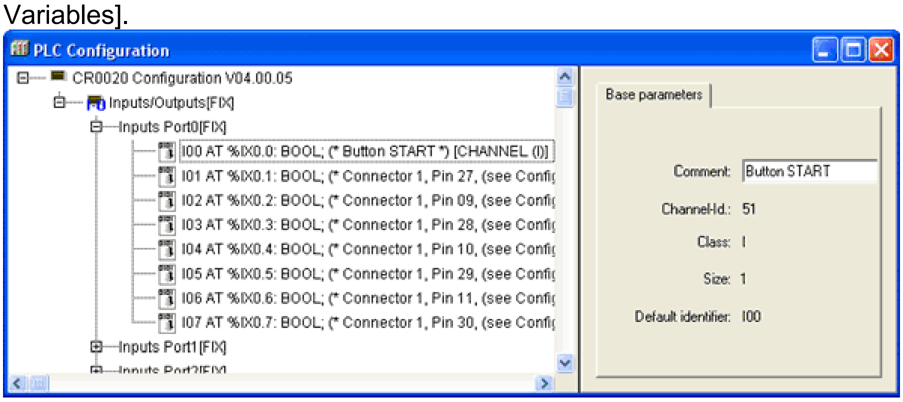
Check for overlapping addresses: ☑

Save configuration files in project: ☐

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

- All important system and error flags
  Depending on the application and the application program, these flags must be processed and evaluated. Access is made via their symbolic names.

- The structure of the inputs and outputs
  These can be directly symbolically designated (highly recommended!) in the window [PLC Configuration] (example → figure below) and are available in the whole project as [Global Variables].



```
CR0020 Configuration V04.00.05
  Inputs/Outputs[FIX]
    Inputs Port0[FIX]
      I00 AT %IX0.0: BOOL; (* Button START *) [CHANNEL (I)]
      I01 AT %IX0.1: BOOL; (* Connector 1, Pin 27, (see Confi
      I02 AT %IX0.2: BOOL; (* Connector 1, Pin 09, (see Confi
      I03 AT %IX0.3: BOOL; (* Connector 1, Pin 28, (see Confi
      I04 AT %IX0.4: BOOL; (* Connector 1, Pin 10, (see Confi
      I05 AT %IX0.5: BOOL; (* Connector 1, Pin 29, (see Confi
      I06 AT %IX0.6: BOOL; (* Connector 1, Pin 11, (see Confi
      I07 AT %IX0.7: BOOL; (* Connector 1, Pin 30, (see Confi
    Inputs Port1[FIX]
    Inputs Port2[FIX]
```

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

5.1.2 Set up programming system via templates

Contents

About the ifm templates 27
Supplement project with further functions 31

3977

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

NOTE

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

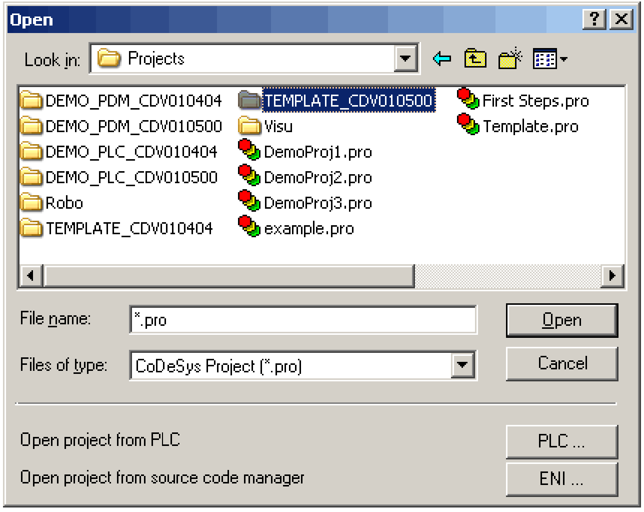
► Open the requested template in CoDeSys via:
[File] > [New from template…]

> CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page 24)

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

► In the CoDeSys menu select: [File] > [New from template...]

► Select directory of the current CD, e.g. `...\Projects\TEMPLATE_CDV010500`:

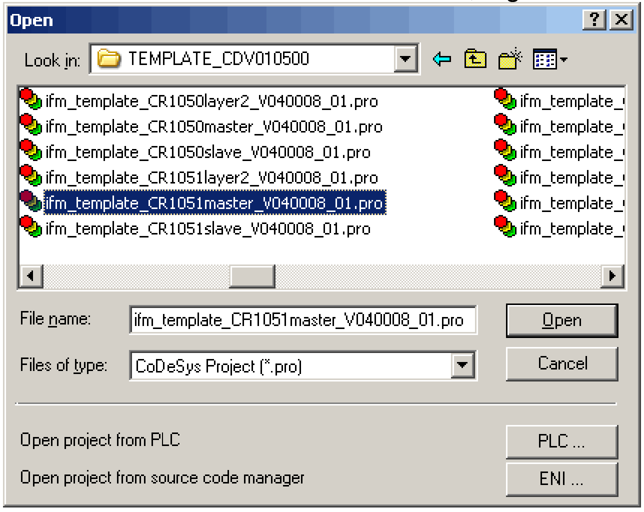


Look in: Projects

File name: *.pro

Files of type: CoDeSys Project (*.pro)

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



Open

Look in: TEMPLATE_CDV010500

ifm_template_CR1050layer2_V040008_01.pro
ifm_template_CR1050master_V040008_01.pro
ifm_template_CR1050slave_V040008_01.pro
ifm_template_CR1051layer2_V040008_01.pro
ifm_template_CR1051master_V040008_01.pro
ifm_template_CR1051slave_V040008_01.pro

File name: ifm_template_CR1051master_V040008_01.pro

Files of type: CoDeSys Project (*.pro)

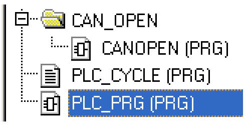
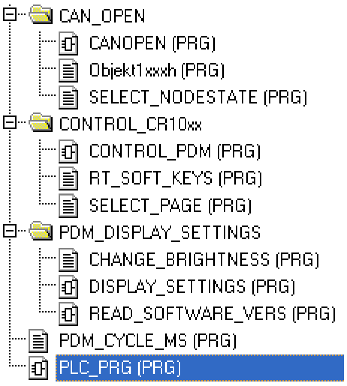
Open project from PLC

Open project from source code manager

► How is the CAN network organised?
Do you want to work on layer 2 basis or is there a master with several slaves (for CANopen)?
(Here an example: CANopen-Slave, → figure above)

► Confirm the selection with [Open].

> A new CoDeSys project is generated with the following folder structure (left):

Example for CR2500 as CANopen master:	Another example for CR1051 as CANopen slave:
 CAN_OPEN CANOPEN (PRG) PLC_CYCLE (PRG) PLC_PRG (PRG)	 CAN_OPEN CANOPEN (PRG) Objekt1xxxh (PRG) SELECT_NODESTATE (PRG) CONTROL_CR10xx CONTROL_PDM (PRG) RT_SOFT_KEYS (PRG) SELECT_PAGE (PRG) PDM_DISPLAY_SETTINGS CHANGE_BRIGHTNESS (PRG) DISPLAY_SETTINGS (PRG) READ_SOFTWARE_VERS (PRG) PDM_CYCLE_MS (PRG) PLC_PRG (PRG)

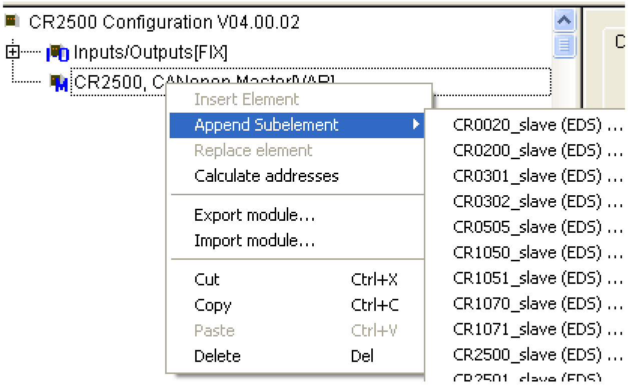
(via the folder structures in templates → section About the ifm templates (→ page 27)).

► Save the new project with [file] > [Save as...], and define suitable directory and project name.

► Configuration of the CAN network in the project:
Double click the element [PLC configuration] above the tabulator [resources] in the CoDeSys project.

► **Right** mouse click in the entry [CR2500, CANopen Master]

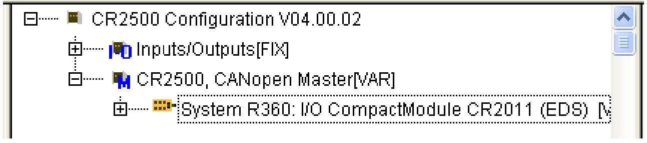
► Click in the context menu [Append subelement]:



> A list of all available EDS files appears in the extended context menu.

► Select requested element, e.g. "System R360": I/O CompactModule CR2011 (EDS)".
The EDS files are in directory `C:\...\CoDeSys V...\Library\PLCConf\`.

> The window [PLC configuration] changes as follows:


CR2500 Configuration V04.00.02
Inputs/Outputs[FIX]
CR2500, CANopen Master[VAR]
System R360: I/O CompactModule CR2011 (EDS)

► Set CAN parameters, PDO mapping and SDOs for the entered slave according to the requirements. Note: Better deselect [Create all SDOs].

► With further slaves proceed as described above.

► Save the project!

This should be a sufficient description of your project. You want to supplement this project with further elements and functions?
→ chapter Supplement project with further functions (→ page 31)

About the ifm templates

Contents

Folder structure in general 27
Programs and functions in the folders of the templates....28
Structure of the visualisations in the templates....30

3981

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

- `ifm_template_CRnnnnLayer2_Vxxyyzz.pro`
  for the operation of the unit with CAN layer 2
- `ifm_template_CRnnnnMaster_Vxxyyzz.pro`
  for the operation of the unit as CANopen master
- `ifm_template_CRnnnnSlave_Vxxyyzz.pro`
  for the operation of the unit as CANopen slave

The templates described here are for:
- CoDeSys from version 2.3.9.6
- on the **ecomat*mobile*** DVD "Software, tools and documentation" from version 010500

The templates all have the same structures.

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

Folder structure in general

3978

The POUs are sorted in the following folders:

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

Programs and functions in the folders of the templates

3980

The above folders contain the following programs and function blocks (all = POUs):

POUs in the folder CAN_OPEN	Description
CANopen	for Controller and PDM, CAN operation as master: Contains the following parameterised POUs: - CAN1_MASTER_EMCY_HANDLER (→ CANx_MASTER_EMCY_HANDLER (→ page 157)), - CAN1_MASTER_STATUS (→ CANx_MASTER_STATUS (→ page 162)), - SELECT_NODESTATE (→ down).
CANopen	for Controller and PDM, CAN operation as slave: Contains the following parameterised POUs: - CAN1_SLAVE_EMCY_HANDLER (→ CANx_SLAVE_EMCY_HANDLER (→ page 169)), - CAN1_SLAVE_STATUS (→ CANx_SLAVE_STATUS (→ page 174)), - SELECT_NODESTATE (→ down).
Objekt1xxxh	for Controller and PDM, CAN operation as slave: Contains the values [STRING] for the following parameters: - ManufacturerDeviceName, e.g.: 'CR1051' - ManufacturerHardwareVersion, e.g.: 'HW_Ver 1.0' - ManufacturerSoftwareVersion, e.g.: 'SW_Ver 1.0'
SELECT_NODESTATE	for PDM, CAN operation as master or slave: Converts the value of the node status [BYTE] into the corresponding text [STRING]: 4 → 'STOPPED' 5 → 'OPERATIONAL' 127 → 'PRE-OPERATIONAL'

POUs in the folder I_O_CONFIGURATION	Description
CONF_IO_CRnnnn	for Controller, CAN operation with layer 2 or as master or slave: Parameterises the operating modes of the inputs and outputs.

POUs in the folder PDM_COM_LAYER2	Description
PLC_TO_PDM	for Controller, CAN operation with layer 2 or as slave: Organises the communication from the Controller to the PDM: - monitors the transmission time, - transmits control data for image change, input values etc.
TO_PDM	for Controller, CAN operation with layer 2 or as slave: Organises the signals for LEDs and keys between Controller and PDM. Contains the following parameterised POUs: - PACK (→ 3S), - PLC_TO_PDM (→ up), - UNPACK (→ 3S).

POUs in the folder CONTROL_CR10nn	**Description**
CONTROL_PDM	for PDM, CAN operation with layer 2 or as master or slave: Organises the image control in the PDM. Contains the following parameterised POUs: - PACK (→ 3S), - PDM_MAIN_MAPPER (→ PDM_MAIN_MAPPER), - PDM_PAGECONTROL (→ PDM_PAGECONTROL), - PDM_TO_PLC (→ down), - SELECT_PAGE (→ down).
PDM_TO_PLC	for PDM, CAN operation with layer 2: Organises the communication from the PDM to the Controller: - monitors the transmission time, - transmits control data for image change, input values etc. Contains the following parameterised POUs: - CAN_1_TRANSMIT (→ CAN_x_TRANSMIT), - CAN_1_RECEIVE (→ CAN_x_RECEIVE).
RT_SOFT_KEYS	for PDM, CAN operation with layer 2 or as master or slave: Provides the rising edges of the (virtual) key signals in the PDM. As many variables as desired (as virtual keys) can be mapped on the global variable SoftKeyGlobal when e.g. a program part is to be copied from a CR1050 to a CR1055. It contains only the keys F1...F3: → For the virtual keys F4...F6 variables have to be created. Map these self-created variables on the global softkeys. Work only with the global softkeys in the program. Advantage: Adaptations are only required in **one** place.
SELECT_PAGE	for PDM, CAN operation with layer 2 or as master or slave: Organises the selection of the visualisations. Contains the following parameterised POUs: - RT_SOFT_KEYS (→ up).

POUs in the folder PDM_DISPLAY_SETTINGS	**Description**
CHANGE_BRIGHTNESS	for PDM, CAN operation with layer 2 or as master or slave: Organises brightness / contrast of the monitor.
DISPLAY_SETTINGS	for PDM, CAN operation with layer 2 or as master or slave: Sets the real-time clock, controls brightness / contrast of the monitor, shows the software version. Contains the following parameterised POUs: - CHANGE_BRIGHTNESS (→ up), - CurTimeEx (→ 3S), - PDM_SET_RTC (→ PDM_SET_RTC), - READ_SOFTWARE_VERS (→ down), (→ 3S).
READ_SOFTWARE_VERS	for PDM, CAN operation with layer 2 or as master or slave: Shows the software version. Contains the following parameterised POUs: - DEVICE_KERNEL_VERSION1 (→ DEVICE_KERNEL_VERSION1), - DEVICE_RUNTIME_VERSION (→ DEVICE_RUNTIME_VERSION), - LEFT (→ 3S).

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

Structure of the visualisations in the templates

3979

Available for the following devices:
- BasicDisplay: CR0451
- PDM: CR10nn

The visualisations are structured in folders as follows:

Folder	Image no.	Description contents
START_PAGE	P00001	Setting / display of... - node ID - CAN baud rate - status - GuardErrorNode - PLC cycle time
__MAIN_MENUES	P00010	Menu screen: - Display setup
____MAIN_MENUE_1		
______DISPLAY_SETUP		
________1_DISPLAY_SETUP1	P65000	Menu screen: - Software version - brightness / contrast - display / set real-time clock
__________1_SOFTWARE_VERSION	P65010	Display of the software version.
__________2_BRIGHTNESS	P65020	Adjustment of brightness / contrast
__________3_SET_RTC	P65030	Display / set real-time clock

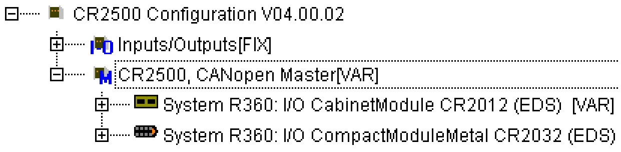
In the templates we have organised the image numbers in steps of 10. This way you can switch into different language versions of the visualisations by means of an image number offset.

Supplement project with further functions

3987

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

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



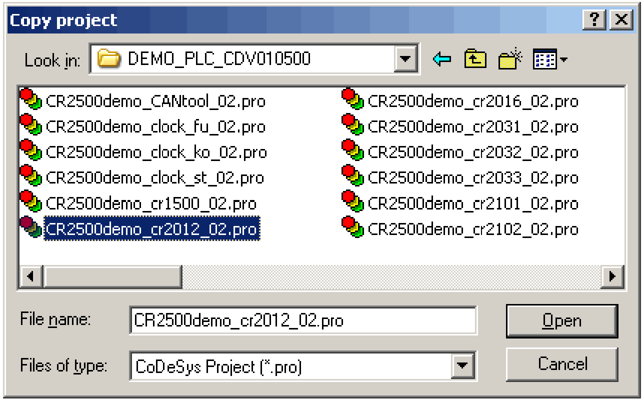
CR2500 Configuration V04.00.02
- Inputs/Outputs[FIX]
- CR2500, CANopen Master[VAR]
  - System R360: I/O CabinetModule CR2012 (EDS) [VAR]
  - System R360: I/O CompactModuleMetal CR2032 (EDS)

Example: PLC configuration

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

► Save CoDeSys project!

► In CoDeSys use [Project] > [Copy...] to open the project containing the requested function:
e.g. `CR2500Demo_CR2012_02.pro` from directory `DEMO_PLC_CDV...` under `C:\...\CoDeSys V...\Projects\`:



Look in: DEMO_PLC_CDV010500

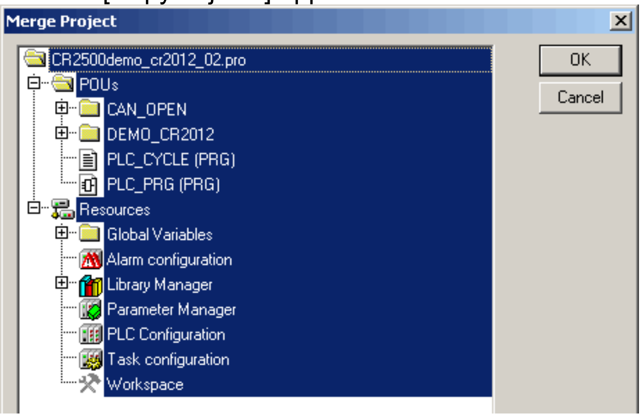
CR2500demo_CANtool_02.pro
CR2500demo_clock_fu_02.pro
CR2500demo_clock_ko_02.pro
CR2500demo_clock_st_02.pro
CR2500demo_cr1500_02.pro
CR2500demo_cr2012_02.pro
CR2500demo_cr2016_02.pro
CR2500demo_cr2031_02.pro
CR2500demo_cr2032_02.pro
CR2500demo_cr2033_02.pro
CR2500demo_cr2101_02.pro
CR2500demo_cr2102_02.pro

File name: CR2500demo_cr2012_02.pro

Files of type: CoDeSys Project (*.pro)

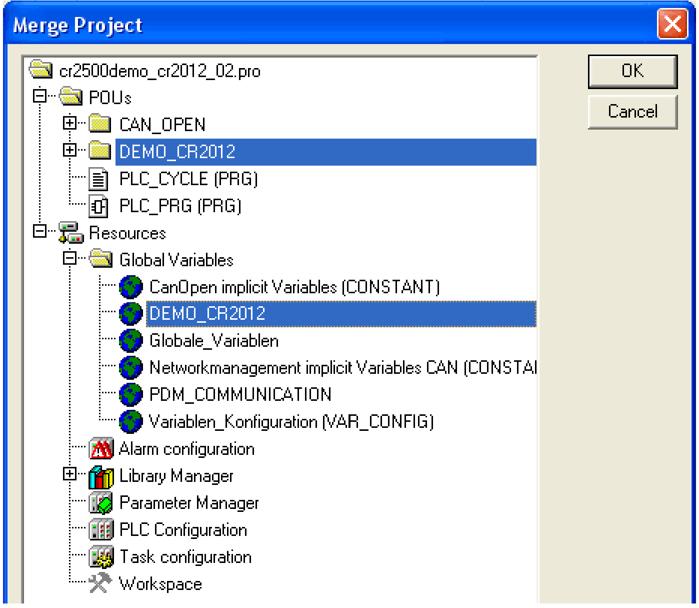
► Confirm the selection with [Open].

> Window [Copy objects] appears:



CR2500demo_cr2012_02.pro
- POUs
  - CAN_OPEN
  - DEMO_CR2012
  - PLC_CYCLE (PRG)
  - PLC_PRG (PRG)
- Resources
  - Global Variables
  - Alarm configuration
  - Library Manager
  - Parameter Manager
  - PLC Configuration
  - Task configuration
  - Workspace

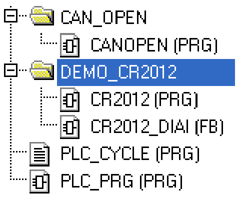
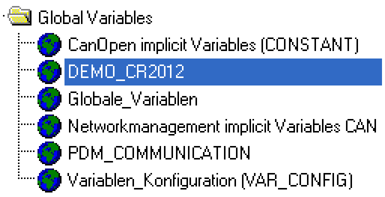
- ► Highlight the elements which contain only the requested function, in this case e.g.:



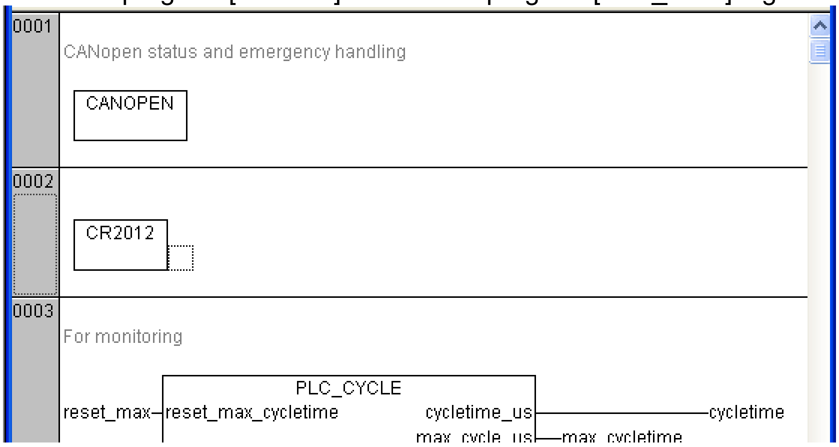
NOTE: In other cases libraries and/or visualisations might be required.

- ► Confirm the selection with [OK].

> In our example project the elements selected in the demo project have been added:

POUs:	Resources:
 CAN_OPEN CANOPEN (PRG) DEMO_CR2012 CR2012 (PRG) CR2012_DIAI (FB) PLC_CYCLE (PRG) PLC_PRG (PRG)	 Global Variables CanOpen implicit Variables (CONSTANT) DEMO_CR2012 Globale_Variablen Networkmanagement implicit Variables CAN PDM_COMMUNICATION Variablen_Konfiguration (VAR_CONFIG)

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



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

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

5.1.3 ifm demo programs

Contents

Demo program for controller 34
Demo programs for PDM and BasicDisplay 36

3982

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

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

Comments in the POUs and in the variable lists help you adapt the demo to your project.

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

The demo programs described here apply for:
- CoDeSys from version 2.3.9.6
- on the ***ecomatmobile*** DVD "Software, tools and documentation" from version 010500

Demo program for controller

3995

Demo program	Function
`CR2500Demo_CanTool_xx.pro`	separate for PDM360, PDM360compact, PDM360smart and Controller: Contains FBs to set and analyse the CAN interface.

`CR2500Demo_ClockFu_xx.pro`
`CR2500Demo_ClockKo_xx.pro`
`CR2500Demo_ClockSt_xx.pro`
	Clock generator for Controller as a function of a value on an analogue input:
	Fu = in function block diagram
	K0 = in ladder diagram
	St = in structured text

`CR2500Demo_CR1500_xx.pro`	Connection of a keypad module CR1500 as slave of a Controller (CANopen master).
`CR2500Demo_CR2012_xx.pro`	I/O cabinet module CR2012 as slave of a Controller (CANopen master), Connection of a joystick with direction switch and reference medium voltage.
`CR2500Demo_CR2016_xx.pro`	I/O cabinet module CR2016 as slave of a Controller (CANopen master), 4 x frequency input, 4 x digital input high side, 4 x digital input low side, 4 x analogue input ratiometric, 4 x PWM1000 output and 12 x digital output.
`CR2500Demo_CR2031_xx.pro`	I/O compact module CR2031 as slave of a Controller (CANopen master), Current measurement on the PWM outputs
`CR2500Demo_CR2032_xx.pro`	I/O compact module CR2032 as slave of a Controller (CANopen master), 4 x digital input, 4 x digital input analogue evaluation, 4 x digital output, 4 x PWM output.

Demo program	Function
`CR2500Demo_CR2033_xx.pro`	I/O compact module CR2033 as slave of a Controller (CANopen master), 4 x digital input, 4 x digital input analogue evaluation, 4 x digital output,
`CR2500Demo_CR2101_xx.pro`	Inclination sensor CR2101 as slave of a Controller (CANopen master).
`CR2500Demo_CR2102_xx.pro`	Inclination sensor CR2102 as slave of a Controller (CANopen master).
`CR2500Demo_CR2511_xx.pro`	I/O smart module CR2511 as slave of a Controller (CANopen master), 8 x PWM output current-controlled.
`CR2500Demo_CR2512_xx.pro`	I/O smart module CR2512 as slave of a Controller (CANopen master), 8 x PWM output. Display of the current current for each channel pair.
`CR2500Demo_CR2513_xx.pro`	I/O smart module CR2513 as slave of a Controller (CANopen master), 4 x digital input, 4 x digital output, 4 x analogue input 0...10 V.
`CR2500Demo_Interrupt_xx.pro`	Example with SET_INTERRUPT_XMS (→ page 293).
`CR2500Demo_Operating_hours_xx.pro`	Example of an operating hours counter with interface to a PDM.
`CR2500Demo_PWM_xx.pro`	Converts a potentiometer value on an input into a normed value on an output with the following POUs: - INPUT_VOLTAGE (→ page 193), - NORM (→ page 196), - PWM100 (→ page 223).
`CR2500Demo_RS232_xx.pro`	Example for the reception of data on the serial interface by means of the Windows hyper terminal.

`StartersetDemo.pro`
`StartersetDemo2.pro`
`StartersetDemo2_fertig.pro`
	Various e-learning exercises with the starter set EC2074.

_xx = indication of the demo version

Demo programs for PDM and BasicDisplay

3996

Demo program	Function

`CR1051Demo_CanTool_xx.pro`
`CR1053Demo_CanTool_xx.pro`
`CR1071Demo_CanTool_xx.pro`
	separate for PDM360, PDM360compact, PDM360smart and Controller:
	Contains FBs to set and analyse the CAN interface.

`CR1051Demo_Input_Character_xx.pro`	Allows to enter any character in a character string: - capital letters, - small letters, - special characters, - figures. Selection of the characters via encoder. Example also suited for e.g. entering a password. Figure P01000: Selection and takeover of characters
`CR1051Demo_Input_Lib_xx.pro`	Demo of INPUT_INT from the library `ifm_pdm_input_Vxxyyzz` (possible alternative to 3S standard). Select and set values via encoder. Figure P10000: 6 values INT Figure P10010: 2 values INT Figure P10020: 1 value REAL
`CR1051Demo_Linear_logging_on_flash_intern_xx.pro`	Writes a CVS data block with the contents of a CAN message in the internal flash memory (`/home/project/daten.csv`), when [F3] is pressed or a CAN message is received on ID 100. When the defined memory range is full the recording of the data is finished. POUs used: - WRITE_CSV_8BYTE, - SYNC. Figure P35010: Display of data information Figure P35020: Display of current data record Figure P35030: Display of list of 10 data records
`CR1051Demo_O2M_1Cam_xx.pro`	Connection of 1 camera O2M100 to the monitor with CAM_O2M. Switching between partial screen and full screen. Figure 39000: Selection menu Figure 39010: Camera image + text box Figure 39020: Camera image as full screen Figure 39030: Visualisation only
`CR1051Demo_O2M_2Cam_xx.pro`	Connection of 2 cameras O2M100 to the monitor with CAM_O2M. Switching between the cameras and between partial screen and full screen. Figure 39000: Selection menu Figure 39010: Camera image + text box Figure 39020: Camera image as full screen Figure 39030: Visualisation only
`CR1051Demo_Powerdown_Retain_bin_xx.pro`	Example with PDM_POWER_DOWN from the library `ifm_CR1051_Vxxyyzz.Lib`, to save retain variable in the file `Retain.bin`. Simulation of ShutDown with [F3].
`CR1051Demo_Powerdown_Retain_bin2_xx.pro`	Example with PDM_POWER_DOWN from the library `ifm_CR1051_Vxxyyzz.Lib`, to save retain variable in the file `Retain.bin`. Simulation of ShutDown with [F3].
`CR1051Demo_Powerdown_Retain_cust_xx.pro`	Example with PDM_POWER_DOWN and the PDM_READ_RETAIN from the library `ifm_CR1051_Vxxyyzz.Lib`, to save retain variable in the file `/home/project/myretain.bin`. Simulation of ShutDown with [F3].
`CR1051Demo_Read_Textline_xx.pro`	The example program reads 7 text lines at a time from the PDM file system using READ_TEXTLINE. Figure P01000: Display of read text

Demo program	Function
`CR1051Demo_Real_in_xx.pro`	Simple example for entering a REAL value in the PDM. Figure P01000: Enter and display REAL value
`CR1051Demo_Ringlogging_on_flash_intern_xx.pro`	Writes a CVS data block in the internal flash memory when [F3] is pressed or a CAN message is received on ID 100. The file names can be freely defined. When the defined memory range is full the recording of the data starts again. POUs used: - WRITE_CSV_8BYTE, - SYNC. Figure P35010: Display of data information Figure P35020: Display of current data record Figure P35030: Display of list of 8 data records
`CR1051Demo_Ringlogging_on_flash_pcmcia_xx.pro`	Writes a CVS data block on the PCMCIA card when [F3] is pressed or a CAN message is received on ID 100. The file names can be freely defined. When the defined memory range is full the recording of the data starts again. POUs used: - WRITE_CSV_8BYTE, - OPEN_PCMCIA, - SYNC. Figure P35010: Display of data information Figure P35020: Display of current data record Figure P35030: Display of list of 8 data records
`CR1051Demo_RW-Parameter_xx.pro`	In a list parameters can be selected and changed. Example with the following POUs: - READ_PARAMETER_WORD, - WRITE_PARAMETER_WORD. Figure P35010: List of 20 parameters

_xx = indication of the demo version

5.2 Function configuration of the inputs and outputs

Contents

Configure inputs 38
Configure outputs 43

1394

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

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

- It must be checked by means of the data sheet if the device used has the described input and output groups (→ data sheet).
- Constants are predefined (e.g. IN_DIGITAL_H) in the device libraries (e.g. `ifm_CR0020_Vx.LIB`) for the configuration of the inputs and outputs.
  For details → Possible operating modes inputs / outputs (→ page 304).

5.2.1 Configure inputs

Contents

Digital inputs 39
Fast inputs 40
Analogue inputs 41
Analogue input group A_IN00...29 (%IW2...31) 42
Input group IN00...IN15 (%IX0.0...%IX0.15) 42

3973

Digital inputs

1015

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

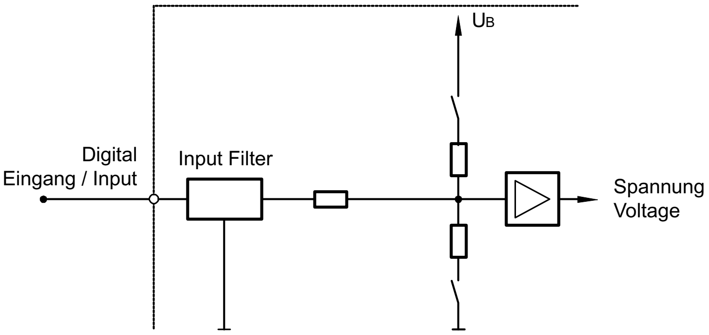
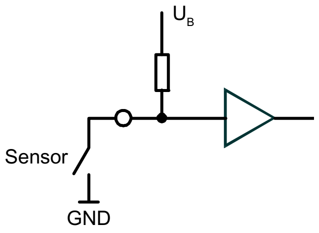
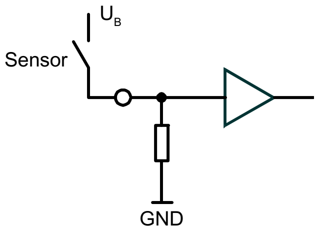


Figure: Block diagram high/low side input for negative and positive sensor signals



High side input for negative sensor signal



Low side input for positive sensor signal

Fast inputs

1018

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

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

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

On FRQx inputs:

- Frequency measurement with FREQUENCY (→ page 200)
- Fast counter with FAST_COUNT (→ page 211)

On CYLx inputs:

- Period measurement with PERIOD (→ page 202) or with PERIOD_RATIO (→ page 204)
- Phase position of 2 fast inputs compared via PHASE (→ page 206)

Info

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

Analogue inputs

1369

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

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

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

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

NOTE

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

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

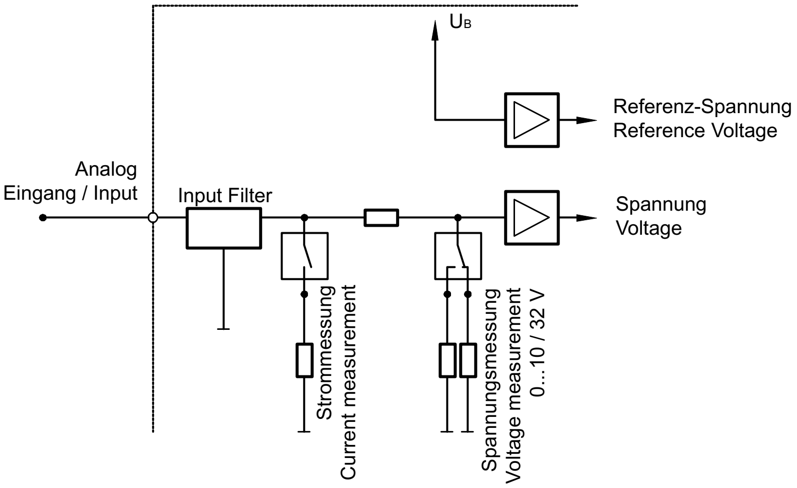


Figure: block diagram of the analogue inputs

Analogue input group A_IN00...29 (%IW2...31)

1395

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

Configuration can be carried out via the system variables A_INx_MODE (→ annex (→ page 299)) or, preferably, via INPUT_ANALOG (→ page 191) (input MODE).

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

When the analogue input FBs are used, the diagnostic function is automatically activated.

Input group IN00...IN15 (%IX0.0...%IX0.15)

1396

These inputs are digital inputs. In addition, a part of these inputs can be configured for negative input signals and frequency measurement (→ annex (→ page 299)).

The inputs IN00...IN07 can only be operated with positive signals. The configuration is possible only for diagnosis via the system variables IN00_MODE...IN07_MODE.

The inputs IN08...IN11 can either be operated as digital inputs or frequency inputs, with positive signals only. The configuration of these inputs is carried out via the system variables IN08_MODE...IN11_MODE.

The evaluation of the input frequency on these inputs can for example be implemented via the following FBs:
- FREQUENCY (→ page 200)
- PERIOD (→ page 202)
- PERIOD_RATIO (→ page 204)

These units affect the equivalent mode byte directly during their initialisation. In case of using these units no manual configuration via mode byte is neccessary.

The inputs IN12...IN15 can either be operated with positive or negative signals. The configuration of these inputs is carried out via the system variables IN12_MODE...IN15_MODE.

Info

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

5.2.2 Configure outputs

Contents

Digital and PWM outputs.....43
Output groups OUT0...OUT11 or OUT0...OUT17 (%QX0.0...%QX0.11/0.17).....44

3976

Digital and PWM outputs

1355

Three types of controller outputs can be distinguished:

- Digital outputs with and without diagnostic function
- Digital outputs with and without diagnostic function and additional PWM mode
- PWM outputs

⚠ WARNING

Property damage or bodily injury due to malfunctions possible!

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

In this mode the overload protection OUT_OVERLOAD_PROTECTION is not available.

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

NOTE

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

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

Output groups OUT0...OUT11 or OUT0...OUT17 (%QX0.0...%QX0.11/0.17)

1397

These outputs are digital outputs. In addition, a part of these outputs can be configured for PWM signals (only voltage output, no current control) (→ Address assignment inputs / outputs (→ page 300)).

The outputs OUT00...OUT07 can either be operated as digital outputs (max. 4 A each) or PWM outputs (→ chapter PWM functions (→ page 213)). The following has to be observed:

- PWM0...PWM3 can be operated at different frequencies.
- PWM4...PWM7 must be operated at the same frequency.

The outputs OUT08...OUT11 can only be operated as digital outputs (max. 4 A each).
No configuration is possible.

The outputs OUT12...OUT17 can only be operated as digital outputs (max. 10 A each).
No configuration is possible.

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

Current measurement on the high-current outputs

2130

Unfortunately, the read-back values of the high-current outputs are not proportional to the actual currents. As a guide value please take into account the following information:

Measuring range [A]	= RAW value	Tolerance [%]
4	197	+50 / -50
10	529	+30 / -30

5.3 Hints to wiring diagrams

1426

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

Examples:

12 GND_A

12	Terminal number
GND_A	Terminal designation

30 %IX0.7 BL

30	Terminal number
%IX0.7	IEC address for a binary input
BL	Hardware version of the input, here: **B**inary **L**ow side

47 %QX0.3 BH/PH

47	Terminal number
%QX0.3	IEC address for a binary output
BH/PH	Hardware version of the output, here: **B**inary **H**igh side or **P**WM**H**igh side

The different abbreviations have the following meaning:

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

Allocation of the input/output channels:

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

! NOTE

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

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

6 Limitations and programming notes

Contents

Limits of the device......47
Programming notes for CoDeSys projects......51

3055

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

6.1 Limits of the device

7358

NOTE

Note the limits of the device! → data sheet

6.1.1 CPU frequency

8005

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

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

Monitor family / article no.	CPU frequency [MHz]
BasicDisplay: CR0451	50
PDM360: CR1050, CR1051, CR1060	50
PDM360compact: CR1052, CR1053, CR1055, CR1056	50
PDM360NG: CR108n, CR9042	400
PDM360smart: CR1070, CR1071	20

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

6.1.2 Above-average stress

1480

Applies only to the following devices:
- CabinetController: CR030n
- PCB controller: CS0015

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

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

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

ATTENTION

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

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

6.1.3 Watchdog behaviour

1490

For (nearly) all programable **ecomat*mobile*** devices the program runtime is monitored by a watchdog of CoDeSys.

If the maximum watchdog time is exceeded:
- the device carries out a reset and starts again
- only SafetyController CR7nnn: the controller remains in the reset; the LED goes out.

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

Controller	Watchdog [ms]
BasicController: CR040n	100
BasicDisplay: CR0451	100
CabinetController: CR030n	100...200
ClassicController: CR0020, CR0032, CR0505	100
ExtendedController: CR0200, CR0232	100
PCB controller: CS0015	100...200
SafetyController: CR7nnn	100
SmartController: CR25nn	100...200
PDM360: CR1050, CR1051, CR1060	no watchdog
PDM360compact: CR1052, CR1053, CR1055, CR1056	no watchdog
PDM360NG: CR108n, CR9042	monitored by Linux *)
PDM360smart: CR1070, CR1071	100...200

*) The Linux kernel and critical processes are separately monitored (different times). If triggered:
- all processes are stopped (reset)
- all outputs are switched off
- the screen goes black
- the status LED flashes red at 5 Hz
- restart necessary via voltage off/on

6.1.4 Available memory

3962

Applies only to the following devices:
- CabinetController: CR0303

Physical memory	Physically existing FLASH memory (non-volatile, slow memory)	1 Mbytes
	Physically existing SRAM ¹) (volatile, fast memory)	256 Kbytes
	Physically existing EEPROM (non-volatile, slow memory)	---
	Physically existing FRAM ²) (non-volatile, fast memory)	2 Kbytes
Use of the FLASH memory	Memory reserved for the code of the IEC application	576 Kbytes
	Memory for data other than the IEC application that can be written by the user such as files, bitmaps, fonts	176 Kbytes
	Memory for data other than the IEC application that can be processed by the user by means of FBs such as FLASHREAD, FLASHWRITE	16 Kbytes
RAM	Memory for the data in the RAM reserved for the IEC application	80 Kbytes
Remanent memory	Memory for the data declared as VAR_RETAIN in the IEC application	256 bytes
	Memory for the flags agreed as RETAIN in the IEC application	---
	Remanent memory freely available to the user. Access is made via FRAMREAD, FRAMWRITE.	1536 bytes
	FRAM ²) freely available to the user. Access is made via the address operator.	---

¹) SRAM indicates here all kinds of volatile and fast memories.
²) FRAM indicates here all kinds of non-volatile and fast memories.

6.2 Programming notes for CoDeSys projects

Contents

FB, FUN, PRG in CoDeSys 51
Note the cycle time! 52
Creating application program 52
Save 54
Using ifm downloader 54
Certification and distribution of the safety-related software 55
Changing the safety-relevant software after certification 55

7426

Here you receive tips how to program the device.

► See the notes in the CoDeSys programming manual
→ **ecomatmobile** DVD "Software, tools and documentation".

6.2.1 FB, FUN, PRG in CoDeSys

8473

In CoDeSys we differentiate between the following types of units (POUs):

FB = function block

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

FUN = function

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

PRG = program

- A PRG may have several inputs and several outputs.
- A PRG may be called only once within a project.
- Allowed: in a PRG call of PRG, FB or FUN.

NOTE

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

POU-calls must not be recursive (POU must not call itself), also not indirectly.

Background:

By calling a function all variables...
- will become initialised and
- after return will lose their validity.

Function blocks have two calls:
- one initialising call and
- the call to do something.

Therefore, that means for a FB call inside a function, that there is every time an additional initialising call.

6.2.2 Note the cycle time!

8006

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

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

NOTICE

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

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

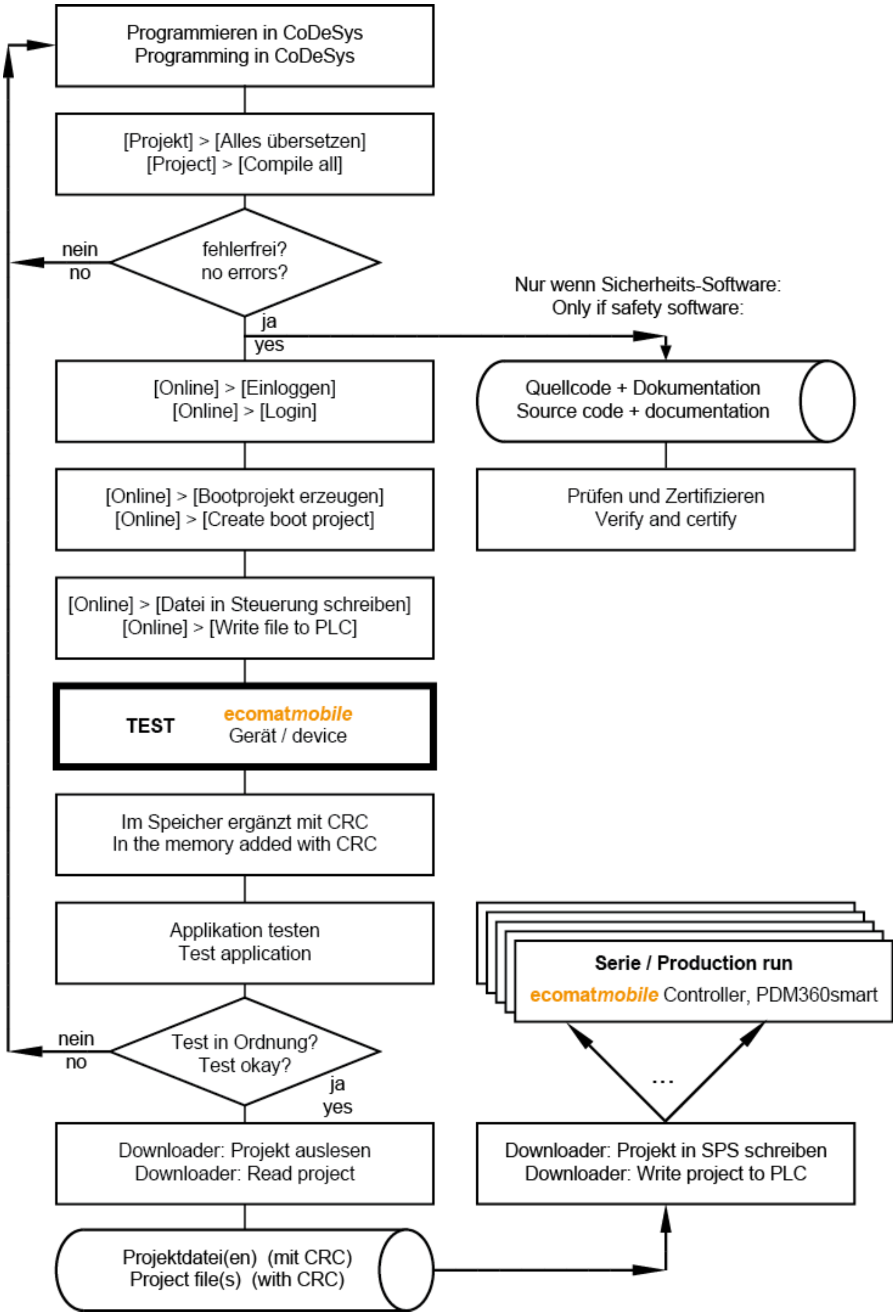
6.2.3 Creating application program

8007

The application program is generated by the CoDeSys programming system and loaded in the controller several times during the program development for testing:
In CoDeSys: [Online] > [Write file in the controller].

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

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


Programmieren in CoDeSys
Programming in CoDeSys
[Projekt] > [Alles übersetzen]
[Project] > [Compile all]
fehlerfrei?
no errors?
nein
no
ja
yes
Nur wenn Sicherheits-Software:
Only if safety software:
[Online] > [Einloggen]
[Online] > [Login]
Quellcode + Dokumentation
Source code + documentation
[Online] > [Bootprojekt erzeugen]
[Online] > [Create boot project]
Prüfen und Zertifizieren
Verify and certify
[Online] > [Datei in Steuerung schreiben]
[Online] > [Write file to PLC]
TEST
ecomatmobile
Gerät / device
Im Speicher ergänzt mit CRC
In the memory added with CRC
Applikation testen
Test application
Serie / Production run
ecomatmobile Controller, PDM360smart
Test in Ordnung?
Test okay?
nein
no
ja
yes
...
Downloader: Projekt auslesen
Downloader: Read project
Downloader: Projekt in SPS schreiben
Downloader: Write project to PLC
Projektdatei(en) (mit CRC)
Project file(s) (with CRC)

Graphics: Creation and distribution of the (certified) software

6.2.4 Save

7430

Applies only to the following devices:
- Controller CRnn32
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360NG: CR108n, CR9042

NOTE

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

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

► Menu [Online] > [Create boot project] (this must be carried out again after every change!).

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

6.2.5 Using ifm downloader

8008

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

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

NOTE

The **ifm** downloader cannot be used for the following devices:
- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360: CR1050, CR1051, CR1060,
- PDM360compact: CR1052, CR1053, CR1055, CR1056,
- PDM360NG: CR108n, CR9042

6.2.6 Certification and distribution of the safety-related software

8009

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

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

6.2.7 Changing the safety-relevant software after certification

8010

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

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

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

7 Error codes and diagnostic information

Contents

Overview 56
Response to the system error 57

1444

To ensure maximum operational reliability the operating system checks the **ecomat*mobile*** controller in the start phase (reset phase) and during the program execution by internal error checks.

7.1 Overview

2765

The following error flags are set in case of an error:

Error	Description
CANx_BUSOFF	CAN interface x: Interface is not on the bus
CANx_LASTERROR ¹)	CAN interface x: Error number of the last CAN transmission: 0= no error ≠0 → CAN specification → LEC
CANx_WARNING	CAN interface x: Warning threshold reached (> 96)
ERROR	Set ERROR bit ³) / switch off the relay *)
ERROR_MEMORY	Memory error
ERROR_POWER	Undervoltage/overvoltage error
ERROR_TEMPERATURE ²)	Excessive temperature error (> 85 °C)
ERROR_VBBR	Terminal voltage error VBB_R

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

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

²) Flag NOT available for CR250n, CR0301, CR0302.

³) If device has output ERROR: By setting the ERROR system flag the ERROR output (terminal 13) is set to FALSE. In the "error-free state" the output ERROR = TRUE (negative logic).

*) Relay NOT available for CR250n and CR030n.

The following diagnostic messages are only available for devices with periphery terminals:

Diagnostic message	Type	Description
ERROR_BREAK_Qx *)	BYTE	Wire break error on the output group x
ERROR_Ix	BYTE	Peripheral error on the input group x
ERROR_SHORT_Qx *)	BYTE	Short circuit error on the output group x

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

*) Flags only available for ClassicController, ExtendedController, SafetyController.

NOTE

In adverse cases the output transistor can already switch off a disturbed output before the operating system could detect the error. The corresponding error flag is then NOT set.

We recommend that the application programmer (additionally) evaluates the error by reading back the outputs.

Complete list of the device-specific error codes and diagnostic messages
→ chapter system flags (→ page 305).

7.2 Response to the system error

1445

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

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

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

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

7.2.1 Notes on devices with monitoring relay

1446

Available for the following devices:
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

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

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

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

NOTICE

Premature wear of the relay contacts possible.

► Only use this function for a general switch-off of the outputs in case of an "emergency".

► In normal operation switch off the relays only without load!
To do so, first switch off the outputs via the application program!

7.2.2 Example process for response to a system error

1447

The system determines an excessive temperature in the controller.

The operating system sets the error bit ERROR_TEMPERATURE.

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

> The application program switches off outputs.

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

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

► Rectify the cause of the error.

> The operating system resets the error bit ERROR_TEMPERATURE.

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

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

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

8 Using CAN

Contents

General about CAN 59
Physical connection of CAN 65
Exchange of CAN data 69
Description of the CAN standard program units 73
CAN units acc. to SAE J1939 99
ifm CANopen libraries 117
CAN errors and error handling 182

1163

8.1 General about CAN

Contents

Topology 60
CAN interfaces 61
Available CAN interfaces and CAN protocols 61
System configuration 63

1164

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

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

8.1.1 Topology

1244

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

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

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

NOTE

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

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

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

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

8.1.2 CAN interfaces

269

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

- CAN at level 2 (layer 2)
- CANopen (→ page 117) protocol to CiA 301/401 for master/slave operation (via CoDeSys)
- CAN Network variables (→ page 150) (via CoDeSys)
- Protocol SAE J1939 (→ page 99) (for engine management)
- Bus load detection
- Error frame counter
- Download interface
- 100 % bus load without package loss

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

Informative: more interesting CAN protocols:

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

8.1.3 Available CAN interfaces and CAN protocols

6467

In the **ifm** devices the following CAN interfaces and CAN protokols are available:

Interface **Device**	**CAN 1**	**CAN 2**	**CAN 3**	**CAN 4**
default download identifier	ID 127	ID 126	ID 125	ID 124
BasicController: CR040n	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	---	---
BasicDisplay: CR0451	CAN layer 2 CANopen SAE J1939	---	---	---
CabinetController: CR0301, CR0302	CAN layer 2 CANopen SAE J1939	---	---	---
CabinetController: CR0303	CAN layer 2 CANopen SAE J1939	CAN layer 2 SAE J1939	---	---
ClassicController: CR0020, CR0505	CAN layer 2 CANopen SAE J1939	CAN layer 2 SAE J1939	---	---
ClassicController: CR0032	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939

Interface **Device**	**CAN 1**	**CAN 2**	**CAN 3**	**CAN 4**
default download identifier	ID 127	ID 126	ID 125	ID 124
ExtendedController: CR0200	**CPU 1 CAN 1** CAN layer 2 CANopen SAE J1939	**CPU 1 CAN 2** CAN layer 2 SAE J1939	**CPU 2 CAN 1** ID 127 CAN layer 2 CANopen SAE J1939	**CPU 2 CAN 2** ID 126 CAN layer 2 SAE J1939
ExtendedController: CR0232	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939
PCB controller: CS0015	CAN layer 2 CANopen SAE J1939	---	---	---
SafetyController: CR7021, CR7506	CAN layer 2 CANopen CANopen Safety SAE J1939	CAN layer 2 CANopen Safety SAE J1939	---	---
ExtendedSafetyController: CR7201	**CPU 1 CAN 1** CAN layer 2 CANopen CANopen Safety SAE J1939	**CPU 1 CAN 2** CAN layer 2 CANopen Safety SAE J1939	**CPU 2 CAN 1** ID 127 CAN layer 2 CANopen SAE J1939	**CPU 2 CAN 2** ID 126 CAN layer 2 SAE J1939
SmartController: CR2500	CAN layer 2 CANopen SAE J1939	CAN layer 2 SAE J1939	---	---
PDM360: CR1050, CR1051, CR1060	CAN layer 2 CANopen	CAN layer 2 CANopen SAE J1939	---	---
PDM360compact: CR1052, CR1053, CR1055, CR1056	CAN layer 2 CANopen	---	---	---
PDM360smart: CR1070, CR1071	CAN layer 2 CANopen SAE J1939	---	---	---
PDM360NG: CR108n, CR9042	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939	CAN layer 2 CANopen SAE J1939

8.1.4 System configuration

2271
2270

The controllers are delivered with the following download identifier (= ID):
- ID 127 for CAN interface 1
- ID 126 for CAN interface 2 (if available)
- ID 125 for CAN interface 3 (if available)
- ID 124 for CAN interface 4 (if available)

The download system uses this identifier for the first communication with a non configured module via CAN.
The download IDs can be set as follows:
- via the PLC browser of the programming system,
- via the the downloader or the maintenance tool or
- via the application program.

Via the mode "Autoconfig" of the boot loader only CAN interface 1 can be set.

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

NOTE

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

Comparison of download-ID vs. COB-ID:

Controller program download		CANopen	
Download-ID	**COB-ID SDO**	**Node ID**	**COB-ID SDO**
1…127	TX: 580_{16} + download ID	1…127	TX: 580_{16} + node ID
	RX: 600_{16} + download ID		RX: 600_{16} + node ID

TX = slave sends to master
RX = slave receives from master

NOTE

The CAN download ID of the device must match the CAN download ID set in CoDeSys!

In the CAN network the CAN download IDs must be unique!

For the CabinetController CR0303 the download ID can either be set:

- using the integrated rotary switches S2+S3 OR:
- using CAN1_DOWNLOADID (→ page 77) in the application program if **both** integrated rotary switches (S2+S3) are set to "F".

! NOTE

The download ID activated in the application program permanently changes the factory setting to the new value!

8.2 Physical connection of CAN

Contents

Network structure 65
CAN bus level....66
CAN bus level according to ISO 11992-1 66
Bus cable length....67
Wire cross-sections....68

1177

The mechanisms of the data transmission and error handling described in the chapters Exchange of CAN data (→ page 69) and CAN errors (→ page 182) are directly implemented in the CAN controller. ISO 11898 describes the physical connection of the individual CAN participants in layer 1.

8.2.1 Network structure

1178

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

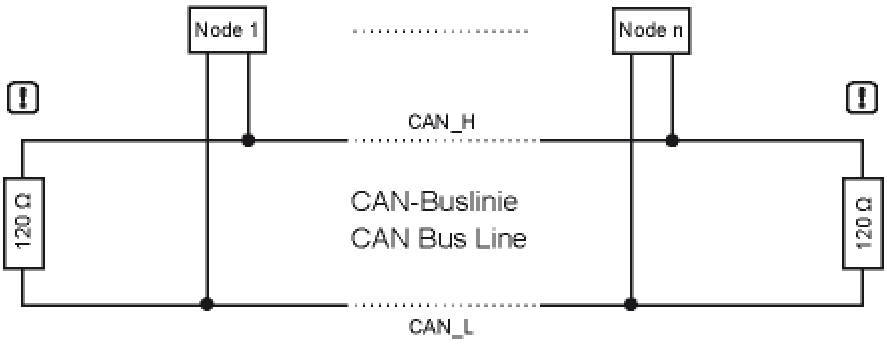


Figure: CAN network line structure

NOTE

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

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

Spurs

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

8.2.2 CAN bus level

1179

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

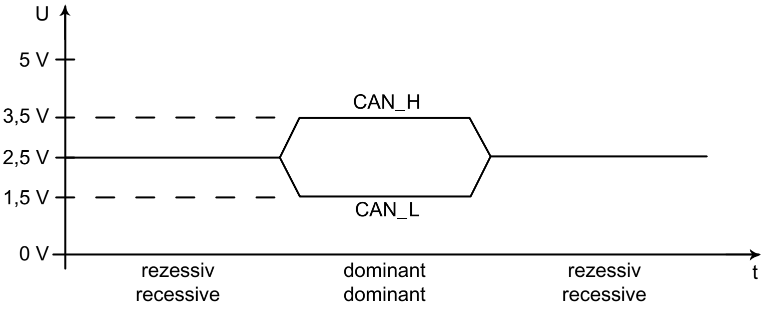


Figure: CAN bus level

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

8.2.3 CAN bus level according to ISO 11992-1

1182

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

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

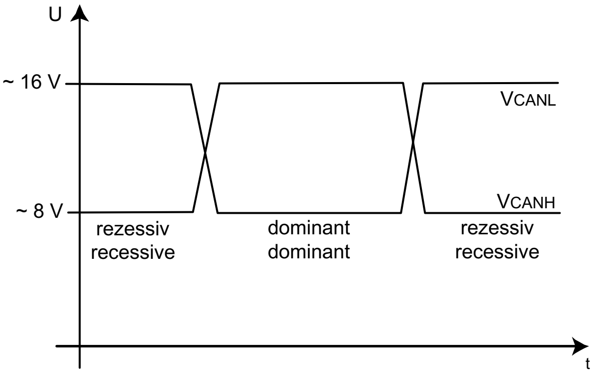


Figure: voltage level to ISO 11992-1 (here: 12 V system)

8.2.4 Bus cable length

1180

The length of the bus cable depends on:

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

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

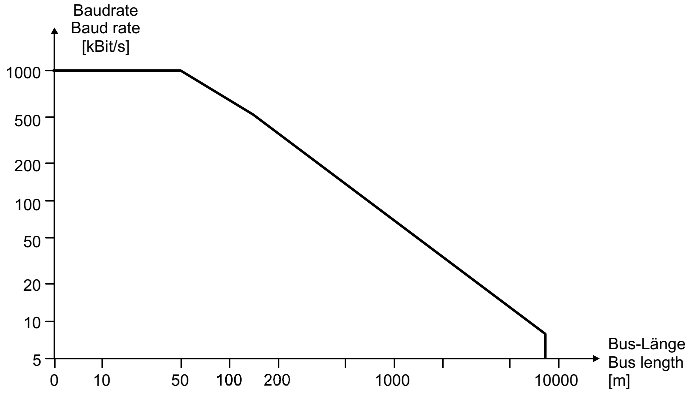


Figure: bus cable length

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

Table: Dependencies bus length / baud rate / bit time

8.2.5 Wire cross-sections

1181

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

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

Depending on the EMC requirements the bus cables can be laid out as follows:
- in parallel,
- as twisted pair
- and/or shielded.

8.3 Exchange of CAN data

Contents

Hints 70
Data reception 72
Data transmission..........72

1168

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

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

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

- CANx_RECEIVE (→ page 90) (receive CAN data) and
- CANx_TRANSMIT (→ page 88) (transmit CAN data).

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

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

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

The CAN message consists of a CAN identifier (CAN-ID (→ page 70)) and maximum 8 data bytes. The ID does not represent the transmit or receive module but identifies the message. To transmit data it is necessary that a transmit object is declared in the transmit module and a receive object in at least one other module. Both declarations must be assigned to the same identifier.

8.3.1 Hints

8394

CAN-ID

1166

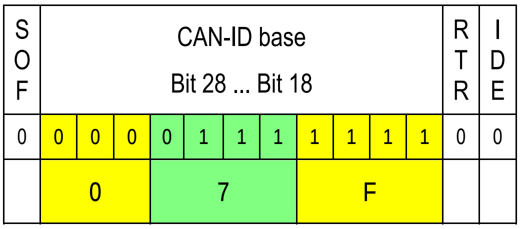
Depending of the CAN-ID the following CAN identifiers are free available for the data transfer:

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

NOTE

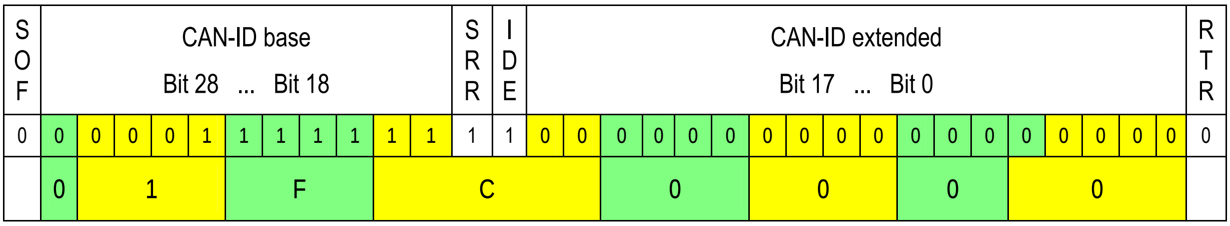
In some devices the 29 bits CAN-ID is not available for all CAN interfaces, → data sheet.

Example 11 bits CAN-ID (base):



SOF	CAN-ID base Bit 28 ... Bit 18											RTR	IDE
0	0	0	0	0	1	1	1	1	1	1	1	0	0
	0			7				F					

Example 29 bits CAN-ID (extended):



SOF	CAN-ID base Bit 28 ... Bit 18											SRR	IDE	CAN-ID extended Bit 17 ... Bit 0																		RTR
0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1				F				C						0				0				0			0					

Legend:
SOF = **S**tart **o**f **f**rame
Edge of recessive to dominant

RTR = **R**emote **t**ransmission **r**equest
dominant: This message sends data
recessive: This message requests data

IDE = **Id**entifier **e**xtension flag
dominant: After this control bits follows
recessive: After this the second part of the 29 bits identifier follows

SRR = **S**ubstitute **r**emote **r**equest
recessive: Extended CAN-ID: Replaces the RTR bit at this position

Summary CAN / CANopen

3956

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

8.3.2 Data reception

1169

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

Each identifier has such a buffer (queue). Depending on the application software this buffer is emptied according to the FiFo principle (**F**irst **I**n, **F**irst **O**ut) via CANx_RECEIVE (→ page 90).

8.3.3 Data transmission

1170

By calling CANx_TRANSMIT (→ page 88) the application program transfers exactly one CAN message to the CAN controller. As feedback you are informed whether the message was successfully transferred to the CAN controller. Which then automatically carries out the actual transfer of the data on the CAN bus.

The transmit order is rejected if the controller is not ready because it is in the process of transferring a data object. The transmit order must then be repeated by the application program. This information is indicated by a bit.

If several CAN messages are ready for transmission, the message with the lowest ID is transmitted first. Therefore, the programmer must assign the CAN ID (→ page 70) very carefully.

8.4 Description of the CAN standard program units

Contents

CAN1_BAUDRATE 75
CAN1_DOWNLOADID 77
CAN1_EXT 79
CAN1_EXT_TRANSMIT 81
CAN1_EXT_RECEIVE 83
CAN1_EXT_ERRORHANDLER 85
CAN2 86
CANx_TRANSMIT 88
CANx_RECEIVE 90
CANx_RECEIVE_RANGE 92
CANx_EXT_RECEIVE_ALL 95
CANx_ERRORHANDLER 97

1186

The CAN FBs are described for use in the application program.

NOTE

To use the full capacity of CAN it is absolutely necessary for the programmer to define an exact **bus concept** before starting to work:

- How many data objects are needed with what identifiers?
- How is the **ecomat*mobile*** device to react to possible CAN errors?
- How often must data be transmitted? CANx_TRANSMIT (→ page 88) and CANx_RECEIVE (→ page 90) must be called accordingly.

► Check whether the transmit orders were successfully assigned to CANx_TRANSMIT (output RESULT) or ensure that the received data is read from the data buffer of the queue using CANx_RECEIVE and processed in the rest of the program immediately.

To be able to set up a communication connection, the same transmission rate (baud rate) must first be set for all participants of the CAN network. For the controller this is done using CAN1_BAUDRATE (→ page 75) (for the 1st CAN interface) or via CAN2 (→ page 86) (for the 2nd CAN interface).

Irrespective of whether the devices support one or several CAN interfaces the FBs related to the interface are specified by a number in the CAN FB (e.g. CAN1_TRANSMIT or CAN2_RECEIVE). To simplify matters the designation (e.g. CANx_TRANSMIT) is used for all variants in the documentation.

ⓘ NOTE

When installing the **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_CDVxxyyzz`

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

> CoDeSys creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
→ chapter Set up programming system via templates (→ page 24)

In this example data objects are exchanged with other CAN participants via the identifiers 1 and 2. To do so, a receive identifier must exist for the transmit identifier (or vice versa) in the other participant.

8.4.1 CAN1_BAUDRATE

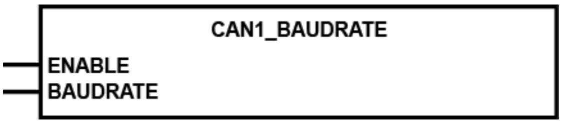
651

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

654

CAN1_BAUDRATE sets the transmission rate for the bus participant.

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

> After executing the FB the new value is stored in the device and will even be available after a power failure.

ATTENTION

Please note for CR250n, CR0301, CR0302 and CS0015:

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

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

► Afterwards block the unit again with ENABLE = FALSE!

NOTE

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

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

Parameters of the inputs

655

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
BAUDRATE	WORD	Baud rate [kbits/s] permissible values: 50, 100, 125, 250, 500, 1000 preset value = 125 kbits/s

8.4.2 CAN1_DOWNLOADID

645

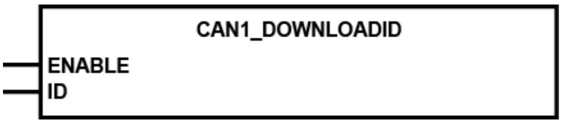
= CAN1 Download-ID

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

648

CAN1_DOWNLOADID sets the download identifier for the first CAN interface.

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

ATTENTION

Please note for CR250n, CR0301, CR0302 and CS0015:

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

- ► Only carry out the unit **once** during initialisation in the first program cycle!
- ► Afterwards block the unit again with ENABLE = FALSE!

NOTE

Make sure that a different download ID is entered for each device in the same network!

If the device is operated in the CANopen network, the download ID must not coincide with any module ID (node number) of the other participants, either!

ExtendedController: In the slave module the download ID becomes effective after voltage OFF/ON.

Parameters of the inputs

649

Parameter	Data type	Description
ENABLE	BOOL	TRUE (or only 1 cycle): ID is set FALSE: unit is not executed
ID	BYTE	download identifier permissible values: 1…127

8.4.3 CAN1_EXT

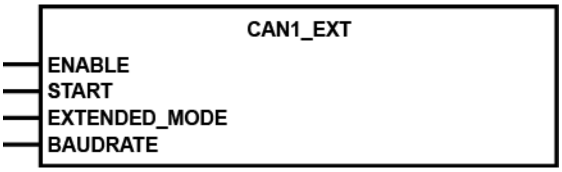
4192

Unit type = function block (FB)

Contained in the library: `ifm_CAN1_EXT_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4333

CAN1_EXT initialises the first CAN interface for the extended identifier (29 bits).

The FB has to be retrieved if the first CAN interface e.g. with the function libraries for SAE J1939 (→ page 99) is to be used.

A change of the baud rate will become effective after voltage OFF/ON. The baud rates of CAN 1 and CAN 2 can be set differently.

The input START is only set for one cycle during reboot or restart of the interface.

NOTE

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

Parameters of the inputs

4334

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
START	BOOL	TRUE (in the 1st cycle): interface is initialised FALSE: initialisation cycle completed
EXTENDED_MODE	BOOL	TRUE: identifier of the 1st CAN interface operates with 29 bits FALSE: identifier of the 1st CAN interface operates with 11 bits
BAUDRATE	WORD	baud rate [kbits/s] permissible values = 50, 100, 125, 250, 500, 1000 preset value = 125 kbits/s

8.4.4 CAN1_EXT_TRANSMIT

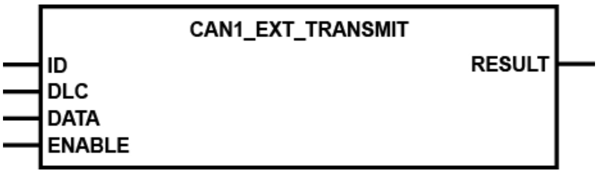
4307

Unit type = function block (FB)

Contained in the library: `ifm_CAN1_EXT_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4337

CAN1_EXT_TRANSMIT transfers a CAN data object (message) to the CAN controller for transmission.

The FB is called for each data object in the program cycle; this is done several times in case of long program cycles. The programmer must ensure by evaluating the output RESULT that his transmit order was accepted. To put it simply, at 125 kbits/s one transmit order can be executed per 1 ms.

The execution of the FB can be temporarily blocked via the input ENABLE = FALSE. This can, for example, prevent a bus overload.

Several data objects can be transmitted virtually at the same time if a flag is assigned to each data object and controls the execution of the FB via the ENABLE input.

NOTE

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page 79).

Parameters of the inputs

4380

Parameter	Data type	Description
ID	DWORD	number of the data object identifier permissible values: 11-bit ID = 0...2 047, 29-bit ID = 0...536 870 911
DLC	BYTE	number of bytes to be transmitted from the array DATA permissible values = 0...8
DATA	ARRAY[0...7] OF BYTE	the array contains max. 8 data bytes
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active

Parameters of the outputs

614

Parameter	Data type	Description
RESULT	BOOL	TRUE (only 1 cycle): the unit has accepted the transmit order

8.4.5 CAN1_EXT_RECEIVE

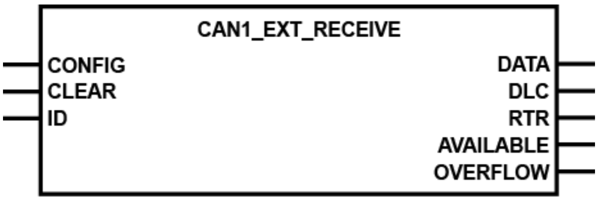
4302

Unit type = function block (FB)

Contained in the library: `ifm_CAN1_EXT_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4336

CAN1_EXT_RECEIVE configures a data receive object and reads the receive buffer of the data object.

The FB must be called once for each data object during initialisation to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CAN1_EXT_RECEIVE is called for reading the corresponding receive buffer, this is done several times in case of long program cycles The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed.

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

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE at least 1 data object has been lost.

NOTE

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page 79).

Parameters of the inputs

2172

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only for 1 cycle): configure data object FALSE: this function is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed
ID	WORD	number of the data object identifier permissible values normal frame = 0...2 047 (2^{11}) permissible values extended frame = 0...536 870 912 (2^{29})

Parameters of the outputs

632

Parameter	Data type	Description
DATA	ARRAY[0...7] OF BYTES	the array contains a maximum of 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
RTR	BOOL	not supported
AVAILABLE	BYTE	number of received messages
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

CAN1_EXT_ERRORHANDLER

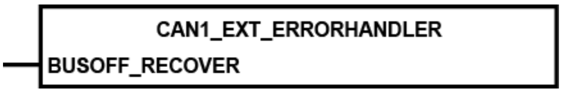
4195

Unit type = function block (FB)

Contained in the library: `ifm_CAN1_EXT_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

4335

CAN1_EXT_ERRORHANDLER monitors the first CAN interface and evaluates the CAN errors. If a certain number of transmission errors occurs, the CAN participant becomes error passive. If the error frequency decreases, the participant becomes error active again (= normal condition).

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

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

ⓘ NOTE

If the automatic bus recover function is to be used (default setting) CAN1_EXT_ERRORHANDLER must **not** be integrated and instanced in the program!

Parameters of the inputs

2177

Parameter	Data type	Description
BUSOFF_RECOVER	BOOL	TRUE (only for 1 cycle): > reboot of the CAN interface x > remedy "bus off" status FALSE: this function is not executed

8.4.6 CAN2

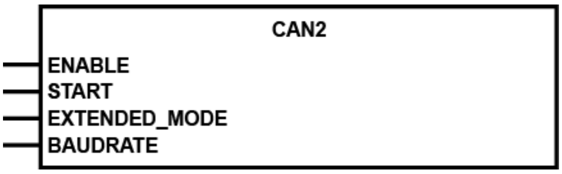
639

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Symbol in CoDeSys:



Description

642

CAN2 initialises the 2nd CAN interface.

The FB must be called if the 2nd CAN interface is to be used.

A change of the baud rate will become effective after voltage OFF/ON. The baud rates of CAN 1 and CAN 2 can be set differently.

The input START is only set for one cycle during reboot or restart of the interface.

For the 2nd CAN interface the libraries for SAE J1939 (→ page 99) and ISO 11992, among others, are available. The FBs to ISO 11992 are only available in the CR2501 on the 2nd CAN interface.

NOTE

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

Parameters of the inputs

643

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
START	BOOL	TRUE (in the 1st cycle): interface is initialised FALSE: initialisation cycle completed
EXTENDED_MODE	BOOL	TRUE: identifier of the 2nd CAN interface operates with 29 bits FALSE: identifier of the 2nd CAN interface operates with 11 bits
BAUDRATE	WORD	Baud rate [kbits/s] permissible values: 50, 100,125, 250, 500, 800, 1000 preset value = 125 kbits/s

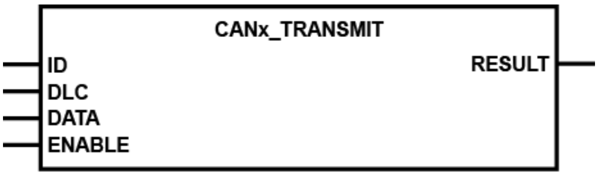
8.4.7 CANx_TRANSMIT

609

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_TRANSMIT

9362

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_TRANSMIT)
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

CAN2_TRANSMIT

9363

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_TRANSMIT)
- SmartController: CR2500

Description

612

CANx_TRANSMIT transmits a CAN data object (message) to the CAN controller for transmission.

The FB is called for each data object in the program cycle, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the FB output RESULT that his transmit order was accepted. Simplified it can be said that at 125 kbits/s one transmit order can be executed per ms.

The execution of the FB can be temporarily blocked (ENABLE = FALSE) via the input ENABLE. So, for example a bus overload can be prevented.

Several data objects can be transmitted virtually at the same time if a flag is assigned to each data object and controls the execution of the FB via the ENABLE input.

! NOTE

If CAN2_TRANSMIT is to be used, the second CAN interface must be initialised first using CAN2 (→ page 86).

Parameters of the inputs

613

Parameter	Data type	Description
ID	WORD	number of the data object identifier permissible values = 0...2 047
DLC	BYTE	number of bytes to be transmitted from the array DATA permissible values = 0...8
DATA	ARRAY[0...7] OF BYTES	the array contains a maximum of 8 data bytes
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active

Parameters of the outputs

614

Parameter	Data type	Description
RESULT	BOOL	TRUE (only 1 cycle): the unit has accepted the transmit order

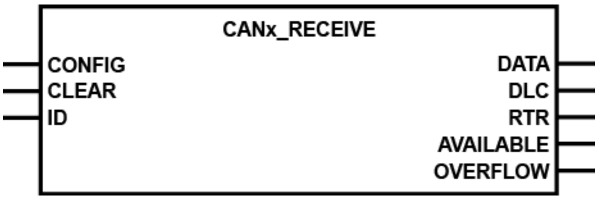
8.4.8 CANx_RECEIVE

627

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_RECEIVE

9354

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_RECEIVE)
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

CAN2_RECEIVE

9355

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_RECEIVE)
- SmartController: CR2500

Description

630

CANx_RECEIVE configures a data receive object and reads the receive buffer of the data object.

The FB must be called once for each data object during initialisation, in order to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CANx_RECEIVE is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed.

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

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE at least 1 data object has been lost.

NOTE

If CAN2_RECEIVE is to be used, the second CAN interface must be initialised first using CAN2 (→ page 86).

Parameters of the inputs

631

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only 1 cycle): Configure data object FALSE: unit is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed
ID	WORD	number of the data object identifier permissible values = 0...2 047

Parameters of the outputs

632

Parameter	Data type	Description
DATA	ARRAY[0...7] OF BYTES	the array contains a maximum of 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
RTR	BOOL	not supported
AVAILABLE	BYTE	number of received messages
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

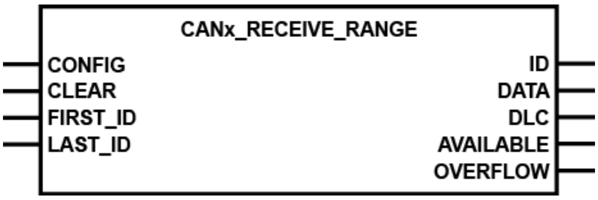
8.4.9 CANx_RECEIVE_RANGE

4179

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_RECEIVE_RANGE

9359

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB` (xx > 05)

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_RECEIVE)
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

CAN2_RECEIVE_RANGE

9360

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB` (xx > 05)

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_RECEIVE)
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Description

2295

CANx_RECEIVE_RANGE configures a sequence of data receive objects and reads the receive buffer of the data objects.

For the first CAN interface max. 2048 IDs per bit are possible.
For the second CAN interface max. 256 IDs per 11 OR 29 bits are possible.
The second CAN interface requires a long initialisation time. To ensure that the watchdog does not react, the process should be distributed to several cycles in the case of bigger ranges. → Example (→ page 94).

The FB must be called once for each sequence of data objects during initialisation to inform the CAN controller about the identifiers of the data objects.

The FB must NOT be mixed with CANx_RECEIVE (→ page 90) or CANx_RECEIVE_RANGE for the same IDs at the same CAN interfaces.

In the further program cycle CANx_RECEIVE_RANGE is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer has to ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from buffer SOFORT and are further processed as the data are only available for one cycle.

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

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE, at least 1 data object has been lost.

Receive buffer: max. 16 software buffers per identifier.

Parameters of the inputs

2290

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only for 1 cycle): configure data object FALSE: this function is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed
FIRST_ID	CAN1: WORD CAN2: DWORD	number of the first data object identifier of the sequence permissible values normal frame = 0...2 047 (2^{11}) permissible values extended frame = 0...536 870 912 (2^{29})
LAST_ID	CAN1: WORD CAN2: DWORD	number of the last data object identifier of the sequence permissible values normal frame = 0...2 047 (2^{11}) permissible values extended frame = 0...536 870 912 (2^{29}) LAST_ID has to be bigger than FIRST_ID.

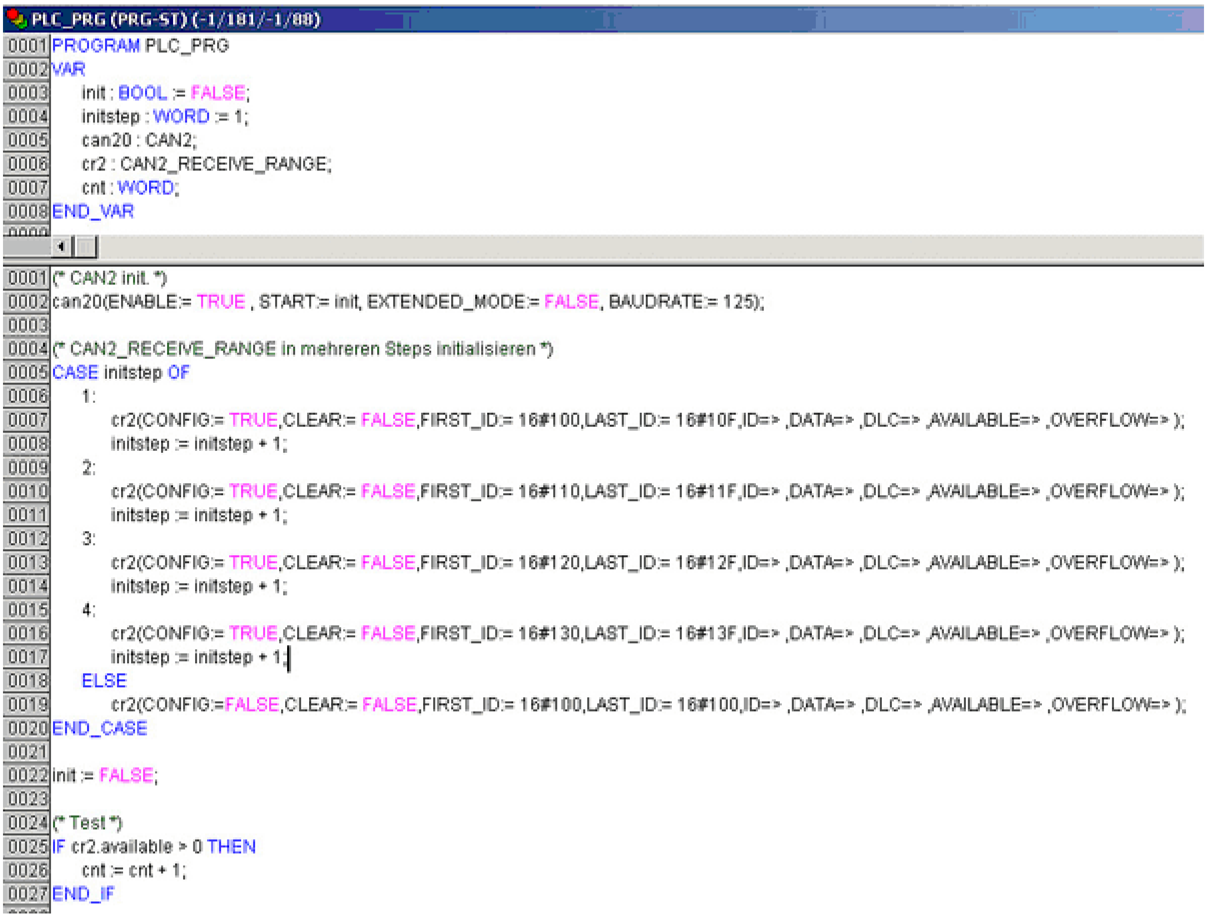
Parameters of the outputs

4381

Parameter	Data type	Description
ID	CAN1: WORD CAN2: DWORD	ID of the transmitted data object
DATA	ARRAY[0...7] OF BYTE	the array contains max. 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
AVAILABLE	BYTE	number of messages in the buffer
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

Example: Initialisation of CANx_RECEIVE_RANGE in 4 cycles

2294



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

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

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

init := FALSE;

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

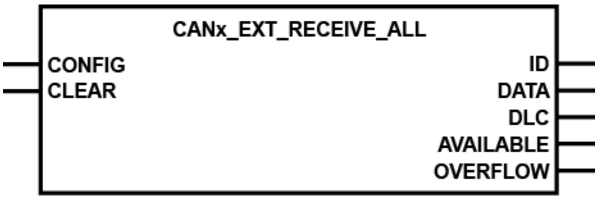
8.4.10 CANx_EXT_RECEIVE_ALL

4183

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_EXT_RECEIVE_ALL

9351

Contained in the library: `ifm_CAN1_EXT_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_RECEIVE)
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

CAN2_EXT_RECEIVE_ALL

9352

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
  POU not for safety signals!
  (For safety signals → CAN_SAFETY_RECEIVE)
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Description

4326

CANx_EXT_RECEIVE_ALL configures all data receive objects and reads the receive buffer of the data objects.

The FB must be called once during initialisation to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CANx_EXT_RECEIVE_ALL is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed.

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

By evaluating the output OVERFLOW, an overflow of the data buffer can be detected. If OVERFLOW = TRUE at least 1 data object has been lost.

Receive buffer: max. 16 software buffers per identifier.

Parameters of the inputs

4329

Parameter	Data type	Description
CONFIG	BOOL	TRUE (only for 1 cycle): configure data object FALSE: unit is not executed
CLEAR	BOOL	TRUE: deletes the data buffer (queue) FALSE: this function is not executed

Parameters of the outputs

2292

Parameter	Data type	Description
ID	DWORD	ID of the transmitted data object
DATA	ARRAY[0...7] OF BYTE	the array contains max. 8 data bytes
DLC	BYTE	number of bytes transmitted in the array DATA possible values = 0...8
AVAILABLE	BYTE	number of messages in the buffer
OVERFLOW	BOOL	TRUE: overflow of the data buffer → loss of data! FALSE: buffer not yet full

8.4.11 CANx_ERRORHANDLER

633

Unit type = function block (FB)

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

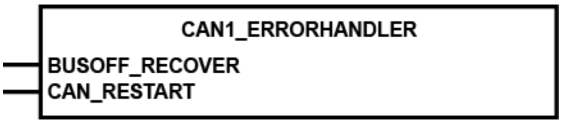
CAN1_ERRORHANDLER

9344

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



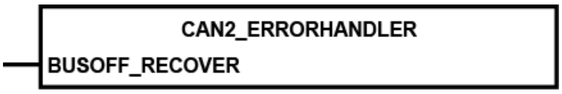
CAN2_ERRORHANDLER

9345

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Symbol in CoDeSys:



Description

636

Error routine for monitoring the CAN interfaces

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

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

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

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

The procedures for the restart of the interfaces are different:

- For CAN interface 1 or devices with only one CAN interface:
  set the input CAN_RESTART = TRUE (only 1 cycle)
- For CAN interface 2:
  set the input START = TRUE (only 1 cycle) in CAN2 (→ page 86)

NOTE

In principle, CAN2 must be executed to initialise the second CAN interface, before FBs can be used for it.

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

Parameters of the inputs

637

Parameter	Data type	Description
BUSOFF_RECOVER	BOOL	TRUE (only 1 cycle): remedy 'bus off' status FALSE: this function is not executed
CAN_RESTART	BOOL	TRUE (only 1 cycle): completely reinitialise CAN interface 1 FALSE: this function is not executed

8.5 CAN units acc. to SAE J1939

Contents

CAN for the drive engineering 100
Units for SAE J1939 104

7482

The network protocol SAE J1939 describes the communication on a CAN bus in utility vehicles for submitting diagnosis data (e.g. motor speed, temperature) and control information.

8.5.1 CAN for the drive engineering

Contents

Identifier acc. to SAE J1939 101
Example: detailed message documentation 102
Example: short message documentation 103

7678

With the standard SAE J1939 the CiA bietet offers to the user a CAN bus protocol for the drive engineering. For this protocol the CAN controller of the 2nd interface is switched to the "extended mode". This means that the CAN messages are transferred with a 29-bit identifier. Due to the longer identifier numerous messages can be directly assigned to the identifier.

For writing the protocol this advantage was used and certain messages were combined in ID groups. The ID assignment is specified in the standards SAE J1939 and ISO 11992. The protocol of ISO 11992 is based on the protocol of SAE J1939.

Standard	Application area
SAE J1939	Drive management
ISO 11992	"Truck & Trailer Interface"

The 29-bit identifier consists of two parts:
- an 11-bit ID and
- an 18-bit ID.

As for the software protocol the two standards do not differ because ISO 11992 is based on SAE J1939. Concerning the hardware interface, however, there is one difference: higher voltage level for ISO 11992.

NOTE

To use the functions to SAE J1939 the protocol description of the aggregate manufacturer (e.g. for motors, gears) is definitely needed. For the messages implemented in the aggregate control device this description must be used because not every manufacturer implements all messages or implementation is not useful for all aggregates.

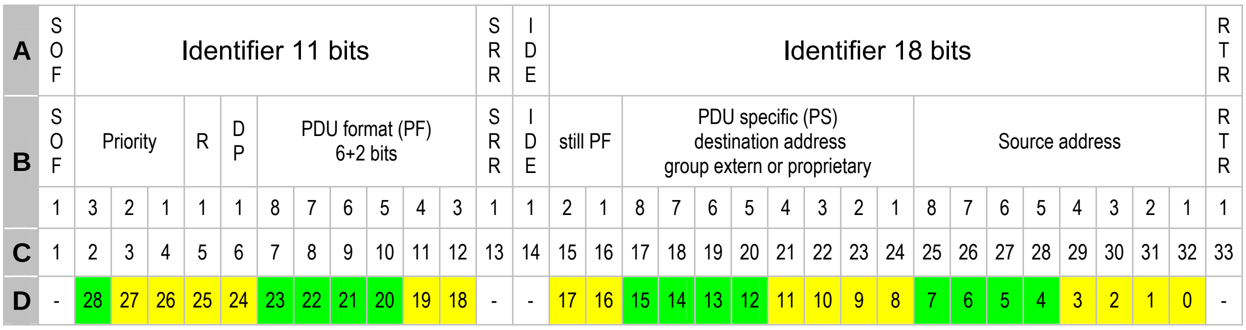
The following information and tools should be available to develop programs for functions to SAE J1939:

- List of the data to be used by the aggregates
- Overview list of the aggregate manufacturer with all relevant data
- CAN monitor with 29-bit support
- If required, the standard SAE J1939

Identifier acc. to SAE J1939

7675

For the data exchange with SAE J1939 the 29 bit identifiers are determinant. This identifier is pictured schematically as follows:



A	SOF	Identifier 11 bits											SRR	IDE	Identifier 18 bits																		RTR
B	SOF	Priority			R	DP	PDU format (PF) 6+2 bits						SRR	IDE	still PF		PDU specific (PS) destination address group extern or proprietary								Source address								RTR
	1	3	2	1	1	1	8	7	6	5	4	3	1	1	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	1
C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
D	-	28	27	26	25	24	23	22	21	20	19	18	-	-	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-

Legend:

A = CAN extended message format
B = J1939 message format
C = J1939 message bit position
D = CAN 29 bit ID position

SOF = **S**tart **o**f **f**rame
SRR = **S**ubstitute **r**emote **r**equest
IDE = **Id**entifier **e**xtension flag
RTR = **R**emote **t**ransmission **r**equest
PDU = **P**rotocol **D**ata **U**nit
PGN = **P**arameter **G**roup **N**umber = PDU format (PF) + PDU source (PS)

(→ CAN-ID (→ page 70))

To do so, the 3 essentially communication methods with SAE J1939 are to be respected:

- destination specific communication with PDU1 (PDU format 0...239)
- broadcast communication with PDU2 (PDU format 240...255)
- proprietary communication with PDU1 or PDU2

Example: detailed message documentation

7679

ETC1: Electronic Transmission Controller #1 (3.3.5) $0CF00203_{16}$

Transmission repetition rate	RPT	10 ms
Data length	LEN	8 Bytes
PDU format	PF	240
PDU specific	PS	2
Default priority	PRIO	3
Data Page	PG	0
Source Address	SA	3
Parameter group number	PGN	$00F002_{16}$
Identifier	ID	$0CF00203_{16}$
Data Field	SRC	The meaning of the data bytes 1...8 is not further described. It can be seen from the manufacturer's documentation.

As in the example of the manufacturer all relevant data has already been prepared, it can be directly transferred to the FBs.

Meaning:

Designation in the manufacturer's documentation	Unit input library function	Example value
Transmission repetition rate	RPT	T#10ms
Data length	LEN	8
PDU format	PF	240
PDU specific	PS	2
Default priority	PRIO	3
Data page	PG	0
Source address / destination address	SA / DA	3
Data field	SRC / DST	array address

Depending on the required function the corresponding values are set. For the fields SA / DA or SRC / DST the meaning (but not the value) changes according to the receive or transmit function.

The individual data bytes must be read from the array and processed according to their meaning.

Example: short message documentation

7680

But even if the aggregate manufacturer only provides a short documentation, the function parameters can be derived from the identifier. In addition to the ID, the "transmission repetition rate" and the meaning of the data fields are also always needed.

If the protocol messages are not manufacturer-specific, the standard SAE J1939 or ISO 11992 can also serve as information source.

Structure of the identifier 0CF00203$_{16}$:

PRIO, reserved, PG		**PF + PS**				**SA / DA**	
0	C	F	0	0	2	0	3

As these values are hexadecimal numbers of which individual bits are sometimes needed, the numbers must be further broken down:

SA / DA		**Source / Destination Address (hexadecimal)**		**Source / Destination Address (decimal)**	
0	3	00	03	0	3

PF		**PDU format (PF) (hexadecimal)**		**PDU format (PF) (decimal)**	
F	0	0F	00	16	0

PS		**PDU specific (PS) (hexadecimal)**		**PDU specific (PS) (decimal)**	
0	2	00	02	0	2

PRIO, reserved, PG		**PRIO, reserved, PG (binary)**	
0	C	0000	1100

Out of the 8 bits (0C$_{16}$) only the 5 least significant bits are needed:

Not necessary			**Priority**			**res.**	**PG**
x	x	x	0_2	1_2	1_2	0_2	0_2
x			03_{10}			0_{10}	0_{10}

Further typical combinations for "PRIO, reserve., PG"

18$_{16}$:

Not necessary			**priority**			**res.**	**PG**
x	x	x	1_2	1_2	0_2	0_2	0_2
x			6_{10}			0_{10}	0_{10}

1C$_{16}$:

Not necessary			**priority**			**res.**	**PG**
x	x	x	1_2	1_2	1_2	0_2	0_2
x			7_{10}			0_{10}	0_{10}

8.5.2 Units for SAE J1939

Contents

J1939_x 105
J1939_x_RECEIVE 107
J1939_x_TRANSMIT 109
J1939_x_RESPONSE 111
J1939_x_SPECIFIC_REQUEST 113
J1939_x_GLOBAL_REQUEST 115

8566

Here you find funktion blocks of the CAN function for SAE J1939.

NOTE

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page 79).

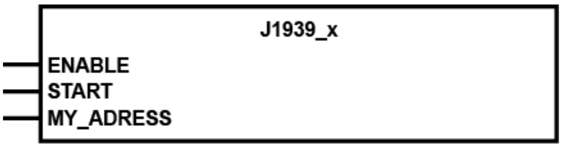
J1939_x

2274

Unit type = function block (FB)

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

Symbol in CoDeSys:



J1939_1

9375

Contained in the library: `ifm_J1939_1_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

J1939_2

9376

Contained in the library: `ifm_J1939_2_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Description

435

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

NOTE

J1939 communication via the 1st CAN interface:

► First initialise the interface via CAN1_EXT (→ page 79)!

J1939 communication via the 2nd CAN interface:

► Initialise the interface first with CAN2 (→ page 86)!

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

The protocol handler is started if the input START is set to TRUE for one cycle.

Using MY_ADDRESS, a device address is assigned to the controller. It must differ from the addresses of the other J1939 bus participants. It can then be read by other bus participants.

Parameters of the inputs

469

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
START	BOOL	TRUE (only for 1 cycle): protocol handler started FALSE: during further processing of the program
MY_ADRESS	BYTE	node ID of the device

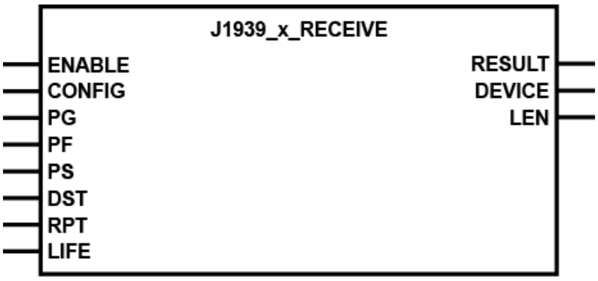
J1939_x_RECEIVE

2278

Unit type = function block (FB)

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

Symbol in CoDeSys:



J1939_1_RECEIVE

9393

Contained in the library: `ifm_J1939_1_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

J1939_2_RECEIVE

9394

Contained in the library: `ifm_J1939_2_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Description

2288

J1939_x_RECEIVE serves for receiving one individual message or a block of messages.

To do so, the FB must be initialised for one cycle via the input CONFIG. During initialisation, the parameters PG, PF, PS, RPT, LIFE and the memory address of the data array DST are assigned.

► The address must be determined by means of the operator ADR and assigned to the FB!

► The receipt of data must be evaluated via the RESULT byte. If RESULT = 1 the data can be read from the memory address assigned via DST and can be further processed.

> When a new message is received, the data in the memory address DST is overwritten.

> The number of received message bytes is indicated via the output LEN.

> If RESULT = 3, no valid messages have been received in the indicated time window (LIFE * RPT).

NOTE

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

Parameters of the inputs

457

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
CONFIG	BOOL	TRUE (only for 1 cycle): for the configuration of the data object FALSE: during further processing of the program
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
DST	DWORD	target address of the array in which the received data is stored ► The address must be determined by means of the operator ADR and assigned to the FB!
RPT	TIME	monitoring time Within this time window the messages must be received repeatedly. Otherwise, an error will be signalled. If no monitoring is requested, RPT must be set to T#0s.
LIFE	BYTE	number of permissible faulty monitoring calls

Parameters of the outputs

458

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data has been received 3 = signalling of errors: nothing has been received during the time window (LIFE*RPT)
DEVICE	BYTE	device address of the sender
LEN	WORD	number of bytes received

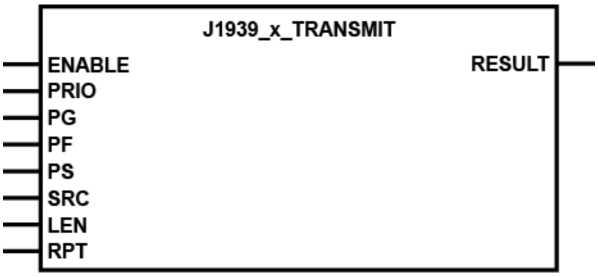
J1939_x_TRANSMIT

2279

Unit type = function block (FB)

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

Symbol in CoDeSys:



J1939_1_TRANSMIT

4322

Contained in the library: `ifm_J1939_1_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

J1939_2_TRANSMIT

4324

Contained in the library: `ifm_J1939_2_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Description

2298

J1939_x_TRANSMIT is responsible for transmitting individual messages or blocks of messages. To do so, the parameters PG, PF, PS, RPT and the address of the data array SRC are assigned to the FB.

- ► The address must be determined by means of the operator ADR and assigned to the FB!
- ► In addition, the number of data bytes to be transmitted and the priority (typically 3, 6 or 7) must be assigned.

Given that the transmission of data is processed via several control cycles, the process must be evaluated via the RESULT byte. All data has been transmitted if RESULT = 1.

Info

If more than 8 bytes are to be sent, a "multi package transfer" is carried out.

Parameters of the inputs

439

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
PRIO	BYTE	message priority (0...7)
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
SRC	DWORD	memory address of the data array whose content is to be transmitted ► The address must be determined by means of the operator ADR and assigned to the FB!
LEN	WORD	number of bytes to be transmitted
RPT	TIME	repeat time during which the data messages are transmitted cyclically

Parameters of the outputs

440

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

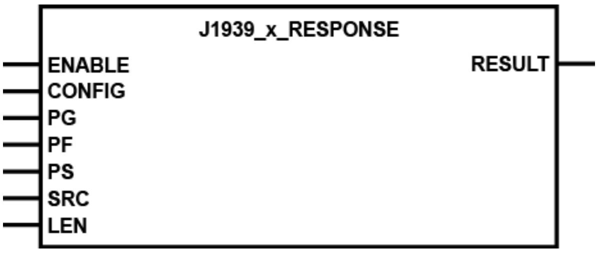
J1939_x_RESPONSE

2280

Unit type = function block (FB)

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

Symbol in CoDeSys:



J1939_1_RESPONSE

9399

Contained in the library: `ifm_J1939_1_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

J1939_2_RESPONSE

9400

Contained in the library: `ifm_J1939_2_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Description

2299

J1939_x_RESPONSE handles the automatic response to a request message.

This FB is responsible for the automatic sending of messages to "Global Requests" and "Specific Requests". To do so, the FB must be initialised for one cycle via the input CONFIG.

The parameters PG, PF, PS, RPT and the address of the data array SRC are assigned to the FB.

► The address must be determined by means of the operator ADR and assigned to the FB!

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

Parameters of the inputs

451

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
CONFIG	BOOL	TRUE (only for 1 cycle): for the configuration of the data object FALSE: during further processing of the program
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
SRC	DWORD	memory address of the data array whose content is to be transmitted ► The address must be determined by means of the operator ADR and assigned to the FB!
LEN	WORD	number of bytes to be transmitted

Parameters of the outputs

440

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

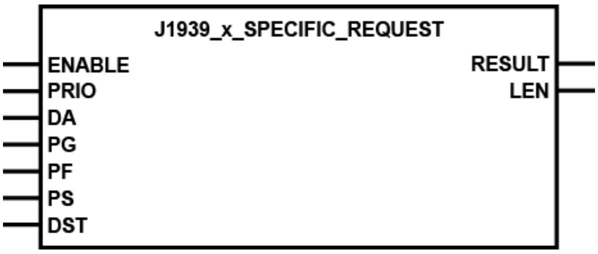
J1939_x_SPECIFIC_REQUEST

2281

Unit type = function block (FB)

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

Symbol in CoDeSys:



J1939_1_SPECIFIC_REQUEST

8884

Contained in the library: `ifm_J1939_1_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

J1939_2_SPECIFIC_REQUEST

8885

Contained in the library: `ifm_J1939_2_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Description

2300

J1939_x_SPECIFIC_REQUEST is responsible for the automatic requesting of individual messages from a specific J1939 network participant. To do so, the logical device address DA, the parameters PG, PF, PS and the address of the array DST in which the received data is stored are assigned to the FB.

► The address must be determined by means of the operator ADR and assigned to the FB!

► In addition, the priority (typically 3, 6 or 7) must be assigned.

► Given that the request of data can be handled via several control cycles, this process must be evaluated via the RESULT byte. All data has been received if RESULT = 1.

> The output LEN indicates how many data bytes have been received.

Parameters of the inputs

445

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
PRIO	BYTE	priority (0...7)
DA	BYTE	logical address (target address) of the called device
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
DST	DWORD	target address of the array in which the received data is stored ► The address must be determined by means of the operator ADR and assigned to the FB!

Parameters of the outputs

446

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data transmission completed 2 = unit active (data transmission) 3 = error, data cannot be sent
LEN	WORD	number of data bytes received

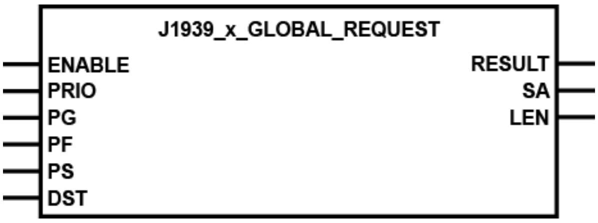
J1939_x_GLOBAL_REQUEST

2282

Unit type = function block (FB)

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

Symbol in CoDeSys:



J1939_1_GLOBAL_REQUEST

4315

Contained in the library: `ifm_J1939_1_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

J1939_2_GLOBAL_REQUEST

9423

Contained in the library: `ifm_J1939_2_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR2500

Description

2301

J1939_x_GLOBAL_REQUEST is responsible for the automatic requesting of individual messages from all (global) active J1939 network participants. To do so, the logical device address DA, the parameters PG, PF, PS and the address of the array DST in which the received data is stored are assigned to the FB.

► The address must be determined by means of the operator ADR and assigned to the FB!

► In addition, the priority (typically 3, 6 or 7) must be assigned.

► Given that the request of data can be handled via several control cycles, this process must be evaluated via the RESULT byte. All data has been received if RESULT = 1.

> The output LEN indicates how many data bytes have been received.

Parameters of the inputs

463

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
PRIO	BYTE	priority (0...7)
PG	BYTE	page address (normally = 0)
PF	BYTE	PDU format byte
PS	BYTE	PDU specific byte
DST	DWORD	target address of the array in which the received data is stored ► The address must be determined by means of the operator ADR and assigned to the FB!

Parameters of the outputs

464

Parameter	Data type	Description
RESULT	BYTE	0 = not active 1 = data transmission completed 2 = unit active (data transmission) 3 = error, data cannot be sent
SA	BYTE	logical device address (sender address) of the called device
LEN	WORD	number of data bytes received

8.6 ifm CANopen libraries

Contents

Technical about CANopen 118
Libraries for CANopen 156

1856

❶ NOTE

The following devices support CANopen only for the 1st CAN interface:
- Controller CR0020, CR200, CR0301, CR0302, CR0303, CR0505, CR2500, CR2501, CR2502, CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- PDM360smart: CR1070, CR1071

If the CANopen master has already been added, the device can no longer be used as a CANopen slave via CoDeSys.

Implementation of a separate protocol on interface 2 or using the protocol to SAE J1939 or ISO 11992 is possible at any time.

The following devices can be used on all CAN interfaces with all protocols:
- BasicController: CR040n
- BasicDisplay: CR0451
- Controller CRnn32
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360NG: CR108n, CR9042

8.6.1 Technical about CANopen

Contents

CANopen network configuration, status and error handling 118
CANopen support by CoDeSys..... 119
CANopen master..... 121
CANopen slave 142
CANopen network variables..... 150

7773

CANopen tables (→ page 306) you will find in the annex.

CANopen network configuration, status and error handling

7471

For all programmable devices the CANopen interface of CoDeSys is used. Whereas the network configuration and parameter setting of the connected devices are directly carried out via the programming software, the error messages can only be reached via nested variable structures in the CANopen stack. The documentation below shows you the structure and use of the network configuration and describes the units of the **ifm** CANopen device libraries.

The chapters CANopen support by CoDeSys (→ page 119), CANopen master (→ page 121), CANopen slave (→ page 142) and CAN network variables (→ page 150) describe the internal units of the CoDeSys CANopen stacks and their use. They also give information of how to use the network configurator.

The chapters concerning the libraries `ifm_CRnnnn_CANopenMaster_Vxxyyzz.lib` and `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` describe all units for error handling and polling the device status when used as master or slave.

NOTE

Irrespective of the device used the structure of the function interfaces of all libraries is the same. The slight differences (e.g. CANOPEN_LED_STATUS) are directly described in the corresponding FBs.

It is absolutely necessary to use only the corresponding device-specific library. The context can be seen from the integrated article number of the device.
Example CR0020: → `ifm_CR0020_CANopenMaster_Vxxyyzz.lib`

→ chapter Setup the target (→ page 21)

When other libraries are used the device can no longer function correctly.

CANopen support by CoDeSys

1857

General information about CANopen with CoDeSys

2075

CoDeSys is one of the leading systems for programming control systems to the international standard IEC 61131. To make CoDeSys more interesting for users many important functions were integrated in the programming system, among them a configurator for CANopen. This CANopen configurator enables configuration of CANopen networks (with some restrictions) under CoDeSys.

CANopen is implemented as a CoDeSys library in IEC 61131-3. The library is based on simple basic CAN functions called CAN driver.

Implementation of the CANopen functions as CoDeSys library enables simple scaling of the target system. The CANopen function only uses target system resources if the function is really used. To use target system resources carefully CoDeSys automatically generates a data basis for the CANopen master function which exactly corresponds to the configuration.

From the CoDeSys programming system version 2.3.6.0 onwards an **ecomat*mobile*** controller can be used as CANopen master and as CANopen slave.

NOTE:

For all **ecomat*mobile*** controllers and the PDM360smart you must use CANopen libraries with the following addition:

- For CR0032 target version up to V01, all other devices up to V04.00.05: "**OptTable**"
- For CR0032 target version from V02 onwards, all other devices from V05 onwards: "**OptTableEx**"

If a new project is created, these libraries are in general automatically loaded. If you add the libraries via the library manager, you must ensure a correct selection.

The CANopen libraries without this addition are used for all other programmable devices (e.g. PDM360compact).

CANopen terms and implementation

1858

According to the CANopen specification there are no masters and slaves in a CAN network. Instead of this there is an NMT master (NMT = network management), a configuration master, etc. according to CANopen. It is always assumed that all participants of a CAN network have equal rights.

Implementation assumes that a CAN network serves as periphery of a CoDeSys programmable controller. As a result of this an **ecomat*mobile*** controller or a PDM360 display is called CANopen master in the CAN configurator of CoDeSys. This master is an NMT master and configuration master. Normally the master ensures that the network is put into operation. The master takes the initiative to start the individual nodes (= network nodes) known via the configuration. These nodes are called slaves.

To bring the master closer to the status of a CANopen slave an object directory was introduced for the master. The master can also act as an SDO server (SDO = Service Data Object) and not only as SDO client in the configuration phase of the slaves.

IDs (addresses) in CANopen

3952

In CANopen there are different types of addresses (IDs):

- COB ID
  The **C**ommunication **Ob**ject **Id**entifier addresses the message (= the communication object) in the list of devices. A communication object consists of one or more CAN messages with a specific functionality, e.g.
  - PDO (**P**rocess **D**ata **O**bject = message object with process data),
  - SDO (**S**ervice **D**ata **O**bject.= message object with service data),
  - emergency (message object with emergency data),
  - time (message object with time data) or
  - error control (message object with error messages).
- CAN ID
  The **CAN Id**entifier defines CAN messages in the complete network. The CAN ID is the main part of the arbitration field of a CAN data frame. The CAN ID value determines implicity the priority for the bus arbitration.
- Download ID
  The download ID indicates the node ID for service communication via SDO for the program download and for debugging.
- Node ID
  The **Node Id**entifier is a unique descriptor for CANopen devices in the CAN network. The Node ID is also part of some pre-defined connectionsets (→ function code (→ page 308)).

Comparison of download-ID vs. COB-ID:

Controller program download		**CANopen**	
Download ID	**COB ID SDO**	**Node ID**	**COB ID SDO**
1…127	TX: 580_{16} + download ID	1…127	TX: 580_{16} + node ID
	RX: 600_{16} + download ID		RX: 600_{16} + node ID

TX = slave sends to master
RX = slave receives from master

CANopen master

Contents

Differentiation from other CANopen libraries 121
Create a CANopen project 123
Add and configure CANopen slaves 127
Master at runtime 131
Start the network 133
Network states 134

1859

Differentiation from other CANopen libraries

1990

The CANopen library implemented by 3S (Smart Software Solutions) differentiates from the systems on the market in various points. It was not developed to make other libraries of renowned manufacturers unnecessary but was deliberately optimised for use with the CoDeSys programming and runtime system.

The libraries are based on the specifications of CiA DS301, V402.

For users the advantages of the CoDeSys CANopen library are as follows:

- Implementation is independent of the target system and can therefore be directly used on every controller programmable with CoDeSys.
- The complete system contains the CANopen configurator and integration in the development system.
- The CANopen functionality is reloadable. This means that the CANopen FBs can be loaded and updated without changing the operating system.
- The resources of the target system are used carefully. Memory is allocated depending on the used configuration, not for a maximum configuration.
- Automatic updating of the inputs and outputs without additional measures.

The following functions defined in CANopen are at present supported by the **ifm** CANopen library:

- **Transmitting PDOs:** master transmits to slaves (slave = node, device)
  Transmitting event-controlled (i.e. in case of a change), time-controlled (RepeatTimer) or as synchronous PDOs, i.e. always when a SYNC was transmitted by the master. An external SYNC source can also be used to initiate transmission of synchronous PDOs.
- **Receiving PDOs:** master receives from slave
  Depending on the slave: event-controlled, request-controlled, acyclic and cyclic.
- **PDO mapping**
  Assignment between a local object directory and PDOs from/to the CANopen slave (if supported by the slave).
- **Transmitting and receiving SDOs** (unsegmented, i.e. 4 bytes per entry in the object directory)
  Automatic configuration of all slaves via SDOs at the system start.
  Application-controlled transmission and reception of SDOs to/from configured slaves.
- **Synchronisation**
  Automatic transmission of SYNC messages by the CANopen master.

- **Nodeguarding**
  Automatic transmission of guarding messages and lifetime monitoring for every slave configured accordingly.
  We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.
- **Heartbeat**
  Automatic transmission and monitoring of heartbeat messages.
- **Emergency**
  Reception of emergency messages from the configured slaves and message storage.
- Set **Node-ID** and **baud rate** in the slaves
  By calling a simple function, node ID and baud rate of a slave can be set at runtime of the application.

The following functions defined in CANopen are at present **not** supported by the CANopen 3S (Smart Software Solutions) library:

- Dynamic identifier assignment,
- Dynamic SDO connections,
- SDO transfer block by block, segmented SDO transfer (the functionality can be implemented via CANx_SDO_READ (→ page 178) and CANx_SDO_WRITE (→ page 180) in the corresponding **ifm** device library).
- All options of the CANopen protocol which are not mentioned above.

Create a CANopen project

1860

Below the creation of a new project with a CANopen master is completely described step by step. It is assumed that you have already installed CoDeSys on your processor and the Target and EDS files have also been correctly installed or copied.

► A more detailed description for setting and using the dialogue [controller and CANopen configuration] is given in the CoDeSys manual under [Resources] > [PLC Configuration] or in the Online help.

> After creation of a new project (→ chapter Setup the target (→ page 21)) the CANopen master must first be added to the controller configuration via [Insert] > [Append subelement]. For controllers with 2 or more CAN interfaces interface 1 is automatically configured for the master.

> The following libraries and software modules are automatically integrated:

- The `Standard.LIB` which provides the standard functions for the controller defined in IEC `61131`.
- The `3S_CanOpenManager.LIB` which provides the CANopen basic functionalities (possibly `3S_CanOpenManagerOptTable.LIB` for the C167 controller)
- One or several of the libraries `3S_CANopenNetVar.LIB`, `3S_CANopenDevice.LIB` and `3S_CANopenMaster.LIB` (possibly `3S_...OptTable.LIB` for the C167 controller) depending on the requested functionality
- The system libraries `SysLibSem.LIB` and `SysLibCallback.LIB`
- To use the prepared network diagnostic, status and EMCY functions, the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` must be manually added to the library manager. Without this library the network information must be directly read from the nested structures of the CoDeSys CANopen libraries.

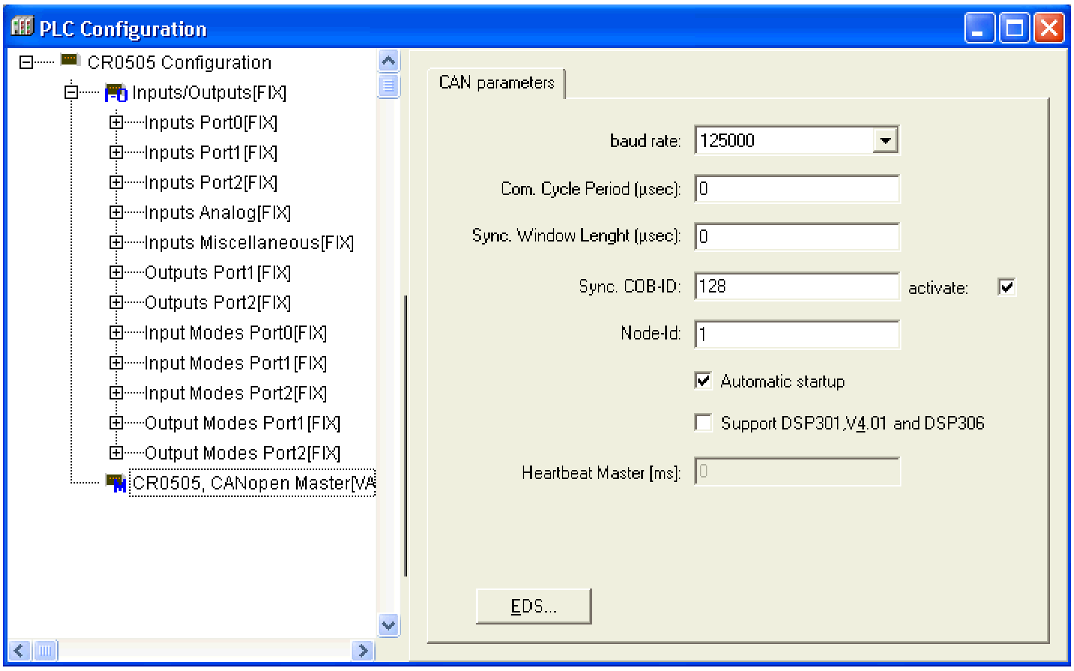
> The following libraries and software modules must still be integrated:

- The device library for the corresponding hardware, e.g. `ifm_CR0020_Vxxyyzz.LIB`. This library provides all device-specific functions.
- EDS files for all slaves to be operated on the network. The EDS files are provided for all CANopen slaves by **ifm electronic**. → chapter Set up programming system via templates (→ page 24)
  For the EDS files of other manufacturers' nodes contact the corresponding manufacturer.

CANopen master: Tab [CAN parameters]

1967

The most important parameters for the master can be set in this dialogue window. If necessary, the contents of the master EDS file can be viewed via the button [EDS...]. This button is only indicated if the EDS file (e.g. CR0020MasterODEntry.EDS) is in the directory `...\CoDeSys V2.3\Library\PLCConf`.
During the compilation of the application program the object directory of the master is automatically generated from this EDS file.



baud rate: 125000
Com. Cycle Period (µsec): 0
Sync. Window Lenght (µsec): 0
Sync. COB-ID: 128 activate: ☑
Node-Id: 1
☑ Automatic startup
☐ Support DSP301,V4.01 and DSP306
Heartbeat Master [ms]: 0

CAN parameters: Baud rate

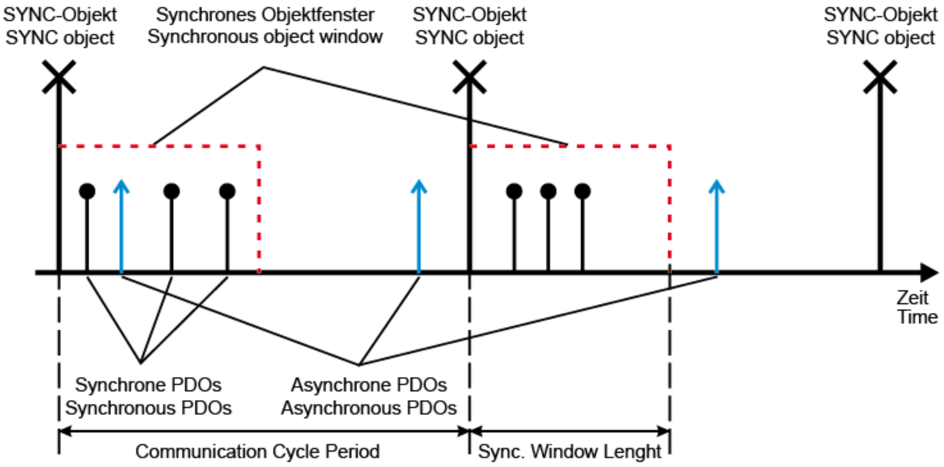
10028

Select the baud rate for the master.
The baud rate must correspond to the transmission speed of the other network participants.

CAN parameters: Communication Cycle Period/Sync. Window Length

10029

After expiry of the [Communication Cycle Period] a SYNC message is transmitted by the master.



The [Sync. Window Length] indicates the time during which synchronous PDOs are transmitted by the other network participants and must be received by the master.

As in most applications no special requirements are made for the SYNC object, the same time can be set for [Communication Cycle Period] and [Sync. Window Length].
Please ensure the time is entered in [µs] (the value 50 000 corresponds to 50 ms).

CAN parameters: Sync. COB ID

10030

In this field the identifier for the SYNC message can be set. It is always transmitted after the communication cycle period has elapsed. The default value is 128 and should normally not be changed. To activate transmission of the SYNC message, the checkbox [activate] must be set.

NOTE

The SYNC message is always generated at the start of a program cycle. The inputs are then read, the program is processed, the outputs are written to and then all synchronous PDOs are transmitted.

Please note that the SYNC time becomes longer if the set SNYC time is shorter than the program cycle time.

Example: communication cycle period = 10 ms and program cycle time = 30 ms.
The SYNC message is only transmitted after 30 ms.

CAN parameters: Node ID

10031

Enter the node number (not the download ID!) of the master in this field.
The node number may only occur once in the network, otherwise the communication is disturbed.

CAN parameters: Automatic startup

10032

After successful configuration the network and the connected nodes are set to the state [operational] and then started.

If the checkbox is not activated, the network must be started manually.

CAN parameters: Heartbeat

10033

If the other participants in the network support heartbeat, the option [support DSP301, V4.01...] can be selected. If necessary, the master can generate its own heartbeat signal after the set time has elapsed.

Add and configure CANopen slaves

Contents

CANopen slave: Tab [CAN parameters] 128
Tab [Receive PDO-Mapping] and [Send PDO-Mapping].. 129
Tab [Service Data Objects] 130

1861

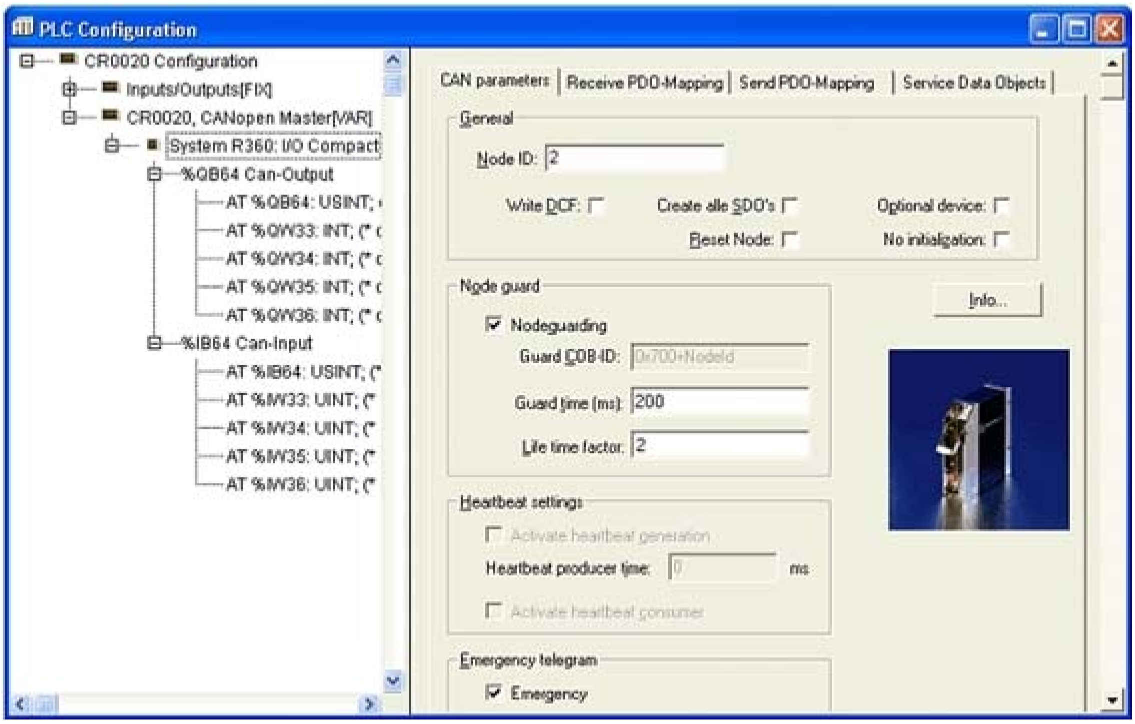
Next you can add the CANopen slaves. To do so, you must call again the dialogue in the controller configuration [Insert] > [Append subelement]. A list of the CANopen device descriptions (EDS files) stored in the directory PLC_CONF is available. By selecting the corresponding device it is directly added to the tree of the controller configuration.

NOTE

If a slave is added via the configuration dialogue in CoDeSys, source code is dynamically integrated in the application program for every node. At the same time every additionally inserted slave extends the cycle time of the application program. This means: In a network with many slaves the master can process no further time-critical tasks (e.g. FB OCC_TASK).

A network with 27 slaves has a basic cycle time of 30 ms.

Please note that the maximum time for a PLC cycle of approx. 50 ms should not be exceeded (watchdog time: 100 ms).



CR0020 Configuration
  Inputs/Outputs[FIX]
  CR0020, CANopen Master[VAR]
    System R360: I/O Compact
      %QB64 Can-Output
        AT %QB64: USINT;
        AT %QW33: INT; (*
        AT %QW34: INT; (*
        AT %QW35: INT; (*
        AT %QW36: INT; (*
      %IB64 Can-Input
        AT %IB64: USINT; (*
        AT %IW33: UINT; (*
        AT %IW34: UINT; (*
        AT %IW35: UINT; (*
        AT %IW36: UINT; (*

General
Node ID: 2
Write DCF: ☐ Create alle SDO's ☐ Optional device: ☐
Reset Node: ☐ No initialization: ☐

Node guard
☑ Nodeguarding
Guard COB-ID: 0x700+NodeId
Guard time (ms): 200
Life time factor: 2

Heartbeat settings
☐ Activate heartbeat generation
Heartbeat producer time: 0 ms
☐ Activate heartbeat consumer

Emergency telegram
☑ Emergency

CANopen slave: Tab [CAN parameters]

1968

CAN parameters: Node ID

10036

The node ID is used to clearly identify the CAN module and corresponds to the number on the module set between 1 and 127.

The ID is entered decimally and is automatically increased by 1 if a new module is added.

CAN parameters: Write DCF

10037

If [Write DCF] is activated, a DCF file is created after adding an EDS file to the set directory for compilation files. The name of the DCF file consists of the name of the EDS file and appended node ID.

CAN parameters: Create all SDOs

10038

If this option is activated, SDOs are generated for all communication objects.

Default values are not written again!

CAN parameters: Node reset

10039

The slave is reset ("load") as soon as the configuration is loaded to the controller.

CAN parameters: Optional device

10040

If the option [optional device] is activated, the master tries only once to read from this node. In case of a missing response, the node is ignored and the master goes to the normal operating state.

If the slave is connected to the network and detected at a later point in time, it is automatically started. To do so, you must have selected the option [Automatic startup] in the CAN parameters of the master.

CAN parameters: No initialization

10041

If this option is activated, the master immediately takes the node into operation without transmitting configuration SDOs. (Nevertheless, the SDO data is generated and stored in the controller.)

CAN parameters: Nodeguarding / heartbeat settings

10042

Depending on the device you can choose:
- [nodeguarding] and [life time factor] must be set OR
- [heartbeat] must be set.

We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.

CAN parameters: Emergency telegram

10043

This option is normally selected. The EMCY messages are transferred with the specified identifier.

CAN parameters: Communication cycle

10044

In special applications a monitoring time for the SYNC messages generated by the master can be set here.

Please note that this time must be longer than the SYNC time of the master. The optimum value must be determined experimentally, if necessary.

In most cases nodeguarding and heartbeat are sufficient for node monitoring.

Tab [Receive PDO-Mapping] and [Send PDO-Mapping]

1969

With the tabs [Receive PDO-Mapping] and [Send PDO-Mapping] in the configuration dialogue of a CAN module the module mapping (assignment between local object directory and PDOs from/to the CANopen slave) described in the EDS file can be changed (if supported by the CAN module).

All [mappable] objects of the EDS file are available on the left and can be added to or removed from the PDOs (Process Data Objects) on the right.

The [StandardDataTypes] can be added to generate spaces in the PDO.

PDO-Mapping: Insert

10046

With the button [Insert] you can generate more PDOs and insert the corresponding objects. The inputs and outputs are assigned to the IEC addresses via the inserted PDOs.

In the controller configuration the settings made can be seen after closing the dialogue. The individual objects can be given symbolic names.

PDO-Mapping: Properties

10047

The PDO properties defined in the standard can be edited in a dialogue via properties.

COB-ID	Every PDO message requires a clear COB ID (communication object identifier). If an option is not supported by the module or the value must not be changed, the field is grey and cannot be edited.
Inhibit Time	The inhibit time (100 µs) is the minimum time between two messages of this PDO so that the messages which are transferred when the value is changed are not transmitted too often. The unit is 100 µs.
Transmission Type	For transmission type you receive a selection of possible transmission modes for this module: **acyclic – synchronous** After a change the PDO is transferred with the next SYNC. **cyclic – synchronous** The PDO is transferred synchronously. [Number of SYNCs] indicates the number of the synchronisation messages between two transmissions of this PDO. **asynchronous – device profile specific** The PDO is transmitted on event, i.e. when the value is changed. The device profile defines which data can be transferred in this way. **asynchronous – manufacturer specific** The PDO is transmitted on event, i.e. when the value is changed. The device manufacturer defines which data is transferred in this way. **(a)synchronous – RTR only** These services are not implemented. **Number of SYNCs** Depending on the transmission type this field can be edited to enter the number of synchronisation messages (definition in the CAN parameter dialogue of [Com. Cycle Period], [Sync Window Length], [Sync. COB ID]) after which the PDO is to be transmitted again. **Event-Time** Depending on the transmission type the period in milliseconds [ms] required between two transmissions of the PDO is indicated in this field.

Tab [Service Data Objects]

1970

Index, name, value, type and default

Here all objects of the EDS or DCF file are listed which are in the range from index 2000_{16} to $9FFF_{16}$ and defined as writable. Index, name, value, type and default are indicated for every object. The value can be changed. Select the value and press the [space bar]. After the change you can confirm the value with the button [Enter] or reject it with [ESC].

For the initialisation of the CAN bus the set values are transferred as SDOs (Service Data Object) to the CAN module thus having direct influence on the object directory of the CANopen slave. Normally they are written again at every start of the application program – irrespective of whether they are permanently stored in the CANopen slave.

Master at runtime

Contents

Reset of all configured slaves on the bus at the system start 131
Polling of the slave device type 131
Configuration of all correctly detected devices 131
Automatic configuration of slaves 132
Start of all correctly configured slaves 132
Cyclical transmission of the SYNC message 132
Nodeguarding with lifetime monitoring 132
Heartbeat from the master to the slaves 132
Reception of emergency messages 132

8569

Here you find information about the functionality of the CANopen master libraries at runtime.

The CANopen master library provides the CoDeSys application with implicit services which are sufficient for most applications. These services are integrated for users in a transparent manner and are available in the application without additional calls. The following description assumes that the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` was manually added to the library manager to use the network diagnostic, status and EMCY functions.

Services of the CANopen master library:

Reset of all configured slaves on the bus at the system start

8570

To reset the slaves, the NMT command "Reset Remote Node" is used as standard explicitly for every slave separately. (NMT stands for **N**etwork **M**anagemen**t** according to CANopen. The individual commands are described in the CAN document DSP301.) In order to avoid overload of slaves having less powerful CAN controllers it is useful to reset the slaves using the command "All Remote Nodes". The service is performed for **all** configured slaves using CANx_MASTER_STATUS (→ page 162) with GLOBAL_START=TRUE. If the slaves are to be reset **individually**, this input must be set to FALSE.

Polling of the slave device type

8021

Polling of the slave device type using SDO (polling for object 100016) and comparison with the configured slave ID:

Indication of an error status for the slaves from which a wrong device type was received. The request is repeated after 0.5 s if ...
- no device type was received
- AND the slave was **not** identified as optional in the configuration
- AND the timeout has **not** elapsed.

Configuration of all correctly detected devices

8022

Every SDO is monitored for a response and repeated if the slave does not respond within the monitoring time.

Automatic configuration of slaves

8023

Automatic configuration of slaves using SDOs while the bus is in operation:
Prerequisite: The slave logged in the master via a bootup message.

Start of all correctly configured slaves

8574

Start of all correctly configured slaves after the end of the configuration of the corresponding slave:

To start the slaves the NMT command "Start remote node" is normally used. As for the "reset" this command can be replaced by "Start All Remote Nodes".

The service can be called via CANx_Master_STATUS with GLOBAL_START=TRUE.

Cyclical transmission of the SYNC message

8025

This value can only be set during the configuration.

Nodeguarding with lifetime monitoring

8576

Setting of nodeguarding with lifetime monitoring for every slave possible:

The error status can be monitored for max. 8 slaves via CANx_MASTER_STATUS with ERROR_CONTROL=TRUE.

We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.

Heartbeat from the master to the slaves

8577

The error status can be monitored for max. 8 slaves via CANx_MASTER_STATUS with ERROR_CONTROL=TRUE.

Reception of emergency messages

8578

Reception of emergency messages for every slave, the emergency messages received last are stored separately for every slave:

The error messages can be read via CANx_MASTER_STATUS with EMERGENCY_OBJECT_SLAVES=TRUE.

In addition this FB provides the EMCY message generated last on the output GET_EMERGENCY.

Start the network

1863

Here you find information about how to start the CANopen network.

After downloading the project to the controller or a reset of the application the master starts up the CAN network again. This always happens in the same order of actions:

- All slaves are reset unless they are marked as "No initialization" in the configurator. They are reset individually using the NMT command "Reset Node" (81_{16}) with the node ID of the slave. If the flag GLOBAL_START was set via CANx_MASTER_STATUS (→ page 162), the command is used once with the node ID 0 to start up the network.
- All slaves are configured. To do so, the object 1000_{16} of the slave is polled.
  - If the slave responds within the monitoring time of 0.5 s, the next configuration SDO is transmitted.
  - If a slave is marked as "optional" and does not respond to the polling for object 1000_{16} within the monitoring time, it is marked as not available and no further SDOs are transmitted to it.
  - If a slave responds to the polling for object 1000_{16} with a type other than the configured one (in the lower 16 bits), it is configured but marked as a wrong type.
- All SDOs are repeated as long as a response of the slave was seen within the monitoring time. Here the application can monitor start-up of the individual slaves and possibly react by setting the flag SET_TIMEOUT_STATE in the NODE_STATE_SLAVE array of CANx_MASTER_STATUS.
- If the master configured a heartbeat time unequal to 0, the heartbeat is generated immediately after the start of the master controller.
- After all slaves have received their configuration SDOs, guarding starts for slaves with configured nodeguarding.
- If the master was configured to [Automatic startup], all slaves are now started individually by the master. To do so, the NMT command "Start Remote Node" (1_{16}) is used. If the flag GLOBAL_START was set via CANx_Master_STATUS, the command is used with the node ID 0 and so all slaves are started with "Start all Nodes".
- All configured TX-PDOs are transmitted at least once (for the slaves RX-PDOs).
- If [Automatic startup] is deactivated, the slaves must be started separately via the flag START_NODE in the NODE_STATE_SLAVE array or via the input GLOBAL_START of CANx_MASTER_STATUS.

Network states

Contents

Boot up of the CANopen master 134
Boot up of the CANopen slaves 135
Start-up of the network without [Automatic startup] 138
The object directory of the CANopen master 140

1864

Here you read how to interpret the states of the CANopen network and how to react.

For the start-up (→ page 133) of the CANopen network and during operation the individual functions of the library pass different states.

! NOTE

In the monitor mode (online mode) of CoDeSys the states of the CAN network can be seen in the global variable list "CANOpen implicit variables". This requires exact knowledge of CANopen and the structure of the CoDeSys CANopen libraries.

To facilitate access CANx_MASTER_STATUS (→ page 162) from the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` is available.

Boot up of the CANopen master

1971

During boot-up of the CAN network the master passes different states which can be read via the output NODE_STATE of CANx_MASTER_STATUS (→ page 162).
(Network state of the master → next chapter)

Whenever a slave does not respond to an SDO request (upload or download), the request is repeated. The master leaves state 3, as described above, but not before all SDOs have been transmitted successfully. So it can be detected whether a slave is missing or whether the master has not correctly received all SDOs. It is of no importance for the master whether a slave responds with an acknowledgement or an abort. It is only important for the master whether he received a response at all.

An exception is a slave marked as "optional". Optional slaves are asked for their 1000_h object only once. If they do not respond within 0.5 s, the slave is first ignored by the master and the master goes to state 5 without further reaction of this slave.

NMT state for CANopen master

9964

State hex	dec	Description
00	0	not defined
01	1	Master waits for a boot-up message of the node. OR: Master waits for the expiry of the given guard time.
02	2	- Master waits for 300 ms. - Master requests the object 1000_{16}. - Then the state is set to 3.
03	3	The master configures its slaves. To do so, all SDOs generated by the configurator are transmitted to the slaves one after the other: - The Master sends to the slave a SDO read request (index 1000_{16}). - The generated SDOs are compressed into a SDO array. - The slave knows it's first SDO and the number of it's SDOs.
05	5	After transmission of all SDOs to the slaves the master goes to state 5 and remains in this state. State 5 is the normal operating state for the master.

To read the node state out of the FB:

Used function block	Node state is found here
CANx_MASTER_STATUS CANx_SLAVE_STATUS	output NODE_STATE
CANOPEN_GETSTATE	output NODESTATE

Boot up of the CANopen slaves

1972

You can read the states of a slave via the array NODE_STATE_SLAVE of CANx_MASTER_STATUS (→ page 162). During boot up of the CAN network the slave passes the states -1, 1 and 2 automatically.
(Network state of the slave → next chapter)

NMT state for CANopen slave

9965

State hex	dec	Description
FF	-1	The slave is reset by the NMT message "Reset Node" and automatically goes to state 1.
00	0	not defined
01	1	state = waiting for BOOTUP After max. 2 s or immediately on reception of its boot up message the slave goes to state 2.
02	2	state = BOOTUP After a delay of 0.5 s the slave automatically goes to state 3.
03	3	state = PREPARED The slave is configured in state 3. The slave remains in state 3 as long as it has received all SDOs generated by the configurator. It is not important whether during the slave configuration the response to SDO transfers is abort (error) or whether the response to all SDO transfers is no error. Only the response as such received by the slave is important – not its contents. If in the configurator the option "Reset node" has been activated, a new reset of the node is carried out after transmitting the object 1011_{16} sub-index 1 which then contains the value "load". The slave is then polled again with the upload of the object 1000_{16}. Slaves with a problem during the configuration phase remain in state 3 or directly go to an error state (state > 5) after the configuration phase.
04	4	state = PRE-OPERATIONAL A node always goes to state 4 except for the following cases: • it is an "optional" slave and it was detected as non available on the bus (polling for object 1000_{16}) OR: • the slave is present but reacted to the polling for object 1000_{16} with a type in the lower 16 bits other than expected by the configurator.
05	5	state = OPERATIONAL State 5 is the normal operating state of the slave: [Normal Operation]. If the master was configured to [Automatic startup], the slave starts in state 4 (i.e. a "start node" NMT message is generated) and the slave goes automatically to state 5. If the flag GLOBAL_START was set, the master waits until all slaves are in state 4. All slaves are then started with the NMT command [Start All Nodes].
61	97	A node goes to state 97 if it is optional (optional device in the CAN configuration) and has not reacted to the SDO polling for object 1000_{16}. If the slave is connected to the network and detected at a later point in time, it is automatically started. To do so, you must have selected the option [Automatic startup] in the CAN parameters of the master.
62	98	A node goes to state 98 if the device type (object 1000_{16}) does not correspond to the configured type.
63	99	In case of a nodeguarding timeout the slave is set to state 99. As soon as the slave reacts again to nodeguard requests and the option [Automatic startup] is activated, it is automatically started by the master. Depending on the status contained in the response to the nodeguard requests, the node is newly configured or only started. To start the slave manually it is sufficient to use the method [NodeStart].

Nodeguard messages are transmitted to the slave ...
- if the slave is in state 4 or higher AND
- if nodeguarding was configured.

To read the node state out of the FB:

Used function block	Node state is found here
CANx_MASTER_STATUS CANx_SLAVE_STATUS	output NODE_STATE
CANOPEN_GETSTATE	output NODESTATE

CANopen status of the node

1973

Node status according to CANopen (with these values the status is also coded by the node in the corresponding messages).

Status hex	dec		CANopen status	Description
00	0	BOOTUP	Node received the boot-up message.
04	4	PREPARED	Node is configured via SDOs.
05	5	OPERATIONAL	Node participates in the normal exchange of data.
7F	127	PRE-OPERATIONAL	Node sends no data, but can be configred by the master.

If nodeguarding active: the most significant status bit toggles between the messages.

Read the node status from the function block:

Function block used	Node status is found here
CANx_MASTER_STATUS CANx_SLAVE_STATUS	Structure element LAST_STATE from the array NODE_STATE_SLAVE
CANOPEN_GETSTATE	Output LASTNODESTATE

Start-up of the network without [Automatic startup]

Contents

Starting the network with GLOBAL_START 138
Starting the network with START_ALL_NODES 138
Initialisation of the network with RESET_ALL_NODES 139
Access to the status of the CANopen master 139

8583

Sometimes it is necessary that the application determines the instant to start the CANopen slaves. To do so, the option [Automatic startup] of the CANopen master must be deactivated in the configuration. It is then up to the application to start the slaves.

Starting the network with GLOBAL_START

1974

In a CAN network with many participants (in most cases more than 8) it often happens that NMT messages in quick succession are not detected by all (mostly slow) IO nodes (e.g. CompactModules CR2013). The reason for this is that these nodes must listen to all messages with the ID 0. NMT messages transmitted at too short intervals overload the receive buffer of such nodes.

A help for this is to reduce the number of NMT messages in quick succession.

► To do so, set the input GLOBAL_START of CANx_Master_STATUS (→ page 162) to TRUE (with [Automatic startup]).

> The CANopen master library uses the command "Start All Nodes" instead of starting all nodes individually using the command "Start Node".

> GLOBAL_START is executed only once when the network is initialised.

> If this input is set, the controller also starts nodes with status 98 (see above). However, the PDOs for these nodes remain deactivated.

Starting the network with START_ALL_NODES

1975

If the network is not automatically started with GLOBAL_START of CANx_Master_STATUS (→ page 162), it can be started at any time, i.e. every node one after the other. If this is not requested, the option is as follows:

► Set the input START_ALL_NODES of CANx_Master_STATUS to TRUE.
START_ALL_NODES is typically set by the application program at runtime.

> If this input is set, nodes with status 98 (see above) are started. However, the PDOs for these nodes remain deactivated.

Initialisation of the network with RESET_ALL_NODES

1976

The same reasons which apply to the command START_ALL_NODES also apply to the NMT command RESET_ALL_NODES (instead of RESET_NODES for every individual node).

► To do so, the input RESET_ALL_NODES of CANx_MASTER_STATUS (→ page 162) must be set to TRUE.

> This resets all nodes once at the same time.

Access to the status of the CANopen master

1977

You should poll the status of the master so that the application code is not processed before the IO network is ready. The following code fragment example shows one option:

Variable declaration

```
VAR
    FB_MasterStatus := CR0020_MASTER_STATUS;
    :
END_VAR
```

program code

```
If   FB_MasterStatus.NODE_STATE = 5 THEN
     <application code>
END_IF
```

By setting the flag TIME_OUT_STATE in the array NODE_STATE_SLAVE of CANx_Master_STATUS (→ page 162) the application can react and, for example, jump the non configurable node.

The object directory of the CANopen master

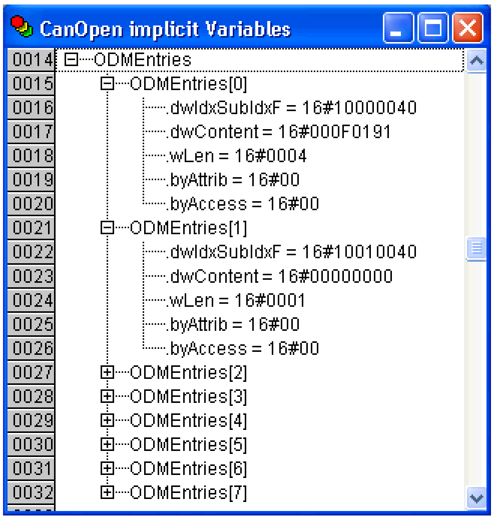
1978

In some cases it is helpful if the CANopen master has its own object directory. This enables, for example, the exchange of data of the application with other CAN nodes.

The object directory of the master is generated using an EDS file named `CRnnnnMasterODEntry.EDS` during compilation and is given default values. This EDS file is stored in the directory `CoDeSys Vn\Library\PLCconf`. The content of the EDS file can be viewed via the button [EDS...] in the configuration window [CAN parameters].

Even if the object directory is not available, the master can be used without restrictions.

The object directory is accessed by the application via an array with the following structure:



```
0014 ⊟ODMEntries
0015     ⊟ODMEntries[0]
0016         .dwIdxSubIdxF = 16#10000040
0017         .dwContent = 16#000F0191
0018         .wLen = 16#0004
0019         .byAttrib = 16#00
0020         .byAccess = 16#00
0021     ⊟ODMEntries[1]
0022         .dwIdxSubIdxF = 16#10010040
0023         .dwContent = 16#00000000
0024         .wLen = 16#0001
0025         .byAttrib = 16#00
0026         .byAccess = 16#00
0027     ⊞ODMEntries[2]
0028     ⊞ODMEntries[3]
0029     ⊞ODMEntries[4]
0030     ⊞ODMEntries[5]
0031     ⊞ODMEntries[6]
0032     ⊞ODMEntries[7]
```

Structure element	Description
dwIdxSubIdxF	Structure of the component 16#iiiissff: iiii – index (2 bytes, bits 16...31), Idx ss – sub-index (1 byte, bits 8...15), SubIdx ff – flags (1 byte, bits 0...7), F Meaning of the flag bits: bit 0: write bit 1: content is a pointer to an address bit 2: mappable bit 3: swap bit 4: signed value bit 5: floating point bit 6: contains more sub-indices
dwContent	contains the contents of the entry
wLen	length of the data
byAttrib	initially intended as access authorisation can be freely used by the application of the master
byAccess	in the past access authorisation can be freely used by the application of the master

On the platform CoDeSys has no editor for this object directory.

The EDS file only determines the objects used to create the object directory. The entries are always generated with length 4 and the flags (least significant byte of the component of an object directory entry dwIdxSubIdxF) are always given the value 1. This means both bytes have the value 41_{16}.

If an object directory is available in the master, the master can act as SDO server in the network. Whenever a client accesses an entry of the object directory by writing, this is indicated to the application via the flag OD_CHANGED in CANx_MASTER_STATUS (→ page 162). After evaluation this flag must be reset.

The application can use the object directory by directly writing to or reading the entries or by pointing the entries to IEC variables. This means: when reading/writing to another node these IEC variables are directly accessed.

If index and sub-index of the object directory are known, an entry can be addressed as follows:

```
I := GetODMEntryValue(16#iiiiss00, pCanOpenMaster[0].wODMFirstIdx,
pCanOpenMaster[0].wODMFirstIdx + pCanOpenMaster[0]. wODMCount;
```

For "iii" the index must be used and for "ss" the sub-index (as hex values).

The number of the array entry is available in I. You can now directly access the components of the entry.
It is sufficient to enter address, length and flags so that this entry can be directly transferred to an IEC variable:

```
ODMEntries[I].dwContent := ADR(<variable name>);
ODMEntries[I].wLen := sizeof(<variable name>);
ODMEntries[I]. dwIdxSubIdxF := ODMEntries[I]. dwIdxSubIdxF OR
OD_ENTRYFLG_WRITE OR OD_ENTRYFLG_ISPOINTER;
```

It is sufficient to change the content of "dwContent" to change only the content of the entry.

CANopen slave

Contents

Functionality of the CANopen slave library 142
CANopen slave configuration 143
Access to the CANopen slave at runtime 149

1865

A CoDeSys programmable controller can also be a CANopen slave in a CAN network.

Functionality of the CANopen slave library

1979

The CANopen slave library in combination with the CANopen configurator provides the user with the following options:

- In CoDeSys: configuration of the properties for nodeguarding/heartbeat, emergency, node ID and baud rate at which the device is to operate.
- Together with the parameter manager in CoDeSys, a default PDO mapping can be created which can be changed by the master at runtime. The PDO mapping is changed by the master during the configuration phase. By means of mapping IEC variables of the application can be mapped to PDOs. This means IEC variables are assigned to the PDOs to be able to easily evaluate them in the application program.
- The CANopen slave library provides an object directory. The size of this object directory is defined while compiling CoDeSys. This directory contains all objects which describe the CANopen slave and in addition the objects defined by the parameter manager. In the parameter manager only the list types parameters and variables can be used for the CANopen slave.
- The library manages the access to the object directory, i.e. it acts as SDO server on the bus.
- The library monitors nodeguarding or the heartbeat consumer time (always only of one producer) and sets corresponding error flags for the application.
- An EDS file can be generated which describes the configured properties of the CANopen slave so that the device can be integrated and configured as a slave under a CANopen master.

The CANopen slave library explicitly does not provide the following functionalities described in CANopen (all options of the CANopen protocol which are not indicated here or in the above section are not implemented either):

- Dynamic SDO and PDO identifiers
- SDO block transfer
- Automatic generation of emergency messages. Emergency messages must always be generated by the application using CANx_SLAVE_EMCY_HANDLER (→ page 169) and CANx_SLAVE_SEND_EMERGENCY (→ page 171). To do so, the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` provides these FBs.
- Dynamic changes of the PDO properties are currently only accepted on arrival of a StartNode NMT message, not with the mechanisms defined in CANopen.

CANopen slave configuration

Contents

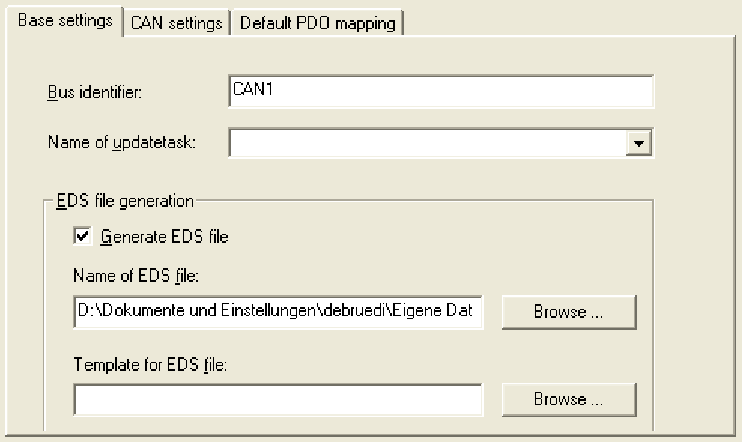
Tab [Base settings] 143
Tab [CAN settings] 145
Tab [Default PDO mapping] 146
Changing the standard mapping by the master configuration 148

1980

To use the controller as CANopen slave the CANopen slave must first be added via [Insert] > [Append subelement]. For controllers with 2 or more CAN interfaces the CAN interface 1 is automatically configured as a slave. All required libraries are automatically added to the library manager.

Tab [Base settings]

1981



Bus identifier: CAN1

Name of updatetask:

☑ Generate EDS file

Name of EDS file: D:\Dokumente und Einstellungen\debruedi\Eigene Dat

Template for EDS file:

Base settings: Bus identifier

10049

Parameter is currently not used.

Base settings: Name of updatetask

10050

Name of the task where the CANopen slave is called.

Base settings: Generate EDS file

10051

If an EDS file is to be generated from the settings to be able to add the CANopen slave to any master configuration, the option [Generate EDS file] must be activated and the name of a file must be indicated. As an option a template file can be indicated whose entries are added to the EDS file of the CANopen slave. In case of overlapping the template definitions are not overwritten.

Example of an object directory

1991

The following entries could for example be in the object directory:

```
[FileInfo]
FileName=D:\CoDeSys\lib2\plcconf\MyTest.eds
FileVersion=1
FileRevision=1
Description=EDS for CoDeSys-Project:
D:\CoDeSys\CANopenTestprojekte\TestHeartbeatODsettings_Device.pro
CreationTime=13:59
CreationDate=09-07-2005
CreatedBy=CoDeSys
ModificationTime=13:59
ModificationDate=09-07-2005
ModifiedBy=CoDeSys

[DeviceInfo]
VendorName=3S Smart Software Solutions GmbH
ProductName=TestHeartbeatODsettings_Device
ProductNumber=0x33535F44
ProductVersion=1
ProductRevision=1
OrderCode=xxxx.yyyy.zzzz
LMT_ManufacturerName=3S GmbH
LMT_ProductName=3S_Dev
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_100=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1
SimpleBootUpMaster=1
SimpleBootUpSlave=0
ExtendedBootUpMaster=1
ExtendedBootUpSlave=0

...

[1018sub0]
ParameterName=Number of entries
ObjectType=0x7
DataType=0x5
AccessType=ro
DefaultValue=2
PDOMapping=0

[1018sub1]
ParameterName=VendorID
ObjectType=0x7
DataType=0x7
AccessType=ro
DefaultValue=0x0
PDOMapping=0

[1018sub2]
ParameterName=Product Code
ObjectType=0x7
DataType=0x7
AccessType=ro
DefaultValue=0x0
PDOMapping=0
```

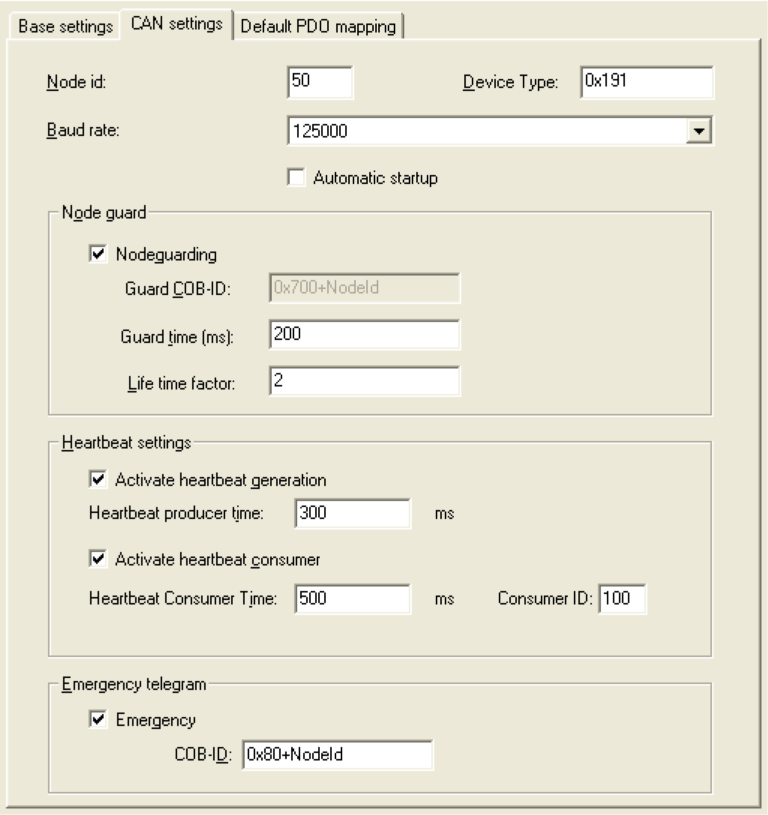
For the meaning of the individual objects please see the CANopen specification DS301.

In addition to the prescribed entries, the EDS file contains the definitions for SYNC, guarding, emergency and heartbeat. If these objects are not used, the values are set to 0 (preset). But as the objects are present in the object directory of the slave at runtime, they are written to in the EDS file.

The same goes for the entries for the communication and mapping parameters. All 8 possible sub-indices of the mapping objects $16xx_{16}$ or $1Axx_{16}$ are present, but possibly not considered in the sub-index 0.
NOTE: Bit mapping is not supported by the library!

Tab [CAN settings]

1982



Node id: 50
Device Type: 0x191
Baud rate: 125000
☐ Automatic startup
☑ Nodeguarding
Guard COB-ID: 0x700+NodeId
Guard time (ms): 200
Life time factor: 2
☑ Activate heartbeat generation
Heartbeat producer time: 300 ms
☑ Activate heartbeat consumer
Heartbeat Consumer Time: 500 ms Consumer ID: 100
☑ Emergency
COB-ID: 0x80+NodeId

Here you can set the **node ID** and the **baud rate**.

Device type
(this is the default value of the object 1000_{16} entered in the EDS) has 191_{16} as default value (standard IO device) and can be freely changed.
The index of the CAN controller results from the position of the CANopen slave in the controller configuration.

The **nodeguarding** parameters, the **heartbeat** parameters and the emergency COB ID can also be defined in this tab. The CANopen slave can only be configured for the monitoring of a heartbeat.
We recommend: It is better to work with the heartbeat function for current devices since then the bus load is lower.

NOTE

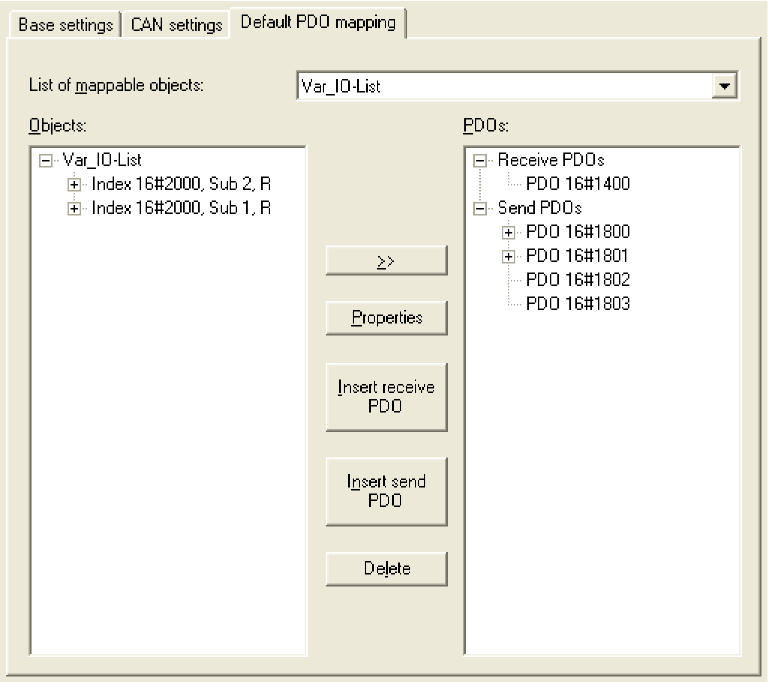
When applying guarding or heartbeat AND
when creating an EDS file for integration with a CANopen master:

► enter guard time = 0
enter life time factor = 0
enter heartbeat time = 0

> The values set for the CANopen master are transmitted to the CANopen slave during configuration. Thus, the CANopen master has safely activated the guarding or heartbeat for this node.

Tab [Default PDO mapping]

1983



List of mappable objects: Var_IO-List

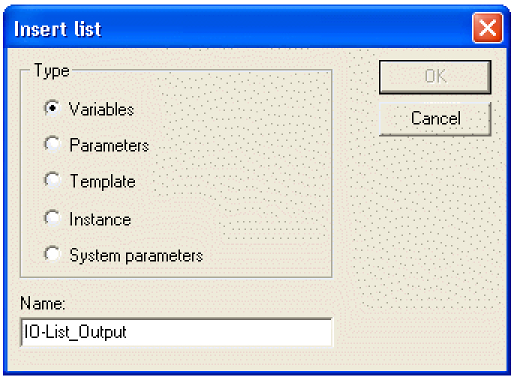
In this tab the assignment between local object directory (OD editor) and PDOs transmitted/received by the CANopen slave can be defined. Such an assignment is called "mapping".

In the object directory entries used (variable OD) the connection to variables of the application is made between object index/sub-index. You only have to ensure that the sub-index 0 of an index containing more than one sub-index contains the information concerning the number of the sub-indices.

Example: list of variables

10052

On the first receive PDO (COB ID = 512 + node ID) of the CANopen slave the data für variable PLC_PRG.a shall be received.



Type
● Variables
○ Parameters
○ Template
○ Instance
○ System parameters

Name: IO-List_Output

Info

[Variables] and [parameters] can be selected as list type.

For the exchange of data (e.g. via PDOs or other entries in the object directory) a variable list is created.

The parameter list should be used if you do not want to link object directory entries to application variables. For the parameter list only the index 1006_{16} / SubIdx 0 is currently predefined. In this entry the value for the "Com. Cycle Period" can be entered by the master. This signals the absence of the SYNC message.

So you have to create a variable list in the object directory (parameter manager) and link an index/sub-index to the variable `PLC_PRG.a`.

- ► To do so, add a line to the variable list (a click on the right mouse button opens the context menu) and enter a variable name (any name) as well as the index and sub-index.
- ► The only allowed access right for a receive PDO is [write only].
- ► Enter "`PLC_PRG.a`" in the column [variable] or press [F2] and select the variable.

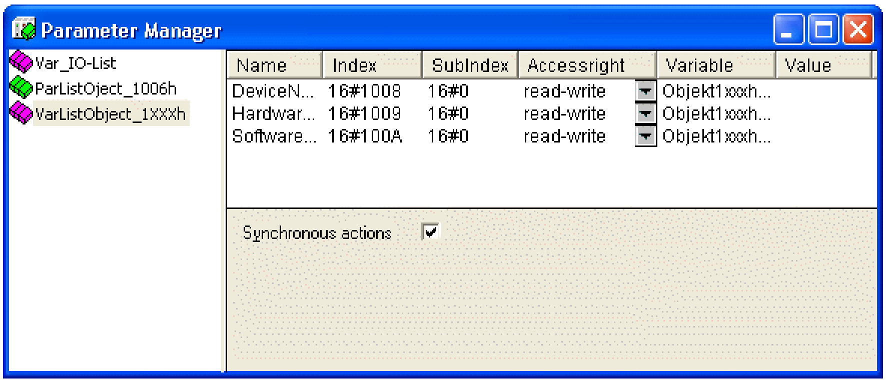
NOTE

Data to be read by the CANopen master (e.g. inputs, system variables) must have the access right [read only].

Data to be written by the CANopen master (e.g. outputs in the slave) must have the access right [write only].

SDO parameters to be written and at the same time to be read from and written to the slave application by the CANopen master must have the access right [read-write].

To be able to open the parameter manager the parameter manager must be activated in the target settings under [Network functionality]. The areas for index/sub-index already contain sensible values and should not be changed.



Name	Index	SubIndex	Accessright	Variable	Value
DeviceN...	16#1008	16#0	read-write	Objekt1xxxh...	
Hardwar...	16#1009	16#0	read-write	Objekt1xxxh...	
Software...	16#100A	16#0	read-write	Objekt1xxxh...	

Synchronous actions ☑

In the default PDO mapping of the CANopen slave the index/sub-index entry is then assigned to a receive PDO as mapping entry. The PDO properties can be defined via the dialogue known from chapter Add and configure CANopen slaves (→ page 127).

Only objects from the parameter manager with the attributes [read only] or [write only] are marked in the possibly generated EDS file as mappable (= can be assigned) and occur in the list of the mappable objects. All other objects are not marked as mappable in the EDS file.

NOTE

If more than 8 data bytes are mapped to a PDO, the next free identifiers are then automatically used until all data bytes can be transferred.

To obtain a clear structure of the identifiers used you should add the correct number of the receive and transmit PDOs and assign them the variable bytes from the list.

Changing the standard mapping by the master configuration

1984

You can change the default PDO mapping (in the CANopen slave configuration) within certain limits by the master.

The rule applies that the CANopen slave cannot recreate entries in the object directory which are not yet available in the standard mapping (default PDO mapping in the CANopen slave configuration). For a PDO, for example, which contains a mapped object in the default PDO mapping no second object can be mapped in the master configuration.

So the mapping changed by the master configuration can at most contain the PDOs available in the standard mapping. Within these PDOs there are 8 mapping entries (sub-indices).

Possible errors which may occur are not displayed, i.e. the supernumerary PDO definitions / supernumerary mapping entries are processed as if not present.

In the master the PDOs must always be created starting from 1400_{16} (receive PDO communication parameter) or 1800_{16} (transmit PDO communication parameter) and follow each other without interruption.

Access to the CANopen slave at runtime

1985

Setting of the node numbers and the baud rate of a CANopen slave

1986

For the CANopen slave the node number and the baud rate can be set at runtime of the application program.

- ► For setting the **node number** CANx_SLAVE_NODEID (→ page 168) of the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` is used.
- ► For setting the **baud rate** CAN1_BAUDRATE (→ page 75) or CAN1_EXT (→ page 79) or CANx of the corresponding device library is used for the controllers and the PDM360smart. For PDM360 or PDM360compact CANx_SLAVE_BAUDRATE is available via the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib`.

Access to the OD entries by the application program

1987

As standard, there are entries in the object directory which are mapped to variables (parameter manager).

However, there are also automatically generated entries of the CANopen slave which cannot be mapped to the contents of a variable via the parameter manager. Via CANx_SLAVE_STATUS (→ page 174) these entries are available in the library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`.

Change the PDO properties at runtime

1988

If the properties of a PDO are to be changed at runtime, this is done by another node via SDO write access as described by CANopen.

As an alternative, it is possible to directly write a new property, e.g. the "event time" of a send PDO and then transmit a command "StartNode-NMT" to the node although it has already been started. As a result of this the device reinterprets the values in the object directory.

Transmit emergency messages via the application program

1989

To transmit an emergency message via the application program CANx_SLAVE_EMCY_HANDLER (→ page 169) and CANx_SLAVE_SEND_EMERGENCY (→ page 171) can be used. The library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` provides these functions.

CANopen network variables

Contents

General information 150
Configuration of CANopen network variables 151
Particularities for network variables 155

1868

General information

2076

Network variables
Network variables are one option to exchange data between two or several controllers. For users the mechanism should be easy to use. At present network variables are implemented on the basis of CAN and UDP. The variable values are automatically exchanged on the basis of broadcast messages. In UDP they are implemented as broadcast messages, in CAN as PDOs. These services are not confirmed by the protocol, i.e. it is not checked whether the receiver receives the message. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

Object directory
The object directory is another option to exchange variables. This is a 1 to 1 connection using a confirmed protocol. The user can check whether the message arrived at the receiver. The exchange is not carried out automatically but via the call of FBs from the application program.
→ chapter The object directory of the CANopen master (→ page 140)

Configuration of CANopen network variables

Contents

Settings in the target settings 151
Settings in the global variable lists 152

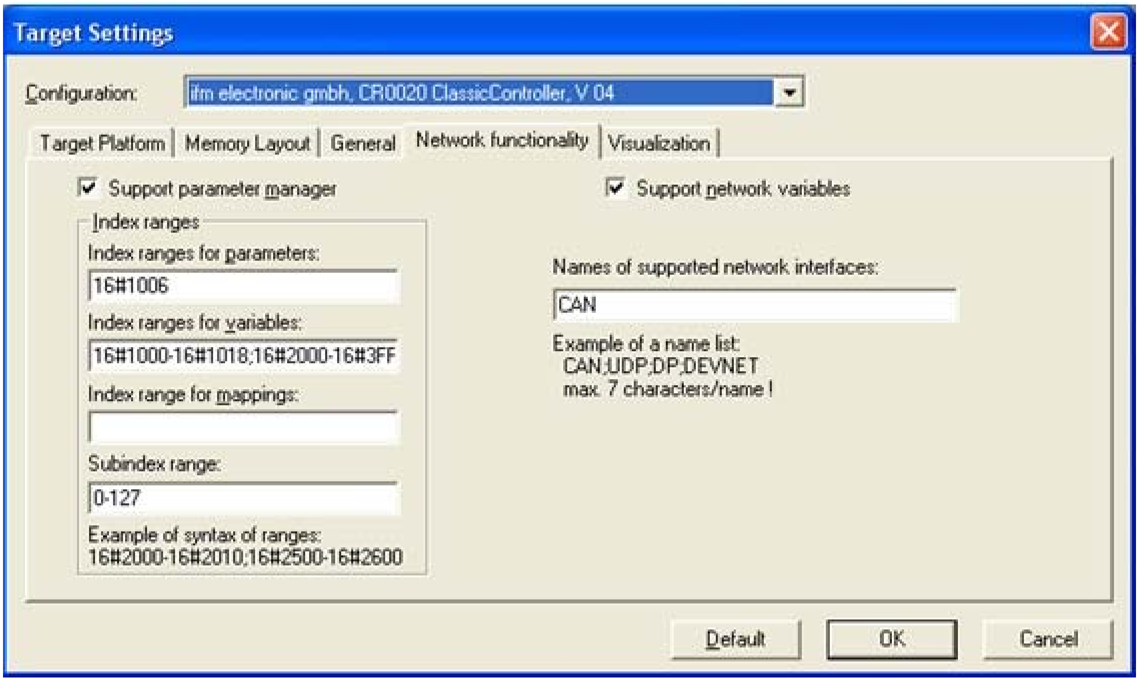
1869

To use the network variables with CoDeSys you need the following libraries:
- `3s_CanDrv.lib`
- `3S_CANopenManager.lib`
- `3S_CANopenNetVar.lib`
- `SysLibCallback.lib`.

CoDeSys automatically generates the required initialisation code and the call of the network blocks at the start and end of the cycle.

Settings in the target settings

1994



Target Settings

Configuration: ifm electronic gmbh, CR0020 ClassicController, V 04

Target Platform | Memory Layout | General | Network functionality | Visualization

☑ Support parameter manager

Index ranges

Index ranges for parameters: 16#1006

Index ranges for variables: 16#1000-16#1018;16#2000-16#3FF

Index range for mappings:

Subindex range: 0-127

Example of syntax of ranges:
16#2000-16#2010;16#2500-16#2600

☑ Support network variables

Names of supported network interfaces: CAN

Example of a name list:
CAN;UDP;DP;DEVNET
max. 7 characters/name !

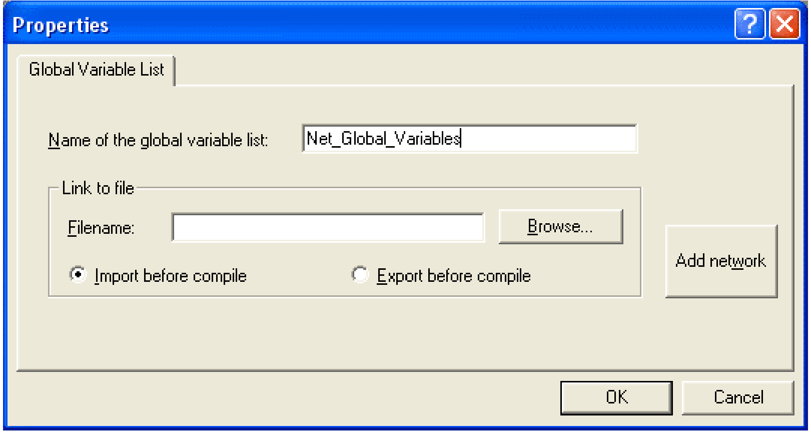
- ► Select the dialogue box [Target settings].
- ► Select the tab [Network functionality].
- ► Activate the check box [Support network variables].
- ► Enter the name of the requested network, here CAN, in [Names of supported network interfaces].
- ► To use network variables you must also add a CANopen master or CANopen slave (device) to the controller configuration.
- ► Please note the particularities when using network variables for the corresponding device types.
  → Chapter Particularities for network variables (→ page 155)

Settings in the global variable lists

1995

- ► Create a new global variable list. In this list the variables to be exchanged with other controllers are defined.
- ► Open the dialogue with the menu point [Object Properties].

> The window [Properties] appears:



Name of the global variable list: Net_Global_Variables

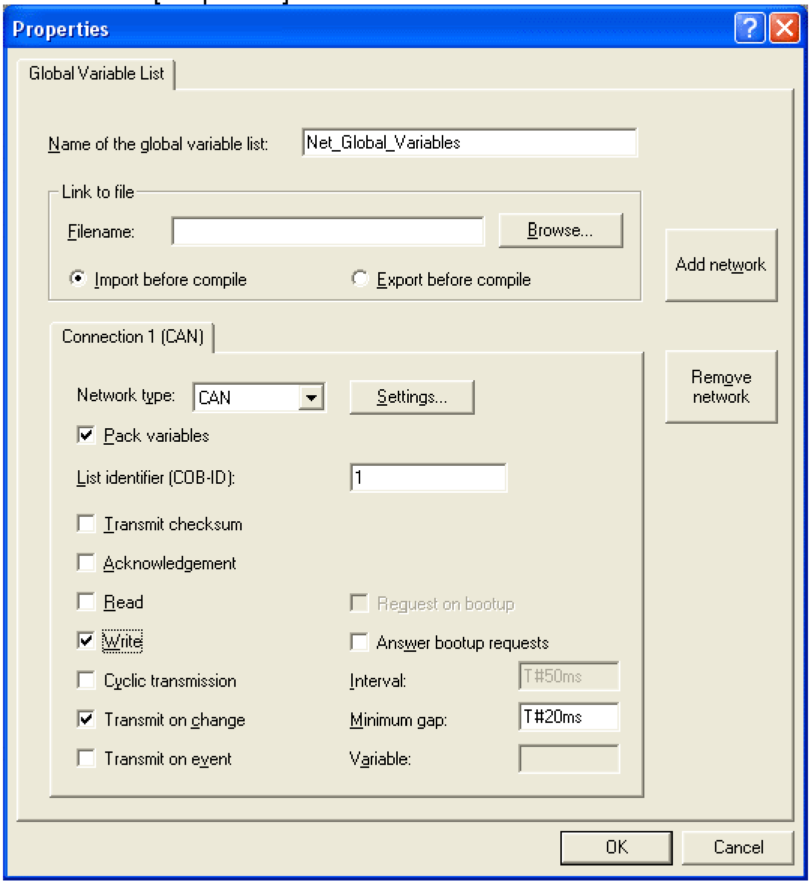
Filename:

● Import before compile ○ Export before compile

If you want to define the network properties:

- ► Click the button [Add network].
If you have configured several network connections, you can also configure here several connections per variable list.

> The window [Properties] extends as follows:



Name of the global variable list: Net_Global_Variables

Filename:

● Import before compile ○ Export before compile

Network type: CAN

☑ Pack variables

List identifier (COB-ID): 1

☐ Transmit checksum

☐ Acknowledgement

☐ Read ☐ Request on bootup

☑ Write ☐ Answer bootup requests

☐ Cyclic transmission Interval: T#50ms

☑ Transmit on change Minimum gap: T#20ms

☐ Transmit on event Variable:

Meaning of the options:

Global variable list: Network type

10055

As network type you can enter one of the network names indicated in the target settings.
If you click on the button [Settings] next to it, you can select the CAN interface:

1. CAN interface: value = 0
2. CAN interface: value = 1
etc.

Global variable list: Pack variables

10056

If this option is activated with [**v**], the variables are combined, if possible, in one transmisson unit. For CAN the size of a transmission unit is 8 bytes.

If it is not possible to include all variables of the list in one transmission unit, several transmission units are formed for this list.

If the option is not activated, every variable has its own transmission unit.

If [Transmit on change] is configured, it is checked separately for every transmission unit whether it has been changed and must be transmitted.

Global variable list: List identifier (COB-ID)

10057

The basic identifier is used as a unique identification to exchange variable lists of different projects. Variable lists with identical basic identifier are exchanged. Ensure that the definitions of the variable lists with the same basic identifier match in the different projects.

NOTE

In CAN networks the basic identifier is directly used as COB-ID of the CAN messages. It is not checked whether the identifier is also used in the remaining CAN configuration.

To ensure a correct exchange of data between two controllers the global variable lists in the two projects must match. To ensure this you can use the feature [Link to file]. A project can export the variable list file before compilation, the other projects should import this file before compilation.

In addition to simple data types a variable list can also contain structures and arrays. The elements of these combined data types are transmitted separately.

Strings must not be transmitted via network variables as otherwise a runtime error will occur and the watchdog will be activated.

If a variable list is larger than a PDO of the corresponding network, the data is split up to several PDOs. Therefore it cannot be ensured that all data of the variable list is received in **one** cycle. Parts of the variable list can be received in different cycles. This is also possible for variables with structure and array types.

Global variable list: Transmit checksum

10058

This option is not supported.

Global variable list: Acknowledgement

10059

This option is not supported.

Global variable list: Read

10060

The variable values of one (or several) controllers are read.

Global variable list: Write

10061

The variables of this list are transmitted to other controllers.

! NOTE

You should only select one of these options for every variable list, i.e. either only read or only write.

If you want to read or write several variables of a project, please use several variable lists (one for reading, one for writing).

To get the same data structure for the communication between two participants you should copy the variable list from one controller to the other.

In a network the same variable list should only be exchanged between two participants.

Global variable list: Cyclic transmission

10062

Only valid if [write] is activated. The values are transmitted in the specified [interval] irrespective of whether they have changed.

Global variable list: Transmit on change

10063

The variable values are only transmitted if one of the values has been changed. With [Minimum gap] (value > 0) a minimum time between the message packages can be defined.

Global variable list: Transmit on event

10064

If this option is selected, the CAN message is only transmitted if the indicated binary [variable] is set to TRUE. This variable cannot be selected from the list of the defined variables via the input help.

Particularities for network variables

1992

Device	Description
ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506	Network variables are only supported on interface 1 (enter the value 0). **CANopen master** Transmit and receive lists are processed directly. You only have to make the settings described above. **CANopen slave** Transmit lists are processed directly. For receive lists you must also map the identifier area in the object directory to receive PDOs. It is sufficient to create only two receive PDOs and to assign the first object the first identifier and the second object the last identifier. If the network variables are only transferred to one identifier, you only have to create one receive PDO with this identifier. **Important!** Please note that the identifier of the network variables and of the receive PDOs must be entered as **decimal** value.
ClassicController: CR0032 ExtendedController: CR0232	Network variables are supported on all CAN interfaces. (All other informations as above)
BasicController: CR040n	Network variables are supported on all CAN interfaces. (All other informations as above)
BasicDisplay: CR0451	Only one CAN interface is available (enter value = 0). (All other informations as above)
PDM360smart: CR1070, CR1071	Only one CAN interface is available (enter value = 0). **CANopen master** Transmit and receive lists are processed directly. You only have to make the settings described above. **CANopen slave** Transmit lists are processed directly. For receive lists you must additionally map the identifier area in the object directory to receive PDOs. It is sufficient to create only two receive PDOs and to assign the first object the first identifier and the second object the last identifier. If the network variables are only transferred to one identifier, you only have to create one receive PDO with this identifier. **Important!** Please note that the identifier of the network variables and of the receive PDOs must be entered as **decimal** value.
PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056	Network variables are supported on interface 1 (value = 0) and 2 (value = 1). **CANopen master** Transmit and receive lists are processed directly. You only have to make the settings described above. **CANopen slave** Transmit and receive lists are processed directly. You only have to make the settings described above. **Important!** If [support network variables] is selected in the PDM360 or PDM360compact, you must at least create one variable in the global variable list and call it once in the application program. Otherwise the following error message is generated when compiling the program: Error 4601: Network variables 'CAN': No cyclic or freewheeling task for network variable exchange found.
PDM360NG: CR108n, CR9042	Network variables are supported on all CAN interfaces. (All other informations as above)

8.6.2 Libraries for CANopen

Contents

ifm library for the CANopen master 156
ifm library for the CANopen slave 167
Further ifm libraries for CANopen 177

8587

ifm library for the CANopen master

Contents

CANx_MASTER_EMCY_HANDLER 157
CANx_MASTER_SEND_EMERGENCY........ 159
CANx_MASTER_STATUS........ 162

1870

The library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` provides a number of FBs for the CANopen master which will be explained below.

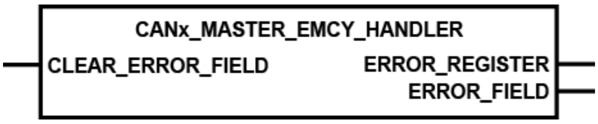
CANx_MASTER_EMCY_HANDLER

2006

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_MASTER_EMCY_HANDLER

9411

Contained in the library: `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Description

2009

CANx_MASTER_EMCY_HANDLER monitors the device-specific error status of the master. The FB must be called in the following cases:

- the error status is to be transmitted to the network and
- the error messages of the application are to be stored in the object directory.

NOTE

If application-specific error messages are to be stored in the object directory, CANx_MASTER_EMCY_HANDLER must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY (→ page 159).

Parameters of the inputs

2010

Parameter	Data type	Description
CLEAR_ERROR_FIELD	BOOL	TRUE: deletes the contents of the array ERROR_FIELD FALSE: this function is not executed

Parameters of the outputs

2011

Parameter	Data type	Description
ERROR_REGISTER	BYTE	shows the content of the object directory index 1001_{16} (Error Register)
ERROR_FIELD	ARRAY [0...5] OF WORD	the array [0...5] shows the contents of the object directory index 1003_{16} (Error Field) - ERROR_FIELD[0]: number of stored errors - ERROR_FIELD[1...5]: stored errors, the most recent error is in index [1]

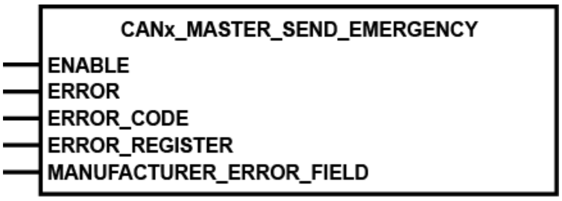
CANx_MASTER_SEND_EMERGENCY

2012

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_MASTER_SEND_EMERGENCY

9430

Contained in the library: `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Description

2015

CANx_MASTER_SEND_EMERGENCY transmits application-specific error states. The FB is called if the error status is to be transmitted to other devices in the network.

NOTE

If application-specific error messages are to be stored in the object directory, CANx_MASTER_EMCY_HANDLER (→ page 157) must be called **after** (repeatedly) calling CANx_MASTER_SEND_EMERGENCY.

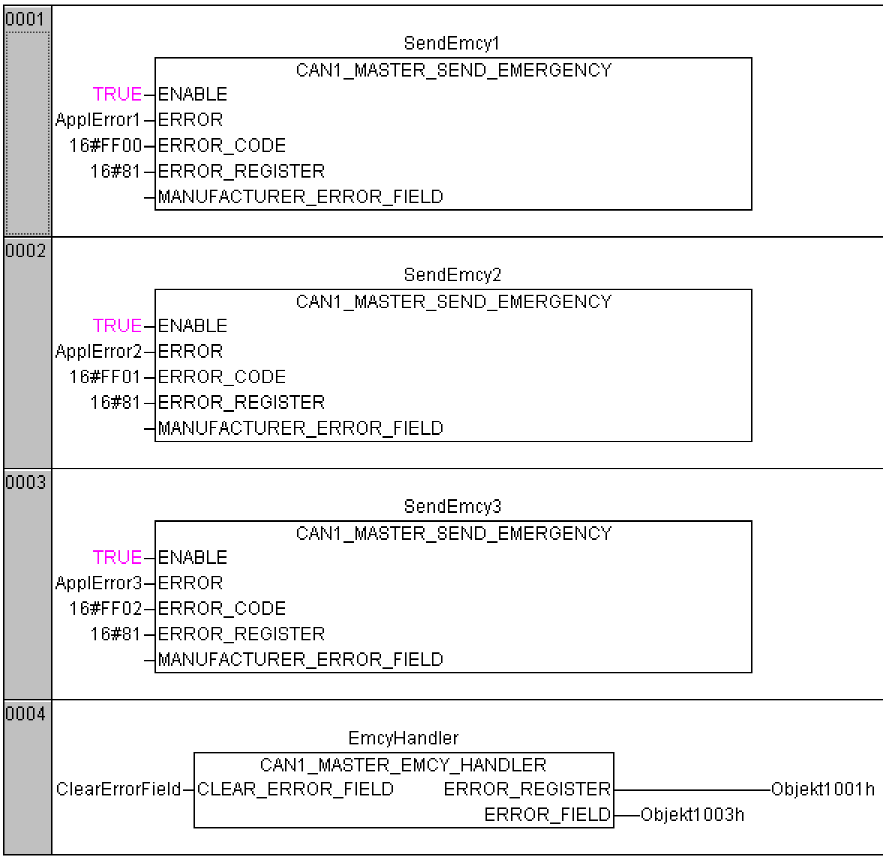
Parameters of the inputs

2016

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
ERROR	BOOL	FALSE → TRUE (edge): transmits the given error code TRUE → FALSE (edge) AND the fault is no longer indicated: the message that there is no error is sent after a delay of approx. 1 s else: this function is not executed
ERROR_CODE	WORD	The error code provides detailed information about the detected fault. The values should be entered according to the CANopen specification. → chapter Overview CANopen error codes (→ page 187)
ERROR_REGISTER	BYTE	This object reflects the general error state of the CANopen network participant. The values should be entered according to the CANopen specification.
MANUFACTURER_ERROR_FIELD	ARRAY [0...4] OF BYTE	Here, up to 5 bytes of application-specific error information can be entered. The format can be freely selected.

Example: CANx_MASTER_SEND_EMERGENCY

2018



In this example 3 error messages will be generated subsequently:

1. ApplError1, Code = FF00$_{16}$ in the error register 81$_{16}$
2. ApplError2, Code = FF01$_{16}$ in the error register 81$_{16}$
3. ApplError3, Code = FF02$_{16}$ in the error register 81$_{16}$

CAN1_MASTER_EMCY_HANDLER sends the error messages to the error register "Object 1001$_{16}$" in the error array "Object 1003$_{16}$".

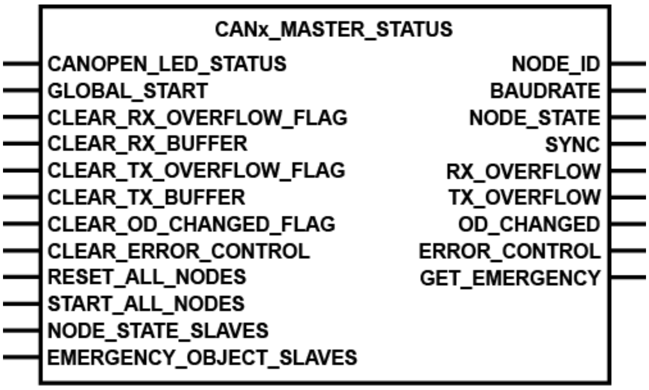
CANx_MASTER_STATUS

2021

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_MASTER_STATUS

9441

Contained in the library: `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Description

2024

Status indication of the device used with CANopen.

CANx_MASTER_STATUS shows the status of the device used as CANopen master. Furthermore, the status of the network and of the connected slaves can be monitored.

The FB simplifies the use of the CoDeSys CANopen master libraries. We urgently recommend to carry out the evaluation of the network status and of the error messages via this FB.

Parameters of the inputs

2025

Parameter	Data type	Description
CANOPEN_LED_STATUS	BOOL	(input not available for PDM devices) TRUE: the status LED of the controller is switched to the mode "CANopen": flashing frequency 0.5 Hz = PRE-OPERATIONAL flashing frequency 2.0 Hz = OPERATIONAL The other diagnostic LED signals are not changed by this operating mode.

Parameter	Data type	Description
GLOBAL_START	BOOL	TRUE: all connected network participants (slaves) are started simultaneously during network initialisation FALSE: the connected network participants are started one after the other further information → chapter Starting the network with GLOBAL_START (→ page 138)
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "receive buffer overflow" FALSE: this function is not executed
CLEAR_RX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the receive buffer FALSE: this function is not executed
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "transmit buffer overflow" FALSE: this function is not executed
CLEAR_TX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the transmit buffer FALSE: this function is not executed
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE → TRUE (edge): delete flag "data in the object directory changed" FALSE: this function is not executed
CLEAR_ERROR_CONTROL	BOOL	FALSE → TRUE (edge): delete the guard error list (ERROR_CONTROL) FALSE: this function is not executed
RESET_ALL_NODES	BOOL	FALSE → TRUE (edge): reset all nodes FALSE: this function is not executed
START_ALL_NODES	BOOL	TRUE: All connected network participants (slaves) are started simultaneously at runtime of the application program. FALSE: The connected network participants must be started one after the other further information → chapter Starting the network with START_ALL_NODES (→ page 138)
NODE_STATE_SLAVES	DWORD	shows the status of all network nodes ► The address must be determined by means of the operator ADR and assigned to the FB! example code → chapter Example: CANx_MASTER_STATUS (→ page 165) further information → chapter Master at runtime (→ page 131)
EMERGENCY_OBJECT_SLAVES	DWORD	shows the most recent occurred error messages of all network nodes ► The address must be determined by means of the operator ADR and assigned to the FB! further information → chapter Access to the structures at runtime of the application (→ page 166)

Parameters of the outputs

2029

Parameter	Data type	Description
NODE_ID	BYTE	node ID of the master
BAUDRATE	WORD	baud rate of the master
NODE_STATE	INT	current status of the master
SYNC	BOOL	SYNC signal of the master This is set in the tab [CAN parameters] (→ page 124) of the master depending on the set time [Com. Cycle Period].
RX_OVERFLOW	BOOL	error flag "receive buffer overflow"
TX_OVERFLOW	BOOL	error flag "transmit buffer overflow"
OD_CHANGED	BOOL	flag "object directory master was changed"
ERROR_CONTROL	ARRAY [0...7] OF BYTE	The array contains a list (max. 8) of the missing network nodes (guard or heartbeat error). further information → chapter Access to the structures at runtime (→ page 166)
GET_EMERGENCY	STRUCT EMERGENY_MESSAGE	at the output the data for the structure EMERGENCY_MESSAGE are available the most recent error message of a network node is always displayed To obtain a list of all occurred errors, the array "EMERGENCY_OBJECT_SLAVES" must be evaluated.

Parameters of internal structures

2030

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

Parameter	Data type	Description
CANx_EMERGENY_MESSAGE	STRUCT	NODE_ID: BYTE ERROR_CODE: WORD ERROR_REGISTER: BYTE MANUFACTURER_ERROR_FIELD: ARRAY[0...4] OF BYTE The structure is defined by the global variables of the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`.
CANx_NODE_STATE	STRUCT	NODE_ID: BYTE NODE_STATE: BYTE LAST_STATE: BYTE RESET_NODE: BOOL START_NODE: BOOL PREOP_NODE: BOOL SET_TIMEOUT_STATE: BOOL SET_NODE_STATE: BOOL The structure is defined by the global variables of the library `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`.

Detailed description of the functionalities of the CANopen master and the mechanisms → chapter CANopen master (→ page 121).

Using the controller CR0020 as an example the following code fragments show the use of CANx_MASTER_STATUS (→ page 162).

Example: CANx_MASTER_STATUS

2031

Slave information

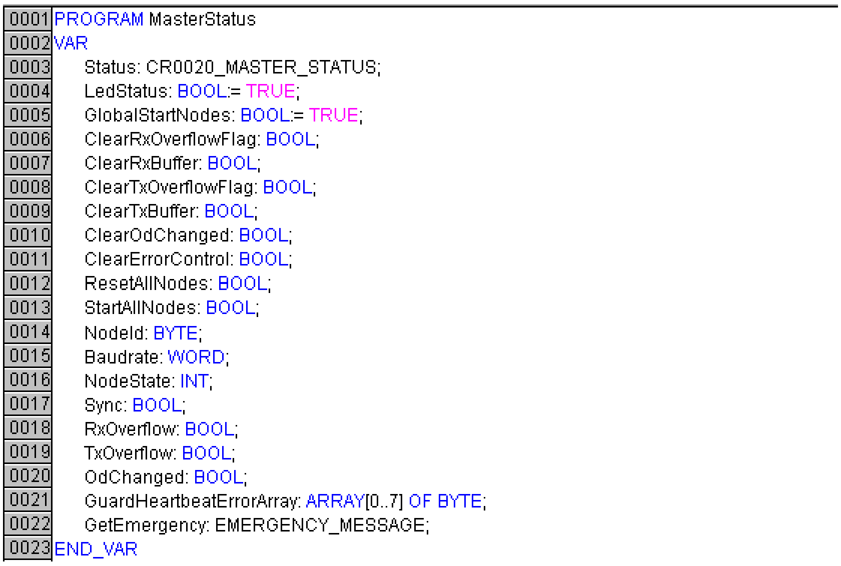
2033

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

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

! NOTE

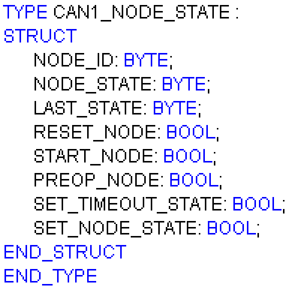
The numbers of the array elements do **not** correspond to the node ID. The identifier can be read from the corresponding structure under NODE_ID.



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

Structure node status

2034



```
TYPE CAN1_NODE_STATE :
STRUCT
    NODE_ID: BYTE;
    NODE_STATE: BYTE;
    LAST_STATE: BYTE;
    RESET_NODE: BOOL;
    START_NODE: BOOL;
    PREOP_NODE: BOOL;
    SET_TIMEOUT_STATE: BOOL;
    SET_NODE_STATE: BOOL;
END_STRUCT
END_TYPE
```

Structure Emergency_Message

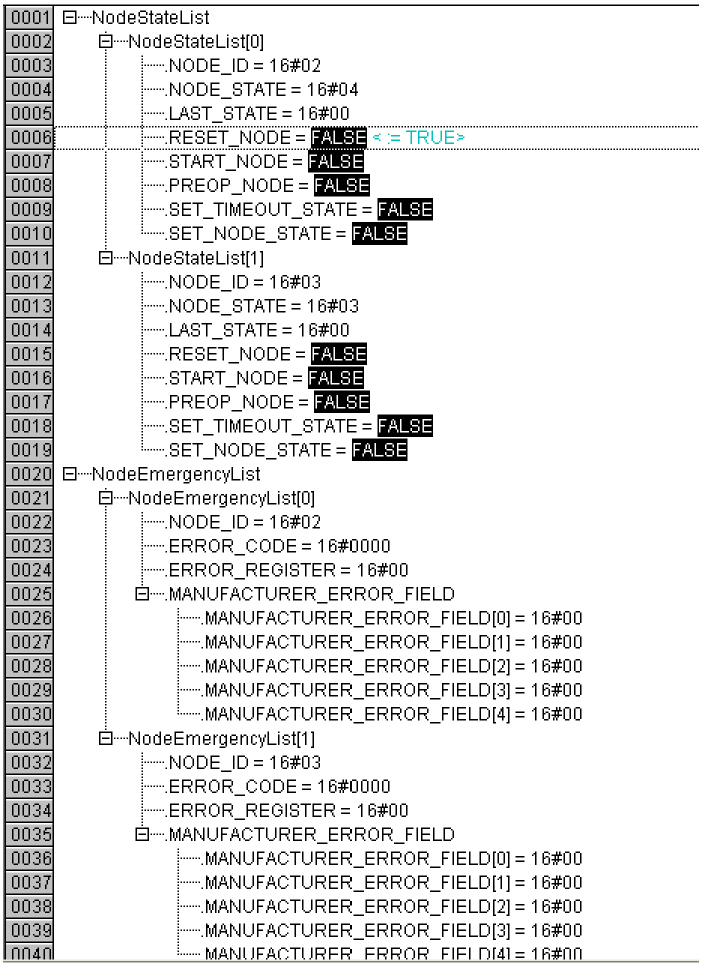
2035

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

Access to the structures at runtime of the application

2036

At runtime you can access the corresponding array element via the global variables of the library and therefore read the status or EMCY messages or reset the node.



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

If ResetSingleNodeArray[0].RESET_NODE is set to TRUE for a short time in the example given above, the first node is reset in the configuration tree.

Further information concerning the possible error codes → chapter CAN errors and error handling (→ page 182).

ifm library for the CANopen slave

Contents

CANx_SLAVE_NODEID 168
CANx_SLAVE_EMCY_HANDLER........ 169
CANx_SLAVE_SEND_EMERGENCY........ 171
CANx_SLAVE_STATUS 174

1874

The library `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` provides a number of FBs for the CANopen slave which will be explained below.

CANx_SLAVE_NODEID

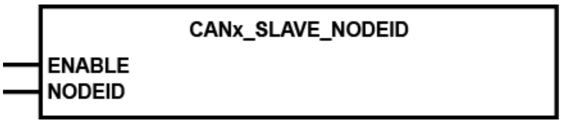
2044

= CANx Slave Node-ID

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_SLAVE_NODEID

9499

Contained in the library: `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Description

2049

CANx_SLAVE_NODEID enables the setting of the node ID of a CANopen slave at runtime of the application program.

Normally, the FB is called once during initialisation of the controller, in the first cycle. Afterwards, the input ENABLE is set to FALSE again.

Parameters of the inputs

2047

Parameter	Data type	Description
ENABLE	BOOL	FALSE → TRUE (edge): set node ID FALSE: unit is not executed
NODEID	BYTE	value of the new node number

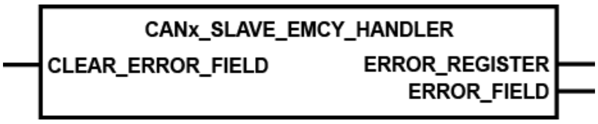
CANx_SLAVE_EMCY_HANDLER

2050

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_SLAVE_EMCY_HANDLER

9493

Contained in the library: `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Description

2053

CANx_SLAVE_EMCY_HANDLER monitors the device-specific error status (device operated as slave).

The FB must be called in the following cases:

- the error status is to be transmitted to the CAN network and
- the error messages of the application are to be stored in the object directory.

NOTE

If application-specific error messages are to be stored in the object directory, CANx_SLAVE_EMCY_HANDLER must be called **after** (repeatedly) calling CANx_SLAVE_SEND_EMERGENCY (→ page 171).

Parameters of the inputs

2054

Parameter	Data type	Description
CLEAR_ERROR_FIELD	BOOL	FALSE → TRUE (edge): delete ERROR FIELD FALSE: unit is not executed

Parameters of the outputs

2055

Parameter	Data type	Description
ERROR_REGISTER	BYTE	shows the contents of the object directory index 1001_{16} (Error Register).
ERROR_FIELD	ARRAY [0...5] OF WORD	the array [0...5] shows the contents of the object directory index 1003_{16} (Error Field): - ERROR_FIELD[0]: Number of stored errors - ERROR_FIELD[1...5]: stored errors, the most recent error is in index [1]

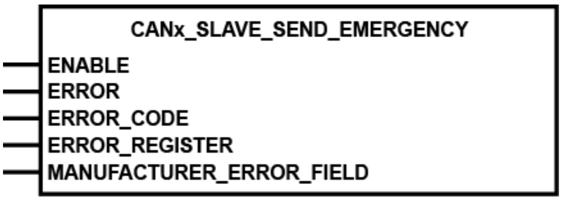
CANx_SLAVE_SEND_EMERGENCY

2056

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_SLAVE_SEND_EMERGENCY

9505

Contained in the library: `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360: CR1050, CR1051, CR1060
- PDM360compact: CR1052, CR1053, CR1055, CR1056
- PDM360smart: CR1070, CR1071

Description

2059

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

The FB is called if the error status is to be transmitted to other devices in the network.

! NOTE

If application-specific error messages are to be stored in the object directory, CANx_SLAVE_EMCY_HANDLER (→ page 169) must be called **after** (repeatedly) calling CANx_SLAVE_SEND_EMERGENCY.

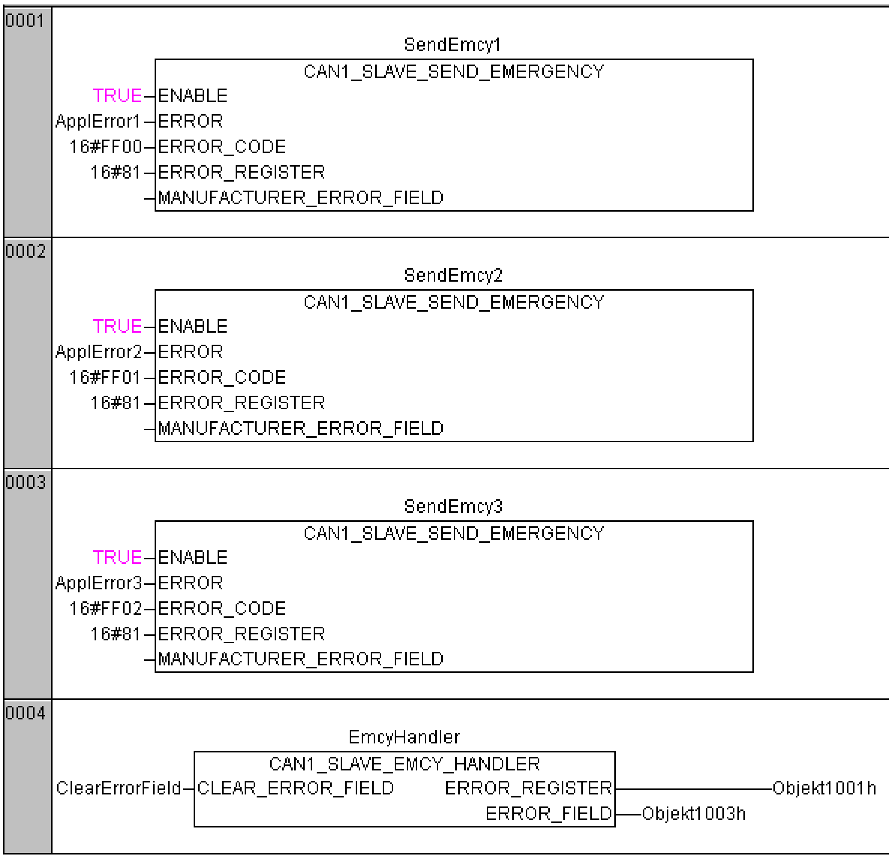
Parameters of the inputs

2060

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
ERROR	BOOL	FALSE → TRUE (edge): transmits the given error code TRUE → FALSE (edge) AND the fault is no longer indicated: the message that there is no error is sent after a delay of approx. 1 s else: this function is not executed
ERROR_CODE	WORD	The error code provides detailed information about the detected fault. The values should be entered according to the CANopen specification. → chapter Overview of the CANopen error codes (→ page 187)
ERROR_REGISTER	BYTE	This object reflects the general error state of the CANopen network participant. The values should be entered according to the CANopen specification.
MANUFACTURER_ERROR_FIELD	ARRAY [0...4] OF BYTE	Here, up to 5 bytes of application-specific error information can be entered. The format can be freely selected.

Example: CANx_SLAVE_SEND_EMERGENCY

2062



In this example 3 error messages will be generated subsequently:

1. ApplError1, Code = FF00$_{16}$ in the error register 81$_{16}$
2. ApplError2, Code = FF01$_{16}$ in the error register 81$_{16}$
3. ApplError3, Code = FF02$_{16}$ in the error register 81$_{16}$

CAN1_SLAVE_EMCY_HANDLER sends the error messages to the error register "Object 1001$_{16}$" in the error array "Object 1003$_{16}$".

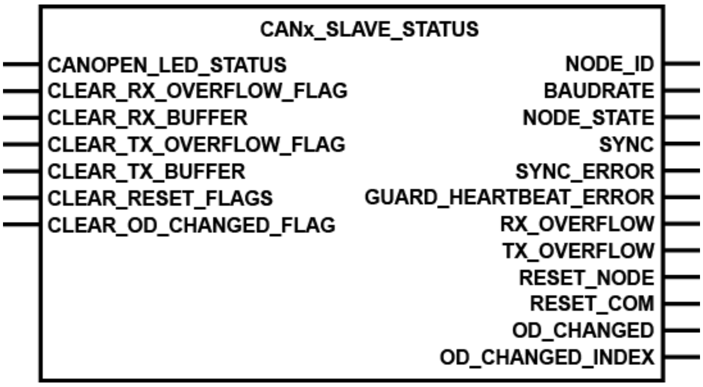
CANx_SLAVE_STATUS

2063

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_SLAVE_STATUS

9516

Contained in the library: `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

Description

2066

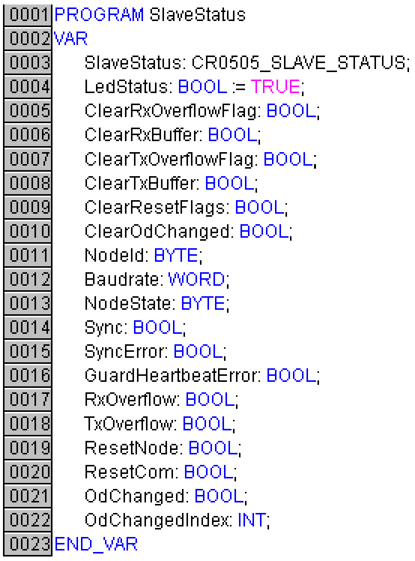
CANx_SLAVE_STATUS shows the status of the device used as CANopen slave. The FB simplifies the use of the CoDeSys CANopen slave libraries. We urgently recommend to carry out the evaluation of the network status via this FB.

Info

For a detailed description of the FBs of the CANopen slave and the mechanisms:
→ chapter CANopen slave (→ page 142).

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

Example:



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

Parameters of the inputs

2067

Parameter	Data type	Description
GLOBAL_START	BOOL	TRUE: all connected network participants (slaves) are started simultaneously during network initialisation FALSE: the connected network participants (slaves) are started one after the other further information → chapter Starting the network with GLOBAL_START (→ page 138)
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "receive buffer overflow" FALSE: this function is not executed
CLEAR_RX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the receive buffer FALSE: this function is not executed
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (edge): delete error flag "transmit buffer overflow" FALSE: this function is not executed
CLEAR_TX_BUFFER	BOOL	FALSE → TRUE (edge): delete data in the transmit buffer FALSE: this function is not executed
CLEAR_RESET_FLAG	BOOL	FALSE → TRUE (edge): delete the flags "nodes reset" and "communications interface reset" FALSE: this function is not executed
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE → TRUE (edge): delete the flags "data in the object directory changed" and "index position" FALSE: this function is not executed

Parameters of the outputs

2068

Parameter	Data type	Description
NODE_ID	BYTE	ode ID of the slave
BAUDRATE	WORD	baud rate of the slave
NODE_STATE	BYTE	current status of the slave
SYNC	BOOL	received SYNC signal of the master
SYNC_ERROR	BOOL	no SYNC signal of the master received OR: the set SYNC time (ComCyclePeriod in the master) was exceeded
GUARD_HEARTBEAT_ERROR	BOOL	no guard or heartbeat signal of the master received OR: the set times were exceeded
RX_OVERFLOW	BOOL	error flag "receive buffer overflow"
TX_OVERFLOW	BOOL	error flag "transmit buffer overflow"
RESET_NODE	BOOL	the CAN stack of the slave was reset by the master This flag can be evaluated by the application and, if necessary, be used for further reactions.
RESET_COM	BOOL	the communication interface of the CAN stack was reset by the master This flag can be evaluated by the application and, if necessary, be used for further reactions.
OD_CHANGED	BOOL	flag "object directory master was changed"
OD_CHANGED_INDEX	INT	the output shows the changed index of the object directory

Further ifm libraries for CANopen

Contents

CANx_SDO_READ 178
CANx_SDO_WRITE.. 180

2071

Here we present further **ifm** FBs which are sensible additions for CANopen.

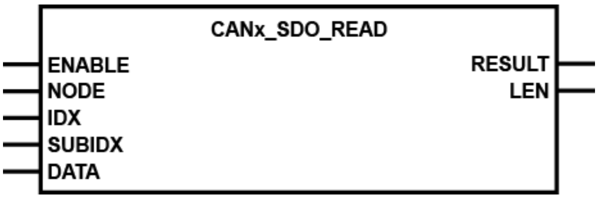
CANx_SDO_READ

621

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_SDO_READ

9442

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Description

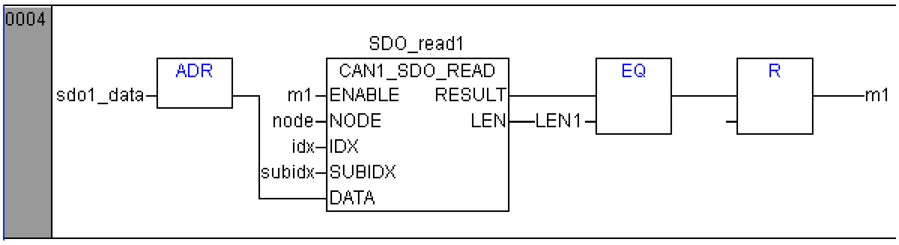
624

CANx_SDO_READ reads the SDO (→ page 130) with the indicated indexes from the node.

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

- all ecomatmobile controllers - PCB controller: CS0015 - PDM360smart: CR1070, CR1071	- PDM360: CR1050, CR1051, CR1060 - PDM360compact: CR1052, CR1053, CR1055, CR1056
From the device library `ifm_CRnnnn_Vxxyyzz.LIB`	From the device library ifm_CANx_SDO_Vxxyyzz.LIB
Prerequisite: Node must be in the mode "PRE-OPERATIONAL" or "OPERATIONAL".	Prerequisite: The node must be in the mode "CANopen master" or "CANopen slave".
For controllers, only CAN1_SDO_READ is available.	

Example:



Parameters of the inputs

625

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
NODE	BYTE	number of the node
IDX	WORD	index in object directory
SUBIDX	BYTE	sub-index referred to the index in the object directory
DATA	DWORD	address of the receive data array permissible length = 0...255 transmission with ADR operator

Parameters of the outputs

626

Parameter	Data type	Description
RESULT	BYTE	0 = unit inactive 1 = execution of the unit completed 2 = unit active 3 = error: unit has not been executed
LEN	WORD	length of the entry in "number of bytes" The value for LEN must correspond to the length of the receive array. Otherwise, problems with SDO communication will occur.

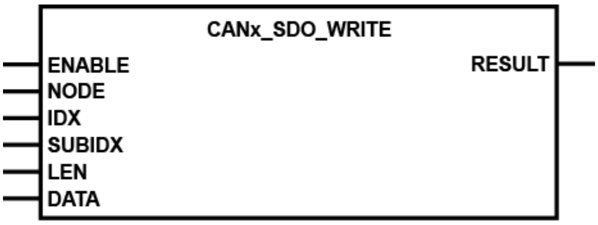
CANx_SDO_WRITE

615

Unit type = function block (FB)

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

Symbol in CoDeSys:



CAN1_SDO_WRITE

9451

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR2500
- PDM360smart: CR1070, CR1071

Description

618

CANx_SDO_WRITE writes the SDO (→ page 130) with the specified indexes to the node.

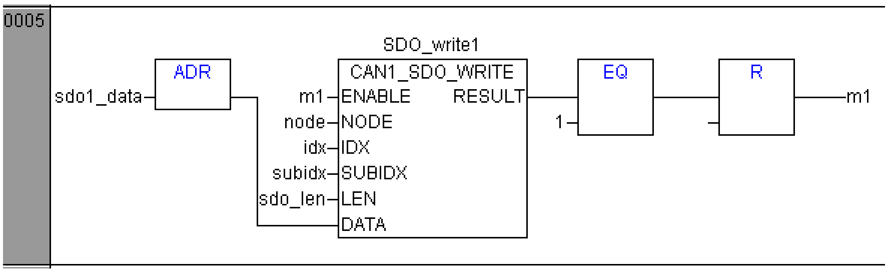
Using this FB, the entries can be written to the object directory. So it is possible to selectively set the node parameters.

- all **ecomat*mobile*** controllers - PCB controller: CS0015 - PDM360smart: CR1070, CR1071	- PDM360: CR1050, CR1051, CR1060 - PDM360compact: CR1052, CR1053, CR1055, CR1056
From the device library `ifm_CRnnnn_Vxxyyzz.LIB`	From the device library ifm_CANx_SDO_Vxxyyzz.LIB
Prerequisite: the node must be in the state "PRE-OPERATIONAL" or "OPERATIONAL" and in the mode "CANopen master".	Prerequisite: The node must be in the mode "CANopen master" or "CANopen slave".
For controllers, there only is CAN1_SDO_WRITE available.	

NOTE

The value for LEN must correspond to the length of the transmit array. Otherwise, problems with SDO communication will occur.

Example:



Parameters of the inputs

619

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
NODE	BYTE	number of the node
IDX	WORD	index in object directory
SUBIDX	BYTE	sub-index referred to the index in the object directory.
LEN	WORD	length of the entry in "number of bytes" The value for LEN must correspond to the length of the transmit array. Otherwise, problems with SDO communication will occur.
DATA	DWORD	address of the transmit data array permissible length = 0...255 transmission with ADR operator

Parameters of the outputs

620

Parameter	Data type	Description
RESULT	BYTE	0 = unit inactive 1 = execution of the unit stopped 2 = unit active 3 = error: unit has not been executed

8.7 CAN errors and error handling

Contents

CAN errors 182
Structure of an EMCY message........ 185
Overview CANopen error codes 187

1171

The error mechanisms described are automatically processed by the CAN controller integrated in the controller. This cannot be influenced by the user. (Depending on the application) the user should react to signalled errors in the application software.

Goal of the CAN error mechanisms:

- Ensuring uniform data objects in the complete CAN network
- Permanent functionality of the network even in case of a faulty CAN participant
- Differentiation between temporary and permanent disturbance of a CAN participant
- Localisation and self-deactivation of a faulty participant in 2 steps:
  - error passive
  - disconnection from the bus (bus off)
  This gives a temporarily disturbed participant a "rest".

To give the interested user an overview of the behaviour of the CAN controller in case of an error, error handling is easily described below. After error detection the information is automatically prepared and made available to the programmer as CAN error bits in the application software.

8.7.1 CAN errors

Contents

Error message........ 182
Error counter 183
Participant, error active 183
Participant, error passive 183
Participant, bus off........ 184

8589

Error message

1172

If a bus participant detects an error condition, it immediately transmits an error flag. The transmission is then aborted or the correct messages already received by other participants are rejected. This ensures that correct and uniform data is available to all participants. Since the error flag is directly transmitted the sender can immediately start to repeat the disturbed message as opposed to other fieldbus systems (they wait until a defined acknowledgement time has elapsed). This is one of the most important features of CAN.

One of the basic problems of serial data transmission is that a permanently disturbed or faulty bus participant can block the complete system. Error handling for CAN would increase such a risk. To exclude this, a mechanism is required which detects the fault of a participant and disconnects this participant from the bus, if necessary.

Error counter

1173

A transmit and receive error counter are integrated in the CAN controller. They are counted up (incremented) for every faulty transmit or receive operation. If a transmission was correct, these counters are counted down (decremented).

However, the error counters are more incremented in case of an error than decremented in case of success. Over a defined period this can lead to a considerable increase of the counts even if the number of the undisturbed messages is greater than the number of the disturbed messages. Longer undisturbed periods slowly reduce the counts. So the counts indicate the relative frequency of disturbed messages.

If the participant itself is the first to detect errors (= self-inflicted errors), the error is more severely "punished" for this participant than for other bus participants. To do so, the counter is incremented by a higher amount.

If the count of a participant exceeds a defined value, it can be assumed that this participant is faulty. To prevent this participant from disturbing bus communication by active error messages (error active), it is switched to "error passive".

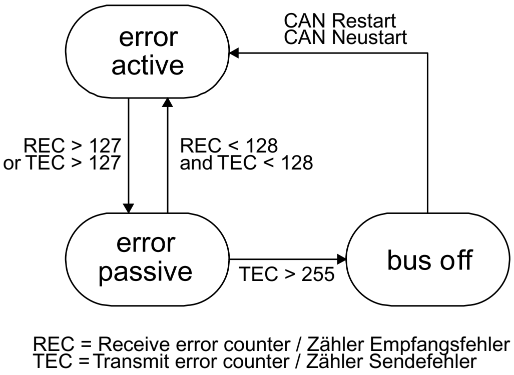


Figure: mechanism of the error counter

error active
→ participant, error active (→ page 183)

error passive
→ participant, error passive (→ page 183)

bus off
→ participant, bus off (→ page 184)

CAN restart
→ participant, bus off (→ page 184)

Participant, error active

1174

An error active participant participates in the bus communication without restriction and is allowed to signal detected errors by transmitting the active error flag. As already described the transmitted message is destroyed.

Participant, error passive

1175

An error passive participant can also communicate without restriction. However, it is only allowed to identify a detected error by a passive error flag, which does not interfere with the bus communication. An error passive participant becomes error active again if it is below a defined count value.

To inform the user about incrementing of the error counter, the system variable CANx_WARNING is set if the value of the error counter is > 96. In this state the participant is still error active.

Participant, bus off

1176

If the error count value continues to be incremented, the participant is disconnected from the bus (bus off) after exceeding a maximum count value.

To indicate this state the flag CANx_BUSOFF is set in the application program.

ⓘ NOTE

The error CANx_BUSOFF is automatically handled and reset by the operating system. If the error is to be handled or evaluated more precisely via the application program, CANx_ERRORHANDLER (→ page 97) must be used. The error CANx_BUSOFF must then be reset explicitly by the application program.

8.7.2 Structure of an EMCY message

Contents

A distinction is made between the following errors: 185
Structure of an error message 185
Identifier 186
EMCY error code 186
Object 0x1003 (error field) 186
Signalling of device errors 186

8591

Under CANopen error states are indicated via a simple standardised mechanism. For a CANopen device every occurrence of an error is indicated via a special message which details the error.

If an error or its cause disappears after a certain time, this event is also indicated via the EMCY message. The errors occurred last are stored in the object directory (object 1003_{16}) and can be read via an SDO access (→ CANx_SDO_READ (→ page 178)). In addition, the current error situation is reflected in the error register (object 1001_{16}).

A distinction is made between the following errors:

8046

Communication error

- The CAN controller signals CAN errors.
  (The frequent occurrence is an indication of physical problems. These errors can considerably affect the transmission behaviour and thus the data rate of a network.)
- Life guarding or heartbeat error

Application error

- Short circuit or wire break
- Temperature too high

Structure of an error message

8047

The structure of an error message (EMCY message) is as follows:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
EMCY error code as entered in the object 1003_{16}		object 1001_{16}	manufacturer-specific information				

Identifier

8048

The identifier for the error message consists of the sum of the following elements:

EMCY default identifier 128 (80_{16})
+
node ID

EMCY error code

8049

It gives detailed information which error occurred. A list of possible error codes has already been defined in the communication profile. Error codes which only apply to a certain device class are defined in the corresponding device profile of this device class.

Object 0x1003 (error field)

8050

The object 1003_{16} represents the error memory of a device. The sub-indices contain the errors occurred last which triggered an error message.

If a new error occurs, its EMCY error code is always stored in the sub-index 1_{16}. All other older errors are moved back one position in the error memory, i.e. the sub-index is incremented by 1. If all supported sub-indices are used, the oldest error is deleted. The sub-index 0_{16} is increased to the number of the stored errors. After all errors have been rectified the value "0" is written to the error field of the sub-index 1_{16}.

To delete the error memory the value "0" can be written to the sub-index 0_{16}. Other values must not be entered.

Signalling of device errors

1880

As described, EMCY messages are transmitted if errors occur in a device. In contrast to programmable devices error messages are automatically transmitted by decentralised input/output modules (e.g. CompactModules CR2033).
Corresponding error codes → corresponding device manual.

Programmable devices only generate an EMCY message automatically (e.g. short circuit on an output) if CANx_MASTER_EMCY_HANDLER (→ page 157) or CANx_SLAVE_EMCY_HANDLER (→ page 169) is integrated in the application program.

Overview of the automatically transmitted EMCY error codes for all **ifm** devices programmable with CoDeSys → chapter Overview of the CANopen error codes (→ page 187).

If in addition application-specific errors are to be transmitted by the application program, CANx_MASTER_SEND_EMERGENCY (→ page 159) or CANx_SLAVE_SEND_EMERGENCY (→ page 171) are used.

8.7.3 Overview CANopen error codes

8545

Error Code (hex)	Meaning
00xx	Reset or no error
10xx	Generic error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains voltage
32xx	Voltage inside the device
33xx	Output voltage
40xx	Temperature
41xx	Ambient temperature
42xx	Device temperature
50xx	Device hardware
60xx	Device software
61xx	Internal software
62xx	User software
63xx	Data set
70xx	Additional modules
80xx	Monitoring
81xx	Communication
8110	CAN overrun-objects lost
8120	CAN in error passiv mode
8130	Life guard error or heartbeat error
8140	Recovered from bus off
8150	Transmit COB-ID collision
82xx	Protocol error
8210	PDO not procedded due to length error
8220	PDO length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

Object 0x1001 (error register)

8547

This object reflects the general error state of a CANopen device. The device is to be considered as error free if the object 1001_{16} signals no error any more.

Bit	Meaning
0	generic error
1	current
2	voltage
3	temperature
4	communication error
5	device profile specific
6	reserved – always 0
7	manufacturer specific

For an error message more than one bit in the error register can be set at the same time.

Example: CR2033, message "wire break" at channel 2 (→ installation manual of the device):

COB-ID	DLC	Byte 0	Byte 1	Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
80_{16} + node ID		**00**	**FF**	**81**	**10**	**00**	00	00	00

Error-Code = $FF00_{16}$

Error register = 81_{16} = $1000\ 0001_2$, thus it consists of the following errors:
- generic error
- manufacturer specific

Concerned channel = 0010_{16} = $0000\ 0000\ 0001\ 0000_2$ = wire break channel 2

Manufacturer specific information

8548

A device manufacturer can indicate additional error information. The format can be freely selected.

Example:

In a device two errors occur and are signalled via the bus:

- Short circuit of the outputs:
  Error code 2300_{16},
  the value 03_{16} ($0000\ 0011_2$) is entered in the object 1001_{16}
  (generic error and current error)

- CAN overrun:
  Error code 8110_{16},
  the value 13_{16} ($0001\ 0011_2$) is entered in the object 1001_{16}
  (generic error, current error and communication error)

>> CAN overrun processed:
Error code 0000_{16},
the value 03_{16} ($0000\ 0011_2$) is entered in the object 1001_{16}
(generic error, current error, communication error reset)
It can be seen only from this information that the communication error is no longer present.

Overview CANopen EMCY codes (CR030n)

2671

All indications (hex) for the 1st CAN interface

EMCY code object 1003_{16}		Object 1001_{16}	Manufacturer-specific information					
Byte 0	1	2	3	4	5	6	7	Description
00	21	03	I0					Diagnosis analogue current inputs
00	31	05						Terminal voltage VBBo/VBBs
00	61	11						Memory error
00	80	11						CAN1 monitoring SYNC error (only slave)
00	81	11						CAN1 warning threshold (> 96)
10	81	11						CAN1 receive buffer overrun
11	81	11						CAN1 transmit buffer overrun
30	81	11						CAN1 guard/heartbeat error (only slave)

9 In-/output functions

Contents

Processing input values 190
Adapting analogue values 195
Counter functions for frequency and period measurement 198
PWM functions 213
Controller functions 245

1590

In this chapter you will find FBs which allow you to read and process the signals of the in- and outputs.

9.1 Processing input values

Contents

INPUT_ANALOG........ 191
INPUT_VOLTAGE........ 193
INPUT_CURRENT 194

1602

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

9.1.1 INPUT_ANALOG

519

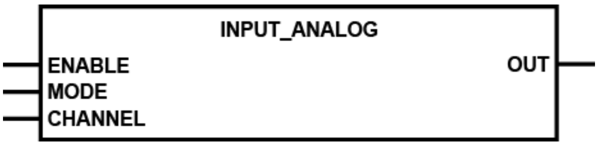
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
  (For safety signals use SAFE_ANALOG_OK in addition!)

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

522

INPUT_ANALOG enables current and voltage measurements at the analogue channels.

The FB provides the current analogue value at the selected analogue channel. The measurement and the output value result from the operating mode specified via MODE (digital input, 0...20 mA, 0...10 V, 0...30 V).

MODE	Input operating mode	Output OUT	Unit
IN_DIGITAL_H	digital input	0 / 1	---
IN_CURRENT	current input	0...20 000	µA
IN_VOLTAGE10	voltage input	0...10 000	mV
IN_VOLTAGE30	voltage input	0...30 000	mV
IN_VOLTAGE32	voltage input	0...32 000	mV
IN_RATIO	voltage input ratiometric	0...1 000	‰

For parameter setting of the operating mode, the indicated global system variables should be used. The analogue values are provided as standardised values.

NOTE

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

*) Relay exists only in the following devices:
CR0020, CR0032, CR0200, CR0232, CR0505, CR7020, CR7021, CR7200, CR7201, CR7505, CR7506

Parameters of the inputs

523

Parameter	Data type	Description		
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active		
MODE	BYTE	IN_DIGITAL_H	digital input	
		IN_CURRENT	current input	0...20 000 µA
		IN_VOLTAGE10	voltage input	0...10 000 mV
		IN_VOLTAGE30	voltage input	0...30 000 mV
		IN_VOLTAGE32	voltage input	0...32 000 mV
		IN_RATIO	ratiometric analogue input	
INPUT_CHANNEL	BYTE	input channel		

Parameters of the outputs

524

Parameter	Data type	Description
OUT	WORD	output value

9.1.2 INPUT_VOLTAGE

507

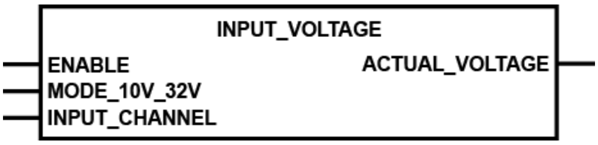
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

510

INPUT_VOLTAGE processes analogue voltages measured on the analogue channels.

> The FB returns the current input voltage in [mV] on the selected analogue channel. The measurement refers to the voltage range defined via MODE_10V_32V (10 000 mV or 32 000 mV).

Info

INPUT_VOLTAGE is a compatibility FB for older programs. In new programs, the more powerful INPUT_ANALOG (→ page 191) should be used.

Parameters of the inputs

511

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
MODE_10V_32V	BOOL	TRUE: voltage range 0...32 V FALSE: voltage range 0...10 V
INPUT_CHANNEL	BYTE	input channel

Parameters of the outputs

512

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

9.1.3 INPUT_CURRENT

513

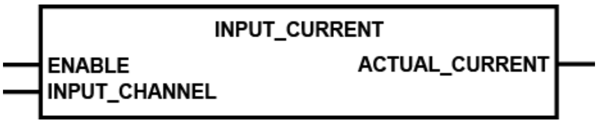
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

516

INPUT_CURRENT returns the actual input current in [µA] at the analogue current inputs.

Info

INPUT_CURRENT is a compatibility FB for older programs. In new programs, the more powerful INPUT_ANALOG (→ page 191) should be used.

Parameters of the inputs

517

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
INPUT_CHANNEL	BYTE	analogue current inputs 4...7

Parameters of the outputs

518

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

9.2 Adapting analogue values

Contents

NORM 196

1603

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

9.2.1 NORM

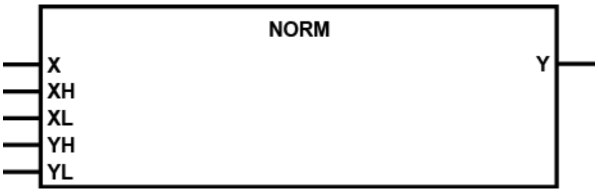
401

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

404

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

The FB normalises a value of type WORD within the limits of XH and XL to an output value within the limits of YH and YL. This FB is for example used for generating PWM values from analogue input values.

NOTE

The value for X must be in the defined input range between XL and XH (there is no internal plausibility check of the value).

Due to rounding errors the normalised value can deviate by 1.

If the limits (XH/XL or YH/YL) are defined in an inverted manner, normalisation is also done in an inverted manner.

Parameters of the inputs

405

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

Parameters of the outputs

406

Parameter	Data type	Description
Y	WORD	normalised value

Example 1

407

lower limit value input	0	XL
upper limit value input	100	XH
lower limit value output	0	YL
upper limit value output	2000	YH

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

from X =	50	0	100	75
to Y =	1000	0	2000	1500

Example 2

408

lower limit value input	2000	XL
upper limit value input	0	XH
lower limit value output	0	YL
upper limit value output	100	YH

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

from X =	1000	0	2000	1500
to Y =	50	100	0	25

9.3 Counter functions for frequency and period measurement

Contents

Applications 198
Use as digital inputs 199

1591

Depending on the controller up to 16 fast inputs are supported which can process input frequencies of up to 30 kHz. Further to the pure frequency measurement at the inputs FRQ, the inputs ENC can be also used to evaluate incremental encoders (counter function) with a maximum frequency of 10 kHz. The inputs CYL are used for period measurement of slow signals.

Input	Frequency [kHz]	Description
FRQ 0 / ENC 0	30 / 10	frequency measurement / encoder 1, channel A
FRQ 1 / ENC 0	30 / 10	frequency measurement / encoder 1, channel B
FRQ 2 / ENC 1	30 / 10	frequency measurement / encoder 2, channel A
FRQ 3 / ENC 1	30 / 10	frequency measurement / encoder 2, channel B
CYL 0 / ENC 2	10	period measurement / encoder 3, channel A
CYL 1 / ENC 2	10	period measurement / encoder 3, channel B
CYL 2 / ENC 3	10	period measurement / encoder 4, channel A
CYL 3 / ENC 3	10	period measurement / encoder 4, channel B

The following functions are available for easy evaluation:

9.3.1 Applications

1592

It must be taken into account that the different measuring methods can cause errors in the frequency detection.

FREQUENCY (→ page 200) is suitable for frequencies between 100 Hz and 30 kHz; the error decreases at high frequencies.

PERIOD (→ page 202) carries out a period measurement. It is thus suitable for frequencies lower than 1000 Hz. In principle it can also measure higher frequencies, but this has a significant impact on the cycle time. This must be taken into account when setting up the application software.

9.3.2 Use as digital inputs

Contents

FREQUENCY 200
PERIOD 202
PERIOD_RATIO 204
PHASE 206
INC_ENCODER 208
FAST_COUNT 211

1593

If the fast inputs (FRQx / CYLx) are used as "normal" digital inputs, the increased sensitivity to interfering pulses must be taken into account (e.g. contact bouncing for mechanical contacts). The standard digital input has an input frequency of 50 Hz. If necessary, the input signal must be debounced by means of the software.

FREQUENCY

537

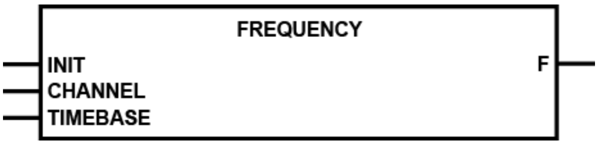
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
  (For safety signals use SAFE_FREQUENCY_OK together with PERIOD (→ page 202)!)
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

540

FREQUENCY measures the signal frequency at the indicated channel. Maximum input frequency → data sheet.

This FB measures the frequency of the signal at the selected CHANNEL. To do so, the positive edge is evaluated. Depending on the TIMEBASE, frequency measurements can be carried out in a wide value range. High frequencies require a short time base, low frequencies a correspondingly longer time base. The frequency is provided directly in [Hz].

! NOTE

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

Parameters of the inputs

541

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
TIMEBASE	TIME	time base

NOTE

The FB may provide wrong values before initialisation.

► Only evaluate the output if the FB has been initialised.

Parameters of the outputs

542

Parameter	Data type	Description
F	REAL	frequency in [Hz]

PERIOD

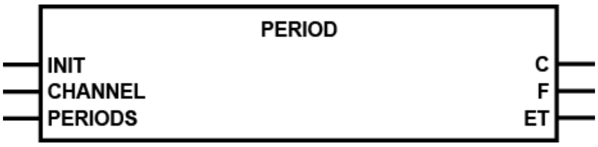
370

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
  (For safety signals use SAFE_FREQUENCY_OK together with FREQUENCY (→ page 200) in addition!)
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

373

PERIOD measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel. Maximum input frequency → data sheet.

This FB measures the frequency and the cycle time of the signal at the selected CHANNEL. To calculate, all positive edges are evaluated and the average value is determined by means of the number of indicated PERIODS.

In case of low frequencies there will be inaccuracies when using FREQUENCY. To avoid this, PERIOD can be used. The cycle time is directly indicated in [µs].

The maximum measuring range is approx. 71 min.

NOTE

For PERIOD only the inputs CYL0...CYL3 can be used.

Frequencies < 0.5 Hz are no longer clearly indicated!

Parameters of the inputs

374

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
PERIODS	BYTE	number of periods to be compared

NOTE

The FB may provide wrong values before initialisation.

► Do not evaluate the output before the FB has been initialised.

We urgently recommend to program an own instance of this FB for each channel to be evaluated. Otherwise, wrong values may be provided.

Parameters of the outputs

375

Parameter	Data type	Description
C	DWORD	cycle time of the detected periods in [µs]
F	REAL	frequency of the detected periods in [Hz]
ET	TIME	time elapsed since the beginning of the period measurement (can be used for very slow signals)

PERIOD_RATIO

364

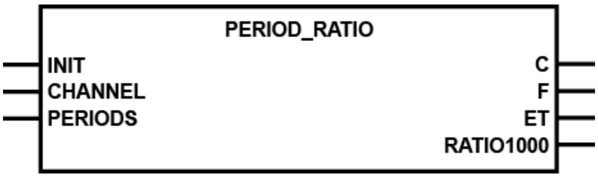
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

367

PERIOD_RATIO measures the frequency and the cycle period (cycle time) in [µs] during the indicated periods at the indicated channel. In addition, the mark-to-space ratio is indicated in per mill. Maximum input frequency → data sheet.

This FB measures the frequency and the cycle time of the signal at the selected CHANNEL. To calculate, all positive edges are evaluated and the average value is determined by means of the number of indicated PERIODS. In addition, the mark-to-space ratio is indicated in [‰].

For example: In case of a signal ratio of 25 ms high level and 75 ms low level the value RATIO1000 is provided as 250 ‰.

In case of low frequencies there will be inaccuracies when using FREQUENCY. To avoid this, PERIOD_RATIO can be used. The cycle time is directly indicated in [µs].

The maximum measuring range is approx. 71 min.

NOTE

For PERIOD_RATIO only the inputs CYL0...CYL3 can be used.

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

Frequencies < 0.05 Hz are no longer clearly indicated!

Parameters of the inputs

368

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
PERIODS	BYTE	number of periods to be compared

ⓘ NOTE

The FB may provide wrong values before initialisation.

► Do not evaluate the output before the FB has been initialised.

We urgently recommend to program an own instance of this FB for each channel to be evaluated. Otherwise, wrong values may be provided.

Parameters of the outputs

369

Parameter	Data type	Description
C	DWORD	cycle time of the detected periods in [µs]
F	REAL	frequency of the detected periods in [Hz]
ET	TIME	time elapsed since the beginning of the last change in state of the input signal (can be used for very slow signals)
RATIO1000	WORD	mark-to-space ratio in [‰]

PHASE

358

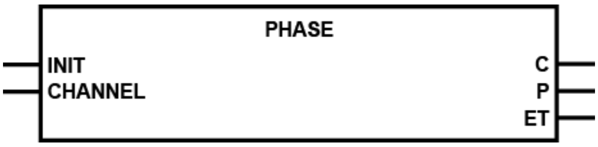
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

361

PHASE reads a pair of channels with fast inputs and compares the phase position of the signals. Maximum input frequency → data sheet.

This FB compares a pair of channels with fast inputs so that the phase position of two signals towards each other can be evaluated. An evaluation of the cycle period is possible even in the range of seconds.

NOTE

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

Parameters of the inputs

362

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the input channel pair (0/2): 0 = channel pair 0 = inputs 0 + 1 2 = channel pair 1 = inputs 2 + 3

NOTE

The FB may provide wrong values before initialisation.

► Do not evaluate the output before the FB has been initialised.

We urgently recommend to program an own instance of this FB for each channel to be evaluated. Otherwise, wrong values may be provided.

Parameters of the outputs

363

Parameter	Data type	Description
C	DWORD	cycle period in [µs]
P	INT	angle of the phase shift (0...360 °)
ET	TIME	time elapsed since the beginning of the period measurement (can be used for very slow signals)

INC_ENCODER

4187

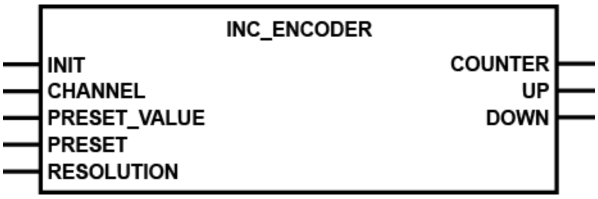
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

4330
2602

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

Two frequency inputs form the input pair which is evaluated by means of the FB. The following table shows the permissible limit frequencies and the max. number of incremental encoders that can be connected:

Device	Limit frequency	max. number of encoders
BasicController: CR040n	1 kHz	2
CabinetController: CR030n	10 kHz	2
ClassicController: CR0020, CR0505	10 kHz	4
ClassicController: CR0032	30 kHz	4
ExtendedController: CR0200	10 kHz	8
ExtendedController: CR0232	30 kHz	8
PCB controller: CS0015	0.5 kHz	2
SafetyController: CR7020, CR7021, CR7505, CR7506	10 kHz	4
SafetyController: CR7032	30 kHz	4
ExtendedSafetyController: CR7200, CR7201	10 kHz	8
ExtendedSafetyController: CR7232	30 kHz	8
SmartController: CR25nn	10 kHz	2
PDM360smart: CR1071	1 kHz	2

NOTE

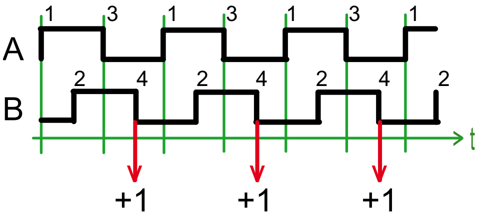
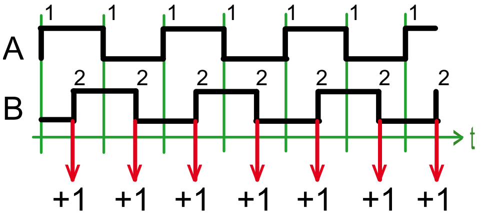
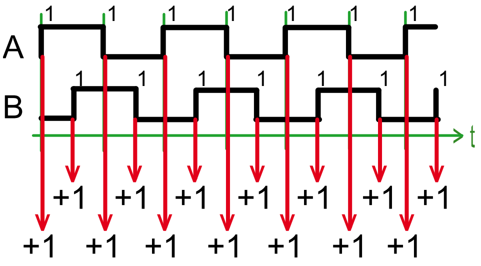
Depending on the further load on the unit the limit frequency might fall when "many" encoders are evaluated.

If the load is too high the cycle time can get unacceptably long (→ Limitations and programming notes (→ page 47)).

Via PRESET_VALUE the counter can be set to a preset value. The value is adopted if PRESET is set to TRUE. Afterwards, PRESET must be set to FALSE again for the counter to become active again.

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

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

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

Parameters of the inputs

4332
529

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the input channel pair (0...3): 0 = channel pair 0 = inputs 0 + 1 1 = channel pair 1 = inputs 2 + 3 2 = channel pair 2 = inputs 4 + 5 3 = channel pair 3 = inputs 6 + 7
PRESET_VALUE	DINT	preset value of the counter
PRESET	BOOL	TRUE (only 1 cycle): preset value is adopted FALSE: counter active
RESOLUTION	BYTE	factor of the encoder resolution (1, 2, 4): 1 = normal resolution 2 = double resolution 4 = 4-fold resolution all other values count as "1"

Parameters of the outputs

530

Parameter	Data type	Description
COUNTER	DINT	current counter value
UP	BOOL	TRUE: counter counts upwards FALSE: counter stands still
DOWN	BOOL	TRUE: counter counts downwards FALSE: counter stands still

FAST_COUNT

567

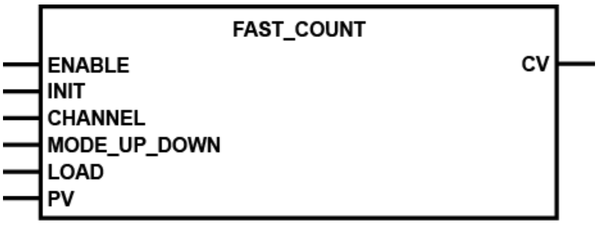
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

570

FAST_COUNT operates as counter block for fast input pulses.

This FB detects fast pulses at the FRQ input channels 0...3. With the FRQ input channel 0 FAST_COUNT operates like the block CTU. Maximum input frequency → data sheet.

NOTE

For the **ecomatmobile** controllers channel 0 can only be used as up counter. The channels 1...3 can be used as up and down counters.

Parameters of the inputs

571

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed starting from the start value FALSE: unit is not executed
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
CHANNEL	BYTE	number of the fast input channel (0...x, value depends on the device, → data sheet)
MODE_UP_DOWN	BOOL	TRUE: counter counts downwards. FALSE: counter counts upwards
LOAD	BOOL	TRUE: start value PV being loaded FALSE: start value "0" being loaded
PV	DWORD CR1071: WORD	start value (preset value)

! NOTE

After setting the parameter ENABLE the counter counts as from the indicated start value.

The counter does NOT continue from the value which was valid at the last deactivation of ENABLE.

Parameters of the outputs

572

Parameter	Data type	Description
CV	DWORD CR1071: WORD	output value of the counter

9.4 PWM functions

Contents

Availability of PWM.. 213
PWM signal processing.. 214
Hydraulic control in PWMi.. 227

2303

In this chapter you will find out more about the pulse width modulation in the **ifm** device.

9.4.1 Availability of PWM

8472

PWM is available in the following devices:

	Number of available PWM outputs	of which current-controlled (PWMi)	PWM frequency [Hz]
BasicController: CR0401	8	0	20...250
BasicController: CR0403	12	2	20...250
CabinetController: CR0301	4	0	25...250
CabinetController: CR0302, CR0303	8	0	25...250
ClassicController: CR0020	12	8	25...250
ClassicController: CR0505	8	8	25...250
ClassicController: CR0032	16	16	2...2000
ExtendedController: CR0200	24	16	25...250
ExtendedController: CR0232	32	32	2...2000
PCB controller: CS0015	8	0	25...250
SafetyController: CR7020, CR7021	12	8	25...250
SafetyController: CR7505, CR0506	8	8	25...250
ExtendedSafetyController: CR7200, CR7201	24	16	25...250
SmartController: CR25nn	4	4	25...250
PDM360smart: CR1071	4	0	25...250

9.4.2 PWM signal processing

Contents

PWM – introduction 214
PWM functions and their parameters 215

1526

PWM – introduction

6889

The abbreviation PWM stands for **p**ulse **w**idth **m**odulation. It is mainly used to trigger proportional valves (PWM valves) for mobile and robust controller applications. Also, with an additional component (accessory) for a PWM output the pulse-width modulated output signal can be converted into an analogue output voltage.

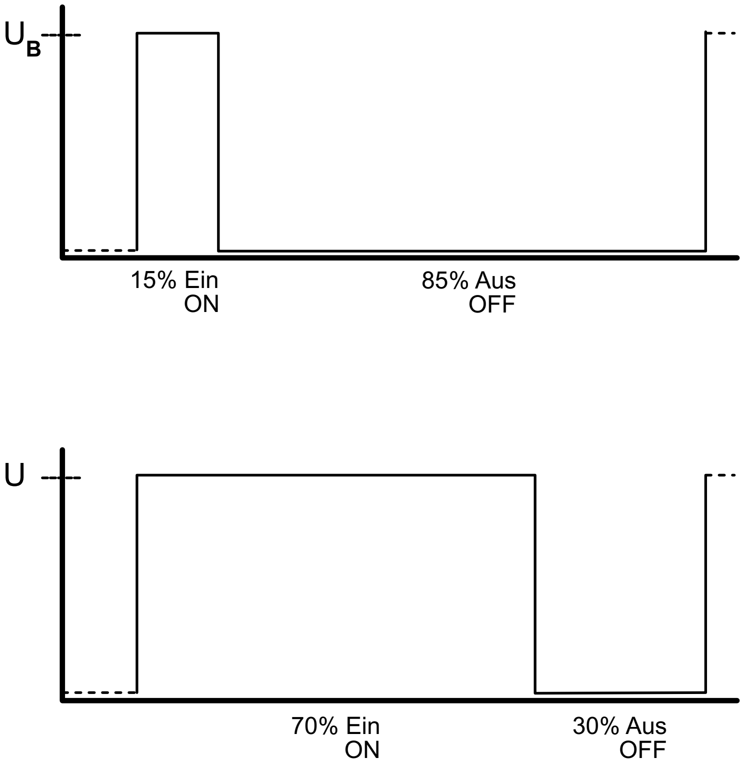


Figure: PWM principle

The PWM output signal is a pulsed signal between GND and supply voltage. Within a defined period (PWM frequency) the mark-to-space ratio is then varied. Depending on the mark-to-space ratio, the connected load determines the corresponding RMS current.

The PWM function of the **ecomat*mobile*** controller is a hardware function provided by the processor. To use the integrated PWM outputs of the controller, they must be initialised in the application program and parameterised corresponding to the requested output signal.

PWM functions and their parameters

Contents

PWM / PWM1000 215
PWM frequency 215
PWM channels 0...3 216
Calculation of the RELOAD value 216
Calculation examples RELOAD value 217
PWM channels 4...7 / 8...11 218
PWM dither 220
Ramp function 220
PWM 221
PWM100 223
PWM1000 225

1527

PWM / PWM1000

1528

Depending on the application and the requested resolution, PWM or PWM1000 can be selected for the application programming. High accuracy and thus resolution is required when using the control functions. This is why the more technical PWM FB is used in this case.

If the implementation is to be kept simple and if there are no high requirements on the accuracy, PWM1000 (→ page 225) can be used. For this FB the PWM frequency can be directly entered in [Hz] and the mark-to-space ratio in steps of 1 ‰.

PWM frequency

1529

Depending on the valve type, a corresponding PWM frequency is required. For the PWM function the PWM frequency is transmitted via the reload value (PWM (→ page 221)) or directly as a numerical value in [Hz] (PWM1000 (→ page 225)). Depending on the controller, the PWM outputs differ in their operating principle but the effect is the same.

The PWM frequency is implemented by means of an internally running counter, derived from the CPU pulse. This counter is started with the initialisation of the PWM. Depending on the PWM output group (0...3 and / or 4...7 or 4...11), it counts from $FFFF_{16}$ backwards or from 0000_{16} forwards. If a transmitted comparison value (VALUE) is reached, the output is set. In case of an overflow of the counter (change of the counter reading from 0000_{16} to $FFFF_{16}$ or from $FFFF_{16}$ to 0000_{16}), the output is reset and the operation restarts.

If this internal counter shall not operate between 0000_{16} and $FFFF_{16}$, another preset value (RELOAD) can be transmitted for the internal counter. In doing so, the PWM frequency increases. The comparison value must be within the now specified range.

PWM channels 0...3

1530

These 4 PWM channels allow the most flexibility for the parameter setting. The PWM channels 0...3 are available in all **ecomatmobile** controller versions; depending on the type they feature a current control or not.

For each channel an own PWM frequency (RELOAD value) can be set. There is a free choice between PWM (→ page 221) and PWM1000 (→ page 225).

Calculation of the RELOAD value

1531

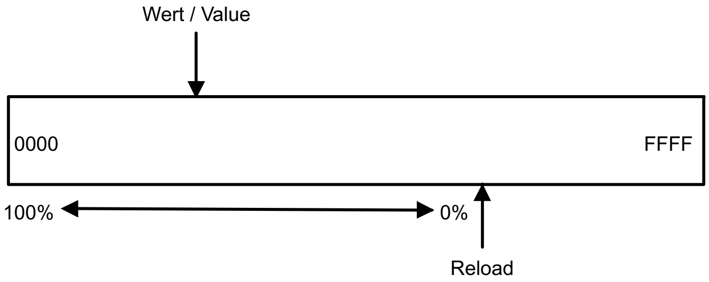


Figure: RELOAD value for the PWM channels 0...3

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

	ClassicController **ExtendedController** **SafetyController** **CabinetController (CR0303)**	**SmartController** **CabinetController (CR0301/CR0302)** **PCB controller**
DIV64 = 0	RELOAD = 20 MHz / f_{PWM}	RELOAD = 10 MHz / f_{PWM}
DIV64 = 1	RELOAD = 312.5 kHz / f_{PWM}	RELOAD = 156.25 kHz / f_{PWM}

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1). In case of frequencies below 305 Hz respectively 152 Hz (according to the controller), DIV64 must be set to "1" to ensure that the RELOAD value is not greater than $FFFF_{16}$.

Calculation examples RELOAD value

1532

ClassicController **ExtendedController** **SafetyController** **CabinetController (CR0303)**		**SmartController** **CabinetController (CR0301/CR0302)** **PCB controller**
The PWM frequency shall be 400 Hz. $\frac{20\ \text{MHz}}{400\ \text{Hz}} = 50\,000_{10} = \text{C350}_{16} = \text{RELOAD}$ Thus the permissible range of the PWM value is the range from 0000_{16} to C350_{16}. The comparison value at which the output switches must then be between 0000_{16} and C350_{16}.		The PWM frequency shall be 200 Hz. $\frac{10\ \text{MHz}}{200\ \text{Hz}} = 50\,000_{10} = \text{C350}_{16} = \text{RELOAD}$ Thus the permissible range of the PWM value is the range from 0000_{16} to C350_{16}. The comparison value at which the output switches must then be between 0000_{16} und C350_{16}.

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

Mark-to-space ratio	Switch-on time	Value for mark-to-space ratio
Minimum	0 %	C350_{16}
Maximum	100 %	0000_{16}

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

PWM channels 4...7 / 8...11

1533

These 4/8 PWM channels can only be set to one common PWM frequency. For programming, PWM and PWM1000 must not be mixed.

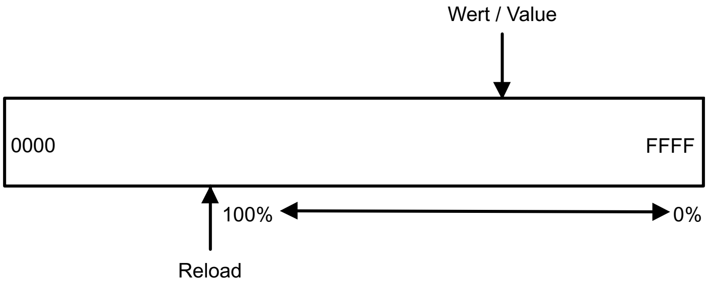


Figure: RELOAD value for PWM channels 4...7 / 8...11

The RELOAD value of the internal PWM counter is calculated (for all **ecomat*mobile*** controllers) on the basis of the parameters DIV64 and the CPU frequency as follows:

DIV64 = 0	RELOAD = $10\ 000_{16}$ – (2.5 MHz / f_{PWM})
DIV64 = 1	RELOAD = $10\ 000_{16}$ – (312.5 kHz / f_{PWM})

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1). In case of PWM frequencies below 39 Hz, DIV64 must be set to "1" to ensure that the RELOAD value is not smaller than 0000_{16}.

Example:

The PWM frequency shall be 200 Hz.

$$\frac{2.5\ \text{MHz}}{200\ \text{Hz}} = 12\,500_{10} = 30D4_{16}$$

RELOAD value = $10\,000_{16} – 30D4_{16} = CF2C_{16}$.

Thus the permissible range of the PWM value is the range from $CF2C_{16}$ to $FFFF_{16}$.

The comparison value at which the output switches must then be between $CF2C_{16}$ and $FFFF_{16}$.

! NOTE

The PWM frequency is the same for all PWM outputs (4...7 or 4...11).

PWM and PWM1000 must not be mixed.

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

Mark-to-space ratio	Switch-on time	Value for mark-to-space ratio
Minimum	0 %	$FFFF_{16}$
Maximum	100 %	$CF2C_{16}$

Between minimum and maximum triggering 12 500 intermediate values (PWM values) are possible.

! NOTE

for ClassicController and ExtendedController applies:

If the PWM outputs 4...7 are used (regardless of whether current-controlled or via one of the PWM FBs) the same frequency and the corresponding reload value have to be set for the outputs 8...11. This means that the same FBs have to be used for these outputs.

PWM dither

1534

For certain hydraulic valve types a so-called dither frequency must additionally be superimposed on the PWM frequency. If valves were triggered over a longer period by a constant PWM value, they could block due to the high system temperatures.

To prevent this, the PWM value is increased or reduced on the basis of the dither frequency by a defined value (DITHER_VALUE). As a consequence a vibration with the dither frequency and the amplitude DITHER_VALUE is superimposed on the constant PWM value. The dither frequency is indicated as the ratio (divider, DITHER_DIVIDER * 2) of the PWM frequency.

Ramp function

1535

In order to prevent abrupt changes from one PWM value to the next, e.g. from 15 % ON to 70 % ON (→ figure in PWM – introduction (→ page 214)), it is possible to delay the increase by using PT1. The ramp function used for PWM is based on the CoDeSys library `UTIL.LIB`. This allows a smooth start e.g. for hydraulic systems.

NOTE

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

There you also find projects with examples regarding this subject. It is strongly recommended to follow the shown procedure.
→ chapter ifm demo programs (→ page 34)

NOTE

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

PWM

320

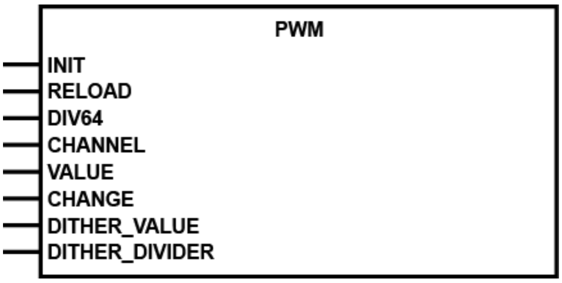
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

323

PWM is used for initialisation and parameter setting of the PWM outputs.

PWM has a more technical background. Due to their structure, PWM values can be very finely graded. So, this FB is suitable for use in controllers.

PWM is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the parameter RELOAD is also assigned.

NOTE

The value RELOAD must be identical for the channels 4...7 (for the ClassicController or ExtendedController: 4...11).

For these channels, PWM and PWM1000 (→ page 225) must not be mixed.

The PWM frequency (and so the RELOAD value) is internally limited to 5 kHz.

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1).

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:

- via OUTPUT_CURRENT *)
  *) Applies only to the following devices:
  - ClassicController: CR0020, CR0032, CR0505
  - ExtendedController: CR0200, CR0232
  - SafetyController: CR7nnn
  - SmartController: CR25nn
- or for example using the **ifm** unit EC2049 (series element for current measurement).

PWM_Dither is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the DIVIDER for the determination of the dither frequency and the VALUE are assigned.

Info

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

Parameters of the inputs

324

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
RELOAD	WORD	Value for the determination of the PWM frequency (→ chapter Calculation of the RELOAD value (→ page 216))
DIV64	BOOL	CPU cycle / 64
CHANNEL	BYTE	current PWM channel / output
VALUE	WORD	current PWM value
CHANGE	BOOL	TRUE: new PWM value is adopted FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	WORD	amplitude of the dither value (→ chapter PWM dither (→ page 220))
DITHER_DIVIDER	WORD	dither frequency = PWM frequency / DIVIDER * 2

PWM100

332

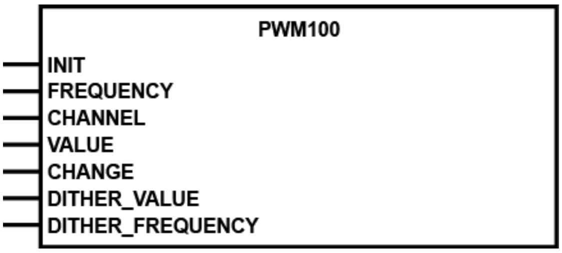
IMPORTANT: New **ecomat*mobile*** controllers only support PWM1000 (→ page 225).

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7200, CR7505
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

335

PWM100 handles the initialisation and parameter setting of the PWM outputs.

The FB enables a simple application of the PWM FB in the **ecomat*mobile*** controller. The PWM frequency can be directly indicated in [Hz] and the mark-to-space ratio in steps of 1 %. This FB is **not** suited for use in controllers, due to the relatively coarse grading.

The FB is called once for each channel in the initialisation of the application program. For this, the input INIT must be set to TRUE. During initialisation, the parameter FREQUENCY is also assigned.

NOTE

The value FREQUENCY must be identical for the channels 4...7 (for the ClassicController or ExtendedController: 4...11).

For these channels, PWM (→ page 221) and PWM100 must not be mixed.

The PWM frequency is limited to 5 kHz internally.

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:

- via OUTPUT_CURRENT *)
  *) Applies only to the following devices:
  - ClassicController: CR0020, CR0032, CR0505
  - ExtendedController: CR0200, CR0232
  - SafetyController: CR7nnn
  - SmartController: CR25nn
- or for example using the **ifm** unit EC2049 (series element for current measurement).

DITHER is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the value FREQUENCY for determining the dither frequency and the dither value (VALUE) are transmitted.

Info

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

Parameters of the inputs

336

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
FREQUENCY	WORD	PWM frequency in [Hz]
CHANNEL	BYTE	current PWM channel / output
VALUE	BYTE	current PWM value
CHANGE	BOOL	TRUE: new PWM value is adopted FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	BYTE	amplitude of the dither value in [%]
DITHER_FREQUENCY	WORD	dither frequency in [Hz]

PWM1000

326

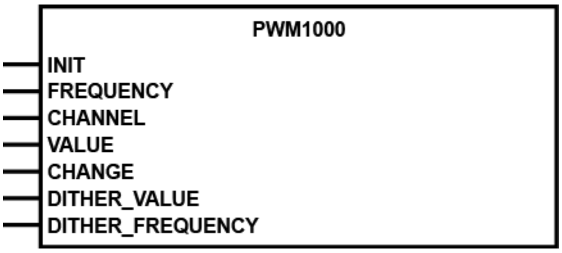
Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

NOTE: For the extended side of the ExtendedControllers the FB name ends with "_E".

Symbol in CoDeSys:



Description

329

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

The FB enables a simple use of the PWM FB in the **ecomat*mobile*** device. The PWM frequency can be directly indicated in [Hz] and the mark-to-space ratio in steps of 1 ‰.

The FB is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the parameter FREQUENCY is also assigned.

NOTE

The value FREQUENCY must be identical for the channels 4...7 (for the ClassicController or ExtendedController: 4...11).

For these channels, PWM (→ page 221) and PWM1000 must not be mixed.

The PWM frequency is limited to 5 kHz internally.

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:

- via OUTPUT_CURRENT *)
  *) Applies only to the following devices:
  - ClassicController: CR0020, CR0032, CR0505
  - ExtendedController: CR0200, CR0232
  - SafetyController: CR7nnn
  - SmartController: CR25nn
- or for example using the **ifm** module EC2049 (series element for current measurement).

DITHER is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the value FREQUENCY for determining the dither frequency and the dither value (VALUE) are transmitted.

Info

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

Parameters of the inputs

330

Parameter	Data type	Description
INIT	BOOL	TRUE (for only 1 cycle): unit is initialised FALSE: during further processing of the program
FREQUENCY	WORD	PWM frequency in [Hz]
CHANNEL	BYTE	current PWM channel / output
VALUE	WORD	current PWM value
CHANGE	BOOL	TRUE: new PWM value is adopted FALSE: the changed PWM value has no influence on the output
DITHER_VALUE	WORD	amplitude of the dither value in [%]
DITHER_FREQUENCY	WORD	dither frequency in [Hz]

9.4.3 Hydraulic control in PWMi

Contents

The purpose of this library? – An introduction 227
What does a PWM output do? 228
What is the dither? 229
Functions of the library ifm_HYDRAULIC_CR0303........ 231

1559

ifm electronic offers the user special functions to control hydraulic systems as a special field of current regulation with PWM.

The purpose of this library? – An introduction

1560

Thanks to the FBs of this library you can fulfil the following tasks:

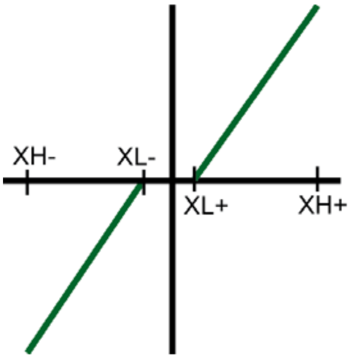
Standardise the output signals of a joystick

1561

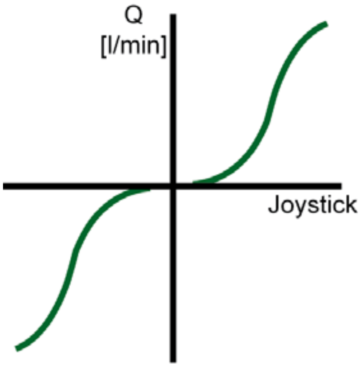
It is not always intended that the whole movement area of the joy stick influences the movement of the machine.

Often the area around the neutral position of the joy stick is to be spared because the joy stick does not reliably supply 0 V in this neutral position.

Here in this figure the area between XL- and XL+ is to be spared.



The FBs of this library enable you to adapt the characteristic curve of your joy stick according to your requirements – on request even freely configurable:



What does a PWM output do?

1563

PWM stands for "**p**ulse **w**idth **m**odulation" which means the following principle:

In general, digital outputs provide a fixed output voltage as soon as they are switched on. The value of the output voltage can*not* be changed here. The PWM outputs, however, split the voltage into a quick sequence of many square-wave pulse trains. The pulse duration [switched on] / pulse duration [switched off] ratio determines the effective value of the requested output voltage. This is referred to as the switch-on time in [%].

Info

In the following sketches the current profiles are shown as a stylised straight line. In reality the current flows to an e-function.

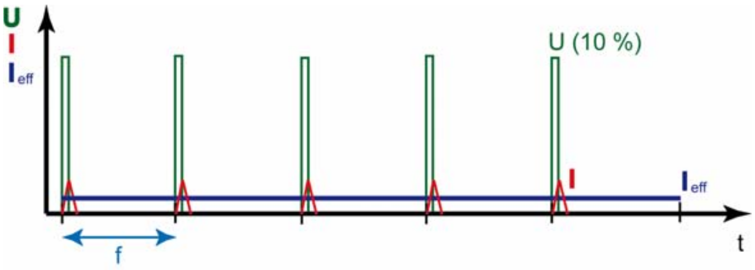


Figure: The profile of the PWM voltage U and the coil current I at 10 % switch-on time:
The effective coil current I_{eff} is also 10 %

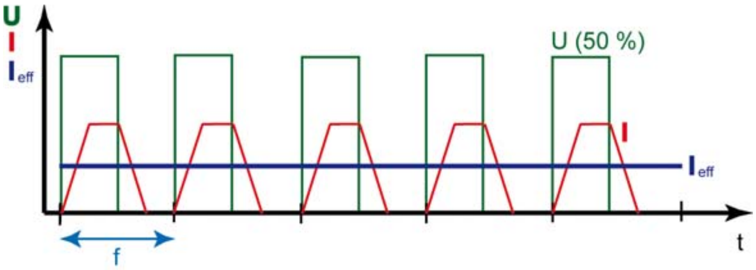


Figure: The profile of the PWM voltage U and the coil current I at 50 % switch-on time:
The effective coil current I_{eff} is also 50 %

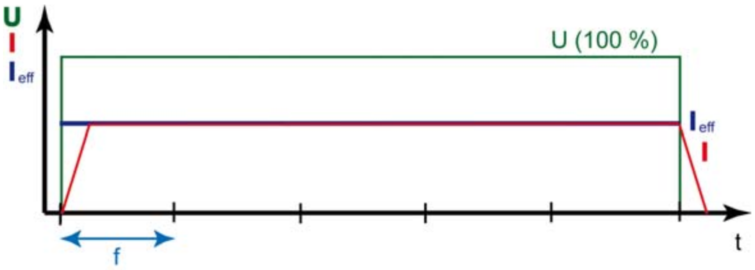


Figure: The profile of the PWM voltage U and the coil current I at 100 % switch-on time:
The effective coil current I_{eff} is also 100 %

What is the dither?

1564

If a proportional hydraulic valve is controlled, its piston does not move right away and at first not proportional to the coil current. Due to this "slip stick effect" – a kind of "break-away torque" – the valve needs a slightly higher current at first to generate the power it needs to move the piston from its off position. The same also happens for each other change in the position of the valve piston. This effect is reflected in a jerking movement, especially at very low manipulating speeds.

Technology solves this problem by having the valve piston move slightly back and forth (dither). The piston is continuously vibrating and cannot "stick". Also a small change in position is now performed without any delay, a "running start" so to speak.

Advantage: The hydraulic cylinder controlled in that way can be moved more sensitively.

Disadvantage: The valve becomes measurably hotter with dither than without because the valve coil is now working continuously.

That means that the "golden means" has to be found.

When is a dither useful?

1565

When the PWM output provides a pulse frequency that is small enough (standard value: up to 250 Hz) so that the valve piston continuously moves at a minimum stroke, an additional dither is not required (→ next figure):

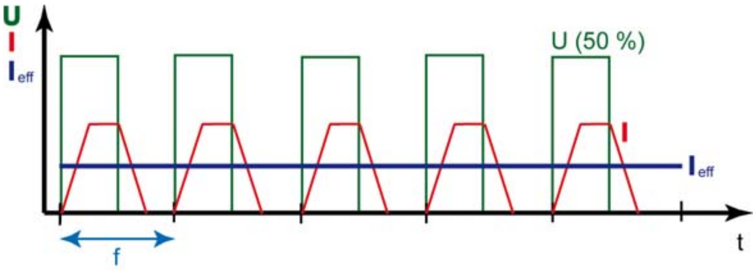


Figure: Balanced PWM signal; no dither required.

At a higher PWM frequency (standard value 250 Hz up to 1 kHz) the remaining movement of the valve piston is so short or so slow that this effectively results in a standstill so that the valve piston can again get stuck in its current position (and will do so!) (→ next figures):

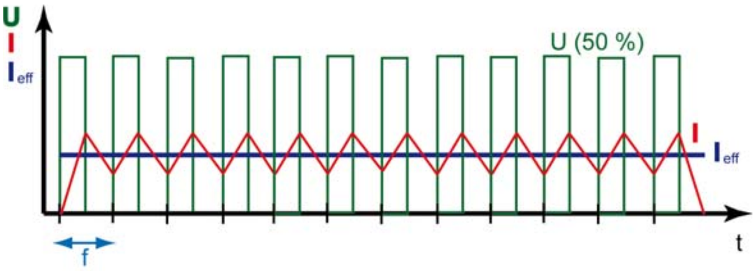


Figure: A high frequency of the PWM signal results in an almost direct current in the coil. The valve piston does not move enough any longer. With each signal change the valve piston has to overcome the break-away torque again.

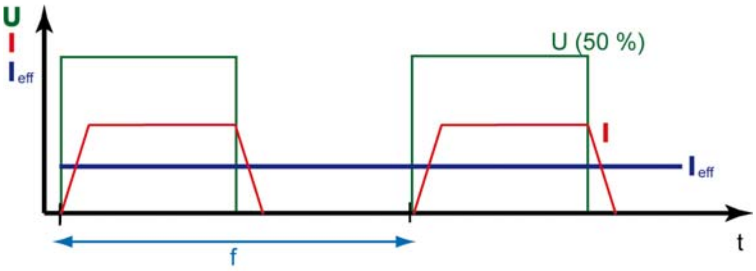


Figure: Too low frequencies of the PWM signal only allow rare, jerking movements of the valve piston. Each pulse moves the valve piston again from its off position; every time the valve piston has to overcome the break-away torque again.

NOTE

With a switch-on time below 10 % and above 90 % the dither does not have any measurable effect any longer. In such cases it makes sense and it is necessary to superimpose the PWM signal with a dither signal.

Dither frequency and amplitude

1566

The mark/space ratio (the switch-on time) of the PWM output signal is switched with the dither frequency. The dither amplitude determines the difference of the switch-on times in the two dither half-waves.

NOTE

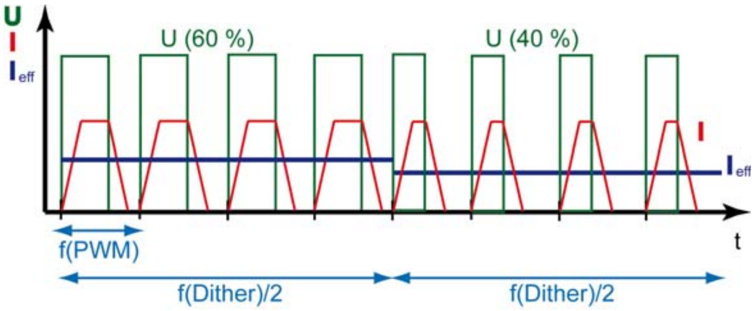
The dither frequency must be an integer part of the PWM frequency. Otherwise the hydraulic system would not work evenly but it would oscillate.

Example Dither

1567

The dither frequency is 1/8 of the PWM frequency.
The dither amplitude is 10 %.

With the switch-on time of 50 % in the figure, the actual switch-on time for 4 pulses is 60 % and for the next 4 pulses it is 40 % which means an average of 50 % switch-on time. The resulting effective coil current will be 50 % of the maximum coil current.



The result is that the valve piston always oscillates around its off position to be ready to take a new position with the next signal change without having to overcome the break-away torque before.

Functions of the library ifm_HYDRAULIC_CR0303

Contents

JOYSTICK_0 232
JOYSTICK_1 235
JOYSTICK_2 239
NORM_HYDRAULIC 242

9986

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

- JOYSTICK_0
- JOYSTICK_1
- JOYSTICK_2
- NORM_HYDRAULIC

The following FBs are needed from the library `UTIL.Lib` (in the CoDeSys package):

- RAMP_INT
- CHARCURVE

These FBs are automatically activated by the FBs of `ifm_HYDRAULIC_CR0303_Vxxyyzz.Lib` and configured.

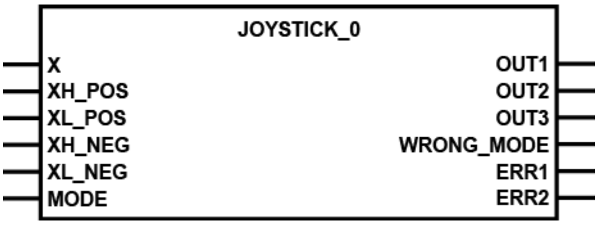
JOYSTICK_0

6250

Unit type = function block (FB)

Contained in the library:	Available for the following devices:
ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	- ClassicController: CR0020, CR0505 - ExtendedController: CR0200 - SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 - SmartController: CR25nn
`ifm_hydraulic_32bit_Vxxyyzz.Lib`	- ClassicController: CR0032 - ExtendedController: CR0232
`ifm_hydraulic_CR0303_Vxxyyzz.Lib`	- CabinetController: CR0303

Symbol in CoDeSys:



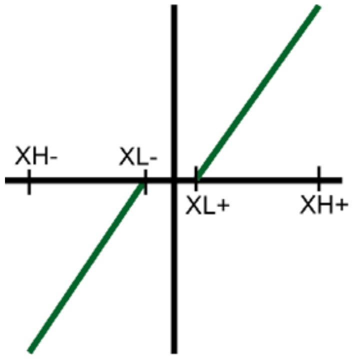
Description

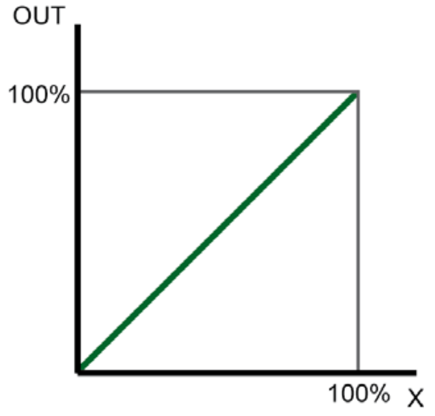
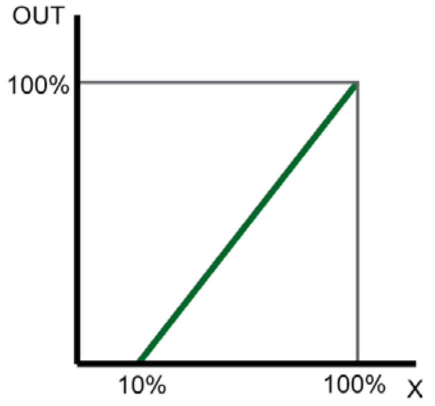
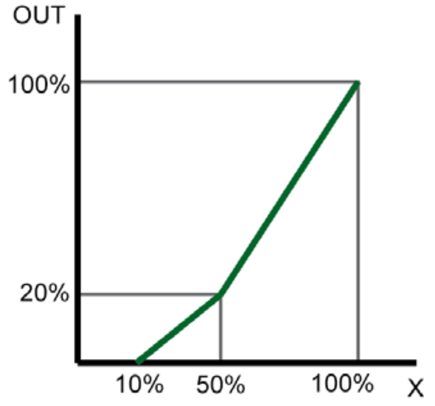
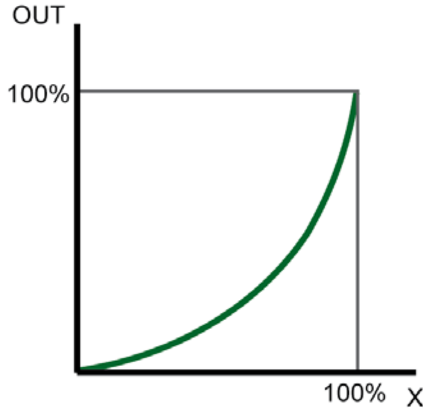
432

JOYSTICK_0 scales signals from a joystick to clearly defined characteristic curves, standardised to 0...1000.

For this FB the characteristic curve values are specified (→ figures):

- Rising edge of the ramp = 5 increments/PLC cycle
- Falling edge of the ramp = no edge

The parameters XL_POS (XL+), XH_POS (XH+), XL_NEG (XL-) and XH_NEG (XH-) are used to evaluate the joystick movements only in the requested area. The values for the positive and negative area may be different. The values for XL_NEG and XH_NEG are negative here.	

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

Parameters of the inputs

433

Parameter	Data type	Description
X	INT	preset value input in [increments]
XH_POS	INT	max. preset value positive direction in [increments] (negative values also permissible)
XL_POS	INT	min. preset value positive direction in [increments] (negative values also permissible)
XH_NEG	INT	max. preset value negative direction in [increments] (negative values also permissible)
XL_NEG	INT	min. preset value negative direction in [increments] (negative values also permissible)
MODE	BYTE	mode selection characteristic curve: 0 = linear (0|0 – 1000|1000) 1 = linear with dead band (0|0 – 100|0 – 1000|1000) 2 = 2-step linear with dead band (0|0 – 100|0 – 500|200 – 1000|1000) 3 = curve rising

Parameters of the outputs

6252

Parameter	Data type	Description
OUT1	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve left
OUT2	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve right
OUT3	INT	standardised output value pairs of values 0 to 10 [increments] e.g. for valve on output module (e.g. CR2011 or CR2031)
WRONG_MODE	BOOL	error: invalid mode
ERR1	BYTE	error code for rising edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array
ERR2	BYTE	error code for falling edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array

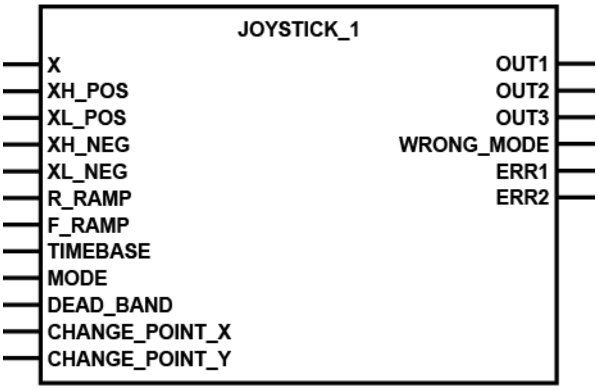
JOYSTICK_1

6255

Unit type = function block (FB)

Contained in the library:	Available for the following devices:
ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	- ClassicController: CR0020, CR0505 - ExtendedController: CR0200 - SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 - SmartController: CR25nn
`ifm_hydraulic_32bit_Vxxyyzz.Lib`	- ClassicController: CR0032 - ExtendedController: CR0232
`ifm_hydraulic_CR0303_Vxxyyzz.Lib`	- CabinetController: CR0303

Symbol in CoDeSys:

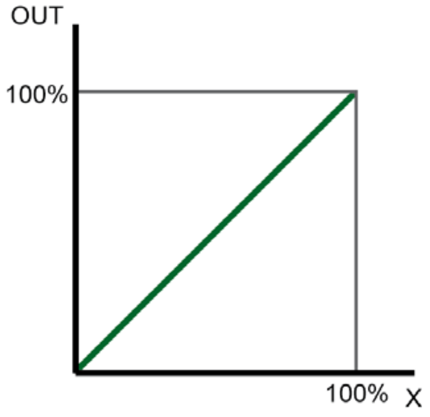


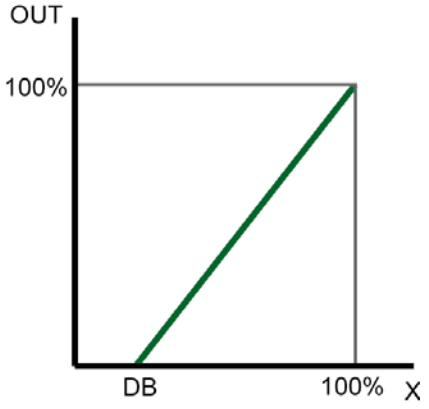
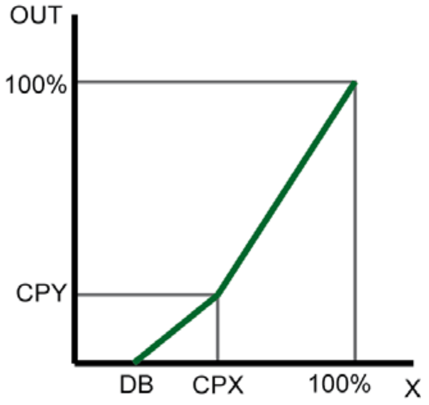
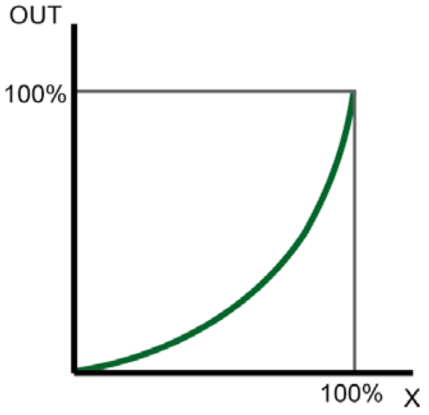
Description

425

JOYSTICK_1 scales signals from a joystick to configurable characteristic curves, standardised to 0...1000.

For this FB the characteristic curve values can be configured (→ figures):

Mode 0: Linear characteristic curve 100 % = 1000 increments	

Mode 1: Characteristic curve linear with dead band Value for the dead band (DB) can be set in % of 1000 increments 100 % = 1000 increments DB = Dead_Band	
Mode 2: 2-step linear characteristic curve with dead band Values can be configured to: Dead band: 0…DB in % of 1000 increments Step: X = CPX in % of 1000 increments Y= CPY in % of 1000 increments 100 % = 1000 increments DB = Dead_Band CPX = Change_Point_X CPY = Change_Point_Y	
Characteristic curve mode 3: Curve rising (line is fixed)	

Parameters of the inputs

6256

Parameter	Data type	Description
X	INT	preset value input in [increments]
XH_POS	INT	max. preset value positive direction in [increments] (negative values also permissible)
XL_POS	INT	min. preset value positive direction in [increments] (negative values also permissible)
XH_NEG	INT	max. preset value negative direction in [increments] (negative values also permissible)
XL_NEG	INT	min. preset value negative direction in [increments] (negative values also permissible)
R_RAMP	INT	rising edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
F_RAMP	INT	falling edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
TIMEBASE	TIME	reference for rising / falling edge of the ramp: t#0s = rising / falling edge in [increments/PLC cycle] else = rising / falling edge in [increments/TIMEBASE]
MODE	BYTE	mode selection characteristic curve: 0 = linear (0|0 – 1000|1000) 1 = linear with dead band (DB) (0|0 – DB…|0 – 1000|1000) 2 = 2-step linear with dead band (DB) (0|0 – DB|0 – CPX|CPY – 1000|1000) 3 = curve rising
DEAD_BAND	BYTE	adjustable dead band (DB) in [% of 1000 increments]
CHANGE_POINT_X	BYTE	for mode 2: ramp step, value for X in [% of 1000 increments]
CHANGE_POINT_Y	BYTE	for mode 2: ramp step, value for Y in [% of 1000 increments]

Parameters of the outputs

6252

Parameter	Data type	Description
OUT1	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve left
OUT2	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve right
OUT3	INT	standardised output value pairs of values 0 to 10 [increments] e.g. for valve on output module (e.g. CR2011 or CR2031)
WRONG_MODE	BOOL	error: invalid mode
ERR1	BYTE	error code for rising edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array
ERR2	BYTE	error code for falling edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array

JOYSTICK_2

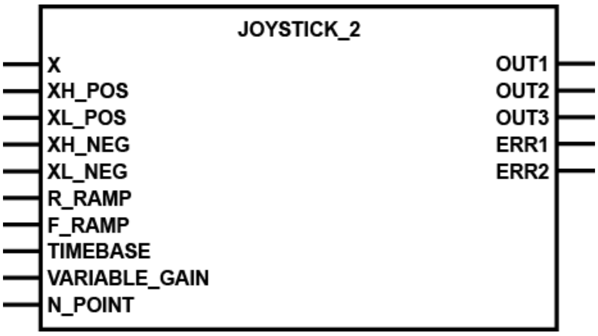
6258

6255

Unit type = function block (FB)

Contained in the library:	Available for the following devices:
ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	- ClassicController: CR0020, CR0505 - ExtendedController: CR0200 - SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 - SmartController: CR25nn
`ifm_hydraulic_32bit_Vxxyyzz.Lib`	- ClassicController: CR0032 - ExtendedController: CR0232
`ifm_hydraulic_CR0303_Vxxyyzz.Lib`	- CabinetController: CR0303

Symbol in CoDeSys:

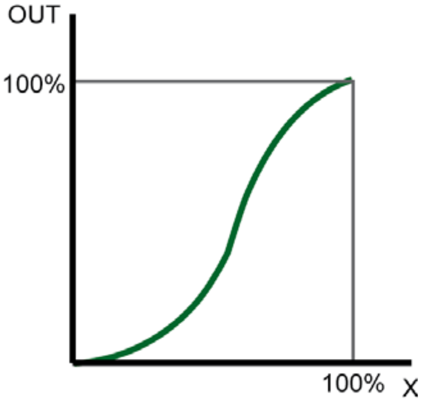


Description

418

JOYSTICK_2 scales the signals from a joystick to a configurable characteristic curve. Free selection of the standardisation.

For this FB, the characteristic curve is freely configurable (→ figure):

Characteristic curve freely configurable	

Parameters of the inputs

6261

Parameter	Data type	Description
X	INT	preset value input in [increments]
XH_POS	INT	max. preset value positive direction in [increments] (negative values also permissible)
XL_POS	INT	min. preset value positive direction in [increments] (negative values also permissible)
XH_NEG	INT	max. preset value negative direction in [increments] (negative values also permissible)
XL_NEG	INT	min. preset value negative direction in [increments] (negative values also permissible)
R_RAMP	INT	rising edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
F_RAMP	INT	falling edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE] 0 = without ramp
TIMEBASE	TIME	reference for rising and falling edge of the ramp: t#0s = rising / falling edge in [increments/PLC cycle] else = rising / falling edge in [increments/TIMEBASE]
VARIABLE_GAIN	ARRAY [0..10] OF POINT	pairs of values describing the curve the first pairs of values indicated in N_POINT are used. N = 2…11 example: 9 pairs of values declared as variable VALUES: VALUES: ARRAY[0..10] OF POINT := (X:=0,Y:=0),(X:=200,Y:=0), (X:=300,Y:=50), (X:=400,Y:=100), (X:=700,Y:=500), (X:=1000,Y:=900), (X:=1100,Y:=950), (X:=1200,Y:=1000), (X:=1400,Y:=1050); There may be blanks between the values.
N_POINT	BYTE	number of points (pairs of values in VARIABLE_GAIN) by which the curve characteristic is defined. N = 2…11

Parameters of the outputs

420

Parameter	Data type	Description
OUT1	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve left
OUT2	WORD	standardised output value pairs of values 0 to 10 [increments] e.g. for valve right
OUT3	INT	standardised output value pairs of values 0 to 10 [increments] e.g. for valve on output module (e.g. CR2011 or CR2031)
ERR1	BYTE	error code for rising edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array
ERR2	BYTE	error code for falling edge: 0 = no error 1 = error in array: wrong sequence 2 = initial value IN not contained in value range of array 4 = invalid number N for array

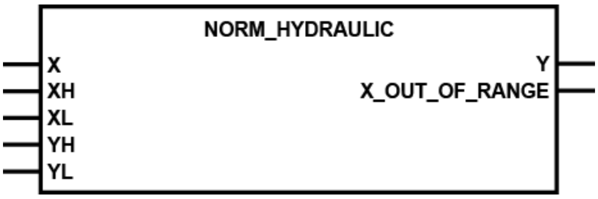
NORM_HYDRAULIC

394

Unit type = function block (FB)

Contained in the library:	Available for the following devices:
ifm_hydraulic_16bitOS04_Vxxyyzz.Lib ifm_hydraulic_16bitOS05_Vxxyyzz.Lib	- ClassicController: CR0020, CR0505 - ExtendedController: CR0200 - SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 - SmartController: CR25nn
`ifm_hydraulic_32bit_Vxxyyzz.Lib`	- ClassicController: CR0032 - ExtendedController: CR0232
`ifm_hydraulic_CR0303_Vxxyyzz.Lib`	- CabinetController: CR0303

Symbol in CoDeSys:



Description

397

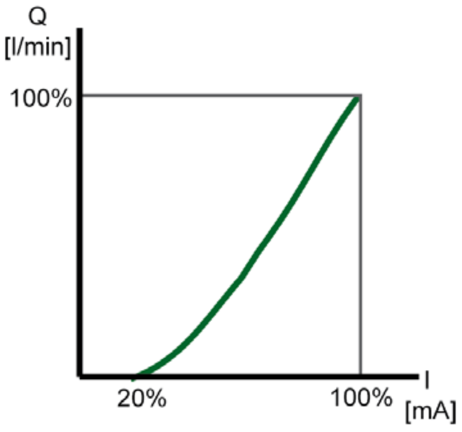
NORM_HYDRAULIC standardises input values with fixed limits to values with new limits.

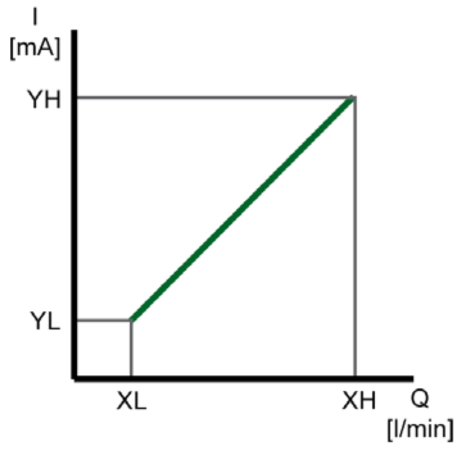
Please note: This FB corresponds to the 3S FB NORM_DINT from the CoDeSys library `UTIL.Lib`.

The FB standardises a value of type DINT within the limits of XH and XL to an output value within the limits of YH and YL.

Due to rounding errors deviations from the standardised value of 1 may occur. If the limits (XH/XL or YH/YL) are indicated in inversed form, standardisation is also inverted.

If X outside the limits XL…XH, the error message X_OUT_OF_RANGE = TRUE.

Typical characteristic curve of a hydraulic valve: The oil flow will not start before 20% of the coil current has been reached. At first the oil flow is not linear.	

Characteristics of the FB	

Parameters of the inputs

398

Parameter	Data type	Description
X	DINT	desired value input
XH	DINT	max. input value [increments]
XL	DINT	min. input value [increments]
YH	DINT	max. output value [increments], e.g.: valve current [mA] / flow [l/min]
YL	DINT	min. output value [increments], e.g.: valve current [mA] / flow [l/min]

Parameters of the outputs

399

Parameter	Data type	Description
Y	DINT	standardised output value
X_OUT_OF_RANGE	BOOL	error: X is beyond the limits XH and XL

Example: NORM_HYDRAULIC

400

Parameter	Case 1	Case 2	Case 3
Upper limit value input XH	100	100	2000
Lower limit value input XL	0	0	0
Upper limit value output YH	2000	0	100
Lower limit value output YL	0	2000	0
Non standardised value X	20	20	20
Standardised value Y	400	1600	1

Case 1:
Input with relatively coarse resolution.
Output with high resolution.
1 X increment results in 20 Y increments.

Case 2:
Input with relatively coarse resolution.
Output with high resolution.
1 X increment results in 20 Y increments.
Output signal is inverted as compared to the input signal.

Case 3:
Input with high resolution.
Output with relatively coarse resolution.
20 X increments result in 1 Y increment.

9.5 Controller functions

Contents

General 245
Setting rule for a controller 247
Functions for controllers 248

1622

9.5.1 General

1623

Controlling is a process during which the unit to be controlled (control variable x) is continuously detected and compared with the reference variable w. Depending on the result of this comparison, the control variable is influenced for adaptation to the reference variable.

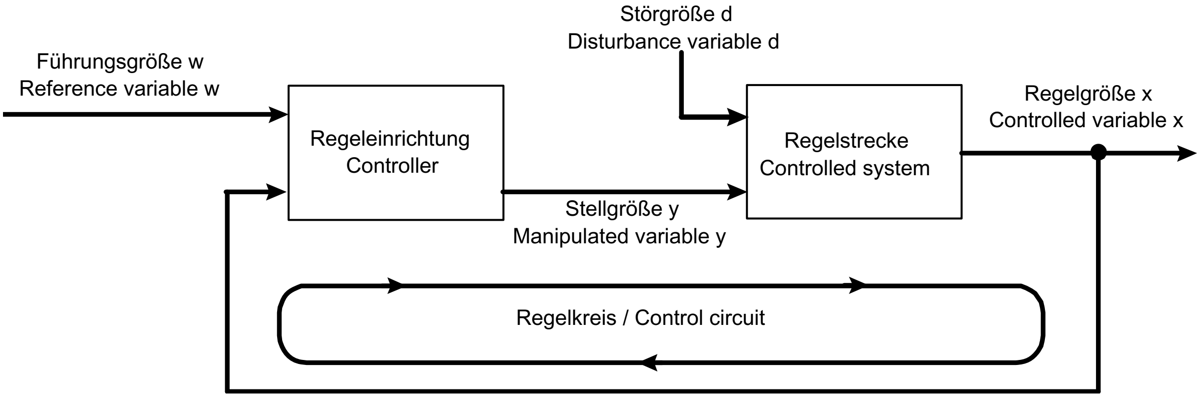


Figure: Principle of controlling

The selection of a suitable control device and its optimum setting require exact indication of the steady-state behaviour and the dynamic behaviour of the controlled system. In most cases these characteristic values can only be determined by experiments and can hardly be influenced.

Three types of controlled systems can be distinguished:

Self-regulating process

1624

For a self-regulating process the control variable x goes towards a new final value after a certain manipulated variable (steady state). The decisive factor for these controlled systems is the amplification (steady-state transfer factor KS). The smaller the amplification, the better the system can be controlled. These controlled systems are referred to as P systems (P = proportional).

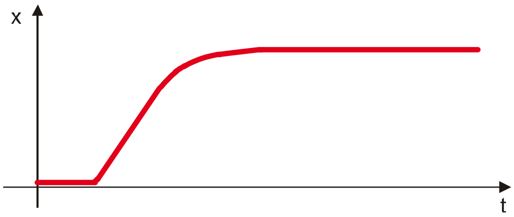


Figure: P controller = self-regulating process

Controlled system without inherent regulation

1625

Controlled systems with an amplifying factor towards infinity are referred to as controlled systems without inherent regulation. This is usually due to an integrating performance. The consequence is that the control variable increases constantly after the manipulated variable has been changed or by the influence of an interfering factor. Due to this behaviour it never reaches a final value. These controlled systems are referred to as I systems (I = integral).

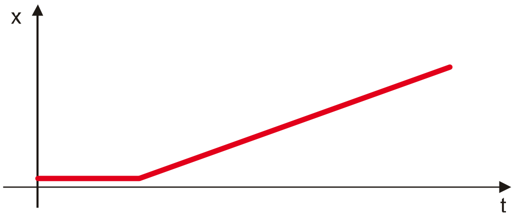


Figure: I controller = controlled system without inherent regulation

Controlled system with delay

1626

Most controlled systems correspond to series systems of P systems (systems with compensation) and one or several T1 systems (systems with inertia). A controlled system of the 1st order is for example made up of the series connection of a throttle point and a subsequent memory.

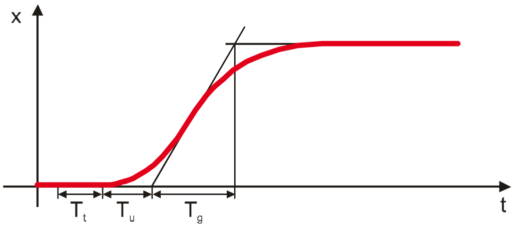


Figure: PT system = controlled system with delay

For controlled systems with dead time the control variable does not react to a change of the control variable before the dead time T_t has elapsed. The dead time T_t or the sum of $T_t + T_u$ relates to the controllability of the system. The controllability of a system is the better, the greater the ratio T_g/T_u.

The controllers which are integrated in the library are a summary of the preceding basic functions. It depends on the respective controlled system which functions are used and how they are combined.

9.5.2 Setting rule for a controller

1627

For controlled systems, whose time constants are unknown the setting procedure to Ziegler and Nickols in a closed control loop is of advantage.

Setting control

1628

At the beginning the controlling system is operated as a purely P-controlling system. In this respect the derivative time T_V is set to 0 and the reset time T_N to a very high value (ideally to ∞) for a slow system. For a fast controlled system a small T_N should be selected.

Afterwards the gain 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 **d**ifferential 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	$1.7 * KP_{critical}$	$0.50 * T_{critical}$	$0.125 * T_{critical}$

NOTE

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

Damping of overshoot

1629

To dampen overshoot PT1 (→ page 251) (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 PID or GLR controller).

9.5.3 Functions for controllers

Contents

DELAY 249
PT1 251
PID1 252
PID2 254
GLR 257

1634

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

DELAY

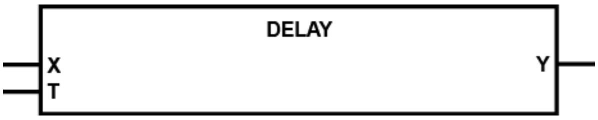
585

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

588

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

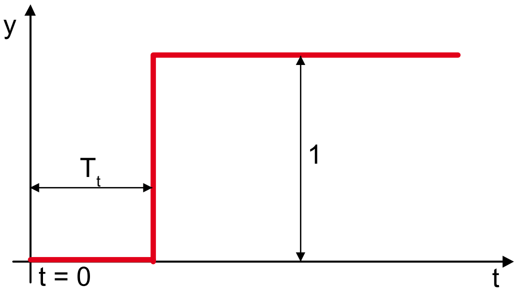


Figure: Time characteristics of DELAY

ⓘ NOTE

To ensure that the FB works correctly, it must be called in each cycle.

Parameters of the inputs

589

Parameter	Data type	Description
X	WORD	input value
T	TIME	time delay (dead time)

Parameters of the outputs

590

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

PT1

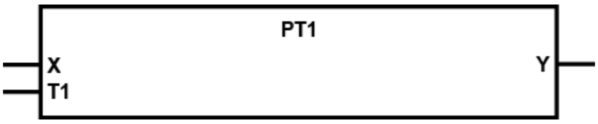
338

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

341

PT1 handles a controlled system with a first-order time delay.

This FB is a proportional controlled system with a time delay. It is for example used for generating ramps when using the PWM FBs.

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

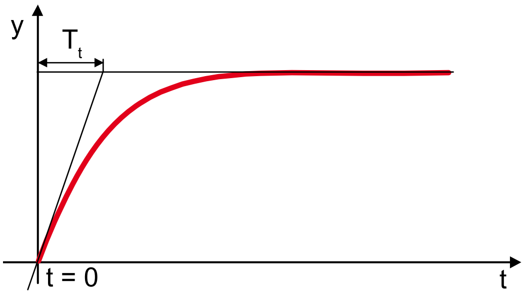


Figure: Time characteristics of PT1

Parameters of the inputs

342

Parameter	Data type	Description
X	INT	input value
T1	TIME	delay time (time constant)

Parameters of the outputs

343

Parameter	Data type	Description
Y	INT	output variable

PID1

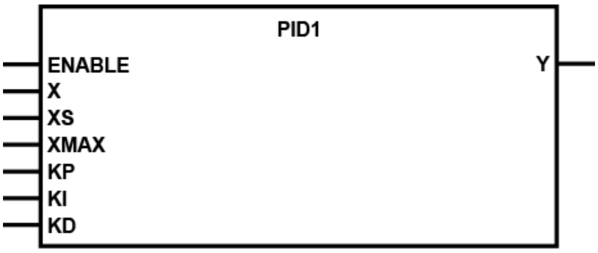
351

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

354

PID1 handles a PID controller.

The change of the manipulated variable of a PID controller has a **p**roportional, **i**ntegral and **d**ifferential component. The manipulated variable changes first by an amount which depends on the rate of change of the input value (D component). After the end of the derivative action time the manipulated variable returns to the value corresponding to the proportional range and changes in accordance with the reset time.

❗ NOTE

The manipulated variable Y is already standardised to the PWM FB (RELOAD value = 65,535). Note the reverse logic:
65,535 = minimum value
0 = maximum value.

Note that the input values KI and KD depend on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

If X > XS, the manipulated variable is increased.
If X < XS, the manipulated variable is reduced.

The manipulated variable Y has the following time characteristics:

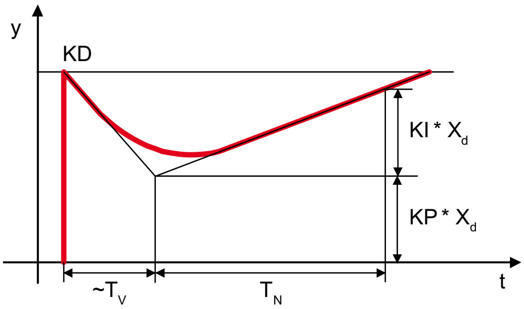


Figure: Typical step response of a PID controller

Parameters of the inputs

355

Parameter	Data type	Description
X	WORD	actual value
XS	WORD	desired value
XMAX	WORD	maximum value of the target value
KP	BYTE	constant of the proportional component
KI	BYTE	integral value
KD	BYTE	proportional component of the differential component

Parameters of the outputs

356

Parameter	Data type	Description
Y	WORD	manipulated variable

Recommended settings

357

KP = 50
KI = 30
KD = 5

With the values indicated above the controller operates very quickly and in a stable way. The controller does not fluctuate with this setting.

► To optimise the controller, the values can be gradually changed afterwards.

PID2

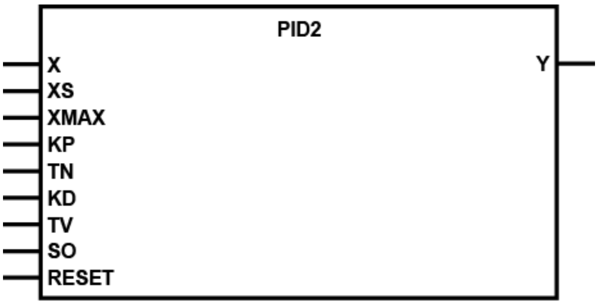
9167

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

347

PID2 handles a PID controller with self optimisation.

The change of the manipulated variable of a PID controller has a **p**roportional, **i**ntegral and **d**ifferential component. The manipulated variable changes first by an amount which depends on the rate of change of the input value (D component). After the end of the derivative action time TV the manipulated variable returns to the value corresponding to the proportional component and changes in accordance with the reset time TN.

The values entered at the inputs KP and KD are internally divided by 10. So, a finer grading can be obtained (e.g.: KP = 17, which corresponds to 1.7).

NOTE

The manipulated variable Y is already standardised to the PWM FB (RELOAD value = 65,535). Note the reverse logic:
65,535 = minimum value
0 = maximum value.

Note that the input value KD depends on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

If X > XS, the manipulated variable is increased.
If X < XS, the manipulated variable is reduced.

A reference variable is internally added to the manipulated variable.
Y = Y + 65,536 – (XS / XMAX * 65,536).

The manipulated variable Y has the following time characteristics.

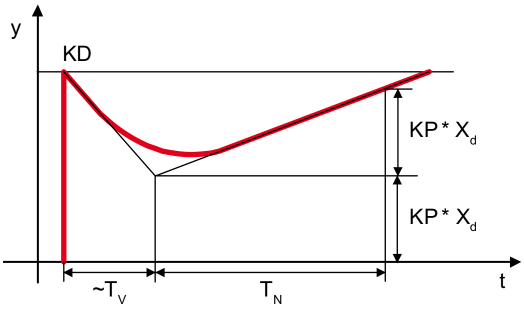


Figure: Typical step response of a PID controller

Parameters of the inputs

348

Parameter	Data type	Description
X	WORD	actual value
XS	WORD	desired value
XMAX	WORD	maximum value of the desired value
KP	BYTE	constant of the proportional component (/10)
TN	TIME	reset time (integral component)
KD	BYTE	proportional component of the differential component (/10)
TV	TIME	derivative action time (differential component)
SO	BOOL	self optimisation
RESET	BOOL	Reset

Parameters of the outputs

349

Parameter	Data type	Description
Y	WORD	manipulated variable

Recommended setting

9127
350

- ► Select TN according to the time characteristics of the system:
  fast system = small TN
  slow system = large TN
- ► Slowly increment KP gradually, up to a value at which still definitely no fluctuation will occur.
- ► Readjust TN if necessary.
- ► Add **d**ifferential component only if necessary:
  Select a TV value approx. 2...10 times smaller than TN.
  Select a KD value more or less similar to KP.

Note that the maximum control deviation is + 127. For good control characteristics this range should not be exceeded, but it should be exploited to the best possible extent.

Function input SO (self-optimisation) clearly improves the control performance. A precondition for achieving the desired characteristics:

- The controller is operated with **I** component (TN > 50 ms)
- Parameters KP and especially TN are already well adjusted to the actual controlled system.
- The control range (X – XS) of ± 127 is utilised (if necessary, increase the control range by multiplying X, XS and XMAX).

► When you have finished setting the parameters, you can set SO = TRUE.

> This will significantly improve the control performance, especially reducing overshoot.

GLR

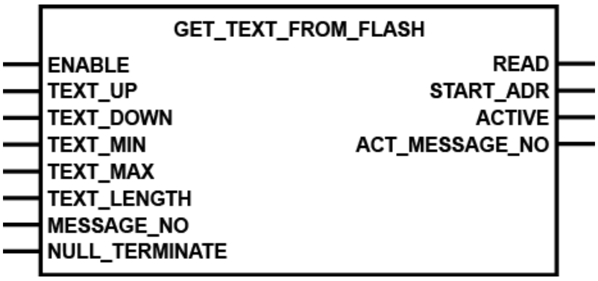
531

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



Description

534

GLR handles a synchro controller.

The synchro controller is a controller with PID characteristics.

The values entered at the inputs KP and KD are internally divided by 10. So, a finer grading can be obtained (e.g.: KP = 17, which corresponds to 1.7).

The manipulated variable referred to the greater actual value is increased accordingly.
The manipulated variable referred to the smaller actual value corresponds to the reference variable.
Reference variable = 65 536 – (XS / XMAX * 65 536).

NOTE

The manipulated variables Y1 and Y2 are already standardised to the PWM FB
(RELOAD value = 65 535). Note the reverse logic:
65 535 = minimum value
0 = maximum value.

Note that the input value KD depends on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

Parameters of the inputs

535

Parameter	Data type	Description
X1	WORD	actual value channel 1
X2	WORD	actual value channel 2
XS	WORD	desired value = reference variable
XMAX	WORD	maximum value of the desired value
KP	BYTE	constant of the proportional component (/10)
TN	TIME	reset time (integral component)
KD	BYTE	proportional component of the differential component (/10)
TV	TIME	derivative action time (differential component)
RESET	BOOL	Reset

Parameters of the outputs

536

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

10 Communication via interfaces

Contents

Use of the serial interface 259

8602

Here we show you functions to use for communication via interfaces.

10.1 Use of the serial interface

Contents

SERIAL_SETUP.. 260
SERIAL_TX.. 262
SERIAL_RX.. 263
SERIAL_PENDING.. 265

1600

NOTE

In principle, the serial interface is not available for the user because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.
For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

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

10.1.1 SERIAL_SETUP

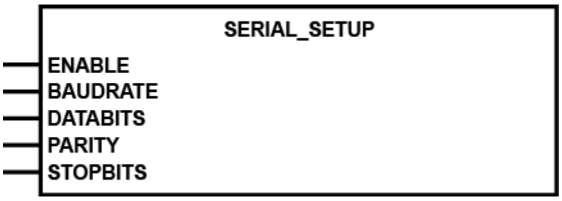
302

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

305

SERIAL_SETUP initialises the serial RS232 interface.

SERIAL_SETUP sets the serial interface to the indicated parameters. Using the input ENABLE, the FB is activated for one cycle.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.
For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

ATTENTION

The driver module of the serial interface can be damaged!

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

► Do not disconnect the serial interface while live.

Parameters of the inputs

306

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only 1 cycle): interface is initialised FALSE: during further processing of the program
BAUDRATE	WORD	baud rate (permissible values = 9 600, 19 200, 28 800, (57 600)) preset value → data sheet
DATABITS	BYTE	data bits (permissible values: 7 or 8) preset value = 8
PARITY	BYTE	parity (permissible values: 0=none, 1=even, 2=uneven) preset value = 0
STOPBITS	BYTE	stop bits (permissible values: 1 or 2) preset value = 1

10.1.2 SERIAL_TX

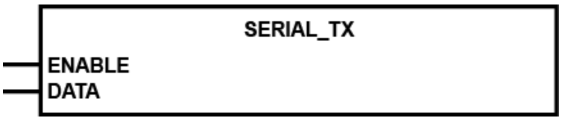
296

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

299

SERIAL_TX transmits one data byte via the serial RS232 interface.

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

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.
For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

Parameters of the inputs

300

Parameter	Data type	Description
ENABLE	BOOL	TRUE: transmission enabled FALSE: transmission blocked
DATA	BYTE	byte to be transmitted

10.1.3 SERIAL_RX

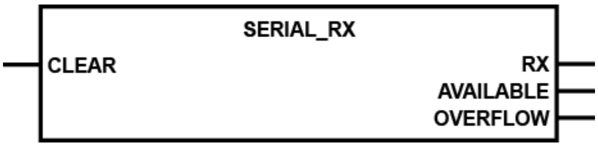
308

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

311

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

Then, the value of AVAILABLE is decremented by 1.

If more than 1000 data bytes are received, the buffer overflows and data is lost. This is indicated by the bit OVERFLOW.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.
For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

Parameters of the inputs

312

Parameter	Data type	Description
CLEAR	BOOL	TRUE: receive buffer is deleted FALSE: this function is not executed

Parameters of the outputs

313

Parameter	Data type	Description
RX	BYTE	byte data received from the receive buffer
AVAILABLE	WORD	number of data bytes received 0 = no valid data available
OVERFLOW	BOOL	TRUE: overflow of the data buffer, loss of data!

Example:

3 bytes are received:

1st call of SERIAL_RX
1 valid value at output RX
→ AVAILABLE = 3

2nd call of SERIAL_RX
1 valid value at output RX
→ AVAILABLE = 2

3rd call of SERIAL_RX
1 valid value at output RX
→ AVAILABLE = 1

4th call of SERIAL_RX
invalid value at the output RX
→ AVAILABLE = 0

If AVAILABLE = 0, the FB can be skipped during processing of the program.

10.1.4 SERIAL_PENDING

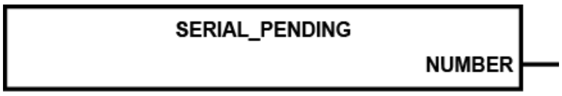
314

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

317

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

In contrast to SERIAL_RX (→ page 263) the contents of the buffer remain unchanged after calling this FB.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.
For CRnn32: Debugging of the application software is then only possible via all 4 CAN interfaces or via USB.

Parameters of the outputs

319

Parameter	Data type	Description
NUMBER	WORD	number of data bytes received

11 Managing the data

Contents

Software reset 266
Reading / writing the system time 268
Saving, reading and converting data in the memory........ 271
Data access and data check 282

8606

Here we show you functions how to read or manage data in the device.

11.1 Software reset

Contents

SOFTRESET........ 267

1594

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

11.1.1 SOFTRESET

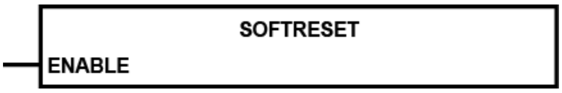
260

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

263

SOFTRESET leads to a complete reboot of the controller.

The FB can for example be used in conjunction with CANopen if a node reset is to be carried out. The behaviour of the controller after a SOFTRESET corresponds to that after switching the supply voltage off and on.

> **NOTE**
>
> In case of active communication, the long reset period must be taken into account because otherwise guarding errors will be signalled.

Parameters of the inputs

264

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active

11.2 Reading / writing the system time

Contents

TIMER_READ 269
TIMER_READ_US 270

1601

The following FBs offered by **ifm electronic** allow you to read the continually running system time of the controller and to evaluate it in the application program, or to change the system time as needed.

11.2.1 TIMER_READ

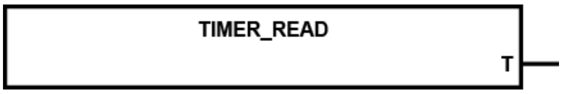
236

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

239

TIMER_READ reads the current system time.

When the supply voltage is applied, the controller generates a clock pulse which is counted upwards in a register. This register can be read using the FB call and can for example be used for time measurement.

NOTE

The system timer goes up to FFFF FFFF_{16} at the maximum (corresponds to about 49.7 days) and then starts again from 0.

Parameters of the outputs

241

Parameter	Data type	Description
T	TIME	current system time (resolution [ms])

11.2.2 TIMER_READ_US

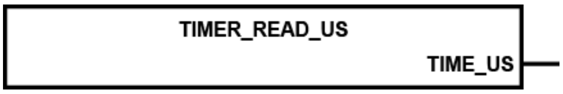
657

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

660

TIMER_READ_US reads the current system time in [µs].

When the supply voltage is applied, the device generates a clock pulse which is counted upwards in a register. This register can be read by means of the FB call and can for example be used for time measurement.

Info

The system timer runs up to the counter value 4 294 967 295 µs at the maximum and then starts again from 0.

4 294 967 295 µs = 4 295 s = 71.6 min = 1.2 h

Parameters of the outputs

662

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

11.3 Saving, reading and converting data in the memory

Contents

Automatic data backup.. 271
Manual data storage 272

1595

11.3.1 Automatic data backup

1596

The **ecomat*mobil*** devices 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 filed as RETAIN variables.

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

If the supply voltage returns, the saved data is read from the memory via the operating system and written back in the flag area.

11.3.2 Manual data storage

Contents

MEMCPY 273
FLASHWRITE 274
FLASHREAD 276
FRAMWRITE 278
FRAMREAD 280

1597

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

Depending on the device the following memories are available:

- **EEPROM memory**
  Available for the following devices:
  - CabinetController: CR0301, CR0302
  - PCB controller: CS0015
  - SmartController: CR25nn
  Slow writing and reading.
  Limited writing and reading frequency.
  Any memory area can be selected.
  Storing data with E2WRITE.
  Reading data with E2READ.

- **FRAM memory** [1])
  Available for the following devices:
  - CabinetController: CR0303
  - ClassicController: CR0020, CR0032, CR0505
  - ExtendedController: CR0200, CR0232
  - SafetyController: CR7nnn
  - PDM360smart: CR1070, CR1071
  Fast writing and reading.
  Unlimited writing and reading frequency.
  Any memory area can be selected.
  Storing data with FRAMWRITE.
  Reading data with FRAMREAD.
  [1]) FRAM indicates here all kinds of non-volatile and fast memories.

- **Flash memory**
  For all devices.
  Fast writing and reading.
  Limited writing and reading frequency.
  Really useful only for storing large data quantities.
  Before anew writing, the memory contents must be deleted.
  Storing data with FLASHWRITE.
  Reading data with FLASHREAD.

Info

By means of the storage partitioning (→ data sheet or operating instructions) the programmer can find out which memory area is available.

MEMCPY

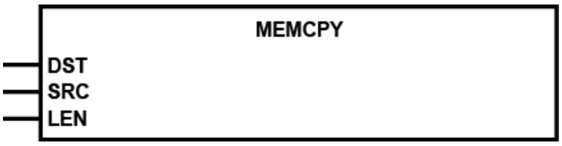
409

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

412

MEMCPY enables writing and reading different types of data directly in the memory.

The FB writes the contents of the address of SRC to the address DST. In doing so, as many bytes as indicated under LEN are transmitted. So it is also possible to transmit exactly one byte of a word file.

► The address must be determined by means of the operator ADR and assigned to the FB!

Parameters of the inputs

413

Parameter	Data type	Description
DST	DWORD	address of the target variables ► The address must be determined by means of the operator ADR and assigned to the FB!
SRC	DWORD	address of the source variables ► The address must be determined by means of the operator ADR and assigned to the FB!
LEN	WORD	number of data bytes

FLASHWRITE

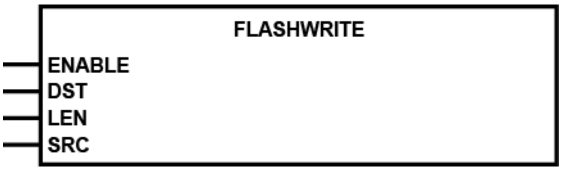
555

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

558

WARNING

Danger due to uncontrollable process operations!

The status of the inputs/outputs is "frozen" during execution of FLASHWRITE.

► Do not execute this FB when the machine is running!

FLASHWRITE enables writing of different data types directly into the flash memory.

The FB writes the contents of the address SRC into the flash memory. In doing so, as many bytes as indicated under LEN are transmitted.

► The address must be determined by means of the operator ADR and assigned to the FB!

An erasing operation must be carried out before the memory is written again. This is done by writing any content to the address "0".

Info

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

Parameters of the inputs

559

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
DST	WORD CR0301, CR0302, CS0015: INT	relative start address in the memory (→ table below)
LEN	WORD CR0301, CR0302, CS0015: INT	number of data bytes (→ table below)
SRC	DWORD CR0301, CR0302, CS0015: DINT	address of the source variables ► The address must be determined by means of the operator ADR and assigned to the FB!

Device	permissible values for DST dec | hex		permissible values for LEN dec | hex	
CabinetController: CR030n	0...16 383	0...3FFF	0...16 383	0...3FFF
ClassicController: CR0020, CR0505	0...65 535	0...FFFF	0...65 535	0...FFFF
ExtendedController: CR0200	0...65 535	0...FFFF	0...65 535	0...FFFF
PCB controller: CS0015	0...16 383	0...3FFF	0...16 383	0...3FFF
SafetyController: CR7021, CR7201, CR7506	0...65 535	0...FFFF	0...65 535	0...FFFF
SmartController: CR25nn	0...65 535	0...FFFF	0...65 535	0...FFFF
PDM360smart: CR1070, CR1071	0...16 383	0...3FFF	0...16 384	0...4000

FLASHREAD

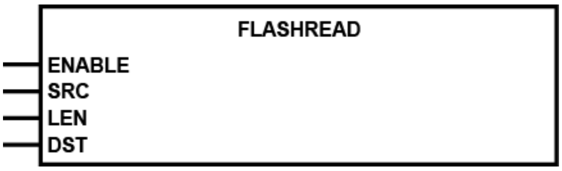
561

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

564

FLASHREAD enables reading of different types of data directly from the flash memory.

The FB reads the contents as from the address of SRC from the flash memory. In doing so, as many bytes as indicated under LEN are transmitted.

► The address must be determined by means of the operator ADR and assigned to the FB!

Parameters of the inputs

565

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
SRC	WORD CR0301, CR0302, CS0015: INT	relative start address in the memory (→ table below)
LEN	WORD CR0301, CR0302, CS0015: INT	number of data bytes (→ table below)
DST	DWORD CR0301, CR0302, CS0015: DINT	address of the target variables ► The address must be determined by means of the operator ADR and assigned to the FB!

Device	**permissible values for SRC dec | hex**		**permissible values for LEN dec | hex**	
CabinetController: CR030n	0...16 383	0...3FFF	0...16 383	0...3FFF
ClassicController: CR0020, CR0505	0...65 535	0...FFFF	0...65 535	0...FFFF
ExtendedController: CR0200	0...65 535	0...FFFF	0...65 535	0...FFFF
PCB controller: CS0015	0...16 383	0...3FFF	0...16 383	0...3FFF
SafetyController: CR7021, CR7201, CR7506	0...65 535	0...FFFF	0...65 535	0...FFFF
SmartController: CR25nn	0...16 383	0...3FFF	0...16 383	0...3FFF
PDM360smart: CR1070, CR1071	0...16 383	0...3FFF	0...16 384	0...4000

FRAMWRITE

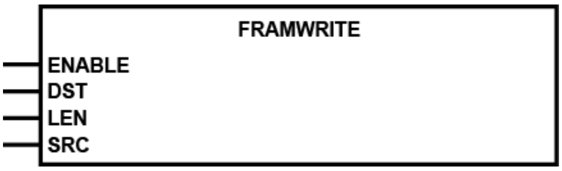
543

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

546

FRAMWRITE enables the quick writing of different data types directly into the FRAM memory 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.

► The address must be determined by means of the operator ADR and assigned to the FB!

The FRAM memory can be written in several partial segments which are independent of each other. Monitoring of the memory segments must be carried out in the application program.

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

Parameters of the inputs

547

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
DST	WORD CR0303: INT	relative start address in the memory (→ table below)
LEN	WORD CR0303: INT	number of data bytes (→ table below)
SRC	DWORD CR0303: DINT	address of the source variables ► The address must be determined by means of the operator ADR and assigned to the FB!

Device	**permissible values for SRC dec | hex**		**permissible values for LEN dec | hex**	
CabinetController: CR0303	512...2 047	200...7FF	0...128	0...80
ClassicController: CR0020, CR0505	0...1 023	0...3FF		
ExtendedController: CR0200	0...1 023	0...3FF		
SafetyController: CR7021, CR7201, CR7506	0...1 023	0...3FF		
PDM360smart: CR1070, CR1071	512...2 047	200...7FF	0...128	0...80

FRAMREAD

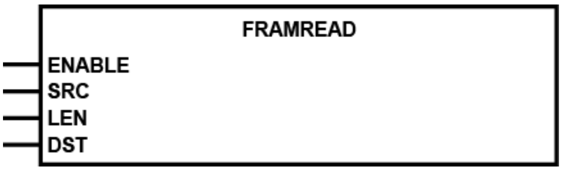
549

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR0303
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SafetyController: CR7nnn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

552

FRAMREAD enables quick reading of different data types directly from the FRAM memory 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.

► The address must be determined by means of the operator ADR and assigned to the FB!

The FRAM memory can be read in several independent partial segments. Monitoring of the memory segments must be carried out in the application program.

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

Parameters of the inputs

553

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
SRC	WORD CR0303: INT	relative start address in the memory (→ table below)
LEN	WORD CR0303: INT	number of data bytes (→ table below)
DST	DWORD CR0303: DINT	address of the target variables ► The address must be determined by means of the operator ADR and assigned to the FB!

Device	**permissible values for DST dec | hex**		**permissible values for LEN dec | hex**	
CabinetController: CR0303	0...2 047	0...7FF	0...128	0...80
ClassicController: CR0020, CR0505	0...1 023	0...3FF		
ExtendedController: CR0200	0...1 023	0...3FF		
SafetyController: CR7021, CR7201, CR7506	0...1 023	0...3FF		
PDM360smart: CR1070, CR1071	0...2 047	0...7FF	0...128	0...80

11.4 Data access and data check

Contents

SET_DEBUG 283
SET_IDENTITY 284
GET_IDENTITY 286
SET_PASSWORD 288
CHECK_DATA 290

1598

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

11.4.1 SET_DEBUG

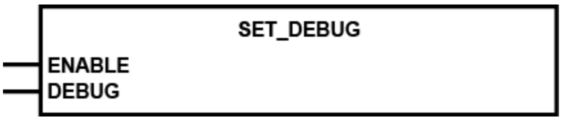
290

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn

Symbol in CoDeSys:



Description

293

SET_DEBUG handles the DEBUG mode without active test input (→ chapter TEST mode (→ page 19)).

If the input DEBUG of the FB is set to TRUE, the programming system or the downloader, for example, can communicate with the device and execute system commands (e.g. for service functions via the GSM modem CANremote).

! NOTE

In this operating mode a software download is not possible because the test input is not connected to supply voltage.

Parameters of the inputs

294

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active
DEBUG	BOOL	TRUE: debugging via the interfaces possible FALSE: debugging via the interfaces not possible

11.4.2 SET_IDENTITY

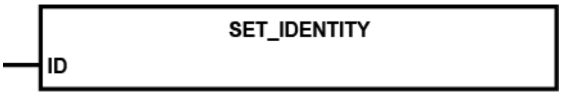
284

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



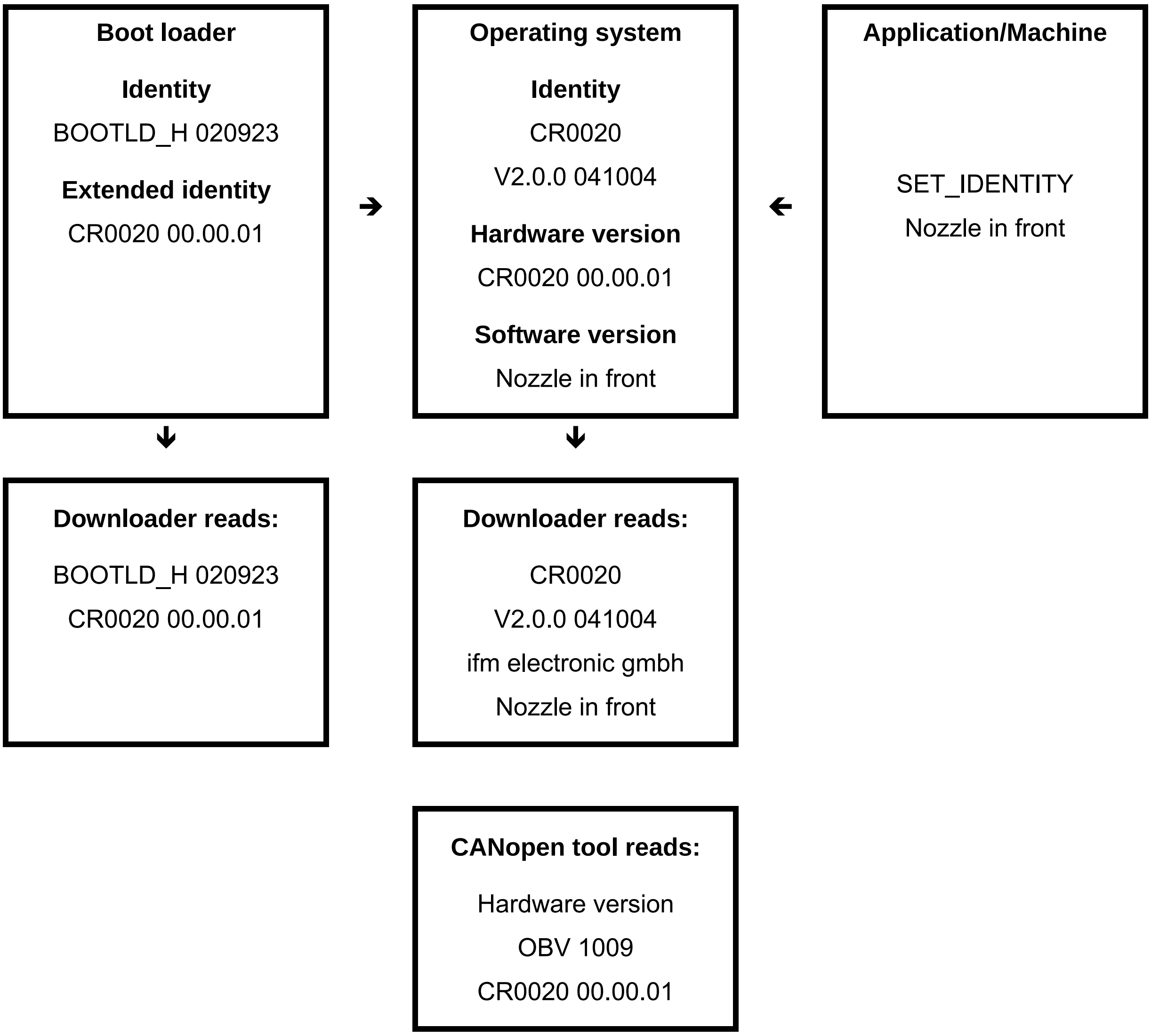
Description

287

SET_IDENTITY sets an application-specific program identification.

Using this FB, a program identification can be created by the application program. This identification (i.e. the software version) can be read via the software tool DOWNLOADER.EXE in order to identify the loaded program.

The following figure shows the correlations of the different identifications as indicated by the different software tools. (Example: ClassicController CR0020):



Parameters of the inputs

288

Parameter	Data type	Description
ID	STRING(80)	any string with a maximum length of 80 characters

11.4.3 GET_IDENTITY

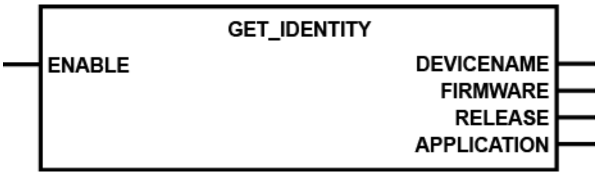
2212

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

2344

GET_IDENTITY reads the application-specific program identification stored in the controller.

With this FB the stored program identification can be read by the application program. The following information is available:

- Hardware name and version
  e.g.: "CR0032 00.00.01"
- Name of the runtime system
  e.g.: "CR0032"
- Version and build of the runtime system
  e.g.: "V00.00.01 071128"
- Name of the application
  e.g.: "Crane1704"

The name of the application can be changed with SET_IDENTITY (→ page 284).

Parameters of the inputs

2609

Parameter	Data type	Description
ENABLE	BOOL	TRUE: unit is executed FALSE: unit is not executed > POU inputs and outputs are not active

Parameters of the outputs

2610

Parameter	Data type	Description
DEVICENAME	STRING(31)	hardware name and version as string of max. 31 characters e.g.: "CR0032 00.00.01"
FIRMWARE	STRING(31)	name of the runtime system as string of max. 31 characters e.g.: "CR0032"
RELEASE	STRING(31)	version and build of the runtime system as string of max. 31 characters e.g.: "V00.00.01 071128"
APPLICATION	STRING(79)	name of the application as string of max. 79 characters e.g.: "Crane1704"

11.4.4 SET_PASSWORD

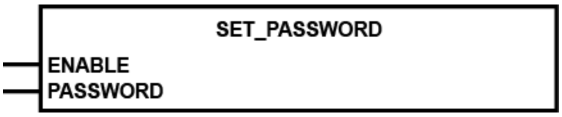
266

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

269

SET_PASSWORD sets a user password for the program and memory upload with the DOWNLOADER.

If the password is activated, reading of the application program or the data memory with the software tool DOWNLOADER is only possible if the correct password has been entered.

If an empty string (default condition) is assigned to the input PASSWORD, an upload of the application software or of the data memory is possible at any time.

ATTENTION

Please note for CR250n, CR0301, CR0302 and CS0015:

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

- ► Only carry out the unit **once** during initialisation in the first program cycle!
- ► Afterwards block the unit again with ENABLE = FALSE!

NOTE

The password is reset when loading a new application program.

Parameters of the inputs

270

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only 1 cycle): ID set FALSE: unit is not executed
PASSWORD	STRING(16)	password (maximum string length 16)

11.4.5 CHECK_DATA

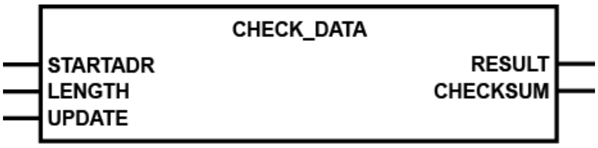
603

Unit type = function block (FB)

Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SafetyController: CR7nnn
- SmartController: CR25nn
- PDM360smart: CR1070, CR1071

Symbol in CoDeSys:



Description

606

CHECK_DATA stores the data in the application data memory via a CRC code.

The FB serves for monitoring a range of the data memory (possible WORD addresses as from %MW0) for unintended changes to data in safety-critical applications. To do so, the FB determines a CRC checksum of the indicated data range.

► The address must be determined by means of the operator ADR and assigned to the FB!

► In addition, the number of data bytes LENGTH (length as from the STARTDR) must be indicated.

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

Data changes in the memory (e.g. by the application program or **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.

NOTE

This FB is a safety function. However, the controller does not automatically become a safety controller by using this FB. Only a tested and approved controller with a special operating system can be used as safety controller.

Parameters of the inputs

607

Parameter	Data type	Description
STARTADR	DINT	start address of the monitored data memory (WORD address as from %MW0)
LENGTH	WORD	length of the monitored data memory in [byte]
UPDATE	BOOL	TRUE: changes to data permissible FALSE: changes to data not permitted

Parameters of the outputs

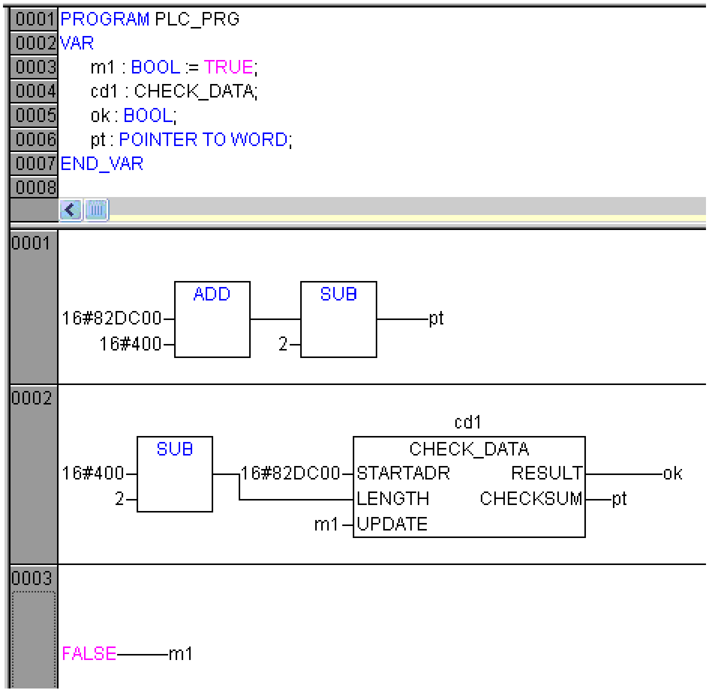
608

Parameter	Data type	Description
RESULT	BOOL	TRUE: CRC checksum ok FALSE: CRC checksum faulty (data modified)
CHECKSUM	WORD	result of the CRC checksum evaluation

Example: CHECK_DATA

4168

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



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

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

12 Optimising the PLC cycle

Contents

Processing interrupts... 292

8609

Here we show you functions to optimise the PLC cycle.

12.1 Processing interrupts

Contents

SET_INTERRUPT_XMS.. 293
SET_INTERRUPT_I... 296

1599

The PLC cyclically processes the stored application program in its full length. The cycle time can vary due to program branchings which depend e.g. on external events (= conditional jumps). This can have negative effects on certain functions.

By means of systematic interrupts of the cyclic program it is possible to call time-critical processes independently of the cycle in fixed time periods or in case of certain events.

Since interrupt functions are principally not permitted for SafetyControllers, they are thus not available.

12.1.1 SET_INTERRUPT_XMS

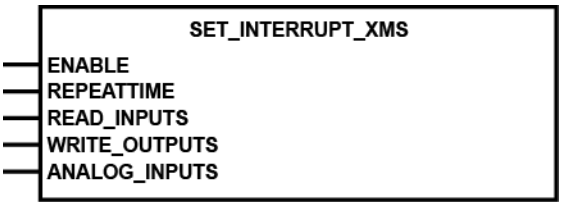
272

Unit type = function block (FB)

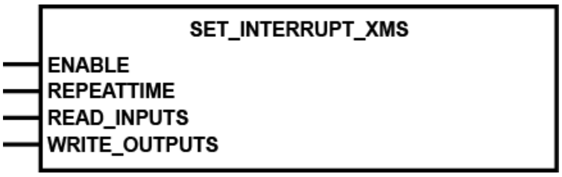
Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- PCB controller: CS0015
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



(only for devices with analogue channels)



(for devices without analogue channels)

Description

275

SET_INTERRUPT_XMS handles the execution of a program part at an interval of x ms.

In the conventional PLC the cycle time is decisive for real-time monitoring. So, the PLC is at a disadvantage as compared to customer-specific controllers. Even a "real-time operating system" does not change this fact when the whole application program runs in one single block which cannot be changed.

A possible solution would be to keep the cycle time as short as possible. This often leads to splitting the application up to several control cycles. This, however, makes programming complex and difficult.

Another possibility is to call a certain program part at fixed intervals (every x ms) independently of the control cycle.

The time-critical part of the application is integrated by the user in a block of the type PROGRAM (PRG). This block is declared as the interrupt routine by calling SET_INTERRUPT_XMS once (during initialisation). As a consequence, this program block is always processed after the REPEATTIME has elapsed (every x ms). If inputs and outputs are used in this program part, they are also read and written in the defined cycle. Reading and writing can be stopped via the FB inputs READ_INPUTS, WRITE_OUTPUTS and ANALOG_INPUTS.

So, in the program block all time-critical events can be processed by linking inputs or global variables and writing outputs. So, timers can be monitored more precisely than in a "normal cycle".

ⓘ NOTE

To avoid that the program block called by interrupt is additionally called cyclically, it should be skipped in the cycle (with the exception of the initialisation call).

Several timer interrupt blocks can be active. The time requirement of the interrupt functions must be calculated so that all called functions can be executed. This in particular applies to calculations, floating point arithmetic or controller functions.

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

ⓘ NOTE

The uniqueness of the inputs and outputs in the cycle is affected by the interrupt routine. Therefore only part of the inputs and outputs is serviced. If initialised in the interrupt program, the following inputs and outputs will be read or written.

Inputs, digital:

%IX0.0...%IX0.7 (CRnn32)

%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)

%IX0.0, %IX0.8 (SmartController)

IN08...IN11 (CabinetController)

IN0...IN3 (PCB controller)

Inputs, analogue:

%IX0.0...%IX0.7 (CRnn32)

All channels (selection bit-coded) (all other controller)

Outputs, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)

%QX0.0, %QX0.8 (SmartController)

OUT00...OUT03 (CabinetController)

OUT0...OUT7 (PCB controller)

Global variants, too, are no longer unique if they are accessed simultaneously in the cycle and by the interrupt routine. This problem applies in particular to larger data types (e.g. DINT).

All other inputs and outputs are processed once in the cycle, as usual.

Parameters of the inputs

276

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only 1 cycle): changes to data allowed FALSE: changes to data not allowed (during processing of the program)
REPEATTIME	TIME	Time window during which the interrupt is triggered.
READ_INPUTS	BOOL	TRUE: inputs integrated into the routine are read (if necessary, set inputs to IN_FAST). FALSE: this function is not executed
WRITE_OUTPUTS	BOOL	TRUE: outputs integrated into the routine are written to. FALSE: this function is not executed
ANALOG_INPUTS	BYTE	(only for devices with analogue channels) TRUE: analogue inputs integrated into the routine are read and the raw value of the voltage is transferred to the system flags ANALOG_IRQxx FALSE: this function is not executed

12.1.2 SET_INTERRUPT_I

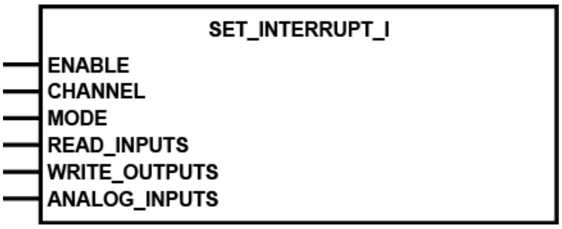
278

Unit type = function block (FB)

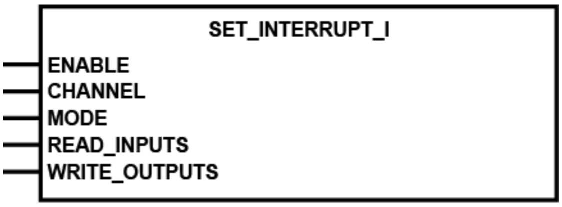
Contained in the library: `ifm_CRnnnn_Vxxyyzz.LIB`

Available for the following devices:
- CabinetController: CR030n
- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- PCB controller: CS0015
- SmartController: CR25nn
- PDM360smart: CR1071

Symbol in CoDeSys:



(only for devices with analogue channels)



(for devices without analogue channels)

Description

281

SET_INTERRUPT_I handles the execution of a program part by an interrupt request via an input channel.

In the conventional PLC the cycle time is decisive for real-time monitoring. So the PLC is at a disadvantage as compared to customer-specific controllers. Even a "real-time operating system" does not change this fact when the whole application program runs in one single block which cannot be changed.

A possible solution would be to keep the cycle time as short as possible. This often leads to splitting the application up to several control cycles. This, however, makes programming complex and difficult.

Another possibility is to call a certain program part only upon request by an input pulse independently of the control cycle.

The time-critical part of the application is integrated by the user in a block of the type PROGRAM (PRG). This block is declared as the interrupt routine by calling SET_INTERRUPT_I once (during initialisation). As a consequence, this program block will always be executed if an edge is detected on the input CHANNEL. If inputs and outputs are used in this program part, these are also read and written in the interrupt routine, triggered by the input edge. Reading and writing can be stopped via the FB inputs READ_INPUTS, WRITE_OUTPUTS and ANALOG_INPUTS.

So in the program block all time-critical events can be processed by linking inputs or global variables and writing outputs. So FBs can only be executed if actually called by an input signal.

ⓘ NOTE

The program block should be skipped in the cycle (except for the initialisation call) so that it is not cyclically called, too.

The input (CHANNEL) monitored for triggering the interrupt cannot be initialised and further processed in the interrupt routine.

The inputs must be in the operating mode IN_FAST, otherwise the interrupts cannot be read.

ⓘ NOTE

The uniqueness of the inputs and outputs in the cycle is affected by the interrupt routine. Therefore only part of the inputs and outputs is serviced. If initialised in the interrupt program, the following inputs and outputs will be read or written.

Inputs, digital:

%IX0.0...%IX0.7 (CRnn32)

%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)

%IX0.0, %IX0.8 (SmartController)

IN08...IN11 (CabinetController)

IN0...IN3 (PCB controller)

Inputs, analogue:

%IX0.0...%IX0.7 (CRnn32)

All channels (selection bit-coded) (all other controller)

Outputs, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)

%QX0.0, %QX0.8 (SmartController)

OUT00...OUT03 (CabinetController)

OUT0...OUT7 (PCB controller)

Global variants, too, are no longer unique if they are accessed simultaneously in the cycle and by the interrupt routine. This problem applies in particular to larger data types (e.g. DINT).

All other inputs and outputs are processed once in the cycle, as usual.

Parameters of the inputs

282

Parameter	Data type	Description
ENABLE	BOOL	TRUE (only for 1 cycle): changes to data permissible FALSE: changes to data not permitted (during processing of the program)
CHANNEL	BYTE	interrupt input Classic/ExtendedController: 0 = %IX1.4 1 = %IX1.5 2 = %IX1.6 3 = %IX1.7 SmartController: 0 = %IX0.0 1 = %IX0.8 CabinetController: 0 = IN08 (etc.) 3 = IN11 CS0015: 0 = IN0 (etc.) 3 = IN3
MODE	BYTE	type of edge at the input CHANNEL which triggers the interrupt 1 = rising edge 2 = falling edge 3 = rising and falling edge
READ_INPUTS	BOOL	TRUE: inputs integrated into the routine are read (if necessary, set inputs to IN_FAST) FALSE: this function is not executed
WRITE_OUTPUTS	BOOL	TRUE: outputs integrated into the routine are written FALSE: this function is not executed
ANALOG_INPUTS	BYTE	(only for devices with analogue channels) selection of the inputs bit-coded: 0_{10} = no input selected 1_{10} = 1st analogue input selected ($0000\ 0001_2$) 2_{10} = 2nd analogue input selected ($0000\ 0010_2$) ... 128_{10} = 8th analogue input selected ($1000\ 0000_2$) A combination of the inputs is possible via an OR operation of the values. Example: Select 1st and 3rd analogue input: ($0000\ 0001_2$) OR ($0000\ 0100_2$) = ($0000\ 0101_2$) = 5_{10}

13 Annex

Contents

Address assignment and I/O operating modes 299
System flags 305
CANopen tables 306
Overview of the files and libraries used 319

1664

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

13.1 Address assignment and I/O operating modes

Contents

Address assignment inputs / outputs 300
Addresses / variables of the I/Os 302
Possible operating modes inputs / outputs 304

1656

→ also data sheet

13.1.1 Address assignment inputs / outputs

1657

IEC address	Name IO variable	Configuration with variable	Default value	Possible configuration
%IX0.0 %IW10	IN00 A_IN08	-- --	-- --	only L digital diagnosis for IN00
%IX0.1 %IW11	IN01 A_IN09	-- --	-- --	only L digital diagnosis for IN01
%IX0.2 %IW12	IN02 A_IN10	-- --	-- --	only L digital diagnosis for IN02
%IX0.3 %IW13	IN03 A_IN11	-- --	-- --	only L digital diagnosis for IN03
%IX0.4 %IW14	IN04 A_IN12	-- --	-- --	only L digital diagnosis for IN04
%IX0.5 %IW15	IN05 A_IN13	-- --	-- --	only L digital diagnosis for IN05
%IX0.6 %IW16	IN06 A_IN14	-- --	-- --	only L digital diagnosis for IN06
%IX0.7 %IW17	IN07 A_IN15	-- --	-- --	only L digital diagnosis for IN07
%IX0.8 %IW18	IN08 A_IN16	IN08_MODE --	1 --	L digital | FRQ00 diagnosis for IN08
%IX0.9 %IW19	IN09 A_IN17	IN09_MODE --	1 --	L digital | FRQ01 diagnosis for IN09
%IX0.10 %IW20	IN10 A_IN18	IN10_MODE --	1 --	L digital | FRQ02 diagnosis for IN10
%IX0.11 %IW21	IN11 A_IN19	IN11_MODE --	1 --	L digital | FRQ03 diagnosis for IN11
%IX0.12 %IW22	IN12 A_IN20	IN12_MODE --	1 --	L digital | H digital diagnosis for IN12
%IX0.13 %IW23	IN13 A_IN21	IN13_MODE --	1 --	L digital | H digital diagnosis for IN13
%IX0.14 %IW24	IN14 A_IN22	IN14_MODE --	1 --	L digital | H digital diagnosis for IN14
%IX0.15 %IW25	IN15 A_IN23	IN15_MODE --	1 --	L digital | H digital diagnosis for IN15
%IX0.16 %IW2	A_IN16 A_IN00	A_IN16_MODE --	16 --	analogue U|I | L digital diagnosis for IN16
%IX0.17 %IW3	A_IN17 A_IN01	A_IN17_MODE --	16 --	analogue U|I | L digital diagnosis for IN17
%IX0.18 %IW4	A_IN18 A_IN02	A_IN18_MODE --	16 --	analogue U|I | L digital diagnosis for IN18
%IX0.19 %IW5	A_IN19 A_IN03	A_IN19_MODE --	16 --	analogue U|I | L digital diagnosis for IN19
%IX0.20 %IW6	A_IN20 A_IN04	A_IN20_MODE --	16 --	analogue U|I | L digitalD diagnosis for IN20
%IX0.21 %IW7	A_IN21 A_IN05	A_IN21_MODE --	16 --	analogue U|I | L digitalD diagnosis for IN21
%IX0.22 %IW8	A_IN22 A_IN06	A_IN22_MODE --	16 --	analogue U|I | L digital diagnosis for IN22
%IX0.23 %IW9	A_IN23 A_IN07	A_IN23_MODE --	16 --	analogue U|I | L digital diagnosis for IN23

IEC address	Name IO variable	Configuration with variable	Default value	Possible configuration
%IW26	A_IN24	--	--	diagnosis for OUT12
%IW27	A_IN25	--	--	diagnosis for OUT13
%IW28	A_IN26	--	--	diagnosis for OUT14
%IW29	A_IN27	--	--	diagnosis for OUT15
%IW30	A_IN28	--	--	diagnosis for OUT16
%IW31	A_IN29	--	--	diagnosis for OUT17
%QX0.0	OUT00	--	--	H digital / PWM0
%QX0.1	OUT01	--	--	H digital / PWM1
%QX0.2	OUT02	--	--	H digital / PWM2
%QX0.3	OUT03	--	--	H digital / PWM3
%QX0.4	OUT04	--	--	H digital / PWM4
%QX0.5	OUT05	--	--	H digital / PWM5
%QX0.6	OUT06	--	--	H digital / PWM6
%QX0.7	OUT07	--	--	H digital / PWM7
%QX0.8	OUT08	--	--	only H digital
%QX0.9	OUT09	--	--	only H digital
%QX0.10	OUT10	--	--	only H digital
%QX0.11	OUT11	--	--	only H digital
%QX0.12 %IW26	OUT12 CURR12	-- --	-- --	only H digital current measurement of OUT12
%QX0.13 %IW27	OUT13 CURR13	-- --	-- --	only H digital current measurement of OUT13
%QX0.14 %IW28	OUT14 CURR14	-- --	-- --	only H digital current measurement of OUT14
%QX0.15 %IW29	OUT15 CURR15	-- --	-- --	only H digital current measurement of OUT15
%QX0.16 %IW30	OUT16 CURR16	-- --	-- --	only H digital current measurement of OUT16
%QX0.17 %IW31	OUT17 CURR17	-- --	-- --	only H digital current measurement of OUT17

PWM description → chapter PWM signal processing (→ page 214)

FRQ description → chapter Counter functions for frequency and period measurement (→ page 198)

13.1.2 Addresses / variables of the I/Os

1658

IEC address	I/O variable	Remark
%IB0	--	input byte 0 (%IX0.0...%IX0.7)
%IB1	--	input byte 1 (%IX0.8...%IX0.15)
%IW2	A_IN16	analogue input word 16
%IW3	A_IN17	analogue input word 17
%IW4	A_IN18	analogue input word 18
%IW5	A_IN19	analogue input word 19
%IW6	A_IN20	analogue input word 20
%IW7	A_IN21	analogue input word 21
%IW8	A_IN22	analogue input word 22
%IW9	A_IN23	analogue input word 23
%IW10	A_IN00	diagnosis of IN00 / %IX0.0
%IW11	A_IN01	diagnosis of IN01 / %IX0.1
%IW12	A_IN02	diagnosis of IN02 / %IX0.2
%IW13	A_IN03	diagnosis of IN03 / %IX0.3
%IW14	A_IN04	diagnosis of IN04 / %IX0.4
%IW15	A_IN05	diagnosis of IN05 / %IX0.5
%IW16	A_IN06	diagnosis of IN06 / %IX0.6
%IW17	A_IN07	diagnosis of IN07 / %IX0.7
%IW18	A_IN08	diagnosis of IN08 / %IX0.8
%IW19	A_IN09	diagnosis of IN09 / %IX0.9
%IW20	A_IN10	diagnosis of IN10 / %IX0.10
%IW21	A_IN11	diagnosis of IN11 / %IX0.11
%IW22	A_IN12	diagnosis of IN012 / %IX0.12
%IW23	A_IN13	diagnosis of IN13 / %IX0.13
%IW24	A_IN14	diagnosis of IN14 / %IX0.14
%IW25	A_IN15	diagnosis of IN15 / %IX0.15
%IW26	CURR12	current measurement of OUT12 / %QX0.12
%IW27	CURR13	current measurement of OUT13 / %QX0.13
%IW28	CURR14	current measurement of OUT14 / %QX0.14
%IW29	CURR15	current measurement of OUT15 / %QX0.15
%IW30	CURR16	current measurement of OUT16 / %QX0.16
%IW31	CURR17	current measurement of OUT17 / %QX0.17
%IW37	SUPPLY_VOLTAGE	Supply voltage on VBBs in [mV]
%IW38	REF_VOLTAGE	measurement of the reference voltage at X20:14
%QB3	IN00_MODE	configuration byte for %IX0.0
%QB4	IN01_MODE	configuration byte for %IX0.1
%QB5	IN02_MODE	configuration byte for %IX0.2
%QB6	IN03_MODE	configuration byte for %IX0.3
%QB7	IN04_MODE	configuration byte for %IX0.4

IEC address	I/O variable	Remark
%QB8	IN05_MODE	configuration byte for %IX0.5
%QB9	IN06_MODE	configuration byte for %IX0.6
%QB10	IN07_MODE	configuration byte for %IX0.7
%QB11	IN08_MODE	configuration byte for %IX0.8
%QB12	IN09_MODE	configuration byte for %IX0.9
%QB13	IN10_MODE	configuration byte for %IX0.10
%QB14	IN11_MODE	configuration byte for %IX0.11
%QB15	IN12_MODE	configuration byte for %IX0.12
%QB16	IN13_MODE	configuration byte for %IX0.13
%QB17	IN14_MODE	configuration byte for %IX0.14
%QB18	IN15_MODE	configuration byte for %IX0.15
%QB19	A_IN16_MODE	configuration byte for %IW2
%QB20	A_IN17_MODE	configuration byte for %IW3
%QB21	A_IN18_MODE	configuration byte for %IW4
%QB22	A_IN19_MODE	configuration byte for %IW5
%QB23	A_IN20_MODE	configuration byte for %IW6
%QB24	A_IN21_MODE	configuration byte for %IW7
%QB25	A_IN22_MODE	configuration byte for %IW8
%QB26	A_IN23_MODE	configuration byte for %IW9
%QB27	REFERENCE_VOLTAGE_5	output of the reference voltage 5 V at X20:14 *)
%QB28	REFERENCE_VOLTAGE_10	output of the reference voltage 10 V at X20:14 *)
flag bit	ERROR_Ix	input voltage too high / too low
flag bit	ERROR_I2	input current too high

*) Which reference voltage is output?

%QB28	%QB27	X20:14
0	0	0 V
0	1	5 V
1	0	10 V
1	1	0 V

13.1.3 Possible operating modes inputs / outputs

1659

Inputs	Operating mode	Config. value	Outputs	Operating mode	Config. value
A_IN16… A_IN23	IN_DIGITAL_H (plus)	1			
	IN_CURRENT	4			
	IN_VOLTAGE10	8			
	IN_VOLTAGE30	16 (default)			
	IN_RATIO	32			
	IN_DIAGNOSTIC	64			
IN08…IN11	IN_DIGITAL_H (plus)	1 (default)			
	IN_DIAGNOSTIC	64			
	IN_DIGITAL_FAST	128			
I12…I15	IN_DIGITAL_H (plus)	1 (default)			
	IN_DIGITAL_L (minus)	2			
	IN_DIAGNOSTIC	64			

13.2 System flags

1693

(→ chapter Error codes and diagnostic information (→ page 56))

System flags	Type	Description
CANx_BAUDRATE	WORD	CAN interface x: Currently set baud rate
CANx_BUSOFF	BOOL	CAN interface x: Interface is not on the bus
CANx_LASTERROR [1]	BYTE	CAN interface x: Error number of the last CAN transmission: 0= no error ≠0 → CAN specification → LEC
CANx_WARNING	BOOL	CAN interface x: Warning threshold reached (> 96)
DOWNLOADID	WORD	Currently set download identifier
ERROR	BOOL	Set ERROR bit
ERROR_Ix	BYTE	Peripheral error on the input group x
ERROR_MEMORY	BOOL	Memory error
ERROR_OVERCURRENT_OUTxx	BOOL	Error: OUT12...17 overcurrent
ERROR_POWER	BOOL	Error: undervoltage / overvoltage
ERROR_TEMPERATUR	BOOL	Error: excess temperature (> 85 °C)
LED	WORD	LED colour for "active" (= on)
LED_MODE	WORD	Flashing frequency from the data structure "LED_MODES"
LED_X	WORD	LED colour for "pause" (= out)
REF_VOLTAGE	WORD	Diagnosis reference voltage output X20:14
REFERENCE_VOLTAGE_5	BYTE	Setting the reference voltage output X20:14 to 5 V
REFERENCE_VOLTAGE_10	BYTE	Setting the reference voltage output X20:14 to 10 V
SERIAL_MODE	BOOL	Switch on the serial communication
SERIALBAUDRATE	WORD	Baud rate of the RS-232 interface
SUPPLY_VOLTAGE	WORD	Supply voltage on VBBs in [mV]
TEST	BOOL	Enable the programming mode
WARNING_OVERCURRENT_OUTxx	BOOL	Warning: OUT12...17 overcurrent

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

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

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

NOTE

Only symbol names should be used for programming since the corresponding flag addresses can change in case of an extension of the PLC configuration.

13.3 CANopen tables

Contents

IDs (addresses) in CANopen 306
Structure of CANopen messages........ 307
Bootup messsage........ 311
Network management (NMT)........ 312
CANopen error code 316

9941

The following tables will inform you about important values and settings of the CANopen interfaces.

13.3.1 IDs (addresses) in CANopen

3952

In CANopen there are different types of addresses (IDs):

- COB ID
  The **C**ommunication **Ob**ject **Id**entifier addresses the message (= the communication object) in the list of devices. A communication object consists of one or more CAN messages with a specific functionality, e.g.
  - PDO (**P**rocess **D**ata **O**bject = message object with process data),
  - SDO (**S**ervice **D**ata **O**bject.= message object with service data),
  - emergency (message object with emergency data),
  - time (message object with time data) or
  - error control (message object with error messages).
- CAN ID
  The **CAN Id**entifier defines CAN messages in the complete network. The CAN ID is the main part of the arbitration field of a CAN data frame. The CAN ID value determines implicity the priority for the bus arbitration.
- Download ID
  The download ID indicates the node ID for service communication via SDO for the program download and for debugging.
- Node ID
  The **Node Id**entifier is a unique descriptor for CANopen devices in the CAN network. The Node ID is also part of some pre-defined connectionsets (→ function code (→ page 308)).

Comparison of download-ID vs. COB-ID:

Controller program download		**CANopen**	
Download ID	**COB ID SDO**	**Node ID**	**COB ID SDO**
1…127	TX: 580_{16} + download ID	1…127	TX: 580_{16} + node ID
	RX: 600_{16} + download ID		RX: 600_{16} + node ID

TX = slave sends to master
RX = slave receives from master

13.3.2 Structure of CANopen messages

Contents

Structure of the COB ID 307
Function code / Predefined Connectionset 308
SDO command bytes 309
SDO abort code 310

9971

A CANopen message consists of the COB ID and up to 8-byte data:

COB ID			DLC	Byte 1		Byte 2		Byte 3		Byte 4		Byte 5		Byte 6		Byte 7		Byte 8	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Details are given in the following chapters.

NOTE: Please note the reversed byte order!

Examples:

Value [hex]	Data type	Byte 1		Byte 2		Byte 3		Byte 4		Byte 5		Byte 6		Byte 7		Byte 8	
12	BYTE	1	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–
1234	WORD	3	4	1	2	–	–	–	–	–	–	–	–	–	–	–	–
12345678	DWORD	7	8	5	6	3	4	1	2	–	–	–	–	–	–	–	–

Structure of the COB ID

9972

The first part of a message is the COB ID. Structure of the 11-bit COB ID:

Nibble 0				**Nibble 1**				**Nibble 2**			
11	10	9	8	7	6	5	4	3	2	1	0
--	**3**	**2**	**1**	**0**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
--	**function code**				**node ID**						

The COB ID consists of the function code (→ page 308) and the node ID.

Example:
Communication object = TPDO1 (TX)
Node number of the device = $20_{16} = 32_{10}$

Calculation:
Function code for the communication object TPDO1 = 3_{16}
Significance of the function code in the 11-bit COB ID = 3_{16} x 80_{16} = 180_{16}
Add the node number (20_{16}) ⇒ the COB ID is: $1A0_{16}$

1				**A**				**0**			
3	2	1	0	3	2	1	0	3	2	1	0
0	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
--	$3_{16} = 3_{10}$				$20_{16} = 32_{10}$						

Function code / Predefined Connectionset

9966

In the "CANopen Predefined Connectionset" some function codes are predefined.

When using the predefined connectionset you can operate a CANopen network of up to 127 participants without the risk of a double assignment of COB IDs.

Broadcast or multicast messages:

Communication object	Function code [hex]	COB ID [hex]	Related parameter objects [hex]
NMT	0	000	
SYNC	1	080	1005, 1006, 1007, 1028
TIME	2	100	1012, 1013

Point-to-point messages:

Communication object	Function code [hex]	COB ID [hex]	Related parameter objects [hex]
EMERGENCY	1	080 + node ID	1014, 1015
TPDO1 (TX)	3	180 + node ID	1800
RPDO1 (RX)	4	20016 + node ID	1400
TPDO2 (TX)	5	280 + node ID	1801
RPDO2 (RX)	6	30016 + node ID	1401
TPDO3 (TX)	7	380 + node ID	1802
RPDO3 (RX)	8	400 + node ID	1402
TPDO4 (TX)	9	480 + node ID	1803
RPDO4 (RX)	A	500 + node ID	1403
Default SSDO (TX)	B	58016 + node ID	1200
Default CSDO (RX)	C	60016 + node ID	1280
NMT Error Control	E	70016 + node ID	1016, 1017

TX = slave sends to master
RX = slave receives from master

SSDO = server SDO
CSDO = client SDO

SDO command bytes

9968

Structure of an SDO message:

COB ID	DLC	Command	Index		Sub-index	Data *)			
XXX	8	byte	byte 0	byte 1	byte	byte 0	byte 1	byte 2	byte 3

*) depending on the data to be transmitted

NOTE: Please note the reversed byte order!

An SDO COB ID consists of:

CANopen	
Node ID	**COB ID SDO**
1…127	TX: 580_{16} + node ID
	RX: 600_{16} + node ID

TX = slave sends to master
RX = slave receives from master

DLC (data length code) indicates the number of the data bytes (for SDO: DLC = 8).

SDO command bytes:

Command hex | dec		Message	Data length	Description
21	33	request	more than 4 bytes	send data to slave
22	34	request	1…4 bytes	send data to slave
23	35	request	4 bytes	send data to slave
27	39	request	3 bytes	send data to slave
2B	43	request	2 bytes	send data to slave
2F	47	request	1 byte	send data to slave
40	64	request	---	request data from slave
42	66	response	1…4 bytes	send data from slave to master
43	67	response	4 bytes	send data from slave to master
47	71	response	3 bytes	send data from slave to master
4B	75	response	2 bytes	send data from slave to master
4F	79	response	1 byte	send data from slave to master
60	96	response	---	data transfer ok: send confirmation of receipt from slave to master
80	128	response	4 bytes	data transfer failed send abort message from slave to master → chapter SDO abort code (→ page 310)

SDO abort code

9970

NOTE: The SDO abort code is NOT part of the emergency message!

Abord code [hex]	Description
0503 0000	toggle bit not alternated
0504 0000	SDO protocol timed out
0504 0001	client/server command specifier not valid or unknown
0504 0002	invalid block size (block mode only)
0504 0003	invalid sequence number (block mode only)
0504 0004	CRC error (block mode only)
0504 0005	out of memory
0601 0000	unsupported access to an object
0601 0001	attempt to read a write only object
0601 0002	attempt to write a read only object
0602 0000	object does not exist in the object dictionary
0604 0041	object cannot be mapped to the PDO
0604 0042	the number and length of the objects to be mapped would exceed PDO length
0604 0043	general parameter incompatibility reason
0604 0047	general internal incompatibility in the device
0606 0000	access failed due to an hardware error
0607 0010	data type does not match, length of service parameter does not match
0607 0012	data type does not match, length of service parameter too high
0607 0013	data type does not match, length of service parameter too low
0609 0011	sub-index does not exist
0609 0030	value range of parameter exceeded (only for write access)
0609 0031	value of parameter written too high
0609 0032	value of parameter written too low
0609 0036	maximum value is less than minimum value
0800 0000	general error
0800 0020	data cannot be transferred or stored to the application
0800 0021	data cannot be transferred or stored to the application because of local control
0800 0022	data cannot be transferred or stored to the application because of the present device state
0800 0023	object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error)

13.3.3 Bootup messsage

9961

After booting the CAN participate sends the boot-up message once:

hex	dec
700_{16} + node ID	1 792_{10} + node ID

The participant is now capable of communicating in the CAN network.

Structure:
The node ID of the participant is $7D_{16}$ = 125_{10}.
The boot-up message is: $77D_{16}$ = 1 917_{10}

13.3.4 Network management (NMT)

Contents

Network management commands 312
NMT state 313

9974

Network management commands

9962

With the following network management commands the user can influence the operating mode of individual or all CAN participants. Structure:

Byte 1	Byte 2	Byte 2
COB ID	command	node ID

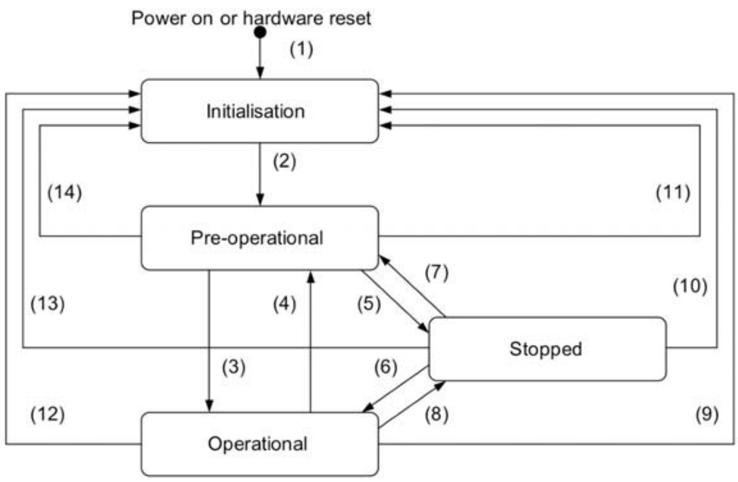
Node ID = 00 ⇒ command valid for all nodes in the network at the same time

COB ID	NMT command		Description	
00	$01_{16} = 01_{10}$	node ID	start_remode_node	start CAN participate
00	$02_{16} = 02_{10}$	node ID	stop_remode_node	stop CAN participate
00	$80_{16} = 128_{10}$	node ID	enter_pre-operational	switch to pre-operational
00	$81_{16} = 129_{10}$	node ID	reset node	reset CAN participate
00	$82_{16} = 130_{10}$	node ID	reset communication	reset CAN communication

NMT state

9963

The status byte informs about the state of the CAN participant.



Graphics: state transitions under CANopen

Permitted transitions:

(1) State is automatically reached at power on

(2) Internal initialisation completed – node automatically goes to PRE-OPERATIONAL

(3) NMT service "Start Remote Node Indication"

(4) + (7) NMT service "Enter PRE-OPERATIONAL Indication"

(5) + (8) NMT service "Stop Remote Node Indication"

(6) NMT service "Start Remote Node Indication"

(9)...(11) NMT service "Reset Node Indication"

(12)...(14) NMT service "Reset Communication Indication"

NMT state for CANopen master

9964

State		Description
hex	dec	
00	0	not defined
01	1	Master waits for a boot-up message of the node. OR: Master waits for the expiry of the given guard time.
02	2	- Master waits for 300 ms. - Master requests the object 1000_{16}. - Then the state is set to 3.
03	3	The master configures its slaves. To do so, all SDOs generated by the configurator are transmitted to the slaves one after the other: - The Master sends to the slave a SDO read request (index 1000_{16}). - The generated SDOs are compressed into a SDO array. - The slave knows it's first SDO and the number of it's SDOs.
05	5	After transmission of all SDOs to the slaves the master goes to state 5 and remains in this state. State 5 is the normal operating state for the master.

To read the node state out of the FB:

Used function block	Node state is found here
CANx_MASTER_STATUS CANx_SLAVE_STATUS	output NODE_STATE
CANOPEN_GETSTATE	output NODESTATE

NMT state for CANopen slave

9965

State hex	State dec	Description
FF	-1	The slave is reset by the NMT message "Reset Node" and automatically goes to state 1.
00	0	not defined
01	1	state = waiting for BOOTUP After max. 2 s or immediately on reception of its boot up message the slave goes to state 2.
02	2	state = BOOTUP After a delay of 0.5 s the slave automatically goes to state 3.
03	3	state = PREPARED The slave is configured in state 3. The slave remains in state 3 as long as it has received all SDOs generated by the configurator. It is not important whether during the slave configuration the response to SDO transfers is abort (error) or whether the response to all SDO transfers is no error. Only the response as such received by the slave is important – not its contents. If in the configurator the option "Reset node" has been activated, a new reset of the node is carried out after transmitting the object 1011_{16} sub-index 1 which then contains the value "load". The slave is then polled again with the upload of the object 1000_{16}. Slaves with a problem during the configuration phase remain in state 3 or directly go to an error state (state > 5) after the configuration phase.
04	4	state = PRE-OPERATIONAL A node always goes to state 4 except for the following cases: • it is an "optional" slave and it was detected as non available on the bus (polling for object 1000_{16}) OR: • the slave is present but reacted to the polling for object 1000_{16} with a type in the lower 16 bits other than expected by the configurator.
05	5	state = OPERATIONAL State 5 is the normal operating state of the slave: [Normal Operation]. If the master was configured to [Automatic startup], the slave starts in state 4 (i.e. a "start node" NMT message is generated) and the slave goes automatically to state 5. If the flag GLOBAL_START was set, the master waits until all slaves are in state 4. All slaves are then started with the NMT command [Start All Nodes].
61	97	A node goes to state 97 if it is optional (optional device in the CAN configuration) and has not reacted to the SDO polling for object 1000_{16}. If the slave is connected to the network and detected at a later point in time, it is automatically started. To do so, you must have selected the option [Automatic startup] in the CAN parameters of the master.
62	98	A node goes to state 98 if the device type (object 1000_{16}) does not correspond to the configured type.
63	99	In case of a nodeguarding timeout the slave is set to state 99. As soon as the slave reacts again to nodeguard requests and the option [Automatic startup] is activated, it is automatically started by the master. Depending on the status contained in the response to the nodeguard requests, the node is newly configured or only started. To start the slave manually it is sufficient to use the method [NodeStart].

Nodeguard messages are transmitted to the slave ...
- if the slave is in state 4 or higher AND
- if nodeguarding was configured.

To read the node state out of the FB:

Used function block	Node state is found here
CANx_MASTER_STATUS CANx_SLAVE_STATUS	output NODE_STATE
CANOPEN_GETSTATE	output NODESTATE

CANopen status of the node

1973

Node status according to CANopen (with these values the status is also coded by the node in the corresponding messages).

Status hex | dec		CANopen status	Description
00	0	BOOTUP	Node received the boot-up message.
04	4	PREPARED	Node is configured via SDOs.
05	5	OPERATIONAL	Node participates in the normal exchange of data.
7F	127	PRE-OPERATIONAL	Node sends no data, but can be configred by the master.

If nodeguarding active: the most significant status bit toggles between the messages.

Read the node status from the function block:

Function block used	Node status is found here
CANx_MASTER_STATUS CANx_SLAVE_STATUS	Structure element LAST_STATE from the array NODE_STATE_SLAVE
CANOPEN_GETSTATE	Output LASTNODESTATE

13.3.5 CANopen error code

Contents

Emergency messages........ 316
Overview CANopen error codes 317
Object 0x1001 (error register)........ 318

9967

Emergency messages

9973

Device errors in the slave or problems in the CAN bus trigger emergency messages:

COB ID	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
80_{16} + node ID		error code		object 1001_{16}	device-specific				

NOTE: Please note the reversed byte order!

Overview CANopen error codes

8545

Error Code (hex)	Meaning
00xx	Reset or no error
10xx	Generic error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains voltage
32xx	Voltage inside the device
33xx	Output voltage
40xx	Temperature
41xx	Ambient temperature
42xx	Device temperature
50xx	Device hardware
60xx	Device software
61xx	Internal software
62xx	User software
63xx	Data set
70xx	Additional modules
80xx	Monitoring
81xx	Communication
8110	CAN overrun-objects lost
8120	CAN in error passiv mode
8130	Life guard error or heartbeat error
8140	Recovered from bus off
8150	Transmit COB-ID collision
82xx	Protocol error
8210	PDO not procedded due to length error
8220	PDO length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

Object 0x1001 (error register)

8547

This object reflects the general error state of a CANopen device. The device is to be considered as error free if the object 1001_{16} signals no error any more.

Bit	Meaning
0	generic error
1	current
2	voltage
3	temperature
4	communication error
5	device profile specific
6	reserved – always 0
7	manufacturer specific

For an error message more than one bit in the error register can be set at the same time.

Example: CR2033, message "wire break" at channel 2 (→ installation manual of the device):

COB-ID	DLC	Byte 0	Byte 1	Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
80_{16} + node ID		**00**	**FF**	**81**	**10**	**00**	00	00	00

Error-Code = $FF00_{16}$

Error register = 81_{16} = $1000\ 0001_2$, thus it consists of the following errors:
- generic error
- manufacturer specific

Concerned channel = 0010_{16} = $0000\ 0000\ 0001\ 0000_2$ = wire break channel 2

13.4 Overview of the files and libraries used

Contents

Installation of the files and libraries 319
General overview 320
What are the individual files and libraries used for? 322

2711

(as on 2011-03-02)

Depending on the unit and the desired function, different libraries and files are used. Some are automatically loaded, others must be inserted or loaded by the programmer.

13.4.1 Installation of the files and libraries

2721

Factory setting: the device contains only the boot loader.

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

Certain libraries automatically integrate further libraries into the project.
Some FBs in **ifm** libraries (`ifm_*.LIB`) e.g. are based on FBs in CoDeSys libraries (`3S_*.LIB`).

13.4.2 General overview

2712

File name	Description and memory location 3)
`ifm_CRnnnn_Vxxyyzz.CFG` 1) `ifm_CRnnnn_Vxx.CFG` 2)	PLC configuration per device only 1 device-specific file inlcudes: IEC and symbolic addresses of the inputs and outputs, the flag bytes as well as the memory allocation …`\CoDeSys V*\Targets\ifm\ifm_CRnnnncfg\Vxxyyzz`
`CAA-*.CHM`	Online help per device only 1 device-specific file inlcudes: online help for this device …`\CoDeSys V*\Targets\ifm\Help\`… (language)
`ifm_CRnnnn_Vxxyyzz.H86` `ifm_CRnnnn_Vxxyyzz.RESX`	Operating system / runtime system (must be loaded into the controller / monitor when used for the first time) per device only 1 device-specific file …`\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn`
`ifm_Browser_CRnnnn.INI`	CoDeSys browser commands (CoDeSys needs the file for starting the project) per device only 1 device-specific file inlcudes: commands for the browser in CoDeSys …`\CoDeSys V*\Targets\ifm`
`ifm_Errors_CRnnnn.INI`	CoDeSys error file (CoDeSys needs the file for starting the project) per device only 1 device-specific file inlcudes: device-specific error messages from CoDeSys …`\CoDeSys V*\Targets\ifm`
`ifm_CRnnnn_Vxx.TRG`	Target file per device only 1 device-specific file inlcudes: hardware description for CoDeSys, e.g.: memory, file locations …`\CoDeSys V*\Targets\ifm`
`ifm_*_Vxxyyzz.LIB`	General libraries per device several files are possible …`\CoDeSys V*\Targets\ifm\Library`
`ifm_CRnnnn_Vxxyyzz.LIB`	Device-specific library per device only 1 device-specific file inlcudes: POUs of this device …`\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn`
`ifm_CRnnnn_*_Vxxyyzz.LIB`	Device-specific libraries per device several files are possible → following tables …`\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn`

Legend:

*	any signs
CRnnnn	article number of the controller / monitor
V*	CoDeSys version
Vxx	version number of the ifm software
yy	release number of the ifm software
zz	patch number of the ifm software

1) valid for CRnn32 target version up to V01, all other devices up to V04

2) valid for CRnn32 target version from V02 onwards, CR040n target version from V01 onwards, all other devices from V05 onwards

3) memory location of the files:
`System drive (C: / D:) \ program folder\ ifm electronic`

ⓘ NOTE

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

- operating system (`CRnnnn_Vxxyyzz.H86` / `CRnnnn_Vxxyyzz.RESX`)
- PLC configuration (`CRnnnn_Vxx.CFG`)
- device library (`ifm_CRnnnn_Vxxyyzz.LIB`)
- and the further files (→ chapter Overview of the files and libraries used (→ page 319))

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

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

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

IMPORTANT: the following files must also be loaded:

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

13.4.3 What are the individual files and libraries used for?

Contents

Files for the operating system / runtime system 322
Target file 322
PLC configuration file 322
ifm device libraries 323
ifm CANopen libraries master / slave 323
CoDeSys CANopen libraries 324
Specific ifm libraries 325

2713

The following overview shows which files/libraries can and may be used with which unit. It may be possible that files/libraries which are not indicated in this list can only be used under certain conditions or the functionality has not yet been tested.

Files for the operating system / runtime system

2714

File name	Function	Available for:
`ifm_CRnnnn_Vxxyyzz.H86` `ifm_CRnnnn_Vxxyyzz.RESX`	operating system / runtime system	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn
`ifm_Browser_CRnnnn.INI`	CoDeSys browser commands	all ecomatmobile controllers PDM: CR10nn
`ifm_Errors_CRnnnn.INI`	CoDeSys error file	all ecomatmobile controllers PDM: CR10nn

Target file

2715

File name	Function	Available for:
`ifm_CRnnnn_Vxx.TRG`	Target file	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn

PLC configuration file

2716

File name	Function	Available for:
`ifm_CRnnnn_Vxxyyzz.CFG`	PLC configuration	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn

ifm device libraries

2717

File name	Function	Available for:
`ifm_CRnnnn_Vxxyyzz.LIB`	device-specific library	all ecomatmobile controllers BasicDisplay: CR0451 PDM: CR10nn
`ifm_CR0200_MSTR_Vxxyyzz.LIB`	library without extended functions	ExtendedController: CR0200
`ifm_CR0200_SMALL_Vxxyyzz.LIB`	library without extended functions, reduced functions	ExtendedController: CR0200

ifm CANopen libraries master / slave

2718

These libraries are based on the CoDeSys libraries (3S CANopen POUs) and make them available to the user in a simple way.

File name	Function	Available for:
`ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`	CANopen master emergency and status handler	all ecomatmobile controllers *) PDM: CR10nn *)
`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`	CANopen slave emergency and status handler	all ecomatmobile controllers *) PDM: CR10nn *)
`ifm_CANx_SDO_Vxxyyzz.LIB`	CANopen SDO read and SDO write	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`ifm_CANopen_NT_Vxxyyzz.LIB`	CANopen POUs in the CAN stack	BasicController: CR040n BasicDisplay: CR0451 PDM360NG: CR108n, CR9042

*) but NOT for...
- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360NG: CR108n, CR9042

CoDeSys CANopen libraries

2719

For the following devices these libraries are NOT useable:
- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360NG: CR108n, CR9042

File name	Function	Available for:
`3S_CanDrvOptTable.LIB` 1) `3S_CanDrvOptTableEx.LIB` 2)	CANopen driver	all ecomat*mobile* controllers PDM360smart: CR1070, CR1071
`3S_CanDrv.LIB` 3)		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`3S_CANopenDeviceOptTable.LIB` 1) `3S_CANopenDeviceOptTableEx.LIB` 2)	CANopen slave driver	all ecomat*mobile* controllers PDM360smart: CR1070, CR1071
`3S_CANopenDevice.LIB` 3)		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`3S_CANopenManagerOptTable.LIB` 1) `3S_CANopenManagerOptTableEx.LIB` 2)	CANopen network manager	all ecomat*mobile* controllers PDM360smart: CR1070, CR1071
`3S_CANopenManager.LIB` 3)		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`3S_CANopenMasterOptTable.LIB` 1) `3S_CANopenMasterOptTableEx.LIB` 2)	CANopen master	all ecomat*mobile* controllers PDM360smart: CR1070, CR1071
`3S_CANopenMaster.LIB` 3)		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`3S_CANopenNetVarOptTable.LIB` 1) `3S_CANopenNetVarOptTableEx.LIB` 2)	Driver for network variables	all ecomat*mobile* controllers PDM360smart: CR1070, CR1071
`3S_CANopenNetVar.LIB` 3)		PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056

1) valid for CRnn32 target version up to V01, all other devices up to V04
2) valid for CRnn32 target version from V02 onwards, all other devices from V05 onwards
3) For the following devices: This library is without function used as placeholder:
- BasicController: CR040n
- BasicDisplay: CR0451
- PDM360NG: CR108n, CR9042

Specific ifm libraries

2720

File name	Function	Available for:
`ifm_RawCAN_NT_Vxxyyzz.LIB`	CANopen POUs in the CAN stack based on Layer 2	BasicController: CR040n BasicDisplay: CR0451 PDM360NG: CR108n, CR9042
`ifm_J1939_NT_Vxxyyzz.LIB`	J1939 communication POUs in the CAN stack	BasicController: CR040n BasicDisplay: CR0451 PDM360NG: CR108n, CR9042
`NetVarClib.LIB`	additional driver for network variables	BasicController: CR040n BasicDisplay: CR0451 PDM360NG: CR108n, CR9042
`ifm_J1939_Vxxyyzz.LIB`	J1939 communication POUs	up to target V04: CabinetController: CR0303 ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7200, CR7505 SmartController: CR2500 PDM360smart: CR1070, CR1071
`ifm_J1939_x_Vxxyyzz.LIB`	J1939 communication POUs	from target V05: CabinetController: CR0303 ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 SmartController: CR2500 PDM360smart: CR1070, CR1071
`ifm_CRnnnn_J1939_Vxxyyzz.LIB`	J1939 communication POUs	ClassicController: CR0032 ExtendedController: CR0232
`ifm_PDM_J1939_Vxxyyzz.LIB`	J1939 communication POUs	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`ifm_CANx_LAYER2_Vxxyyzz.LIB`	CAN POUs on the basis of layer 2: CAN transmit, CAN receive	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`ifm_CAN1E_Vxxyyzz.LIB`	changes the CAN bus from 11 bits to 29 bits	up to target V04: PDM360smart: CR1070, CR1071

File name	Function	Available for:
`ifm_CAN1_EXT_Vxxyyzz.LIB`	changes the CAN bus from 11 bits to 29 bits	from target V05: CabinetController: CR030n ClassicController: CR0020, CR0505 ExtendedController: CR0200 PCB controller: CS0015 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 SmartController: CR25nn PDM360smart: CR1070, CR1071
`ifm_CAMERA_O2M_Vxxyyzz.LIB`	camera POUs	PDM360: CR1051
`CR2013AnalogConverter.LIB`	analogue value conversion for I/O module CR2013	all ecomatmobile controllers PDM: CR10nn
`ifm_Hydraulic_16bitOS04_Vxxyyzz.LIB`	hydraulic POUs for R360 controllers	up to target V04: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7200, CR7505 SmartController: CR25nn
`ifm_Hydraulic_16bitOS05_Vxxyyzz.LIB`	hydraulic POUs for R360 controllers	from target V05: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SafetyController: CR7020, CR7021, CR7200, CR7201, CR7505, CR7506 SmartController: CR25nn
`ifm_Hydraulic_32bit_Vxxyyzz.LIB`	hydraulic POUs for R360 controllers	ClassicController: CR0032 ExtendedController: CR0232
`ifm_Hydraulic_CR0303_Vxxyyzz.LIB`	hydraulic POUs for R360 controllers	CabinetController: CR0303
`ifm_SafetyIO_Vxxyyzz.LIB`	safety POUs	SafetyController: CR7nnn
`ifm_PDM_UTIL_Vxxyyzz.LIB`	auxiliary functions PDM	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`ifm_PDMng_UTIL_Vxxyyzz.LIB`	auxiliary functions PDM	PDM360NG: CR1082, CR9042
`ifm_PDMsmart_UTIL_Vxxyyzz.LIB`	auxiliary functions PDM	PDM360smart: CR1070, CR1071
`ifm_PDM_Input_Vxxyyzz.LIB`	alternative input POUs PDM	PDM: CR10nn
`ifm_CR107n_Init_Vxxyyzz.LIB`	initialisation POUs PDM360smart	PDM360smart: CR1070, CR1071
`ifm_PDM_File_Vxxyyzz.LIB`	file POUs PDM360	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056 PDM360NG: CR1082, CR9042
`ifm_PDM360NG_linux_syscall_asynch_LIB`	send Linux commands to the system	PDM360NG: CR1082, CR9042
`ifm_PDM360NG_USB_Vxxyyzz.LIB`	manage devices at the USB interface	PDM360NG: CR1082, CR9042
`ifm_PDM360NG_USB_LL_Vxxyyzz.LIB`	auxiliary library for `ifm_PDM360NG_USB_Vxxyyzz.LIB`	PDM360NG: CR1082, CR9042
`Instrumente_x.LIB`	predefined display instruments	PDM: CR10nn

File name	Function	Available for:
`Symbols_x.LIB`	predefined symbols	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056
`Segment_x.LIB`	predefined 7-segment displays	PDM360: CR1050, CR1051, CR1060 PDM360compact: CR1052, CR1053, CR1055, CR1056

Further libraries on request.

14 Glossary of Terms

A

Address

This is the "name" of the bus participant. All participants need a unique address so that the signals can be exchanged without problem.

Application software

Software specific to the application, implemented by the machine manufacturer, generally containing logic sequences, limits and expressions that control the appropriate inputs, outputs, calculations and decisions

Necessary to meet the specific (→SRP/CS) requirements.
→ Programming language, safety-related

Architecture

Specific configuration of hardware and software elements in a system.

B

Baud

Baud, abbrev.: Bd = unit for the data transmission speed. Do not confuse baud with "bits per second" (bps, bits/s). Baud indicates the number of changes of state (steps, cycles) per second over a transmission length. But it is not defined how many bits per step are transmitted. The name baud can be traced back to the French inventor J. M. Baudot whose code was used for telex machines.

1 MBd = 1024 x 1024 Bd = 1 048 576 Bd

Bus

Serial data transmission of several participants on the same cable.

C

CAN

CAN = **C**ontroller **A**rea **N**etwork

CAN is a priority controlled fieldbus system for larger data volumes. It is available in different variants, e.g. "CANopen" or "CAN in Automation" (CiA).

CAN stack

CAN stack = stack of tasks for CAN data communication.

Category (CAT)

Classification of the safety-related parts of a control system in respect of their resistance to faults and their subsequent behaviour in the fault condition. This safety is achieved by the structural arrangement of the parts, fault detection and/or by their reliability.
(→ EN 954).

CCF

Common **C**ause **F**ailure
Failures of different items, resulting from a common event, where these failures are not consequences of each other.

CiA

CiA = CAN in Automation e.V.

User and manufacturer organisation in Germany / Erlangen. Definition and control body for CAN and CAN-based network protocols.

Homepage → http://www.can-cia.org

CiA DS 304

DS = **D**raft **S**tandard

CAN device profile CANopen safety for safety-related communication.

CiA DS 401

DS = **D**raft **S**tandard

CAN device profile for digital and analogue I/O modules

CiA DS 402

DS = **D**raft **S**tandard

CAN device profile for drives

CiA DS 403

DS = **D**raft **S**tandard

CAN device profile for HMI

CiA DS 404

DS = **D**raft **S**tandard

CAN device profile for measurement and control technology

CiA DS 405

DS = **D**raft **S**tandard

Specification for interface to programmable controllers (IEC 61131-3)

CiA DS 406

DS = **D**raft **S**tandard

CAN device profile for encoders

CiA DS 407

DS = **D**raft **S**tandard

CAN application profile for local public transport

Clamp 15

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

COB-ID

COB = **C**ommunication **Ob**ject
ID = **Id**entifier

Via the COB-ID the participants distinguish the different messages to be exchanged.

CoDeSys

CoDeSys® is a registered trademark of 3S – Smart Software Solutions GmbH, Germany.

"CoDeSys for Automation Alliance" associates companies of the automation industry whose hardware devices are all programmed with the widely used IEC 61131-3 development tool CoDeSys®.

Homepage → http://www.3s-software.com

CRC

CRC = **C**yclic **R**edundancy **C**heck

CRC is a method of information technology to determine a test value for data, to detect faults during the transmission or duplication of data.

Prior to the transmission of a block of data, a CRC value is calculated. After the end of the transaction the CRC value is calculated again at the target location. Then, these two test values are compared.

Cycle time

This is the time for a cycle. The PLC program performs one complete run.

Depending on event-controlled branchings in the program this can take longer or shorter.

D

DC

Direct **C**urrent

DC

Diagnostic **C**overage
Diagnostic coverage is the measure of the effectiveness of diagnostics as the ratio between the failure rate of detected dangerous failures and the failure rate of total dangerous failures:

Formula: DC = failure rate detected dangerous failures / total dangerous failures

Designation	Range
none	DC < 60 %
low	60 % < DC < 90 %
medium	90 % < DC < 99 %
high	99 % < DC

Table: Diagnostic coverage DC

An accuracy of 5 % is assumed for the limit values shown in the table.

Diagnostic coverage can be determined for the whole safety-related system or for only parts of the safety-related system.

Demand rate rd

The demand rate r_d is the frequency of demands to a safety-related reaction of an SRP/CS per time unit.

Diagnosis

During the diagnosis, the "state of health" of the device is checked. It is to be found out if and what faults are given in the device.

Depending on the device, the inputs and outputs can also be monitored for their correct function.
- wire break,
- short circuit,
- value outside range.

For diagnosis, configuration and log data can be used, created during the "normal" operation of the device.

The correct start of the system components is monitored during the initialisation and start phase. Errors are recorded in the log file.

For further diagnosis, self-tests can also be carried out.

Diagnostic coverage

Diagnostic **C**overage
Diagnostic coverage is the measure of the effectiveness of diagnostics as the ratio between the failure rate of detected dangerous failures and the failure rate of total dangerous failures:

Formula: DC = failure rate detected dangerous failures / total dangerous failures

Designation	Range
none	DC < 60 %
low	60 % < DC < 90 %
medium	90 % < DC < 99 %
high	99 % < DC

Table: Diagnostic coverage DC

An accuracy of 5 % is assumed for the limit values shown in the table.

Diagnostic coverage can be determined for the whole safety-related system or for only parts of the safety-related system.

Dither

Dither is a component of the PWM signals to control hydraulic valves. It has shown for electromagnetic drives of hydraulic valves that it is much easier for controlling the valves if the control signal (PWM pulse) is superimposed by a certain frequency of the PWM frequency. This dither frequency must be an integer part of the PWM frequency.
→ chapter What is the dither? (→ page 229)

Diversity

In technology diversity is a strategy to increase failure safety.

The systems are designed redundantly, however different implementations are used intentionally and not any individual systems of the same design. It is assumed that systems of the same performance, however of different implementation, are sensitive or insensitive to different interference and will therefore not fail simultaneously.

The actual implementation may vary according to the application and the requested safety:

- use of components of several manufacturers,
- use of different protocols to control devices,
- use of totally different technologies, for example an electrical and a pneumatic controller,
- use of different measuring methods (current, voltage),
- two channels with reverse value progression:
  channel A: 0...100 %
  channel B: 100...0 %

DRAM

DRAM = **D**ynamic **R**andom **A**ccess **M**emory

Technology for an electronic memory module with random access (Random Access Memory, RAM). The memory element is a capacitor which is either charged or discharged. It becomes accessible via a switching transistor and is either read or overwritten with new contents.

The memory contents are volatile: the stored information is lost in case of lacking operating voltage or too late restart.

DTC

DTC = **D**iagnostic **T**rouble **C**ode = error code
Faults and errors 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 motor

EDS-file

EDS = **E**lectronic **D**ata **S**heet, e.g. for:

- File for the object directory in the master
- CANopen device descriptions

Via EDS devices and programs can exchange their specifications and consider them in a simplified way.

Embedded software

System software, basic program in the device, virtually the operating system.

The firmware establishes the connection between the hardware of the device and the user software. This software is provided by the manufacturer of the controller as a part of the system and cannot be changed by the user.

EMCY

abbreviation for emergency

EMV

EMC = **E**lectro **M**agnetic **C**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.

Ethernet

Ethernet is a widely used, manufacturer-independent technology which enables data transmission in the network at a speed of 10 or 100 million bits per second (Mbps). Ethernet belongs to the family of so-called "optimum data transmission" on a non exclusive transmission medium. The concept was developed in 1972 and specified as IEEE 802.3 in 1985.

EUC

EUC = "Equipment Under Control"

EUC is equipment, machinery, apparatus or plant used for manufacturing, process, transportation, medical or other activities (→ IEC 61508-4, section 3.2.3). Therefore, the EUC is the set of all equipment, machinery, apparatus or plant that gives rise to hazards for which the safety-related system is required.

If any reasonably foreseeable action or inaction leads to hazards with an intolerable risk arising from the EUC, then safety functions are necessary to achieve or maintain a safe state for the EUC. These safety functions are performed by one or more safety-related systems.

F

Failure

Failure is the termination of the ability of an item to perform a required function.

After a failure, the item has a fault. Failure is an event, fault is a state.

The concept as defined does not apply to items consisting of software only.

Failure, dangerous

A dangerous failure has the potential to put the SRP/SC in a hazardous or fail-to-function state. Whether or not the potential is realized can depend on the channel architecture of the system; in redundant systems a dangerous hardware failure is less likely to lead to the overall dangerous or fail-to-function state.

Failure, systematic

A systematic failure is a failure related in a deterministic way (not coincidental) to a certain cause. The systematic failure can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors.

Corrective maintenance without modification of the system will usually not eliminate the failure cause.

Fault

A fault is the state of an item characterized by the inability to perform the requested function, excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources.

A fault is often the result of a failure of the item itself, but may exist without prior failure.

In ISO 13849-1 "fault" means "random fault".

Fault tolerance time

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

The max. cycle time of the application program (in the worst case 100 ms, → Watchdog (→ page 49)) and the possible delay and response times due to switching elements have to be considered.

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

FiFo

FiFo (**F**irst **I**n, **F**irst **O**ut) = operation of the stack: the data package which was written into a stack at first will be read at first too. For every identifier there is such one buffer (as a queue) available.

Firmware

System software, basic program in the device, virtually the operating system.

The firmware establishes the connection between the hardware of the device and the user software. This software is provided by the manufacturer of the controller as a part of the system and cannot be changed by the user.

First fault occurrence time

Time until the first failure of a safety element.

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

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

Flash memory

Flash ROM (or flash EPROM or flash memory) combines the advantages of semiconductor memory and hard disks. Just like every other semiconductor memory the flash memory does not require moving parts. And the data is maintained after switch-off, similar to a hard disk.

The flash ROM evolved from the EEPROM (**E**lectrical **E**rasable and **P**rogrammable **R**ead-**O**nly **M**emory). The storage function of data in the flash ROM is identical to the EEPROM. Similar to a hard disk, the data are however written and deleted blockwise in data blocks up to 64, 128, 256, 1024, ... bytes at the same time.

Advantages of flash memories

- The stored data are maintained even if there is no supply voltage.
- Due to the absence of moving parts, flash is noiseless and insensitive to shocks and magnetic fields.
- In comparison to hard disks, flash memories have a very short access time. Read and write speed are virtually constant across the entire memory area.
- The memory size that can be obtained has no upper limit, due to the simple and space-saving arrangement of the storage cells.

Disadvantages of flash memories

- A storage cell can tolerate a limited number of write and delete processes:
  - Multi-level cells: typ. 10 000 cycles
  - Single level cells: typ. 100 000 cycles
- Given that a write process writes memory blocks of between 16 and 128 Kbytes at the same time, memory cells which require no change are used as well.

FMEA

FMEA = **F**ailure **M**ode and **E**ffects **A**nalysis

Method of reliability engineering, to find potential weak points. Within the framework of quality or security management, the FMEA is used preventively to prevent faults and increase the technical reliability.

FRAM

FRAM, or also FeRAM, means **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.

Functional safety

Part of the overall safety referred to the →EUC and the EUC control system which depends on the correct functioning of the electric or electronic safety-related system, safety-related systems of other technologies and external devices for risk reduction.

H

Harm

Physical injury or damage to health.

Hazard

Hazard is the potential source of harm.

A distinction is made between the source of the hazard, e.g.:
- mechanical hazard,
- electrical hazard,
or the nature of the potential harm, e.g.:
- electric shock hazard,
- cutting hazard,
- toxic hazard.

The hazard envisaged in this definition is either permanently present during the intended use of the machine, e.g.:
- motion of hazardous moving elements,
- electric arc during a welding phase,
- unhealthy posture,
- noise emission,
- high temperature,
or the hazard may appear unexpectedly, e.g.:
- explosion,
- crushing hazard as a consequence of an unintended/unexpected start-up,
- ejection as a consequence of a breakage,
- fall as a consequence of acceleration/deceleration.

Heartbeat

The participants regularly send short signals. In this way the other participants can verify if a participant has failed. No master is necessary.

HMI

HMI = **H**uman **M**achine **I**nterface

I

ID

ID = **Id**entifier

Name to differentiate the devices / participants connected to a system or the message packets transmitted between the participants.

IEC user cycle

IEC user cycle = PLC cycle in the CoDeSys application program.

Instructions

Superordinate word for one of the following terms:
installation instructions, data sheet, user information, operating instructions, device manual, installation information, online help, system manual, programming manual, etc.

Intended use

Use of a product in accordance with the information provided in the instructions for use.

IP address

IP = **I**nternet **P**rotocol

The IP address is a number which is necessary to clearly identify an internet participant. For the sake of clarity the number is written in 4 decimal values, e.g. 127.215.205.156.

ISO 11898

Standard: "Road vehicles – Controller area network"

Part 1: "Data link layer and physical signalling"

Part 2: "High-speed medium access unit"

Part 3: "Low-speed, fault-tolerant, medium dependent interface"

Part 4: "Time-triggered communication"

Part 5: "High-speed medium access unit with low-power mode"

ISO 11992

Standard: "Interchange of digital information on electrical connections between towing and towed vehicles"

Part 1: "Physical and data-link layers"

Part 2: "Application layer for brakes and running gear"

Part 3: "Application layer for equipment other than brakes and running gear"

Part 4: "Diagnostics"

ISO 16845

Standard: "Road vehicles – Controller area network (CAN) – Conformance test plan"

L

LED

LED = **L**ight **E**mitting **D**iode

Light emitting diode, also called luminescent diode, an electronic element of high coloured luminosity at small volume with negligible power loss.

Life, mean

Mean **T**ime **T**o **F**ailure (MTTF) or: mean life.

The $MTTF_d$ is the expectation of the mean time to dangerous failure.

Designation	Range
low	3 years < $MTTF_d$ < 10 years
medium	10 years < $MTTF_d$ < 30 years
high	30 years < $MTTF_d$ < 100 years

Table: Mean time of each channel to the dangerous failure $MTTF_d$

Link

A link is a cross-reference to another part in the document or to an external document.

LSB

Least **S**ignificant **B**it/Byte

M

MAC-ID

MAC = **M**anufacturer‘s **A**ddress **C**ode
= manufacturer's serial number

→ID = **Id**entifier

Every network card has a MAC address, a clearly defined worldwide unique numerical code, more or less a kind of serial number. Such a MAC address is a sequence of 6 hexadecimal numbers, e.g. "00-0C-6E-D0-02-3F".

Master

Handles the complete organisation on the bus. The master decides on the bus access time and polls the →slaves cyclically.

Mission time TM

Mission time T_M is the period of time covering the intended use of an SRP/CS.

Misuse

The use of a product in a way not intended by the designer.

The manufacturer of the product has to warn against readily predictable misuse in his user information.

MMI

HMI = **H**uman **M**achine **I**nterface
→ HMI (→ page 334)

Monitoring

Safety function which ensures that a protective measure is initiated:

- if the ability of a component or an element to perform its function is diminished.
- if the process conditions are changed in such a way that the resulting risk increases.

MRAM

MRAM means **M**agnetoresistive **R**andom **A**ccess **M**emory. The information is stored by means of magnetic storage elements. The property of certain materials is used to change their electrical resistance when exposed to magnetic fields.

Advantages of MRAM as compared to conventional RAM memories:

- non volatile (like FRAM), but:
- access time only approx. 35 ns,
- unlimited number of access cycles possible.

MSB

Most **S**ignificant **B**it/Byte

MTBF

Mean **T**ime **B**etween **F**ailures (MTBF)
Is the expected value of the operating time between two consecutive failures of items that are maintained.

NOTE: For items that are NOT maintained the mean life →MTTF is the expected value (mean value) of the distribution of lives.

MTTF

Mean **T**ime **T**o **F**ailure (MTTF) or: mean life.

MTTFd

Mean **T**ime **T**o **F**ailure (MTTF) or: mean life.

The $MTTF_d$ is the expectation of the mean time to dangerous failure.

Designation	Range
low	3 years < $MTTF_d$ < 10 years
medium	10 years < $MTTF_d$ < 30 years
high	30 years < $MTTF_d$ < 100 years

Table: Mean time of each channel to the dangerous failure $MTTF_d$

Muting

Muting is the temporary automatic suspension of a safety function(s) by the SRP/CS.

Example: The safety light curtain is bridged, if the closing tools have reached a finger-proof distance to each other. The operator can now approach the machine without any danger and guide the workpiece.

N

NMT

NMT = **N**etwork **M**anagemen**t** = (here: in the CAN bus)

The NMT master controls the operating states of the NMT slaves.

Node

This means a participant in the network.

Node Guarding

Network participant

Configurable cyclic monitoring of each slave configured accordingly. The master 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.

Operating system

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

Operational

Operating state of a CANopen participant. In this mode SDOs, NMT commands and PDOs can be transferred.

P

PC card

→PCMCIA card

PCMCIA card

PCMCIA = Personal Computer Memory Card International Association, a standard for expansion cards of mobile computers.
Since the introduction of the cardbus standard in 1995 PCMCIA cards have also been called PC card.

PDM

PDM = **P**rocess and **D**ialogue **M**odule

Device for communication of the operator with the machine / plant.

PDO

PDO = **P**rocess **D**ata **O**bject

The time-critical process data is transferred by means of the "process data objects" (PDOs). The PDOs can be freely exchanged between the individual nodes (PDO linking). In addition it is defined whether data exchange is to be event-controlled (asynchronous) or synchronised. Depending on the type of data to be transferred the correct selection of the type of transmission can lead to considerable relief for the CAN bus.

These services are not confirmed by the protocol, i.e. it is not checked whether the message reaches the receiver. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

PDU

PDU = **P**rotocol **D**ata **U**nit

The PDU is an item of the CAN protocol SAE J1939. PDU indicates a part of the destination or source address.

Performance Level

Performance **L**evel
According to ISO 13849-1, a specification (PL a...e) of safety-related parts of control systems to perform a safety function under foreseeable conditions.

PES

Programmable **E**lectronic **S**ystem
A programmable electronic system is a system ...
- for control, protection or monitoring,
- dependent for its operation on one or more programmable electronic devices,
- including all elements of the system such as input and output devices.

PGN

PGN = **P**arameter **G**roup **N**umber
PGN = PDU format (PF) + PDU source (PS)

The parameter group number is an item of the CAN protocol SAE J1939. PGN collects the address parts PF and PS.

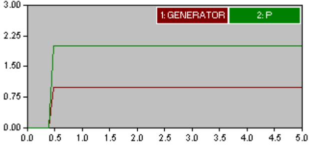
Pictogram

Pictograms are figurative symbols which convey information by a simplified graphic representation.

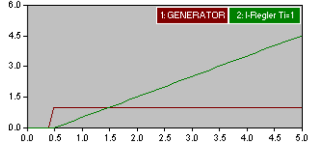
→ Chapter What do the symbols and formats mean? (→ page 7)

PID controller

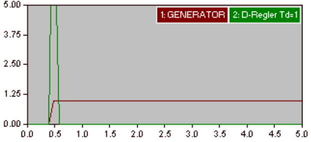
P = proportional part
The P controller exclusive consists of a proportional part of the amplification K_p. The output signal is proportional to the input signal.



I = integral part
An I controller acts to the manipulating variable by phasing integration of the control deviation with emphasis on the reset time T_N.



D = differential part
The D controller doesn't react on the control deviation but only on their speed of change.



PL

Performance **L**evel
According to ISO 13849-1, a specification (PL a...e) of safety-related parts of control systems to perform a safety function under foreseeable conditions.

PLC configuration

Part of the CoDeSys user interface.

► The programmer tells the programming system which hardware is to be programmed.

> CoDeSys loads the corresponding libraries.

> Reading and writing the peripheral states (inputs/outputs) is possible.

PLr

Using the "required performance level" PL_r the risk reduction for each safety function according to ISO 13849 is achieved.

For each selected safety function to be carried out by a SRP/CS, a PL_r shall be determined and documented. The determination of the PL_r is the result of the risk assessment and refers to the amount of the risk reduction.

Pre-Op

Pre-Op = PRE-OPERATIONAL mode

Operating status of a CANopen participant. After application of the supply voltage each participant automatically passes into this state. In the CANopen network only SDOs and NMT commands can be transferred in this mode but no process data.

prepared

Operating status of a CANopen participant. In this mode only NMT commands are transferred.

Process image

Process image is the status of the inputs and outputs the PLC operates with within one cycle.

- At the beginning of the cycle the PLC reads the conditions of all inputs into the process image.
  During the cycle the PLC cannot detect changes to the inputs.
- During the cycle the outputs are only changed virtually (in the process image).
- At the end of the cycle the PLC writes the virtual output states to the real outputs.

Programming language, safety-related

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

- Limited variability language (LVL) that provides the capability of combining predefined, application-specific library functions.
  In CoDeSys these are LD (ladder diagram) and FBD (function block diagram).
- Full variability language (FVL) provides the capability of implementing a wide variety of functions.
  These include e.g. C, C++, Assembler. In CoDeSys it is ST (structured text).

► Structured text is recommended exclusively in separate, certified functions, usually in embedded software.

► In the "normal" application program only LD and FBD should be used. The following minimum requirements shall be met.

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

► Modular and clear structure of the program. Consequence: simple testability.

► Functions are represented in a comprehensible manner:
- for the operator on the screen (navigation)
- readability of a subsequent print of the document.

► Use symbolic variables (no IEC addresses).

► Use meaningful variable names and comments.

► Use easy functions (no indirect addressing, no variable fields).

► Defensive programming.

► Easy extension or adaptation of the program possible.

Protective measure

Measure intended to achieve risk reduction, e.g.:

- fault-excluding design,
- safeguarding measures (guards),
- complementary protective measures (user information),
- personal protective equipment (helmet, protective goggles).

PWM

PWM = pulse width modulation

Via PWM a digital output (capability provided by the device) can provide an almost analogue voltage by means of regular fast pulses. The PWM output signal is a pulsed signal between GND and supply voltage.

Within a defined period (PWM frequency) the mark-to-space ratio is varied. Depending on the mark-to-space ratio, the connected load determines the corresponding RMS current.
→ chapter PWM signal processing (→ page 214)
→ chapter What does a PWM output do? (→ page 228)

R

Ratio

Measurements can also be performed ratiometrically. The input signal generates an output signal which is in a defined ratio to the input signal. This means that analogue input signals can be evaluated without additional reference voltage. A fluctuation of the supply voltage has no influence on this measured value.
→ Chapter Counter functions (→ page 198)

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.

Redundant

Redundancy is the presence of more than the necessary means so that a function unit performs a requested function or that data can represent information.

Several kinds of redundancy are distinguished:

- Functional redundancy aims at designing safety-related systems in multiple ways in parallel so that in the event of a failure of one component the others ensure the task.
- In addition it is tried to separate redundant systems from each other with regard to space. Thus the risk that they are affected by a common interference is minimised.
- Finally, components from different manufacturers are sometimes used to avoid that a systematic fault causes all redundant systems to fail (diverse redundancy).

The software of redundant systems should differ in the following aspects:

- specification (different teams),
- specification language,
- programming (different teams),
- programming language,
- compiler.

Remanent

Remanent data is protected against data loss in case of power failure.

The operating system for example automatically copies the remanent data to a flash memory as soon as the voltage supply falls below a critical value. If the voltage supply is available again, the operating system loads the remanent data back to the RAM memory.

The data in the RAM memory of a controller, however, is volatile and normally lost in case of power failure.

Reset, manual

The manual reset is an internal function within the SRP/CS used to restore manually one or more safety functions before re-starting a machine.

Residual risk

Risk remaining after protective measures have been taken. The residual risk has to be clearly warned against in operating instructions and on the machine.

Risk

Combination of the probability of occurrence of harm and the severity of that harm.

Risk analysis

Combination of ...

- the specification of the limits of the machine (intended use, time limits),
- hazard identification (intervention of people, operating status of the machine, foreseeable misuse) and
- the risk estimation (degree of injury, extent of damage, frequency and duration of the risk, probability of occurrence, possibility of avoiding the hazard or limiting the harm).

Risk assessment

Overall process comprising risk analysis and risk evaluation.

According to Machinery Directive 2006/42/EU the following applies: "The manufacturer of machinery or his authorised representative must ensure that a risk assessment is carried out in order to determine the health and safety requirements which apply to the machinery. The machinery must then be designed and constructed taking into account the results of the risk assessment." (→ Annex 1, General principles)

Risk evaluation

Judgement, on the basis of the risk analysis, of whether risk reduction objectives have been achieved.

ro

RO = read only for reading only

Unidirectional data transmission: Data can only be read and not changed.

RTC

RTC = **R**eal **T**ime **C**lock

Provides (batter-backed) the current date and time. Frequent use for the storage of error message protocols.

rw

RW = read/ write

Bidirectional data transmission: Data can be read and also changed.

S

SAE J1939

The network protocol SAE J1939 describes the communication on a CAN bus in commercial vehicles for transmission of diagnosis data (e.g.motor speed, temperature) and control information.
→ CiA DS 402

Standard: "Recommended Practice for a Serial Control and Communications Vehicle Network"

Part 2: "Agricultural and Forestry Off-Road Machinery Control and Communication Network"

Part 3: "On Board Diagnostics Implementation Guide"

Part 5: "Marine Stern Drive and Inboard Spark-Ignition Engine On-Board Diagnostics Implementation Guide"

Part 11: "Physical Layer – 250 kBits/s, Shielded Twisted Pair"

Part 13: "Off-Board Diagnostic Connector"

Part 15: "Reduced Physical Layer, 250 kBits/s, Un-Shielded Twisted Pair (UTP)"

Part 21: "Data Link Layer"

Part 31: "Network Layer"

Part 71: "Vehicle Application Layer"

Part 73: "Application Layer – Diagnostics"

Part 81: "Network Management Protocol"

Safety function

Function of the machine whose failure can result in an immediate increase of the risk(s). The designer of such a machine therefore has to:
- safely prevent a failure of the safety function,
- reliably detect a failure of the safety function in time,
- bring the machine into a safe state in time in the event of a failure of the safety function.

Safety-standard types

The safety standards in the field of machines are structured as below:

Type-A standards (basic safety standards) giving basic concepts, principles for design, and general aspects that can be applied to all machinery. Examples: basic terminology, methodology (ISO 12100-1), technical principles (ISO 12100-2), risk assessment (ISO 14121), ...

Type-B standards (generic safety standards) dealing with one safety aspect or one type of safeguard that can be used across a wide range of machinery.

- Type-B1 standards on particular safety aspects. Examples: safety distances (EN 294), hand/arm speeds (EN 999), safety-related parts of control systems (ISO 13849), temperatures, noise, ...
- Type-B2 standards on safeguards. Examples: emergency stop circuits ((ISO 13850), two-hand controls, interlocking devices or electro-sensitive protective equipment (ISO 61496), ...

Type-C standards (machine safety standards) dealing with detailed safety requirements for a particular machine or group of machines.

SCT

In CANopen safety the **S**afeguard **C**ycle **T**ime (SCT) monitors the correct function of the periodic transmission (data refresh) of the SRDOs. The data must have been repeated within the set time to be valid. Otherwise the receiving controller signals a fault and passes into the safe state (= outputs switched off).

SD card

An SD memory card (short for **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.

SDO is a specification for a manufacturer-dependent data structure for standardised data access. "Clients" ask for the requested data from "servers". The SDOs always consist of 8 bytes. Longer data packages are distributed to several messages.

Examples:

- Automatic configuration of all slaves via SDOs at the system start,
- reading error messages from the object directory.

Every SDO is monitored for a response and repeated if the slave does not respond within the monitoring time.

Self-test

Test program that actively tests components or devices. The program is started by the user and takes a certain time. The result is a test protocol (log file) which shows what was tested and if the result is positive or negative.

SIL

According to IEC 62061 the safety-integrity level SIL is a classification (SIL CL 1...4) of the safety integrity of the safety functions. It is used for the evaluation of electrical/electronic/programmable electronic (E/E/EP) systems with regard to the reliability of safety functions. The safety-related design principles that have to be adhered to so that the risk of a malfunction can be minimised result from the required level.

Slave

Passive participant on the bus, only replies on request of the →master. Slaves have a clearly defined and unique →address in the bus.

SRDO

Safe data is exchanged via SRDOs (**S**afety-**R**elated **D**ata **O**bjects). An SRDO always consists of two CAN messages with different identifiers:

- message 1 contains the original user data,
- message 2 contains the same data which are inverted bit by bit.

SRP/CS

Safety-**R**elated **P**art of a **C**ontrol **S**ystem

Part of a control system that responds to safety-related input signals and generates safety-related output signals.

The combined safety-related parts of a control system start at the point where the safety-related input signals are initiated (including, for example, the actuating cam and the roller of the position switch) and end at the output of the power control elements (including, for example, the main contacts of a contactor).

SRVT

The SRVT (**S**afety-**R**elated **O**bject **V**alidation **T**ime) ensures with CANopen safety that the time between the SRDO-message pairs is adhered to.

Only if the redundant, inverted message has been transmitted after the original message within the SRVT set are the transmitted data valid. Otherwise the receiving controller signals a fault and will pass into the safe state (= outputs switched off).

State, safe

The state of a machine is said to be safe when there is no more hazard formed by it. This is usually the case if all possible dangerous movements are switched off and cannot start again unexpectedly.

Symbols

Pictograms are figurative symbols which convey information by a simplified graphic representation.

→ Chapter What do the symbols and formats mean? (→ page 7)

System variable

Variable to which access can be made via IEC address or symbol name from the PLC.

T

Target

The target indicates the target system where the PLC program is to run. The target contains the files (drivers and if available specific help files) required for programming and parameter setting.

TCP

The **T**ransmission **C**ontrol **P**rotocol is part of the TCP/IP protocol family. Each TCP/IP data connection has a transmitter and a receiver. This principle is a connection-oriented data transmission. In the TCP/IP protocol family the TCP as the connection-oriented protocol assumes the task of data protection, data flow control and takes measures in the event of data loss.

(compare: →UDP)

Template

A template can be filled with content.
Here: A structure of pre-configured software elements as basis for an application program.

Test rate rt

The test rate r_t is the frequency of the automatic tests to detect errors in an SRP/CS in time.

U

UDP

UDP (**U**ser **D**atagram **P**rotocol) is a minimal connectionless network protocol which belongs to the transport layer of the internet protocol family. The task of UDP is to ensure that data which is transmitted via the internet is passed to the right application.

At present network variables based on CAN and UDP are implemented. The values of the variables are automatically exchanged on the basis of broadcast messages. In UDP they are implemented as broadcast messages, in CAN as PDOs. These services are not confirmed by the protocol, i.e. it is not checked whether the message is received. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

Uptime, mean

Mean **T**ime **B**etween **F**ailures (MTBF)
Is the expected value of the operating time between two consecutive failures of items that are maintained.

NOTE: For items that are NOT maintained the mean life →MTTF is the expected value (mean value) of the distribution of lives.

Use, intended

Use of a product in accordance with the information provided in the instructions for use.

W

Watchdog

In general the term watchdog is used for a component of a system which watches the function of other components. If a possible malfunction is detected, this is either signalled or suitable program branchings are activated. The signal or branchings serve as a trigger for other co-operating system components to solve the problem.

wo

WO = write only

Unidirectional data transmission: Data can only be changed and not read.

15 Index

A distinction is made between the following errors: 185
About the ifm templates 27
About this manual 7
Above-average stress 48
Access to the CANopen slave at runtime 149
Access to the OD entries by the application program 149
Access to the status of the CANopen master 139
Access to the structures at runtime of the application 166
Activating the PLC configuration 22
Adapting analogue values 195
Add and configure CANopen slaves 127
Address 329
Address assignment and I/O operating modes 299
Address assignment inputs / outputs 300
Addresses / variables of the I/Os 302
Analogue input group A_IN00...29 (%IW2...31) 42
Analogue inputs 41
Annex 299
Application software 329
Applications 198
Architecture 329
Automatic configuration of slaves 132
Automatic data backup 271
Availability of PWM 213
Available CAN interfaces and CAN protocols 61
Available memory 50
Base settings
  Bus identifier 143
  Generate EDS file 143
  Name of updatetask 143
Baud 329
Boot up of the CANopen master 134
Boot up of the CANopen slaves 135
Bootup messsage 311
Bus 329
Bus cable length 67
Calculation examples RELOAD value 217
Calculation of the RELOAD value 216
CAN 329
CAN bus level 66
CAN bus level according to ISO 11992-1 66
CAN errors 182
CAN errors and error handling 182
CAN for the drive engineering 100
CAN interfaces 61
CAN parameters
  Automatic startup 126
  Baud rate 124
  Communication cycle 129
  Communication Cycle Period/Sync. Window Length 125
  Create all SDOs 128
  Emergency telegram 129
  Heartbeat 126
  No initialization 128
  Node ID 125, 128
  Node reset 128
  Nodeguarding / heartbeat settings 129
  Optional device 128
  Sync. COB ID 125
  Write DCF 128
CAN stack 329
CAN units acc. to SAE J1939 99
CAN1_BAUDRATE 75
CAN1_DOWNLOADID 77
CAN1_ERRORHANDLER 97
CAN1_EXT 79
CAN1_EXT_ERRORHANDLER 85
CAN1_EXT_RECEIVE 83
CAN1_EXT_RECEIVE_ALL 95
CAN1_EXT_TRANSMIT 81
CAN1_MASTER_EMCY_HANDLER 157
CAN1_MASTER_SEND_EMERGENCY 159
CAN1_MASTER_STATUS 162
CAN1_RECEIVE 90
CAN1_RECEIVE_RANGE 92
CAN1_SDO_READ 178
CAN1_SDO_WRITE 180
CAN1_SLAVE_EMCY_HANDLER 169
CAN1_SLAVE_NODEID 168
CAN1_SLAVE_SEND_EMERGENCY 171
CAN1_SLAVE_STATUS 174
CAN1_TRANSMIT 88
CAN2 86
CAN2_ERRORHANDLER 97
CAN2_EXT_RECEIVE_ALL 95
CAN2_RECEIVE 90
CAN2_RECEIVE_RANGE 92
CAN2_TRANSMIT 88
CAN-ID 70
CANopen error code 316
CANopen master 121
  Tab [CAN parameters] 124
CANopen network configuration, status and error handling 118
CANopen network variables 150
CANopen slave 142
  Tab [CAN parameters] 128
CANopen slave configuration 143
CANopen status of the node 137, 315
CANopen support by CoDeSys 119
CANopen tables 306
CANopen terms and implementation 119
CANx_ERRORHANDLER 97
CANx_EXT_RECEIVE_ALL 95
CANx_MASTER_EMCY_HANDLER 157
CANx_MASTER_SEND_EMERGENCY 159
CANx_MASTER_STATUS 162
CANx_RECEIVE 90
CANx_RECEIVE_RANGE 92
CANx_SDO_READ 178
CANx_SDO_WRITE 180
CANx_SLAVE_EMCY_HANDLER 169
CANx_SLAVE_NODEID 168
CANx_SLAVE_SEND_EMERGENCY 171
CANx_SLAVE_STATUS 174
CANx_TRANSMIT 88
Category (CAT) 329
CCF 329
Certification and distribution of the safety-related software 55
Change the PDO properties at runtime 149
Changes of the manual (CA) 9
Changing the safety-relevant software after certification 55
Changing the standard mapping by the master configuration 148
CHECK_DATA 290
CiA 329
CiA DS 304 329
CiA DS 401 329
CiA DS 402 330
CiA DS 403 330

CiA DS 404 ... 330
CiA DS 405 ... 330
CiA DS 406 ... 330
CiA DS 407 ... 330
Clamp 15 ... 330
COB-ID ... 330
CoDeSys ... 330
CoDeSys CANopen libraries ... 324
Communication via interfaces ... 259
Configuration of all correctly detected devices ... 131
Configuration of CANopen network variables ... 151
Configurations ... 20
Configure inputs ... 38
Configure outputs ... 43
Controlled system with delay ... 246
Controlled system without inherent regulation ... 246
Controller functions ... 245
Counter functions for frequency and period measurement ... 198
CPU frequency ... 47
CRC ... 330
Create a CANopen project ... 123
Creating application program ... 52
Current measurement on the high-current outputs ... 44
Cycle time ... 330
Cyclical transmission of the SYNC message ... 132
Damping of overshoot ... 247
Data access and data check ... 282
Data reception ... 72
Data transmission ... 72
DC ... 330
DEBUG mode ... 19
DELAY ... 249
Demand rate rd ... 331
Demo program for controller ... 34
Demo programs for PDM and BasicDisplay ... 36
Description of the CAN standard program units ... 73
Diagnosis ... 331
Diagnostic coverage ... 331
Differentiation from other CANopen libraries ... 121
Digital and PWM outputs ... 43
Digital inputs ... 39
Dither ... 331
Dither frequency and amplitude ... 230
Diversity ... 331
DRAM ... 331
DTC ... 332
ECU ... 332
EDS-file ... 332
Embedded software ... 332
EMCY ... 332
EMCY error code ... 186
Emergency messages ... 316
EMV ... 332
Error codes and diagnostic information ... 56
Error counter ... 183
Error message ... 182
Ethernet ... 332
EUC ... 332
Example
  CANx_MASTER_SEND_EMERGENCY ... 161
  CANx_MASTER_STATUS ... 165
  CANx_SLAVE_SEND_EMERGENCY ... 173
  CHECK_DATA ... 291
  detailed message documentation ... 102
  Initialisation of CANx_RECEIVE_RANGE in 4 cycles ... 94
  list of variables ... 147
  NORM_HYDRAULIC ... 244
  short message documentation ... 103
Example 1 ... 197
Example 2 ... 197
Example Dither ... 230
Example of an object directory ... 144
Example process for response to a system error ... 58
Exchange of CAN data ... 69
Failure ... 332
Failure, dangerous ... 332
Failure, systematic ... 333
Fast inputs ... 40
FAST_COUNT ... 211
Fatal error ... 16
Fault ... 333
Fault tolerance time ... 333
FB, FUN, PRG in CoDeSys ... 51
FiFo ... 333
Files for the operating system / runtime system ... 322
Firmware ... 333
First fault occurrence time ... 333
Flash memory ... 333
FLASHREAD ... 276
FLASHWRITE ... 274
FMEA ... 334
Folder structure in general ... 27
FRAM ... 334
FRAMREAD ... 280
FRAMWRITE ... 278
FREQUENCY ... 200
Function code / Predefined Connectionset ... 308
Function configuration of the inputs and outputs ... 38
Functional safety ... 334
Functionality of the CANopen slave library ... 142
Functions for controllers ... 248
Functions of the library ifm_HYDRAULIC_CR0303 ... 231
Further ifm libraries for CANopen ... 177
General ... 245
General about CAN ... 59
General information ... 150
General information about CANopen with CoDeSys ... 119
General overview ... 320
GET_IDENTITY ... 286
Global variable list
  Acknowledgement ... 154
  Cyclic transmission ... 154
  List identifier (COB-ID) ... 153
  Network type ... 153
  Pack variables ... 153
  Read ... 154
  Transmit checksum ... 154
  Transmit on change ... 154
  Transmit on event ... 154
  Write ... 154
GLR ... 257
Harm ... 334
Hazard ... 334
Heartbeat ... 334
Heartbeat from the master to the slaves ... 132
Hints ... 70
Hints to wiring diagrams ... 45
HMI ... 334
How is this manual structured? ... 8
Hydraulic control in PWMi ... 227
ID ... 334
Identifier ... 186
Identifier acc. to SAE J1939 ... 101

IDs (addresses) in CANopen 120, 306
IEC user cycle 334
ifm CANopen libraries 117
ifm CANopen libraries master / slave 323
ifm demo programs 34
ifm device libraries 323
ifm library for the CANopen master 156
ifm library for the CANopen slave 167
Important! 10
In-/output functions 190
INC_ENCODER 208
Information concerning the device 12
Information concerning the software 12
INIT state (Reset) 15
Initialisation of the network with RESET_ALL_NODES 139
Input group IN00...IN15 (%IX0.0...%IX0.15) 42
INPUT_ANALOG 191
INPUT_CURRENT 194
INPUT_VOLTAGE 193
Installation of the files and libraries 319
Instructions 335
Intended use 335
IP address 335
ISO 11898 335
ISO 11992 335
ISO 16845 335
J1939_1 105
J1939_1_GLOBAL_REQUEST 115
J1939_1_RECEIVE 107
J1939_1_RESPONSE 111
J1939_1_SPECIFIC_REQUEST 113
J1939_1_TRANSMIT 109
J1939_2 105
J1939_2_GLOBAL_REQUEST 115
J1939_2_RECEIVE 107
J1939_2_RESPONSE 111
J1939_2_SPECIFIC_REQUEST 113
J1939_2_TRANSMIT 109
J1939_x 105
J1939_x_GLOBAL_REQUEST 115
J1939_x_RECEIVE 107
J1939_x_RESPONSE 111
J1939_x_SPECIFIC_REQUEST 113
J1939_x_TRANSMIT 109
JOYSTICK_0 232
JOYSTICK_1 235
JOYSTICK_2 239
LED 335
Libraries for CANopen 156
Life, mean 335
Limitations and programming notes 47
Limits of the device 47
Link 335
Load the operating system 18
LSB 335
MAC-ID 335
Managing the data 266
Manual data storage 272
Manufacturer specific information 188
Master 336
Master at runtime 131
MEMCPY 273
Mission time TM 336
Misuse 336
MMI 336
Monitoring 336
MRAM 336
MSB 336
MTBF 336
MTTF 336
MTTFd 336
Muting 336
Network management (NMT) 312
Network management commands 312
Network states 134
Network structure 65
NMT 336
NMT state 313
NMT state for CANopen master 135, 313
NMT state for CANopen slave 136, 314
No operating system 16
Node 336
Node Guarding 337
Nodeguarding with lifetime monitoring 132
NORM 196
NORM_HYDRAULIC 242
Note the cycle time! 52
Notes on devices with monitoring relay 57
Obj / object 337
Object 0x1001 (error register) 188, 318
Object 0x1003 (error field) 186
Object directory 337
OBV 337
Operating modes 19
Operating states 15
Operating states and operating system 15
Operating system 337
Operational 337
Optimising the PLC cycle 292
Output groups OUT0...OUT11 or OUT0...OUT17
(%QX0.0...%QX0.11/0.17) 44
Overview 56
Overview CANopen EMCY codes (CR030n) 189
Overview CANopen error codes 187, 317
Overview of the files and libraries used 319
Parameters of internal structures 164
Participant, bus off 184
Participant, error active 183
Participant, error passive 183
Particularities for network variables 155
PC card 337
PCMCIA card 337
PDM 337
PDO 337
PDO-Mapping
    Insert 129
    Properties 130
PDU 337
Performance Level 337
PERIOD 202
PERIOD_RATIO 204
PES 337
PGN 338
PHASE 206
Physical connection of CAN 65
Pictogram 338
PID controller 338
PID1 252
PID2 254
PL 338
PLC configuration 14, 338
PLC configuration file 322

PLr ... 338
Polling of the slave device type ... 131
Possible operating modes inputs / outputs ... 304
Pre-Op ... 338
prepared ... 338
Process image ... 338
Processing input values ... 190
Processing interrupts ... 292
Programming language, safety-related ... 339
Programming notes for CoDeSys projects ... 51
Programs and functions in the folders of the templates ... 28
Protective measure ... 339
PT1 ... 251
PWM ... 221, 339
PWM – introduction ... 214
PWM / PWM1000 ... 215
PWM channels 0...3 ... 216
PWM channels 4...7 / 8...11 ... 218
PWM dither ... 220
PWM frequency ... 215
PWM functions ... 213
PWM functions and their parameters ... 215
PWM signal processing ... 214
PWM100 ... 223
PWM1000 ... 225
Ramp function ... 220
Ratio ... 339
RAW-CAN ... 339
Reading / writing the system time ... 268
Reception of emergency messages ... 132
Recommended setting ... 256
Recommended settings ... 253
Redundant ... 339
Remanent ... 340
Reset of all configured slaves on the bus at the system start ... 131
Reset, manual ... 340
Residual risk ... 340
Response to the system error ... 57
Risk ... 340
Risk analysis ... 340
Risk assessment ... 340
Risk evaluation ... 340
ro ... 340
RTC ... 340
RUN state ... 16
rw ... 341
SAE J1939 ... 341
Safety function ... 341
Safety instructions ... 10
Safety-standard types ... 341
Save ... 54
Saving, reading and converting data in the memory ... 271
SCT ... 341
SD card ... 341
SDO ... 341
SDO abort code ... 310
SDO command bytes ... 309
Self-regulating process ... 245
Self-test ... 342
SERIAL_MODE ... 19
SERIAL_PENDING ... 265
SERIAL_RX ... 263
SERIAL_SETUP ... 260
SERIAL_TX ... 262
Set up programming system ... 20
Set up programming system manually ... 20
Set up programming system via templates ... 24
SET_DEBUG ... 19, 283
SET_IDENTITY ... 284
SET_INTERRUPT_I ... 296
SET_INTERRUPT_XMS ... 293
SET_PASSWORD ... 288
Setting control ... 247
Setting of the node numbers and the baud rate of a CANopen slave ... 149
Setting rule for a controller ... 247
Setting the LED via application program ... 17
Settings in the global variable lists ... 152
Settings in the target settings ... 151
Setup the target ... 21
Signalling of device errors ... 186
SIL ... 342
Slave ... 342
Slave information ... 165
SOFTRESET ... 267
Software reset ... 266
Specific ifm libraries ... 325
SRDO ... 342
SRP/CS ... 342
SRVT ... 342
Standardise the output signals of a joystick ... 227
Start of all correctly configured slaves ... 132
Start the network ... 133
Starting the network with GLOBAL_START ... 138, 163
Starting the network with START_ALL_NODES ... 138
Start-up of the network without [Automatic startup] ... 138
State, safe ... 342
Status LED ... 16
STOP state ... 15
Structure Emergency_Message ... 166
Structure node status ... 165
Structure of an EMCY message ... 185
Structure of an error message ... 185
Structure of CANopen messages ... 307
Structure of the COB ID ... 307
Structure of the visualisations in the templates ... 30
Summary CAN / CANopen ... 71
Supplement project with further functions ... 31
Symbols ... 342
System configuration ... 63
System description ... 12
System flags ... 305
System variable ... 342
Tab [Base settings] ... 143
Tab [CAN settings] ... 145
Tab [Default PDO mapping] ... 146
Tab [Receive PDO-Mapping] and [Send PDO-Mapping] ... 129
Tab [Service Data Objects] ... 130
Target ... 342
Target file ... 322
TCP ... 343
Technical about CANopen ... 118
Template ... 343
TEST mode ... 19
Test rate rt ... 343
The object directory of the CANopen master ... 140
The purpose of this library? – An introduction ... 227
TIMER_READ ... 269
TIMER_READ_US ... 270
Topology ... 60

Transmit emergency messages via the
application program 149
UDP 343
Units for SAE J1939 104
Uptime, mean 343
Use as digital inputs 199
Use of the serial interface 19, 259
Use, intended 343
Using CAN 59
Using ifm downloader 54
Watchdog 343
Watchdog behaviour 49
What are the individual files and libraries used for? 322
What do the symbols and formats mean? 7
What does a PWM output do? 228
What is the dither? 229
What previous knowledge is required? 11
When is a dither useful? 229
Wire cross-sections 68
wo 343