Contents

1 About this manual .................................................................................................................. 5
  1.1 Copyright ......................................................................................................................... 5
  1.2 Overview: documentation modules for ecomatmobile devices .................................. 6
  1.3 CODESYS programming manual .............................................................................. 6
  1.4 What do the symbols and formats mean? .................................................................. 7
  1.5 How is this documentation structured? ...................................................................... 8
  1.6 History of the instructions (CR0020, CR0505) .................................................... 9

2 Safety instructions ........................................................................................................... 10
  2.1 Please note! .................................................................................................................... 10
  2.2 What previous knowledge is required? ..................................................................... 11
  2.3 Start-up behaviour of the controller ....................................................................... 11

3 System description ......................................................................................................... 12
  3.1 Information about the device ..................................................................................... 12
  3.2 Hardware description ............................................................................................... 12
    3.2.1 Hardware structure .............................................................................................. 13
    3.2.2 Operating principle of the delayed switch-off .................................................... 16
    3.2.3 Relays: important notes! ..................................................................................... 17
    3.2.4 Monitoring concept ............................................................................................ 18
    3.2.5 Inputs (technology) ............................................................................................. 21
    3.2.6 Outputs (technology) .......................................................................................... 26
    3.2.7 Note on wiring .................................................................................................... 29
    3.2.8 Safety instructions about Reed relays ............................................................. 29
    3.2.9 Feedback on bidirectional inputs/outputs ....................................................... 30
    3.2.10 Feedback in case of externally supplied outputs ........................................... 31
    3.2.11 Status LED ........................................................................................................ 33
  3.3 Interface description .................................................................................................... 34
    3.3.1 Serial interface ...................................................................................................... 34
    3.3.2 CAN interfaces .................................................................................................... 35
  3.4 Software description ................................................................................................... 36
    3.4.1 Software modules for the device ........................................................................ 36
    3.4.2 Programming notes for CODESYS projects .................................................. 39
    3.4.3 Operating states .................................................................................................. 43
    3.4.4 Operating modes ................................................................................................. 47
    3.4.5 Performance limits of the device ...................................................................... 48

4 Configurations ................................................................................................................ 50
  4.1 Set up the runtime system ......................................................................................... 50
    4.1.1 Reinstall the runtime system ............................................................................. 51
    4.1.2 Update the runtime system ............................................................................... 52
    4.1.3 Verify the installation ......................................................................................... 52
  4.2 Set up the programming system ............................................................................... 53
    4.2.1 Set up the programming system manually ........................................................ 53
    4.2.2 Set up the programming system via templates .................................................. 56
  4.3 Function configuration in general .............................................................................. 57
    4.3.1 Configuration of the inputs and outputs (default setting) .................................. 57
    4.3.2 System variables ............................................................................................... 57
  4.4 Function configuration of the inputs and outputs ...................................................... 58
    4.4.1 Configure inputs ............................................................................................... 59
    4.4.2 Configure outputs ............................................................................................. 63
  4.5 Variables ..................................................................................................................... 66
    4.5.1 Retain variables ................................................................................................. 66
    4.5.2 Network variables ............................................................................................. 66
5 ifm function elements 67
5.1 ifm libraries for the device CR0505 ................................................................. 67
  5.1.1 Library ifm\_CR0505\_V06yyzz.LIB ............................................................... 68
  5.1.2 Library ifm\_CR0505\_CANopenMaster\_V04yyzn.LIB ................................. 70
  5.1.3 Library ifm\_CR0505\_CANopenSlave\_V04yyzn.LIB ..................................... 70
  5.1.4 Library ifm\_CAN1\_EXT\_Vxxystt.LIB ........................................................... 71
  5.1.5 Library ifm\_J1939\_X\_Vxxystt.LIB ............................................................... 71
  5.1.6 Library ifm\_hydraulic\_16bitOS05\_Vxxystt.LIB .......................................... 72
5.2 ifm function elements for the device CR0505 .................................................... 73
  5.2.1 Function elements: CAN layer 2 .................................................................. 73
  5.2.2 Function elements: CANopen master .......................................................... 89
  5.2.3 Function elements: CANopen slave ............................................................. 99
  5.2.4 Function elements: CANopen SDOs ............................................................ 107
  5.2.5 Function elements: SAE J1939 ................................................................. 112
  5.2.6 Function elements: serial interface ............................................................. 124
  5.2.7 Function elements: Optimising the PLC cycle ............................................. 129
  5.2.8 Function elements: processing input values ............................................... 135
  5.2.9 Function elements: adapting analogue values ............................................. 140
  5.2.10 Function elements: counter functions for frequency and period measurement 143
  5.2.11 Function elements: PWM functions ......................................................... 154
  5.2.12 Function elements: hydraulic control ...................................................... 168
  5.2.13 Function elements: controllers ................................................................. 184
  5.2.14 Function elements: software reset ............................................................ 193
  5.2.15 Function elements: measuring / setting of time ......................................... 195
  5.2.16 Function elements: saving, reading and converting data in the memory .... 198
  5.2.17 Function elements: data access and data check ......................................... 207

6 Diagnosis and error handling 214
6.1 Diagnosis ........................................................................................................... 214
6.2 Fault .................................................................................................................. 214
6.3 Reaction in case of an error ............................................................................... 215
6.4 Relay: important notes! ................................................................................... 215
6.5 Response to system errors ................................................................................ 216
6.6 CAN / CANopen: errors and error handling .................................................... 216

7 Annex 217
7.1 System flags ...................................................................................................... 217
  7.1.1 System flags: CAN ...................................................................................... 218
  7.1.2 System flags: SAE J1939 ............................................................................. 218
  7.1.3 System flags: error flags (standard side) .................................................... 219
  7.1.4 System flags: LED (standard side) ............................................................ 220
  7.1.5 System flags: voltages (standard side) ...................................................... 221
  7.1.6 System flags: 16..24 inputs and 0..8 outputs (standard side) ...................... 221
7.2 Address assignment and I/O operating modes .................................................. 222
  7.2.1 Address assignment inputs / outputs ......................................................... 222
  7.2.2 Possible operating modes inputs/outputs .................................................. 225
  7.2.3 Addresses / variables of the I/Os ............................................................... 228

7.3 Error tables ....................................................................................................... 230
  7.3.1 Error flags .................................................................................................. 230
  7.3.2 Errors: CAN / CANopen ............................................................................. 230
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Glossary of Terms</td>
<td>231</td>
</tr>
<tr>
<td>9</td>
<td>Index</td>
<td>244</td>
</tr>
<tr>
<td>10</td>
<td>Notizen • Notes • Notes</td>
<td>248</td>
</tr>
<tr>
<td>11</td>
<td>ifm weltweit • ifm worldwide • ifm à l'échelle internationale</td>
<td>252</td>
</tr>
</tbody>
</table>
1 About this manual

Copyright ............................................................................................................................................... 5
Overview: documentation modules for ecomatmobile devices ............................................................. 6
CODESYS programming manual .......................................................................................................... 6
What do the symbols and formats mean? ............................................................................................. 7
How is this documentation structured? ................................................................................................. 8
History of the instructions (CR0020, CR0505) .................................................................................. 9

1.1 Copyright

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

All product names, pictures, companies or other brands used on our pages are the property of the respective rights owners:
• AS-i is the property of the AS-International Association, (→ www.as-interface.net)
• CAN is the property of the CiA (CAN in Automation e.V.), Germany (→ www.can-cia.org)
• CODESYS™ is the property of the 3S – Smart Software Solutions GmbH, Germany (→ www.codesys.com)
• DeviceNet™ is the property of the ODVA™ (Open DeviceNet Vendor Association), USA (→ www.odva.org)
• EtherNet/IP® is the property of the →ODVA™
• IO-Link® (→ www.io-link.com) is the property of the →PROFIBUS Nutzerorganisation e.V., Germany
• Microsoft® is the property of the Microsoft Corporation, USA (→ www.microsoft.com)
• PROFIBUS® is the property of the PROFIBUS Nutzerorganisation e.V., Germany (→ www.profibus.com)
• PROFINET® is the property of the →PROFIBUS Nutzerorganisation e.V., Germany
• Windows® is the property of the →Microsoft Corporation, USA
1.2 Overview: documentation modules for ecomatmobile devices

The documentation for ecomatmobile devices consists of the following modules:

1. **Data sheet**
   - Contents: Technical data in a table
   - Source: → www.ifm.com > select your country > [Data sheet search] > CR0505 > [Technical data in PDF format]

2. **Installation instructions / operating instructions**
   - Contents: Instructions for installation, electrical installation, (commissioning*), technical data
   - Source: The instructions are supplied with the device
   They are also found on ifm’s homepage:
   → www.ifm.com > select your country > [Data sheet search] > CR0505 > [Operating instructions]

3. **Programming manual + online help**
   - Contents: Description of the configuration and the functions of the device software
   - Source: → www.ifm.com > select your country > [Data sheet search] > CR0505 > [Operating instructions]

4. **System manual "Know-how ecomatmobile"**
   - Contents: Know-how about the following topics:
     • Overview Templates and demo programs
     • CAN, CANopen
     • Control outputs
     • User flash memory
     • Visualisations
     • Overview of the files and libraries used
   - Source: → www.ifm.com > select your country > [Data sheet search] > CR0505 > [Operating instructions]

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

1.3 CODESYS programming manual

In the additional "Programming Manual for CODESYS V2.3" you obtain more details about the use of the programming system.

This manual can be downloaded free of charge from ifm’s website:
→ www.ifm.com > Select your country > [Service] > [Download] > [Systems for mobile machines]
You also find manuals and online help for ecomatmobile at:
→ ecomatmobile DVD "Software, tools and documentation"
1.4 What do the symbols and formats mean?

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

**WARNING**

Death or serious irreversible injuries may result.

**CAUTION**

Slight reversible injuries may result.

**NOTICE**

Property damage is to be expected or may result.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![i]</td>
<td>Important notes concerning malfunctions or disturbances</td>
</tr>
<tr>
<td>![ii]</td>
<td>Other remarks</td>
</tr>
<tr>
<td>![i]</td>
<td>Request for action</td>
</tr>
<tr>
<td>![ii]</td>
<td>Reaction, result</td>
</tr>
<tr>
<td>![i]</td>
<td>&quot;see&quot;</td>
</tr>
<tr>
<td>abc</td>
<td>Cross-reference</td>
</tr>
<tr>
<td>123</td>
<td>Decimal number</td>
</tr>
<tr>
<td>0x123</td>
<td>Hexadecimal number</td>
</tr>
<tr>
<td>0b010</td>
<td>Binary number</td>
</tr>
<tr>
<td>[...]</td>
<td>Designation of pushbuttons, buttons or indications</td>
</tr>
</tbody>
</table>
1.5 How is this documentation structured?

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

How to use this manual:

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

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

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

We reserve the right to make alterations which can result in a change of contents of the documentation. You can find the current version on ifm’s website at:
→ www.ifm.com > Select country > [Data sheet search] > (Article no.) > [Operating instructions]
1.6 History of the instructions (CR0020, CR0505)

What has been changed in this manual? An overview:

<table>
<thead>
<tr>
<th>Date</th>
<th>Theme</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-09-09</td>
<td>PID2 (FB)</td>
<td>parameters of the inputs corrected</td>
</tr>
<tr>
<td>2010-11-10</td>
<td>Terminating resistors</td>
<td>correction in topic 1244</td>
</tr>
<tr>
<td>2011-02-14</td>
<td>TIMER_READ_US (FB)</td>
<td>conversion of max. counter value corrected</td>
</tr>
<tr>
<td>2011-04-05</td>
<td>Memory POUs FRAMREAD, FRAMWRITE, FLASHREAD, FLASHWRITE</td>
<td>permitted values of the parameters SRC, LEN, DST</td>
</tr>
<tr>
<td>2011-04-13</td>
<td>CANopen overview</td>
<td>new: CANopen tables in the annex</td>
</tr>
<tr>
<td>2012-01-09</td>
<td>Memory modules FRAMREAD, FRAMWRITE</td>
<td>Swapped parameters SRC, DST in the table <em>Permissible values</em></td>
</tr>
<tr>
<td>2012-07-16</td>
<td>Runtime system</td>
<td>upgrade to v06</td>
</tr>
<tr>
<td>2012-10-04</td>
<td>diverse</td>
<td>corrections</td>
</tr>
<tr>
<td>2013-06-24</td>
<td>various</td>
<td>new document structure</td>
</tr>
<tr>
<td>2014-04-28</td>
<td>Various function blocks</td>
<td>More precise description of the function block input CHANNEL</td>
</tr>
<tr>
<td>2014-06-30</td>
<td>Name of the documentation</td>
<td>&quot;System manual&quot; renamed as &quot;Programming manual&quot;</td>
</tr>
<tr>
<td>2014-07-31</td>
<td>FB PHASE</td>
<td>Description of parameters of outputs C, ET corrected</td>
</tr>
<tr>
<td>2014-08-26</td>
<td>Description of inputs, outputs</td>
<td>highside / lowside replaced by positive / negative switching</td>
</tr>
</tbody>
</table>
| 2015-01-13 | Structure of documentation for error codes, system flags | • error flags: now only in the annex, chapter System flags  
• CAN / CANopen errors and error handling: now only in the system manual "Know-How"  
• error codes, EMCY codes: now in the annex, chapter Error tables |
| 2015-03-10 | Available memory                               | Description improved                                                    |
| 2015-05-22 | FBs input_analog, input_current, input_voltage | permissible input channels                                              |
| 2015-05-26 | FB J1939_x_GLOBAL_REQUEST                      | More precise description                                                |
| 2015-06-10 | Various function blocks                        | Description of the FB input CHANNEL corrected                           |
2 Safety instructions

2.1 Please note!

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

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

⚠️ WARNING

Non-observance of these instructions can lead to property damage or bodily injury!
ifm electronic gmbh does not assume any liability in this regard.

► The acting person must have read and understood the safety instructions and the corresponding
chapters in this manual before working on and with this device.
► The acting person must be authorised to work on the machine/equipment.
► The acting person must have the qualifications and training required to perform this work.
► Adhere to the technical data of the devices!
   You can find the current data sheet on ifm’s homepage at:
   → www.ifm.com > Select your country > [Data sheet search] > (article number.) > [Technical data
   in PDF format]
   Note the installation and wiring information as well as the functions and features of the devices!
   → supplied installation instructions or on ifm’s homepage:
   → www.ifm.com > Select your country > [Data sheet search] > (article number.) > [Operating
   instructions]
   Please note the corrections and notes in the release notes for the existing documentation,
   available on the ifm website:
   → www.ifm.com > Select your country > [Data sheet search] > (article number.) > [Operating
   instructions]

⚠️ NOTICE

The driver module of the serial interface can be damaged!
Disconnecting or connecting the serial interface while live can cause undefined states which damage
the driver module.

► Do not disconnect or connect the serial interface while live.
2.2 What previous knowledge is required?

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

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

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

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

Adhere to the safety instructions.

2.3 Start-up behaviour of the controller

---

**WARNING**

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

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

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

A restart can, for example, be caused by:

• voltage restoration after power failure
• reset after watchdog response because of too long a cycle time
• error elimination after an E-stop

To ensure a safe behaviour of the controller:

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

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

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

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

► If necessary, ensure that welded relay contacts in the application project cannot trigger or continue hazardous movements.
3 System description

3.1 Information about the device

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

- ClassicController: CR0505 as from production status AI
  Runtime system v06b

3.2 Hardware description
3.2.1 Hardware structure

Conditions

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

Relays

The Controller has 2 internal output relays which can each separate 12 outputs from the terminal voltage VBBx \( x = O / R \).

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

The clamp relay is activated via one channel via the system flag RELAIS_CLAMP_15 (→ graphics). RELAIS_CLAMP_15 is active after the start of the controller.

The clamp relay switches the VBBO voltage on the second output group.

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

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

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

![Block Diagram](image)

(Blue) = system flags

Figure: principle block diagram of supply and relays
## Available memory

### FLASH-Speicher

<table>
<thead>
<tr>
<th>FLASH memory (non-volatile, slow memory)</th>
<th>2 MByte</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall existing in the device</td>
<td></td>
</tr>
<tr>
<td>Thereof the following memory areas are reserved for ...</td>
<td></td>
</tr>
<tr>
<td>maximum size of the application program</td>
<td>704 kByte</td>
</tr>
<tr>
<td>data other than the application program</td>
<td>1 MByte</td>
</tr>
<tr>
<td>user can write data such as files, bitmaps, fonts</td>
<td></td>
</tr>
<tr>
<td>data other than the application program</td>
<td>64 kByte</td>
</tr>
<tr>
<td>read data with <code>FLASHREAD</code> (<a href="#">page 202</a>) or write data with <code>FLASHWRITE</code> (<a href="#">page 203</a>)</td>
<td></td>
</tr>
<tr>
<td>(files: 128 bytes less for header)</td>
<td></td>
</tr>
<tr>
<td>The remaining rest of the memory is reserved for system internal purposes.</td>
<td></td>
</tr>
</tbody>
</table>

### SRAM

<table>
<thead>
<tr>
<th>SRAM (volatile, fast memory)</th>
<th>512 kByte</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall existing in the device</td>
<td></td>
</tr>
<tr>
<td>SRAM indicates here all kinds of volatile and fast memories.</td>
<td></td>
</tr>
<tr>
<td>Thereof the following memory areas are reserved for ...</td>
<td></td>
</tr>
<tr>
<td>data reserved by the application program</td>
<td>160 kByte</td>
</tr>
<tr>
<td>The remaining rest of the memory is reserved for system internal purposes.</td>
<td></td>
</tr>
</tbody>
</table>

### FRAM

<table>
<thead>
<tr>
<th>FRAM (non-volatile, fast memory)</th>
<th>32 kByte</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall existing in the device</td>
<td></td>
</tr>
<tr>
<td>FRAM indicates here all kinds of non-volatile and fast memories.</td>
<td></td>
</tr>
<tr>
<td>Thereof the following memory areas are reserved for ...</td>
<td></td>
</tr>
<tr>
<td>variables in the application program, declared as VAR_RETAIN</td>
<td>1 kByte</td>
</tr>
<tr>
<td>as remanent defined flags (from %MB0...)</td>
<td></td>
</tr>
<tr>
<td>→ Set the end of the memory area by FB <code>MEMORY_RETAIN_PARAM</code> (<a href="#">page 200</a>)</td>
<td></td>
</tr>
<tr>
<td>remanent memory freely available to the user</td>
<td></td>
</tr>
<tr>
<td>Access is made via <code>FRAMREAD</code> (<a href="#">page 204</a>) and <code>FRAMWRITE</code> (<a href="#">page 205</a>)</td>
<td></td>
</tr>
<tr>
<td>The remaining rest of the memory is reserved for system internal purposes.</td>
<td></td>
</tr>
</tbody>
</table>


3.2.2 Operating principle of the delayed switch-off

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

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

The ClassicControllers can be switched off via the program by means of a corresponding connection of the supply voltage inputs and the evaluation of the related system flags. The block diagram in the chapter Hardware structure (→ page 13) shows the context of the individual current paths.

Connect terminal VBBS (23) to the ignition switch

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

In automotive engineering the potential is called “clamp 15”.

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

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

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

Latching

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

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

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

**NOTICE**

Risk of destruction of the relay contacts!
In an emergency situation, “sticking” relay contacts can no longer separate the outputs from the power supply!
Falls VBBS (clamp 15) and VBBO are separated from the power supply at the same time, but the potentials VBBx stay connected to it, then the relays can drop even before the outputs are deactivated by the system.
In this case the relays separate the outputs from the power supply **under load**. This significantly reduces the life cycle of the relays.

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

Operating principle of the monitoring concept

The controller monitors the supply voltages and the system error flags. Depending on the status...
• the controller switches off the internal relays
  > the outputs are de-energised, but retain their logic state
or:
• the runtime system deactivates the controller
  > the program stops
  > the outputs change to logic "0"

Operating principle of the monitoring concept

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

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

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

WARNING

Danger due to unintentional and dangerous start of machine or plant sections!
► When creating the program, the programmer must ensure that no unintentional and dangerous start of machines or plant sections after a fault (e.g. e-stop) and the following fault elimination can occur!
  ⇒ Realise restart inhibit!
► In case of an error, set the outputs concerned to FALSE in the program!

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

Monitoring of the supply voltage VBBR

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

<table>
<thead>
<tr>
<th>ERROR_VBBR = TRUE</th>
<th>supply voltage is missed or too low</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_VBBR = FALSE</td>
<td>supply voltage is in order</td>
</tr>
</tbody>
</table>

► This information is to be processed in the application program!
Monitoring and securing mechanisms

After application of the supply voltage

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

- RAM test (one-time)
- supply voltage
- system data consistency
- CRC of the boot loader
- if exists and is started: CRC of the runtime system
- if exists and is started: CRC of the application program
- memory error:
  - If the test is running: flag ERROR_MEMORY = TRUE (can be evaluated as from the first cycle).
  - If the test is not running: red LED is lit.

If runtime system / application is running

Then the following tests are cyclically carried out:

- Triggering of the watchdog (100 ms)
  Then continuous program check watchdog
- Continuous temperature check
  In case of a fault: system flag ERROR_TEMPERATURE = TRUE
- Continuous voltage monitoring
  In case of a fault: system flag ERROR_POWER = TRUE or ERROR_VBBR = TRUE
- Continuous CAN bus monitoring
- Continuous system data monitoring:
  - program loaded
  - operating mode RUN / STOP,
  - runtime system loaded,
  - node ID,
  - baud rate of CAN and RS232.
- In the operating mode RUN:
  Cyclical I/O diagnosis:
  - short circuit,
  - wire break,
  - overload (current) of the inputs and outputs,
  - cross fault (only for SafetyController).
If the TEST pin is not active

> Write protection for system data in FRAM ¹), e.g.:
  - runtime system loaded,
  - calibration data.
  Implemented via hardware and software.
> Write protection for application program (in the flash memory)
> DEBUG mode
¹) FRAM indicates here all kinds of non-volatile and fast memories.

One-time mechanisms

> CRC monitoring during download or upload.
> It must be checked that the runtime system and the application are assigned to the same device.
3.2.5 Inputs (technology)

Contents

Analogue inputs ............................................................................................................................ 21
Digital inputs .................................................................................................................................... 22
Input group I0 (I00...07 / ANALOG0...7) ...................................................................................... 23
Input group I1 (I10...17 / FRQ0...3) ............................................................................................. 24
Input group I2 (I20...27) .................................................................................................................. 25

Analogue inputs

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

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

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

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

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

Figure: principle block diagram multifunction input
Digital inputs

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

Depending on the device the binary inputs can configured differently. In addition to the protective mechanisms against interference, the binary inputs are internally evaluated via an analogue stage. This enables diagnosis of the input signals. But in the application software the switching signal is directly available as bit information.

For some of these inputs (→ data sheet) the potential can be selected to which it will be switched.
Input group I0 (I00...07 / ANALOG0...7)

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

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

→ chapter Possible operating modes inputs/outputs (→ page 225)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

All inputs show the same behaviour concerning function and diagnosis.

Detailed description → chapter Address assignment inputs / outputs (→ page 222)

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

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

Configuration of each input is made via the application program:

- FB INPUT_ANALOG (→ page 137) > input MODE
- Configuration byte Ixx_MODE

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

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

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ERROR_BREAK_Ix             | DWORD  | input group x: wire break error or (resistance input): short to supply
| (0...x, value depends on the device, → data sheet) | Bit = TRUE: error Bit = FALSE: no error |
| ERROR_SHORT_Ix             | DWORD  | input group x: short circuit error
| (0...x, value depends on the device, → data sheet) | Bit = TRUE: error Bit = FALSE: no error |

NAMUR diagnosis for binary signals of non-electronic switches:

- Equip the switch with an additional resistor connection!

Figure: non-electronic switch S at input Inn
Input group I1 (I10...17 / FRQ0...3)

19487

Inputs I10...13

19490

These inputs are a group of multifunction channels. These inputs can be used as follows (each input separately configurable):
• Binary input plus switching (BL) for positive sensor signal
• Output (→ chapter Outputs (technology) (→ page 26))

→ chapter Possible operating modes inputs/outputs (→ page 225)

These inputs cannot be configured.

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

19497

These inputs are a group of multifunction channels. These inputs can be used as follows (each input separately configurable):
• Binary input plus switching (BL) for positive sensor signal
• Fast input for e.g. incremental encoders and frequency or interval measurement

→ chapter Possible operating modes inputs/outputs (→ page 225)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

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

<table>
<thead>
<tr>
<th>FAST_COUNT (→ page 144)</th>
<th>Counter block for fast input pulses</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQUENCY (→ page 145)</td>
<td>Measures the frequency of the signal arriving at the selected channel</td>
</tr>
<tr>
<td>INC_ENCODER (→ page 146)</td>
<td>Up/Down counter function for the evaluation of encoders</td>
</tr>
<tr>
<td>PERIOD (→ page 149)</td>
<td>Measures the frequency and the cycle period (cycle time) in [μs] at the indicated channel</td>
</tr>
<tr>
<td>PERIOD_RATIO (→ page 151)</td>
<td>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 [%].</td>
</tr>
<tr>
<td>PHASE (→ page 153)</td>
<td>Reads a pair of channels with fast inputs and compares the phase position of the signals</td>
</tr>
</tbody>
</table>
Input group I2 (I20...27)

Inputs I20...23

These inputs are a group of multifunction channels. These inputs can be used as follows (each input separately configurable):
- Binary input plus switching (BL) for positive sensor signal
- Output (→ chapter Outputs (technology) (→ page 26))
→ chapter Possible operating modes inputs/outputs (→ page 225)

These inputs cannot be configured.

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

These inputs are a group of multifunction channels. These inputs can be used as follows (each input separately configurable):
- Binary input plus switching (BL) for positive sensor signal
- Fast input for e.g. incremental encoders and frequency or interval measurement
→ chapter Possible operating modes inputs/outputs (→ page 225)

Sensors with diagnostic capabilities to NAMUR can be evaluated.

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAST_COUNT (→ page 144)</td>
<td>Counter block for fast input pulses</td>
</tr>
<tr>
<td>FREQUENCY (→ page 145)</td>
<td>Measures the frequency of the signal arriving at the selected channel</td>
</tr>
<tr>
<td>INC_ENCODER (→ page 146)</td>
<td>Up/Down counter function for the evaluation of encoders</td>
</tr>
<tr>
<td>PERIOD (→ page 149)</td>
<td>Measures the frequency and the cycle period (cycle time) in [μs] at the indicated channel</td>
</tr>
<tr>
<td>PERIOD_RATIO (→ page 151)</td>
<td>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 [%].</td>
</tr>
<tr>
<td>PHASE (→ page 153)</td>
<td>Reads a pair of channels with fast inputs and compares the phase position of the signals</td>
</tr>
</tbody>
</table>
3.2.6 Outputs (technology)

Binary outputs

The following operating modes are possible for the device outputs (→ data sheet):
- binary output, plus switching (BH) with/without diagnostic function
- binary output minus switched (BL) without diagnostic function

PWM outputs

The following operating modes are possible for the device outputs (→ data sheet):
- PWM output, plus switching (BH) without diagnostic function
Output group Q1Q2 (Q10...13 / Q20...23)

These outputs are a group of multifunction channels. These outputs provide several function options (each output separately configurable):
- binary output, plus switching (BH) with diagnostic function and protection
- analogue current-controlled output (PWMi)
- analogue output with Pulse Width Modulation (PWM)
- binary input (→ chapter Inputs (technology) (→ page 21))
  → chapter Possible operating modes inputs/outputs (→ page 225)

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

Configuration of each output is made via the application program:
- indicate the load currents → FB OUTPUT_CURRENT (→ page 157)
- PWM output: → FB PWM1000 (→ page 166)
- Configuration byte Qxx_MODE

**WARNING**

Dangerous restart possible!
Risk of personal injury! Risk of material damage to the machine/plant!
If in case of a fault an output is switched off via the hardware, the logic state generated by the application program is not changed.

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

The outputs in the PWM mode support no diagnostic functions.

When used as binary output, configuration is carried out using the system variables Q1x_MODE...Q2x_MODE. If the diagnostics is to be used, it must be activated in addition.

Wire break and short circuit of the output signal are indicated separately via the system variables ERROR_BREAK_Q1Q2 or ERROR_SHORT_Q1Q2. The individual output error bits can be masked in the application program, if necessary.

**Example:**
The assignment sets the selected output to the operating mode OUT_DIGITAL_H with diagnostics. Overload protection is activated (preset).

**NOTE**

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

For the limit values please make sure to adhere to the data sheet!
The function OUT_OVERLOAD_PROTECTION is not supported in the pure PWM mode.
Depending on the ambient temperature a short circuit cannot be reliably detected from a certain short circuit current since the output drivers temporarily deactivate themselves for protection against destruction.

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

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

![Figure: principle block diagram](image)

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

**Diagnosis: overload (via current measurement)**

Overload can only be detected on an output with current measurement. Overload is defined as ...
"a nominal maximum current of  12.5 %".

**Diagnosis: wire break (via current measurement)**

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

**Diagnosis: short circuit (via voltage measurement)**

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

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

The individual abbreviations have the following meaning:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Analogue input</td>
</tr>
</tbody>
</table>
| BH           | Binary high side input: minus switching for negative sensor signal  
Binary high side output: plus switching for positive output signal |
| BL           | Binary low side input: plus switching for positive sensor signal  
Binary low side output: minus switching for negative output signal |
| CYL          | Input period measurement |
| ENC          | Input encoder signals |
| FRQ          | Frequency input |
| H bridge     | Output with H-bridge function |
| PWM          | Pulse-width modulated signal |
| PWMi         | PWM output with current measurement |
| IH           | Pulse/counter input, high side: minus switching for negative sensor signal |
| IL           | Pulse/counter input, low side: plus switching for positive sensor signal |
| R            | Read back channel for one output |

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

### 3.2.8 Safety instructions about Reed relays

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

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

- The series resistor must not exceed 5 % of the input resistance RE of the device input (→ data sheet). Otherwise, the signal will not be detected as TRUE.
  
  Example:
  
  RE = 3 000 Ohm
  
  ⇒ max. series resistor = 150 Ohm
3.2.9 Feedback on bidirectional inputs/outputs

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

**NOTICE**

Destruction of outputs if there is inadmissible feedback!

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

The output group is potential free if e.g. ...

- RELAIS = FALSE or
- RELAIS_CLAMP_15 = FALSE.

This potential-free status has the consequence that the voltage is fed back via the protective diode of the output transistor if within an input/output group:

- an input (e.g. I1) = TRUE and
- an output of the same group (e.g. Q2) = TRUE.

**Consequence:**

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

**Remedy:**

Operate an input/output group only as inputs OR only as outputs.

or:

Follow the note below.

Example:

The flag RELAIS switches off the VBO potential of the output group.

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

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

Graphic: examples of inadmissible connection: danger of feedback!

**NOTE**

Help for mixed operated bidirectional inputs/outputs:

- Set the flag RELAIS and/or RELAIS_CLAMP_15 in the application program permanently to TRUE:
  - TRUE ----- RELAIS
  - TRUE ----- RELAIS_CLAMP_15
3.2.10 Feedback in case of externally supplied outputs

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

NOTICE

Destruction of outputs if there is inadmissible feedback!
If actuators are externally controlled, the corresponding potential bar of the same output group must not become potential-free (e.g. for RELAIS = FALSE).
Otherwise the terminal voltage VBBx is fed back to the potential bar of the output group via the protective diode integrated in the output driver of the external connected output. A possibly other set output of this group thus triggers its connected load. The load current destroys the output which feeds back.

► Protect externally supplied outputs by means of blocking diodes!

Example:
The flag RELAIS switches off the supply VBBO of the output group. Without blocking diodes the external switch S1 feeds the supply VBBO via the internal protective diode (red) from output Q1 to the internal potential bar of the outputs. If output Q2 = TRUE (→ graphic), K2 will receive voltage via the protective diode Q1 despite RELAIS = FALSE (red lines). Due to overload this protective diode burns out and the output Q1 is destroyed!

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

Remedy: Insert the blocking diodes V1 and V2 (→ green arrows)!
Successful: If RELAIS = FALSE, K2 remains switched off, even if Q2 = TRUE.
NOTE
Help for externally supplied outputs

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

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

<table>
<thead>
<tr>
<th>LED colour</th>
<th>Flashing frequency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>permanently out</td>
<td>no operating voltage</td>
</tr>
<tr>
<td>yellow</td>
<td>briefly on</td>
<td>initialisation or reset checks</td>
</tr>
<tr>
<td>green / black</td>
<td>5 Hz</td>
<td>no runtime system loaded</td>
</tr>
<tr>
<td>green / black</td>
<td>2 Hz</td>
<td>application RUN</td>
</tr>
<tr>
<td>green</td>
<td>permanently on</td>
<td>application STOP</td>
</tr>
<tr>
<td>red / black</td>
<td>2 Hz</td>
<td>application RUN with error</td>
</tr>
<tr>
<td>red</td>
<td>briefly on</td>
<td>fatal error</td>
</tr>
<tr>
<td>red</td>
<td>permanently on</td>
<td>fatal error (if input TEST = active)</td>
</tr>
</tbody>
</table>

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

### Control the LED in the application program

For this device the status LED can also be set by the application program. To do so, the following system variables are used (→ System flags (→ page 217)):

<table>
<thead>
<tr>
<th>LED</th>
<th>LED color for &quot;active&quot; (for &quot;on&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED_X</td>
<td>LED color for &quot;Pause&quot; (for &quot;Off&quot; or different colour)</td>
</tr>
</tbody>
</table>

---

Color constant from the data structure "LED_COLOR". Permissible entries:
- LED_GREEN
- LED_BLUE
- LED_RED
- LED_WHITE
- LED_MAGENTA
- LED_CYAN
- LED_YELLOW
- LED_ORANGE
- LED_BLACK (= LED off)

<table>
<thead>
<tr>
<th>LED_MODE</th>
<th>Flashing frequency from the data structure &quot;LED_MODES&quot;. Permissible entries:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED_2HZ</td>
<td></td>
</tr>
<tr>
<td>LED_1HZ</td>
<td></td>
</tr>
<tr>
<td>LED_05HZ (= 0.5 Hz)</td>
<td></td>
</tr>
<tr>
<td>LED_0HZ (= constant)</td>
<td></td>
</tr>
</tbody>
</table>

### NOTE

- Do NOT use the LED color RED in the application program.
- In case of an error the LED color RED is set by the runtime system.
  - BUT: If the colors and/or flashing modes are changed in the application program, the above table with the default setting is no longer valid.
3.3 Interface description

This device features a serial interface.
The serial interface can generally be used in combination with the following functions:
• program download
• debugging
• free use of the application

NOTE
The serial interface is not available to the user by default, because it is used for program download and debugging.
The interface can be freely used if the user sets the system flag bit SERIAL_MODE=TRUE. Debugging of the application program is then only possible via one of the 4 CAN interfaces or via USB.

Connections and data → data sheet
3.3.2 CAN interfaces

The devices are equipped with several CAN interfaces depending on the hardware design. Basically, all interfaces can be used with the following functions independently of each other:
• Layer 2: CAN at level 2 (→ chapter Function elements: CAN layer 2 (→ page 73))
• CANopen master (→ chapter Function elements: CANopen master (→ page 89))
• CANopen slave (→ chapter Function elements: CANopen slave (→ page 99))
• CANopen network variables (via CODESYS)
• SAE J1939 (for drive management, → chapter Function elements: SAE J1939 (→ page 112))
• bus load detection
• error frame counter
• download interface
• 100 % bus load without package loss

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

<table>
<thead>
<tr>
<th>CAN interface</th>
<th>CAN 1</th>
<th>CAN 2</th>
<th>CAN 3</th>
<th>CAN 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default download ID</td>
<td>ID 127</td>
<td>ID 126</td>
<td>ID 125</td>
<td>ID 124</td>
</tr>
<tr>
<td>CAN protocols</td>
<td>CAN Layer 2</td>
<td>CAN Layer 2</td>
<td>Interface do not exist</td>
<td>Interface do not exist</td>
</tr>
<tr>
<td></td>
<td>CANopen</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAE J1939</td>
<td>SAE J1939</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Standard baud rate = 125 kBit/s
3.4 Software description

Software modules for the device

<table>
<thead>
<tr>
<th>Software module</th>
<th>Can user change the module?</th>
<th>By means of what tool?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application program with libraries</td>
<td>Yes</td>
<td>CODESYS, MaintenanceTool</td>
</tr>
<tr>
<td>Runtime system *)</td>
<td>Upgrade yes, Downgrade no</td>
<td>MaintenanceTool</td>
</tr>
<tr>
<td>Bootloader</td>
<td>No</td>
<td>---</td>
</tr>
<tr>
<td>(Hardware)</td>
<td>No</td>
<td>---</td>
</tr>
</tbody>
</table>

*) The runtime system version number must correspond to the target version number in the CODESYS target system setting. 
→ chapter Set up the target (→ page 54)

Below we describe this software module:
Bootloader

On delivery ecomatmobile controllers only contain the boot loader. The boot loader is a start program that allows to reload the runtime system and the application program on the device. The boot loader contains basic routines...
• for communication between hardware modules,
• for reloading the operating system.
The boot loader is the first software module to be saved on the device.

Runtime system

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

Normally it is necessary to download the runtime system only once. Then, the application program can be loaded into the controller (also repeatedly) without affecting the runtime system.
The runtime system is provided with this documentation on a separate data carrier. In addition, the current version can be downloaded from the website of ifm electronic gmbh:
→ www.ifm.com > Select your country > [Service] > [Download]

Application program

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.

WARNING

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

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

<table>
<thead>
<tr>
<th>Library</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifm_CR0505_Vxyyzz.LIB</td>
<td>Device-specific library Must always be contained in the application program!</td>
</tr>
<tr>
<td>ifm_CAN1_EXT_Vxyyzz.LIB</td>
<td>(optional) if CAN interface 1 of the device is to operate on 29 bits</td>
</tr>
<tr>
<td>ifm_CR0505_CANopenMaster_Vxyyzz.LIB</td>
<td>(optional) if CAN interface 1 of the device is to be operated as a CANopen master</td>
</tr>
<tr>
<td>ifm_CR0505_CANopenSlave_Vxyyzz.LIB</td>
<td>(optional) if CAN interface 1 of the device is to be operated as a CANopen slave</td>
</tr>
<tr>
<td>ifm_CR0505_J1939_x_Vxyyzz.LIB, x = 1...2 = number of the CAN interface</td>
<td>(optional) if a CAN interface of the device is to communicate with a motor control</td>
</tr>
</tbody>
</table>

→ chapter ifm libraries for the device CR0505 (→ page 67)
3.4.2 Programming notes for CODESYS projects

FB, FUN, PRG in CODESYS
Note the cycle time!
Creating application program
Save boot project
Using ifm downloader
Using ifm maintenance tool

Here you receive tips how to program the device.
► See the notes in the CODESYS programming manual
   → www.ifm.com > select your country > [Data sheet search] > CR0505 > [Operating instructions]
   → ecomatmobile DVD “Software, tools and documentation”.

FB, FUN, PRG in CODESYS

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

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

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

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

NOTE
Function blocks must NOT be called in functions!
Otherwise: During execution the application program will crash.
All function elements must NOT be called recursively, nor indirectly!
An IEC application must contain max. 8,000 function elements!

Background:
All variables of functions...
• are initialised when called and
• become invalid after return to the caller.
Function blocks have 2 calls:
• an initialisation call and
• the actual call to do something.
Consequently that means for the FB call in a function:
• every time there is an additional initialisation call and
• the data of the last call gets lost.
Note the cycle time!

For the programmable devices from the controller family ecomatmobile numerous functions are available which enable use of the devices in a wide range of applications.
As these units use more or fewer system resources depending on their complexity it is not always possible to use all units at the same time and several times.

**NOTICE**

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

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

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

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

In CODESYS: [Online] > [Login] > load the new program.

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

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

**NOTE**

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

- CODESYS menu [Online] > [Create boot project]
  - This is necessary after each change!
- After a reboot, the device starts with the boot project last saved.
- If NO boot project was saved:
  - The device remains in the STOP operation after reboot.
  - The application program is not (no longer) available.
  - The LED lights green.

Using ifm downloader

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

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

Using ifm maintenance tool

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

Here you will find the current ifm Maintenance Tool:

→ www.ifm.com > Select your country > [Service] > [Download] > [Systems for mobile machines]
→ ecomatmobile DVD "Software, tools and documentation" under the tab 'R360 tools [D/E]"
3.4.3 Operating states

Operating states: runtime system is not available ................................................................. 43
Operating states: application program is not available ....................................................... 44
Operating states: application program is available ............................................................ 45
Bootloader state ................................................................................................................. 46
INIT state (Reset) ............................................................................................................. 46
STOP state ....................................................................................................................... 46
RUN state .......................................................................................................................... 46
SYSTEM STOP state ....................................................................................................... 46

After power on the ecomatmobile device can be in one of five possible operating states:
• BOOTLOADER
• INIT
• STOP
• RUN
• SYSTEM STOP (after ERROR STOP)

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

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

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

No runtime system was loaded. The ecomatmobile controller is in the boot loading state. Before loading the application software the runtime system must be downloaded.
> The LED flashes green (5 Hz).

INIT state (Reset)

Premise: a valid runtime system is installed.
This state is passed through after every power on reset:
> The runtime system is initialised.
> Various checks are carried out, e.g. waiting for correctly power supply voltage.
> This temporary state is replaced by the RUN or STOP state.
> The LED lights yellow.
Change out of this state possible into one of the following states:
• RUN
• STOP

STOP state

This state is reached in the following cases:
• From the RESET state if:
  • no program is loaded or
  • the last state before the RESET state was the STOP state
• From the RUN state by the STOP command
  • only for the operating mode = Test (→ chapter TEST mode (→ page 47))
> The LED lights green.

RUN state

This state is reached in the following cases:
• From the RESET state if:
  • the last state before the RESET state was the RUN state
• From the STOP state by the RUN command
  • only for the operating mode = Test (→ chapter TEST mode (→ page 47))
> The LED flashes green (2 Hz).

SYSTEM STOP state

The ecomatmobile controller goes to this state if a non tolerable error (ERROR STOP) was found. This state can only be left by a power-off-on reset.
> The LED lights red.
3.4.4 Operating modes

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

**TEST mode**

**NOTICE**

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

**NOTE**

► Connect the TEST connection to the supply voltage only AFTER you have connected the OPC client!
> Otherwise a fatal error will occur.

This operating mode is reached by applying supply voltage to the test input (→ installation instructions > chapter "Technical data" > chapter "Wiring").
The ecomatmobile controller can now receive commands via one of the interfaces in the RUN or STOP mode and, for example, communicate with the programming system.
Only in the TEST mode the software can be downloaded to the controller.
The state of the application program can be queried via the flag TEST.

**SERIAL_MODE**

The serial interface is available for the exchange of data in the application. Debugging the application software is then only possible via the CAN interface.
This function is switched off as standard (FALSE). Via the flag SERIAL_MODE the state can be controlled and queried via the application program or the programming system.
→ chapter Function elements: serial interface (→ page 124)

**DEBUG mode**

If the input DEBUG of SET_DEBUG (→ page 211) is set to TRUE, the programming system or the downloader, for example, can communicate with the controller and execute system commands (e.g. for service functions via the GSM modem CANremote).
In this operating mode a software download is not possible because the test input (→ chapter TEST mode (→ page 47)) is not connected to supply voltage.
## 3.4.5 Performance limits of the device

![Warning]

Note the limits of the device! → Data sheet

### Above-average stress

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

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

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

The following indications should be seen as reference values:

### Restrictions for the use of FBs

<table>
<thead>
<tr>
<th>Current controller</th>
<th>max. 8</th>
<th>If possible, do not use any other performance-affecting functions!</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYCLE, PERIOD, PERIOD_RATIO, PHASE</td>
<td>1 channel</td>
<td>Input frequency ( \leq 10 \text{ kHz} )</td>
</tr>
<tr>
<td></td>
<td>4 channels</td>
<td>Input frequency ( \leq 2 \text{ kHz} )</td>
</tr>
<tr>
<td>INC_ENCODER</td>
<td>max. 4</td>
<td>If possible, do not use any other performance-affecting functions!</td>
</tr>
</tbody>
</table>

---

**NOTICE**

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

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

In this device, a watchdog monitors the program runtime of the CODESYS application.
If the maximum watchdog time (approx. 100 ms) is exceeded:
> the device performs a reset and reboots.
This you can read in the flag LAST_RESET.
4 Configurations

The device configurations described in the corresponding installation instructions or in the *Annex* (→ page 217) to this documentation are used for standard devices (stock items). They fulfil the requested specifications of most applications.

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

4.1 Set up the runtime system

The device configurations described in the corresponding installation instructions or in the *Annex* (→ page 217) to this documentation are used for standard devices (stock items). They fulfil the requested specifications of most applications.

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

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

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

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

→ www.ifm.com > Select your country > [Service] > [Download]

NOTE

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

• runtime system (ifm_CR0505_Vxxxyyzz.H86),
• PLC configuration (ifm_CR0505_Vxx.CFG),
• device library (ifm_CR0505_Vxxyyzz.LIB) and
• the further files.

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

The basic file name (e.g. "CR0505") 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.

The following files must also be loaded:

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

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

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

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

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

**NOTICE**

Risk of data loss!
When deleting or updating the runtime system all data and programs on the device are deleted.
► Save all required data and programs before deleting or updating the runtime system!

When the operating system software or the CODESYS runtime system is considerably improved, ifm releases a new version. The versions are numbered consecutively (V01, V02, V03, ...).
Please see the respective documentation for the new functions of the new software version. Note whether special requirements for the hardware version are specified in the documentation.
If you have a device with an older version and if the conditions for the hardware and your project are OK, you can update your device to the new software version.
For this operation, the same instructions apply as in the previous chapter 'Reinstall the runtime system'.

4.1.3 Verify the installation

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

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

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

**GET_IDENTITY** (→ page 210) Reads the specific identifications stored in the device:
• hardware name and hardware version of the device
• name of the runtime system in the device
• version and revision no. of the runtime system in the device
• name of the application (has previously been saved by means of **SET_IDENTITY** (→ page 212))
4.2 Set up the programming system

Contents

Set up the programming system manually .......................................................... 53
Set up the programming system via templates .................................................. 56

4.2.1 Set up the programming system manually

Contents

Set up the target .................................................................................................. 54
Activate the PLC configuration (e.g. CR0033) ................................................... 55
Set up the target

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

► Select the requested target file in the dialogue window [Target Settings] in the menu [Configuration].
> The target file constitutes the interface to the hardware for the programming system.
> At the same time, several important libraries and the PLC configuration are loaded when selecting the target.
► If necessary, in the window [Target settings] > tab [Network functionality] > activate [Support parameter manager] and / or activate [Support network variables].
► If necessary, remove the loaded (3S) libraries or complement them by further (ifm) libraries.
► Always complement the appropriate device library ifm_CR0505_Vxxyyzz.LIB manually!

NOTE

The software versions suitable for the selected target must always be used:
• runtime system (ifm_CR0505_Vxxyyzz.H86),
• PLC configuration (ifm_CR0505_Vxx.CFG),
• device library (ifm_CR0505_Vxxyyzz.LIB) and
• the further files.

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

The basic file name (e.g. "CR0505") 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.

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

It may happen that the target system cannot or only partly be programmed with your currently installed version of CODESYS. In such a case, please contact the technical support department of ifm electronic gmbh.
Activate the PLC configuration (e.g. CR0033)

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

► The menu item [PLC Configuration] is reached via the tab [Resources].
  Double-click on [PLC Configuration] to open the corresponding window.
► Click on the tab [Resources] in CODESYS:
  ![Resources Tab]
► In the left column double-click on [PLC Configuration].
> Display of the current PLC configuration (example → following figure):

![PLC Configuration Window]

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

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

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

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

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

► CODESYS creates a new project which shows the basic program structure. It is strongly recommended to follow the shown procedure.
4.3 **Function configuration in general**

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

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

4.3.2 **System variables**

All system variables (→ chapter *System flags* (→ page 217)) have defined addresses which cannot be shifted.

> To indicate and process a watchdog error or causes of a new start the system variable LAST_RESET is set.
> Indication of the selected I/O configuration via mode bytes
4.4 Function configuration of the inputs and outputs

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

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

- It must be checked by means of the data sheet if the device used has the described input and output groups (→ 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 225).
- You find program blocks in the templates for each controller that are called during the 1st cycle after a restart of the controller. The networks programmed there are only used to assign a defined configuration to the input and outputs.
4.4.1 Configure inputs

Safety instructions about Reed relays ................................................................. 59
Binary inputs ..................................................................................................... 60
Fast inputs ......................................................................................................... 61
Analogue inputs ................................................................................................ 62

Valid operating modes → chapter Possible operating modes inputs/outputs (→ page 225)
► Configuration of each input is made via the application program:
  • FB INPUT_ANALOG (→ page 137) > input MODE
  • Configuration byte lxx_MODE

Safety instructions about Reed relays

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

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

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

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

The binary input can be operated in following modes:

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

Depending on the device the binary inputs can configured differently. In addition to the protective mechanisms against interference, the binary inputs are internally evaluated via an analogue stage. This enables diagnosis of the input signals. But in the application software the switching signal is directly available as bit information.

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

**Figure: basic circuit of binary input plus switching (BL) for positive sensor signal:**

Input = open \(\Rightarrow\) signal = low (GND)
Fast inputs

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

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

Appropriate function blocks are e.g.:

**On FRQx inputs:**

- **FAST_COUNT** (→ page 144) Counter block for fast input pulses
- **FREQUENCY** (→ page 145) Measures the frequency of the signal arriving at the selected channel

**On CYLx inputs:**

- **PERIOD** (→ page 148) Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
- **PERIOD_RATIO** (→ page 151) Measures the frequency and the cycle period (cycle time) in [µs] during the indicated periods at the indicated channel. In addition, the mark-to-space ratio is indicated in [%].
- **PHASE** (→ page 153) Reads a pair of channels with fast inputs and compares the phase position of the signals

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

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

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

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

![Diagram](image_url)

**Figure: principle block diagram multifunction input**

**NOTE**

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

- Note the higher input resistances for binary evaluation.
4.4.2 Configure outputs

Valid operating modes → chapter Possible operating modes inputs/outputs (→ page 225)

Configuration of each output is made via the application program:
- indicate the load currents → FB OUTPUT_CURRENT (→ page 157)
- PWM output: → FB PWM1000 (→ page 166)
- Configuration byte Qxx_MODE

Binary and PWM outputs

The following operating modes are possible for the device outputs (→ data sheet):
- binary output, plus switching (BH) with diagnostic function and protection
- analogue output with Pulse Width Modulation (PWM)
- PWM output pair H-bridge without diagnostic function

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

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

**WARNING**

Property damage or bodily injury possible due to malfunctions!

The following applies for outputs in PWM mode:
- there is no diagnostic function
- no ERROR flags are set
- the overload protection OUT_OVERLOAD_PROTECTION is NOT active

**WARNING**

Dangerous restart possible!
Risk of personal injury! Risk of material damage to the machine/plant!
If in case of a fault an output is switched off via the hardware, the logic state generated by the application program is not changed.

Remedy:
- Reset the output logic in the application program!
- Remove the fault!
- Reset the outputs depending on the situation.
Behaviour in case of short circuit, permanent overload or wire break:
(applies as from the hardware version AH, however not in the safety mode)

> System flag ERROR_SHORT_Qx (in case of short circuit or overload) or ERROR_BREAK_Qx (in case of wire break) becomes active.

> Only in case of short circuit/overload: the runtime system deactivates the affected output driver.
> The logic of the affected output remains TRUE.
> After a waiting time the output is activated again, which can lead to periodic switching to short circuit.
> The waiting time increases with the (over)load of the output.
> Switch-on time in case of short circuit typically 50 µs, considerably longer in case of overload.

▶ Evaluate the error flag in the application program!
> Reset the output logic, stop the machine if necessary.
> If required, switch off the output group VBBr via RELAY=FALSE (e.g. via ERROR=TRUE).

After fault elimination:

▶ Reset the error flag ERROR_.Qx.
> The output relay re-enables the output group VBBr.
▶ New setting of the output or restart of the machine.
Output group Q1Q2 (Q10...13 / Q20...23)

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

Using `OUTPUT_CURRENT` (→ page 157) load currents ≥ 100 mA can be indicated.

When used as digital output, configuration is carried out using the system variables Q1x_MODE...Q2x_MODE. If the diagnosis is to be used, it must be activated in addition.

Wire break and short circuit of the output signal are indicated separately via the system variables ERROR_BREAK_Q1Q2 and ERROR_SHORT_Q1Q2. The individual output error bits can be masked in the application program, if necessary.

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

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

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

**OUT_OVERLOAD_PROTECTION** is not supported in the pure PWM mode.

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

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

4.5.1 Retain variables

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

Typical applications for retain variables are for example:
• operating hours which are counted up and retained while the machine is in operation,
• position values of incremental encoders,
• preset values entered in the monitor,
• machine parameters,
i.e. all variables whose values must not get lost when the device is switched off.

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

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

4.5.2 Network variables

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

► Integrate the following library/libraries into the CODESYS project:
  • 3S_CANopenNetVar.lib
5    ifm function elements

Contents

ifm libraries for the device CR0505 ..................................................................................................... 67

ifm libraries for the device CR0505 ..................................................................................................... 73

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

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

5.1    ifm libraries for the device CR0505

Contents

Library ifm_CR0505_V06yyzz.LIB ...................................................................................................... 68
Library ifm_CR0505_CANopenMaster_V04yynn.LIB ........................................................................... 70
Library ifm_CR0505_CANopenSlave_V04yynn.LIB ........................................................................... 70
Library ifm_CAN1_EXT_Vxxyyzz.LIB ................................................................................................. 71
Library ifm_J1939_x_Vxxyyzz.LIB ...................................................................................................... 71
Library ifm_hydraulic_16bitOS05_Vxxyyzz.LIB .................................................................................. 72

Legend for ..._Vxxyyzz.LIB:

<table>
<thead>
<tr>
<th>V</th>
<th>xx: 00...99</th>
<th>target version number</th>
</tr>
</thead>
<tbody>
<tr>
<td>yy:</td>
<td>00...99</td>
<td>release number</td>
</tr>
<tr>
<td>zz:</td>
<td>00...99</td>
<td>patch number</td>
</tr>
</tbody>
</table>

Here you will find a list of the ifm function elements matching this device, sorted according to the CODESYS libraries.
### 5.1.1 Library ifm_CR0505_V06yyzz.LIB

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

<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANALOG_RAW</strong> (→ page 136)</td>
<td>Supplies non-standardised values of the analogue/digital converter for each individual input port</td>
</tr>
</tbody>
</table>
| CANx (→ page 81)         | Initialises CAN interface x  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CAN1_BAUDRATE** (→ page 74) | Sets the transmission rate for the bus participant on CAN interface 1 |
| **CAN1_DOWNLOADID** (→ page 75) | Sets the download identifier for CAN interface 1 |
| CANx_ERRORHANDLER (→ page 82) | Executes a “manual” bus recovery on CAN interface x  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CANx_EXT_RECEIVE_ALL** (→ page 83) | CAN interface x: Configures all data receive objects and reads out the receive buffer of the data objects  
  x = 2 = number of the CAN interface |
| **CANx_RECEIVE** (→ page 84) | CAN interface x: Configures a data receive object and reads out the receive buffer of the data object  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CANx_RECEIVE_RANGE** (→ page 86) | CAN interface x: Configures a sequence of data receive objects and reads out the receive buffer of the data objects  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CANx_SDO_READ** (→ page 108) | CAN interface x: Reads the SDO with the indicated indices from the node  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CANx_SDO_WRITE** (→ page 110) | CAN interface x: writes the SDO with the indicated indices to the node  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CANx_TRANSMIT** (→ page 88) | Transfers a CAN data object (message) to the CAN interface x for transmission at each call  
  x = 1...n = number of the CAN interface (depending on the device, → Data sheet) |
| **CHECK_DATA** (→ page 208) | Generates a checksum (CRC) for a configurable memory area and checks the data of the memory area for undesired changes |
| **DELAY** (→ page 185) | Delays the output of the input value by the time T (dead-time element) |
| **FAST_COUNT** (→ page 144) | Counter block for fast input pulses |
| **FLASHREAD** (→ page 202) | Transfers different data types directly from the flash memory to the RAM |
| **FLASHWRITE** (→ page 203) | Writes different data types directly into the flash memory |
| **FRAMREAD** (→ page 204) | Transfers different data types directly from the FRAM memory to the RAM  
  FRAM indicates here all kinds of non-volatile and fast memories. |
| **FRAMWRITE** (→ page 205) | Writes different data types directly into the FRAM memory  
  FRAM indicates here all kinds of non-volatile and fast memories. |
| **FREQUENCY** (→ page 145) | Measures the frequency of the signal arriving at the selected channel |
| **GET_IDENTITY** (→ page 210) | Reads the specific identifications stored in the device:  
  • hardware name and hardware version of the device  
  • name of the runtime system in the device  
  • version and revision no. of the runtime system in the device  
  • name of the application (has previously been saved by means of **SET_IDENTITY** (→ page 212)) |
<p>| <strong>GLR</strong> (→ page 186) | The synchro controller is a controller with PID characteristics |
| <strong>INC_ENCODER</strong> (→ page 146) | Up/Down counter function for the evaluation of encoders |
| <strong>INPUT_ANALOG</strong> (→ page 137) | Current and voltage measurement on the analogue input channel |
| <strong>INPUT_CURRENT</strong> (→ page 138) | Current measurement on the analogue input channel |
| <strong>INPUT_VOLTAGE</strong> (→ page 139) | Voltage measurement on the analogue input channel |
| <strong>MEMCPY</strong> (→ page 206) | Writes and reads different data types directly in the memory |
| <strong>MEMORY_RETAIN_PARAM</strong> (→ page 200) | Determines the remanent data behaviour for various events |</p>
<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NORM</strong> (∆→ page 141)</td>
<td>Normalises a value [WORD] within defined limits to a value with new limits</td>
</tr>
</tbody>
</table>
| **OCC_TASK** (∆→ page 155)| OCC = Output Current Control  
Current controller for a PWMi output channel  
Each instance of the function is called up in a cycle of 5 ms. |
| **OUTPUT_CURRENT** (∆→ page 157) | Measures the current (average via dither period) on an output channel |  |
| **OUTPUT_CURRENT_CONTROL** (∆→ page 158) | Current controller for a PWMi output channel |  |
| **PERIOD** (∆→ page 149) | Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel |  |
| **PERIOD_RATIO** (∆→ page 154) | Measures the frequency and the cycle period (cycle time) in [µs] during the indicated periods at the indicated channel. In addition, the mark-to-space ratio is indicated in [%]. |
| **PHASE** (∆→ page 153) | Reads a pair of channels with fast inputs and compares the phase position of the signals |  |
| **PID1** (∆→ page 188) | PID controller |  |
| **PID2** (∆→ page 190) | PID controller |  |
| **PT1** (∆→ page 192) | Controlled system with first-order delay |  |
| **PWM** (∆→ page 160)  | Initialises and configures a PWMi-capable output channel  
Definition of the PWM frequency via RELOAD |  |
| **PWM100** (∆→ page 164) | Initialises and configures a PWMi-capable output channel  
Indicate PWM frequency in [Hz]  
Indicate mark-to-space ratio in steps of 1 % |  |
| **PWM1000** (∆→ page 166) | Initialises and configures a PWMi-capable output channel  
The mark-to-space ratio can be indicated in steps of 1 % |  |
| **SERIAL_PENDING** (∆→ page 125) | Determines the number of data bytes stored in the serial receive buffer |  |
| **SERIAL_RX** (∆→ page 126) | Reads a received data byte from the serial receive buffer at each call |  |
| **SERIAL_SETUP** (∆→ page 127) | Initialises the serial RS232 interface |  |
| **SERIAL_TX** (∆→ page 128) | Transmits one data byte via the serial RS232 interface |  |
| **SET_DEBUG** (∆→ page 211) | organises the DEBUG mode or the monitoring mode (depending on the TEST input) |  |
| **SET_IDENTITY** (∆→ page 212) | Sets an application-specific program identification |  |
| **SET_INTERRUPT_I** (∆→ page 130) | Conditional execution of a program part after an interrupt request via a defined input channel |  |
| **SET_INTERRUPT_XMS** (∆→ page 133) | Conditional execution of a program part at an interval of x milliseconds |  |
| **SET_PASSWORD** (∆→ page 213) | Sets a user password for access control to program and memory upload |  |
| **SOFTRESET** (∆→ page 194) | leads to a complete reboot of the device |  |
| **TIMER_READ** (∆→ page 190) | Reads out the current system time in [ms]  
Max. value = 49d 17h 2min 47s 295ms |  |
| **TIMER_READ_US** (∆→ page 197) | Reads out the current system time in [µs]  
Max. value = 1h 11min 34s 967ms 295µs |  |
5.1.2 Library ifm_CR0505_CANopenMaster_V04yynn.LIB

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

\( x = 1 \) = number of the CAN interface

This ifm library contains the following function blocks:

<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANx_MASTER_EMCY_HANDLER (→ page 90)</td>
<td>Handles the device-specific error status of the CANopen master on CAN interface ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
<tr>
<td>CANx_MASTER_SEND_EMERGENCY (→ page 91)</td>
<td>Sends application-specific error status of the CANopen master on CAN interface ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
<tr>
<td>CANx_MASTER_STATUS (→ page 93)</td>
<td>Status indication on CAN interface ( x ) of the device used as CANopen master ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
</tbody>
</table>

5.1.3 Library ifm_CR0505_CANopenSlave_V04yynn.LIB

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

\( x = 1 \) = number of the CAN interface

This ifm library contains the following function blocks:

<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANx_SLAVE_EMCY_HANDLER (→ page 100)</td>
<td>Handles the device-specific error status of the CANopen slave on CAN interface ( x ): • error register (index 0x1001) and • error field (index 0x1003) of the CANopen object directory ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
<tr>
<td>CANx_SLAVE_NODEID (→ page 101)</td>
<td>Enables setting of the node ID of a CANopen slave on CAN interface ( x ) at runtime of the application program ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
<tr>
<td>CANx_SLAVE_SEND_EMERGENCY (→ page 102)</td>
<td>Sends application-specific error status of the CANopen slave on CAN interface ( x ) ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
<tr>
<td>CANx_SLAVE_STATUS (→ page 104)</td>
<td>Shows the status of the device used as CANopen slave on CAN interface ( x ) ( x = 1 \ldots n ) = number of the CAN interface (depending on the device; → Data sheet)</td>
</tr>
</tbody>
</table>
### 5.1.4 Library ifm_CAN1_EXT_Vxxyyzz.LIB

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

This ifm library contains the following function blocks:

<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN1_EXT (→ page 76)</td>
<td>Initialises CAN interface 1 also for the extended mode Set the mode and baud rate</td>
</tr>
<tr>
<td>CAN1_EXT_ERRORHANDLER (→ page 77)</td>
<td>Executes a &quot;manual&quot; bus recovery on CAN interface 1</td>
</tr>
<tr>
<td>CAN1_EXT_RECEIVE (→ page 78)</td>
<td>CAN interface 1: Configures a data receive object and reads out the receive buffer of the data object</td>
</tr>
<tr>
<td>CANx_EXT_RECEIVE_ALL (→ page 83)</td>
<td>CAN interface x: Configures all data receive objects and reads out the receive buffer of the data objects x = 1...n = number of the CAN interface</td>
</tr>
<tr>
<td>CAN1_EXT_TRANSMIT (→ page 80)</td>
<td>Transfers a CAN data object (message) to CAN interface 1 for transmission at each call</td>
</tr>
</tbody>
</table>

### 5.1.5 Library ifm_J1939_x_Vxxyyzz.LIB

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

This ifm library contains the following function blocks:

<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1939_x (→ page 113)</td>
<td>CAN interface x: protocol handler for the communication profile SAE J1939 x = 1...n = number of the CAN interface (depending on the device, → Data sheet)</td>
</tr>
<tr>
<td>J1939_x_GLOBAL_REQUEST (→ page 114)</td>
<td>CAN interface x: handles global requesting and receipt of data from the J1939 network participants x = 1...n = number of the CAN interface (depending on the device, → Data sheet)</td>
</tr>
<tr>
<td>J1939_x_RECEIVE (→ page 116)</td>
<td>CAN interface x: Receives a single message or a message block x = 1...n = number of the CAN interface (depending on the device, → Data sheet)</td>
</tr>
<tr>
<td>J1939_x_RESPONSE (→ page 118)</td>
<td>CAN interface x: handles the automatic response to a request message x = 1...n = number of the CAN interface (depending on the device, → Data sheet)</td>
</tr>
<tr>
<td>J1939_x_SPECIFIC_REQUEST (→ page 120)</td>
<td>CAN interface x: automatic requesting of individual messages from a specific J1939 network participant x = 1...n = number of the CAN interface (depending on the device, → Data sheet)</td>
</tr>
<tr>
<td>J1939_x_TRANSMIT (→ page 122)</td>
<td>CAN interface x: sends individual messages or message blocks x = 1...n = number of the CAN interface (depending on the device, → Data sheet)</td>
</tr>
</tbody>
</table>
This library contains function blocks for hydraulic controls.

This ifm library contains the following function blocks:

<table>
<thead>
<tr>
<th>Function element</th>
<th>Short description</th>
</tr>
</thead>
</table>
| CONTROL_OCC      | OCC = Output Current Control  
Scales the input value [WORD] to an indicated current range |
| JOYSTICK_0       | Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0...1000 |
| JOYSTICK_1       | Scales signals [INT] from a joystick D standardised to 0...1000 |
| JOYSTICK_2       | Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation |
| NORM_HYDRAULIC   | Normalises a value [DINT] within defined limits to a value with new limits |
5.2 ifm function elements for the device CR0505

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

5.2.1 Function elements: CAN layer 2

Here, the CAN function blocks (layer 2) for use in the application program are described.
CAN1_BAUDRATE

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxyyzz.LIB
Symbol in CODESYS:

Description

CAN1_BAUDRATE sets the transmission rate for the bus participant.
► To do so, the corresponding value in kbits/s is entered at the input BAUDRATE.

NOTICE

Please note for CR250n, CR0301, CR0302 and CS0015:
The EEPROM memory module may be destroyed by the permanent use of this unit!
► Only carry out the unit once during initialisation in the first program cycle!
► Afterwards block the unit again with ENABLE = FALSE!

The new baud rate will become effective on RESET (voltage OFF/ON or soft reset).
ExtendedController: In the slave module, the new baud rate will become effective after voltage OFF/ON.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE (in the 1st cycle): Adopt and activate parameters else: this function is not executed</td>
</tr>
<tr>
<td>BAUDRATE</td>
<td>WORD := 125</td>
<td>Baud rate [kbits/s] valid = 20, 50, 100, 125, 250, 500, 1000</td>
</tr>
</tbody>
</table>
**CAN1_DOWNLOADID**

= CAN1 download ID  
Unit type = function block (FB)  
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB  
Symbol in CODESYS:

![Diagram](image)

**Description**

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.

![Warning](image)

**NOTICE**

Please note for CR250n, CR0301, CR0302 and CS0015:  
The EEPROM memory module may be destroyed by the permanent use of this unit!  
▶ Only carry out the unit **once** during initialisation in the first program cycle!  
▶ Afterwards block the unit again with ENABLE = FALSE!

**Parameters of the inputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | TRUE (in the 1st cycle): Adopt and activate parameters  
else: this function is not executed |
| ID        | BYTE      | Set download ID of CAN interface x  
x = 1...n = number of the CAN interface (depending on the device,  
→ Data sheet)  
allowed = 1...127  
preset = 127 - (x-1) |
CAN1_EXT

Unit type = function block (FB)
Unit is contained in the library ifm_CAN1_EXT_Vxxyyzz.LIB

Symbol in CODESYS:

```
CAN1_EXT
ENABLE
START
EXTENDED_MODE
BAUDRATE
```

Description

CAN1_EXT initialises the first CAN interface for the extended identifier (29 bits).
The FB has to be retrieved if the first CAN interface e.g. with the function libraries for SAE J1939 is to be used.
A change of the baud rate will become effective after voltage OFF/ON.
The baud rates of CAN 1 and CAN 2 can be set differently.
The input START is only set for one cycle during reboot or restart of the interface.

كن للـFB must be executed before CAN1_EXT.....

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>START</td>
<td>BOOL</td>
<td>TRUE (in the 1st cycle):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start CAN protocol at CAN interface x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>EXTENDED_MODE</td>
<td>BOOL := FALSE</td>
<td>TRUE: identifier of the CAN interface operates with 29 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: identifier of the CAN interface operates with 11 bits</td>
</tr>
<tr>
<td>BAUDRATE</td>
<td>WORD := 125</td>
<td>Baud rate [Kbits/s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permissible = 50, 100, 125, 250, 500, 800, 1000</td>
</tr>
</tbody>
</table>
CAN1_EXT_ERRORHANDLER

Unit type = function block (FB)
Unit is contained in the library ifm_CAN1_EXT_Vxxyyzz.LIB

Symbol in CODESYS:

Description

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

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

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

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSOFF_RECOVER</td>
<td>BOOL</td>
<td>TRUE (only 1 cycle):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; remedy 'bus off' status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; reboot of the CAN interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: function element is not executed</td>
</tr>
</tbody>
</table>

© ifm electronic gmbh
CAN1_EXT_RECEIVE

Unit type = function block (FB)
Unit is contained in the library ifm_CAN1_EXT_Vxxyyzz.LIB

Symbol in CODESYS:

Description

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

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

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

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

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

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page 76).

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| CONFIG    | BOOL      | TRUE (in the 1st cycle): configure data object  
FALSE: during further processing of the program |
| CLEAR     | BOOL      | TRUE: delete receive buffer  
FALSE: function element is not executed |
| ID        | DWORD     | Number of the data object identifier:  
normal frame (21 IDs):  
0...2 047 = 0x0000 0000...0x0000 07FF  
extended Frame (229 IDs):  
0...536 870 911 = 0x0000 0000...0x1FFF FFFF |


## Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>ARRAY [0..7] OF BYTE</td>
<td>received data, (1...8 bytes)</td>
</tr>
<tr>
<td>DLC</td>
<td>BYTE</td>
<td>Number of bytes received in the DATA array with RDO allowed: 0...8</td>
</tr>
<tr>
<td>RTR</td>
<td>BOOL = FALSE</td>
<td>Received message was a Remote Transmission Request (wird hier nicht unterstützt)</td>
</tr>
<tr>
<td>AVAILABLE</td>
<td>BYTE</td>
<td>Number of remaining data bytes allowed = 0...16 0 = no valid data available</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>BOOL</td>
<td>TRUE: Overflow of the data buffer ⇒ loss of datal FALSE: Data buffer is without data loss</td>
</tr>
</tbody>
</table>
CAN1_EXT_TRANSMIT

Unit type = function block (FB)
Unit is contained in the library ifm_CAN1_EXT_Vxxyyz.LIB

Symbol in CODESYS:

Description

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

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

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

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

If this unit is to be used, the 1st CAN interface must first be initialised for the extended ID with CAN1_EXT (→ page 76).

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>DWORD</td>
<td>Number of the data object identifier: normal frame (211 IDs): 0...2 047 = 0000 0000...0000 07FF extended Frame (229 IDs): 0...536 870 911 = 0000 0000...01FFFF FFFF</td>
</tr>
<tr>
<td>DLC</td>
<td>BYTE</td>
<td>Number of bytes to be transmitted from the DATA array with RDO allowed: 0...8</td>
</tr>
<tr>
<td>DATA</td>
<td>ARRAY [0..7] OF BYTE</td>
<td>data to be sent (1...8 bytes)</td>
</tr>
</tbody>
</table>
| ENABLE    | BOOL      | TRUE: execute this function element  
FALSE: unit is not executed  
> Function block inputs are not active  
> Function block outputs are not specified |

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| RESULT    | BOOL      | TRUE (only for 1 cycle): Function block accepted transmit order  
FALSE: Transmit order was not accepted |
CAN2

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyzz.LIB

Symbol in CODESYS:

```
CAN2
  ENABLE
  START
  EXTENDED_MODE
  BAUDRATE
```

Description

CAN2 initialises the 2nd CAN interface.
The FB must be called if the 2nd CAN interface is to be used.
A change of the baud rate will become effective after voltage OFF/ON.
The baud rates of CAN 1 and CAN 2 can be set differently.
The input START is only set for one cycle during reboot or restart of the interface.
For the 2nd CAN interface the libraries for SAE J1939 and Use of the CAN interface to ISO 11992, among others, are available. The FBs to ISO 11992 are only available in the CR2501 on the 2nd CAN interface.

The FB must be executed before CAN2...

Parameters of the inputs

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

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB
Symbol in CODESYS:

Description

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 81)

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSOFF_RECOVER</td>
<td>BOOL</td>
<td>TRUE (only 1 cycle): &gt; remedy 'bus off' status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; reboot of the CAN interfacex</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: function element is not executed</td>
</tr>
<tr>
<td>CAN_RESTART</td>
<td>BOOL</td>
<td>TRUE (only 1 cycle): completely reinitialise CAN interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: function element is not executed</td>
</tr>
</tbody>
</table>
CANx_EXT_RECEIVE_ALL

\[ x = 1...n \] = number of the CAN interface (depending on the device, \( \rightarrow \) Data sheet)

Unit type = function block (FB)
Unit is contained in the library ifm_CANx_EXT_Vxyyzz.LIB

Symbol in CODESYS:

**Description**

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| CONFIG    | BOOL      | TRUE (in the 1st cycle): configure data object
|           |           | FALSE: during further processing of the program |
| CLEAR     | BOOL      | TRUE: delete receive buffer |
|           |           | FALSE: function element is not executed |

**Parameters of the outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>DWORD</td>
<td>Number of the data object identifier</td>
</tr>
<tr>
<td>DATA</td>
<td>ARRAY [0..7] OF BYTE</td>
<td>received data, ( 1...8 ) bytes</td>
</tr>
<tr>
<td>DLC</td>
<td>BYTE</td>
<td>Number of bytes received in the DATA array with SRDO allowed: ( 0...8 )</td>
</tr>
</tbody>
</table>
| AVAILABLE | BYTE      | Number of remaining data bytes allowed = \( 0...16 \)
|           |           | \( 0 \) = no valid data available |
| OVERFLOW  | BOOL      | TRUE: Overflow of the data buffer \( \Rightarrow \) loss of data |
|           |           | FALSE: Data buffer is without data loss |
CANx_RECEIVE

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

Description

CANx_RECEIVE configures a data receive object and reads the receive buffer of the data object. The FB must be called once for each data object during initialisation, in order to inform the CAN controller about the identifiers of the data objects.

In the further program cycle CANx_RECEIVE is called for reading the corresponding receive buffer, also repeatedly in case of long program cycles. The programmer must ensure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and further processed. Each call of the FB decrements the byte AVAILABLE by 1. If the value of AVAILABLE is 0, there is no data in the buffer.

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

If CAN2_RECEIVE is to be used, the second CAN interface must be initialised first using CAN2 (→ page 81).

Parameters of the inputs

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

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

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

Unit type = function block (FB)

Unit is contained in the library `ifm_CR0505_Vxxxyzz.LIB` (xx > 05)

**Symbol in CODESYS:**

![Diagram of CANx_RECEIVE_RANGE]

**Description**

**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. \( \rightarrow \) Example: Initialisation of CANx_RECEIVE_RANGE in 4 cycles \( \rightarrow \) page 87.

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 \( \rightarrow \) page 84) 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG</td>
<td>BOOL</td>
<td>TRUE (in the 1st cycle); configure data object; FALSE: during further processing of the program</td>
</tr>
<tr>
<td>CLEAR</td>
<td>BOOL</td>
<td>TRUE: delete receive buffer; FALSE: function element is not executed</td>
</tr>
<tr>
<td>FIRST_ID</td>
<td>CAN1: WORD CAN2: DWORD</td>
<td>number of the first data object identifier of the sequence permissible values normal frame = 0...2 047 (2(^11)) permissible values extended frame = 0...536 870 911 (2(^{29}))</td>
</tr>
<tr>
<td>LAST_ID</td>
<td>CAN1: WORD CAN2: DWORD</td>
<td>number of the last data object identifier of the sequence permissible values normal frame = 0...2 047 (2(^11)) permissible values extended frame = 0...536 870 911 (2(^{29})) LAST_ID has to be bigger than FIRST_ID</td>
</tr>
</tbody>
</table>
## Parameters of the outputs

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

### Example: Initialisation of CANx_RECEIVE_RANGE in 4 cycles

```plaintext
BEGIN_VAR
init : BOOL = FALSE;
initstep : WORD = 1;
can10 : CAN1;
can2 : CAN2_RECEIVE_RANGE;
cnt : WORD;
END_VAR

IF can03[enable] = TRUE, START = init, EXTENDED_MODE = FALSE, BAUDRATE = 125 THEN

BEGIN_CASE
CASE initstep OF
    1: 
can2[CONF0] = TRUE, CLEAR = FALSE, FIRST_ID = 16#100, LAST_ID = 16#10F, ID = 16#11F, DLC = DATA, AVAILABLE = OVERFLOW = ;
    initstep = initstep + 1;
    2: 
can2[CONF0] = TRUE, CLEAR = FALSE, FIRST_ID = 16#110, LAST_ID = 16#11F, ID = 16#12F, DLC = DATA, AVAILABLE = OVERFLOW = ;
    initstep = initstep + 1;
    3: 
can2[CONF0] = TRUE, CLEAR = FALSE, FIRST_ID = 16#120, LAST_ID = 16#13F, ID = 16#13F, DLC = DATA, AVAILABLE = OVERFLOW = ;
    initstep = initstep + 1;
    4: 
can2[CONF0] = FALSE, CLEAR = FALSE, FIRST_ID = 16#100, LAST_ID = 16#100, ID = 16#100, DLC = DATA, AVAILABLE = OVERFLOW = ;
    initstep = initstep + 1;
ELSE
    can2[CONF0] = FALSE, CLEAR = FALSE, FIRST_ID = 16#100, LAST_ID = 16#100, ID = 16#100, DLC = DATA, AVAILABLE = OVERFLOW = ;
    initstep = initstep + 1;
END_CASE

init = FALSE;
```

("Test")

IF cnt.available > 0 THEN
    cnt := cnt + 1;
END_IF
```
CANx_TRANSMIT

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

Unit type = function block (FB)

Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB

Symbol in CODESYS:

\[
\begin{array}{c}
\text{CANx_TRANSMIT} \\
\text{ID} \\
\text{DLC} \\
\text{DATA} \\
\text{ENABLE} \\
\text{RESULT}
\end{array}
\]

Description

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.

\( \square \) If CAN2_TRANSMIT is to be used, the second CAN interface must be initialised first using CAN2 (\( \rightarrow \) page 81).

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>WORD</td>
<td>number of the data object identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permissible values = 0...2 047</td>
</tr>
<tr>
<td>DLC</td>
<td>BYTE</td>
<td>Number of bytes to be transmitted from the DATA array</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with RDO allowed: 0...8</td>
</tr>
<tr>
<td>DATA</td>
<td>ARRAY [0..7] OF BYTE</td>
<td>data to be sent (1...8 bytes)</td>
</tr>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>BOOL</td>
<td>TRUE (only for 1 cycle):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Function block accepted transmit order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: Transmit order was not accepted</td>
</tr>
</tbody>
</table>
5.2.2 Function elements: CANopen master

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

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

Unit type = function block (FB)

Unit is contained in the library ifm_CR0505_CANopenMaster_Vxxyyzz.LIB

Symbol in CODESYS:

Description

CANx_MASTER_EMCY_HANDLER manages the device-specific error status of the master. The FB must be called in the following cases:

- the error status is to be transmitted to the network and
- the error messages of the application are to be stored in the object directory.

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

If application-specific error messages are to be stored in the object directory, CANx_MASTER_EMCY_HANDLER must be called after (repeatedly) calling CANx_MASTER_SEND_EMERGENCY (\rightarrow page 91).

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_ERROR_FIELD</td>
<td>BOOL</td>
<td>FALSE \rightarrow TRUE (edge): \begin{itemize} \item transmit content of ERROR_FIELD to function block output \item delete content of ERROR_FIELD in object directory \item else: this function is not executed \end{itemize}</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_REGISTER</td>
<td>BYTE</td>
<td>Shows content of OBV index 0x1001 (error register)</td>
</tr>
<tr>
<td>ERROR_FIELD</td>
<td>ARRAY [0..5] OF WORD</td>
<td>Shows the content of the OBV index 0x1003 (error field)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ERROR_FIELD[0]: number of stored errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ERROR_FIELD[1..5]: Stored errors, the most recent error is shown on index [1]</td>
</tr>
</tbody>
</table>
ifm function elements for the device CR0505

CANx_MASTER_SEND_EMERGENCY

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

Unit type = function block (FB)

Unit is contained in the library ifm_CR0505_CANopenMaster_Vxxyyzz.LIB

Symbol in CODESYS:

![CANx_MASTER_SEND_EMERGENCY](image)

Description

CANx_MASTER_SEND_EMERGENCY transmits application-specific error states. The FB is called if the error status is to be transmitted to other devices in the network.

![If application-specific error messages are to be stored in the object directory, CANx_MASTER_EMCY_HANDLER (→ page 90) must be called after (repeatedly) calling CANx_MASTER_SEND_EMERGENCY.](image)

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td>ERROR</td>
<td>BOOL</td>
<td>Using this input, the information whether the error associated to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configured error code is currently present is transmitted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE ⇒ TRUE (edge): sends the next error code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if input was not TRUE in the last second</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE ⇒ FALSE (edge) AND the fault is no longer indicated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>after a delay of approx. 1 s:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; zero error message is sent</td>
</tr>
<tr>
<td>ERROR_CODE</td>
<td>WORD</td>
<td>The error code provides detailed information about the detected error.</td>
</tr>
<tr>
<td>ERROR_REGISTER</td>
<td>BYTE</td>
<td>ERROR_REGISTER indicates the error type. The value indicated here is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linked by a bit-by-bit OR operation with all the other error messages that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>are currently active. The resulting value is written into the error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>register (index 1001w00) and transmitted with the EMCY message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The values should be entered according to the CANopen specification.</td>
</tr>
<tr>
<td>MANUFACTURER_ERROR_FIELD</td>
<td>ARRAY [0..4] OF BYTE</td>
<td>Here, up to 5 bytes of application-specific error information can be entered. The format can be freely selected.</td>
</tr>
</tbody>
</table>
Example: CANx_MASTER_SEND_EMERGENCY

In this example 3 error messages will be generated subsequently:
1. ApplError1, Code = 0xFF00 in the error register 0x81
2. ApplError2, Code = 0xFF01 in the error register 0x81
3. ApplError3, Code = 0xFF02 in the error register 0x81
CAN1_MASTER_EMERGY_HANDLER sends the error messages to the error register "Object 0x1001" in the error array "Object 0x1003".
CANx_MASTER_STATUS

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_CANopenMaster_Vxxyyzz.LIB

Symbol in CODESYS:

```
CANx_MASTER_STATUS
- CANOPEN_LED_STATUS
- GLOBAL_START
- CLEAR_RX_OVERFLOW_FLAG
- CLEAR_RX_BUFFER
- CLEAR_TX_OVERFLOW_FLAG
- CLEAR_TX_BUFFER
- CLEAR_DO_CHANGED_FLAG
- CLEAR_ERROR_CONTROL
- RESET_ALL_NODES
- START_ALL_NODES
- NODE_STATE
- NODE_STATE_SLAVES
- EMERGENCY_OBJECT_SLAVES
```

Description

Status indication of the device used with CANopen.
CANx_MASTER_STATUS shows the status of the device used as CANopen master. Further possibilities:
• monitoring the network status
• monitoring the status of the connected slaves
• resetting or starting the slaves in the network.

The FB simplifies the use of the CODESYS CANopen master libraries. We urgently recommend to carry out the evaluation of the network status and of the error messages via this FB.
### Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANOPEN_LED_STATUS</td>
<td>BOOL</td>
<td>(input not available for PDM devices)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: the status LED of the controller is switched to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode &quot;CANopen&quot;:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>flashing frequency 0.5 Hz = PRE-OPERATIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>flashing frequency 2.0 Hz = OPERATIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The other diagnostic LED signals are not changed by this operating mode.</td>
</tr>
<tr>
<td>GLOBAL_START</td>
<td>BOOL</td>
<td>TRUE: All connected network participants (slaves)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>are started simultaneously during network initialisation (⇒ state OPERATIONAL).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: The connected network participants are started one after the other.</td>
</tr>
<tr>
<td>CLEAR_RX_OVERFLOW_FLAG</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Clear error flag RX_OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_RX_BUFFER</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Delete data in the receive buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_TX_OVERFLOW_FLAG</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Clear error flag TX_OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_TX_BUFFER</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Delete data in the transmit buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_OD_CHANGED_FLAG</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Delete flag OD_CHANGED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_ERROR_CONTROL</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Delete the guard error list (ERROR_CONTROL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>RESET_ALL_NODES</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): All connected network participants (slaves) are reset via NMT command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>START_ALL_NODES</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): All connected network participants (slaves) are started via NMT command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>NODE_STATE_SLAVES</td>
<td>DWORD</td>
<td>Shows states of all network nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example code → chapter Example: CANx_MASTER_STATUS → page 97</td>
</tr>
<tr>
<td>EMERGENCY_OBJECT_SLAVES</td>
<td>DWORD</td>
<td>Shows the last error messages of all network nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→ chapter Access to the structures at runtime of the application → page 96</td>
</tr>
</tbody>
</table>
Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>BYTE</td>
<td>current node ID of the CANopen master</td>
</tr>
<tr>
<td>BAUDRATE</td>
<td>WORD</td>
<td>current baudrate of the CANopen master in [kBaud]</td>
</tr>
<tr>
<td>NODE_STATE</td>
<td>INT</td>
<td>Current status of CANopen master</td>
</tr>
<tr>
<td>SYNC</td>
<td>BOOL</td>
<td>SYNC signal of the CANopen master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: In the last cycle a SYNC signal was sent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: In the last cycle no SYNC signal was sent</td>
</tr>
<tr>
<td>RX_OVERFLOW</td>
<td>BOOL</td>
<td>TRUE: Error: receive buffer overflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: no overflow</td>
</tr>
<tr>
<td>TX_OVERFLOW</td>
<td>BOOL</td>
<td>TRUE: Error: transmission buffer overflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: no overflow</td>
</tr>
<tr>
<td>OD_CHANGED</td>
<td>BOOL</td>
<td>TRUE: Data in the object directory of the CANopen master have been changed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: no data change</td>
</tr>
<tr>
<td>ERROR_CONTROL</td>
<td>ARRAY [0..7]  OF BYTE</td>
<td>The array contains the list (max. 8) of missing network nodes (guard or heartbeat error)</td>
</tr>
<tr>
<td>GET_EMERGENCY</td>
<td>STRUCT</td>
<td>At the output the data for the structure CANx_EMERGENCY_MESSAGE are available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The last received EMCY message in the CANopen network is always displayed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To obtain a list of all occurred errors, the array “EMERGENCY_OBJECT_SLAVES” must be evaluated.</td>
</tr>
<tr>
<td>NODE_ID</td>
<td>BYTE</td>
<td>node ID of the master</td>
</tr>
<tr>
<td>BAUDRATE</td>
<td>WORD</td>
<td>baud rate of the master</td>
</tr>
<tr>
<td>NODE_STATE</td>
<td>INT</td>
<td>current status of the master</td>
</tr>
<tr>
<td>SYNC</td>
<td>BOOL</td>
<td>SYNC signal of the master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is set in the tab [CAN parameters] of the master depending on the set time [Com. Cycle Period].</td>
</tr>
<tr>
<td>RX_OVERFLOW</td>
<td>BOOL</td>
<td>error flag &quot;receive buffer overflow&quot;</td>
</tr>
</tbody>
</table>
Parameters of internal structures

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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_CR0505_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_CR0505_CANopenMaster_Vxxyyzz.LIB. |

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

Slave information

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

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

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

```
001 PROGRAM MasterStatus
002 VAR
003 Status: CR0505_MASTER_STATUS;
004 LastStatus: BOOL = TRUE;
005 GlobalStartNodes: BOOL = TRUE;
006 ClearRxOverflowFlag: BOOL;
007 ClearRxBuffer: BOOL;
008 ClearTxOverflowFlag: BOOL;
009 ClearTxBuffer: BOOL;
010 ClearDcChanged: BOOL;
011 ClearErrorControl: BOOL;
012 ResetAllNodes: BOOL;
013 SlaveAllNodes: BOOL;
014 NodeId: BYTE;
015 Baudrate: WORD;
016 NodeState: INT;
017 Sync: BOOL;
018 RxOverflow: BOOL;
019 TxOverflow: BOOL;
020 OdChanged: BOOL;
021 GuardHeartbeatErrorArray: ARRAY[0..7] OF BYTE;
022 GetEmergency: EMERGENCY_MESSAGE;
023 END_VAR

Structure node status

```

```
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

```

```
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

At runtime you can access the corresponding array element via the global variables of the library and therefore read the status or EMCY messages or reset the node.

If ResetSingleNodeArray[0].RESET_NODE is set to TRUE for a short time in the example given above, the first node is reset in the configuration tree.

Concerning the possible error codes → system manual "Know-How ecomatmobile" → chapter CAN / CANopen: errors and error handling.
5.2.3 Function elements: CANopen slave

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANx_SLAVE_EMCY_HANDLER</td>
<td>100</td>
</tr>
<tr>
<td>CANx_SLAVE_NODEID</td>
<td>101</td>
</tr>
<tr>
<td>CANx_SLAVE_SEND_EMERGENCY</td>
<td>102</td>
</tr>
<tr>
<td>CANx_SLAVE_STATUS</td>
<td>104</td>
</tr>
</tbody>
</table>

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

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_CANopenSlave_Vxxyyzz.LIB

Symbol in CODESYS:

Description

CANx_SLAVE_EMCY_HANDLER handles the device-specific error status of the CANopen slave:
• error register (index 0x1001) and
• error field (index 0x1003) of the CANopen object directory.

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR_ERROR_FIELD</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): transmit content of ERROR_FIELD to function block output delete content of ERROR_FIELD in object directory else: this function is not executed</td>
</tr>
</tbody>
</table>

Parameters of the outputs

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

= CANx Slave Node-ID
x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_CANopenSlave_Vxxyyzz.LIB

Symbol in CODESYS:

Description

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | FALSE → TRUE (edge):
                        Adopt and activate parameters
                        else: this function is not executed |
| NODEID    | BYTE      | node ID = ID of the node
                        permissible values = 0...127 |
CANx_SLAVE_SEND_EMERGENCY

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_CANopenSlave_Vxxyyzz.LIB

Symbol in CODESYS:

|-- CANx_SLAVE_SEND_EMERGENCY
    |   -- ENABLE
    |   -- ERROR
    |   -- ERROR_CODE
    |   -- ERRORREGISTER
    |   -- MANUFACTURER_ERROR_FIELD

Description

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

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>ERROR</td>
<td>BOOL</td>
<td>Using this input, the information whether the error associated to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configured error code is currently present is transmitted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE ⇒ TRUE (edge): sends the next error code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if input was not TRUE in the last second</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE ⇒ FALSE (edge) AND the fault is no longer indicated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; zero error message is sent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>after a delay of approx. 1 s:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; this function is not executed</td>
</tr>
<tr>
<td>ERROR_CODE</td>
<td>WORD</td>
<td>The error code provides detailed information about the detected error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The values should be entered according to the CANopen specification.</td>
</tr>
<tr>
<td>ERROR_REGISTER</td>
<td>BYTE</td>
<td>ERROR_REGISTER indicates the error type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value indicated here is linked by a bit-by-bit OR operation with all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the other error messages that are currently active. The resulting value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is written into the error register (index 100116/00) and transmitted with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the EMCY message. The values should be entered according to the CANopen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specification.</td>
</tr>
<tr>
<td>MANUFACTURER_ERROR_FIELD</td>
<td>ARRAY [0..4] OF BYTE</td>
<td>Here, up to 5 bytes of application-specific error information can be entered. The format can be freely selected.</td>
</tr>
</tbody>
</table>
Example: CANx_SLAVE_SEND_EMERGENCY

In this example 3 error messages will be generated subsequently:

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

CAN1_SLAVE_EMGY_HANDLER sends the error messages to the error register "Object 0x1001" in the error array "Object 0x1003".
CANx_SLAVE_STATUS

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_CANopenSlave_Vxxyyzz.LIB

Symbol in CODESYS:

Description

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

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

Example:

```
PROGRAM SlaveStatus
VAR
  SlaveStatus: CR0505_SLAVE_STATUS;
  LedStatus: BOOLEAN = TRUE;
  ClearRxOverflowFlag: BOOLEAN;
  ClearRxBuffer: BOOLEAN;
  ClearTxOverflowFlag: BOOLEAN;
  ClearTxBuffer: BOOLEAN;
  ClearResetFlags: BOOLEAN;
  GuardHeartbeat_Error: BOOLEAN;
  RxOverflow: BOOLEAN;
  TxOverflow: BOOLEAN;
  ResetNode: BOOLEAN;
  ResetCom: BOOLEAN;
  OdChanged: BOOLEAN;
  OdChangedIndex: INT;
END_VAR
```
## Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANOPEN_LED_STATUS</td>
<td>BOOL</td>
<td>(input not available for PDM devices)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TRUE:</strong> the status LED of the controller is switched to the mode &quot;CANopen&quot;:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flashing frequency 0.5 Hz = PRE-OPERATIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flashing frequency 2.0 Hz = OPERATIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The other diagnostic LED signals are not changed by this operating mode.</td>
</tr>
<tr>
<td>GLOBAL_START</td>
<td>BOOL</td>
<td><strong>TRUE:</strong> All connected network participants (slaves) are started simultaneously during network initialisation (⇒ state OPERATIONAL).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>FALSE:</strong> The connected network participants are started one after the other.</td>
</tr>
<tr>
<td>CLEAR_RX_OVERFLOW_FLAG</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Clear error flag RX_OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_RX_BUFFER</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Delete data in the receive buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_TX_OVERFLOW_FLAG</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Clear error flag TX_OVERFLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_TX_BUFFER</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Delete data in the transmit buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_RESET_FLAGS</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Clear flag RESET_NODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clear flag RESET_COM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
<tr>
<td>CLEAR_OD_CHANGED_FLAGS</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): Clear flag OD_CHANGED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clear flag OD_CHANGED_INDEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else: this function is not executed</td>
</tr>
</tbody>
</table>
## Parameters of the outputs

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

Contents

CANx_SDO_READ ................................................................. 108
CANx_SDO_WRITE .............................................................. 110

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

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Description

CANx_SDO_READ reads the → SDO (→ page 242) with the indicated indexes from the node.
Prerequisite: Node must be in the mode "PRE-OPERATIONAL" or "OPERATIONAL".
By means of these, the entries in the object directory can be read. So it is possible to selectively read
the node parameters.

Example:

Parameters of the inputs

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>BYTE</td>
<td>feedback of the function block (possible messages → following table)</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>Length of the entry in &quot;number of bytes&quot;. The value for LEN must not be greater than the size of the receive array. Otherwise any data is overwritten in the application.</td>
</tr>
</tbody>
</table>

#### Possible results for RESULT:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 FB is inactive</td>
</tr>
<tr>
<td>1</td>
<td>01 FB execution completed without error – data is valid</td>
</tr>
<tr>
<td>2</td>
<td>02 function block is active (action not yet completed)</td>
</tr>
<tr>
<td>3</td>
<td>03 Error, no data received during monitoring time</td>
</tr>
</tbody>
</table>
**CANx_SDO_WRITE**

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

Unit type: function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

**Symbol in CODESYS:**

```
CANx_SDO_WRITE
ENABLE
NODE
IDX
SUBIDX
LEN
DATA
RESULT
```

**Description**

CANx_SDO_WRITE writes the → SDO (→ page 242) with the specified indexes to the node.
Prerequisite: the node must be in the state "PRE-OPERATIONAL" or "OPERATIONAL".
Using this FB, the entries can be written to the object directory. So it is possible to selectively set the node parameters.

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

**Example:**

```
  SDO_write1
  CANx_SDO_WRITE ENABLE RESULT
  node= node
  id= IDX
  subidx= SUBIDX
  len= LEN
  data
```

```
Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | TRUE: execute this function element  
             FALSE: unit is not executed  
             > Function block inputs are not active  
             > Function block outputs are not specified |
| NODE      | BYTE      | ID of the node  
             permissible values = 1...127 = 0x01...0x7F |
| IDX       | WORD      | index in object directory |
| SUBIDX    | BYTE      | sub-index referred to the index in the object directory |
| LEN       | WORD      | Length of the entry in “number of bytes”  
             The value for LEN must not be greater than the size of the transmit array. Otherwise any data is sent. |
| DATA      | DWORD     | Address of the transmit data array  
             permissible length = 0...255  
             Determine the address by means of the operator ADR and assign it to the FB! |

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| RESULT    | BYTE      | feedback of the function block  
             (possible messages → following table) |

Possible results for RESULT:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 FB is inactive</td>
</tr>
<tr>
<td>1</td>
<td>01 FB execution completed without error – data is valid</td>
</tr>
<tr>
<td>2</td>
<td>02 function block is active (action not yet completed)</td>
</tr>
<tr>
<td>3</td>
<td>03 Error, data cannot be transmitted</td>
</tr>
</tbody>
</table>
5.2.5 Function elements: SAE J1939

For SAE J1939, ifm electronic provides a number of function elements which will be explained in the following.
J1939_x

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_J1939_x_Vxxyyzz.LIB

Symbol in CODESYS:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>Start</td>
<td>BOOL</td>
<td>TRUE (only for 1 cycle):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start J1939 protocol at CAN interface x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>My_Address</td>
<td>BYTE</td>
<td>J1939 address of the device</td>
</tr>
</tbody>
</table>

Description

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

**NOTE**

(for RTS to v05 only)
J1939 communication via the 1st CAN interface:
► First initialise the interface via CAN1_EXT
(→ page 76)!

J1939 communication via the 2nd CAN interface:
► First initialise the interface via CAN2
(→ page 81)!

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.
J1939_x_GLOBAL_REQUEST

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_J1939_x_Vxxyyzz.LIB

Symbol in CODESYS:

Description

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

Info

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

NOTICE

Risk of inadmissible overwriting of data!

► Create a receiver array with a size of 1 785 bytes.
   This is the maximum size of a J1939 message.

► Check the amount of received data:
   the value must not exceed the size of the array created to receive data!

► For every requested message use an own instance of the FB!

► To the destination address DST applies:
   ◮ Determine the address by means of the operator ADR and assign it to the FB!
   ◮ In addition, the priority (typically 3, 6 or 7) must be assigned.

► Given that the request of data can be handled via several control cycles, this process must be evaluated via the RESULT byte.
  • RESULT = 2: the POU is waiting for data of the participants.
  • RESULT = 1: data was received by a participant.
    The output LEN indicates how many data bytes have been received.
    Store / evaluate this new data immediately!
    When a new message is received, the data in the memory address DST is overwritten.
  • RESULT = 0: no participant on the bus sends a reply within 1.25 seconds.
    The FB returns to the non-active state.
    Only now may ENABLE be set again to FALSE!

► For the reception of data from several participants at short intervals:
   call the POU several times in the same PLC cycle and evaluate it at once!
Parameters of the inputs

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

Info

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

Parameters of the outputs

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

Possible results for RESULT:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FB is inactive</td>
</tr>
<tr>
<td>1</td>
<td>FB execution completed without error – data is valid</td>
</tr>
<tr>
<td>2</td>
<td>function block is active (action not yet completed)</td>
</tr>
<tr>
<td>3</td>
<td>Error</td>
</tr>
</tbody>
</table>
J1939_x_RECEIVE

x = 1..n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_J1939_x_Vxxyzz.LIB

Symbol in CODESYS:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>RESULT</td>
</tr>
<tr>
<td>CONFIG</td>
<td>DEVICE</td>
</tr>
<tr>
<td>PG</td>
<td>LEN</td>
</tr>
<tr>
<td>PF</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td></td>
</tr>
<tr>
<td>RPT</td>
<td></td>
</tr>
<tr>
<td>LIFE</td>
<td></td>
</tr>
<tr>
<td>DST</td>
<td></td>
</tr>
</tbody>
</table>

Description

J1939_x_RECEIVE serves for receiving one individual message or a block of messages.
To do so, the FB must be initialised for one cycle via the input CONFIG. During initialisation, the parameters PG, PF, PS, RPT, LIFE and the memory address of the data array DST are assigned.

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

NOTICE

Risk of inadmissible overwriting of data!

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

To the destination address DST applies:

Determine the address by means of the operator ADR and assign it to the FB!

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

The receipt of data must be evaluated via the RESULT byte. If RESULT = 1 the data can be read from the memory address assigned via DST and can be further processed.
> When a new message is received, the data in the memory address DST is overwritten.
> The number of received message bytes is indicated via the output LEN.
> If RESULT = 3, no valid messages have been received in the indicated time window (LIFE • RPT).

This block must also be used if the messages are requested using the FBs J1939_..._REQUEST.
# Parameters of the inputs

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

# Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>BYTE</td>
<td>feedback of the function block (possible messages → following table)</td>
</tr>
<tr>
<td>DEVICE</td>
<td>BYTE</td>
<td>J1939 address of the sender</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>Number of received bytes</td>
</tr>
</tbody>
</table>

**Possible results for RESULT:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00</td>
<td>FB is inactive</td>
</tr>
<tr>
<td>1 01</td>
<td>FB execution completed without error – data is valid</td>
</tr>
<tr>
<td>3 03</td>
<td>Error, no data received during monitoring time</td>
</tr>
</tbody>
</table>
J1939_x_RESPONSE

x = 1...n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_J1939_x_Vxxyyzz.LIB

Symbol in CODESYS:

Description

J1939_x_RESPONSE handles the automatic response to a request message.
This FB is responsible for the automatic sending of messages to "Global Requests" and "Specific Requests". To do so, the FB must be initialised for one cycle via the input CONFIG.
The parameters PG, PF, PS, RPT and the address of the data array SRC are assigned to the FB.

► To the source address SRC applies:
  1. Determine the address by means of the operator ADR and assign it to the FB!
► In addition, the number of data bytes to be transmitted is assigned.

Parameters of the inputs

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>BYTE</td>
<td>feedback of the function block (possible messages → following table)</td>
</tr>
</tbody>
</table>

**Possible results for RESULT:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00</td>
<td>FB is inactive</td>
</tr>
<tr>
<td>1 01</td>
<td>Data transfer completed without errors</td>
</tr>
<tr>
<td>2 02</td>
<td>function block is active (action not yet completed)</td>
</tr>
<tr>
<td>3 03</td>
<td>Error, data cannot be transmitted</td>
</tr>
</tbody>
</table>
### J1939_x_SPECIFIC_REQUEST

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

Unit type = function block (FB)

Unit is contained in the library `ifm_J1939_x_Vxyyzz.LIB`

**Symbol in CODESYS:**

- `ENABLE`
- `PRIO`
- `DA`
- `PG`
- `PF`
- `PS`
- `DST`
- `RESULT`
- `LEN`

**Description**

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

**Info**

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

**NOTICE**

Risk of inadmissible overwriting of data!

- Create a receiver array with a size of 1785 bytes. This is the maximum size of a J1939 message.
- Check the amount of received data: the value must not exceed the size of the array created to receive data!

- To the destination address DST applies:
  - Determine the address by means of the operator ADR and assign it to the FB!
  - In addition, the priority (typically 3, 6 or 7) must be assigned.
  - Given that the request of data can be handled via several control cycles, this process must be evaluated via the RESULT byte. All data has been received if RESULT = 1.
  - The output LEN indicates how many data bytes have been received.
Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element&lt;br&gt;FALSE: unit is not executed&lt;br&gt; &gt; Function block inputs are not active&lt;br&gt; &gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>PRIO</td>
<td>BYTE</td>
<td>message priority (0...7)</td>
</tr>
<tr>
<td>DA</td>
<td>BYTE</td>
<td>J1939 address of the requested device</td>
</tr>
<tr>
<td>PG</td>
<td>BYTE</td>
<td>Data page Value of defined PGN (Parameter Group Number) allowed = 0...1 (normally = 0)</td>
</tr>
<tr>
<td>PF</td>
<td>BYTE</td>
<td>PDU format byte Value of defined PGN (Parameter Group Number) PDU1 (specific) = 0...239 PDU2 (global) = 240...255</td>
</tr>
<tr>
<td>PS</td>
<td>BYTE</td>
<td>PDU specific byte Value of defined PGN (Parameter Group Number) If PF = PDU1 ⊝ PS = DA (Destination Address) (DA = J1939 address of external device) If PF = PDU2 ⊝ PS = GE (Group Extension)</td>
</tr>
<tr>
<td>DST</td>
<td>DWORD</td>
<td>destination address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Info</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN  = [Page] + [PF] + [PS]</td>
<td></td>
</tr>
<tr>
<td>PDU  = [PRIO] + [PGN] + [J1939 address] + [data]</td>
<td></td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>BYTE</td>
<td>feedback of the function block (possible messages → following table)</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>Number of received bytes</td>
</tr>
</tbody>
</table>

Possible results for RESULT:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00</td>
<td>FB is inactive</td>
</tr>
<tr>
<td>1 01</td>
<td>FB execution completed without error – data is valid</td>
</tr>
<tr>
<td>2 02</td>
<td>function block is active (action not yet completed)</td>
</tr>
<tr>
<td>3 03</td>
<td>Error</td>
</tr>
</tbody>
</table>
J1939_x_TRANSMIT

x = 1..n = number of the CAN interface (depending on the device, → Data sheet)
Unit type = function block (FB)
Unit is contained in the library ifm_J1939_x_Vxxyyyz.LIB

Symbol in CODESYS:

Description

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

Info

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

► To the source address SRC applies:
  ▶ Determine the address by means of the operator ADR and assign it to the FB!
  ▶ In addition, the number of data bytes to be transmitted and the priority (typically 3, 6 or 7) must be assigned.
  ▶ Given that the transmission of data is processed via several control cycles, the process must be evaluated via the RESULT byte. All data has been transmitted if RESULT = 1.
  ◄ If more than 8 bytes are to be sent, a "multi package transfer" is carried out.
### Parameters of the inputs

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

**Info**

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

### Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| RESULT    | BYTE      | feedback of the function block  
(possible messages → following table) |

#### Possible results for RESULT:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 FB is inactive</td>
</tr>
<tr>
<td>1</td>
<td>01 FB execution completed without error – data is valid</td>
</tr>
<tr>
<td>2</td>
<td>02 function block is active (action not yet completed)</td>
</tr>
<tr>
<td>3</td>
<td>03 Error, data cannot be transmitted</td>
</tr>
</tbody>
</table>
5.2.6 Function elements: serial interface

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIAL_PENDING</td>
<td>125</td>
</tr>
<tr>
<td>SERIAL_RX</td>
<td>126</td>
</tr>
<tr>
<td>SERIAL_SETUP</td>
<td>127</td>
</tr>
<tr>
<td>SERIAL_TX</td>
<td>128</td>
</tr>
</tbody>
</table>

**NOTE**

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

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

Unit type = function block (FB)
Unit is contained in the library `ifm_CR0505_Vxxyyzz.LIB`

**Symbol in CODESYS:**

```
SERIAL_PENDING

NUMBER
```

**Description**

SERIAL_PENDING determines the number of data bytes stored in the serial receive buffer. In contrast to `SERIAL_RX` (→ page 126) the contents of the buffer remain unchanged after calling this FB.

The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface.

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

**NOTE**

In principle, the serial interface is not available for the user, because it is used for program download and debugging.

The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface.

**Parameters of the outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>WORD</td>
<td>Number of data bytes received</td>
</tr>
</tbody>
</table>
SERIAL_RX

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB

Symbol in CODESYS:

Description

SERIAL_RX reads a received data byte from the serial receive buffer at each call. Then, the value of AVAILABLE is decremented by 1. If more than 1000 data bytes are received, the buffer overflows and data is lost. This is indicated by the bit OVERFLOW. If 7-bit data transmission is used, the 8th bit contains the parity and must be suppressed by the user if necessary. The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface. To do so, set the system flag bit SERIAL_MODE=TRUE!

!! NOTE

In principle, the serial interface is not available for the user, because it is used for program download and debugging. The interface can be freely used if the user sets the system flag bit SERIAL_MODE=TRUE. Then however, program download and debugging are only possible via the CAN interface.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLEAR     | BOOL      | TRUE: delete receive buffer  
FALSE: function element is not executed |

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx</td>
<td>BYTE</td>
<td>Byte data received from the receive buffer</td>
</tr>
</tbody>
</table>
| AVAILABLE | WORD      | Number of remaining data bytes  
0 = no valid data available |
| OVERFLOW  | BOOL      | TRUE: Overflow of the data buffer ⇒ loss of data  
FALSE: Data buffer is without data loss |
SERIAL_SETUP

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

Description

SERIAL_SETUP initialises the serial RS232 interface.
The function block does not necessarily need to be executed in order to be able to use the serial interface. Without function block call the default settings below apply.
Using ENABLE=TRUE for one cycle, the function block sets the serial interface to the indicated parameters. The changes made with the help of the function block are saved non-volatile.

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.

NOTICE

The driver module of the serial interface can be damaged!
Disconnecting or connecting the serial interface while live can cause undefined states which damage the driver module.

Do not disconnect or connect the serial interface while live.

Parameters of the inputs

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Diagram of SERIAL_TX block]

Description

SERIAL_TX transmits one data byte via the serial RS232 interface. Using the input ENABLE the transmission can be enabled or blocked. The SERIAL FBs form the basis for the creation of an application-specific protocol for the serial interface. To do so, set the system flag bit SERIAL_MODE=TRUE!

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
</table>
| In principle, the serial interface is not available for the user, because it is used for program download and debugging.
| The interface can be freely used if the user sets the system flag bit SERIAL_MODE to TRUE. Then however, program download and debugging are only possible via the CAN interface. |

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | TRUE: execute this function element  
FALSE: unit is not executed  
> Function block inputs are not active  
> Function block outputs are not specified |
| DATA      | BYTE      | value to be transmitted |
5.2.7 Function elements: Optimising the PLC cycle

Here we show you functions to optimise the PLC cycle.

Function elements: processing interrupts

The PLC cyclically processes the stored application program in its full length. The cycle time can vary due to program branchings which depend e.g. on external events (= conditional jumps). This can have negative effects on certain functions.

By means of systematic interrupts of the cyclic program it is possible to call time-critical processes independently of the cycle in fixed time periods or in case of certain events.

Since interrupt functions are principally not permitted for SafetyControllers, they are thus not available.
SET_INTERRUPT_I

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Diagram](image)

Description

SET_INTERRUPT_I handles the execution of a program part by an interrupt request via an input channel.

In the conventional PLC the cycle time is decisive for real-time monitoring. So the PLC is at a disadvantage as compared to customer-specific controllers. Even a "real-time operating system" does not change this fact when the whole application program runs in one single block which cannot be changed.

A possible solution would be to keep the cycle time as short as possible. This often leads to splitting the application up to several control cycles. This, however, makes programming complex and difficult.

Another possibility is to call a certain program part only upon request by an input pulse independently of the control cycle:

The time-critical part of the application is integrated by the user in a block of the type PROGRAM (PRG). This block is declared as the interrupt routine by calling SET_INTERRUPT_I once (during initialisation). As a consequence, this program block will always be executed if an edge is detected on the input CHANNEL. If inputs and outputs are used in this program part, these are also read and written in the interrupt routine, triggered by the input edge. Reading and writing can be stopped via the FB inputs READ_INPUTS, WRITE_OUTPUTS and ANALOG_INPUTS.

So in the program block all time-critical events can be processed by linking inputs or global variables and writing outputs. So FBs can only be executed if actually called by an input signal.

**NOTE**

The program block should be skipped in the cycle (except for the initialisation call) so that it is not cyclically called, too.

The input (CHANNEL) monitored for triggering the interrupt cannot be initialised and further processed in the interrupt routine.

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

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

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

NOTE

The uniqueness of the inputs and outputs in the cycle is affected by the interrupt routine. Therefore only part of the inputs and outputs is serviced. If initialised in the interrupt program, the following inputs and outputs will be read or written.

**Inputs, digital:**

- %IX0.0...%IX0.7 (Controller: CR0n3n, CR7n3n)
- %IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)
- %IX0.0, %IX0.8 (SmartController: CR250n)
- IN08...IN11 (CabinetController: CR030n)
- IN0...IN3 (PCB controller: CS0015)

**Inputs, analogue:**

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

**Outputs, digital:**

- %QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)
- %QX0.0, %QX0.8 (SafetyController: CR7mm)
- OUT00...OUT03 (CabinetController: CR030n)
- OUT0...OUT7 (PCB controller: CS0015)

Global variants, too, are no longer unique if they are accessed simultaneously in the cycle and by the interrupt routine. This problem applies in particular to larger data types (e.g. DINT).

All other inputs and outputs are processed once in the cycle, as usual.
## Parameters of the inputs

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

```
++
| ENABLE |
| REPEATTIME |
| READ_INPUTS |
| WRITE_OUTPUTS |
| ANALOG_INPUTS |
```

Description

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.

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

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

The uniqueness of the inputs and outputs in the cycle is affected by the interrupt routine. Therefore only part of the inputs and outputs is serviced. If initialised in the interrupt program, the following inputs and outputs will be read or written.

Inputs, digital:

%IX0.0...%IX0.7 (Controller: CR0n3n, CR7n3n)
%IX0.12...%IX0.15, %IX1.4...%IX1.8 (all other ClassicController, ExtendedController, SafetyController)
%IX0.0, %IX0.8 (SmartController: CR250n)
IN08...IN11 (CabinetController: CR030n)
IN0...IN3 (PCB controller: CS0015)

Inputs, analogue:

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

Outputs, digital:

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

Global variants, too, are no longer unique if they are accessed simultaneously in the cycle and by the interrupt routine. This problem applies in particular to larger data types (e.g. DINT).

All other inputs and outputs are processed once in the cycle, as usual.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE (only for 1 cycle): initialisation of the function block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td>REPEATTIME</td>
<td>TIME</td>
<td>Duration in [ms] between end of program and reboot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The duration between two calls is determined as the sum of REPEATTIME and runtime of the program called via interrupt.</td>
</tr>
<tr>
<td>READ_INPUTS</td>
<td>BOOL</td>
<td>TRUE: read the inputs 0...7 before calling the program and write into the input flags I00...I07</td>
</tr>
<tr>
<td>WRITE_OUTPUTS</td>
<td>BOOL</td>
<td>TRUE: write the current values of the output flags Q00...Q07 to the outputs after completion of the program sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: do not write outputs</td>
</tr>
<tr>
<td>ANALOG_INPUTS</td>
<td>BOOL</td>
<td>TRUE: read inputs 0...7 and write the unfiltered, uncalibrated analogue values to the flags ANALOG_IRQ00...07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: do not write flags ANALOG_IRQ00...07</td>
</tr>
</tbody>
</table>
5.2.8 Function elements: processing input values

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

**NOTE**

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

Therefore different raw values can appear in the PLC configuration for identical devices. Error correction and normalisation are only carried out by ifm function blocks. The function blocks provide the corrected value.
**ANALOG_RAW**

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

**Symbol in CODESYS:**

```
  ANALOG_RAW
   P0
   P12
```

**Description**

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

**Parameters of the outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>ARRAY [0..7] OF WORD</td>
<td>Raw input values of the analogue inputs, port 0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P0.0 for I00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P0.7 for I07</td>
</tr>
<tr>
<td>P12</td>
<td>ARRAY [0..7] OF WORD</td>
<td>Raw input values of the analogue inputs, ports 1+2:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P12.0 for I14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P12.3 for I17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P12.4 for I24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P12.7 for I27</td>
</tr>
</tbody>
</table>
INPUT_ANALOG

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

![IFM ClassicController FB Diagram](image)

Description

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.

<table>
<thead>
<tr>
<th>MODE</th>
<th>Input operating mode</th>
<th>Output OUT</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_DIGITAL_H</td>
<td>digital input</td>
<td>0 / 1</td>
<td>---</td>
</tr>
<tr>
<td>IN_CURRENT</td>
<td>current input</td>
<td>0...20 000</td>
<td>µA</td>
</tr>
<tr>
<td>IN_VOLTAGE10</td>
<td>voltage input</td>
<td>0...10 000</td>
<td>mV</td>
</tr>
<tr>
<td>IN_VOLTAGE30</td>
<td>voltage input</td>
<td>0...30 000</td>
<td>mV</td>
</tr>
<tr>
<td>IN_RATIO</td>
<td>voltage input ratiometric</td>
<td>0...1 000</td>
<td>%</td>
</tr>
</tbody>
</table>

For parameter setting of the operating mode, the indicated global system variables should be used. The analogue values are provided as standardised values.

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

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

Parameters of the inputs

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

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>WORD</td>
<td>Output value according to MODE in case of an invalid setting: OUT = &quot;0&quot;</td>
</tr>
</tbody>
</table>
INPUT_CURRENT

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

**Description**

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 137) should be used.

**Parameters of the inputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>INPUT_CHANNEL</td>
<td>BYTE</td>
<td>Number of input channel allowed = 0..7</td>
</tr>
</tbody>
</table>

**Parameters of the outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL_CURRENT</td>
<td>WORD</td>
<td>input current in [µA]</td>
</tr>
</tbody>
</table>
**INPUT_VOLTAGE**

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

```
--- INPUT_VOLTAGE ---
  | ENABLE | ACTUAL_VOLTAGE |
  | MODE_10V_32V | |
  | INPUT_CHANNEL |
```

**Description**

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 137) should be used.

**Parameters of the inputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>MODE_10V_32V</td>
<td>BOOL</td>
<td>TRUE: voltage range 0...32 V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: voltage range 0...10 V</td>
</tr>
<tr>
<td>INPUT_CHANNEL</td>
<td>BYTE</td>
<td>Number of input channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed = 0...7</td>
</tr>
</tbody>
</table>

**Parameters of the outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL_VOLTAGE</td>
<td>WORD</td>
<td>input voltage in [mV]</td>
</tr>
</tbody>
</table>
5.2.9 Function elements: adapting analogue values

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

Description

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
dvalues.

**NOTE**

- The value for X must be in the defined input range between XL and XH!
  There is no internal plausibility check of the value X.
- Due to rounding errors the normalised value can deviate by 1.
- If the limits (XH/XL or YH/YL) are defined in an inverted manner, normalisation is also done in an
  inverted manner.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>WORD</td>
<td>input value</td>
</tr>
<tr>
<td>XH</td>
<td>WORD</td>
<td>Upper limit of input value range [increments]</td>
</tr>
<tr>
<td>XL</td>
<td>WORD</td>
<td>Lower limit of input value range [increments]</td>
</tr>
<tr>
<td>YH</td>
<td>WORD</td>
<td>Upper limit of output value range</td>
</tr>
<tr>
<td>YL</td>
<td>WORD</td>
<td>Lower limit of output value range</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>WORD</td>
<td>output value</td>
</tr>
</tbody>
</table>
**Example: NORM (1)**

<table>
<thead>
<tr>
<th>lower limit value input</th>
<th>0</th>
<th>XL</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper limit value input</td>
<td>100</td>
<td>XH</td>
</tr>
<tr>
<td>lower limit value output</td>
<td>0</td>
<td>YL</td>
</tr>
<tr>
<td>upper limit value output</td>
<td>2000</td>
<td>YH</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>from X =</th>
<th>50</th>
<th>0</th>
<th>100</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>to Y =</td>
<td>1000</td>
<td>0</td>
<td>2000</td>
<td>1500</td>
</tr>
</tbody>
</table>

**Example: NORM (2)**

<table>
<thead>
<tr>
<th>lower limit value input</th>
<th>2000</th>
<th>XL</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper limit value input</td>
<td>0</td>
<td>XH</td>
</tr>
<tr>
<td>lower limit value output</td>
<td>0</td>
<td>YL</td>
</tr>
<tr>
<td>upper limit value output</td>
<td>100</td>
<td>YH</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>from X =</th>
<th>1000</th>
<th>0</th>
<th>2000</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>to Y =</td>
<td>50</td>
<td>100</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>
5.2.10 Function elements: counter functions for frequency and period measurement

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

Due to the different measuring methods errors can occur when the frequency is determined. The following FBs are available for easy evaluation:

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

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

- The increased sensitivity to noise pulses must be taken into account (e.g. contact bouncing for mechanical contacts).
- The standard digital input can evaluate signals up to 50 Hz.
FAST_COUNT

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Diagram of FAST_COUNT function block]

Description

FAST_COUNT operates as counter block for fast input pulses.
This FB detects fast pulses at the FRQ input channels 0...3. With the FRQ input channel 0
FAST_COUNT operates like the block CTU. Maximum input frequency → data sheet.

Due to the technical design, for the 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; counter stopped</td>
</tr>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unit is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the program</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the fast input channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0...3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...3 for the inputs I14...I17</td>
</tr>
<tr>
<td>MODE_UP_DOWN</td>
<td>BOOL</td>
<td>TRUE: counter counts downwards</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: counter counts upwards</td>
</tr>
<tr>
<td>LOAD</td>
<td>BOOL</td>
<td>TRUE: start value PV is loaded in CV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: function element is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>executed</td>
</tr>
<tr>
<td>PV</td>
<td>DWORD</td>
<td>Start value (preset value) for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>counter</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>DWORD</td>
<td>current counter value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Behaviour in case of overflow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the counter stops at 0 when counting downwards</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• there is an overflow when counting upwards</td>
</tr>
</tbody>
</table>
FREQUENCY

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB
Symbol in CODESYS:

Description

FREQUENCY measures the signal frequency at the indicated channel. Maximum input frequency → data sheet.
This FB measures the frequency of the signal at the selected CHANNEL. To do so, the positive edge is evaluated. Depending on the TIMEBASE, frequency measurements can be carried out in a wide value range. High frequencies require a short time base, low frequencies a correspondingly longer time base. The frequency is provided directly in [Hz].

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): unit is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the fast input channel (0...3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...3 for the inputs I14...I17</td>
</tr>
<tr>
<td>TIMEBASE</td>
<td>TIME</td>
<td>Time basis for frequency measurement (max. 57 s)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>REAL</td>
<td>frequency of the input signal in [Hz]</td>
</tr>
</tbody>
</table>
INC_ENCODER

= Incremental Encoder
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

Description

INC_ENCODER offers up/down counter functions for the evaluation of encoders.
Each input pair to be evaluated by means of the function block is formed by two frequency inputs.
Limit frequency = 30 kHz
max. number of units to be connected: 4 encoders (ExtendedController: max. 8 encoders)

Set preset value:
1. Enter value in PRESET_VALUE
2. Set PRESET to TRUE for one cycle
3. Reset PRESET to FALSE
The function block counts the pulses at the inputs as long as INIT=FALSE and PRESET=FALSE.
The current counter value is available at the output COUNTER.
The outputs UP and DOWN indicate the current counting direction of the counter. The outputs are TRUE if the counter has counted in the corresponding direction in the preceding program cycle. If the counter stops, the direction output in the following program cycle is also reset.

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

- FAST_COUNT (→ page 144)
- FREQUENCY (→ page 145)
- PERIOD (→ page 149)
- PERIOD_RATIO (→ page 151)
- PHASE (→ page 153)

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

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

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>TRUE (only for 1 cycle): Function block is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the input channel pair (0...3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = channel pair 0 = inputs I14 + I15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = channel pair 1 = inputs I16 + I17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = channel pair 2 = inputs I24 + I25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = channel pair 3 = inputs I26 + I27</td>
</tr>
<tr>
<td>PRESET_VALUE</td>
<td>DINT</td>
<td>counter start value</td>
</tr>
<tr>
<td>PRESET</td>
<td>BOOL</td>
<td>FALSE -&gt; TRUE (edge): PRESET_VALUE is loaded to COUNTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: Counter ignores the input pulses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: Counter counts the input pulses</td>
</tr>
<tr>
<td>RESOLUTION</td>
<td>BYTE</td>
<td>evaluation of the encoder resolution:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = counts for every fourth edge (= resolution of the encoder)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02 = counts for every second edge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>04 = counts for every rising and falling edge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other values count as &quot;01&quot;.</td>
</tr>
</tbody>
</table>
### Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTER</td>
<td>DINT</td>
<td>Current counter value</td>
</tr>
</tbody>
</table>
| UP        | BOOL      | TRUE: counter counts upwards in the last cycle  
FALSE: counter counts not upwards in the last cycle |
| DOWN      | BOOL      | TRUE: counter counts downwards in the last cycle  
FALSE: counter counts not downwards in the last cycle |
PERIOD

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB
Symbol in CODESYS:

Description
PERIOD measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel.
Maximum input frequency → data sheet.
This FB measures the frequency and the cycle time of the signal at the selected CHANNEL. To calculate, all positive edges are evaluated and the average value is determined by means of the number of indicated PERIODS.
In case of low frequencies there will be inaccuracies when using FREQUENCY. To avoid this, PERIOD can be used. The cycle time is directly indicated in [µs].
The maximum measuring range is approx. 71 min.

NOTE
For PERIOD only the inputs CYL0...CYL3 can be used.
For PDM360smart: CR1071: all inputs.
Frequencies < 0.5 Hz are no longer clearly indicated!

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| INIT | BOOL | FALSE ⇒ TRUE (edge): unit is initialised
FALSE: during further processing of the program |
| CHANNEL | BYTE | Number of the last input channel (0...3)
0...3 for the inputs I24...I27 |
| PERIODS | BYTE | Number of periods to be compared |

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>DWORD</td>
<td>Cycle time of the detected periods in [µs] allowed = 200…10 000 000 = 0xC8…0x989680 (= 10 seconds)</td>
</tr>
<tr>
<td>F</td>
<td>REAL</td>
<td>Frequency of the input signal in [Hz]</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Time elapsed since the last rising edge on the input (can be used for very slow signals)</td>
</tr>
</tbody>
</table>
PERIOD_RATIO

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

Description

PERIOD_RATIO measures the frequency and the cycle period (cycle time) in [µs] during the indicated periods at the indicated channel. In addition, the mark-to-period ratio is indicated in [%]. Maximum input frequency → data sheet.

This FB measures the frequency and the cycle time of the signal at the selected CHANNEL. To calculate, all positive edges are evaluated and the average value is determined by means of the number of indicated PERIODS. In addition, the mark-to-period ratio is indicated in [%].

For example: In case of a signal ratio of 25 ms high level and 75 ms low level the value RATIO1000 is provided as 250 ‰.

In case of low frequencies there will be inaccuracies when using FREQUENCY. To avoid this, PERIOD_RATIO can be used. The cycle time is directly indicated in [µs].

The maximum measuring range is approx. 71 min.

NOTE

For PERIOD_RATIO only the inputs CYL0...CYL3 can be used.
For PDM360smart: CR1071: all inputs.
The output RATIO1000 provides the value 0 for a mark-to-period ratio of 100 % (input signal permanently at supply voltage).
Frequencies < 0.05 Hz are no longer clearly indicated!

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): unit is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the fast input channel (0...3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...3 for the inputs I24...I27</td>
</tr>
<tr>
<td>PERIODS</td>
<td>BYTE</td>
<td>Number of periods to be compared</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>DWORD</td>
<td>Cycle time of the detected periods in [µs] allowed = 200...10 000 000 = 0xC8...0x989680 (= 10 seconds)</td>
</tr>
<tr>
<td>F</td>
<td>REAL</td>
<td>Frequency of the input signal in [Hz]</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Time passed since the last change of state on the input (can be used in case of very slow signals)</td>
</tr>
<tr>
<td>RATIO1000</td>
<td>WORD</td>
<td>For measuring the interval: Mark-to-space ratio in [%] Preconditions: - pulse duration &gt; 100 µs - frequency &lt; 5 kHz for other measurements: RATIO1000 = 0</td>
</tr>
</tbody>
</table>
PHASE

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

Description

PHASE reads a pair of channels with fast inputs and compares the phase position of the signals.
Maximum input frequency → data sheet.
This FB compares a pair of channels with fast inputs so that the phase position of two signals towards each other can be evaluated. An evaluation of the cycle period is possible even in the range of seconds.

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| INIT      | BOOL      | TRUE (edge): unit is initialised
|           |           | FALSE: during further processing of the program |
| CHANNEL   | BYTE      | Number of the input channel pair (0/1) 0 = channel pair 0 = inputs I14 + I15 1 = channel pair 1 = inputs I16 + I17 |

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>DWORD</td>
<td>period duration of the first input's signal of the channel pair in [µs]</td>
</tr>
<tr>
<td>P</td>
<td>INT</td>
<td>angle of the phase shaft valid measurement: 1...358 °</td>
</tr>
<tr>
<td>ET</td>
<td>TIME</td>
<td>Time elapsed since the last positive edge at the second pulse input of the channel pair</td>
</tr>
</tbody>
</table>
5.2.11 Function elements: PWM functions

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

```
OCC_TASK
---
ENABLE
INIT
OUTPUT_CHANNEL
DESIRED_CURRENT
PWM_FREQUENCY
DITHER_FREQUENCY
DITHER_VALUE
MODE
MANUAL
```

Description

OCC_TASK operates as current controller for the PWM outputs.
The controller is designed as an adaptive controller so that it is self-optimising. If the self-optimising performance is not desired, a value > 0 can be transmitted via the input MANUAL (the self-optimising performance is deactivated). The numerical value represents a compensation value, which has an influence on the integral and differential components of the controller. To determine the best settings of the controller in the MANUAL mode, the value 50 is suitable. Depending on the requested controller characteristics the value can then be incremented step-by-step (controller becomes more sensitive / faster) or decremented (controller becomes less sensitive / slower).

If the input MANUAL is set to 0, the controller is always self-optimising. The performance of the controlled system is permanently monitored and the updated compensation values are automatically and permanently stored in each cycle. Changes in the controlled system are immediately recognised and corrected.

⚠️ NOTE

OCC_TASK operates with a fixed cycle time of 5 ms. No actual values need to be entered because these are detected internally by the FB.
OCC_TASK is based on PWM (→ page 160).

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE            | BOOL      | TRUE: execute this function element  
|                   |           | FALSE: unit is not executed  
|                   |           | > Function block inputs are not active  
|                   |           | > Function block outputs are not specified                                                                                                   |
| INIT              | BOOL      | FALSE: TRUE (edge): unit is initialised  
|                   |           | FALSE: during further processing of the program                                                                                                      |
| OUTPUT_CHANNEL    | BYTE      | Number of the current-controlled output channel (0...7)  
|                   |           | 0...3 for the outputs Q10...Q13  
|                   |           | 4...7 for the outputs Q20...Q23                                                                                                                      |
| DESIRED_CURRENT   | WORD      | desired current value of the output in [mA]                                                                                                      |
| PWM_FREQUENCY     | WORD      | PWM frequency [Hz] for load on input                                                                                                             |
| DITHER_FREQUENCY  | WORD      | dither frequency in [Hz]  
|                   |           | value range = 0...FREQUENCY / 2  
|                   |           | FREQUENCY / DITHER_FREQUENCY must be even-numbered!  
|                   |           | The FB increases all other values to the next matching value.                                                                                           |
| DITHER_VALUE      | BYTE      | peak-to-peak value of the dither in [%]  
|                   |           | permissible values = 0...100 = 0x00...0x64                                                                                                           |
| MODE              | BYTE      | Controller characteristics:  
|                   |           | 0 = very slow increase, no overshoot  
|                   |           | 1 = slow increase, no overshoot  
|                   |           | 2 = minimum overshoot  
|                   |           | 3 = moderate overshoot permissible                                                                                                                      |
| MANUAL            | BYTE      | Value = 0: the controller operates in a self-optimising way  
|                   |           | Value > 0: the self-optimising performance of the closed-loop controller is overwritten (typical: 50)                                                                 |

## Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_RATIO</td>
<td>BYTE</td>
<td>for monitoring purposes: display PWM pulse ratio 0...100%</td>
</tr>
</tbody>
</table>
OUTPUT_CURRENT

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxyyzz.LIB

Symbol in CODESYS:

```
ENABLE  OUTPUT_CURRENT  ACTUAL_CURRENT  DITHER_RELATED
```

Description

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE              | BOOL      | TRUE: execute this function element  
|                     |           | FALSE: unit is not executed  
|                     |           | Function block inputs are not active  
|                     |           | Function block outputs are not specified  
| OUTPUT_CHANNEL      | BYTE      | Number of the current-controlled output channel (0...7)  
|                     |           | 0...3 for the outputs Q10...Q13  
|                     |           | 4...7 for the outputs Q20...Q23  
| DITHER_RELATED      | BOOL      | Current is determined as an average value via...  
|                     |           | TRUE: one dither period  
|                     |           | FALSE: one PWM period  

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL_CURRENT</td>
<td>WORD</td>
<td>Output current in [mA]</td>
</tr>
</tbody>
</table>
**OUTPUT_CURRENT_CONTROL**

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

```
<table>
<thead>
<tr>
<th>ENABLE</th>
<th>PWM_RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td></td>
</tr>
<tr>
<td>OUTPUT_CHANNEL</td>
<td></td>
</tr>
<tr>
<td>ACTUAL_CURRENT</td>
<td></td>
</tr>
<tr>
<td>DESIRED_CURRENT</td>
<td></td>
</tr>
<tr>
<td>PWM_FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>DITHER_FREQUENCY</td>
<td></td>
</tr>
<tr>
<td>DITHER_VALUE</td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td></td>
</tr>
<tr>
<td>MANUAL</td>
<td></td>
</tr>
</tbody>
</table>
```

Description

OUTPUT_CURRENT_CONTROL operates as current controller for the PWM outputs.
The controller is designed as an adaptive controller so that it is self-optimising. If this self-optimising performance is not desired, a value > 0 can be transmitted via the input MANUAL; the self-optimising performance is then deactivated. The numerical value represents a compensation value, which has an influence on the integral and differential components of the controller. To determine the best settings of the controller in the MANUAL mode, the value 50 is suitable. Depending on the requested controller characteristics the value can then be incremented step-by-step (controller becomes more sensitive / faster) or decremented (controller becomes less sensitive / slower).

If the input MANUAL is set to 0, the controller is always self-optimising. The performance of the controlled system is permanently monitored and the updated compensation values are automatically and permanently stored in each cycle. Changes in the controlled system are immediately recognised and corrected.

**NOTE**

To obtain a stable output value OUTPUT_CURRENT_CONTROL should be called cyclically at regular intervals.

OUTPUT_CURRENT_CONTROL is based on PWM (→ page 160).

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element&lt;br&gt;FALSE: unit is not executed&lt;br&gt;Function block inputs are not active&lt;br&gt;Function block outputs are not specified</td>
</tr>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE (\Rightarrow) TRUE (edge): unit is initialised&lt;br&gt;FALSE: during further processing of the program</td>
</tr>
<tr>
<td>OUTPUT_CHANNEL</td>
<td>BYTE</td>
<td>Number of the current-controlled output channel (0...7)&lt;br&gt;0...3 for the outputs Q10...Q13&lt;br&gt;4...7 for the outputs Q20...Q23</td>
</tr>
<tr>
<td>ACTUAL_CURRENT</td>
<td>WORD</td>
<td>Actual current on the PWM output in [mA]&lt;br&gt;Transfer the output value of OUTPUT_CURRENT (→ page 157) to the input ACTUAL_CURRENT.</td>
</tr>
<tr>
<td>DESIRED_CURRENT</td>
<td>WORD</td>
<td>desired current value of the output in [mA]</td>
</tr>
<tr>
<td>PWM_FREQUENCY</td>
<td>WORD</td>
<td>PWM frequency [Hz] for load on input</td>
</tr>
<tr>
<td>DITHER_FREQUENCY</td>
<td>WORD</td>
<td>dither frequency in [Hz]&lt;br&gt;value range = 0...FREQUENCY / 2&lt;br&gt;FREQUENCY / DITHER_FREQUENCY must be even-numbered!&lt;br&gt;The FB increases all other values to the next matching value.</td>
</tr>
<tr>
<td>DITHER_VALUE</td>
<td>BYTE</td>
<td>peak-to-peak value of the dither in [%]&lt;br&gt;permissible values = 0...100 = 0x00...0x64</td>
</tr>
<tr>
<td>MODE</td>
<td>BYTE</td>
<td>Controller characteristics:&lt;br&gt;0 = very slow increase, no overshoot&lt;br&gt;1 = slow increase, no overshoot&lt;br&gt;2 = minimum overshoot&lt;br&gt;3 = moderate overshoot permissible</td>
</tr>
<tr>
<td>MANUAL</td>
<td>BYTE</td>
<td>Value = 0: the controller operates in a self-optimising way&lt;br&gt;Value &gt; 0: the self-optimising performance of the closed-loop controller is overwritten (typical: 50)</td>
</tr>
</tbody>
</table>

### Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_RATIO</td>
<td>BYTE</td>
<td>for monitoring purposes: display PWM pulse ratio 0...100%</td>
</tr>
</tbody>
</table>
PWM

Unit type = function block (FB)
Unit is contained in the library ifm_CRO505_Vxxyzz.LIB

Symbol in CODESYS:

Description

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 these channels, PWM and PWM1000 (→ page 166) must not be mixed.
The PWM frequency (and so the RELOAD value) is internally limited to 5 kHz.

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1).

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:
• via OUTPUT_CURRENT (→ page 157)
• or for example using the ifm unit EC2049 (series element for current measurement).

PWM_Dither is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the DIVIDER for the determination of the dither frequency and the VALUE are assigned.

The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.
Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE → TRUE (edge): unit is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>RELOAD</td>
<td>WORD</td>
<td>Value for the determination of the PWM frequency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(→ chapter Calculation of the RELOAD value (→ page 162))</td>
</tr>
<tr>
<td>DIV64</td>
<td>BOOL</td>
<td>CPU cycle / 64</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the PWM output channel (0...7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...3 for the outputs Q10...Q13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4...7 for the outputs Q20...Q23</td>
</tr>
<tr>
<td>VALUE</td>
<td>WORD</td>
<td>Current PWM value permissible = 0...RELOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = switch-on time 100 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RELOAD = switch-on time 0 %</td>
</tr>
<tr>
<td>CHANGE</td>
<td>BOOL</td>
<td>TRUE: Adopting new value from ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VALUE: after the current PWM period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DITHER_VALUE: after the current Dither period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: the changed PWM value has no influence on the output</td>
</tr>
<tr>
<td>DITHER_VALUE</td>
<td>WORD</td>
<td>peak-to-peak value of the dither in [%]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permissible values = 0...1 000 = 0000...03E8</td>
</tr>
<tr>
<td>DITHER_DIVIDER</td>
<td>WORD</td>
<td>Dither frequency = PWM frequency / DIVIDER * 2</td>
</tr>
</tbody>
</table>

**PWM frequency**

Depending on the valve type, a corresponding PWM frequency is required. For the PWM function the PWM frequency is transmitted via the reload value (PWM) or directly as a numerical value in [Hz] (PWM1000). Depending on the controller, the PWM outputs differ in their operating principle but the effect is the same.

The PWM frequency is implemented by means of an internally running counter, derived from the CPU pulse. This counter is started with the initialisation of the PWM. Depending on the PWM output group (0...3 and / or 4...7 or 4...11), it counts from 0xFFFF backwards or from 0x0000 forwards. If a transmitted comparison value (VALUE) is reached, the output is set. In case of an overflow of the counter (change of the counter reading from 0x0000 to 0xFFFF or from 0xFFFF to 0x0000), the output is reset and the operation restarts.

If this internal counter shall not operate between 0x0000 and 0xFFFF, another preset value (RELOAD) can be transmitted for the internal counter. In doing so, the PWM frequency increases. The comparison value must be within the now specified range.
Calculation of the RELOAD value

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

<table>
<thead>
<tr>
<th>DIV64</th>
<th>RELOAD value</th>
<th>DIV64</th>
<th>RELOAD value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20 MHz / fPWM</td>
<td>1</td>
<td>10 MHz / fPWM</td>
</tr>
</tbody>
</table>

Depending on whether a high or a low PWM frequency is required, the input DIV64 must be set to FALSE (0) or TRUE (1). In case of frequencies below 305 Hz respectively 152 Hz (according to the controller), DIV64 must be set to "1" to ensure that the RELOAD value is not greater than 0xFFFF.

Calculation examples RELOAD value

The PWM frequency shall be 400 Hz.

\[
\text{RELOAD} = \frac{20 \text{ MHz}}{400 \text{ Hz}} = 50 000 = 0xC350
\]

Thus the permissible range of the PWM value is the range from 0x0000...0xC350.

The comparison value at which the output switches must then be between 0x0000 and 0xC350.

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

<table>
<thead>
<tr>
<th>Mark-to-space ratio</th>
<th>Switch-on time</th>
<th>Value for mark-to-space ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0 %</td>
<td>50 000 = 0xC350</td>
</tr>
<tr>
<td>Maximum</td>
<td>100 %</td>
<td>0 = 0x0000</td>
</tr>
</tbody>
</table>

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

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

In order to prevent abrupt changes from one PWM value to the next, e.g. from 15 % ON to 70 % ON, it is possible to delay the increase by using PT1 (→ page 192). The ramp function used for PWM is based on the CODESYS library UTIL.LIB. This allows a smooth start e.g. for hydraulic systems.

**NOTE**

When installing the ecomatmobile DVD "Software, tools and documentation", projects with examples have been stored in the program directory of your PC:

- \ifm\ electronic\CoDeSys V...\Projects\DEMO_PLC_DVD_V... (for controllers) or
- \ifm\ electronic\CoDeSys V...\Projects\DEMO_PDM_DVD_V... (for PDMs).

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

**NOTE**

The PWM function of the controller is a hardware function provided by the processor. The PWM function remains set until a hardware reset (power off and on) has been carried out on the controller.
PWM100

Unit type = function block (FB)

New ecomatmobile controllers only support PWM100 (→ page 166).

Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![PWM100 block diagram]

Description

PWM100 handles the initialisation and parameter setting of the PWM outputs. The FB enables a simple application of the PWM FB in the ecomatmobile controller. The PWM frequency can be directly indicated in [Hz] and the mark-to-space ratio in steps of 1 %. This FB is not suited for use in controllers, due to the relatively coarse grading.

The FB is called once for each channel in the initialisation of the application program. For this, the input INIT must be set to TRUE. During initialisation, the parameter FREQUENCY is also assigned.

NOTE

The value FREQUENCY must be identical for the channels 4...7. For these channels, PWM (→ page 160) and PWM100 must not be mixed.

The PWM frequency is limited to 5 kHz internally.

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:
- via OUTPUT_CURRENT (→ page 157)
- or for example using the ifm unit EC2049 (series element for current measurement).

DITHER is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the value FREQUENCY for determining the dither frequency and the dither value (VALUE) are transmitted.

The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.
## ifm function elements for the device CR0505

### Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): unit is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>FREQUENCY</td>
<td>WORD</td>
<td>PWM frequency in [Hz] allowed = 20...250 = 0x0014...0x00FA</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the PWM output channel (0...7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0…3 for the outputs Q10...Q13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4…7 for the outputs Q20...Q23</td>
</tr>
<tr>
<td>VALUE</td>
<td>BYTE</td>
<td>current PWM value</td>
</tr>
<tr>
<td>CHANGE</td>
<td>BOOL</td>
<td>TRUE: Adopting new value from ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VALUE: after the current PWM period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DITHER_VALUE: after the current Dither period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: the changed PWM value has no influence on the output</td>
</tr>
<tr>
<td>DITHER_VALUE</td>
<td>BYTE</td>
<td>peak-to-peak value of the dither in [%]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permissible values = 0…100 = 0x00…0x64</td>
</tr>
<tr>
<td>DITHER_FREQUENCY</td>
<td>WORD</td>
<td>dither frequency in [Hz] value range = 0...FREQUENCY / 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FREQUENCY / DITHER_FREQUENCY must be even-numbered!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The FB increases all other values to the next matching value.</td>
</tr>
</tbody>
</table>
PWM1000

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

```
  INIT
  FREQUENCY
  CHANNEL
  VALUE
  CHANGE
  DITHER_VALUE
  DITHER_FREQUENCY
```

**Description**

PWM1000 handles the initialisation and parameter setting of the PWM outputs. The FB enables a simple use of the PWM FB in the *ecomatmobile* device. The PWM frequency can be directly indicated in [Hz] and the mark-to-space ratio in steps of 1 ‰.

The FB is called once for each channel during initialisation of the application program. When doing so, input INIT must be set to TRUE. During initialisation, the parameter FREQUENCY is also assigned.

**NOTE**

- The value FREQUENCY must be identical for the channels 4...7.
- For these channels, PWM (→ page 160) and PWM1000 must not be mixed.
- The PWM frequency is limited to 5 kHz internally.

During cyclical processing of the program INIT is set to FALSE. The FB is called and the new PWM value is assigned. The value is adopted if the input CHANGE = TRUE.

A current measurement for the initialised PWM channel can be implemented:
- via OUTPUT_CURRENT (→ page 157)
- 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.

**NOTE**

- The parameters DITHER_FREQUENCY and DITHER_VALUE can be individually set for each channel.
## Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE $\Rightarrow$ TRUE (edge): unit is initialised</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: during further processing of the program</td>
</tr>
<tr>
<td>FREQUENCY</td>
<td>WORD</td>
<td>PWM frequency in [Hz]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed = 20...250 = 0x0014...0x00FA</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the PWM output channel (0...7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...3 for the outputs Q10...Q13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4...7 for the outputs Q20...Q23</td>
</tr>
<tr>
<td>VALUE</td>
<td>WORD</td>
<td>PWM value (mark-to-space ratio) in [%]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed = 0...1 000 = 0x0000...0x03E8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Values &gt; 1 000 are regarded as = 1 000</td>
</tr>
<tr>
<td>CHANGE</td>
<td>BOOL</td>
<td>TRUE: adoption of the new value of ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FREQUENCY: after the current PWM period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VALUE: after the current PWM period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DITHER_VALUE: after the current dither period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DITHER_FREQUENCY: after the current dither period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: the changed PWM value has no influence on the output</td>
</tr>
<tr>
<td>DITHER_VALUE</td>
<td>WORD</td>
<td>peak-to-peak value of the dither in [%]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permissible values = 0...1 000 = 0000...03E8</td>
</tr>
<tr>
<td>DITHER_FREQUENCY</td>
<td>WORD</td>
<td>dither frequency in [Hz]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value range = 0...FREQUENCY / 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FREQUENCY / DITHER_FREQUENCY must be even-numbered!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The FB increases all other values to the next matching value.</td>
</tr>
</tbody>
</table>
5.2.12 Function elements: hydraulic control

The function elements for the device CR0505 are designed for hydraulic control. The library `ifm_hydraulic_16bitS05_Vxxyyz.LIB` contains the following function blocks:

- **CONTROL_OCC**
  - OCC = Output Current Control
  - Scales the input value [WORD] to an indicated current range

- **JOYSTICK_0**
  - Scales signals [INT] from a joystick to clearly defined characteristic curves, standardised to 0...1000

- **JOYSTICK_1**
  - Scales signals [INT] from a joystick standardised to 0...1000

- **JOYSTICK_2**
  - Scales signals [INT] from a joystick to a configurable characteristic curve; free selection of the standardisation

- **NORM_HYDRAULIC**
  - Normalises a value [DINT] within defined limits to a value with new limits

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

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

The following function blocks are needed from the library `ifm_CR0505_Vxxyyz.LIB`:
- **OUTPUT_CURRENT**
  - Measures the current (average via dither period) on an output channel

- **OUTPUT_CURRENT_CONTROL**
  - Current controller for a PWMi output channel

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

Unit type = function block (FB)
Unit is contained in the library ifm_HYDRAULIC_16bit0505_Vxxyyzz.lib

Symbol in CODESYS:

`CONTROL_OCC`  
<table>
<thead>
<tr>
<th>ENABLE</th>
<th>DESIRED_CURRENT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>ACTUAL_CURRENT</td>
<td></td>
</tr>
<tr>
<td>R_RAMP</td>
<td>BREAK</td>
<td></td>
</tr>
<tr>
<td>F_RAMP</td>
<td>SHORT</td>
<td></td>
</tr>
<tr>
<td>TIMEBASE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX_CURRENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN_CURRENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOLERANCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHANNEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWM_FREQUENCY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTHER_FREQUENCY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTHER_VALUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MANUAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

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

Each instance of the FB is called once in each PLC cycle. The FB uses OUTPUT_CURRENT_CONTROL (→ page 158) and OUTPUT_CURRENT (→ page 157) from the library ifm_CR0505_Vxxyyzz.lib. The controller is designed as an adaptive controller so that it is self-optimising.

If this self-optimising performance is not desired, a value > 0 can be transferred via the input MANUAL: → the self-optimising performance is deactivated.

The numerical value in MANUAL represents a compensation value, which has an influence on the integral and differential components of the controller. To determine the best settings of the controller in the MANUAL mode, the value 50 is suitable.

Increase the value MANUAL: → controller becomes more sensitive / faster
Decrease the value MANUAL: → controller becomes less sensitive / slower

If the input MANUAL is set to "0", the controller is always self-optimising. The performance of the controlled system is permanently monitored and the updated compensation values are automatically and permanently stored in each cycle. Changes in the controlled system are immediately recognised and corrected.

The input X of CONTROL_OCC should be supplied by the output of the JOYSTICK FBs.
### Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element  &lt;br&gt;FALSE: unit is not executed  &lt;br&gt;</td>
</tr>
<tr>
<td>INIT</td>
<td>BOOL</td>
<td>FALSE ⇒ TRUE (edge): unit is initialised  &lt;br&gt;FALSE: during further processing of the program</td>
</tr>
<tr>
<td>R_RAMP</td>
<td>INT</td>
<td>Rising edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE]  &lt;br&gt;0 = without ramp</td>
</tr>
<tr>
<td>F_RAMP</td>
<td>INT</td>
<td>Falling edge of the ramp in [increments/PLC cycle] or [increments/TIMEBASE]  &lt;br&gt;0 = without ramp</td>
</tr>
<tr>
<td>TIMEBASE</td>
<td>TIME</td>
<td>Reference for rising and falling edge of the ramp:  &lt;br&gt;t#0s = rising/falling edge in [increments/PLC cycle]  &lt;br&gt;Fast controllers have very short cycle times!  &lt;br&gt;otherwise = rising/falling edge in [increments/TIMEBASE]</td>
</tr>
<tr>
<td>X</td>
<td>WORD</td>
<td>input value</td>
</tr>
<tr>
<td>XH</td>
<td>WORD</td>
<td>Upper limit of input value range [increments]</td>
</tr>
<tr>
<td>XL</td>
<td>WORD</td>
<td>Lower limit of input value range [increments]</td>
</tr>
<tr>
<td>MAX_CURRENT</td>
<td>WORD</td>
<td>Max. valve current in [mA]</td>
</tr>
<tr>
<td>MIN_CURRENT</td>
<td>WORD</td>
<td>Min. valve current in [mA]</td>
</tr>
<tr>
<td>TOLERANCE</td>
<td>BYTE</td>
<td>Tolerance for min. valve current in [increments].  &lt;br&gt;When the tolerance is exceeded, jump to MIN_CURRENT is effected</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>BYTE</td>
<td>Number of the current-controlled output channel (0...7)  &lt;br&gt;0...3 for the outputs Q10...Q13  &lt;br&gt;4...7 for the outputs Q20...Q23</td>
</tr>
<tr>
<td>PWM_FREQUENCY</td>
<td>WORD</td>
<td>PWM frequency [Hz] for load on input</td>
</tr>
<tr>
<td>DITHER_FREQUENCY</td>
<td>WORD</td>
<td>dither frequency in [Hz]  &lt;br&gt;value range = 0...FREQUENCY / 2  &lt;br&gt;FREQUENCY / DITHER_FREQUENCY must be even-numbered!  &lt;br&gt;The FB increases all other values to the next matching value.</td>
</tr>
<tr>
<td>DITHER_VALUE</td>
<td>BYTE</td>
<td>peak-to-peak value of the dither in [%]  &lt;br&gt;permissible values = 0...100 = 0x00...0x64</td>
</tr>
<tr>
<td>MODE</td>
<td>BYTE</td>
<td>Controller characteristics:  &lt;br&gt;0 = very slow increase, no overshoot  &lt;br&gt;1 = slow increase, no overshoot  &lt;br&gt;2 = minimum overshoot  &lt;br&gt;3 = moderate overshoot permissible</td>
</tr>
<tr>
<td>MANUAL</td>
<td>BYTE</td>
<td>Value = 0: the controller operates in a self-optimising way  &lt;br&gt;Value &gt; 0: the self-optimising performance of the closed-loop controller is overwritten (typical: 50)</td>
</tr>
</tbody>
</table>
### Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIRED_CURRENT</td>
<td>WORD</td>
<td>Desired current value in [mA] for OCC (for monitoring purposes)</td>
</tr>
<tr>
<td>ACTUAL_CURRENT</td>
<td>WORD</td>
<td>Output current in [mA]</td>
</tr>
<tr>
<td>BREAK</td>
<td>BOOL</td>
<td>Error: cable interrupted at output</td>
</tr>
<tr>
<td>SHORT</td>
<td>BOOL</td>
<td>Error: short circuit in cable at output</td>
</tr>
</tbody>
</table>
JOYSTICK_0

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

Symbol in CODESYS:

<table>
<thead>
<tr>
<th>X</th>
<th>OUT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XH_POS</td>
<td>OUT2</td>
</tr>
<tr>
<td>XL_POS</td>
<td>OUT3</td>
</tr>
<tr>
<td>XH_NEG</td>
<td>ERROR_MODE</td>
</tr>
<tr>
<td>XL_NEG</td>
<td>ERR1</td>
</tr>
<tr>
<td>MODE</td>
<td>ERR2</td>
</tr>
</tbody>
</table>

Description

JOYSTICK_0 scales signals from a joystick to clearly defined characteristic curves, standardised to 0...1000.

For this FB the characteristic curve values are specified (→ figures):
- Rising edge of the ramp = 5 increments/PLC cycle
  - Fast Controllers have a very short cycle time!
- Falling edge of the ramp = no ramp

The parameters XL_POS (XL+), XH_POS (XH+), XL_NEG (XL-) and XH_NEG (XH-) are used to evaluate the joystick movements only in the requested area.
The values for the positive and negative area may be different. The values for XL_NEG and XH_NEG are negative here.

Mode 0:
characteristic curve linear for the range XL to XH
Mode 1:  
Characteristic curve linear with dead band  
Values fixed to:  
Dead band:  
0…10% of 1000 increments

Mode 2:  
2-step linear characteristic curve with dead band  
Values fixed to:  
Dead band:  
0…10% of 1000 increments  
Step:  
X = 50 % of 1000 increments  
Y = 20 % of 1000 increments

Characteristic curve mode 3:  
Curve rising (line is fixed)
### Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>INT</td>
<td>Input value [increments]</td>
</tr>
<tr>
<td>XH_POS</td>
<td>INT</td>
<td>Max. preset value positive direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>XL_POS</td>
<td>INT</td>
<td>Min. preset value positive direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>XH_NEG</td>
<td>INT</td>
<td>Max. preset value negative direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>XL_NEG</td>
<td>INT</td>
<td>Min. preset value negative direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>MODE</td>
<td>BYTE</td>
<td>Mode selection characteristic curve:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = linear (X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = linear with dead band (X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = 2-step linear with dead band (X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = curve rising (line is fixed)</td>
</tr>
</tbody>
</table>

### Parameters of the outputs

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

Possible results for ERR1 and ERR2:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00</td>
<td>no error</td>
</tr>
<tr>
<td>1 01</td>
<td>Error in array: wrong sequence</td>
</tr>
<tr>
<td>2 02</td>
<td>Error: Input value IN is not contained in value range of array</td>
</tr>
<tr>
<td>4 04</td>
<td>Error: invalid number N for array</td>
</tr>
</tbody>
</table>
JOYSTICK_1

Unit type = function block (FB)
Unit is contained in the library ifm_hydraulic_16bitOS05_Vxxyzz.Lib

Symbol in CODESYS:

Description

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

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

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

Possible results for ERR1 and ERR2:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no error</td>
</tr>
<tr>
<td>1</td>
<td>Error in array: wrong sequence</td>
</tr>
<tr>
<td>2</td>
<td>Error: Input value IN is not contained in value range of array</td>
</tr>
<tr>
<td>4</td>
<td>Error: invalid number N for array</td>
</tr>
</tbody>
</table>
JOYSTICK_2

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

Symbol in CODESYS:

Description

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>INT</td>
<td>Input value [increments]</td>
</tr>
<tr>
<td>XH_POS</td>
<td>INT</td>
<td>Max. preset value positive direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>XL_POS</td>
<td>INT</td>
<td>Min. preset value positive direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>XH_NEG</td>
<td>INT</td>
<td>Max. preset value negative direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>XL_NEG</td>
<td>INT</td>
<td>Min. preset value negative direction [increments] (negative values also permissible)</td>
</tr>
<tr>
<td>R_RAMP</td>
<td>INT</td>
<td>Rising edge of the ramp in [increments/PLC cycle] 0 = no ramp</td>
</tr>
<tr>
<td>F_RAMP</td>
<td>INT</td>
<td>Falling edge of the ramp in [increments/PLC cycle] 0 = no ramp</td>
</tr>
</tbody>
</table>
| TIMEBASE      | TIME          | Reference for rising and falling edge of the ramp:  
|               |               | t#0s = rising/falling edge in [increments/PLC cycle]  
|               |               | Fast controllers have very short cycle times!  
|               |               | otherwise = rising/falling edge in [increments/TIMEBASE]                   |
| VARIABLE_GAIN | ARRAY [0..10] OF POINT | Pairs of values describing the curve  
|               |               | The first pairs of values indicated in N_POINT are used. n = 2...11  
|               |               | Example: 9 pairs of values declared as variable VALUES:  
|               |               | VALUES := ARRAY [0..10] OF POINT :=  
|               |               | (X:=0,Y:=0), (X:=200,Y:=0), (X:=300,Y:=50),  
|               |               | (X:=400,Y:=100), (X:=700,Y:=500),  
|               |               | (X:=1000,Y:=900), (X:=1100,Y:=950),  
|               |               | (X:=1200,Y:=1000), (X:=1400,Y:=1050);  
|               |               | There may be blanks between the values.                                    |
| N_POINT       | BYTE          | Number of points (pairs of values in VARIABLE_GAIN) by which the curve characteristic is defined: n = 2...11 |
## Parameters of the Outputs

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

### Possible Results for ERR1 and ERR2:

<table>
<thead>
<tr>
<th>Value dec</th>
<th>hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>no error</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Error in array: wrong sequence</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Error: Input value IN is not contained in value range of array</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>Error: invalid number N for array</td>
</tr>
</tbody>
</table>
NORM_HYDRAULIC

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

Description

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

This function block corresponds to NORM_DINT from the CODESYS library UTIL.Lib.
The function block standardises a value of type DINT, which is within the limits of XH and XL, to an
output value within the limits of YH and YL.
Due to rounding errors deviations from the standardised value of 1 may occur. If the limits (XH/XL or
YH/YL) are indicated in inversed form, standardisation is also inverted.
If X is outside the limits of XL…XH, the error message will be X_OUT_OF_RANGE = TRUE.

Typical characteristic curve of a hydraulic valve:
The oil flow will not start before 20% of the coil current has been reached.
At first the oil flow is not linear.

Characteristics of the function block
Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>DINT</td>
<td>current input value</td>
</tr>
<tr>
<td>XH</td>
<td>DINT</td>
<td>Max. input value [increments]</td>
</tr>
<tr>
<td>XL</td>
<td>DINT</td>
<td>Min. input value [increments]</td>
</tr>
<tr>
<td>YH</td>
<td>DINT</td>
<td>Max. output value [increments], e.g.: valve current [mA] / flow [l/min]</td>
</tr>
<tr>
<td>YL</td>
<td>DINT</td>
<td>Min. output value [increments], e.g.: valve current [mA], flow [l/min]</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>DINT</td>
<td>output value</td>
</tr>
<tr>
<td>X_OUT_OF_RANGE</td>
<td>BOOL</td>
<td>Error: X is beyond the limits of XH and XL</td>
</tr>
</tbody>
</table>

Example: NORM_HYDRAULIC

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper limit value input XH</td>
<td>100</td>
<td>100</td>
<td>2000</td>
</tr>
<tr>
<td>Lower limit value input XL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Upper limit value output YH</td>
<td>2000</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Lower limit value output YL</td>
<td>0</td>
<td>2000</td>
<td>0</td>
</tr>
<tr>
<td>Non standardised value X</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Standardised value Y</td>
<td>400</td>
<td>1600</td>
<td>1</td>
</tr>
</tbody>
</table>

- Case 1:
  Input with relatively coarse resolution.
  Output with high resolution.
  1 X increment results in 20 Y increments.

- Case 2:
  Input with relatively coarse resolution.
  Output with high resolution.
  1 X increment results in 20 Y increments.
  Output signal is inverted as compared to the input signal.

- Case 3:
  Input with high resolution.
  Output with relatively coarse resolution.
  20 X increments result in 1 Y increment.
5.2.13 Function elements: controllers

Setting rule for a controller

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

At the beginning the controlling system is operated as a purely P-controlling system. In this respect the derivative time $TV$ is set to 0 and the reset time $TN$ to a very high value (ideally to $\infty$) for a slow system. For a fast controlled system a small $TN$ should be selected.

Afterwards the gain $KP$ is increased until the control deviation and the adjustment deviation perform steady oscillation at a constant amplitude at $KP = KP_{critical}$. Then the stability limit has been reached.

Then the time period $T_{critical}$ of the steady oscillation has to be determined.

Add a differential component only if necessary. $TV$ should be approx. 2...10 times smaller than $TN$.

KP should be equal to KD.

Idealised setting of the controlled system:

<table>
<thead>
<tr>
<th>Control unit</th>
<th>$KP = KD$</th>
<th>$TN$</th>
<th>$TV$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$2.0 \cdot KP_{critical}$</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PI</td>
<td>$2.2 \cdot KP_{critical}$</td>
<td>$0.83 \cdot T_{critical}$</td>
<td>—</td>
</tr>
<tr>
<td>PID</td>
<td>$1.7 \cdot KP_{critical}$</td>
<td>$0.50 \cdot T_{critical}$</td>
<td>$0.125 \cdot T_{critical}$</td>
</tr>
</tbody>
</table>

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

To dampen overshoot $PT1$ (→ page 192) (low pass) can be used. In this respect the preset value $XS$ is damped by the PT1 link before it is supplied to the controller function.

The setting variable $T1$ should be approx. 4...5 times greater than $TN$ of the controller.
DELAY

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Diagram of DELAY function block]

Description

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

![Time characteristics of DELAY]

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

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>WORD</td>
<td>input value</td>
</tr>
<tr>
<td>T</td>
<td>TIME</td>
<td>Delay time (dead time)</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>WORD</td>
<td>input value, delayed by the time T</td>
</tr>
</tbody>
</table>
GLR

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Symbol Diagram]

Description

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>WORD</td>
<td>actual value channel 1</td>
</tr>
<tr>
<td>X2</td>
<td>WORD</td>
<td>actual value channel 2</td>
</tr>
<tr>
<td>XS</td>
<td>WORD</td>
<td>preset value</td>
</tr>
<tr>
<td>XMAX</td>
<td>WORD</td>
<td>maximum preset value</td>
</tr>
<tr>
<td>KP</td>
<td>Byte</td>
<td>constant of the proportional component (/10) (positive values only!)</td>
</tr>
<tr>
<td>TN</td>
<td>TIME</td>
<td>integral action time (integral component)</td>
</tr>
<tr>
<td>KD</td>
<td>BYTE</td>
<td>differential component (/10) (positive values only!)</td>
</tr>
<tr>
<td>TV</td>
<td>TIME</td>
<td>derivative action time (differential component)</td>
</tr>
<tr>
<td>RESET</td>
<td>BOOL</td>
<td>TRUE: reset the function element; FALSE: function element is not executed</td>
</tr>
</tbody>
</table>

## Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y1</td>
<td>WORD</td>
<td>manipulated variable channel 1</td>
</tr>
<tr>
<td>Y2</td>
<td>WORD</td>
<td>manipulated variable channel 2</td>
</tr>
</tbody>
</table>
PID1

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

Description

PID1 handles a PID controller.
The change of the manipulated variable of a PID controller has a proportional, integral and differential component. The manipulated variable changes first by an amount which depends on the rate of change of the input value (D component). After the end of the derivative action time the manipulated variable returns to the value corresponding to the proportional range and changes in accordance with the reset time.

NOTE

The manipulated variable Y is already standardised to the PWM FB (RELOAD value = 65,535). Note the reverse logic:
65,535 = minimum value
0 = maximum value.
Note that the input values KI and KD depend on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

If X > 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>WORD</td>
<td>input value</td>
</tr>
<tr>
<td>XS</td>
<td>WORD</td>
<td>preset value</td>
</tr>
<tr>
<td>XMAX</td>
<td>WORD</td>
<td>maximum preset value</td>
</tr>
<tr>
<td>KP</td>
<td>BYTE</td>
<td>proportional component of the output signal</td>
</tr>
<tr>
<td>KI</td>
<td>BYTE</td>
<td>integral component of the output signal</td>
</tr>
<tr>
<td>KD</td>
<td>BYTE</td>
<td>differential component of the output signal</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>WORD</td>
<td>Manipulated variable (0...1000 ‰)</td>
</tr>
</tbody>
</table>

Recommended settings

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

**Symbol in CODESYS:**

![Diagram of PID2 block](image)

**Description**

PID2 handles a PID controller with self optimisation.

The change of the manipulated variable of a PID controller has a proportional, integral and differential component. The manipulated variable changes first by an amount which depends on the rate of change of the input value (D component). After the end of the derivative action time TV the manipulated variable returns to the value corresponding to the proportional component and changes in accordance with the reset time TN.

The values entered at the inputs KP and KD are internally divided by 10. So, a finer grading can be obtained (e.g.: KP = 17, which corresponds to 1.7).

**NOTE**

The manipulated variable Y is already standardised to the PWM FB (RELOAD value = 65,535). Note the reverse logic:
- 65,535 = minimum value
- 0 = maximum value.

Note that the input value KD depends on the cycle time. To obtain stable, repeatable control characteristics, the FB should be called in a time-controlled manner.

If X > 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](image)
Parameters of the inputs

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

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>WORD</td>
<td>Manipulated variable (0...1000 ‰)</td>
</tr>
</tbody>
</table>

Recommended setting

- Select TN according to the time characteristics of the system:
  - fast system = small TN
  - slow system = large TN
- Slowly increment KP gradually, up to a value at which still definitely no fluctuation will occur.
- Readjust TN if necessary.
- Add differential component only if necessary:
  - Select a TV value approx. 2...10 times smaller than TN.
  - Select a KD value more or less similar to KP.

Note that the maximum control deviation is ± 127. For good control characteristics this range should not be exceeded, but it should be exploited to the best possible extent.

Function input SO (self-optimisation) clearly improves the control performance. A precondition for achieving the desired characteristics:
- The controller is operated with I component (TN > 50 ms)
- Parameters KP and especially TN are already well adjusted to the actual controlled system.
- The control range (X – XS) of ± 127 is utilised (if necessary, increase the control range by multiplying X, XS and XMAX).
- When you have finished setting the parameters, you can set SO = TRUE.
  > This will significantly improve the control performance, especially reducing overshoot.
PT1

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyzz.LIB

Symbol in CODESYS:

```
X   PT1   Y
```

Description

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):

\[
y(t) = \frac{1}{1 + \frac{T_1}{t}}
\]

Figure: Time characteristics of PT1

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>INT</td>
<td>Input value [increments]</td>
</tr>
<tr>
<td>T1</td>
<td>TIME</td>
<td>Delay time (time constant)</td>
</tr>
</tbody>
</table>

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>INT</td>
<td>output value</td>
</tr>
</tbody>
</table>
5.2.14 Function elements: software reset

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Symbol Diagram]

Description

SOFTRESET leads to a complete reboot of the device.
The FB can for example be used in conjunction with CANopen if a node reset is to be carried out. FB
SOFTRESET executes an immediate reboot of the controller. The current cycle is not completed.
Before reboot, the retain variables are stored.
The reboot is logged in the error memory.

⚠️ In case of active communication: the long reset period must be taken into account because
otherwise guarding errors will be signalled.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | TRUE: execute this function element
|           |           | FALSE: unit is not executed
|           |           |   > Function block inputs are not active
|           |           |   > Function block outputs are not specified |
5.2.15 Function elements: measuring / setting of time

Using the following function blocks of ifm electronic you can...
• measure time and evaluate it in the application program,
• change time values, if required.
**TIMER_READ**

Unit type = function block (FB)  
Unit is contained in the library `ifm_CR0505_Vxxyyzz.LIB`

Symbol in CODESYS:

```
T
```

Description

TIMER_READ reads the current system time.  
When the supply voltage is applied, the device generates a clock pulse which is counted upwards in a register. This register can be read using the FB call and can for example be used for time measurement.

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

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>TIME</td>
<td>Current system time [ms]</td>
</tr>
</tbody>
</table>
### TIMER_READ_US

Unit type = function block (FB)

Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

**Symbol in CODESYS:**

```
  Timer_Read_US
```

**Description**

TIMER_READ_US reads the current system time in [µs].

When the supply voltage is applied, the device generates a clock pulse which is counted upwards in a register. This register can be read by means of the FB call and can for example be used for time measurement.

**Info**

The system timer runs up to the counter value 4 294 967 295 µs at the maximum and then starts again from 0.

4 294 967 295 µs = 1h 11min 34s 967ms 295µs

**Parameters of the outputs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_US</td>
<td>DWORD</td>
<td>current system time [µs]</td>
</tr>
</tbody>
</table>
5.2.16 Function elements: saving, reading and converting data in the memory

Storage types for data backup
The device provides the following memory types:

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

FRAM memory
FRAM indicates here all kinds of non-volatile and fast memories.
Properties:
• fast writing and reading
• unlimited writing and reading frequency
• any memory area can be selected
• secure data with FRAMWRITE
• read data with FRAMREAD
Automatic saving of data

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

A distinction is made between variables declared as RETAIN and variables in the flag area which can be configured as a remanent block with MEMORY_RETAIN_PARAM (→ page 200).

Details → chapter Variables (→ page 66).

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB
Symbol in CODESYS:

![MEMORY_RETAIN_PARAM symbol]

Description

MEMORY_RETAIN_PARAM determines the remanent data behaviour for various events. Variables declared as VAR_RETAIN in CODESYS have a remanent behaviour from the outset.
Remanent data keep their value (as the variables declared as VAR_RETAIN) after an uncontrolled termination as well as after normal switch off and on of the controller. After a restart the program continues to work with the stored values.
For groups of events that can be selected (with MODE), this function block determines how many (LEN) data bytes (from flag byte %MB0) shall have retain behaviour even if they have not been explicitly declared as VAR_RETAIN.

<table>
<thead>
<tr>
<th>Event</th>
<th>MODE = 0</th>
<th>MODE = 1</th>
<th>MODE = 2</th>
<th>MODE = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power OFF ⇒ ON</td>
<td>Data is newly initialised</td>
<td>Data is remanent</td>
<td>Data is remanent</td>
<td>Data is remanent</td>
</tr>
<tr>
<td>Soft reset</td>
<td>Data is newly initialised</td>
<td>Data is remanent</td>
<td>Data is remanent</td>
<td>Data is remanent</td>
</tr>
<tr>
<td>Cold reset</td>
<td>Data is newly initialised</td>
<td>Data is newly initialised</td>
<td>Data is remanent</td>
<td>Data is remanent</td>
</tr>
<tr>
<td>Reset default</td>
<td>Data is newly initialised</td>
<td>Data is newly initialised</td>
<td>Data is newly initialised</td>
<td>Data is remanent</td>
</tr>
<tr>
<td>Load application program</td>
<td>Data is newly initialised</td>
<td>Data is newly initialised</td>
<td>Data is remanent</td>
<td>Data is remanent</td>
</tr>
<tr>
<td>Load runtime system</td>
<td>Data is newly initialised</td>
<td>Data is newly initialised</td>
<td>Data is newly initialised</td>
<td>Data is remanent</td>
</tr>
</tbody>
</table>

If MODE = 0, only those data have retain behaviour as with MODE=1 which have been explicitly declared as VAR_RETAIN.
If the FB is never called, the flag bytes act according to MODE = 0. The flag bytes which are above the configured area act according to MODE = 0, too.
Once a configuration has been made, it remains on the device even if the application or the runtime system is reloaded.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | TRUE: execute this function element  
FALSE: unit is not executed  
> Function block inputs are not active  
> Function block outputs are not specified |
| LEN       | WORD      | Number of data bytes from flag address %MB0 onwards to show remanent behaviour  
allowed = 0...4 096 = 0x0...0x100  
LEN > 4 096 will be corrected automatically to LEN = 4 096 |
| MODE      | BYTE      | Events for which these variables shall have retain behaviour  
(0...3; → table above)  
For MODE > 3 the last valid setting will remain |
Manual data storage

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

By means of the storage partitioning (→ chapter Available memory (→ page 15)) the programmer can find out which memory area is available.
FLASHREAD

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB

Symbol in CODESYS:

Description

FLASHREAD enables reading of different types of data directly from the flash memory.

> The FB reads the contents as from the address of SRC from the flash memory. In doing so, as many bytes as indicated under LEN are transmitted.
> The contents are read completely during the cycle in which the FB is called up.

► Please make sure that the target memory area in the RAM is sufficient.
► To the destination address DST applies:

1. Determine the address by means of the operator ADR and assign it to the FB!

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block inputs are not active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>SRC</td>
<td>WORD</td>
<td>relative source start address in the memory</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>permissible = 0...65 535 = 0x0000...0xFFFF</td>
</tr>
<tr>
<td>DST</td>
<td>DWORD</td>
<td>start address of the destination variable</td>
</tr>
</tbody>
</table>

1. Determine the address by means of the operator ADR and assign it to the FB!
FLASHWRITE

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB

Symbol in CODESYS:

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: unit is not executed</td>
</tr>
<tr>
<td>DST</td>
<td>WORD</td>
<td>relative destination start address in the memory</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>number of data bytes</td>
</tr>
<tr>
<td>SRC</td>
<td>DWORD</td>
<td>start address of the source variables</td>
</tr>
</tbody>
</table>
```

Description

**WARNING**

Danger due to uncontrollable process operations!
The status of the inputs/outputs is "frozen" during execution of FLASHWRITE.
▷ Do not execute this FB when the machine is running!

FLASHWRITE enables writing of different data types directly into the flash memory.
Using this FB, large data volumes are to be stored during set-up, to which there is only read access in the process.
▷ If a page has already been written (even if only partly), the entire flash memory area needs to be deleted before new write access to this page. This is done by write access to the address 0.
▷ Never write to a page several times! Always delete everything first!
   Otherwise, traps or watchdog errors occur.
▷ Do not delete the flash memory area more often than 100 times. Otherwise, the data consistency in other flash memory areas is no longer guaranteed.
▷ During each SPS cycle, FLASHWRITE may only be started once!
▷ To the destination address DST applies:
   ▶ Determine the address by means of the operator ADR and assign it to the FB!
▷ The FB writes the contents of the address SRC into the flash memory. In doing so, as many bytes as indicated under LEN are transmitted.
   ▶ If start address SRC is outside the permissible range: no data transfer!

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element</td>
</tr>
<tr>
<td>DST</td>
<td>WORD</td>
<td>relative destination start address in the memory</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>number of data bytes</td>
</tr>
<tr>
<td>SRC</td>
<td>DWORD</td>
<td>start address of the source variables</td>
</tr>
</tbody>
</table>

© ifm electronic sensorics gmbh
FRAMREAD

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>SRC</th>
<th>LEN</th>
<th>DST</th>
</tr>
</thead>
</table>

Description

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

The FB reads the contents as from the address of SRC from the FRAM memory. In doing so, as many bytes as indicated under LEN are transmitted.
If the FRAM memory area were to be exceeded by the indicated number of bytes, only the data up to the end of the FRAM memory area will be read.

► To the destination address DST applies:

Determine the address by means of the operator ADR and assign it to the FB!

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE    | BOOL      | TRUE: execute this function element  
|           |           | FALSE: unit is not executed  
|           |           | Function block inputs are not active  
|           |           | Function block outputs are not specified |
| SRC       | WORD      | relative source start address in the memory  
|           |           | zulässig = 0...1 023 = 0x0000...0x03FF |
| LEN       | WORD      | number of data bytes |
| DST       | DWORD     | start address of the destination variable  

Determine the address by means of the operator ADR and assign it to the FB!
FRAMWRITE

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

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

Description

FRAMWRITE enables the quick writing of different data types directly into the FRAM memory ¹). The FB writes the contents of the address SRC to the non-volatile FRAM memory. In doing so, as many bytes as indicated under LEN are transmitted. If the FRAM memory area were to be exceeded by the indicated number of bytes, only the data up to the end of the FRAM memory area will be written.

► To the source address SRC applies:

[1] Determine the address by means of the operator ADR and assigne it to the FB!

[1] If the target address DST is outside the permissible range: no data transfer!

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

Parameters of the inputs

```
Description
```

- **ENABLE**: BOOL
  - TRUE: execute this function element
  - FALSE: unit is not executed
  - Function block inputs are not active
  - Function block outputs are not specified

- **DST**: WORD
  - relative destination start address in the memory
  - permissible = 0...1 023 = 0x0000...0x03FF

- **LEN**: WORD
  - number of data bytes

- **SRC**: DWORD
  - start address of the source variables
  - Determine the address by means of the operator ADR and assigne it to the FB!
MEMCPY

= memory copy
Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

Description

MEMCPY enables writing and reading different types of data directly in the memory. The FB writes the contents of the address of SRC to the address DST.

To the addresses SRC and DST apply:

1. Determine the address by means of the operator ADR and assign it to the FB!

2. In doing so, as many bytes as indicated under LEN are transmitted. So it is also possible to transmit exactly one byte of a word variable.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DST</td>
<td>DWORD</td>
<td>destination address</td>
</tr>
<tr>
<td>SRC</td>
<td>DWORD</td>
<td>Start address in source memory</td>
</tr>
<tr>
<td>LEN</td>
<td>WORD</td>
<td>number (≥ 1) of the data bytes to be transmitted</td>
</tr>
</tbody>
</table>
5.2.17 Function elements: data access and data check

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_DATA</td>
<td>208</td>
</tr>
<tr>
<td>GET_IDENTITY</td>
<td>210</td>
</tr>
<tr>
<td>SET_DEBUG</td>
<td>211</td>
</tr>
<tr>
<td>SET_IDENTITY</td>
<td>212</td>
</tr>
<tr>
<td>SET_PASSWORD</td>
<td>213</td>
</tr>
</tbody>
</table>

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyyz.LIB

Symbol in CODESYS:

Description

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

► Create a separate instance of the function block for each memory area to be monitored.
► Determine the address by means of the operator ADR and assign it to the FB!
► In addition, indicate the number of data bytes LENGTH (length from the STARTADR).

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

Desired change:
Data changes in the memory (e.g. of the application program or ecomatmobile device) are only permitted if the output UPDATE is set to TRUE. The value of the checksum is then recalculated. The output RESULT is permanently TRUE again.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTADR</td>
<td>DINT</td>
<td>start address of the monitored data memory (WORD address as from %MW0)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>WORD</td>
<td>length of the monitored data memory in [byte]</td>
</tr>
</tbody>
</table>
| UPDATE    | BOOL      | TRUE: changes to data permissible   
FALSE: changes to data not permitted |

Parameters of the outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| RESULT    | BOOL      | TRUE: CRC checksum ok   
FALSE: CRC checksum faulty (data modified) |
| CHECKSUM  | DWORD     | current CRC checksum |
Example: CHECK_DATA

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;
c1 : CHECK_DATA;
ok : BOOL;
pt : POINTER TO WORD;
END_VAR
```

```
16#2D30 SUB pt
```

```
16#100 ADD 2
```

```
SUB CHECK_DATA m1 pt
```

```
TRUE m1
```

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

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ENABLE      | BOOL      | TRUE: execute this function element
            |           | FALSE: unit is not executed
            |           | > Function block inputs are not active
            |           | > Function block outputs are not specified      |
```

Parameters of the inputs

Description

GET_IDENTITY reads the specific identifications stored in the device:
- hardware name and hardware version of the device
- name of the runtime system in the device
- version and revision no. of the runtime system in the device
- name of the application (has previously been saved by means of SET_IDENTITY (→ page 212))

Parameters of the outputs

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICENAME</td>
<td>STRING(31)</td>
<td>hardware name as a string of max. 31 characters, e.g.: &quot;CR0403&quot;</td>
</tr>
<tr>
<td>FIRMWARE</td>
<td>STRING(31)</td>
<td>Name of the runtime system in the device as character string of max. 31 characters e.g.: &quot;CR0403&quot;</td>
</tr>
<tr>
<td>RELEASE</td>
<td>STRING(31)</td>
<td>software version as a character string of max. 31 characters</td>
</tr>
<tr>
<td>APPLICATION</td>
<td>STRING(79)</td>
<td>Name of the application as a string of max. 79 characters e.g.: &quot;Crane1704&quot;</td>
</tr>
</tbody>
</table>
```
SET_DEBUG

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

![Symbol Diagram]

Description

SET_DEBUG handles the DEBUG mode without active test input (→ chapter TEST mode (→ page 47)). If the input DEBUG of the FB is set to TRUE, the programming system or the downloader, for example, can communicate with the device and execute system commands (e.g. for service functions via the GSM modem CANremote).

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE: execute this function element&lt;br&gt;FALSE: unit is not executed&lt;br&gt; &gt; Function block inputs are not active&lt;br&gt; &gt; Function block outputs are not specified</td>
</tr>
<tr>
<td>DEBUG</td>
<td>BOOL</td>
<td>TRUE: debugging via the interfaces possible&lt;br&gt;FALSE: debugging via the interfaces not possible</td>
</tr>
</tbody>
</table>
SET_IDENTITY

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB
Symbol in CODESYS:

Description

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):

* BOOTLD_H 020923
* CR0020 00.00.01
* CR0020 V2.0.0 041004
* CR0020 00.00.01
* Nozzle in front *)

Downloader reads:
BOOTLD_H 020923
CR0020 00.00.01

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

CANopen tool reads:
Hardware version
OBV 1009
CR0020 00.00.01

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

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>STRING(80)</td>
<td>Any text with a maximum length of 80 characters</td>
</tr>
</tbody>
</table>
SET_PASSWORD

Unit type = function block (FB)
Unit is contained in the library ifm_CR0505_Vxxyyzz.LIB

Symbol in CODESYS:

Description

SET_PASSWORD sets a user password for the program and memory upload with the DOWNLOADER.
If the password is activated, reading of the application program or the data memory with the software tool DOWNLOADER is only possible if the correct password has been entered.
If an empty string (default condition) is assigned to the input PASSWORD, an upload of the application software or of the data memory is possible at any time.
A new password can be set only after resetting the previous password.

The password is reset when loading a new application program.

Parameters of the inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>BOOL</td>
<td>TRUE (nur 1 Zyklus lang): use new Parameter</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>STRING(16)</td>
<td>password if PASSWORD = &quot;&quot;, then access is possible without enter of a password</td>
</tr>
</tbody>
</table>
6 Diagnosis and error handling

Contents

Diagnosis ........................................................................................................................................... 214
Fault................................................................................................................................................... 214
Reaction in case of an error .............................................................................................................. 215
Relay: important notes! ..................................................................................................................... 215
Response to system errors ............................................................................................................... 216
CAN / CANopen: errors and error handling ................................................................................... 216

The runtime-system (RTS) checks the device by internal error checks:
• during the boot phase (reset phase)
• during executing the application program
→ chapter Operating states (→ page 43)
In so doing a high operating reliability is provided, as much as possible.

6.1 Diagnosis

During the diagnosis, the "state of health" of the device is checked. It is to be found out if and what faults are given in the device.
Depending on the device, the inputs and outputs can also be monitored for their correct function.
- wire break,
- short circuit,
- value outside range.
For diagnosis, configuration and log data can be used, created during the "normal" operation of the device.
The correct start of the system components is monitored during the initialisation and start phase.
Errors are recorded in the log file.
For further diagnosis, self-tests can also be carried out.

6.2 Fault

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

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

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

**NOTE**

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

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

**Complete list of the device-specific error codes and diagnostic messages**

→ chapter System flags (→ page 217).

6.4 Relay: important notes!

Using the logic function via the system flag RELAIS or RELAY_CLAMP_15 (→ chapter Latching (→ page 16)) all other outputs are also switched off. Depending on the application it must now be decided whether by resetting the system flag bit ERROR the relay – and so also the outputs – may be switched on again.

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

**NOTICE**

Premature wear of the relay contacts possible.

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

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

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

In case of serious errors the system flag bit ERROR can additionally be set. At the same time, ERROR = TRUE leads to the following:
- set all relevant outputs to FALSE via the application program,
- the operation LED lights red,
- the ERROR output is set to FALSE and
- the output relays switch off.
- So the outputs protected via these relays are switched off.

After analysis and elimination of the error cause:
- as a general rule, reset all error flags via the application program.
  Without explicit reset of the error flags the flags remain set with the corresponding effect on the application program.

6.6  CAN / CANopen: errors and error handling

→ System manual "Know-How ecomatmobile"
→ chapter CAN / CANopen: errors and error handling
7 Annex

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

7.1 System flags

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

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

→ System manual "Know-How ecomatmobile"
    → chapter Error codes and diagnostic information
## 7.1.1 System flags: CAN

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANx_BAUDRATE</td>
<td>WORD</td>
<td>CAN interface x: set baud rate in [kBaud]</td>
</tr>
<tr>
<td>CANx_BUSOFF</td>
<td>BOOL</td>
<td>CAN interface x: Error &quot;CAN-Bus off&quot; Reset of the error code also resets the flag</td>
</tr>
<tr>
<td>CANx_LASTERROR</td>
<td>BYTE</td>
<td>CAN interface x: Error number of the last CAN transmission:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = no error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = stuff error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = form error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = ack error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = bit1 error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 = bit0 error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = CRC error</td>
</tr>
<tr>
<td>CANx_WARNING</td>
<td>BOOL</td>
<td>CAN interface x: warning threshold reached (&gt; 96)</td>
</tr>
<tr>
<td>DOWNLOADID</td>
<td>WORD</td>
<td>CAN interface x: set download identifier</td>
</tr>
</tbody>
</table>

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

## 7.1.2 System flags: SAE J1939

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1939_TASK</td>
<td>BOOL</td>
<td>Using J1939_TASK, the time requirement for sending J1939 messages is met.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If J1939 messages are to be sent with a repetition time &lt; 50 ms, the runtime system automatically sets J1939_TASK=TRUE. For applications for which the time requirement is &gt; PLC cycle time: Reduce system load with J1939_TASK=FALSE! TRUE: J1939 task is active (= initial value) The task is called every 2 ms. The J1939 stack sends its messages in the required time frame FALSE: J1939 task is not active</td>
</tr>
</tbody>
</table>
## 7.1.3 System flags: error flags (standard side)

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>BOOL</td>
<td>TRUE = set group error message, switch off relay</td>
</tr>
<tr>
<td>ERROR_BREAK_Qx (0...x)</td>
<td>WORD</td>
<td>output group x: wire break error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Bit 0 for output 0] ... [bit z for output z] of this group</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit = TRUE: error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit = FALSE: no error</td>
</tr>
<tr>
<td>ERROR_Ix (0...x)</td>
<td>BYTE</td>
<td>input group x: periphery fault</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Bit 0 for input 0] ... [bit z for input z] of this group</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit = TRUE: error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit = FALSE: no error</td>
</tr>
<tr>
<td>ERROR_MEMORY</td>
<td>BOOL</td>
<td>memory error</td>
</tr>
<tr>
<td>ERROR_POWER</td>
<td>BOOL</td>
<td>Voltage error for VBBS / clamp 15:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: Value out of range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or: difference (VBB15 - VBBS) too great</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; general error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: Value OK</td>
</tr>
<tr>
<td>ERROR_SHRT_HW</td>
<td>WORD</td>
<td>output group x: short circuit error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Bit 0 for output 0] ... [bit z for output z] of this group</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit = TRUE: error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit = FALSE: no error</td>
</tr>
<tr>
<td>ERROR_TEMPERATURE</td>
<td>BOOL</td>
<td>Temperature error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: Value out of range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; general error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: Value OK</td>
</tr>
<tr>
<td>ERROR_VBBx</td>
<td>BOOL</td>
<td>Supply voltage error on VBBx (x = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: Value out of range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; general error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: Value OK</td>
</tr>
</tbody>
</table>
### 7.1.4 System flags: LED (standard side)

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| LED WORD                   | WORD | LED color for "LED switched on":
| 0x0000 = LED_GREEN (preset) |      |             |
| 0x0001 = LED_BLUE          |      |             |
| 0x0002 = LED_RED           |      |             |
| 0x0003 = LED_WHITE         |      |             |
| 0x0004 = LED_BLACK         |      |             |
| 0x0005 = LED_MAGENTA       |      |             |
| 0x0006 = LED_CYAN          |      |             |
| 0x0007 = LED_YELLOW        |      |             |
| LED_X WORD                 | WORD | LED color for "LED switched off":
| 0x0000 = LED_GREEN         |      |             |
| 0x0001 = LED_BLUE          |      |             |
| 0x0002 = LED_RED           |      |             |
| 0x0003 = LED_WHITE         |      |             |
| 0x0004 = LED_BLACK (preset) |      |             |
| 0x0005 = LED_MAGENTA       |      |             |
| 0x0006 = LED_CYAN          |      |             |
| 0x0007 = LED_YELLOW        |      |             |
| LED_MODE WORD              | WORD | LED flashing frequency:
| 0x0000 = LED_2HZ (flashes at 2 Hz; preset) | | |
| 0x0001 = LED_1HZ (flashes at 1 Hz) | | |
| 0x0002 = LED_05HZ (flashes at 0.5 Hz) | | |
| 0x0003 = LED_0HZ (lights permanently with value in LED) | | |
| 0x0004 = LED_5HZ (flashes at 5 Hz) | | |
## 7.1.5 System flags: voltages (standard side)

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAMP_15</td>
<td>BOOL</td>
<td>Voltage monitoring on clamp 15</td>
</tr>
<tr>
<td>RELAIS</td>
<td>BOOL</td>
<td>TRUE: relay energised outputs are supplied with voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: relay de-energised outputs are voltage-free</td>
</tr>
<tr>
<td>RELAIS_CLAMP_15</td>
<td>BOOL</td>
<td>Relay clamp 15 (pin 5)</td>
</tr>
<tr>
<td>SERIAL_BAUDRATE</td>
<td>WORD</td>
<td>Baud rate of the RS232 interface</td>
</tr>
<tr>
<td>SERIAL_MODE</td>
<td>BOOL</td>
<td>Activate serial interface (RS232) for use in the application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: The RS232 interface can be used in the application, but no longer for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>programming, debugging or monitoring of the device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: The RS232 interface cannot be used in the application. Programming,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>debugging or monitoring of the device is possible.</td>
</tr>
<tr>
<td>SUPPLY_VOLTAGE</td>
<td>WORD</td>
<td>supply voltage at VBBS in [mV]</td>
</tr>
<tr>
<td>TEST</td>
<td>BOOL</td>
<td>TRUE: Test input is active: • Programming mode is enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Software download is possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Status of the application program can be queried</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Protection of stored software is not possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: application is in operation</td>
</tr>
</tbody>
</table>

## 7.1.6 System flags: 16..24 inputs and 8..0 outputs (standard side)

<table>
<thead>
<tr>
<th>System flags (symbol name)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALOGx</td>
<td>WORD</td>
<td>Analogue input xx: filtered A/D converter raw value (12 bits) without calibration or standardisation</td>
</tr>
<tr>
<td>Ixx</td>
<td>BOOL</td>
<td>Status on binary input xx</td>
</tr>
<tr>
<td>xx = 00...07 / 10...17 / 20...27</td>
<td></td>
<td>Condition: input is configured as binary input (MODE = IN_DIGITAL_H or IN_DIGITAL_L)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE: Voltage on binary input &gt; 70 % of VBBS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: Voltage on binary input &lt; 30 % of VBBS or: not configured as binary input or: wrong configuration</td>
</tr>
<tr>
<td>Ixx_MODE</td>
<td>BYTE</td>
<td>Operating mode of the input Ixx</td>
</tr>
<tr>
<td>xx = 00...07 / 14...17 / 24...27</td>
<td></td>
<td>→ chapter Possible operating modes inputs/outputs (→ page 225)</td>
</tr>
<tr>
<td>Qxx</td>
<td>BOOL</td>
<td>Status on extended binary input xx: Condition: output is configured as binary output</td>
</tr>
<tr>
<td>xx = 10...13 / 20...23</td>
<td></td>
<td>TRUE: output activated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE: output de-activated (= initial value) or: not configured as binary output</td>
</tr>
<tr>
<td>Qxx_MODE</td>
<td>BYTE</td>
<td>Operating mode of the output Qxx</td>
</tr>
<tr>
<td>xx = 10...13 / 20...23</td>
<td></td>
<td>→ chapter Possible operating modes inputs/outputs (→ page 225)</td>
</tr>
</tbody>
</table>
Annex

Address assignment and I/O operating modes

### 7.2 Address assignment and I/O operating modes

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address assignment inputs / outputs</td>
<td>222</td>
</tr>
<tr>
<td>Possible operating modes inputs/outputs</td>
<td>225</td>
</tr>
<tr>
<td>Addresses / variables of the I/Os</td>
<td>228</td>
</tr>
</tbody>
</table>

→ also data sheet

### 7.2.1 Address assignment inputs / outputs

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs: address assignment (16...24 inputs)</td>
<td>223</td>
</tr>
<tr>
<td>Outputs: address assignment (0...8 outputs)</td>
<td>224</td>
</tr>
</tbody>
</table>

2371
Inputs: address assignment (16...24 inputs)

Abbreviations → chapter Note on wiring (→ page 29)
Operating modes of the inputs/outputs → chapter Possible operating modes inputs/outputs (→ page 225)

<table>
<thead>
<tr>
<th>IEC address</th>
<th>Symbolic address</th>
</tr>
</thead>
<tbody>
<tr>
<td>%IX0.00</td>
<td>I00 ANALOG0</td>
</tr>
<tr>
<td>%IW03</td>
<td></td>
</tr>
<tr>
<td>%IX0.01</td>
<td>I01 ANALOG1</td>
</tr>
<tr>
<td>%IW3</td>
<td></td>
</tr>
<tr>
<td>%IX0.02</td>
<td>I02 ANALOG2</td>
</tr>
<tr>
<td>%IW4</td>
<td></td>
</tr>
<tr>
<td>%IX0.03</td>
<td>I03 ANALOG3</td>
</tr>
<tr>
<td>%IW5</td>
<td></td>
</tr>
<tr>
<td>%IX0.04</td>
<td>I04 ANALOG4</td>
</tr>
<tr>
<td>%IW6</td>
<td></td>
</tr>
<tr>
<td>%IX0.05</td>
<td>I05 ANALOG5</td>
</tr>
<tr>
<td>%IW7</td>
<td></td>
</tr>
<tr>
<td>%IX0.06</td>
<td>I06 ANALOG6</td>
</tr>
<tr>
<td>%IW8</td>
<td></td>
</tr>
<tr>
<td>%IX0.07</td>
<td>I07 ANALOG7</td>
</tr>
<tr>
<td>%IW9</td>
<td></td>
</tr>
<tr>
<td>%IX0.08</td>
<td>I10</td>
</tr>
<tr>
<td>%IX0.09</td>
<td>I11</td>
</tr>
<tr>
<td>%IX0.10</td>
<td>I12</td>
</tr>
<tr>
<td>%IX0.11</td>
<td>I13</td>
</tr>
<tr>
<td>%IX0.12</td>
<td>I14</td>
</tr>
<tr>
<td>%IX0.13</td>
<td>I15</td>
</tr>
<tr>
<td>%IX0.14</td>
<td>I16</td>
</tr>
<tr>
<td>%IX0.15</td>
<td>I17</td>
</tr>
<tr>
<td>%IX1.00</td>
<td>I20</td>
</tr>
<tr>
<td>%IX1.01</td>
<td>I21</td>
</tr>
<tr>
<td>%IX1.02</td>
<td>I22</td>
</tr>
<tr>
<td>%IX1.03</td>
<td>I23</td>
</tr>
<tr>
<td>%IX1.04</td>
<td>I24</td>
</tr>
<tr>
<td>%IX1.05</td>
<td>I25</td>
</tr>
<tr>
<td>%IX1.06</td>
<td>I26</td>
</tr>
<tr>
<td>%IX1.07</td>
<td>I27</td>
</tr>
</tbody>
</table>
**Outputs: address assignment (0...8 outputs)**

Abbreviations → chapter *Note on wiring* (→ page 29)
Operating modes of the inputs/outputs → chapter *Possible operating modes inputs/outputs* (→ page 225)

<table>
<thead>
<tr>
<th>IEC address</th>
<th>Symbolic address</th>
</tr>
</thead>
<tbody>
<tr>
<td>%QX0.00</td>
<td>Q10</td>
</tr>
<tr>
<td>%QX0.01</td>
<td>Q11</td>
</tr>
<tr>
<td>%QX0.02</td>
<td>Q12</td>
</tr>
<tr>
<td>%QX0.03</td>
<td>Q13</td>
</tr>
<tr>
<td>%QX0.04</td>
<td>Q20</td>
</tr>
<tr>
<td>%QX0.05</td>
<td>Q21</td>
</tr>
<tr>
<td>%QX0.06</td>
<td>Q22</td>
</tr>
<tr>
<td>%QX0.07</td>
<td>Q23</td>
</tr>
</tbody>
</table>
## 7.2.2 Possible operating modes inputs/outputs

### Contents

- Inputs: operating modes (16...24 inputs) ................................................................. 225
- Outputs: operating modes (0...8 outputs) ................................................................. 227

### Inputs: operating modes (16...24 inputs)

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

- `=` this configuration value is default

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Possible operating mode</th>
<th>Set with FB</th>
<th>FB input</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>I00...I07</td>
<td>IN_NOMODE</td>
<td>Off</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIGITAL_H</td>
<td>plus</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_CURRENT</td>
<td>0...20 000 µA</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_VOLTAGE10</td>
<td>0...10 000 mV</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_VOLTAGE30</td>
<td>0...30 000 mV</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_RATIO</td>
<td>0...1 000 %</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIAGNOSTIC</td>
<td>with IN_DIGITAL_H</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_FAST</td>
<td>with IN_DIGITAL_H</td>
<td>internal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IN_FAST</td>
<td>0...30 000 Hz</td>
<td>FREQUENCY</td>
<td>Frequency measurement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1...5 000 Hz</td>
<td>PERIOD</td>
<td>Period duration measurement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1...5 000 Hz</td>
<td>PERIOD_RATIO</td>
<td>Period duration and ratio measurement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...30 000 Hz</td>
<td>FAST_COUNT</td>
<td>Counters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0...30 000 Hz</td>
<td>INC_ENCODER</td>
<td>Detect encoder</td>
</tr>
<tr>
<td>I10...I13</td>
<td>IN_NOMODE</td>
<td>Off</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIGITAL_H</td>
<td>plus</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td>I14...I17</td>
<td>IN_NOMODE</td>
<td>Off</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIGITAL_H</td>
<td>plus</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIAGNOSTIC</td>
<td>with IN_DIGITAL_H</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_FAST</td>
<td>with IN_DIGITAL_H</td>
<td>internal</td>
<td></td>
</tr>
<tr>
<td>I20...I23</td>
<td>IN_NOMODE</td>
<td>Off</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIGITAL_H</td>
<td>plus</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td>I24...I27</td>
<td>IN_NOMODE</td>
<td>Off</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIGITAL_H</td>
<td>plus</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIGITAL_L</td>
<td>minus</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_DIAGNOSTIC</td>
<td>with IN_DIGITAL_H</td>
<td>INPUT_ANALOG</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td>IN_FAST</td>
<td>with IN_DIGITAL_H</td>
<td>internal</td>
<td></td>
</tr>
</tbody>
</table>
Annex

Address assignment and I/O operating modes

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Possible operating mode</th>
<th>Set with FB</th>
<th>FB input</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dec</td>
</tr>
<tr>
<td>IN_FAST</td>
<td>0...1 000 Hz</td>
<td>FREQUENCY PHASE</td>
<td>Frequency measurement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1...1 000 Hz</td>
<td>PERIOD</td>
<td>Period duration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1...1 000 Hz</td>
<td>PERIOD_RATIO</td>
<td>Period duration and ratio</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0...1 000 Hz</td>
<td>FAST_COUNT</td>
<td>Counters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0...1 000 Hz</td>
<td>INC_ENCODER</td>
<td>Detect encoder</td>
<td></td>
</tr>
</tbody>
</table>

Set operating modes with the following function block:

- **FAST_COUNT** ([→ page 144](#)): Counter block for fast input pulses
- **FREQUENCY** ([→ page 145](#)): Measures the frequency of the signal arriving at the selected channel
- **INC_ENCODER** ([→ page 146](#)): Up/Down counter function for the evaluation of encoders
- **INPUT_ANALOG** ([→ page 137](#)): Current and voltage measurement on the analogue input channel
- **PERIOD** ([→ page 149](#)): Measures the frequency and the cycle period (cycle time) in [µs] at the indicated channel
- **PERIOD_RATIO** ([→ page 151](#)): Measures the frequency and the cycle period (cycle time) in [µs] during the indicated periods at the indicated channel. In addition, the mark-to-space ratio is indicated in [‰].
- **PHASE** ([→ page 153](#)): Reads a pair of channels with fast inputs and compares the phase position of the signals
### Outputs: operating modes (0...8 outputs)

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

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Possible operating mode</th>
<th>Value set with</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>dec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hex</td>
</tr>
<tr>
<td>Q10...Q13</td>
<td>OUT_NOMODE</td>
<td>Off</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>OUT_DIGITAL_H</td>
<td>plus</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>OUT_CURRENT</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>OUT_DIAGNOSTIC</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>OUT_OVERLOAD_PROTECTION</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>Q20...Q23</td>
<td>OUT_NOMODE</td>
<td>Off</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>OUT_DIGITAL_H</td>
<td>plus</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>OUT_CURRENT</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>OUT_DIAGNOSTIC</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>OUT_OVERLOAD_PROTECTION</td>
<td></td>
<td>128</td>
</tr>
</tbody>
</table>

### Outputs: permitted operating modes

<table>
<thead>
<tr>
<th>Operating mode</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
<th>Q20</th>
<th>Q21</th>
<th>Q22</th>
<th>Q23</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT_NOMODE</td>
<td>Off</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUT_DIGITAL_H</td>
<td>plus</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUT_CURRENT</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUT_DIAGNOSTIC</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUT_OVERLOAD_PROTECTION</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PWM</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
7.2.3 Addresses / variables of the I/Os

Contents

Inputs: addresses and variables (16...24 inputs) ................................................................. 229
Outputs: addresses and variables (0...8 outputs) ............................................................... 229

2376
## Annex

### Address assignment and I/O operating modes

#### Inputs: addresses and variables (16...24 inputs)

<table>
<thead>
<tr>
<th>IEC address</th>
<th>I/O variable</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>%QB8</td>
<td>I00_MODE</td>
<td>Configuration byte for %IX0.00 (I00)</td>
</tr>
<tr>
<td>%QB9</td>
<td>I01_MODE</td>
<td>Configuration byte for %IX0.01 (I01)</td>
</tr>
<tr>
<td>%QB10</td>
<td>I02_MODE</td>
<td>Configuration byte for %IX0.02 (I02)</td>
</tr>
<tr>
<td>%QB11</td>
<td>I03_MODE</td>
<td>Configuration byte for %IX0.03 (I03)</td>
</tr>
<tr>
<td>%QB12</td>
<td>I04_MODE</td>
<td>Configuration byte for %IX0.04 (I04)</td>
</tr>
<tr>
<td>%QB13</td>
<td>I05_MODE</td>
<td>Configuration byte for %IX0.05 (I05)</td>
</tr>
<tr>
<td>%QB14</td>
<td>I06_MODE</td>
<td>Configuration byte for %IX0.06 (I06)</td>
</tr>
<tr>
<td>%QB15</td>
<td>I07_MODE</td>
<td>Configuration byte for %IX0.07 (I07)</td>
</tr>
<tr>
<td>%QB16</td>
<td>I14_MODE</td>
<td>Configuration byte for %IX0.14 (I14)</td>
</tr>
<tr>
<td>%QB17</td>
<td>I15_MODE</td>
<td>Configuration byte for %IX0.15 (I15)</td>
</tr>
<tr>
<td>%QB18</td>
<td>I16_MODE</td>
<td>Configuration byte for %IX0.16 (I16)</td>
</tr>
<tr>
<td>%QB19</td>
<td>I17_MODE</td>
<td>Configuration byte for %IX0.17 (I17)</td>
</tr>
<tr>
<td>%QB20</td>
<td>I24_MODE</td>
<td>Configuration byte for %IX1.04 (I24)</td>
</tr>
<tr>
<td>%QB21</td>
<td>I25_MODE</td>
<td>Configuration byte for %IX1.05 (I25)</td>
</tr>
<tr>
<td>%QB22</td>
<td>I26_MODE</td>
<td>Configuration byte for %IX1.06 (I26)</td>
</tr>
<tr>
<td>%QB23</td>
<td>I27_MODE</td>
<td>Configuration byte for %IX1.07 (I27)</td>
</tr>
<tr>
<td>%IW3</td>
<td>ANALOG0</td>
<td>Analogue value at I00</td>
</tr>
<tr>
<td>%IW4</td>
<td>ANALOG1</td>
<td>Analogue value at I01</td>
</tr>
<tr>
<td>%IW5</td>
<td>ANALOG2</td>
<td>Analogue value at I02</td>
</tr>
<tr>
<td>%IW6</td>
<td>ANALOG3</td>
<td>Analogue value at I03</td>
</tr>
<tr>
<td>%IW7</td>
<td>ANALOG4</td>
<td>Analogue value at I04</td>
</tr>
<tr>
<td>%IW8</td>
<td>ANALOG5</td>
<td>Analogue value at I05</td>
</tr>
<tr>
<td>%IW9</td>
<td>ANALOG6</td>
<td>Analogue value at I06</td>
</tr>
<tr>
<td>%IW10</td>
<td>ANALOG7</td>
<td>Analogue value at I07</td>
</tr>
</tbody>
</table>

#### Outputs: addresses and variables (0...8 outputs)

<table>
<thead>
<tr>
<th>IEC address</th>
<th>I/O variable</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>%QB40</td>
<td>Q10_MODE</td>
<td>Configuration byte for %QX0.00 (Q10)</td>
</tr>
<tr>
<td>%QB41</td>
<td>Q11_MODE</td>
<td>Configuration byte for %QX0.01 (Q11)</td>
</tr>
<tr>
<td>%QB42</td>
<td>Q12_MODE</td>
<td>Configuration byte for %QX0.02 (Q12)</td>
</tr>
<tr>
<td>%QB43</td>
<td>Q13_MODE</td>
<td>Configuration byte for %QX0.03 (Q13)</td>
</tr>
<tr>
<td>%QB44</td>
<td>Q20_MODE</td>
<td>Configuration byte for %QX0.04 (Q20)</td>
</tr>
<tr>
<td>%QB45</td>
<td>Q21_MODE</td>
<td>Configuration byte for %QX0.05 (Q21)</td>
</tr>
<tr>
<td>%QB46</td>
<td>Q22_MODE</td>
<td>Configuration byte for %QX0.06 (Q22)</td>
</tr>
<tr>
<td>%QB47</td>
<td>Q23_MODE</td>
<td>Configuration byte for %QX0.07 (Q23)</td>
</tr>
</tbody>
</table>
7.3  Error tables

Error flags .......................................................................................................................................... 230
Errors: CAN / CANopen .................................................................................................................. 230

7.3.1  Error flags

→ chapter System flags (→ page 217)

7.3.2  Errors: CAN / CANopen

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

EMCY codes: CANx

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

<table>
<thead>
<tr>
<th>EMCY code object 0x1003</th>
<th>Object 0x1001</th>
<th>Manufacturer specific information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0 [hex]</td>
<td>Byte 1 [hex]</td>
<td>Byte 2 [hex]</td>
</tr>
<tr>
<td>00</td>
<td>80</td>
<td>11</td>
</tr>
<tr>
<td>00</td>
<td>81</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>81</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>81</td>
<td>11</td>
</tr>
<tr>
<td>30</td>
<td>81</td>
<td>11</td>
</tr>
</tbody>
</table>

EMCY codes: I/Os, system

The following EMCY messages are sent automatically, if the FB CANx_MASTER_EMCY_HANDLER (→ page 90) is called cyclically.

<table>
<thead>
<tr>
<th>EMCY code object 0x1003</th>
<th>Object 0x1001</th>
<th>Manufacturer specific information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0 [hex]</td>
<td>Byte 1 [hex]</td>
<td>Byte 2 [hex]</td>
</tr>
<tr>
<td>00</td>
<td>21</td>
<td>03</td>
</tr>
<tr>
<td>00</td>
<td>23</td>
<td>03</td>
</tr>
<tr>
<td>02</td>
<td>23</td>
<td>03</td>
</tr>
<tr>
<td>00</td>
<td>31</td>
<td>05</td>
</tr>
<tr>
<td>00</td>
<td>33</td>
<td>05</td>
</tr>
<tr>
<td>00</td>
<td>42</td>
<td>09</td>
</tr>
<tr>
<td>00</td>
<td>61</td>
<td>11</td>
</tr>
</tbody>
</table>
8 Glossary of Terms

A

Address
This is the "name" of the bus participant. All participants need a unique address so that the signals can be exchanged without problem.

Application software
Software specific to the application, implemented by the machine manufacturer, generally containing logic sequences, limits and expressions that control the appropriate inputs, outputs, calculations and decisions.

Architecture
Specific configuration of hardware and/or software elements in a system.

B

Baud
Baud, abbrev.: Bd = unit for the data transmission speed. Do not confuse baud with "bits per second" (bps, bits/s). Baud indicates the number of changes of state (steps, cycles) per second over a transmission length. But it is not defined how many bits per step are transmitted. The name baud can be traced back to the French inventor J. M. Baudot whose code was used for telex machines.

1 MBd = 1024 x 1024 Bd = 1 048 576 Bd

Boot loader
On delivery ecomatmobile controllers only contain the boot loader. The boot loader is a start program that allows to reload the runtime system and the application program on the device.
The boot loader contains basic routines... • for communication between hardware modules, • for reloading the operating system.
The boot loader is the first software module to be saved on the device.

Bus
Serial data transmission of several participants on the same cable.

C

CAN
CAN = Controller Area Network
CAN is a priority-controlled fieldbus system for large data volumes. There are several higher-level protocols that are based on CAN, e.g. 'CANopen' or 'J1939'.

CAN stack
CAN stack = software component that deals with processing CAN messages.
CiA
CiA = CAN in Automation e.V.
User and manufacturer organisation in Germany / Erlangen. Definition and control body for CAN and CAN-based network protocols.
Homepage → www.can-cia.org

CiA DS 304
DS = Draft Standard
CANopen device profile for safety communication

CiA DS 401
DS = Draft Standard
CANopen device profile for binary and analogue I/O modules

CiA DS 402
DS = Draft Standard
CANopen device profile for drives

CiA DS 403
DS = Draft Standard
CANopen device profile for HMI

CiA DS 404
DS = Draft Standard
CANopen device profile for measurement and control technology

CiA DS 405
DS = Draft Standard
CANopen specification of the interface to programmable controllers (IEC 61131-3)

CiA DS 406
DS = Draft Standard
CANopen device profile for encoders

CiA DS 407
DS = Draft Standard
CANopen application profile for local public transport

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

COB ID
COB = Communication Object
ID = Identifier
ID of a CANopen communication object
Corresponds to the identifier of the CAN message with which the communication project is sent via the CAN bus.
Glossary of Terms

CODESYS
CODESYS® is a registered trademark of 3S – Smart Software Solutions GmbH, Germany. ‘CODESYS for Automation Alliance’ associates companies of the automation industry whose hardware devices are all programmed with the widely used IEC 61131-3 development tool CODESYS®. Homepage → www.codesys.com

CSV file
CSV = Comma Separated Values (also: Character Separated Values)
A CSV file is a text file for storing or exchanging simply structured data.
The file extension is .csv.
Example: Source table with numerical values:

<table>
<thead>
<tr>
<th>value 1.0</th>
<th>value 1.1</th>
<th>value 1.2</th>
<th>value 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>value 2.0</td>
<td>value 2.1</td>
<td>value 2.2</td>
<td>value 2.3</td>
</tr>
<tr>
<td>value 3.0</td>
<td>value 3.1</td>
<td>value 3.2</td>
<td>value 3.3</td>
</tr>
</tbody>
</table>

This results in the following CSV file:
value 1.0;value 1.1;value 1.2;value 1.3
value 2.0;value 2.1;value 2.2;value 2.3
value 3.0;value 3.1;value 3.2;value 3.3

Cycle time
This is the time for a cycle. The PLC program performs one complete run.
Depending on event-controlled branchings in the program this can take longer or shorter.

D

Data type
Depending on the data type, values of different sizes can be stored.

<table>
<thead>
<tr>
<th>Data type</th>
<th>min. value</th>
<th>max. value</th>
<th>size in the memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE</td>
<td>8 bits = 1 byte</td>
</tr>
<tr>
<td>BYTE</td>
<td>0</td>
<td>255</td>
<td>8 bits = 1 byte</td>
</tr>
<tr>
<td>WORD</td>
<td>0</td>
<td>65 535</td>
<td>16 bits = 2 bytes</td>
</tr>
<tr>
<td>DWORD</td>
<td>0</td>
<td>4,294,967 295</td>
<td>32 bits = 4 bytes</td>
</tr>
<tr>
<td>SINT</td>
<td>-128</td>
<td>127</td>
<td>8 bits = 1 byte</td>
</tr>
<tr>
<td>USINT</td>
<td>0</td>
<td>255</td>
<td>8 bits = 1 byte</td>
</tr>
<tr>
<td>INT</td>
<td>-32,768</td>
<td>32,767</td>
<td>16 bits = 2 bytes</td>
</tr>
<tr>
<td>UINT</td>
<td>0</td>
<td>65,535</td>
<td>16 bits = 2 bytes</td>
</tr>
<tr>
<td>DINT</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
<td>32 bits = 4 bytes</td>
</tr>
<tr>
<td>UDINT</td>
<td>0</td>
<td>4,294,967 295</td>
<td>32 bits = 4 bytes</td>
</tr>
<tr>
<td>REAL</td>
<td>-3.402823466 • 10^39</td>
<td>3.402823466 • 10^39</td>
<td>32 bits = 4 bytes</td>
</tr>
<tr>
<td>ULINT</td>
<td>0</td>
<td>18,446,744 073 709 551 615</td>
<td>64 Bit = 8 Bytes</td>
</tr>
<tr>
<td>STRING</td>
<td>number of char. + 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DC
Direct Current
Glossary of Terms

**Diagnosis**
During the diagnosis, the “state of health” of the device is checked. It is to be found out if and what faults are given in the device.
Depending on the device, the inputs and outputs can also be monitored for their correct function.
- wire break,
- short circuit,
- value outside range.
For diagnosis, configuration and log data can be used, created during the "normal" operation of the device.
The correct start of the system components is monitored during the initialisation and start phase.
Errors are recorded in the log file.
For further diagnosis, self-tests can also be carried out.

**Dither**
Dither is a component of the PWM signals to control hydraulic valves. It has shown for electromagnetic drives of hydraulic valves that it is much easier for controlling the valves if the control signal (PWM pulse) is superimposed by a certain frequency of the PWM frequency. This dither frequency must be an integer part of the PWM frequency.

**DLC**
Data Length Code = in CANopen the number of the data bytes in a message.
For SDO: DLC = 8

**DRAM**
DRAM = Dynamic Random Access Memory.
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 = Diagnostic Trouble Code = error code
In the protocol J1939 faults and errors will be managed and reported via assigned numbers – the DTCs.

**E**

**ECU**
(1) Electronic Control Unit = control unit or microcontroller
(2) Engine Control Unit = control device of a engine

**EDS-file**
EDS = Electronic Data Sheet, e.g. for:
• File for the object directory in the CANopen master,
• CANopen device descriptions.
Via EDS devices and programs can exchange their specifications and consider them in a simplified way.
Glossary of Terms

Embedded software
System software, basic program in the device, virtually the runtime system. The firmware establishes the connection between the hardware of the device and the application program. The firmware is provided by the manufacturer of the controller as a part of the system and cannot be changed by the user.

EMC
EMC = Electro Magnetic Compatibility. According to the EC directive (2004/108/EEC) concerning electromagnetic compatibility (in short EMC directive) requirements are made for electrical and electronic apparatus, equipment, systems or components to operate satisfactorily in the existing electromagnetic environment. The devices must not interfere with their environment and must not be adversely influenced by external electromagnetic interference.

EMCY
abbreviation for emergency Message in the CANopen protocol with which errors are signalled.

Ethernet
Ethernet is a widely used, manufacturer-independent technology which enables data transmission in the network at a speed of 10...10 000 million bits per second (Mbps). Ethernet belongs to the family of so-called "optimum data transmission" on a non exclusive transmission medium. The concept was developed in 1972 and specified as IEEE 802.3 in 1985.

EUC
EUC = 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
FiFo
FIFO (First In, First Out) = Operating principle of the stack memory: The data packet that was written into the stack memory first, will also be read first. Each identifier has such a buffer (queue).

Flash memory
Flash ROM (or flash EPROM or flash memory) combines the advantages of semiconductor memory and hard disks. Similar to a hard disk, the data are however written and deleted blockwise in data blocks up to 64, 128, 256, 1024, ... bytes at the same time.

Advantages of flash memories
- The stored data are maintained even if there is no supply voltage.
- Due to the absence of moving parts, flash is noiseless and insensitive to shocks and magnetic fields.
Disadvantages of flash memories

- A storage cell can tolerate a limited number of write and delete processes:
  - Multi-level cells: typ. 10 000 cycles
  - Single level cells: typ. 100 000 cycles
- Given that a write process writes memory blocks of between 16 and 128 Kbytes at the same time, memory cells which require no change are used as well.

FRAM
FRAM, or also FeRAM, means Ferroelectric Random Access Memory. The storage operation and erasing operation is carried out by a polarisation change in a ferroelectric layer. Advantages of FRAM as compared to conventional read-only memories:

- non-volatile,
- compatible with common EEPROMs, but:
- access time approx. 100 ns,
- nearly unlimited access cycles possible.

H

Heartbeat
The participants regularly send short signals. In this way the other participants can verify if a participant has failed.

HMI
HMI = Human Machine Interface

ID
ID = Identifier
Name to differentiate the devices / participants connected to a system or the message packets transmitted between the participants.

IEC 61131
Standard: Basics of programmable logic controllers
- Part 1: General information
- Part 2: Production equipment requirements and tests
- Part 3: Programming languages
- Part 5: Communication
- Part 7: Fuzzy Control Programming

IEC user cycle
IEC user cycle = PLC cycle in the CODESYS application program.

Instructions
Superordinate word for one of the following terms:
- installation instructions, data sheet, user information, operating instructions, device manual, installation information, online help, system manual, programming manual, etc.
Intended use
Use of a product in accordance with the information provided in the instructions for use.

IP address
IP = Internet Protocol.
The IP address is a number which is necessary to clearly identify an internet participant. For the sake of clarity the number is written in 4 decimal values, e.g. 127.215.205.156.

ISO 11898
Standard: Road vehicles – Controller area network
• Part 1: Data link layer and physical signalling
• Part 2: High-speed medium access unit
• Part 3: Low-speed, fault-tolerant, medium dependent interface
• Part 4: Time-triggered communication
• Part 5: High-speed medium access unit with low-power mode

ISO 11992
Standard: Interchange of digital information on electrical connections between towing and towed vehicles
• Part 1: Physical and data-link layers
• Part 2: Application layer for brakes and running gear
• Part 3: Application layer for equipment other than brakes and running gear
• Part 4: Diagnostics

ISO 16845
Standard: Road vehicles – Controller area network (CAN) – Conformance test plan

J

J1939
→ SAE J1939

L

LED
LED = Light Emitting Diode.
Light emitting diode, also called luminescent diode, an electronic element of high coloured luminosity at small volume with negligible power loss.

Link
A link is a cross-reference to another part in the document or to an external document.

LSB
Least Significant Bit/Byte
Glossary of Terms

M

MAC-ID
MAC = Manufacturer’s Address Code
= manufacturer’s serial number.
→ID = Identifier

Every network card has a MAC address, a clearly defined worldwide unique numerical code, more or less a kind of serial number. Such a MAC address is a sequence of 6 hexadecimal numbers, e.g. "00-0C-6E-D0-02-3F".

Master
Handles the complete organisation on the bus. The master decides on the bus access time and polls the →slaves cyclically.

Misuse
The use of a product in a way not intended by the designer.
The manufacturer of the product has to warn against readily predictable misuse in his user information.

MMI
→HMI (→ page 236)

MRAM
MRAM = Magnetoresistive Random Access Memory
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 Significant Bit/Byte

N

NMT
NMT = Network Management = (here: in the CANopen protocol).
The NMT master controls the operating states of the NMT slaves.

Node
This means a participant in the network.

Node Guarding
Node = here: network participant
Configurable cyclic monitoring of each →slave configured accordingly. The →master verifies if the slaves reply in time. The slaves verify if the master regularly sends requests. In this way failed network participants can be quickly identified and reported.
Glossary of Terms

O

**Obj / object**
Term for data / messages which can be exchanged in the CANopen network.

**Object directory**
Contains all CANopen communication parameters of a device as well as device-specific parameters and data.

**OBV**
Contains all CANopen communication parameters of a device as well as device-specific parameters and data.

**OPC**
OPC = OLE for Process Control
Standardised software interface for manufacturer-independent communication in automation technology
OPC client (e.g. device for parameter setting or programming) automatically logs on to OPC server (e.g. automation device) when connected and communicates with it.

**Operational**
Operating state of a CANopen participant. In this mode → SDOs, → NMT commands and → PDOs can be transferred.

P

**PC card**
→ PCMCIA card

**PCMCIA card**
PCMCIA = Personal Computer Memory Card International Association, a standard for expansion cards of mobile computers.
Since the introduction of the cardbus standard in 1995 PCMCIA cards have also been called PC card.

**PDM**
PDM = Process and Dialogue Module.
Device for communication of the operator with the machine / plant.

**PDO**
PDO = Process Data Object.
The time-critical process data is transferred by means of the "process data objects" (PDOs). The PDOs can be freely exchanged between the individual nodes (PDO linking). In addition it is defined whether data exchange is to be event-controlled (asynchronous) or synchronised. Depending on the type of data to be transferred the correct selection of the type of transmission can lead to considerable relief for the → CAN bus.
According to the protocol, these services are unconfirmed data transmission: it is not checked whether the receiver receives the message. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).
Glossary of Terms

PDU
PDU = Protocol Data Unit.
The PDU is an item of the →CAN protocol →SAE J1939. PDU indicates a part of the destination or source address.

PES
Programmable Electronic 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 = Parameter Group Number
PGN = PDU format (PF) + PDU source (PS)
The parameter group number is an item of the →CAN protocol →SAE J1939. PGN collects the address parts PF and PS.

Pictogram
Pictograms are figurative symbols which convey information by a simplified graphic representation. (→ chapter What do the symbols and formats mean? (→ page 7))

PID controller
The PID controller (proportional–integral–derivative controller) consists of the following parts:
• P = proportional part
• I = integral part
• D = differential part (but not for the controller CR04nn, CR253n).

PLC configuration
Part of the CODESYS user interface.
► The programmer tells the programming system which hardware is to be programmed.
> CODESYS loads the corresponding libraries.
> Reading and writing the periphery states (inputs/outputs) is possible.

Pre-Op
Pre-Op = PRE-OPERATIONAL mode.
Operating status of a CANopen participant. After application of the supply voltage each participant automatically passes into this state. In the CANopen network only →SDOs and →NMT commands can be transferred in this mode but no process data.

Process image
Process image is the status of the inputs and outputs the PLC operates with within one →cycle.
• At the beginning of the cycle the PLC reads the conditions of all inputs into the process image. During the cycle the PLC cannot detect changes to the inputs.
• During the cycle the outputs are only changed virtually (in the process image).
• At the end of the cycle the PLC writes the virtual output states to the real outputs.
Glossary of Terms

PWM

PWM = pulse width modulation
The PWM output signal is a pulsed signal between GND and supply voltage. Within a defined period (PWM frequency) the mark-to-space ratio is varied. Depending on the mark-to-space ratio, the connected load determines the corresponding RMS current.

R

ratiometric
Measurements can also be performed ratiometrically. If the output signal of a sensor is proportional to its supply voltage then via ratiometric measurement (= measurement proportional to the supply) the influence of the supply's fluctuation can be reduced, in ideal case it can be eliminated.
→ analogue input

RAW-CAN

RAW-CAN means the pure CAN protocol which works without an additional communication protocol on the CAN bus (on ISO/OSI layer 2). The CAN protocol is international defined according to ISO 11898-1 and guarantees in ISO 16845 the interchangeability of CAN chips in addition.

remenant
Remanent data is protected against data loss in case of power failure. The runtime system for example automatically copies the remanent data to a flash memory as soon as the voltage supply falls below a critical value. If the voltage supply is available again, the runtime system loads the remanent data back to the RAM memory. The data in the RAM memory of a controller, however, is volatile and normally lost in case of power failure.

ro

RO = read only for reading only
Unidirectional data transmission: Data can only be read and not changed.

RTC

RTC = Real Time Clock
Provides (batter-backed) the current date and time. Frequent use for the storage of error message protocols.

Runtime system

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

rw

RW = read/ write
Bidirectional data transmission: Data can be read and also changed.
Glossary of Terms

S

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

Standard: Recommended Practice for a Serial Control and Communications Vehicle Network
- Part 2: Agricultural and Forestry Off-Road Machinery Control and Communication Network
- Part 3: On Board Diagnostics Implementation Guide
- Part 5: Marine Stern Drive and Inboard Spark-Ignition Engine On-Board Diagnostics Implementation Guide
- Part 11: Physical Layer – 250 kBits/s, Shielded Twisted Pair
- Part 13: Off-Board Diagnostic Connector
- Part 15: Reduced Physical Layer, 250 kBits/s, Un-Shielded Twisted Pair (UTP)
- Part 21: Data Link Layer
- Part 31: Network Layer
- Part 71: Vehicle Application Layer
- Part 73: Application Layer – Diagnostics
- Part 81: Network Management Protocol

SD card
An SD memory card (short for Secure Digital Memory Card) is a digital storage medium that operates to the principle of flash storage.

SDO
SDO = Service Data Object.
The SDO is used for access to objects in the CANopen object directory. 'Clients' ask for the requested data from 'servers'. The SDOs always consist of 8 bytes.

Examples:
- Automatic configuration of all slaves via SDOs at the system start,
- Reading error messages from the object directory.
Every SDO is monitored for a response and repeated if the slave does not respond within the monitoring time.

Self-test
Test program that actively tests components or devices. The program is started by the user and takes a certain time. The result is a test protocol (log file) which shows what was tested and if the result is positive or negative.

Slave
Passive participant on the bus, only replies on request of the master. Slaves have a clearly defined and unique address in the bus.

stopped
Operating status of a CANopen participant. In this mode only NMT commands are transferred.

Symbols
Pictograms are figurative symbols which convey information by a simplified graphic representation. (→ chapter What do the symbols and formats mean? (→ page 7))
System variable
Variable to which access can be made via IEC address or symbol name from the PLC.

T

Target
The target contains the hardware description of the target device for CODESYS, e.g.: inputs and outputs, memory, file locations.
Corresponds to an electronic data sheet.

TCP
The Transmission Control Protocol is part of the TCP/IP protocol family. Each TCP/IP data connection has a transmitter and a receiver. This principle is a connection-oriented data transmission. In the TCP/IP protocol family the TCP as the connection-oriented protocol assumes the task of data protection, data flow control and takes measures in the event of data loss. (compare: →UDP)

Template
A template can be filled with content.
Here: A structure of pre-configured software elements as basis for an application program.

U

UDP
UDP (User Datagram Protocol) is a minimal connectionless network protocol which belongs to the transport layer of the internet protocol family. The task of UDP is to ensure that data which is transmitted via the internet is passed to the right application.
At present network variables based on →CAN and UDP are implemented. The values of the variables are automatically exchanged on the basis of broadcast messages. In UDP they are implemented as broadcast messages, in CAN as →PDOs.
According to the protocol, these services are unconfirmed data transmission: it is not checked whether the receiver receives the message. Exchange of network variables corresponds to a "1 to n connection" (1 transmitter to n receivers).

Use, intended
Use of a product in accordance with the information provided in the instructions for use.

W

Watchdog
In general the term watchdog is used for a component of a system which watches the function of other components. If a possible malfunction is detected, this is either signalled or suitable program branchings are activated. The signal or branchings serve as a trigger for other co-operating system components to solve the problem.

wo
WO = write only
Unidirectional data transmission: Data can only be changed and not read.
Index

EMCY codes
- CANx .............................................................. 230
- CANx, system .................................................. 230

Error flags ............................................................................................ 230

Error tables .......................................................................................... 230

Example
- CANx_MASTER_SEND_EMERGENCY ................. 92
- CANx_SLAVE_SEND_STATUS ......................... 97
- CANx_SLAVE_SEND_EMERGENCY .................... 103
- CHECK_DATA .................................................. 209
- Initiation of CANx_RECEIVE_RANGE in 4 cycles ... 87
- NORM (1) .......................................................... 142
- NORM (2) .......................................................... 142
- NORM_HYDRAULIC ........................................... 183

F

Fast inputs .............................................................................................. 61
- FAST_COUNT ...................................................... 144
- Fault ................................................................. 214
- FB, FUN, PRG in CODESYS ................................ 39
- Feedback in case of externally supplied outputs ...... 31
- Feedback on bidirectional inputs/outputs ............. 30
- Feedback on outputs .......................................... 30
- FIFO ................................................................. 235
- Flash memory ................................................... 198, 235
- FLASHREAD .................................................... 202
- FLASH-Speicher ................................................ 15
- FLASHWRITE .................................................... 203
- FRAM ................................................................. 15, 236
- FRAM_READ ..................................................... 204
- FREQUENCY ..................................................... 145
- Function configuration ........................................... 57, 58
- Function configuration in general ....................... 57

Function elements
- adapting analogue values .............................................. 140
- CAN layer 2 ........................................................ 73
- CANopen master .................................................... 89
- CANopen SDOs ................................................... 107
- CANopen slave .................................................... 99
- controllers .......................................................... 184
- counter functions for frequency and period measurement .... 143
- data access and data check ......................................... 207
- hydraulic control ...................................................... 168
- measuring / setting of time ........................................ 195
- Optimising the PLC cycle .......................................... 129
- processing input values .......................................... 135
- processing interrupts ............................................. 129
- PWM functions .................................................... 154
- SAE J1939 .......................................................... 112
- saving, reading and converting data in the memory .... 198
- serial interface ...................................................... 124
- software reset ...................................................... 193

G

GET_IDENTIY .............................................................. 210
- GLR ................................................................. 186

H

Hardware description ................................................................. 12
Hardware structure ................................................................. 13
Heartbeat ........................................................................................... 236
History of the instructions (CR0020, CR0050) .......... 9
HMI ......................................................................................... 236
How is this documentation structured? ....................... 8

I

ID ......................................................................................... 236
- IEC 61131 ........................................................... 236
- IEC user cycle ...................................................... 236
- If runtime system / application is running ............. 19
- If the TEST pin is not active ................................. 20
- ifm function elements ........................................... 67
- ifm function elements for the device CR0050 ....... 73
- ifm libraries for the device CR0050 ............. 67
- ifm worldwide • ifm à l’échelle internationale .... 252
- INC_ENCODER ......................................................... 146
- Information about the device .......... ...................... 12
- INIT state (Reset) .................................................. 46
- Input group 10 (00...07 / ANALOG0...7) ......... 23
- Input group 11 (00...17 / FRQ0...3) ................. 24
- Input group 12 (00...27) ..................................... 25
- INPUT_ANALOG ................................................ 137
- INPUT_CURRENT ................................................ 136
- INPUT_VOLTAGE .............................................. 139

Inputs
- address assignment (16...24 inputs) ....................... 223
- addresses and variables (16...24 inputs) ............... 229
- operating modes (16...24 inputs) ......................... 225
- inputs (technology) .............................................. 21
- inputs 10...13 .................................................... 24
- inputs 14...17 / FRQ0...3 ..................................... 24
- inputs 20...23 .................................................... 25
- inputs 24...27 / CYLO...3 ..................................... 25
- instructions ......................................................... 236
- Intended use ......................................................... 237
- Interface description ............................................ 34
- Interrupt ........................................................... 48
- Interrupt processing ............................................. 129
- IP address .......................................................... 237
- ISO 11898 ........................................................... 237
- ISO 11992 ........................................................... 237
- ISO 16845 ........................................................... 237

J

J1939 ................................................................. 237
- J1939_x .............................................................. 113
- J1939_x_GLOBAL_REQUEST ......................... 114
- J1939_x_RECEIVE ............................................ 116
- J1939_x_RESPONSE ......................................... 118
- J1939_x_SPECIFIC_REQUEST ......................... 120
- J1939_x_TRANSMIT ......................................... 122
- JOYSTICK_0 .................................................... 172
- JOYSTICK_1 .................................................... 175
- JOYSTICK_2 .................................................... 179

Index

Retain variables ................................................................. 66
ro ................................................................. 241
RTC ................................................................. 241
Run ................................................................. 46
RUN state ................................................................. 46
Runtime system ................................................................. 37, 241
rw ................................................................. 241

S

SAE J1939 ................................................................. 112, 242
Safety instructions .......................................................... 10
Safety instructions about Reed relays ..................................... 29, 59
Save boot project ............................................................ 42
Save, read and convert data .................................................. 198
SD card ................................................................. 242
SDO ................................................................. 242
Self-test ................................................................. 242
Serial interface ............................................................... 34, 124
Serial mode ................................................................. 47
SERIAL_MODE ........................................................ 47
SERIAL_PENDING .................................................. 125
SERIAL_RX .......................................................... 126
SERIAL_SETUP ...................................................... 127
SERIAL_TX .......................................................... 128
Set up the programming system ............................................. 53
Set up the programming system manually ................................ 53
Set up the programming system via templates ................................ 56
Set up the runtime system ................................................... 50
Set up the target ............................................................ 54
SET_DEBUG ......................................................... 211
SET_IDENTITY ...................................................... 212
SET_INTERRUPT_I ................................................ 130
SET_INTERRUPT_XMS ........................................ 133
SET_PASSWORD .................................................... 213
Setting control .............................................................. 184
Setting rule .............................................................. 184
Setting rule for a controller ................................................. 184
Slave ................................................................. 242
Slave information ........................................................... 97
SOFTRESET ......................................................... 194
Software controller configuration ........................................ 54
Software description ...................................................... 36
Software modules for the device .......................................... 36
Software reset ............................................................ 193
SRAM ................................................................. 15
Start-up behaviour of the controller ..................................... 11
Status LED ............................................................ 33
STOP state ............................................................. 46
Stopp ................................................................. 46
stopped .............................................................. 242
Storage types for data backup .............................................. 198
Structure Emergency_Message ........................................... 97
Structure node status ....................................................... 97
Symbols ................................................................. 242
System description ......................................................... 12
System flags ............................................................. 217
16...24 inputs and fl...0 outputs (standard side) ......................... 221
CAN ................................................................. 218
error flags (standard side) ................................................. 219
LED (standard side) .................................................... 220
SAE J1939 ............................................................. 218
voltages (standard side) ................................................... 221
SYSTEM STOP state ............................................... 46
System time ............................................................. 196
System variable ............................................................ 243
System variables ........................................................... 57

T

Target ................................................................. 243
TCP ................................................................. 243
Template .............................................................. 243
Test ................................................................. 47
TEST mode ........................................................... 47
TIMER_READ ....................................................... 196
TIMER_READ_US ................................................ 197

U

UDP ................................................................. 243
Update the runtime system ................................................ 52
Use, intended ............................................................ 243
Using ifm downloader .................................................... 42
Using ifm maintenance tool .............................................. 42

V

Variables ................................................................. 66
Verify the installation ....................................................... 52

W

watchdog .............................................................. 49, 243
Watchdog .............................................................. 243
Watchdog behaviour ...................................................... 49
What do the symbols and formats mean? .............................. 7
What previous knowledge is required? ................................ 11
Wiring ................................................................. 29
wo ................................................................. 243

© ifm electronic GmbH
10 Notizen • Notes • Notes