System manual

Single Board Controller (pcb controller)

CS0015

System manual single board controller, October 1999

Guarantee

This manual was written with the utmost care. However, we cannot assume any guarantee for the contents.

Since errors cannot be avoided despite all efforts we appreciate any comment.

We reserve the right to make technical alterations to the product which might result in a change of contents of the manual.

1. General

1.1. Safety instructions

Observe the information of the description. Non-observance of the notes, operation which is not in accordance with use as prescribed below, wrong installation or handling can result in serious harm concerning the safety of persons and plants.

The instructions are for authorised persons according to the EMC and low voltage guidelines. The controllers must be installed and commissioned by a skilled electrician (programmer or service technician).

This description is part of the unit. It contains texts and drawings concerning the correct handling of the controller and must be read before installation or use.

Make sure that the external voltage is generated and applied in accordance with the criteria for safe extra-low voltage (SELV) as this can supply the connected controller, sensors and actuators without additional measures.

The wiring of all signals in connection with the SELV circuit of the unit must also comply with the SELV criteria (safe extra-low voltage, safe electrical separation from other electric circuits).

If the supplied SELV voltage as an external connection to ground (SELV becomes PELV) the responsibility lies with the user and the respective national regulations for installation must be complied with. All statements in these operation instructions refer to the unit the SELV voltage of which is not grounded.

The terminals may only be supplied with the signals indicated in the technical data or on the unit label and only the approved accessories of ifm electronic gmbh may be connected.

The waste heat generated when operating the controller has to escape freely, i.e. during installation sufficient convection of and protection against hot components has to be ensured.

In the case of malfunctions or uncertainties please contact the manufacturer. Tampering with the units can lead to considerable risks for the safety of persons and plant. It is not permitted and leads to the exclusion of any liability and warranty claims.

1.2. Function and features

The single board controller ecomat 100 type CS0015 (in this manual called CS0015) has been designed for industrial use. It has to be installed in a housing or a control cabinet and operated in accordance with the applicable regulations.

The controller contains integrated CMOS components which can be destroyed by electrostatic discharge. Such discharge can already occur when touched by hand. Measures have to be taken to prevent or divert electrostatic discharge during transport, mounting, programming, setting of switches and operation of the controller.

The controller CS0015 is not approved for safety relevant tasks in the sense of protection of persons.

The application software can easily be created by the user with the ecolog 100*plus* software.

All software functions and programming processes described in this documentation refer to the ecolog 100*plus* programming software the knowledge of which is required for this description.

The user also has to observe the software version (especially the operating system of the CS0015 and the function libraries) that is used. Software levels are marked by suffixed letters in alphabetic order in the file names (e.g. CS0015_G.M66 or TDM_C.LIB). When revising existing application projects the user should find out about incompatibilities between the old and the new versions.

The user is responsible for the safe functioning of the application programs which he creates himself. If necessary, he must additionally obtain an approval according to the corresponding national regulations by the relevant testing and supervisory organisations.

1.3. Technical data

Housing:	open pcb
Dimensions:	162 x 126 x 75 mm (WxHxD)
Connections:	voltage supply and inputs/outputs: Phoenix Contact CMBICON RM5,08 CAN/serial interface: Phoenix Contact MINI-CMBICON RM3,81
Operating temperature:	0°C ... +40°C
Protection rating:	IP00 , degree of soiling 2
Supply voltage:	U_B nominal 24 V DC (-15% ... +25%)
Power consumption:	≤ 150 mA, without external load
Processor:	CMOS microprocessor C 167C
Display:	two LED's red and green for status and error display
Unit monitoring:	watchdog (200ms)
Memory:	256 kByte program memory 256 kByte data memory (volatile) with 1 kByte data memory protected against power failure (256 Byte autosave)
Interfaces:	CAN, version 2.0 B (ISO/DIS-11898), 10 ... 1000 kBaud Protocol: CANopen or free communication profile unit class: CANopen master/slave; CAN: FullCAN serial interface RS 232 C, 9.6 kBaud no. of participants: 2 (master/slave)

Binary inputs IN0 ... IN15

Inputs IX0.0 ... IX0.15: common reference point GND

display	LED yellow
input voltage	24 C DC (nominal)
simultaneity factor	100 % at 24 V DC 50 % at 30 V DC
input level	+ 15 ... + 30 V DC
output level	0 ... + 5.5 V DC (or input current < 1.5 mA)
input frequency	500 Hz (IN0 ... IN3) 25 Hz (IN4 ... IN15)

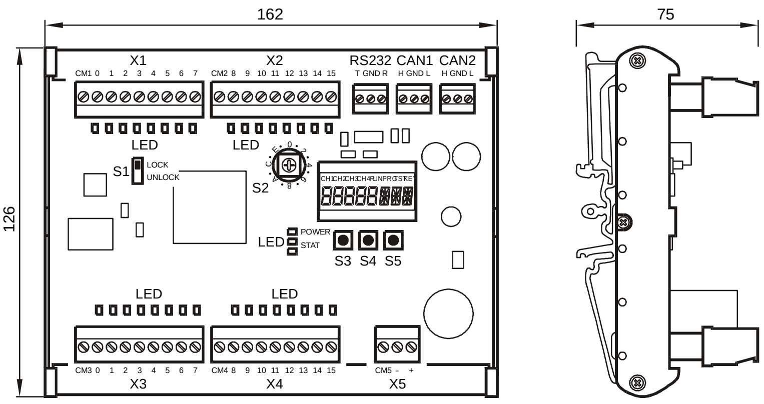
Binary outputs OUT0 ... OUT15

Outputs QX0.0 ... QX0.15: common supply voltage for 8 outputs each (X3/X4) +24 V DC.

display	LED red
switching voltage	12 ... 34 V DC, nominal 24 V DC
switching current	1.1 A
simultaneity factor	100%
short-circuit protection	>6 A (electronic)
output frequency	max. 200 Hz

If precise specifications are observed higher currents (max. 1.9 A) can be switched.

Scale drawing:



1.4. Installation of the controller

The single board controller is supplied in a rail housing for installation on a mounting rail. It can be installed on mounting rails type TS 32 or TS 35.

Make sure that the waste heat generated during the operation of the controller can escape.

1.5. Electrical connection

Before commissioning make sure that the following connections have the correct potentials.

Designation	Pin no.	Potential
Supply voltage	X5 +	+ 24 V DC
Mass	X5 -	GND
Interference suppression GND	X5 CM5	GND
Supply voltage outputs 0 ... 7 (High-Side)	X3 CM3	+ 24 V DC
Supply voltage outputs 8 ... 15 (High-Side)	X4 CM4	+ 24 V DC
Supply voltage outputs Low-Side without monitoring relay	15 (GNDo)	GND
Interference supp. GND inputs	X1 CM1	GND
Interference supp. GND inputs	X2 CM2	GND
Programming interface RS 232	(RxD)	Pin 03, PC 9-pin SUB-D
	(TxD)	Pin 02, PC 9-pin SUB-D
	(CM_5)	Pin 05, PC 9-pin SUB-D
CAN-Interface	(CAN_H)	CAN_H further participant
	(CAN_L)	CAN_L further participant
	(CAN_{GND})	GND further participant

In order to ensure an improved electrical interference oppression of the controller, interference oppression GND X1, X2 (inputs) and X5 (power supply) can be connected with ground (GND)
There is no connection to the GND of the controller voltage supply via these connections.

1.6. Protection of the controller modules

The output channels are electronically protected against overload and short circuit (> 6A) per channel. It is, however, recommended to separately protect the individual circuits in order to protect the whole system (cabling and controller). The total current of 10 A of the individual output groups (max. 8 outputs - e.g. OUT0 ... OUT7) also has to be taken into account.

2. LCD display and operating elements

The CS0015 is equipped with an LCD display, three programmable pushbuttons, a turn switch and a switch for releasing the programming.

They can for instance be used to parameterize the machine set-up. Due to the position of the switches directly on the printed-circuit board handling should be by specialist personnel only.

☞ **These elements are not suitable for the permanent operation of machine functions. In such cases one of the dialogue units made by ifm electronic (e.g. display CR1000 with full graphics capabilities or the CS0014 data display) should be connected via the CAN bus. These displays are suited for the hard requirements of industrial use due to their mechanical construction**

2.1. Switch S1 programming release

With this slide switch the controller can be put in the programming or operating mode.

In the ***LOCK*** position the operating mode is activated and the program memory is protected against the loss of data.

If the controller is to be loaded into a new program or if a communication connection between the controller and the programming system ecolog 100^{plus} has to be established, the switch has to be in position ***UNLOCK***.

☞ **Please note that in normal operation the switch is in position *LOCK* since data loss can also be caused by glitches, i.e. without any connection to the programming system.**

2.2. Rotary coding switch S2

The rotary coding switch can be used for selecting parameters saved in the program (e.g. times and counts) or certain program flows (e.g. setting-up operation). The switch can be turned by means of a little slotted screwdriver. The switch has no mechanical end stop.

The position of the switch 0 ... 15 (0 ... F Hex) can be scanned via the system variable S2 (IEC address %IB2) and can be further processed in the user program.

2.3. Pushbuttons S3 ... S5

The pushbuttons located close to the LCD display can be used e.g. to control the display functions. Each pushbutton is equipped with a normally-open contact.

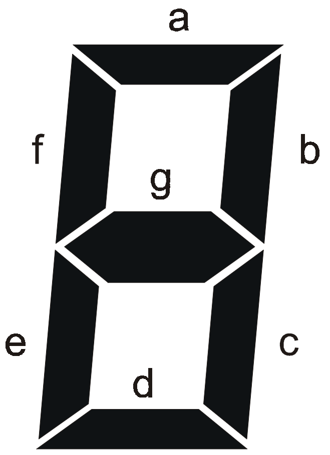
The operation of the pushbuttons can be scanned as bit information (TRUE) via the system variables S3 ... S5 (IEC address %IX1.8, %IX2.0, %IX2.8) and can be further processed in the user program.

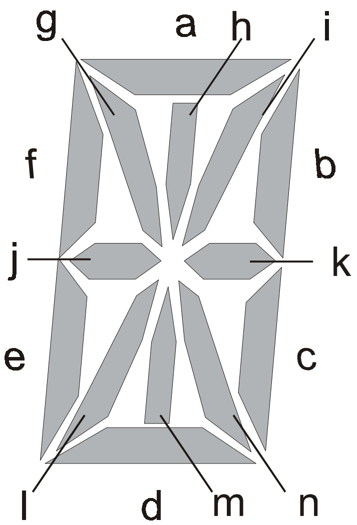
2.4. LCD display

The CS0015 controller is equipped with an LCD display. This display can be used e.g. for displaying operating states. All display elements can be freely programmed via the user program. Via the function calls LCD_SEGMENTS and LCD_TEXT individual display segments or numbers and letters can be shown.

The display contains the following elements:

5 x 7-segment
3 x 14-segment
8 fixed texts





The individual display segments are marked by means of letters. The individual segment is set when the corresponding bit is set in ARRAY.

The function STR can convert a value into a string (chain of characters). The result of this function can be used directly as input value for the function LCD_TEXT.

3. States and operating system

3.1. Operating modes

When the supply voltage is applied, the controller module may be in one of 5 possible operating modes:

Reset

This status is run through after each power-on reset. The operating system is initialised. Different checks are carried out. This status is only temporary and is superseded by the run status.

⇨ The LEDs STAT are lit red and green for a short time

Run

This status is reached:

- from the reset status (Autostart)
- from the stop status by means of the run command
  prerequisite: test mode
- with the CANopen NMT master via the function
  PREOPERATIONAL or OPERATIONAL

⇨ The LED STAT flashes green or red (RUN with error)

Stop

This status is reached:

- from the reset status if no program is loaded
- from the run status by giving the stop command via the interface
  prerequisite: test mode
- with the CANopen NMT master via the function
  PREPARED.

⇨ The LED STAT is constantly lit green

Fatal Error

The controller passes into this status if a non-tolerable error is found. This status can only be left via a reset.

⇨ The LED STAT is constantly lit red.

No operating system

No operating system has been loaded, the controller is in the bootloading status. Before loading the application software a download of the operating system must be carried out.

⇨ The LED flashes green (fast).

3.2. Status LEDs

These operating states are shown in red and green by means of two status LEDs (LED STAT).

LED colour	Flash frequency	Description
green/red	constantly on	Reset checks
green	5 Hz	no operating system loaded
green	0.5 Hz	Run, CANopen: PREOPERATIONAL
	2.0 Hz	Run, CANopen: OPERATIONAL
	constantly on	Stop, CANopen: PRERPARED
red	0.5 Hz	Run w. error (CANopen: PREOPERATIONAL)
	2.0 Hz	Run w. error (CANopen: OPERATIONAL)
	constantly on	Fatal error

The operating states STOP (PREPARED) and RUN (PRE-OPERATIONAL / OPERATIONAL) can be changed by the programming system or the network master.

☞ The user program is processed in the RUN state. The controller only takes part in the CANopen communication (PDO processing, see chapter 5) when it is set to OPERATIONAL. To see the current operating state in the application program the user can evaluate the flag COP_PREOPERATIONAL. The flag is TRUE when the state is PREOPERATIONAL, otherwise it is FALSE.

3.3. Loading the operating system

When the unit is shipped an operating system is in general not loaded in the controller (LED STAT flashes green at 5 Hz). In this operating state only the boot loader is active. It provides the minimum functions for the loading process of the operating system (e.g. the support of the serial and the CAN interface).

In general, the download of the operating system only has to be carried out once. The application program can then be loaded in the controller (even several times). The advantage of this process is that the EPROM does not need to be replaced for an operating system update and that customer-specific operating systems can be realised for certain applications.

The operating system is provided together with this documentation on a separate data carrier.

☞ **The programmer has to ensure that the same software level of the operating system (CS..._x.H86), of the controller configuration (CS..._x.M66) and the unit library (CS..._x.LIB) are used. If not, an error message is generated during the download of the application software. Software states are marked by suffixed letters in alphabetical order in the file name (e.g. CS0015_G.H86). The basic file always has to be the same.**

Operating system download

The operating system and the application software are loaded directly from the programming system. The download can be carried out via the serial and via the CAN interface. The following points have to be observed:

New controller

On delivery, the controller module does not contain an operating system. When the supply voltage is applied it therefore goes into the state "No operating system loaded". Only the bootloader is active.

⇨ For downloading activate the controller configuration screen via the button or via the menu item *Window / PLC Configuration.*

⇨ The requested controller configuration (CS..._x.M66) is called via the menu item *Insert / Firmware.*

⇨ The connection between controller and PC can then be established with *Online / Login*. The interface via which the connection is made depends on the setting in *Extras / HW-Config* (serial or CAN) and the following parameterisation of the PC interface under *Online / Communication Parameters..*.

☞ **A communication connection to the controller is only established when a project is loaded and when this is translated without errors.**

⇨ The download process is started by selecting the menu item *Extras / Load Hex file* and selecting file (CS..._x.H86) in the screen *PLC Configuration*.

The new controller configuration file has to be used for all application programs to be loaded in the controller.

Operating system update

In general, a new operating system software can be loaded in the controller at a later time. This process corresponds in most parts to the one described above.

As opposed to the delivery state of the controller, an operating system is loaded, i.e. the controller is in the STOP or RUN mode.

⇨ The controller configuration of the operating system loaded at the current time is activated so that the programming system can establish the connection between controller and PC.

⇨ The controller configuration screen is activated via the button or via menu item *Window / PLC Configuration.*

⇨ The requested controller configuration (CS..._x.M66) is called via menu item *Insert / Firmware*.

⇨ The connection between controller and PC is established via *Online / Login.* The serial interface for establishing the connection depends on the setting in *Extras / HW Config* (serial or CAN) and the subsequent parameterisation of the PC interface under *Online / Communication Parameters..*.

☞ **It does not matter which project file is loaded (as long as the project can be booted with routine PLC_PRG). The translation processes started with the login can be ignored. The system message:**

Program has changed! Do you want to download the new program?

can be answered with NO.

⇨ Menu item *Extras / Load Hex file* in the screen *PLC Configuration* deletes the current operating system in the controller. The LED of the controller module flashes fast (5 Hz).

⇨ Reset the controller since the online connection between PC and controller does no longer exist after the operating system has been deleted.

⇨ After the reset the new operating system can be loaded. The process is the same as for "New controller".

The new controller configuration file now has to be used for all application programs to be loaded in the controller from now on.

3.4. Operating modes

Independent of the operating states the controller can be operated in different operating modes. The control bits can be set and reset via the application software or in programming operation with the programming software ecolog 100*plus* (window: *Global Variables*).

Programming

This operating mode is activated when the slide switch S1 is in position ***UNLOCK***. In the RUN or STOP states the controller can now accept commands via one of the interfaces. The state of the user program can be scanned via the flag UNLOCK.

Serial Mode

The serial interface is available for a data exchange in the application. Debugging of the application software is only possible via the CAN interface.

This function is switched off as a default (FALSE). The state of the user program or the programming system can be controlled and queried via the flag SERIAL_MODE.

4. Error codes and error classes

In order to ensure maximum operational reliability the operating system carries out internal error checks in the controller during the start-up phase (reset phase) and during the program execution.

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

Error	Error description
CAN_INIT_ERROR	CAN module cannot be initialised
CAN_DATA_ERROR	CAN inconsistent data
CAN_RX_OVERRUN_ERROR	CAN overrun, received data
CAN_TX_OVERRUN_ERROR	CAN overrun, transmission data
CAN_BUS_OFF_ERROR	CAN not on the bus
CAN_ERROR	CAN-Bus collective error bit
ERROR	collective error bit (general)
ERROR_MEMORY	memory error
COP_SYNCFAIL_ERROR	SYNC object was not transferred
COP_GUARDFAIL_ERROR	guarding object is missing (only in the slave)
COP_GUARDFAIL_NODEID	number of missing slave (only in the master)

4.1. Reaction to system error

It is the programmer's responsibility to react to error flags.

The specific error bits should be processed in the user program and then have to be reset. The error bit provides an error description which can be further processed if required.

In the case of severe errors the ERROR bit can be set additionally which also causes the LED STAT to light red and the outputs to be switched off.

☞ Depending on the application it has to be decided if the outputs can be switched on again by resetting the ERROR bit.

⚠ **When using CAN for communication make sure to use the function CAN_ERRORHANDLER. This function ensures that all CAN errors are detected as a group alarm, are counted and CAN is started again.**

Example

A CAN-BUS-OFF error occurs.

The operating system sets the CAN-BUS-OFF-ERROR bit.

The user program detects this state by polling the corresponding bits.

If required the ERROR bit can be set:
As a result the operating display LED flashes red and all outputs are switched off.

The error is removed by restarting CAN via the function call CAN_RESTART. The CAN-BUS-OFF-ERROR bit is deleted automatically.

If required the ERROR bit has to be deleted via the user program. The LED flashes green.

5. CAN in the CS0015

5.1. Technical specifications

Bus type:	FULL-CAN
Physical layer:	ISO/DIS 11898
Baud rate:	10 kBit/s ... 1 MBit/s
Protocol:	CANopen free protocol

2048 data objects in the system (CAN specification 2.0B)

Identifier use

1 ... 2048	identifiers freely available for the data transfer

From these the following identifiers are reserved:

220 ... 221	reserved for the display tdm R 360
223 ... 252	device identifiers of the participants
254	device identifier of an unconfigured module
255	identifier of the download system (e.g. PC)

System configuration

The CS0015 is delivered with the device identifier 254 (ID 32) as participant 0. The download system uses this identifier for the first communication with an unconfigured module.

Only **one** unconfigured module may be connected with the network. After the new participant number 1 ... 30 (corresponds to the node identifier 1 ... 30) was assigned via the programming software, a download or debugging can be performed and another device can be integrated into the system (also see section 5.5).

5.2. Exchange of data via CAN

The exchange of data via CAN is based on the internationally standardized CAN protocol of the data link layer (level 2) of the 7-layer ISO/OSI reference model according to ISO 11898.

Each bus participant can send messages (multi-master capability). The exchange of data operates similar to radio. Data are sent to the bus without sender or address. The data are only qualified by their identifier. It is the job of each participant to receive the transmitted data and to check by means of the identifier whether the data are relevant for this participant.

This operation is automatically carried out by the CAN controller in conjunction with the operating system. To avoid processing each CAN message it is possible to only let a certain part of the bus data reach the CAN controller by indicating a so-called acceptance mask (CAN_ACCEPTANCE). The use of this special function only makes sense if data are not relevant for certain bus participants and time optimization in a plc module is absolutely required for CAN processing. To employ this function hardware knowledge of the CAN controller is necessary. This information is provided in the manufacturer's documentation or can be obtained from the technical support of ifm electronic gmbh.

For the normal exchange of data via CAN the programmer only has to inform the system of the data objects with their identifiers by means of the functions CAN_RECEIVE and CAN_TRANSMIT when designing the software. Via these functions the RAM address of the operating data, the data type and the selected identifier are combined to form a data object. They then participate in the data exchange via the CAN bus. The transmit and receive objects can be defined from all valid IEC data types (e.g. BOOL, WORD, INT, ARRAY).

The CAN message consists of an identifier and max. 8 data bytes. The identifier can be freely selected between 1 and 2048. As already mentioned, it does not represent the sender or receiver module but qualifies the message. To transmit data it is necessary that in the sender module a transmit object is declared and a receiver object in **at least one** other module. Both declarations must be assigned to the same identifier.

Receive data

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

A buffer (queue) is available for each identifier. It is emptied by means of the function CAN_RECEIVE to the FIFO principle (First In, First Out) depending on the application software. In the queue **max. 30** data transmissions are stored temporarily. More data transmissions can only be stored after the buffer has been emptied. The reception of a new CAN message leads to an overflow of the queue, which is indicated to the user by the OVERFLOW bit.

Transmit data

By calling the function CAN_TRANSMIT the application program transfers exactly one CAN message to the CAN controller. As feedback you receive the information whether the message has been successfully transferred to the CAN controller which then performs the actual transfer of the data to the CAN bus.

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

5.3. CAN errors and error handling

The error mechanisms described below are automatically processed by the CAN controller integrated in the plc. This is not influenced by the user. He must/should only react to errors signalled in the application software.

Goal of the CAN error mechanisms:

- Ensuring uniform data objects in the whole CAN network
- Permanent function of the network also in case of a faulty CAN participant
- Distinction between temporary and permanent disturbance of a CAN participant
- Locating and automatic switch-off of a faulty participant in 2 steps (error-passive, bus-off). This gives a temporarily disturbed participant a "break".

To give the interested user an overview of the operating characteristics of the CAN controller in case of an error, a simple description of the error handling will be given below. After the error detection the information is processed automatically and is available to the programmer as CAN error bits in the application software.

Error message

If a bus participant detects an error condition, it immediately sends an error flag, thus causing the abort of the transmission or rejection of the correct messages already received by the other participants. This ensures that all participants are provided with correct and uniform data. Since the error flag is transferred immediately, the sender can immediately start to repeat the disturbed message as opposed to other field bus systems (which wait until a defined acknowledgement time has elapsed). This is one of the most important features of CAN.

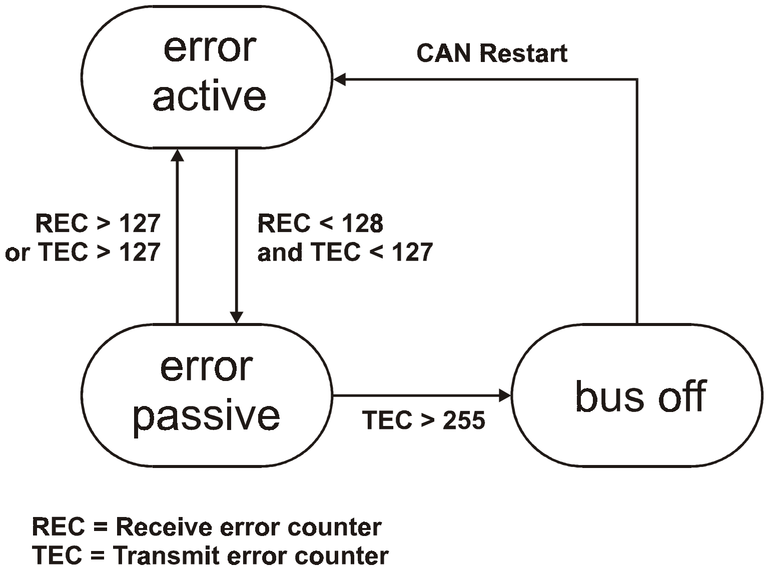
One of the fundamental problems of the serial data transmission is that a permanently disturbed or faulty bus participant can block the whole system. This would be a danger especially for the error handling method of CAN. To exclude this case, a mechanism is required which detects a faulty participant and switches it off from the bus, if necessary.

Error counters

To do so, the CAN controller incorporates a transmission error counter and a reception error counter. They are counted up (incremented) for each erroneous transmission or reception. If a transmission was correct, these counters are counted down again (decremented).

However, in case of an error these error counters are incremented more than they are decremented in case of no error. During a certain time period this can lead to a substantial increase of the counts even if the number of undisturbed messages is greater than the number of disturbed messages. But longer time periods without errors reduce the counts again.

Thus the counts are a measure for the relative frequency of disturbances.
If a participant immediately detects an error it made (it is responsible for the error), this participant is more severely "punished" for the error than the other bus participants. To do so, the counter is incremented by a higher amount. If the count exceeds a certain value, it can be assumed that this participant is faulty. To prevent this participant from further disturbing the bus communication by means of **active error messages** (error-active), it will become **error-passive**.



Participant, error-active

An error-active participant takes part in the bus communication without restriction and is allowed to signal detected errors by sending the active error flag. As already described, this corrupts the transferred message.

Participant, error-passive

An error-passive participant is still capable of communicating without restriction. But it is only allowed to signal an error it detected by means of a passive error flag which does not interfer with the bus operation. An error-passive participant becomes again error-active if its count is again below a defined value.

Participant, bus-off

If the error count continues to increment, the participant is switched off from the bus (bus-off) after a maximum count of the participant has been exceeded.

The bus-off state can only be removed by a reset (CAN_RESTART) of the CAN controller.

This is why the function CAN_ERRORHANDLER should be used which registers all CAN error states and, if necessary, resets the CAN controller. At the same time an error counter is available to the application program. It could for example be used to take further action depending on the count (e.g. error LED).

But a detailed error analysis can only be performed by means of an exact evaluation of the error bits.

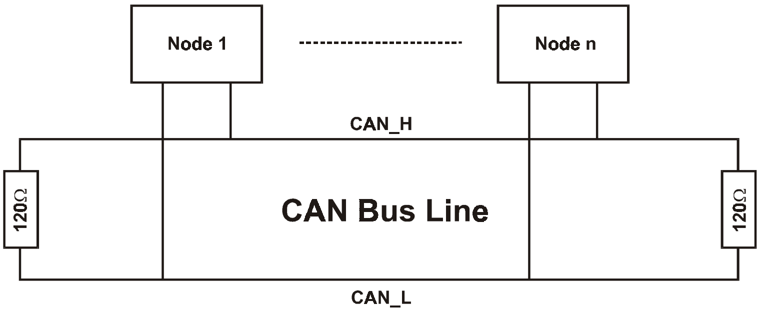
5.4. The physical CAN link

The data transmission and error handling mechanisms described in sections 5.2 and 5.3 are directly implemented in the CAN controller. The physical link of the individual CAN participants is described in layer 1 in ISO 11898.

Network structure

The standard ISO 11898 assumes a line-structured set-up of the CAN network.

In addition, the line must be fitted with a terminating resistor of 120 Ω at its two ends. In principle, ifm electronic's devices fitted with a CAN interface have no terminating resistors.

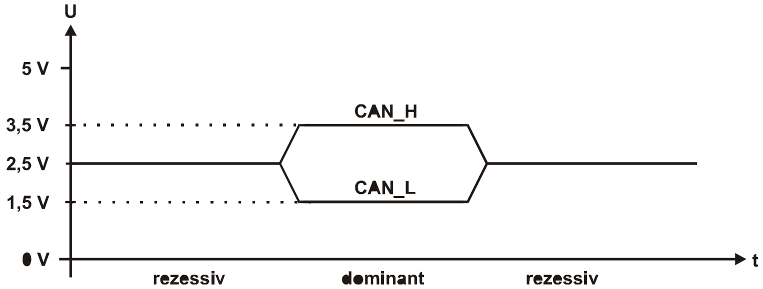


Ideally, no spur should lead to the bus participants (node 1 ... node n) because depending on the total cable length and the transmission time reflections occur in the bus. To avoid this leading to system errors, the spurs to a bus participant (e.g. I/O module) should not exceed a certain length. 2 m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

Bus level

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

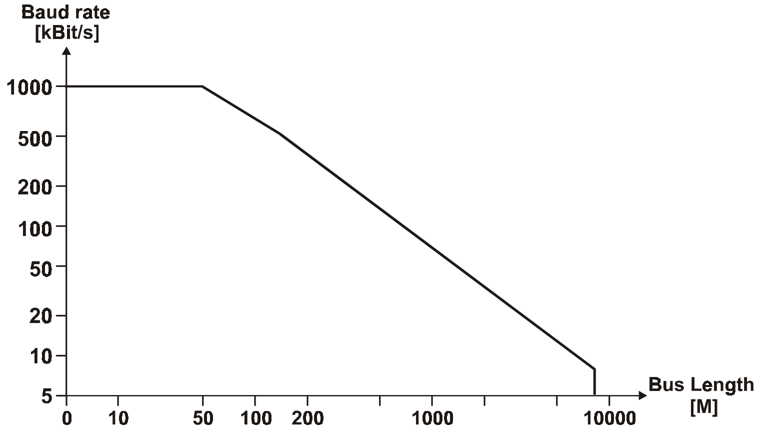
l



This differential transmission with a common return line considerably improves the transmission safety. Interfering voltages which affect the system externally or mass potential offsets influence both signal lines with the same interfering quantities. They are therefore ignored when the difference is formed.

Bus cable length

The bus cable length depends on the characteristics of the bus connection (cable, connector), the cable resistance and the necessary transmission rate (baud rate). As described above, the length of the spurs must also be considered for the network design. For the sake of simplicity, the following dependence between bus length and baud rate can be assumed.



Wire cross-sections

For the design of the CAN network the wire cross-section of the bus cable used must be taken into account. The following table describes the dependence of the wire cross-sections on the number of the bus participants referred to a transmission rate of 1 Mbit/s and a maximum cable length of 40 m (cable resistance r = 70 mΩ/m).

Cable length	32 bus nodes	64 bus nodes	100 bus nodes
100 m	0.25 mm2	0.25 mm2	0.25 mm2
250 m	0.34 mm2	0.50 mm2	0.50 mm2
500 m	0.75 mm2	0.75 mm2	1.00 mm2

Depending on the EMC requirements the bus cables can be laid in parallel or as a twisted pair with or without screen.

5.5. General remarks on the CAN utilization

If in connection with the plc CS0015 CAN or CANopen is used, some points must be taken into account. They concern the physical structure of the CAN network and the correct software handling.

Physical network structure

The following applies to the CAN network structure:

- Ensure that the selected data transmission rate is not higher than needed. A low transmission rate increases the operational reliability.
- The cable length must match the data transmission rate. For CS0015 it is typically 400 m at 125 kBaud.
- Lay the bus cable in a line and avoid spurs. Ensure clean and firm terminal locations to avoid unnecessary contact resistance. If necessary, lay the cables as a twisted pair with or without a screen.
- Fit both ends of the bus cable with a terminating resistor of 120 Ω.
- The higher the number of the participants in the network, the more carefully the network must be laid out (cable version, cable length, etc.).

Software for CAN and CANopen

In principle, the CS0015 can directly take part in the CAN communication (layer 2) by using the functions CAN_TRANSMIT and CAN_RECEIVE. In the CANopen mode the programmer is supplied with the defined services.

The following points must be considered:

- In the direct CAN mode in layer 2 the programmer is responsible for all services. The plc is in this state after a program download or a reset command by the programming system.
- For the direct CAN mode the cyclical integration of the function block CAN_ERRORHANDLER is recommended. Otherwise, the application program must perform a CAN_RESTART in the case of BUS_OFF.
- After a program download or a reset command by the programming system the plc is not yet a CANopen device.
  To change to the CANopen mode the flag CAN_OPEN must be set at the start of the program. The CS0015 then operates as a CANopen slave.
- If a CS0015 slave is stopped via the programming software, a following node start command of the CANopen master is ignored. However, a stop command of the master (NMM_SET_PREPARED) is always executed.

- In case of a missing guarding reply of the CS0015 slave the master continuously sends node resets. This can lead to problems when logging on the programming system via the CAN interface. In this case the master must be switched off.
  If the CS0015 is also to operate as a CANopen master, it must be initialized with the function NMM_SET_NMT_MASTER.
  If the plc is stopped (via PC), it retains the CANopen functions, but the master functions are interrupted (e.g. no SYNC message).
- All participants of the CAN network must be clearly assigned a module ID.

Device IDs in the CS0015

To communicate with the participants in the CAN network each must have a defined device identifier. It is of no importance whether the plc is used as network master, as CANopen slave or for the direct CAN communication. Make also sure that the device identifiers do not overlap with the IDs of the I/O modules. The CS0015 is supplied with the default ID 32 (under CANopen). In the programming software ecolog 100*plus* the node ID 32 is designated as the module ID no. 0.

Module ID ecolog 100*plus*	**Node ID CANopen**	**Device ID debugger**
(default) 0	(unconf.) 32	0xFE
1	1	0xDF
2	2	0xE0
3	3	0xE1
:	:	:
29	29	0xFB
30	30	0cFC

The device ID can be assigned online via ecolog 100*plus*.

5.6. Description of the CAN function blocks

The CAN function blocks for use in the application program will be described below.

☞ To utilize the full functions of CAN it is absolutely required for the programmer to create a precise bus concept before starting to work. The number of the data objects with their identifiers must be defined as well as a reaction to possible CAN errors. Also, the frequency with which data must be transmitted has to be taken into account. So the functions CAN_TRANSMIT and CAN_RECEIVE must be called just as frequently. The programmer must additionally monitor whether his transmission orders have been passed on successfully to CAN_TRANSMIT (bit RESULT) or must make sure that the data received are read from the data buffer of the queue with CAN_RECEIVE and are immediately processed in the program.

To be able to set up a communication link the same transmission rate (baud rate) must first be set for all participants of the CAN network. For the CS0015 this is done with the function CAN_BAUDRATE.

Example program

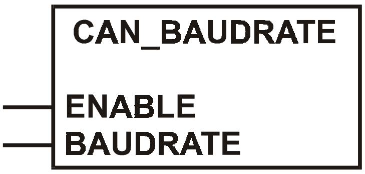
An example program in function block diagram (FBD) is stored on the program diskette ecolog 100^{plus} (CAN3_66.PRO). In this example data objects are exchanged with another CAN participant via the identifiers 1 and 2. To do so, the other participant must have a receive identifier for the transmit identifier (or vice versa).

☞ The function CAN_ACCEPTANCE is not further described here because the application requires thorough hardware knowledge of the CAN controller. Users who need this special feature are requested to contact the technical support.

Function CAN_BAUDRATE

Library **CSxxxx.LIB**

Function symbol



Purpose Sets the transmission rate for the bus participant.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
BAUDRATE	WORD	Value of the baud rate to be set in kBit/s (10, 20, 50, 100, 125, 250, 500, 1000)

Function outputs, none

Description

With the function CAN_BAUDRATE the transmission rate is set for the plc module. To do so, the corresponding value in kBit/s is indicated at the function input BAUDRATE. After the execution of the function this new value is stored in the device and is also available again after a power failure. The factory default for the baud rate of the modules is 125 kBit/s.

☞ The function should be executed only once during the initialization in the first program cycle. After that it is disabled via the input ENABLE.
The baud rate becomes immediately valid after the function call.

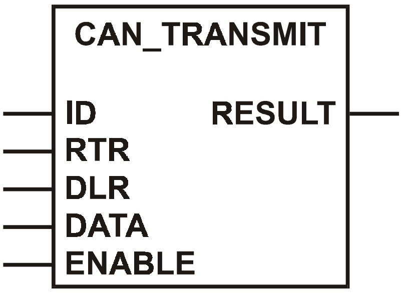
Function

CAN_TRANSMIT

Library

CSxxxx.LIB

Function symbol



Purpose

Passes a CAN data object (message) on to the CAN controller for transmission.

Parameters

Function inputs

Name	Data type	Description
ID	WORD	Contains the number of the data object identifier 0 ... 2048.
RTR	BYTE	Not used, therefore value 0
DLC	BYTE	Number of the bytes to be transmitted from the array DATA (permitted values 0 ... 8).
DATA	ARRAY	The array contains max. 8 data bytes.
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.

Function outputs

Name	Data type	Description
RESULT	BOOL	TRUE: The function has accepted the transmission order.

Description

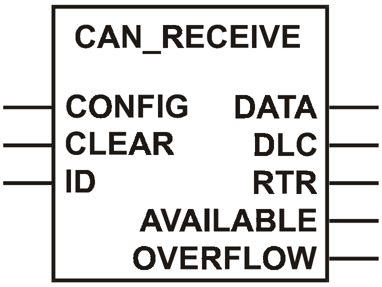
CAN_TRANSMIT is called for each data object in the program cycle, for long program cycles several times. The programmer must ensure by evaluating the bit RESULT that his transmission order has been accepted. It can be said that for 125 kBit/s one transmission order can be executed every 1 ms.

Via the bit input ENABLE the execution of the function can be disabled temporarily. This can for example prevent a bus overload. Also, several data objects can be sent quasi simultaneously if each data object is assigned a flag used to control the execution of the function via the ENABLE input.

Function CAN_RECEIVE

Library **CSxxxx.LIB**

Function symbol



Purpose

Configures a data reception object and reads the reception buffer of the data object.

Parameters

Function inputs

Name	Data type	Description
CONFIG	BOOL	For the configuration of the data object the bit must be set TRUE once. For data transmission to commence the CONFIG bit must be set to FALSE.
CLEAR	BOOL	Deletes the data buffer (queue).
ID	WORD	Contains the number of the data object identifier 0 ... 2048.

Function outputs

Name	Data type	Description
DATA	ARRAY	The array contains max. 8 data bytes.
DLC	BYTE	The number of the transmitted bytes in the array DATA, possible values 0 ... 8.
RTR	BYTE	Is not used
AVAILABLE	BYTE	Number of the messages received
OVERFLOW	BOOL	TRUE: Overflow of the data buffer. Data loss! FALSE: The buffer is not yet full.

Description

CAN_RECEIVE must be called once for each data object during the initialization phase to inform the CAN controller of the identifiers of the data objects.

In the further program cycles CAN_RECEIVE is called to read the corresponding reception buffer, for long program cycles this is done several times. The programmer must make sure by evaluating the byte AVAILABLE that newly received data objects are retrieved from the buffer and are further processed. Each call of the function decrements the byte AVAILABLE by 1. If the value of AVAILABLE equals 0, the buffer contains no data.

By evaluating the bit OVERFLOW an overflowing data buffer can be detected. If the bit OVERFLOW is set, at least 1 data object is **lost**.

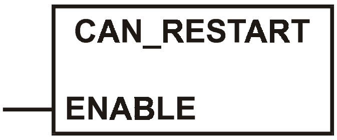
Function

CAN_RESTART

Library

CSxxxx.LIB

Function symbol



Purpose

Restart of the CAN participant after "serious" transmission errors (bus-off state).

Parameters

Function inputs

Name	**Data type**	**Description**
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.

Function outputs, none

Description

CAN enables a distinction between a temporary and a permanent disturbance of a bus participant. As described in section 5.3, three function states are available.

If a participant is **error-active,** this is the normal state.

If a certain number of transmission errors occurs, the participant becomes **error-passive**. If the error frequency is reduced, the participant becomes again **error-active**.

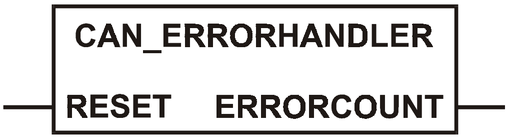
If a participant is already error-passive and transmission errors continue to occur, it is switched off from the bus (**bus-off**) and the error flag CAN_BUS_OFF_ERROR is set. A return to the bus is only possible with the function CAN_RESTART. The error flag is reset after a successful return.

The input ENABLE suppresses the execution of the function.

Function CAN_ERRORHANDLER

Library CSxxxx.LIB

Function symbol



Purpose Minimum error routine to monitor CAN.

Parameters Function inputs

Name	Data type	Description
RESET	BOOL	Deletes the error counter.

Function outputs

Name	Data type	Description
ERROR-COUNT	WORD	Error counter, contains the number of the errors occurred.

Description

CAN_ERRORCOUNT evaluates all possible CAN errors and totals the number of the errors in the counter ERRORCOUNT. In the case of a bus-off error the function tries to return the participant to the bus. To do so, the function CAN_RESTART is integrated.

The programmer's job is to locate the precise error cause by evaluating the error counter and the error bits supplied by the system. Via the function input RESET the counter can then be set to 0 again.

☞ In each application software where the CAN communication is utilized (also for the communication with a CAN display) at least this function should be employed and processed cyclically.

5.7. CANopen in the CS0015

The CAN layers 1 and 2 described at the beginning of chapter 5 control the physical link and the transmission of the data between the bus participants. For a practical CAN application this means that the programmer is responsible for the definition of the data protocol for the special application.

To obtain a uniform protocol layer for networking the different participants which describes the meaning of the transmitted data the CAN Application Layer (CAL) was determined as layer 7. CANopen is based on CAL and defines which data are to be transmitted by which CAL services. The meaning of the data for the corresponding device type (I/O module, drives, encoders, etc.) is also defined. With these definitions the application programmer can access all components with CANopen capability independent of the manufacturer and without much work on the protocol. CANopen participants which belong to the same device family have organized their data in the same way. The characteristics of these device classes are indicated in the "device profiles" (DS-40x).

Despite this definition the basic CAN structure which allows each bus participant to send messages (data) to the network is maintained. Only the network master (NMT master) exists once and is mainly used for running up and monitoring the system.

The mechanisms described below are to give a rough overview of the CANopen functions. If you wish to utilise the full CANopen functions, please contact CAN in Automation Technical Centre.

General information on CANopen

In principle, each CANopen node has an object directory which can be accessed via "Service Data Objects" (SDOs). In addition, there are at least two "Process Data Objects" (PDOs) for transmitting and receiving process data, a "Node Guarding Object" to monitor the network as well as an "Emergency Object" to indicate error states.

The object-oriented identifiers (11 bits) are called "CAN Object IDs" (COB IDs) under CANopen. Via the 4 most significant bits (MSBs) they are divided into 16 groups. The remaining 7 bits are used to distinguish 127 CANopen nodes. This ensures a clear assignment of the individual object types to the nodes. This definition is a default assignment.

It is defined in the "predefined connection set”. Whether this default is adhered to or not depends on the corresponding application. To ensure a high flexibility as regards the selection of CANopen devices from different manufacturers you should carefully consider whether non-adherence is required.

Object	Code (binary)	COB IDs (decimal)	Default function
NMT	0000 0000000	0	Network managem.
SYNC	0001 0000000	128	Synchronization
EMCY	0001 xxxxxxx	129 - 255	Error states
TIME STAMP	0010 0000000	256	Network time
PDO1(tx)	0011 xxxxxxx	385 - 511	Synchronous PDO
PDO1(rx)	0100 xxxxxxx	513 - 639	Synchronous PDO
PDO2(tx)	0101 xxxxxxx	641 - 767	Asynchronous PDO
PDO2(rx)	0110 xxxxxxx	769 - 895	Asynchronous PDO
SDO(tx)	1011 xxxxxxx	1409 - 1535	Master->slave SDO
SDO(rx)	1100 xxxxxxx	1537 - 1663	Slave->master SDO
Nodeguarding	1110 xxxxxxx	1793 - 1919	Node/life guarding

The object directory

All node parameters are stored in the object directory of the corresponding CANopen node. To ensure a clear identification a directory entry is marked by an index (IDX, length 16 bits) and a subindex (SUBIDX, length 8 bits). Depending on the parameter type they are stored in the individual index areas. The meaning of the individual indexes for the communication and standard parameters are defined for the individual device types in the CANopen standard. In addition, an area for manufacturer-specific data is available. In this area the configuration parameters for the I/O modules from ifm electronic gmbh are for example stored.

Index (hex)	Object
0000	Not used
0001 - 009F	Data types
00A0 - 0FFF	Reserved
1000 - 1FFF	Area for the communication profile
2000 - 5FFF	Area for the manufacturer-specific data
6000 - 9FFF	Area for standard device parameters
A000 - FFFF	Area for gen. IEC1131 network variables

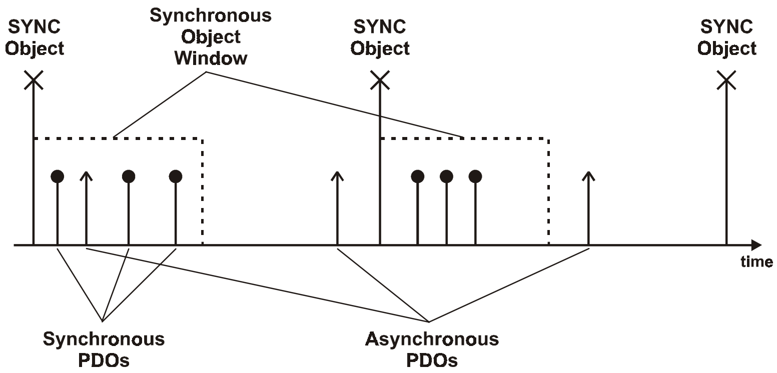
Service Data Object (SDO)

A read and write access to the object directory is achieved via the "Service Data Objects" (SDOs).

The SDOs are used for all data in CANopen which are not time critical. In principle, they are only transmitted from point to point (network master / slave). The SDOs are chiefly used to transmit the configuration data of the CAN participant during the booting phase.

Process Data Object (PDO)

The time critical process data are transferred by means of the "Process Data Objects" (PDOs). The PDOs can be exchanged between the individual nodes in any way (PDO linking). It is also defined whether the data exchange is event-controlled (asynchronous) or synchronized. Depending on the type of data to be transferred the right choice of the transmission type can considerably relieve the amount of data transmitted on the CAN bus. The default setting of the I/O modules from ifm electronic gmbh specifies a synchronous transmission of analog input data and all output data and an event-controlled transmission of digital input data.



Node Guarding Object

To detect communication errors in the network node guarding is used. Each bus node is cyclically accessed by the network master via the defined node guarding COB ID. If no reply is given within the defined guard time, the master signals an error. Via the life time (life time factor x guard time) it can also be defined after how many unsuccessful attempts the error message is to be created.

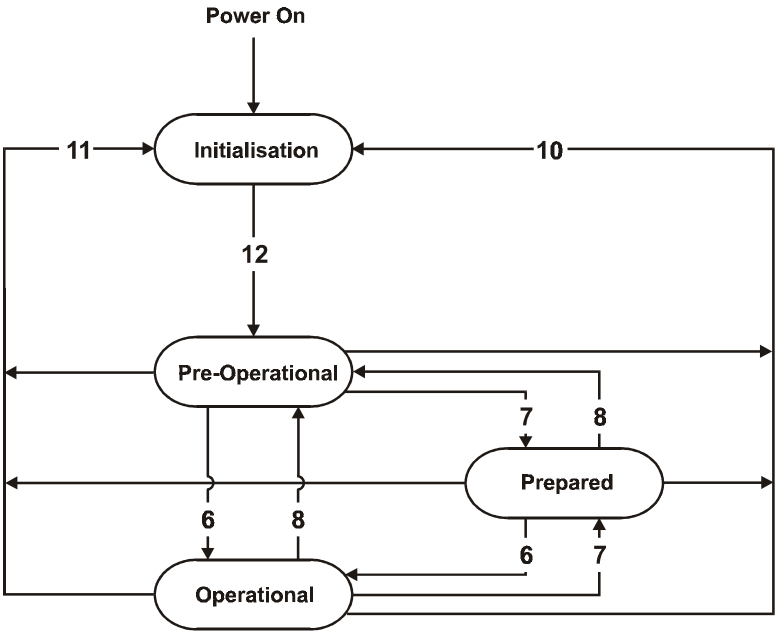
Emergency Object

If an internal error occurs in a bus participant (e.g. wrong configuration parameter, short circuit at the output) an EMCY object is created. This EMCY object is standardized and is sent once when the error occurs and once when the error state has disappeared.

In the object directory of the node these errors are stored. To do so, the "error register", the "manufacturer-specific status register" and the "error history" are available.

Boot-up routine

During the boot-up routine the network master allows the network to run up. In this process the master is informed of the most important communication parameters and, if necessary, guarding is activated. During the boot-up routine the configuration parameters should also be transferred. The node should be in the "pre-operational" state.



State	Description
6	Start remote node indication
7	Stop remote node indication
8	Enter pre-operational state indication
10	Reset node indication
11	Reset communication indication
12	Initialization finished - enter pre-operational automatically

To ensure a successful boot-up routine at least the node number and baud rate of the CAN participant must be set. The baud rate of the master must conform to this. This setting is done via DIP switches in the node or an additional parameter setting software. Since the plc CS0015 also allows a description of the object directory via the SDOs, setting can also be done via the plc.

☛ **To ensure that the CS0015 operates in the CANopen mode the flag CAN_OPEN must be set to TRUE at the program start (during the initialization).**

5.8. The CS0015 as CANopen slave

The CS0015 can also be used as a programmable input/output module under CANopen. It behaves like a CANopen slave. As CANopen slave the CS0015 is classified as a "programmable device" according to CiA DS 405.

To use the CS0015 as CANopen slave the system bit CAN_OPEN must be set.

Object directory

The device parameters can be accessed via the object directory. If they are identified as read/write, they can be changed via SDO_WRITE and by the NMT master or by an external parameter setting system.

The object directory in the CS0015 has three main areas. The CANopen communication parameters are stored as from index 1000 hex.
As from index 2000 hex the manufacturer-specific data of baud rate and node number are stored.
As from index A000 hex starts the area for the general IEC1131 network variables. They are transferred via the PDOs. The identifiers and the transmission types of the PDOs are entered in this area.
For the exact structure of the object directory see point 1.6 in the appendix.

Baud rate and node number

The baud rate and node number are entered in the manufacturer-specific area of the object directory from index 20F0 / 20F1 hex and 20F2 / 20F3 hex. The baud rate or node number can be changed via a SDO by the master, a function call or the programming system. If the change is made via SDO_WRITE, both entries in the object directory must have the same contents. The change of the baud rate only becomes valid after a reset, that of the node ID at once.

Index	Subindex	Name	Default value
20F0	0	Node ID	32
20F1	0	Node ID	32
20F2	0	Baud rate	3
20F3	0	Baud rate	3

☛ **On no account are two participants with the same node number allowed in the network.**

For setting the baud rate the following parameters are allowed:

Number	Baud rate (kBit/s)
0	1000
1	500
2	250
3	125
4	100
5	50
6	20
7	10

Retentive data

Via the manufacturer-specific area of the object directory it is possible to transfer a max. 256-byte data block to the CS0015 slave by means of SDO_WRITE. These data are stored in the flash memory in a non-volatile way and can be further processed in the application program via the retain addresses %MB0 ... %MB255 (%MW0 ... %MW127). Thus this data area is available as freely defined parameter set.

PDOs

In the "predefined connection set” to CiA DS 401 the first two RX and TX PDOs are defined depending on the node number. With these PDOs 16 data bytes each can be sent and received. If more PDOs are required, they must be "manually" defined in the application program by means of the functions PDO_RX_CONFIG and PDO_TX_CONFIG. The identifiers must then be assigned in rising order from 380 hex. If the "predefined connection set" is not used, the COB IDs for PDO 1 and PDO 2 must also start from 380 hex. A total of 2 x 8 PDOs can be set up.

☞

Since the COB IDs for the PDOs are not stored (exception PDO 1 and 2 in the "predefined connection set") they must be re-initialized **once** for all PDOs in the initialization routine after each start of the plc. In principle, the PDO IDs which are not included in the "predefined connection set" have the same default in all devices (RX PDOs from 380 hex, TX PDOs from 388 hex). They must therefore be reconfigured with PDO_TX/RX_CONFIG if several CS0015 slaves are used. Otherwise there would be conflicts with the IDs.

RX-PDO	ID	TX-PDO	ID
RX-PDO 1	pred. c. set	TX-PDO 1	pred. c. set
RX-PDO 2	pred. c. set	TX-PDO 2	pred. c. set
RX-PDO 3	382 hex	TX-PDO 3	38A hex
RX-PDO 4	383 hex	TX-PDO 4	38B hex
RX-PDO 5	384 hex	TX-PDO 5	38C hex
RX-PDO 6	385 hex	TX-PDO 6	38D hex
RX-PDO 7	386 hex	TX-PDO 7	38E hex
RX-PDO 8	387 hex	TX-PDO 8	38F hex

PDO mapping

A conventional PDO mapping is not possible in the CS0015 since this is not necessary for a plc.
Via the application program the data relevant to the CANopen network can be directly written into the PDOs or read from them. Network variables in the area from %MW 2000 for the received data and from %MW 2032 for the data to be transmitted can be immediately processed by the application program (see appendix 1.5). Thus 8 x 4 transmission words (TX-PDOs) and 8 x 4 reception words (RX-PDOs) are available to the user.

Monitoring the PDO reception

The detection whether new data have been transferred is not supported by CANopen. If this function is required, it must be created by the programmer. This can be done as follows:

- Write the signature in the receive object
- PDO contains a toggle bit or consecutive number
- Use the function block CAN_RECEIVE

Transmission types

The transmission types SYNC, i.e. synchronous transmission after a PDO SYNC object or ASYNC, i.e. transmission after a change of the network variables (event due to a change) are supported. The COB ID of the sync object can be configured.

The indication of an inhibit time can delay the sending of ASYNC objects. So considerably fluctuating process values can cause an extremely high bus load in the case of an event-controlled evaluation. If the inhibit time is indicated, the next PDO cannot be sent to the bus before the time has elapsed.
If strategically important values are to be transferred in the ASYNC mode, a single transmission may not be safe enough. Via the function block PDO_TX_REFRESH the important PDO can be repeated from time to time.

As default setting all PDOs are transmitted after a data change (ASYNC mode).

Node guarding

If a CS0015 is accessed by the NMT master once by means of a guarding object, it is fully controlled by the NMT master by means of the cyclical node guarding. If the CAN communication is disturbed, a guarding error message is created in the NMT master. Also, in the CS0015 CANopen slave the flag COP_EVENT_GUARDFAIL is set.

☞ The programmer must evaluate these error messages in his software, specially for critical applications.

ResetNode

If a ResetNode is triggered by the CANopen master, a complete restart of the CS0015 slave would normally have to be carried out (as for a watchdog reset for example). To achieve a higher flexibility, this is controlled by the application program for CANopen.
The flag COP_EVENT_RESETNODE = TRUE tells the user whether a reset was triggered. If it is necessary, the user can then call the function block SOFTRESET. After that the flag must be reset.
In the CS0015 master a long guard time or lifetime must be set to compensate for the long reset phase of the slave.

Emergency objects

If an error occurs in the CS0015 CANopen slave, it is transferred to the master in an emergency object. The COB ID of the EMCY object can be configured.

The emergency objects (consisting of 8 data bytes) are split up in three parts according to CANopen.

1. Emergency code (error code, EMCY), byte 0 and byte 1
2. Error register (error reg.), byte 2
3. Data (additional information), byte 3 ... byte 7

The following errors are transferred:

EMCY code	Error reg.	Description
0x1000	Bit 0	Error (general), output ERROR set, LED red
0x2100	Bit 1	Wire break
0x2300	Bit 1	Short circuit, overload, too high temperature
0x3200	Bit 2	Error undervoltage / overvoltage
0x4000	Bit 3	Error device temperature (> 85°C)
0x8100	Bit 4	Guarding error, no guard object received
0x8200	Bit 4	SYNC error, no Sync object received

EMCY code	Data byte	Description
0x2100	Byte 3	Wire break bit QX0.0 ... QX0.7
	Byte 4	Wire break bit QX0.8 ... QX0.15
	Byte 5	Wire break bit QX0.16 ... QX0.23
0x2300	Byte 3	Short circuit bit QX0.0 ... QX0.7
	Byte 4	Short circuit bit QX0.8 ... QX0.15
	Byte 5	Short circuit bit QX0.16 ... QX0.23
0x8200	Byte 3	Bit 0, CAN error
	Byte 3	Bit 1, SYNC error

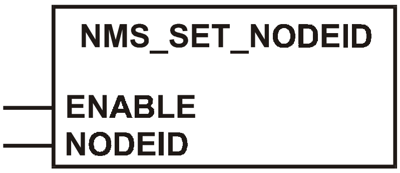
Function

NMS_SET_NODEID

Library

COB.LIB

Function symbol



Purpose

The node ID of the CANopen slave is set.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODEID	BYTE	Number of the identifier (1 ... 30)

Function outputs, none

Description

Via the function NMS_SET_NODEID the node number of the CANopen slave can be set during the initialization. To do so, the function is called once. Via the function input ENABLE the execution of the function is controlled.

As NODEID a number between 1 and 30 can be indicated.

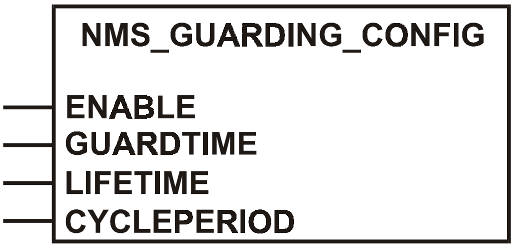
☞

With the execution of the function the node ID becomes immediately valid. This also immediately changes the TX and RX PDOs of the "predefined connection set" which depend on the node ID. The node ID remains valid until it is set again via the function call or the programming system.

Function NMS_GUARDING_CONFIG

Library COB.LIB

Function symbol



Purpose

The guard time for a CS0015 CANopen slave is set.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
GUARDTIME	TIME	Time between two monitoring calls 0 ms = no monitoring 1ms .. 65535ms = monitoring time
LIFETIME	BYTE	Number of the permitted erroneous monitoring calls
CYCLEPERIOD	TIME	Time between two SYNC objects 0 ms = no monitoring 1ms ... 65535ms = monitoring time

Function outputs, none

Description

Via the function NMS_GUARDING_CONFIG the permitted times for the node guarding and the SYNC objects can be set in the CS0015 CANopen slave during the initialization. To do so, the function is called once. The execution of the function is controlled by the function input ENABLE.

If within the specified times the corresponding objects (for node guarding possibly x number of the lifetime cycles) are not received by the CS0015 slave, the corresponding error bits (COP_GUARDFAIL_ERROR and COP_SYNCFAIL_ERROR) are set. They must then be evaluated by the application program.

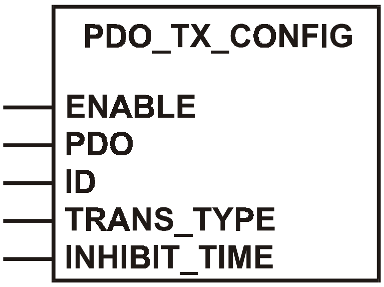
Also, the flag COP_SYNC can be evaluated. It is always TRUE for precisely one cycle.

The specified times must be a little longer than the times set in the master.

Function PDO_TX_CONFIG

Library COB.LIB

Function symbol



Purpose

Initializes a transmit PDO in the CS0015 CANopen slave.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
PDO	BYTE	Number of the TX-PDO (1 ... 8)
ID	WORD	Identifier of the TX-PDO (from 380 hex)
TRANS_TYPE	BYTE	Type of PDO transmission The types SYNC (0,1 ... 240) and ASYNC (255) are supported.
INHIBIT_TIME	TIME	Delay times for the asynchronous transmission mode (0 ... 65535ms)

Function outputs, none

Description

PDO_TX_CONFIG initializes a transmit PDO for the CANopen slave. This function must be executed once during the initialization with ENABLE = TRUE. After that ENABLE must be set to FALSE.

At the function input PDO the corresponding number from 1 ... 8 is indicated.

PDOs which are not to be utilized via the "predefined connection set" must start with an identifier from 380 hex. Otherwise, this can lead to overlapping with other system identifiers. As transmission types (TRANS_TYPE) the modes SYNC (1) and ASYNC (255) are available. If a transmission is not to be carried out for each SYNC object, a value between 1 and 240 (number of the SYNC objects between two accesses) can be entered.

To ensure data transmission in the SYNC mode the SYNC ID of the master and the slave must match. As default value no SYNC ID is entered for the slave.

If necessary, the INHIBIT_TIME (waiting time) must be indicated in the ASYNC mode. Otherwise, considerably fluctuating values can lead to an extremely high bus load.

If strategically important values are to be transmitted in the ASYNC mode, a single transmission may not be safe enough. Via the function block PDO_TX_REFRESH the important PDO can be repeated from time to time.

The default for all TX PDOs is asynchronous transmission.

☞ **If the function PDO_TX_CONFIG is used in a CANopen master, it must be processed before the execution of the function NMM_SET_NMT_MASTER because it triggers an internal CANopen reset. This leads to the loss of the master functions. This is why the initialization must be performed in two steps (start the master booting one cycle later - see example program).**

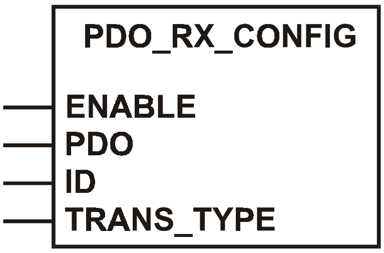
Function

PDO_RX_CONFIG

Library

COB.LIB

Function symbol



Purpose

Initializes a receive PDO in the CS0015 CANopen slave.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
PDO	BYTE	Number of the RX-PDO (1 ... 8)
ID	WORD	Identifier of the RX-PDO (from 380 hex)
TRANS_TYPE	BYTE	Type of the PDO transmission The types SYNC (0, 1 ... 240) and ASYNC (255) are supported.

Function outputs, none

Description

PDO_RX_CONFIG initializes a receive PDO for the CANopen slave. This function must be executed once during the initialization with ENABLE = TRUE. After that ENABLE must be set FALSE.

At the function input PDO the corresponding number from 1 ... 8 is indicated.

PDOs which are not to be utilized via the "predefined connection set" must start with an identifier from 380 hex. Otherwise, this can lead to overlapping with other system identifiers. As transmission types (TRANS_TYPE) the modes SYNC (1) and ASYNC (255) are available. If a transmission is not to be carried out for each SYNC object, a value between 1 and 240 (number of the SYNC objects between two accesses) can be entered.

To ensure data transmission in the SYNC mode, the SYNC ID of the master and the slave must match. As default value no SYNC ID is entered for the slave.

The default for all RX-PDOs is asynchronous transmission.

☞ **If the function PDO_RX_CONFIG is used in a CANopen master, it must be processed before the execution of the function NMM_SET_NMT_MASTER because it triggers an internal CANopen reset. This leads to the loss of the master functions. This is why the initialization must be carried out in two steps (start the master booting one cycle later - see example program).**

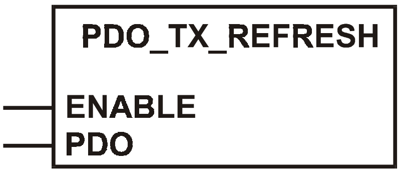
Function

PDO_TX_REFRESH

Library

COB.LIB

Function symbol



Purpose

A sent TX-PDO is transmitted once more.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
PDO	BYTE	Number of the TX-PDO (1 ... 8)

Function outputs, none

Description

Especially if strategically important values are to be transmitted in the ASYNC mode, a single transmission may not be safe enough. Via the function block PDO_TX_REFRESH the important PDO can be repeated from time to time.

The function must not be executed in each cycle because this would lead to CAN overload. The execution can therefore be controlled via the function input ENABLE.

At the function input PDO the corresponding number from 1 ... 8 is indicated.

5.9. The CS0015 as CANopen master

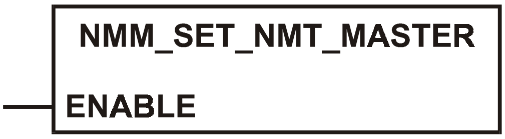
A typical CANopen network has a network master. The functions which will be described below provide all fundamental services to design a master software for the plc CS0015. By using the functions the slave nodes can be integrated into the CAN network, configured and monitored. For a simple introduction to CANopen (especially in applications which "only" require a decentralized extension of the input/output level) the two functions COP_MSTR_BOOTUP and COP_MSTR_MAIN were created in the programming language ST. They use the functions presented below. Details will be given in section 5.10.

☞ **To ensure that the CS0015 operates as CANopen master the flag CAN_OPEN must be set to TRUE at the program start (during the initialization) and the function NMM_SET_NMT_MASTER must be called once.**

Function NMM_SET_NMT_MASTER

Library COB.LIB

Function symbol



Purpose

Initializes the plc module as master.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.

Function outputs, none

Description

NMM_SET_NMT_MASTER initializes the plc as CANopen master. If this function is not called, the plc only operates as a "normal" CANopen participant (slave) in the network.

The network master is responsible for the configuration and monitoring of the network. In a CANopen network only one NMT master, i.e. a master with management function is allowed.

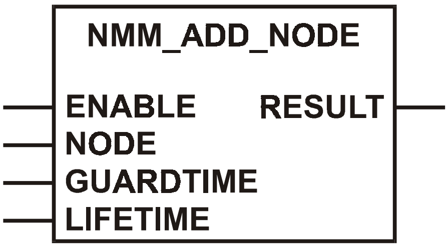
The programmer's job is to evaluate all status information provided by the NMT master to operate a safe network.

If the functions PDO_RX_CONFIG and PDO_TX_CONFIG are used in a CANopen master, they must be processed before the execution of the NMM_SET_NMT_MASTER function because they trigger an internal CANopen reset. This leads to the loss of the master functions. This is why the initialization must be carried out in two steps (start the master boot-up one cycle later - see example program).

Function NMM_ADD_NODE

Library COB.LIB

Function symbol



Purpose

Initializes a guarding object for the specified node.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	Node number from 1 ... 127
GUARDTIME	TIME	Time between two monitoring calls
LIFETIME	BYTE	Number of the permitted erroneous monitoring calls

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 1 = not successful 2 = invalid parameters

Description

NMM_ADD_NODE initializes the CANopen node and a guarding object in the NMT master. The lifetime factor determines how often an erroneous call is allowed. The function must be called once for each node during the initialization. An example is stored in the file NMT_MSTR.PRO.

The node guarding is not executed before having been started via the function NMM_START_GUARDING.

☛

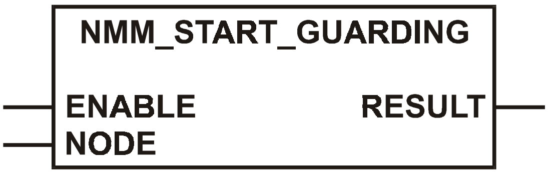
The programmer's job is to locate the exact error cause and to react depending on the application by evaluating the guarding and the other error bits provided by the system.

If a node is not initialized with NMM_ADD_NODE, it cannot be accessed either by other master functions (e.g. SDO_WRITE) independent of the missing node guarding.

Function NMM_START_GUARDING

Library COB.LIB

Function symbol



Purpose Starts the node guarding for one or all initialized nodes.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 2 = invalid parameters

Description

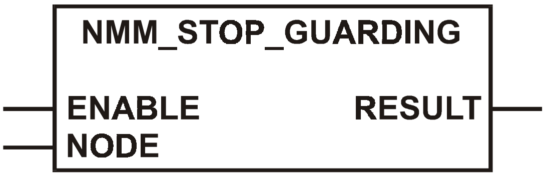
NMM_START_GUARDING starts the node guarding for an individual node or all connected nodes (whole network). To do so, a guarding object must first be initialized for the specified CANopen node with NMM_ADD_NODE.

The programmer's job is to locate the exact error cause and to react depending on the application by evaluating the guarding and the other error bits provided by the system.

Function NMM_STOP_GUARDING

Library COB.LIB

Function symbol



Purpose

Stops the node guarding for one or all initialized nodes.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 2 = invalid parameters

Description

NMM_STOP_GUARDING stops the node guarding for an individual node or all connected nodes (whole network).

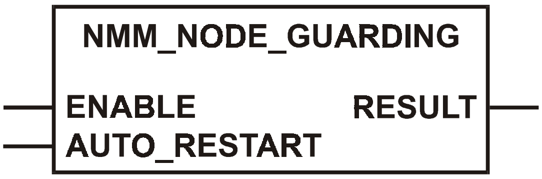
☞ If the node guarding is disabled, the plc no longer detects a missing node.

The programmer's job is to locate the exact error cause and to react depending on the application by evaluating the guarding and the other error bits provided by the system.

Function NMM_NODE_GUARDING

Library COB.LIB

Function symbol



Purpose

The function calls the monitoring of all initialized CANopen nodes.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
AUTO_RE-START	BOOL	TRUE: The monitored node is automatically set to operational after a guarding error. FALSE: The node remains in the pre-operational state.

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful > 0 = missing nodes 0xFF = erroneous call

Description

NMM_NODE_GUARDING organizes the node guarding for all initialized nodes in the whole network. The function must be called cyclically. If several nodes are missing, they are indicated one after the other. The node guarding is only executed if the network monitoring was started with the function NMM_START_GUARDING. The AUTO-RESTART function input allows the automatic start of a node by the master after a guarding error. If AUTO_RESTART is set to TRUE, the node is automatically set again to "operational" after a NODE_RESET. If the input is set to FALSE, the node remains in the "pre-operational" state.

Working with AUTO_RESTART = TRUE is recommended.

☞ If the node guarding is disabled, the plc no longer detects a missing node.

The programmer's job is to locate the exact error cause and to react depending on the application by evaluating the guarding and the other error bits provided by the system.

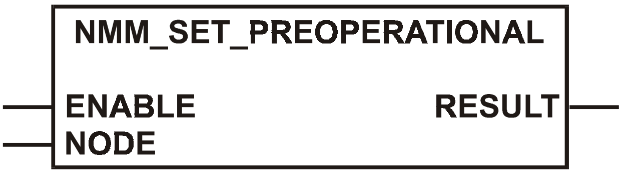
Function

NMM_SET_PREOPERATIONAL

Library

COB.LIB

Function symbol



Purpose

Sets an individual node or the whole network to the "pre-operational" state.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 1 = transmission error 2 = invalid parameters 255 = NMT master not active

Description

NMM_SET_PREOPERATIONAL sets the specified node or the whole network to the "pre-operational" state (also see section 5.7). After the initialization of one (or all) network node(s), it is normally set to the "pre-operational" state. In this state the node (or the nodes) can communicate with the NMT master responsible for the network management only via the SDOs.

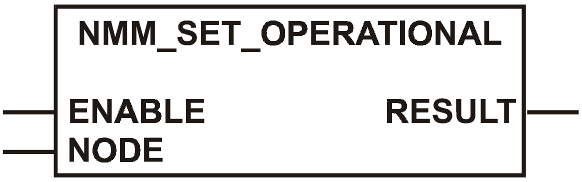
Function

NMM_SET_OPERATIONAL

Library

COB.LIB

Function symbol



Purpose

Sets an individual node or the whole network to the "operational" state.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed FALSE: The function is not processed
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 1 = transmission error 2 = invalid parameters 255 = NMT master not active

Description

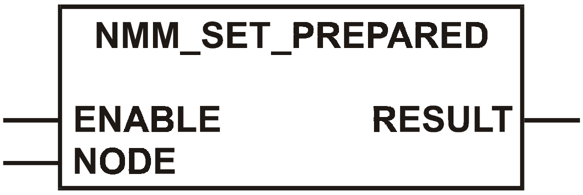
NMM_SET_OPERATIONAL sets the specified node or the whole network to the "operational" state (also see section 5.7). After the initialization of one or all network nodes the "operational" state is reached after the "pre-operational" state. In this state the node (or the nodes) can communicate with the NMT master responsible for the network management and with all other network participants via all communication services (SDOs and PDOs).

☞ The network master too must be set once to the "operational" state to start a correct communication.

Function ## NMM_SET_PREPARED

Library **COB.LIB**

Function symbol



Purpose

Sets an individual node or the whole network to the state "prepared".

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 1 = transmission error 2 = invalid parameters 255 = NMT master not active

Description

NMM_SET_PREPARED sets the specified node or the whole network to the state "prepared" (also see section 5.7). In this state the node (or the nodes) no longer participates in the PDO communication. Also, it is no longer possible to communicate via the SDOs.

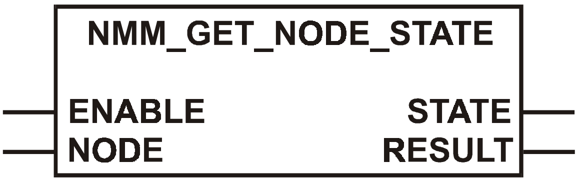
.

This state is often utilized for user-specific needs, for example to temporarily switch off one or all participants from the bus. The state "prepared" can only be removed by the functions NMM_SET_PREOPERATIONAL / NMM_SET_OPERATIONAL.

Function NMM_GET_NODE_STATE

Library COB.LIB

Function symbol



Purpose

Returns the network status of a CANopen node.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
STATE	BYTE	Status to the CANopen specification
RESULT	BYTE	Result: 0 = successful 2 = invalid parameters 255 = NMT master not active

Description

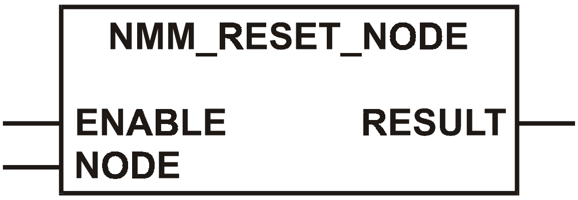
NMM_GET_NODE_STATE returns the current network status (pre-operational, operational, prepared) of one or all nodes. The value results from the CANopen specification.

127	State pre-operational
5	State operational
4	State prepared

Function NMM_RESET_NODE

Library COB.LIB

Function symbol



Purpose

Resets the application and communication parameters for one or all nodes to the default values.

.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 ... 127

Function outputs

Name	Data type	Description
RESULT	BYTE	Result: 0 = successful 1 = transmission error 2 = invalid parameters 255 = NMT master not active

Description

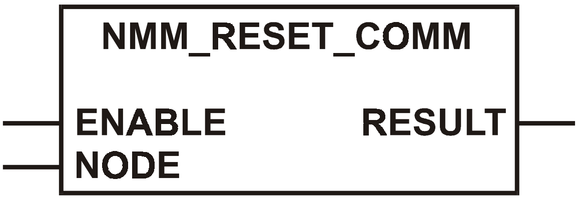
NMM_RESET_NODE performs a reset for the node called (or all nodes in the network). All non-volatile data remain stored in the node. After the reset the node passes in the normal initialization routine.

The exact operating characteristics after a reset are described in the device-specific documents.

Function NMM_RESET_COMM

Library COB.LIB

Function symbol



Purpose

Resets the communication parameters for one or all nodes to the default values.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	All initialized nodes: 0 Initialized node: 1 .. 127

Function outputs

Name	Data type	Description
STATE	BYTE	Status to CANopen specification
RESULT	BYTE	Result: 0 = successful 1 = transmission error 2 = invalid parameters 255 = NMT master not active

Description

NMM_RESET_COMM performs a reset for the node called (or all nodes in the network) for the CAN interface. All non-volatile data remain stored in the node.

The exact operating characteristics after a reset are described in the device-specific documents.

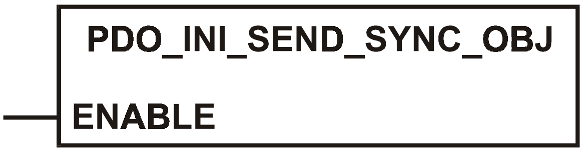
Function

PDO_INI_SEND_SYNC_OBJ

Library

COB.LIB

Function symbol



Purpose

Initializes the PDO SYNC object for the synchronous scanning of I/O data.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.

Function outputs, none

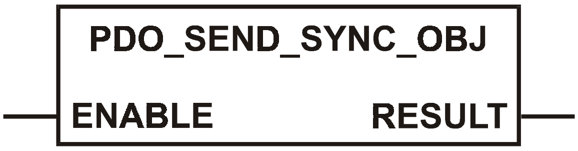
Description

PDO_INI_SEND_SYNC_OBJ initializes the SYNC object for the synchronous scanning of data in the CANopen network (also see section 5.7 Process Data Objects). The function must be called once during the initialization. Via the function PDO_SEND_SYNC_OBJ the SYNC object is then transmitted.

Function **PDO_SEND_SYNC_OBJ**

Library **COB.LIB**

Function symbol



Purpose

Sends the synchronization object.

Parameters

Function inputs

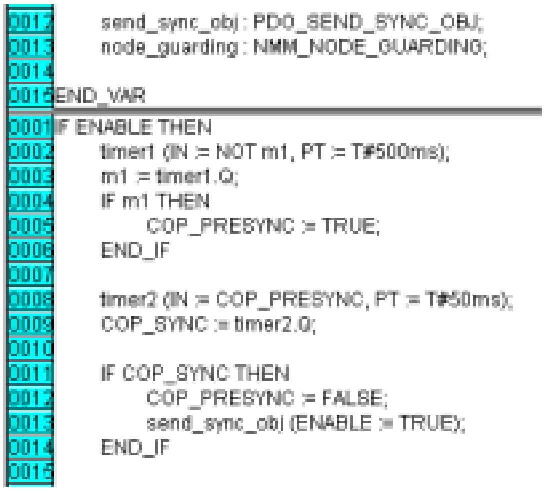
Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.

Function outputs

Name	Data type	Description
RESULT	BOOL	TRUE: The function was processed successfully.

Description

PDO_SEND_SYNC_OBJ sends a SYNC object to the CANopen network. SYNC objects are used for the synchronous scanning of data (also see section 5.7 Process Data Objects). This function must be called cyclically. As shown in the example the time is controlled by means of the two system flags COP_PRESYNC and COB_SYNC.

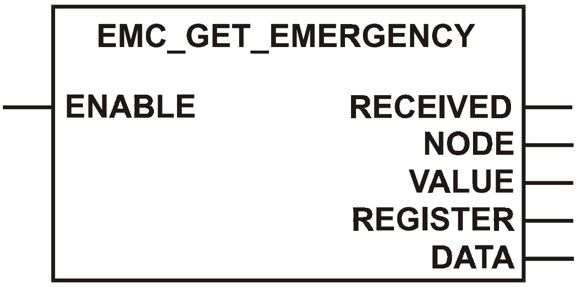


```
    send_sync_obj : PDO_SEND_SYNC_OBJ;
    node_guarding : NMM_NODE_GUARDING;

END_VAR
IF ENABLE THEN
    timer1 (IN := NOT m1, PT := T#500ms);
    m1 := timer1.Q;
    IF m1 THEN
        COP_PRESYNC := TRUE;
    END_IF

    timer2 (IN := COP_PRESYNC, PT := T#50ms);
    COP_SYNC := timer2.Q;

    IF COP_SYNC THEN
        COP_PRESYNC := FALSE;
        send_sync_obj (ENABLE := TRUE);
    END_IF
```

Function EMC_GET_EMERGENCY

Library COB.LIB

Function symbol



Purpose

Read the CANopen emergency object.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.

Function outputs

Name	Data type	Description
RECEIVED	BOOL	TRUE: New error data available
NODE	BYTE	Node number
VALUE	WORD	Error code of the emergency object
REGISTER	BYTE	Error register to index 0x1001
DATA	ARRAY	Manufacturer-specific error information

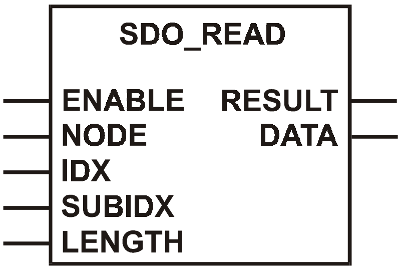
Description

The function EMC_GET_EMERGENCY scans the error data of the connected network nodes. As soon as new data are available the output RECEIVED is set to TRUE for one cycle. The error occurred can then be analysed by scanning the node number (NODE), the error code (VALUE) and the error register (REGISTER). In addition, the DATA output provides the manufacturer-specific node information. For the I/O modules from ifm electronic you are for example informed of a wire break or short circuit at the outputs.

Function SDO_READ

Library COP.LIB

Function symbol



Purpose

Reads the SDO with the specified indexes from the node.

Parameters

Function inputs

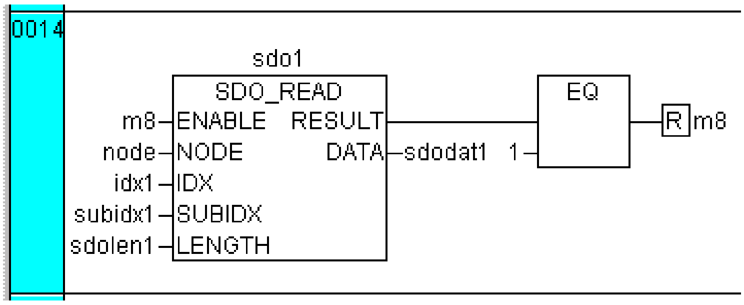
Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	Node number
IDX	WORD	Index in the object directory
SUBIDX	WORD	Subindex referred to the index in the object directory
LENGTH	WORD	Length of the entry in number of bytes

Function outputs

Name	Data type	Description
RESULT	BYTE	0 Function inactive 1 Function execution finished 2 Function active
DATA	ARRAY	Data read (array, length 0 ... 255)

Description

With the function SDO_READ the entries in the object directory can be read. This allows a selective reading of the node parameters. To be able to utilize this function the node must be in the state "pre-operational" or "operational".



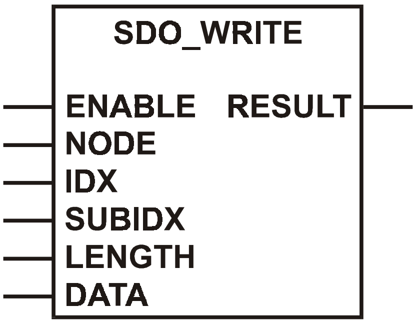
The input ENABLE controls the execution of the function. But since with each call of the function the data array is transferred, the function is a load for the plc cycle even in case of ENABLE=FALSE. This is why SDO_READ should be skipped if the function is not utilized.

The value for LENGTH should conform to the length of the expected data object.

Function SDO_WRITE

Library COP.LIB

Function symbol



Purpose

Writes the SDO with the specified indexes to the node.

Parameters

Function inputs

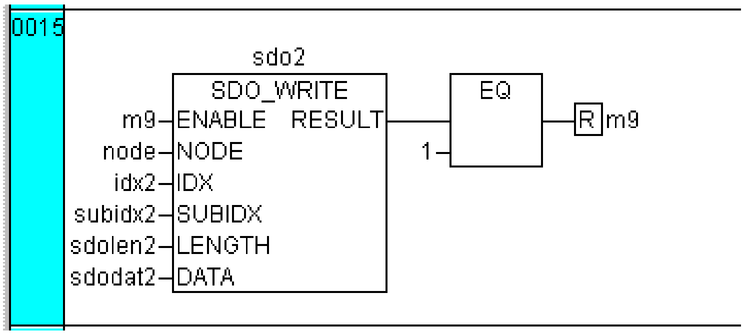
Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NODE	BYTE	Node number
IDX	WORD	Index in the object directory
SUBIDX	WORD	Subindex referred to the index in the object directory
LENGTH	WORD	Length of the entry in "number of bytes"
DATA	ARRAY	Transmission data (array, length 0 ... 255)

Function outputs

Name	Data type	Description
RESULT	BYTE	0 Function inactive 1 Function execution finished 2 Function active

Description

With the function SDO_WRITE the entries can be written in the object directory. This allows a selective setting of the node parameters. To be able to utilize this function the node must be in the state "pre-operational" or "operational".



The input ENABLE controls the execution of the function. But since with each call of the function the data array is transferred, the function is a load for the plc cycle even in case of ENABLE=FALSE. This is why SDO_WRITE should be skipped if the function is not utilized.

☞ **The value for LENGTH must conform to the length of the transmission array. Otherwise, the SDO communication is disturbed.**

5.10. Functions for CANopen I/O modules from ifm electronic

A control solution for an application very often consists of a central plc and one or several decentralised input/output modules. In such applications the central plc is at the same time the network master (see section 5.9). To allow the user a simple design of such applications the functions described below can be used.

If the user wants to use the complete CANopen functionality he will have to refer to the functions described in the chapters before in which case the functions described below become obsolete.

COP_MSTR_BOOTUP and COP_MSTR_MAIN were intentionally written in the language ST. So they can be extended or modified, if this is desired (source code NMT_MSTR.PRO).

The other functions are specially used for the configuration and evaluation of the I/O modules from ifm electronic. With the functions SLAVE_CRxxxx_CONFIG the programmer can directly set the node configuration of the inputs and outputs via the application software or read it from a selected node.

With the functions SLAVE_CRxxxx_WORK the input and output data (digital and analog) are exchanged (read and write) by means of the cyclical call via a defined flag area. These flag addresses enable a direct access to the process data in the application. The flag addresses are organized as follows:

Address	Bit address	Meaning
%MW1010		1st slave, 1st connection, analog I/O data
	%MX1010.0	1st slave, 1st connection, digital I/O data
%MW1011		1st slave, 2nd connection, analog I/O data
	%MX1011.0	1st slave, 2nd connection, digital I/O data
%MW1012		1st slave, 3rd connection, analog I/O data
	%MX1012.0	1st slave, 3rd connection, digital I/O data
:	:	:
%MW1017		1st slave, 8th connection, analog I/O data
	%MX1017.0	1st slave, 8th connection, digital I/O data
%MW1020		2nd slave, 1st connection, analog I/O data
	%MX1020.0	2nd slave, 1st connection, digital I/O data
%MW1021		2nd slave, 2nd connection, analog I/O data
	%MX1021.0	2nd slave, 2nd connection, digital I/O data
:	:	:
%MW1327		32nd slave, 8th connection, analog I/O data
	%MX1327.0	32nd slave, 8th connection, digital I/O data

The last position of the word address describes the connection of the node no. 1 - 8 (0 - 7), the second and third position the node number 1 - 32 (1 - 20 hex). As standard, the predefined address area is rated for 32 I/O modules.

Basic program structure

To utilize the I/O modules in a control application the following program structure can be used. In a standard application it supports the use of up to 31 I/O modules. As the 32nd participant the plc CS0015 configured as network master (NMT master) is connected. A node with the address 0 is not allowed because this address is used for the system-wide controlling of all nodes (also see NMM_NMT functions in section 5.9).

Program step 1

COP_MSTR_BOOTUP
The function initializes the plc as master and the connected nodes. It is only executed in the booting phase. In this function the flag CAN_OPEN is set to TRUE.

Program step 2

COP_MSTR_MAIN
Due to its cyclic call the function creates the SYNC object for the synchronous transmission of the I/O data.

Program step 3

SLAVE_CRxxxx_CONFIG
Slave configuration for each connected I/O node.
After a successful configuration this function is again de-activated.

Program step 4

NMM_SET_OPERATIONAL
A call with the parameter NODE = 0 sets the whole network (also the NMT master) to the operational mode. This function may only be executed once.

Program step 5

SLAVE_CRxxxx_WORK
Due to the cyclical call of the function the I/O data of the slave modules are written to or read from the defined flag area of the CS0015.

Program step 6

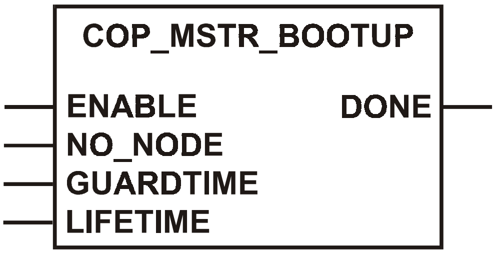
EMC_GET_EMERGENCY
The function provides the emergency (error) data of the connected nodes.

The example program EA_SLAVE.PRO in the directory DEMO\CS0015 shows the software structure for two nodes. It can serve as the basis for extending an application software. If only one slave node is connected to the CS0015, all function calls for the second node must be removed. This program also includes some other master functions (e.g. SDO_READ, SDO_WRITE). These functions enable online communication with the connected slaves via the programming system.

Function COP_MSTR_BOOTUP

Library **NMT_MSTR.LIB**

Function symbol



Purpose

Initializes the plc module as CANopen NMT master and all connected I/O nodes.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NO_NODE	BYTE	Number of the connected nodes without NMT master
GUARDTIME	TIME	Guard time for node monitoring
LIFETIME	BYTE	Lifetime factor for node monitoring

Function outputs

Name	Data type	Description
DONE	BOOL	FALSE: BOOTUP is still active TRUE: BOOTUP is finished

Description

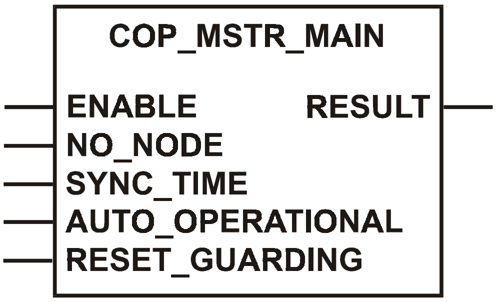
COP_MSTR_BOOTUP sets the CS0015 to the CANopen mode and initializes the plc as NMT master. At the same time the master is informed of the number of the connected nodes (NO_NODE) with the defined guard time (GUARDTIME and LIFETIME factor). After the booting operation (> 2 s) the function output DONE is set to TRUE.

After a successful booting the execution of the function must be disabled via the input ENABLE.

Function COP_MSTR_MAIN

Library NMT_MSTR.LIB

Function symbol



Purpose Cyclically generates the SYNC object and monitors the connected nodes

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
NO_NODE	BYTE	Number of the connected nodes without NMT master
SYNC_TIME	TIME	Time between two SYNC objects for the synchronous scanning of data
AUTO_OPERATIONAL	BOOL	TRUE: The monitored node is automatically set to "operational" after a guarding error. FALSE: The node remains in the "pre-operational" state.
RESET_GUARDING	BOOL	TRUE: Delete the guarding error register

Function outputs

Name	Data type	Description
RESULT	ARRAY	The error register can store max. 8 undetected nodes.

Description COP_MSTR_MAIN must be cyclically executed in the program. This generates the SYNC object for the connected slave modules. The network is monitored at the same time. If a slave fails, the number of the node is entered in the array RESULT. Thus max. 8 errors can be stored. They are entered in the order of their occurrence. The error memory can be deleted again via the function input RESET_GUARDING.

Via the function input AUTO_OPERATIONAL the automatic restart of a node can be selected after a guarding error. If AUTO_OPERATIONAL is set to TRUE, the corresponding node is set again to the mode OPERATIONAL after removal of the disturbance. Thus it directly participates again in the PDO exchange (I/O data are read and written). In the case of AUTO_OPERATIONAL = FALSE the node remains in the state "pre-operational" after the removal of the disturbance. It must then be set selectively to the state "operational" via the NMT master (function NMM_SET_OPERATIONAL).

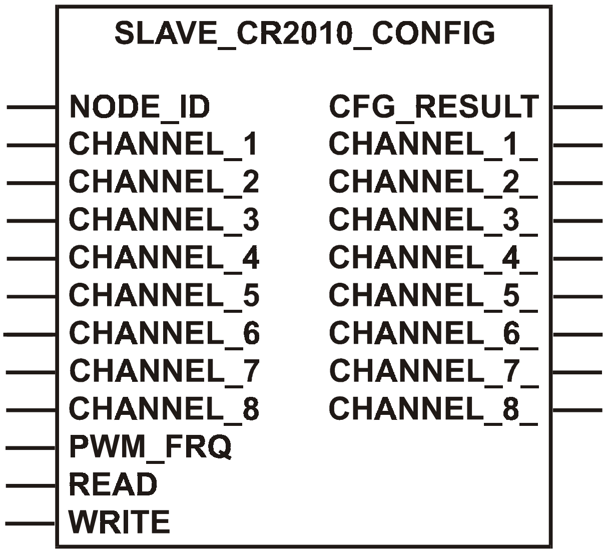
☞ If fast operations are to be processed, the SYNC times must be adapted. They can only be changed in the source code (NMT_MSTR.PRO).

Function

SLAVE_CR2010_CONFIG

Library **CSxxxx.LIB**

Function symbol



Purpose Sets parameters for or reads the configuration of an I/O module.

Parameters Function inputs

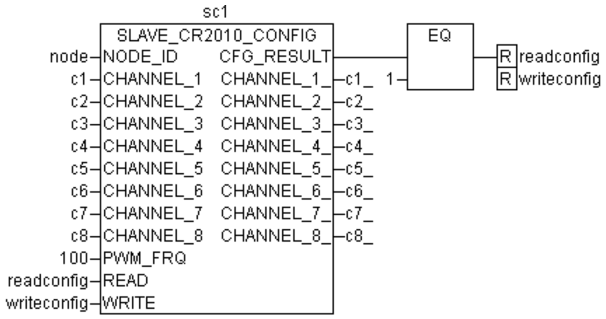
Name	Data type	Description
NODE_ID	BYTE	Node number
CHANNEL_1	BYTE	Configuration parameter for channel 1 0 = OFF, 1 = binary input
CHANNEL_2	BYTE	Configuration parameter for channel 2 0 = OFF, 2 = binary output 3 = analog input, 4 = analog output
CHANNEL_3	BYTE	Configuration parameter for channel 3 0 = OFF, 1 = binary input
CHANNEL_4	BYTE	Configuration parameter for channel 4 0 = OFF, 2 = binary output 3 = analog input, 4 = analog output
CHANNEL_5	BYTE	Configuration parameter for channel 5 0 = OFF, 1 = binary input
CHANNEL_6	BYTE	Configuration parameter for channel 6 0 = OFF, 2 = binary output 3 = analog input, 4 = analog output
CHANNEL_7	BYTE	Configuration parameter for channel 7 0 = OFF, 1 = binary input
CHANNEL_8	BYTE	Configuration parameter for channel 8 0 = OFF, 2 = binary output 3 = analog input, 4 = analog output
PWM_FRQ	BYTE	PWM frequency in Hz (20 ... 200 Hz)
READ	BOOL	Read current module configuration
WRITE	BOOL	Write current module configuration

Function outputs

Name	Data type	Description
CFG_RESULT	BYTE	1 = The configuration has been read or written successfully. 2 = The configuration has not yet been read or written.
CHANNEL_1_	BYTE	Current configuration parameters for channel 1
CHANNEL_2_	BYTE	Current configuration parameters for channel 2
CHANNEL_3_	BYTE	Current configuration parameters for channel 3
CHANNEL_4_	BYTE	Current configuration parameters for channel 4
CHANNEL_5_	BYTE	Current configuration parameters for channel 5
CHANNEL_6_	BYTE	Current configuration parameters for channel 6
CHANNEL_7_	BYTE	Current configuration parameters for channel 7
CHANNEL_8_	BYTE	Current configuration parameters for channel 8

Description

The function SLAVE_CR2010_CONFIG sets or reads the I/O configuration parameters of the 8-channel modules from ifm. The requested configuration is set with the parameters. To check the execution of the function the inputs READ or WRITE should remain set until the function output CFG_RESULT has the value 1. If the data have not yet been updated or if they cannot be read or written, the function output CFG_RESULT has the value 0.



The parameters can be assigned to the function during the run time of the application program. A function block is not necessary for each node.

☞ The configuration data for the I/O module only become active in the state "pre-operational". If the configuration is performed in the state "operational", the new settings do not become valid before passing into the mode "pre-operational" -> "operational".

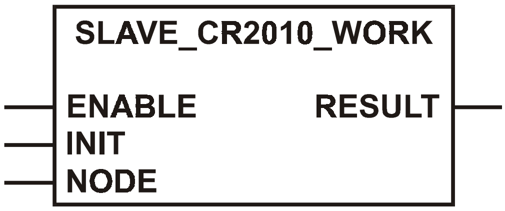
This function has to be called once with READ = TRUE to initialise the controller. Without this call the controller cannot process the I/O data of the module.

The function SLAVE_CR2010_CONFIG corresponds exactly to the function SLAVE_8_CONFIG. For new applications pleases use only the function containing the article number.

Function SLAVE_CR2010_WORK

Library **CSxxxx.LIB**

Function symbol



Purpose Writes or reads the I/O data of a module

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: The function is processed. FALSE: The function is not processed.
INIT	BOOL	TRUE: The function is initialized. FALSE: The data are updated
NODE	BYTE	Node number

Function outputs

Name	Data type	Description
RESULT	BOOL	The function was executed successfully.

Description With the function SLAVE_CR2010_WORK the I/O data for the 8-channel modules from ifm are updated. To do so, this function must be called once for each node in the program cycle. In the first program cycle the input INIT must additionally be set to TRUE once. Thus the operating system of the plc is informed of the configured modules.

The function SLAVE_CR2010_WORK corresponds exactly to the function SLAVE_8_WORK. For new applications please use only the function containing the article number.

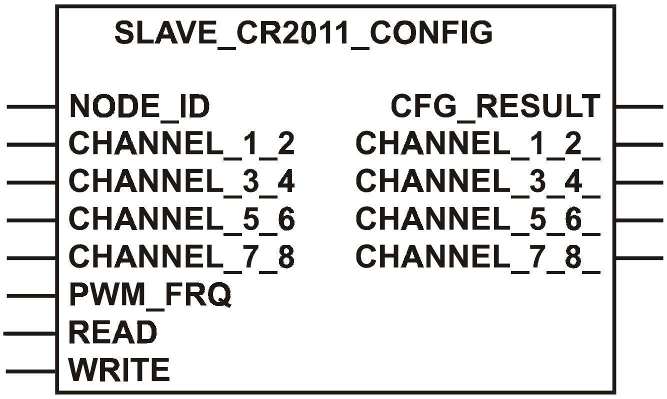
Function

SLAVE_CR2011_CONFIG

Library

CSxxxx.LIB

Function symbol



Purpose

Parameterizes or reads the configuration of an output module.

Parameters

Function inputs

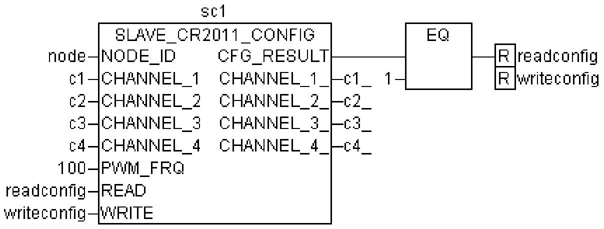
Name	Data type	Description
NODE_ID	BYTE	node number
CHANNEL_1_2	BYTE	Config. parameter for channel 3/4 0 = OFF, 2 = binary output 4 = analog outp. (PWM)., 5 = analog outp. (current regulated)
CHANNEL_3_4	BYTE	Config. parameter for channel 3/4 0 = OFF = binary output 4 = analog outp. (PWM)., 5 = analog outp. (current regulated)
CHANNEL_5_6	BYTE	Config. parameter for channel 3/4 0 = OFF, 2 = binary output 4 = analog outp. (PWM)., 5 = analog outp. (current regulated)
CHANNEL_7_8	BYTE	Config. parameter for channel 3/4 0 = OFF, 2 = binary output 4 = analog outp. (PWM)., 5 = analog outp. (current regulated)
PWM_FRQ	BYTE	PWM frequency in Hz (20 ... 200 Hz)
READ	BOOL	read current module config.
WRITE	BOOL	write current module config.

Function outputs

Name	Data type	Description
CFG_RESULT	BYTE	1 = configuration was successfully read or written 2 = configuration not yet read or written
CHANNEL_1_2_	BYTE	current configuration parameters for channel 1/2
CHANNEL_3_4_	BYTE	current configuration parameters for channel 3/4
CHANNEL_5_6_	BYTE	current configuration parameters for channel 5/6
CHANNEL_7_8_	BYTE	current configuration parameters for channel 7/8

Description

The function SLAVE_CR2011_CONFIG sets or reads the configuration parameters of the ifm 8-channel output modules. The requested configuration is set via the parameters. To check the function flow the inputs READ or WRITE should remain set until value 1 appears at the function output CFG_RESULT. If the data are not the current data or have not yet been updated or if they cannot be written or read value 0 appears at the function output CFG_RESULT.



The parameters can be allocated to the function for the run of the application program.

The configuration data for the I/O module are only accepted in the "pre-operational" state. When the configuration is carried out in the state "operational" the new settings only become valid after switching into the mode "pre-operational" and then into "operational".

The function has to be called once with READ = TRUE to initialize the controller. Without this call the controller cannot process the I/O data.

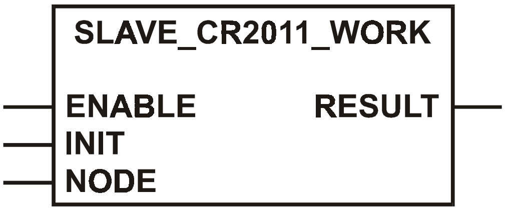
Function

SLAVE_CR2011_WORK

Library

CSxxxx.LIB

Function symbol



Purpose

Writes or reads the I/O data of a module

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is processed FALSE: function is not processed
INIT	BOOL	TRUE: function is initialized FALSE: data are updated
NODE	BYTE	node number

Function outputs

Name	Data type	Description
RESULT	BOOL	The function was executed successfully

Description

The SLAVE_CR2011_WORK updates the data for the ifm 8-channel output module. For this purpose the function has to be called once per program cycle for each node. In addition, in the first program cycle the input INIT has to be set to TRUE once. Thus the configured modules are introduced to the operating system of the controller.

The input/output date are transferred to the defined flag ranges as described above. For modules of type CR2011 it has to be noted that the analog set values (PWM value or current) are entered as signed values. According to the sign the left or right socket of the output pair (1/2, 3/4, 5/6 7/8) is triggered.

The data are organised as follows:

One or several output pairs configured as digital outputs:

Address	Bit address	Description
%MW1010		1st slave, 1st connection
	%MX1010.0	1. Slave, 1st connection, digital output
		1st slave, 1st connection, digital output
%MW1011		1st slave, 2nd connection
	%MX1011.0	1st slave, 2nd connection, digital output
%MW1012		1st slave, 3rd connection
	%MX1012.0	1st slave, 3rd connection, digital output
:	:	:
%MW1017		1st slave, 8th connection
	%MX1017.0	1st slave, 8th connection, digital output

One or several output pairs configured as analog output (PWM):

Address	Chan		Description
%MW1010	1		1st slave, value > 0; channel 1; value < 0 channel 2
%MW1011	2	▨	no entry
%MW1012	3		1st slave, value > 0; channel 3; value < 0 channel 4
%MW1013	4	▨	no entry
%MW1014	5		1st slave, value > 0; channel 5; value < 0 channel 6
%MW1015	6	▨	no entry
%MW1016	7		1st slave, value > 0; channel 7; value < 0 channel 8
%MW1017	8	▨	no entry

One or more output pairs configured as analog output (current regulated):

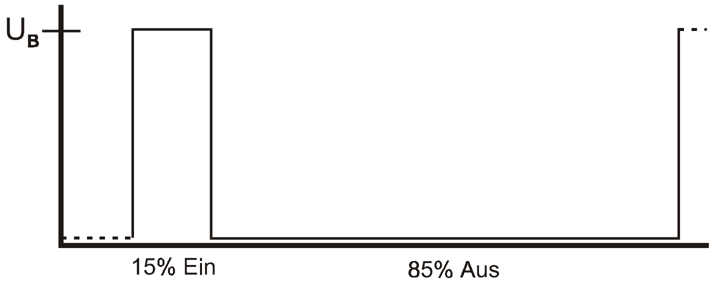
Address	chan	Description
%MW1010	1	1st slave, set value > 0; chan. 1; set value < 0 chan. 2
%MW1011	2	actual value of the channel in mA
%MW1012	3	1st slave, set value > 0; chan. 3; set value < 0 chan. 4
%MW1013	4	actual value of the channel in mA
%MW1014	5	1st slave, set value > 0; chan. 5; set value < 0 chan. 6
%MW1015	6	actual value of the channel in mA
%MW1016	7	1st slave, set value > 0; chan. 7; set value < 0 chan. 8
%MW1017	8	actual value of the channel in mA

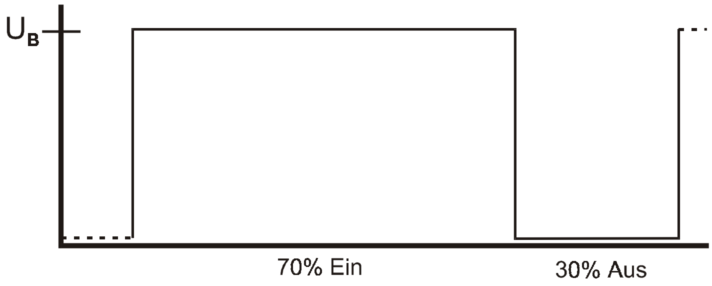
Please note that the selected configuration always applies to two outputs (1/2, 3/4, 5/6 or 7/8).

6. PWM in the CS0015

PWM is an abbreviation for Pulse Width Modulation. In the field of controllers it is mainly used for triggering proportional valves (PWM valves) and drives. Furthermore, an analog output voltage can be generated from the pulse-width modulated output signal by adding (accessory) a PWM output.

The PWM output signal is a pulsed signal between GND and supply voltage. The pulse/break ratio is varied within a defined period (PWM frequency). The current flowing through the connected load depends on the pulse/break ratio.





The PWM function of the controller CS0015 is a hardware function provided by the µcontroller. In order to use the 8 integrated PWM outputs of the controller they need to be initialised in the user program and to be parameterised in accordance with the requested output signal.

The outputs 0 ... 7 (connector X3) can be used as PWM channel in the controller CS0015.

PWM or PWM100

Depending on the application and the requested resolution you can choose between the functions PWM and PWM100 when programming the application. If the application requires a high accuracy and resolution, the more technical PWM function is used rather than the PWM100.

If the implementation time is to be kept low and if there are no high demands with regard to accuracy the function PWM100 can be used. In this function the PWM frequency can be entered in Hz and the pulse/break ratio in 1% steps.

PWM frequency

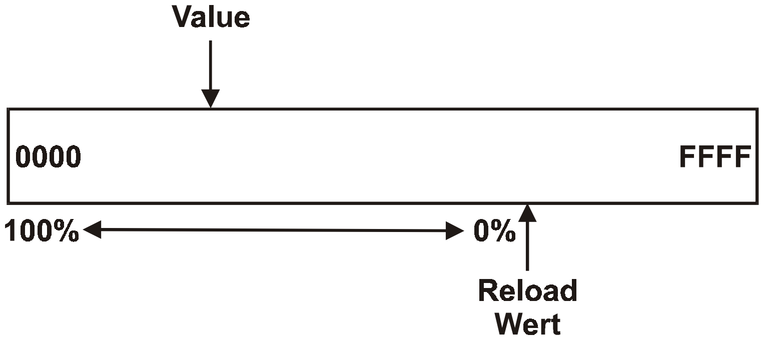
Each type of valve requires a certain PWM frequency. The frequency for the PWM function is transferred via the reload value (function PWM) or directly as a figure in Hz (function PWM100). The controller CS0015 has 2 x 4 PWM outputs which differ in their operation, but not in their effects.

The PWM frequency is achieved by means of an internal counter based on the CPU cycle. The counter is started when the PWM function is initialised. Depending on the PWM output group (0..3 or 4...7) the counter either counts down from FFFF Hex or up from 0000 Hex. When a comparative value (VALUE) has been reached the output is set. The output is reset when the counter overflows (count changes from 0000 Hex to FFFF Hex or from FFFF Hex to 0000 Hex) and the process is restarted.

If the internal counter does not run between 0000 Hex and FFFF Hex another preset value (RELOAD value) for the internal counter can be transferred. This increases the PWM frequency. The comparative value has to be within the newly defined range.

PWM channels 0 ... 3

These four PWM channels offer the highest flexibility during parameterisation. An independent PWM frequency (RELOAD value) can be set for each channel, and it is possible to select between the functions PWM or PWM100.



Calculating the RELOAD value

The reload value of the internal PWM counter is calculated as follows based on the input DIV64:

DIV64 = 0: f_{PWM} = 10.00 MHz / Reload
DIV64 = 1: f_{PWM} = 156.25 kHz / Reload

The input DIV64 has to be set to 0 or 1 depending on whether a high or a low PWM frequency is required. For PWM frequencies < 152 Hz DIV64 has to be set to 1 so that the reload value does not get bigger than FFFF Hex.

Example

The PWM frequency should be 200 Hz.

$$\text{Reload value} \Rightarrow \frac{10\text{ MHz}}{200\text{ Hz}} = 50000 \Rightarrow \text{C350 Hex}$$

The permissible range for the PWM value is

from 0000 Hex to C350 Hex

The comparative value at which the output switches has to be between 0000 Hex and C350 Hex.

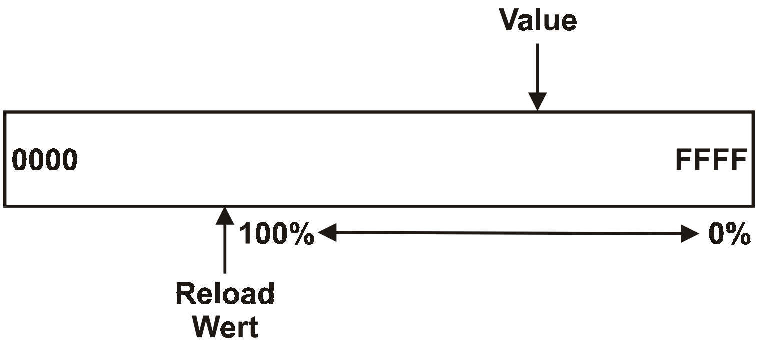
This results in the following pulse / break ratios:

minimum pulse / break ratio (0 % on):	C350 Hex
maximum pulse / break ratio (100 % on):	0000 Hex

50000 intermediate values (PWM values) are possible between maximum and minimum.

PWM channels 4 ... 7

These four PWM channels can only be set to a common PWM frequency. The functions PWM and PWM100 must not be mixed during programming.



Calculation of the RELOAD value

The reload value of the internal PWM counter is calculated as follows based on the input DIV64:

DIV64 = 0: f_{PWM} = 2.50 MHz / (10000 Hex - Reload)
DIV64 = 1: f_{PWM} = 156.25 kHz / (10000 Hex - Reload)

The input DIV64 has to be set to 0 or 1 depending on whether a high or a low PWM frequency is required. For PWM frequencies < 39 Hz DIV64 has to be set to 1 so that the reload value does not get smaller than 0000 Hex.

Example

The PWM frequency should be 200 Hz.

$$\frac{2.5\text{ MHz}}{200\text{ Hz}} = 12500 \Rightarrow \text{30D4 Hex}$$

Reload value ⇒ 10000 Hex - 30D4 Hex = CF2C Hex

The permissible range for the PWM value is

from CF2C Hex to FFFF Hex

The comparative value at which the output switches has to be between CF2C Hex and FFFF Hex.

The PWM frequency is the same for all PWM outputs. Functions PWM and PWM100 must not be mixed.

This results in the following pulse / break ratios:

minimum pulse / break ratio (0 % on):	FFFF Hex
maximum pulse / break ratio (100 % on):	CF2C Hex

12500 intermediate values (PWM values) are possible between maximum and minimum.

PWM dither

In some hydraulic valve types the PWM frequency has to be superimposed by a so-called dither frequency (jitter frequency). If these valves were triggered with a constant PWM value over a longer period of time they might stick due to the high system temperatures. To prevent this, the PWM value is increased or decreased by a defined value (DITHER_VALUE) based on the dither frequency. As a result, the constant PWM value is superimposed by a beat with the dither frequency and the amplitude DITHER_VALUE. The dither frequency is stated as a ratio (divider, DITHER_DIVIDER) of the PWM frequency

☛ The function PWM_DITHER has to be initialised once for each PWM output with DELTA selected individually. The dither frequency can be different for channels 0...3, it has to be the same for channels 4...7.

Ramp function

If you do not want a hard change from one PWM value to the next (e.g. from 15% on to 70% on, see graphics in this chapter) you can e.g. use the PT1 function (see chapter 9) to achieve a delayed increase. This can also be accomplished by counting up step by step to the new set value in the application software. This way hydraulic systems can e.g. be soft started.

Program example

A program example for the PWM functions of the ecomat CS0015 is saved on the program diskette ecolog 100*plus*.

☛ **The PWM function of the controller CS0015 is a hardware function provided by the processor. When the PWM function is initialized at one of the outputs (0 ... 3 or 4 ... 7) the function remains set until a hardware reset (switching on and off of the supply voltage) has been carried out at the controller.**

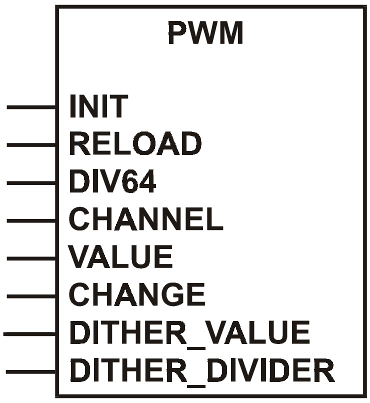
When the PWM function is activated at one of the outputs 0 ... 3 or 4 ... 7, all four outputs in the group are switched in the PWM mode which means that these outputs are no longer available as digital outputs. With the PWM function the switching characteristics of the digital output can be emulated with PWM-maximal (100%) and PWM-minimal (0%) if necessary.

The maximum PWM frequency depends on the output transistors used. For the CS0015 it is 200 Hz. Higher frequencies lead to high imprecisions.

Function PWM

Library CSxxxx.LIB

Function symbol



Purpose

The function is used to initialise and parameterise the PWM outputs.

Parameters

Function inputs

Name	Data type	Description
INIT	BOOL	TRUE: PWM output is initialised FALSE: PWM is allocated new values
RELOAD	WORD	value to define the PWM frequency
DIV64	BOOL	CPU cycle / 64
CHANNEL	BYTE	current PWM channel/output
VALUE	WORD	current PWM value
CHANGE	BOOL	TRUE: new PWM value is taken over FALSE: changed PWM value has no influence on the output
DITHER_ VALUE	WORD	amplitude of the dither value
DITHER_ DIVIDER	WORD	dither frequency = PWM frequency/DIVIDER

Function outputs, none

Description

Function PWM has more than just a technical background. Due to their construction the PWM values can be read out at a very high resolution, so that this function is suitable for high-accuracy proportional control.

Function PWM is called up once for each channel during initialisation of the user program. Input INIT has to be set to TRUE. During initialisation the parameter RELOAD is transferred.

☞ **The RELOAD value has to be the same for channels 4...7. The functions PWM and PWM100 must not be mixed.**

The PWM frequency (and thus the RELOAD value) is internally limited to 10 kHz.

The input DIV64 has to be set to 0 or 1 depending on whether a high or low PWM frequency is required.

While the program is running INIT must be set to FALSE. The function is called and the new PWM value is transferred. The value is accepted when input CHANGE = TRUE.

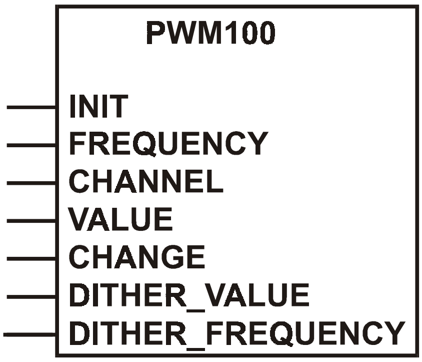
PWM_DITHER is called up once for each channel during the initialisation of the user program. Input INIT has to be set to TRUE. During initialisation the DIVIDER (divisor) for establishing the dither frequency and DELTA are transferred.

☞ **The DIVIDER value has to be the same for channels 4...7. DELTA can be set individually for each channel**.

Function PWM100

Library CSxxxx.LIB

Function symbol



Purpose

The function is used to initialised and parameterise the PWM outputs.

Parameters

Function inputs

Name	Data type	Description
INIT	BOOL	TRUE: PWM100 is initialised FALSE: PWM100 is allocated new values
FREQUENCY	WORD	PWM frequency in Hz
CHANNEL	BYTE	current PWM channel/output
VALUE	BYTE	current PWM value
CHANGE	BOOL	TRUE: new PWM value is accepted FALSE: changed PWM value has no influence on the output
DITHER_ VALUE	BYTE	amplitude of the dither value in percent
DITHER_ FREQUENCY	WORD	dither frequency in Hz

Function outputs, none

Description

Function PWM100 allows a simple use of the PWM functions. The PWM frequency can be stated directly in Hz and the pulse /break ratio in 1% steps. This function is not suited for setting up high-accuracy proportional controls.

Function PWM100 is called up once for each channel during the initialisation of the user program. The input INIT has to be set to TRUE. During initialisation the parameter FREQUENCY is transferred.

☞ **The FREQUENCY value has to be the same for channels 4...7. The functions PWM and PWM100 must not be mixed.**

The PWM frequency is internally limited to 10 kHz.

While the program is running INIT must be set to FALSE. The function is called up and the new PWM value is transferred. The value is accepted if input CHANGE = TRUE.

PWM_FREQUENCY is called up once for each channel during the initialisation of the user program. Input INIT has to be set to TRUE. During initialisation the frequency and the value (DITHER_VALUE) are transferred.

The PWM_FREQUENCY value has to be the same for channels 4...7. DITHER_VALUE can be set individually for each channel.

7. Fast inputs

The controller ecomat R 360 has a total of 4 fast inputs which can process input frequencies up to 500 Hz. Apart from measuring the frequency at the inputs FRQ0...FRQ3 the inputs ENC0 and ENC1 can also be used to evaluate encoders (counter functions).

Input	Connection	Description
FRQ 0/ENC 0	X1, In 1	frequency measurement / channel A, encoder 1
FRQ 1/ENC 0	X1, In 2	frequency measurement / channel B, encoder 1
FRQ 2/ENC 1	X1, In 0	frequency measurement / channel A, encoder 2
FRQ 3/ENC 1	X1, In 3	frequency measurement / channel B, encoder 2

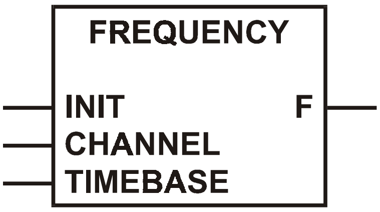
The functions FREQUENCY, CYCLE and INC_ENCODER are available for simple evaluation.

☞ **If the fast inputs of the controller CS0015 are used as "normal" digital inputs the increased sensitivity to noise has to be taken into account (e.g. contact bouncing in the case of mechanical contacts). The standard digital input has an input frequency of 50 Hz. If required, the input signal has to be debounced by means of the software.**

Function FREQUENCY

Library CSxxxx.LIB

Function symbol



Purpose

The function FREQUENCY measures the signal frequency at the defined channel.

Parameters

Function inputs

Name	Data type	Description
INIT	BOOL	TRUE: FREQUENCY is initialised FALSE: frequency measurement active
CHANNEL	BYTE	input number (0 ... 3)
TIMEBASE	TIME	time basis

Function output

Name	Data type	Description
F	WORD	frequency in Hz

Description

FREQUENCY measures the frequency of the signal at the selected channel (CHANNEL). The positive edge is evaluated. Depending on the time base (TIMEBASE) frequency measurements can be carried out over a wide range. High frequencies require a short time base, lower frequencies require a longer time base. The frequency is stated in Hz. For low frequencies the function FREQUENCY causes imprecisions. To avoid those, the function CYCLE can be used.

☞ Only the inputs FRQ0...FRQ3 can be used for function FREQUENCY.

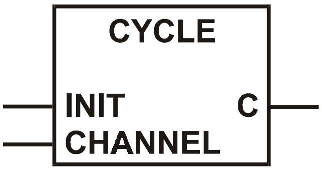
Function

CYCLE

Library

CSxxxx.LIB

Function symbol



Purpose

The function CYCLE measures the period (cycle time) in ms at the defined channel.

Parameters

Function inputs

Name	Data type	Description
INIT	BOOL	TRUE: CYCLE is initialised FALSE: in the cyclical program run
CHANNEL	BYTE	number of the input (0 ... 3)

Function output

Name	Data type	Description
C	WORD	cycle time in ms

Description

CYCLE measures the cycle time of the signal at the selected channel (CHANNEL). The rising edge is evaluated. In the case of low frequencies the function FREQUENCY causes imprecisions. To avoid these, the function CYCLE can be used. The cycle time is displayed in ms.

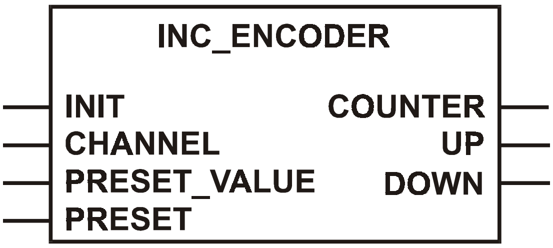
The maximum measurement range is 65535 ms (= 15 Hz).

☞ For the function CYCLE only the inputs FRQ 0 ... FRQ 3 can be used.

Function **INC_ENCODER**

Library **CSxxxx.LIB**

Function symbol



Purpose Up/down counter to evaluate encoders

Parameters Function inputs

Name	Data type	Description
INIT	BOOL	TRUE: INC_ENCODER is initialised FALSE: counter is active
CHANNEL	BYTE	number of the input pair (0,1)
PRESET_ VALUE	WORD	preset counter value
PRESET	BOOL	TRUE: preset value is accepted FALSE: counter is active

Function outputs

Name	Data type	Description
COUNTER	WORD	actual value
UP	BOOL	TRUE: counter counts up
DOWN	BOOL	TRUE: counter counts down

Description

The function INC_ENCODER is an up/down counter. Two frequency inputs form an input pair which is evaluated via the function. A total of 2 incremental encoders can be connected.

The counter can be set to a preset value via PRESET_VALUE. The value is accepted when PRESET is set to TRUE. PRESET then has to be reset to FALSE so that the counter becomes active. Output COUNTER shows the current count.

The outputs UP and DOWN show the current count direction of the counter. The outputs are TRUE when the counter has counted in the direction in question in the previous program cycle. When the counter stops the directional output is reset in the following program cycle.

8. Functions for the integrated display

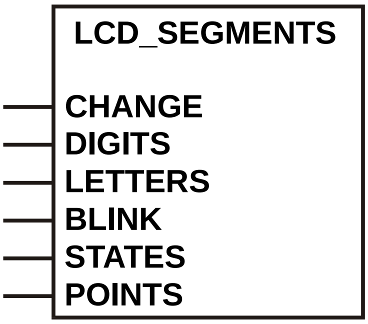
Function

LCD_SEGMENTS

Library

CSxxxx.LIB

Function symbol



Purpose

The function LCD_SEGMENTS triggers the segments in the display.

Parameters

Function inputs

Name	Data type	Description
CHANGE	BOOL	TRUE: The new values are transferred to the display.
DIGITS	ARRAY	According to the set bit, the segment in question is triggered. One array element is available for each figure (array length 0...5 byte).
LETTERS	ARRAY	According to the set bit the segment in question is triggered. One array element is available for each letter (array length 0...2 word)
BLINK	BYTE	Exactly one bit is assigned to each figure or letter. For each letter one array element is available (array length 0...2 word).
STATES	BYTE	Triggering of the fixed display symbols. When the bit is set, the element is triggered.
POINTS	BYTE	Triggering of fixed decimal points. One point is assigned to each figure. When the bit is set, the point is triggered.

Function outputs, none

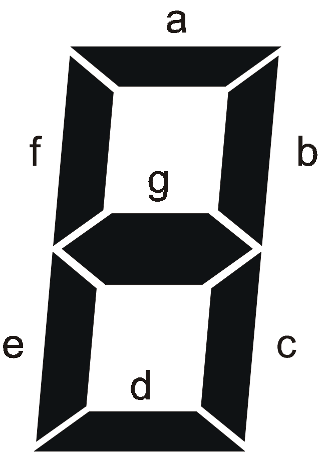
Description

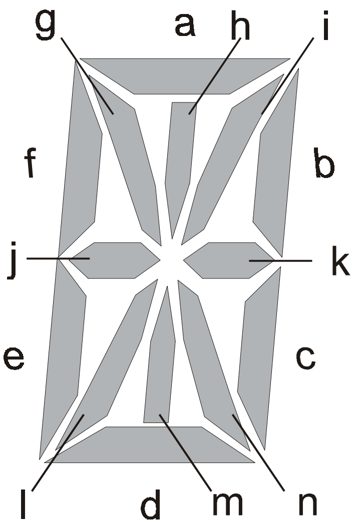
With the function LCD_SEGMENTS the individual segments on the integrated display can be triggered. When the bit is set and the input CHANGE is set to to TRUE the segments are set. When input CHANGE is set to FALSE, the changes are not effective.

Allocation of the segments, attributes, symbols and decimal points to the individual bits (bit 0 corresponds to the LSB):

Bit	Digit 1-5	Letter 1-3	Flash	Symbol	Point
0	e	d	Digit 1	CH1	Digit 1
1	f	e	Digit 2	CH2	Digit 2
2	d	f	Digit 3	CH3	Digit 3
3	g	l	Digit 4	CH4	Digit 4
4	a	j	Digit 5	RUN	Digit 5
5	░	g	Letter 1	PRG	░
6	c	m	Letter 2	TST	░
7	b	h	Letter 3	KEY	░
8	▨	a	▨	▨	▨
9	▨	n	▨	▨	▨
10	▨	k	▨	▨	▨
11	▨	i	▨	▨	▨
12	▨	░	▨	▨	▨
13	▨	c	▨	▨	▨
14	▨	b	▨	▨	▨
15	▨	░	▨	▨	▨

░ bit not used
▨ invalid range

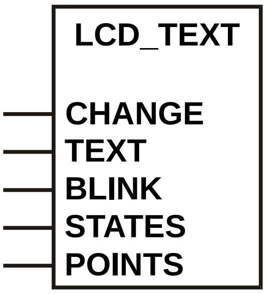




Function LCD_TEXT

Library CSxxxx.LIB

Function symbol



Purpose

With the function LCD_TEXT a string of maximum 8 digits can be directly transferred to the display.

Parameters

Function inputs

Name	Data type	Description
CHANGE	BOOL	TRUE: The new values are transferred to the display..
TEXT	STRING	A text with the maximum length of 8 digits can be displayed.
BLINK	BYTE	Exactly one bit is assigned to each figure or letter. When the bit is set, that position flashes.
STATES	BYTE	Triggering of fixed display symbols. When the bit is set, the element is triggered.
POINTS	BYTE	Triggering of fixed decimal points. One point is assigned to each digit. When the bit is set, the point is triggered.

Function outputs, none

Description

With the function LCD_Text text with a maximum length of 8 digits can be displayed. Values (e.g. from calculations) have to be converted into text (String) first by means of the function STR. The string elements are shown on the individual display elements according to the formatting.

Please remember that letters can only be shown insufficiently on the 7-segment elements.

Changes are accepted when input CHANGE is set to TRUE.
When input CHANGE is set to FALSE the modifications do not become effective.

Assignment of attributes, symbols and decimal points to the individual bits (bit 0 corresponds to the LSB):

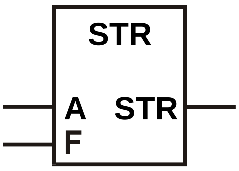
Bit	Flash	Symbol	Point
0	Digit 1	CH1	Digit 1
1	Digit 2	CH2	Digit 2
2	Digit 3	CH3	Digit 3
3	Digit 4	CH4	Digit 4
4	Digit 5	RUN	Digit 5
5	Letter 1	PRG	▨
6	Letter 2	TST	▨
7	Letter 3	KEY	▨

▨ bit not used

Function STR

Library CSxxxx.LIB

Function symbol



Purpose

The function STR converts a value into a string and formats it.

Parameters

Function inputs

Name	Data type	Description
A	DINT	address of the variable to be converted
F	STRING	formatting instruction

Function outputs

Name	Data type	Description
STR	STRING	formatted string

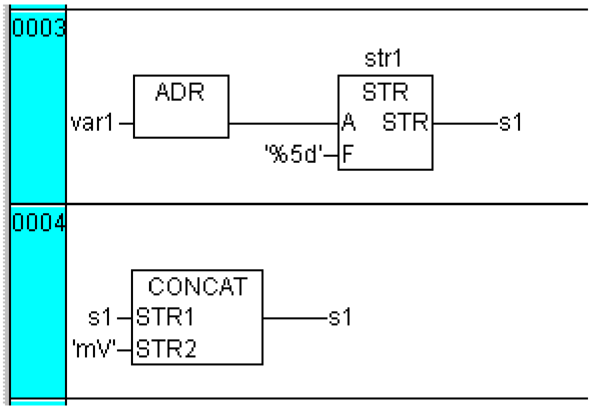
Description

SFR converts a variable value into a string and formats it.

For this purpose the address of the variable has to be transferred to the function. The address is formed with the ADR operator. Additionally a formatting string can be transferred to the function in accordance with the table.

As a result the function provides the converted string which can then e.g. be displayed directly on the integrated display.

With functions from the standard library ST167.LIB the strings can be further processed or linked.



Overview of possible formatting signs :

Sign	**Type of variable**	**Output format**
%d	Integer	signed decimal number
%u	Integer without sign	non-signed decimal number
%o	Integer without sign	non-signed octal number
%x	Integer without sign	non-signed hexadecimal number (0123456789abcdef)
%X	Integer without sign	non-signed hexadecimal number (0123456789ABCDEF)
%f	floating point	floating point number [-]*dddd.dddd*
%e	floating point	floating point number [-]*d.dddd*e[-]*dd*
%E	floating point	floating point number [-]*d..dddd*E[-]*dd*
%g	floating point	floating point number
%G	floating point	floating point number
%C	sign	individual sign

9. Other functions

9.1. Software reset

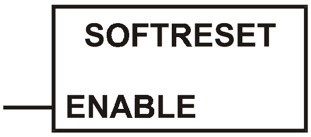
Function

SOFTRESET

Library

CSxxxx.LIB

Function symbol



Purpose

The function SOFTRESET restarts the controller completely.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is processed FALSE: function is not processed

Function outputs, none

Description

SOFTRESET carries out a complete restart of the controller. The function can e.g. be used in connection with CANopen when a node reset is to be carried out. After a SOFTRESET the controller behaves as though the supply voltage has been switched off and on.

☞

In a running communication the long reset phase has to be observed, otherwise a guarding error is shown.

9.2. Save data in memory and read

Automatic saving of data

The controller CS0015 offers the possibility to save data (BOOL, BYTE, WORD, DWORD) in a remanent flash memory. When the supply voltage drops off the data backup is started automatically, if the data are saved in the flag range MW0 ... MW127 (MB0 ... MB255).

The advantage of the automatic saving is that the backup is also started in the case of a sudden voltage drop or an interruption of the supply voltage and the current data values are saved (e.g. counts).

When the supply voltage returns, the saved data are read out from the FLASH via the operating system and are written back in the flag range.

This data range can also be accessed via the CANopen object list (index from 2000 Hex).

Manual saving of data

Apart from the possibility of automatically saving data in the flag range up to MW127 (MB255) the data range between MB256...MB1024 can be saved in the integrated serial EEPROM via a function call. To read out the data another function call needs to be carried out. The data are written or read as a complete block.

Direct memory access

In general the programmer has direct read and write access to the non-remanent flag range via the corresponding IEC addresses.

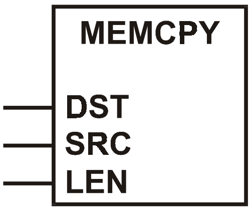
IEC Byte address	IEC Word address	Description
%MB0 ... %MB255	%MW0 ... %MW127	remanent data, automatic saving
%MB256 ... %MB1023	%MW128 ... %MW511	volatile data, can only be saved by calling E2WRITE
%MB1025 ... %MB7935	%MW512 ... %MW3967	volatile data

From the memory mapping (see annex 1.5) the programmer can get information on the available memory range.

Function **MEMCPY**

Library **CSxxxx.LIB**

Function symbol



Purpose

The function MEMCPY enables the writing and reading of different data types directly in the memory.

Parameters

Function inputs

Name	Data type	Description
DST	DWORD	address of target variable
SRC	DWORD	address of source variable
LEN	WORD	number of data bytes

Function outputs, none

Description

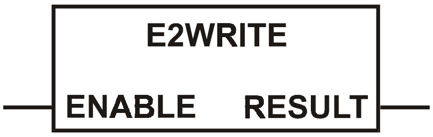
MEMCPY writes the contents of the address from SRC to the address DST transferring exactly as many bytes as were defined under LEN. This enables the transfer of exactly one byte of a word date.

The address has to be found with the function ADR and has to be transferred to MEMCPY.

Function E2WRITE

Library CSxxxx.LIB

Function symbol



Purpose

The function E2WRITE writes a data block in the serial EEPROM.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed

Function outputs

Name	Data type	Description
RESULT	BYTE	0 = function is inactive 1 = function is completed 2 = function is working

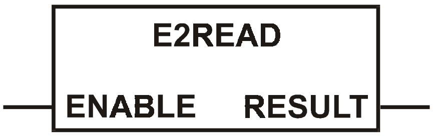
Description

E2WRITE writes the flag range MW128...MW511 in the serial EEPROM. Since the processing of the function requires some time the execution has to be monitored via the function output RESULT. If RESULT = 1 the input ENABLE has to be reset to FALSE.

Function E2READ

Library CSxxxx.LIB

Function symbol



Purpose

The function E2READ reads a data block from a serial EEPROM.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed

Function outputs

Name	Data type	Description
RESULT	BYTE	0 = function is inactive 1 = function is completed 2 = function is working

Description

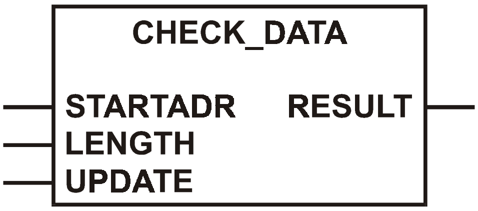
E2READ reads a data block from the serial EEPROM and writes it in the flag range MW128...MW511. As the processing of this function requires some time the execution has to be monitored via the function output RESULT. If RESULT = 1 the input ENABLE has to be reset to FALSE.

Function

CHECK_DATA

Library **CSxxxx.LIB**

Function symbol



Purpose Saving the data in the user data memory via a CRC code

Parameters Function inputs

Name	Data type	Description
STARTADR	DINT	start address of the monitored data memory (address from %MW0...)
LENGTH	WORD	length of the monitored memory (Byte)
UPDATE	BOOL	TRUE: data changes permissible FALSE: data changes not permissible

Function outputs

Name	Data type	Description
RESULT	BOOL	TRUE: function is executed FALSE: function is not executed

Description

In safety-relevant applications the function CHECK_DATA monitors a range of the data memory (possible addresses from %MW0...) for unwanted data changes. For this purpose the function creates a CRC check sum over the stated data range. If there is an unwanted change of data RESULT = FALSE. The result can then be used for further actions (e.g. switching off the outputs).

The start address has to be transferred to the function via the address operator ADR. In addition, the number of data bytes LENGTH (length from STARTADR) has to be indicated. Only if UPDATE = TRUE can the data in the memory range be changed (e.g. via the user program or tdm) and no error message RESULT = FALSE is generated.

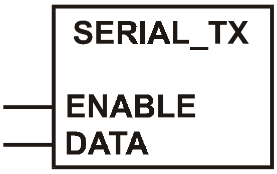
⚠ **The function is a safety function. However, the use of this function does not automatically make the controller a safety controller.**
Only tested and approved controllers with a special operating system can be used as safety controllers.

9.3. Use of the serial interface

Function SERIAL_TX

Library **CSxxxx.LIB**

Function symbol



Purpose Transfers a data byte via the serial RS232 interface.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: transfer released FALSE: transfer blocked
DATA	BYTE	byte date to be transferred

Function outputs, none

Description SERIAL_TX transfers the data byte DATA via the serial interface. The transfer can be released or blocked via the function input ENABLE.

The SERIAL functions are the basis for creating a user-specific protocol for the serial interface.

In general the serial interface is not available to the user as it is used for the program download and the debugging. If the user sets the system flag bit SERIAL_MODE to TRUE the interface can be used. Program download and debugging are **only** possible via the **CAN interface**.

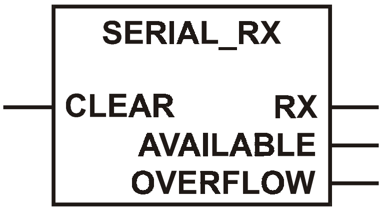
Function

SERIAL_RX

Library

CSxxxx.LIB

Function symbol



Purpose

Reads a received data byte from the serial receiving buffer.

Parameters

Function inputs

Name	Data type	Description
CLEAR	BOOL	TRUE: receiving buffer is deleted FALSE: no data can be stored in buffer

Function output

Name	Data type	Description
RX	BYTE	received byte data from the receiving buffer
AVAILABLE	WORD	number of received data bytes
OVERFLOW	BOOL	overflow of the data buffer, loss of data!

Description

With each call SERIAL_RX reads a data byte from the serial receiving buffer. The value of AVAILABLE is then decremented by 1. If no more data are in the buffer AVAILABLE is 0.

If more than 1000 data bytes are received the buffer overflows and data are lost. This is shown via the bit OVERFLOW.

The SERIAL function is the basis for creating a user-specific protocol for the serial interface.

☞

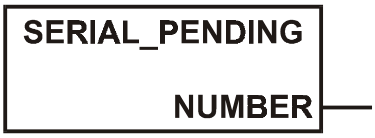
In general the serial interface is not available to the user as it is used for the program download and the debugging. If the user sets the system flag bit SERIAL_MODE to TRUE the interface can be used. Program download and debugging are **only** possible via the **CAN interface**.

Function

SERIAL_PENDING

Library **CSxxxx.LIB**

Function symbol



Purpose

The function determinates the number of data bytes saved in the serial receiving buffer.

Parameters

Function inputs, none

Function output

Name	Data type	Description
NUMBER	WORD	number of received data bytes

Description

SERIAL_PENDING determinates the number of data bytes received in the receiving buffer. As opposed to the function SERIAL_RX the contents of the buffer remains unchanged after calling this function.

The SERIAL functions are the basis for creating a user-specific protocol for the serial interface.

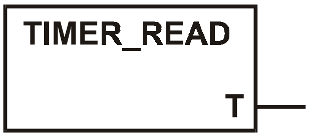
☞ In general the serial interface is not available to the user as it is used for the program download and the debugging. If the user sets the system flag bit SERIAL_MODE to TRUE the interface can be used. Program download and debugging are **only** possible via the **CAN interface**.

9.4. Reading the system time

Function TIMER_READ

Library **CSxxxx.LIB**

Function symbol



Purpose The current system time is read in seconds.

Parameters Function inputs, none

Function output

Name	Data type	Description
T	TIME	Current system time in seconds

Description When the supply voltage is applied a time cycle is generated in the unit and is counted up in a register. This register can be read by means of the function call and can be used e.g. for measuring time.

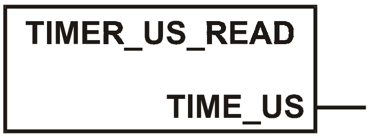
☞ The system timer runs to max. 10 m 55 s 350 ms and then starts again at 0.

Function

TIMER_US_READ

Library **CSxxxx.LIB**

Function symbol



Parameter The current system time is read in µseconds.

Parameters Function inputs, none

Function output

Name	Data type	Description
TIME_US	DWORD	Current system time in µseconds

Description When the supply voltage is applied a time cycle is generated in the unit and is counted up in a register. This register can be read by means of the function call and can be used e.g. for measuring time.

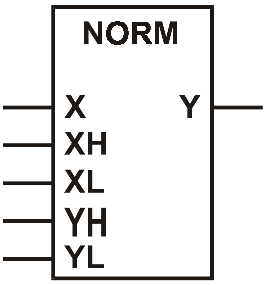
☞ The system timer runs up to a max. value of 4294967295 (µs) and then starts again at 0.

9.5. Processing of variables

Function NORM

Library **CSxxxx.LIB**

Function symbol



Purpose

Norms a value within set limits to a value with new limits

Parameters

Function inputs

Name	Data type	Description
X	WORD	output value
XH	WORD	bottom limit input value range
XL	WORD	upper limit input value range
YH	WORD	bottom limit output value range
YL	WORD	upper limit output value range

Function output

Name	Data type	Description
Y	WORD	normed data type BYTE

Description

The function NORM norms a value of type WORD which lies within the limits XH and HL to an output value within the limits YH and YL.

This function is e.g. used for creating PWM values from analog input values.

Example

not normed value:	**50**
bottom limit value input:	0
upper limit value input:	100
bottom limit value output:	0
upper limit value output:	2000
normed value:	**1000**

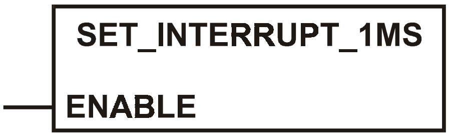
☞ Due to rounding errors in hexadecimal figures a normed value might deviate by 1. If the limits (XH/XL or YH/YL) are stated invertedly the norming is also inverted.

9.6. Real-time processing

Function SET_INTERRUPT_1MS

Library **CRxxxx.LIB**

Function symbol



Purpose Real-time processing of program parts in 1ms cycle.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: Data changes permissible FALSE: Data changes not permissible

Function outputs, none

Description

In classical plc's the cycle time is of extreme importance for real-time processing. Compared to customer-specific controllers this is a disadvantage for the plc. A "real-time operating system" does not make a difference if the complete application program runs in a single task.

A possible solution would be to keep the cycle time short. This often means that the aplication is distributed to several control cycles which makes programming complicated and difficult.

Another possibility would be to call up a certain program part at fixed intervals (in this case 1 ms) independent of the control cycle.

The user combines the time-critical part of the application in a module type PROGRAM (PRG). This module is declared as a 1 ms interrupt routine by once (at the time of initialisation) calling up the function module SET_INTERRUPT_1MS. As a result this program module is processed every millisecond. To avoid calling it up cyclically as well it should be skipped in the cycle (with the exception of the initialising call). If inputs and outputs in this program parts are used, they are also read or written in the 1 ms cycle. That way all time-critical events in the program module can be processed by linking inputs or global variables and writing outputs. Timers can be monitored more accurately as would be possible in the "normal" cycle.

☛

Only one timer interrupt module must be active at one time. You can, however, change to other interrupt modules within the running program depending on the program state.

The time requirement has to be kept short! For this reason calculations, floating point arithmetics or control-loop functions should not be used in ths module:

Important:

The interrupt routine cancels the definiteness of the inputs and outputs in the cycle so that only one part per millisecond can be served.

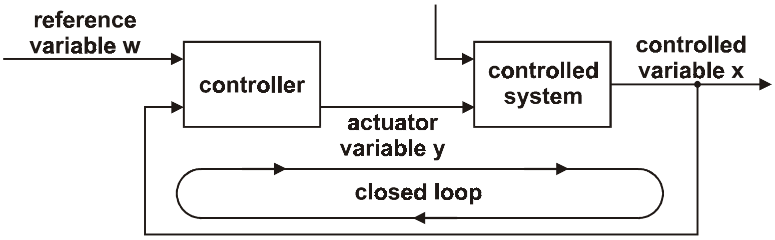
Inputs:	%IX0.00 ... %IX0.07
Outputs:	%QX0.00 ... %QX0.07

All other inputs and outputs are processed once per cycle as usual.

Global variables also lose their definiteness when they accessed more or less simultaneously in the cycle and by the interrupt routine. This problem specially concerns big data types (e.g. DINT).

10. Closed-loop control functions

Closed-loop control is a process in which the variable to be controlled (controlled variable x) is permanently monitored and compared with the reference variable (or preset value). The result of this comparison influences the controlled variable for adjustment to the reference variable.

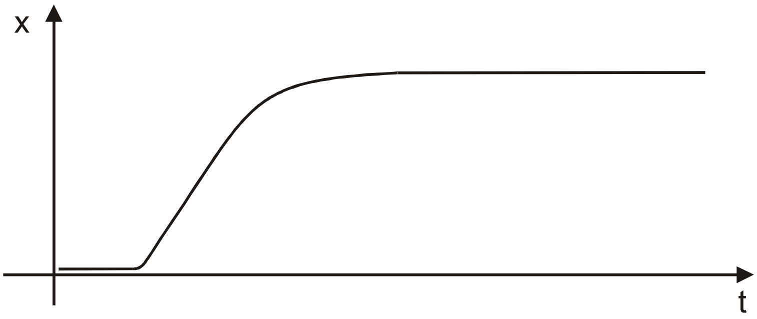


Exact information on the steady-state behaviour and on the dynamic behaviour of the controlled system is required for selecting a suitable controller. In most cases the characteristics are experimentally found and can hardly be influenced.

We distinguish between three types of controlled systems.

Controlled systems with compensation

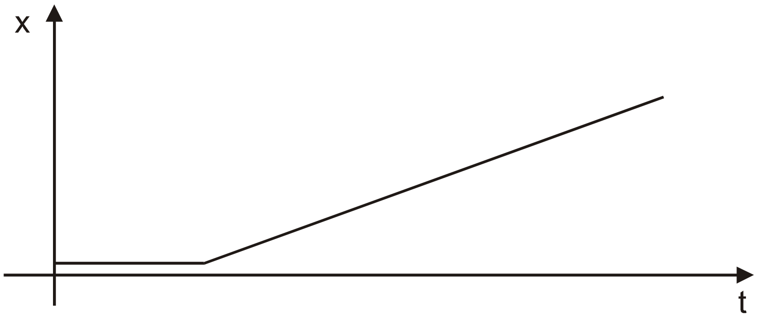
In a controlled system with compensation control variable x is moving towards a new final value (steady state). Important in these controlled systems is the gain (transfer coefficient K_S). The smaller the gain the better the system can be controlled.



These controlled systems are called P(roportional) systems.

Controlled systems without compensation

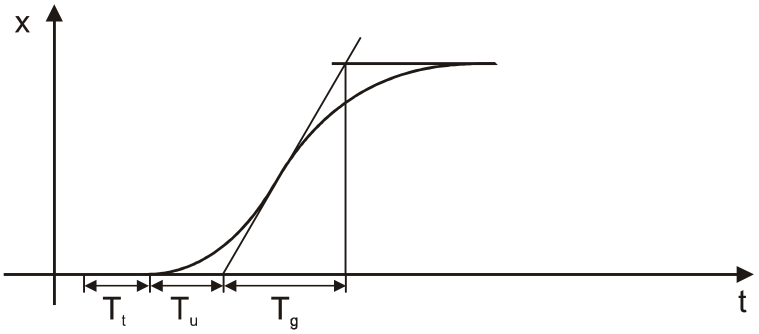
Controlled systems with an amplification factor to infinite are called controlled systems without compensation because of their integrating behaviour. It means that the controlled variable continuously grows after a change of the variable or due to interference. It therefore never reaches a final value.



These controlled systems are called I(ntegral) systems.

Controlled system with delay

Most controlled systems are a series connection of P systems (systems with compensation) and one or several T_1 systems (systems with inertia). A first level controlled systems is e.g. created by connecting a restrictor and a subsequent memory.



In controlled systems with delay the controlled variable responds to a change of the variable only after a delay T_t. The delay T_t or the sum of $T_t + T_u$ is the measure for the possibility to adjust the system. The bigger the T_q / T_u ratio the better the possibility to adjust the system.

The controllers integrated in the library are a summary of the presented basic functions. The functions used and their combination depend on the controlled system.

10.1. Adjustment rule for a controller

The adjustment process by Ziegler and Nickols is of advantage in a closed control loop in the case of controlled systems with unknown time constant.

Adjustment rule

The controller is first operated as a pure P system. The rate time TV is set to 0 and the reset time is set to a very high value (ideally to ∞) for a slow system. For a fast controlled system a small TN should be chosen. The proportional-action coefficient KP is then increased until the system deviation and the variable deviation at KP = $KP_{critical}$ execute constant oscillation at a constant amplitude. The stability limit has been reached. The time period $T_{critical}$ of the permanent oscillation has to be determined. Only add a D part if required. TV should be approx. 2 – 10 times smaller than TN and KP = KD.

The ideal controlled system should be set as follows:

Controlling system	KP = KD	TN	TV
P	2.0 * $KP_{critical}$	-	-
PI	2.2 * $KP_{critical}$	0.83 * $T_{critical}$	-
PID	1.7 * $KP_{critical}$	0.50 * $T_{critical}$	0.125 * $T_{critical}$

☞ **Please note that the controlled system is not affected by the oscillation. In sensitive controlled systems KP should only be increased up to a value at which no oscillation will occur.**

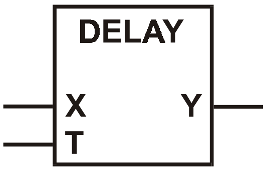
Attenuation of overshooting

The PT1 (low pass filter) function can be used to attenuate overshooting. The set value XS should be attenuated by PT1 before it is integrated in the controller function. The setting variable for T1 should be approx. 4 – 5 times bigger than TN (of the PID or GLR controller).

Function DELAY

Library CSxxxx.LIB

Function symbol



Purpose

Delays the output of the input value by the time T (delay variable).

Parameters

Function input

Name	Data type	Description
X	WORD	input value
T	TIME	delay time

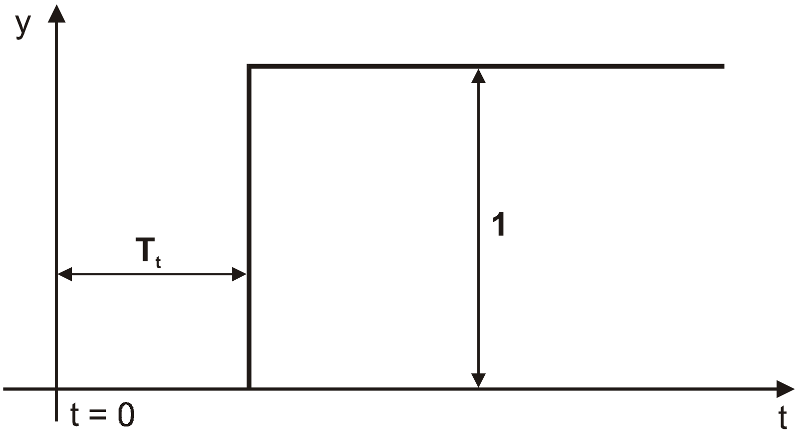
Function output

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

Description

The function DELAY is used to delay an input value by the time T.

The output value y has the following time characteristic.

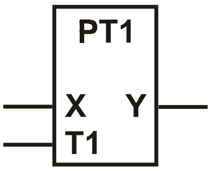


☞ For the function to work without problems it has to be called in each cycle.

Function **PT1**

Library **CSxxxx.LIB**

Function symbol



Purpose Controlled system with first order delay

Parameters Function inputs

Name	Data type	Description
X	WORD	input value
T1	TIME	delay time

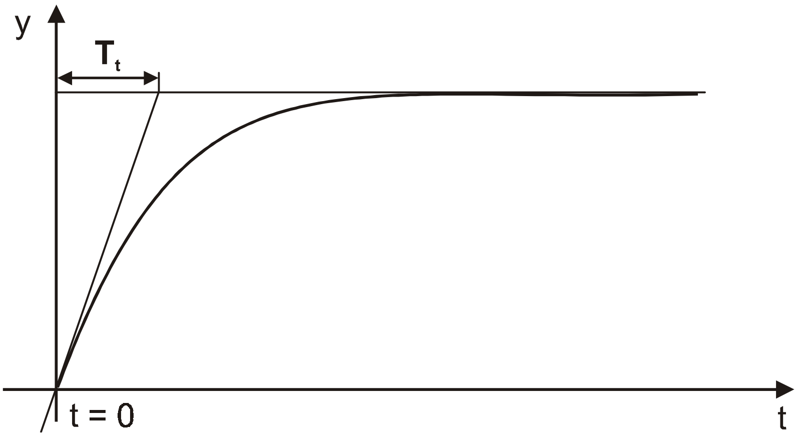
Function output

Name	Data type	Description
Y	WORD	variable

Description

The function PT1 is a proportional system with one time constant. It is used e.g. for establishing ramps when using the PWM functions.

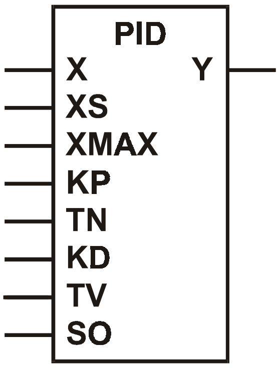
The output variable y (of the low pass filter) has the following unit step response.



Function PID

Library CSxxxx.LIB

Function symbol



Purpose PID controller

Parameter Function inputs

Name	Data type	Description
X	WORD	actual value
XS	WORD	preset value
XMAX	WORD	maximum value of the preset value
KP	BYTE	constant of the P component (/10)
TN	TIME	reset time (I component)
KD	BYTE	proportional part of the D component (/10)
TV	TIME	rate time (D component)
SO	BOOL	self optimisation

Function output

Name	Data type	Description
Y	WORD	variable (actuator value)

Description

The change in the output of the PID controller consists of a proportional, an integral and a differential component. The variable first changes by an amount (D share) depending on the change speed of the input variable. After the rate time TV has elapsed the variable goes back to a value corresponding to the proportional part and then changes in accordance with the reset time TN.

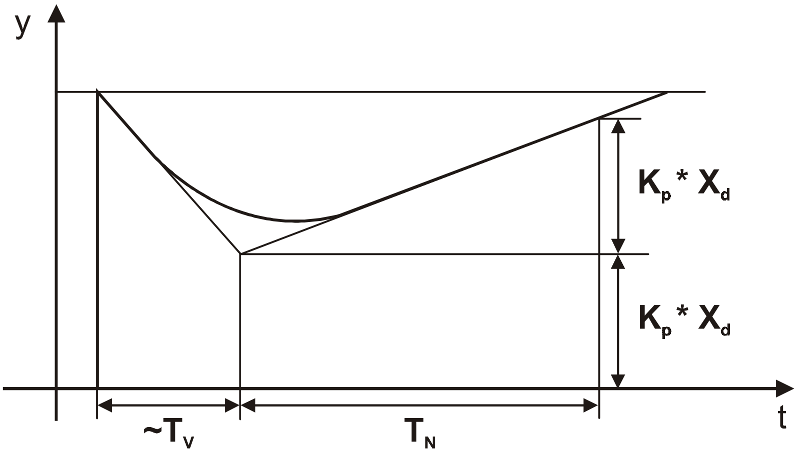
The values entered at function inputs KP and KD are internally divided by 10 to achieve a finer resolution (e.g. KP = 17 corresponds to 1.7)

Variable Y is already normed to the PWM function (RELOAD-Wert = 65535). Observe the inverse logic (65535 = minimum value, 0 = maximum value).

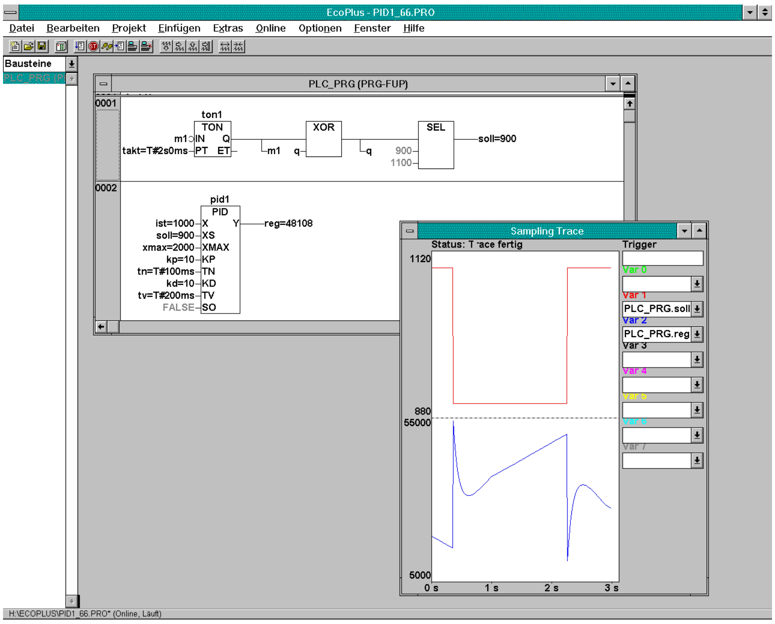
If X > XS, the value is increased.
If X < XS, the value is decreased.

A reference variable is internally added to the value of the manipulated variable: Y = Y + 65536 - (XS / XMAX x 65536).

Variable y has the following time characteristics.



Typical step response of a PID controller



Recommended adjustment:

- Select TN according to the time response of the system (fast system = small TN, slow system = big TN)
- Increase KP slowly step by step up to a value at which no oscillation will occur.
- Readjust TN if necessary.
- Add D part only if rquired: select TV approx. 2 - 10 times smaller than TN. KD should be approx. the same as KP.

Please note that the max. deviation is +/- 127. To get a good dynamic performance this range should not be exceeded, but should be used as fully as possible.

Function input SO (self-optimisation) considerably improves the control characteristics on the condition that the requested features have been reached:

- The controller is operated with I share (TN ≥ 50 ms)
- The parameters KP and especially TN are already well adapted to the real controlled system.
- The control range (X - XS) of +/- 127 is used (if necessary extend the control range by multiplying X, XS and XMAX).

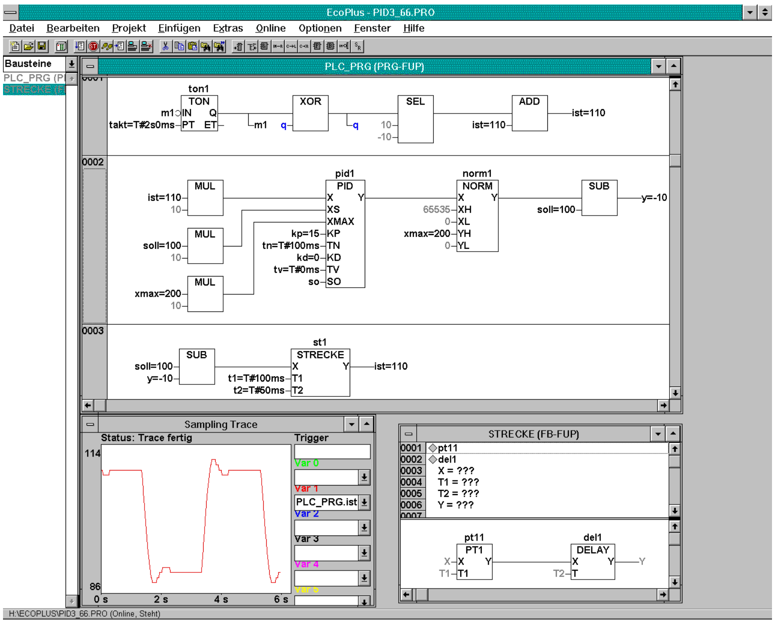
After the parameter settings have been finalised set SO = TRUE. The control characteristics are considerably improved. Overshooting in particular is reduced.

Example

PI controller in a simulated system.

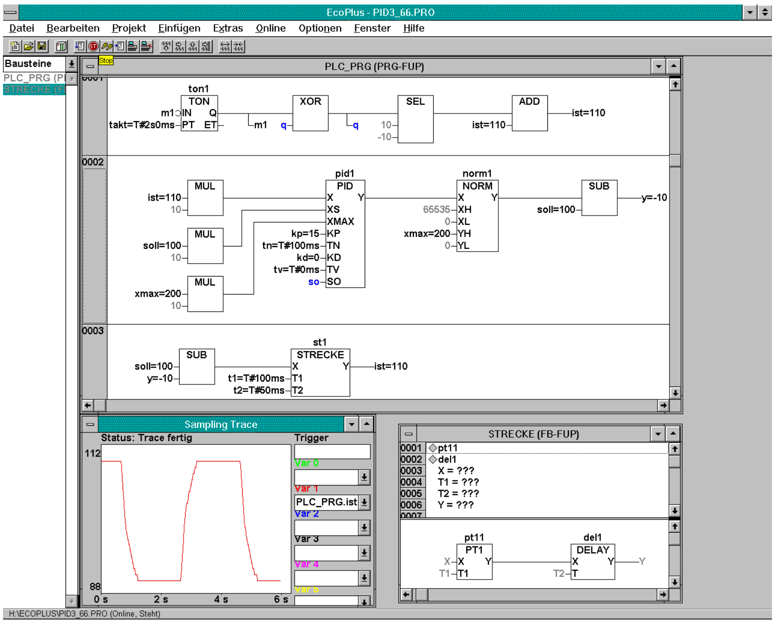
SO = FALSE.

The example shows that overshooting as well as a control range spread occur. The signal is 'in steps' due to the small controlled variable.



SO = TRUE

Overshooting does not occur.



Example

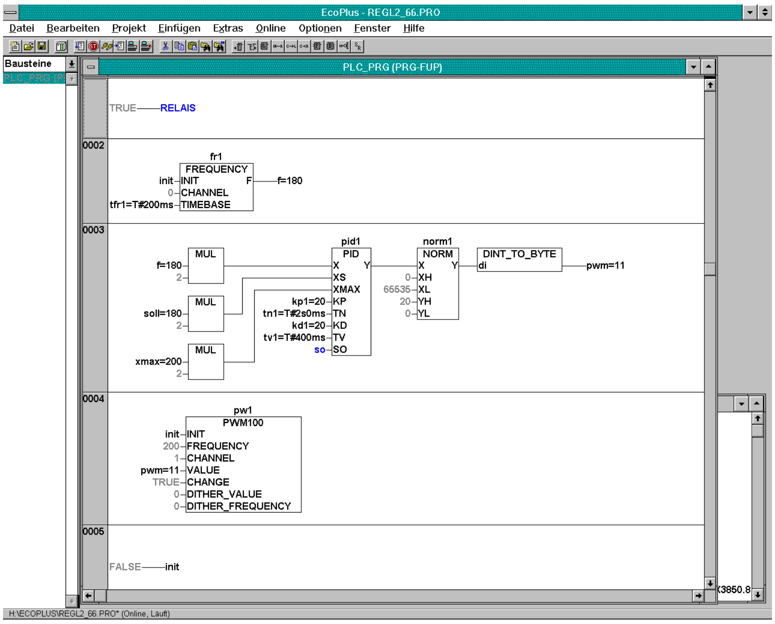
Rotational speed control via PID controller

Features:

- double control range spread
- self-optimisation
- adaptation of the controller output Y to a PWM function module
- TN was adapted to the relatively slow behaviour of the system (centrifugal mass!)
- Overshooting is relatively low despite the D share

Special feature

The motor in the example reaches its maximum speed with 20% PWM. The function module NORM takes this into account.



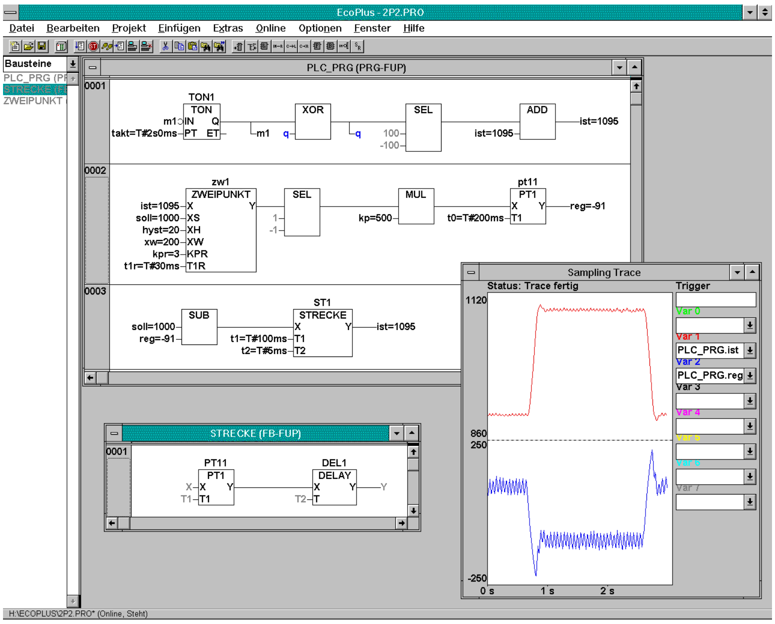
Example

P controller

This P controller consists of a 2-point controller with PT1 feeback and a subsequent PT1 element. The controlled system is simulated.

This controller is particularly robust and is thus suited for difficult system.

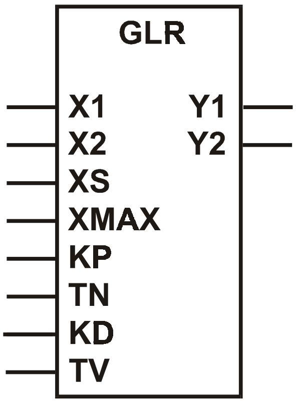
Please note the intended natural oscillation of the controller caused by its internal feedback. The initial rough switching behaviour of the 2-point controller is improved and the switching frequency increases.



Function **GLR**

Library **CSxxxx.LIB**

Function symbol



Purpose synchro-controller

Parameters Function input

Name	Data type	Description
X1	WORD	actual value channel 1
X2	WORD	actual value channel 2
XS	WORD	preset value = reference variable
XMAX	WORD	maximum value of the preset value
KP	BYTE	constant of the P share (/10)
TN	TIME	reset time (I share)
KD	BYTE	proportional share of the D share (/10)
TV	TIME	rate time (D share)

Function outputs

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

Description The synchro-controller is a controller with PID behaviour.

The values entered at function inputs KP and KD are internally divided by 10 so that a finer grading can be achieved (e.g. KP = 17 corresponds to 1.7)

The manipulated variables Y1 and Y2 are already normed to the PWM function (RELOAD value = 65535). Note the inverse logic (65535 = minimum value, 0 = maximum value).

The manipulated variable for the higher actual value is increased, the variable for the smaller actual value corresponds to the reference variable.

Reference variable = 65536 - (XS / XMAX x 65536).

11. Functions of the ecomat tdm R 360

The ecomat tdm R 360 is a programmable dialogue unit with graphics capabilities for displaying data, text, graphics and messages. The functions described below do not deal with the programming of the units, but present the necessary functions for exchanging data with the controller CS0015. The actual programming of the display, e.g. setting up graphic pictures and defining communication parameters , is done with the easy-to-use windows editor ecolog tdm R 360.

☞ Independent of whether or not the controller or the display is programmed via the serial interface the data between a control module and a display are exchanged via the CAN bus. We refer to the description in chapter 6.

The library TDM_x.LIB

As opposed to other libraries the TDM_x.LIB is not programmed in a high-level language (e.g. 'C') or in Assembler, but in the IEC language 'Structured text' (ST). This has the advantage that the expert user can adapt and extend functions to his own requirements.

In the functions TDM_CONFIG and TDM_DATA_TRANSFER the number of variables that can be exchanged with the tdm R 360 is limited.

The basic library only allows communication between a control module and a display. The identifiers are prefixed to the *global variables*. These values have to be entered as communication parameters in the ecolog tdm R 360 software, menu item *Device parameterise..., CAN interface*.

☞

receiving identifier	rxid : WORD := 220
transmitting identifier	txid : WORD := 221

To access a display with e.g. several control modules via the BAN bus each control module has to be allocated its own transmitting identifier in the global variables.

The functions saved in the library support the exchange of data for preset and actual values, the calling of so-called plc pictures, the polling and triggering of unit functions (keyboard, LEDs, unit parameters).

Function groups

The functions can be subdivided in the following groups:

• data exchange, variable definition of preset and actual values	TDM_DATA_TRANSFER TDM_CONFIG TDM_READ_INTERN TDM_WRITE_INTERN
• setting and resetting of plc pictures and messages	TDM_PICTURE TDM_MESSAGE TDM_REFRESH
• polling and evaluation of unit status and resetting of the LEDs	TDM_CONTROL_STATUS_REPORT TDM_REQUEST_STATUS TDM_REPORT_STATUS TDM_REPORT_KEYDATA TDM_LED TDM_SINGLE_LED_ON_OFF
• unit check	TDM_PARAM TDM_RESET

Program example

You will find a program example in function block diagram (FBD) on the program diskette ecolog 100^{plus}. In this simple program the general program setup and the data exchange between the CS0015 and the tdm R 360 are shown .

The following function descriptions will not describe the operation and programming of the display series ecomat tdm R 360. For this information please refer to the unit and software manuals.

11.1. Data exchange and variable definition

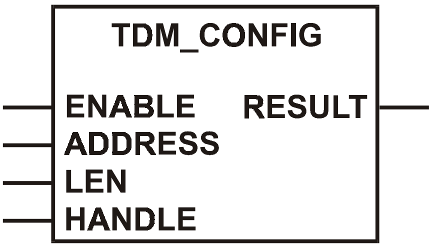
Function

TDM_CONFIG

Library

TDM.LIB

Function symbol



Purpose

The function serves to define those data objects (variables) during initialisation which are to be shown in the tdm R 360.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
ADDRESS	DINT	variable address
LEN	BYTE	number of bytes to be transferred
HANDLE	WORD	designation (number) of the variable in the tdm

Function outputs

Name	Data type	Description
RESULT	BOOL	TRUE: function call was successful

Description

TDM_CONFIG is only called once during the initialisation routine of the application software. The execution can then be blocked via the function call ENABLE.

The input ADDRESS must be allocated the physical address of the variable. Determine the hardware address with the address operator ADR. The result has to be transferred to ADDRESS.

LEN sets the number of bytes to be transferred from the address (e.g. 2 = 2 bytes (WORD), 4 = 4 bytes (DWORD).

Depending on the preset value in the library TDM_x.LIB only 50 values can be defined.

HANDLE is allocated the set variable number from the tdm R 360. HANDLE is the tdm address of the variable. A handle number must only be allocated once in the tdm as well as in the application software.

RESULT shows if the function call was successful. Preset and target values from the controller can only be shown in the display after a one-time successful function call for each variable to be exchanged.

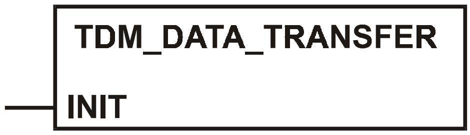
Function

TDM_DATA_TRANSFER

Library

TDM.LIB

Function symbol



Purpose

This function handles the complete data exchange between the tdm R 360 and the controller module.

Parameters

Function inputs

Name	Data type	Description
INIT	BOOL	TRUE: function initialisation FALSE: cyclic function call

Function outputs, none

Description

DATA_TRANSFER is responsible for the complete communication between the display and the controller. By integrating this function the target and preset values, setting and resetting of pictures and messages and the complete unit status are transferred.

To initialise the function it has to be called **once** with TRUE at the INIT input. In subsequent cyles the INIT input must be set to FALSE.

☞

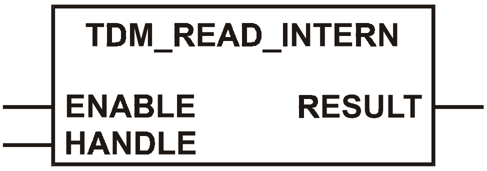
If CANopen is used together with the tdm functions the function has to be called again with INIT = TRUE after a NMT_RESET_NODE/_COMM. Since the tdm functions use direct CAN objects the definitions for those are lost.

In longer controller cycles the function should be called several times to extend the data throughput between the units.

Function TDM_READ_INTERN

Library TDM.LIB

Function symbol



Purpose The function reads an internal tdm R 360 variable.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
HANDLE	WORD	designation (number) of the internal variable in the tdm R 360

Function outputs

Name	Data type	Description
RESULT	BOOL	TRUE: HANDLE was found (was defined)

Description

As opposed to the variables which are processed and generated in the controller as target and preset values and are automatically updated by the operating system of the units, the internal tdm variables (e.g. clock or values saved in the tdm R 360) cannot be 'automatically' read and written.

Just like any other variable an internal variable has to be identified to TDM_CONFIG of the application software. The following read process is executed once by calling the function TDM_READ_INTERN. The function input ENABLE can block the execution.

HANDLE is allocated the defined number of the internal variable from the tdm R 360. A handle number must only be allocated once in the tdm R 360 as well as in the application software.

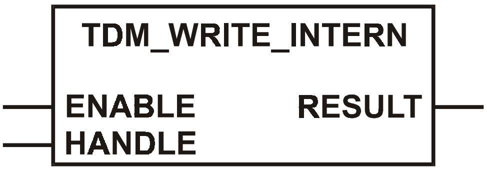
RESULT = TRUE shows if the handle stated when calling the function has been found.

The programmer himself has to ensure the administration of internal data in the controller by means of suitable software routines. A changed internal variable is not automatically transferred to the controller, but only when called for via TDM_READ_INTERN.

Function TDM_WRITE_INTERN

Library TDM.LIB

Function symbol



Purpose

The function writes an internal variable to the tdm R 360.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
HANDLE	WORD	designation (number) of the internal variable in the tdm

Function outputs

Name	Data type	Description
RESULT	BOOL	TRUE: HANDLE was found (was defined)

Description

As opposed to the variables which are processed and generated in the controller as target and preset values and are automatically updated by the operating system of the units, the internal tdm variables (e.g. clock or values saved in the tdm R 360) cannot be 'automatically' read and written.

Just like any other variable an internal variable has to be identified to TDM_CONFIG of the application software. The following read process is executed once by calling the function TDM_WRITE_INTERN. The function input ENABLE can block the execution.

HANDLE is allocated the defined number of the internal variable from the tdm R 360. A handle number must only be allocated once in the tdm R 360 as well as in the application software.

RESULT = TRUE shows if the handle stated when calling the function has been found.

The programmer himself has to ensure the administration of internal data in the controller by means of suitable software routines. A changed internal variable is not automatically transferred to the DISPLAY, but only when called for via TDM_WRITE_INTERN.

11.2. Setting and resetting of pictures and messages

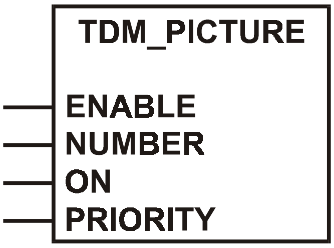
Function

TDM_PICTURE

Library

TDM.LIB

Function symbol



Purpose

The function sets or resets a plc picture..

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
NUMBER	BYTE	tdm picture number
ON	BOOL	TRUE: picture is displayed (set) FALSE: picture is not displayed (reset)
PRIORITY	BOOL	TRUE: picture is displayed (set) as priority picture

Function outputs, none

Description

The call TDM_PICTURE sets or resets a plc picture.

To save cycle time the function input ENABLE can block the execution of the function.

TDM_PICTURE can, but does not have to be called cyclically. A one-time call of the function with the value TRUE being allocated to the function input ON sets the picture defined in input NUMBER. A further call with ON = TRUE has no effect (but requires cycle time for the check). If ON = FALSE is set and the function with the corresponding picture number is called, the picture is reset.

A call if the function with PRIORITY = TRUE marks the plc picture additionally as priority picture. Priority pictures are displayed immediately independent of the current unit status of the display (e.g. preset value entry). All other display activities are suppressed.

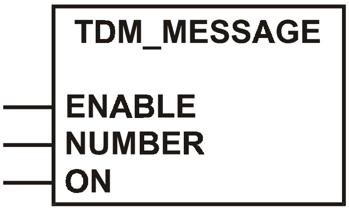
Function

TDM_MESSAGE

Library

TDM.LIB

Function symbol



Purpose

The function sets or resets a message.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
NUMBER	BYTE	tdm message number
ON	BOOL	TRUE: message is displayed (set) FALSE: message is not displayed (reset)

Function outputs, none

Description

The call TDM_MESSAGE sets or resets a plc message.

To save cycle time the function input ENABLE can block the execution of the function.

TDM_MESSAGE can, but does not have to be called cyclically. A one-time call of the function with the value TRUE being allocated to the function input ON sets the picture defined in input NUMBER. A further call with ON = TRUE has no effect (but requires cycle time for the check). If ON = FALSE is set and the function with the corresponding picture number is called, the message is reset.

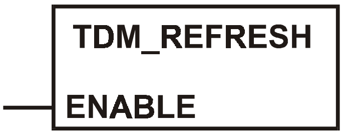
Function

TDM_REFRESH

Library

TDM.LIB

Function symbol



Purpose

The function refreshes the current status of the pictures and messages between controller and display.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed

Function outputs, none

Description

The currently set plc pictures and messages are no longer displayed e.g. after a power failure at the display. By calling TDM_REFRESH the current status is transferred to the display independent of TDM_PICTURE and TDM_MESSAGE.

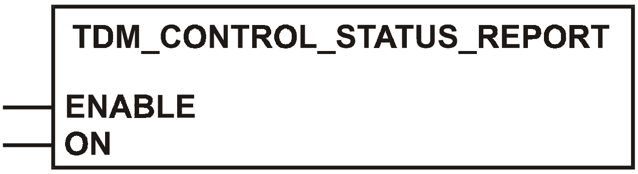
This function does not have to, but should be integrated in the application software. The use of this function relieves the programmer from his job of permanently monitoring the unit status and activating the plc pictures and messages by calling them.

11.3. The unit status and the LEDs

Function **TDM_CONTROL_STATUS_REPORT**

Library **TDM.LIB**

Function symbol



Purpose

The function activates or deactivates the automatic status report of the tdm R 360.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
ON	BOOL	TRUE: automatic status report is switched on FALSE: automatic status report is switched off

Function outputs, none

Description

The status report provides the application software with all relevant unit information:

- currently displayed picture
- currently displayed message
- keyboard status (which key has been pressed or released)
- LED is set or not set
- unit status (e.g. preset value entry active)

TDM_CONTROL_STATUS_REPORT activates the automatic status report. Each change of the above points is transferred to the controller.

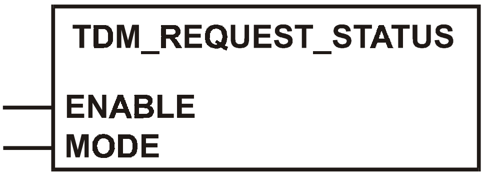
The current status can be evaluated with TDM_REPORT_STATUS and TDM_REPORT_KEYDATA.

The function has to be called only once for switching on or switching off. The execution of the function can then be blocked via the input ENABLE. It is, however, useful to call the functions in certain intervals (e.g. every 500 ms) when TDM_CONTROL_STATUS_REPORT is active. That way it is prevented that the monitoring of the unit status does not work e.g. after a failure of the tdm power supply.

Function TDM_REQUEST_STATUS

Library **TDM.LIB**

Function symbol



Purpose

The function requests the current status report of the tdm R 360.

Parameters

Function input

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
MODE	BYTE	value for selecting the feedback parameters

Function outputs, none

Description

The status report provides the application software with all the relevant unit information:

- currently displayed picture
- currently displayed message
- keyboard status (which key has been pressed or released)
- LED is set or not set
- unit status (e.g. preset value entry active)
- status of the signal output (not CR1000)

TDM_REQUEST_STATUS requests a one-time status report from the display. MODE defines which data are to be polled. Like a one-shot display the status of the above points is transferred to the controller.

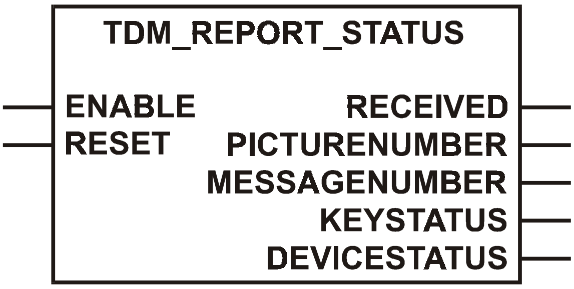
Value	Output via	Description
0	TDM_REPORT_STATUS	unit status
1	TDM_REPORT_KEYDATA	keyboard status (keys 1-32)
2	TDM_REPORT_KEYDATA	keyboard status (keys 33-64)
3	TDM_REPORT_KEYDATA	LED status (keys 1-32)
4	TDM_REPORT_KEYDATA	LED status (keys 33-64)
5	TDM_REPORT_OUTPUT	status of signal output

The function has to be called again for each status report. To block the execution of the function the input ENABLE can be set to FALSE.

Function TDM_REPORT_STATUS

Library TDM.LIB

Function symbol



Purpose The function shows the current status report of the tdm R 360.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
RESET	BOOL	TRUE: sets output RECEIVED to FALSE

Function outputs

Name	Data type	Description
RECEIVED	BOOL	TRUE: new data received FALSE: no new data received
PICTURE-NUMBER	WORD	number of the current picture
MESSAGE-NUMBER	WORD	number of the current message
KEYSTATUS	BYTE	keyboard status
DEVICE-STATUS	BYTE	unit status

Description The function TDM_REPORT_STATUS is called cyclically. Apart from the current picture and message number the status of the keys and the unit are displayed.

For the individual values please refer to the tdm R 360 manual

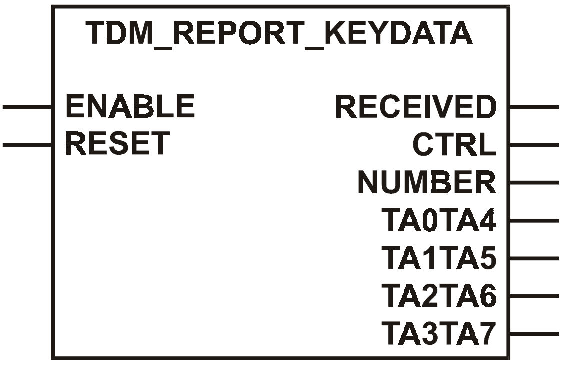
The keyboard bits are generally converted into decimal figures.

The unit status is displayed directly as a decimal value.

Function TDM_REPORT_KEYDATA

Library TDM.LIB

Function symbol



Purpose

The function shows the current keyboard status of the tdm R 360.

Parameters

Function input

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
RESET	BOOL	TRUE: sets output RECEIVED to FALSE

Function outputs

Name	Data type	Description
RECEIVED	BOOL	TRUE: new data received FALSE: no new data received
CTRL	BYTE	control parameters for keys and LED polling
NUMBER	BYTE	only for CTRL = 0, contains key number and status
TA0TA4	BYTE	for CTRL = 0 ... 4, status bytes of keys or LEDs (bit-oriented)
TA1TA5	BYTE	for CTRL = 0 ... 4, status bytes of keys or LEDs (bit-oriented)
TA2TA6	BYTE	for CTRL = 0 ... 4, status bytes of keys or LEDs (bit-oriented)
TA3TA7	BYTE	for CTRL = 0...4, status bytes of keys or LEDs (bit-oriented)

Description

The function TDM_REPORT_KEYDATA can be polled cyclically. It transfers the current key and LED status.

TA0 ... TA7 provide the key/LED status in bytes (e.g. bit-oriented). With CTRL you select the key/LED groups to be polled.

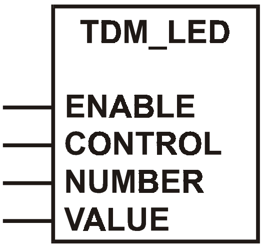
For the individual values please refer to the tdm R 360 manual.

CTRL always returns the value 0 when TDM_CONTROL_STATUS_REPORT is switched on.

Function **TDM_LED**

Library **TDM.LIB**

Function symbol



Purpose The function activates or deactivates the tdm LEDs.

Parameters Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
CONTROL	BYTE	control parameter for setting / resetting the LEDs
NUMBER	BYTE	LED mask number or individual LED number
VALUE	BYTE	mask value when NUMBER = LED mask number

Function outputs, none

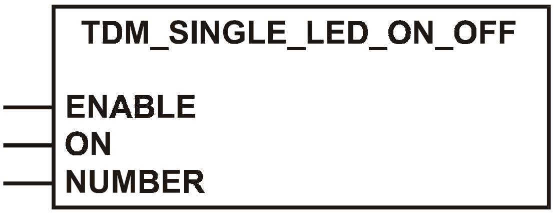
Description The function TDM_LED can be polled cyclically. This function changes the status of the keyboard LEDs (set/reset). Depending on the defined control parameters the LEDs can be processed individually or bit-wise in groups of 8 via a mask value.

For the individual values please refer to the tdm R 360 manual.

Function TDM_SINGLE_LED_ON_OFF

Library TDM.LIB

Function symbol



Purpose

The function activates or deactivates a single tdm LED.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
ON	BOOL	TRUE: LED switched on FALSE: LED switched off
NUMBER	BYTE	individual LED number

Function outputs, none

Description

TDM_SINGLE_LED_ON_OFF can switch an individual LED on (ON=TRUE) or off (ON=FALSE).

As opposed to the function TDM_LED, TDM_SINGLE_LED_ON_OFF can be set permanently. The corresponding CAN command is only transferred once within the function thus preventing an excess workload on the CAN bus.

For the individual values please refer to the tdm R 360 manual.

11.4. Unit control

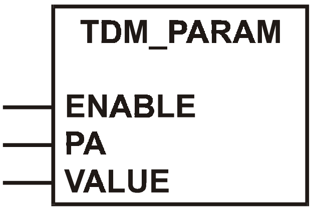
Function

TDM_PARAM

Library

TDM.LIB

Function symbol



Purpose

The function monitors and transfers the unit parameters to the basic setting of the display.

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed
PA	BYTE	parameter code
VALUE	BYTE	additional value, to be stated depending on the parameter code

Function outputs, none

Description

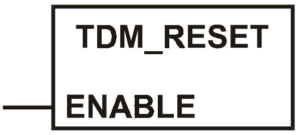
The function TDM_PARAM should only be called to set new unit parameters. With TDM_PARAM you can e.g. change the brightness, the softkey mask or the scroll time for the messages.

For the individual values please refer to the tdm R 360 manual.

Function TDM_RESET

Library **TDM.LIB**

Function symbol



Purpose

The function resets all unit parameters of the tdm R 360 to the basic setting

Parameters

Function inputs

Name	Data type	Description
ENABLE	BOOL	TRUE: function is executed FALSE: function is not executed

Function outputs, none

Description

The display is restarted. It behaves as though the voltage has been switched off and switched on again.

Annex 1. Address allocation CS0015

Annex 1.1. Complete overview

Address	Symbol	Description
%IX0.00-%IX0.07	-	Inputs (byte %IB0, word %IW0)
%IX0.08-%IX0.15	-	Inputs (byte %IB1, word %IW0)
%QX0.00-%QX0.07	-	Outputs (byte %QB0, word %QW0)
%QX0.08-%QX0.15	-	Outputs (byte %QB1, word %QW0)
%IB2	S2	16-step rotary switch
%IX1.8	S3	pushbutton S3
%IX2.0	S4	pushbutton S4
%IX2.8	S5	pushbutton S5
Flag bit*	ERROR	Set error bit
Flag bit*	ERROR_MEMORY	Memory error
Flag bit*	UNLOCK	Release programming mode (FALSE)
Flag bit*	SERIAL_MODE	Switch on serial communication (FALSE)
Flag bit*	CAN_OPEN	Switch on CANopen mode (FALSE)
Flag bit*	ISO_DIRECTION	Transmit or receive data (FALSE)
Flag bit*	CAN_ERROR	CAN-Bus error (collective error bit)
Flag bit*	CAN_INIT_ERROR	CAN initialisation error
Flag bit*	CAN_BUS_OFF_ERROR	CAN-Bus off error
Flag bit*	CAN_DATA_ERROR	CAN-Data error
Flag bit*	CAN_TX_OVERRUN_ERROR	CAN-TX-Overrun error
Flag bit*	CAN_RX_OVERRUN_ERROR	CAN-RX-Overrun error
Flag bit*	COP_SYNCFAIL_ERROR	SYNC object missing
Flag bit*	COP_GUARDFAIL_ERROR	Guarding object missing
Flag byte*	COP_GUARDFAIL_NODEID	Number of missing CANopen slave
Flag bit*	COP_PREOPERATIONAL	CANopen mode preoperational
Flag bit*	COP_PRESYNC	Presync flag
Flag bit*	COP_SYNC	Sync flag
Flag bit*	COP_EVENT_RESETCOM	Communication reset triggered by the master
Flag bit*	COP_EVENT_RESETNODE	Node reset triggered by the master
Flag bit*	COP_GUARDING_AGAIN	Restart Node-Guarding after Reset-Node
Merkerword*	CANBAUDRATE	Currently set CAN-baud rate
Merkerbyte*	NODEID	Currently set node number
Flag bit*	TEXT_MODE	CAN communication compact display (FALSE)
Flag bit*	TEXT_KEY_F1	Function key F1 compact display
Flag bit*	TEXT_KEY_F2	Function key F2 compact display
Flag bit*	TEXT_KEY_F3	Function key F3 compact display
Flag bit*	TEXT_KEY_ESC	ESC key compact display
Flag bit*	TEXT_KEY_LEFT	Key arrow-LEFT compact display
Flag bit*	TEXT_KEY_RIGHT	Key arrow-RIGHT compact display
Flag bit*	TEXT_KEY_DOWN	Key arrow-DOWN compact display
Flag bit*	TEXT_KEY_UP	Key arrow-UP compact display
Flag bit*	TEXT_KEY_ENTER	ENTER key compact display
Flag bit*	TEXT_LED_F1	LED function key F1 compact display
Flag bit*	TEXT_LED_F2	LED function key F1 compact display
Flag bit*	TEXT_LED_F3	LED function key F1 compact display

Annex 1.2. Inputs and outputs

Name	Bit address	Terminal	Comment
I0	%IW0		Input word 0
	%IX0.0	X1, In 0	Frequency/Cycle 0, Encoder 0
	%IX0.1	X1, In 1	Frequency/Cycle 1, Encoder 0
	%IX0.2	X1, In 2	Frequency/Cycle 2, Encoder 0
	%IX0.3	X1, In 3	Frequency/Cycle 3, Encoder 0
	%IX0.4	X1, In 4	
	%IX0.5	X1, In 5	
	%IX0.6	X1, In 6	
	%IX0.7	X1, In 7	
	%IX0.8	X2, In 8	
	%IX0.9	X2, In 9	
	%IX0.10	X2, In 10	
	%IX0.11	X2, In 11	
	%IX0.12	X2, In 12	
	%IX0.13	X2, In 13	
	%IX0.14	X2, In 14	
	%IX0.15	X2, In 15	
Q0	%QW0		Output word 0
	%QX0.00	X3, Out 0	PWM 0
	%QX0.01	X3, Out 1	PWM 1
	%QX0.02	X3, Out 2	PWM 2
	%QX0.03	X3, Out 3	PWM 3
	%QX0.04	X3, Out 4	PWM 4
	%QX0.05	X3, Out 5	PWM 5
	%QX0.06	X3, Out 6	PWM 6
	%QX0.07	X3, Out 7	PWM 7
	%QX0.08	X4, Out 8	
	%QX0.09	X4, Out 9	
	%QX0.10	X4, Out 10	
	%QX0.11	X4, Out 11	
	%QX0.12	X4, Out 12	
	%QX0.13	X4, Out 13	
	%QX0.14	X4, Out 14	
	%QX0.15	X4, Out 15	

Annex 1.3. The flag range

Contents	Flag address	Comment
	%MW 4096	
I/O and system data	%MW 3968	do not write
TX - PDOs	%MW 2032	64 bytes TX-PDOs
RX - PDOs	%MW 2000	64 bytes RX-PDOs
slave data	%MB 1329 %MW 1010	Data range for 32 CANopen I/O slaves
remanent data (function)	%MW 511 %MW 128	768 byte to be saved via function call
remanent data retains)	%MW 127 % MW 0	256 bytes remanent data

The complete flag range in the CS0015 covers 8 kByte. The highlighted fields are allocated directly via the operating system and can only be used for the purpose stated. The remaining memory space can be used by the programming system. It has to be checked for each individual case if it is available to the user. If possible direct addressing should be avoided.

Annex 1.4. CANopen unit interface

- The unit is classified and marked in unit class "Programmable Device" in accordance with CiA DS 405.

- 1 server SDO and the 4 default PDOs are set up in accordance with CiA DS 401. The default Identifiers are allocated in accordance with the "predefined connection set". 2 x 6 PDOs are available in addition.

- The COB-IDs of PDOs and the transfer type (synch / asynch) of the individual PDO can be configured.

- The I/O module expects a SYNC object in the slave mode. The CAN identifier of the synch object can be configured. After a change the ID is automatically saved.

- In the master mode the I/O module generates a SYNC object. The Can identifier of the synch object can be configured. After a change the ID is automatically saved.

- The I/O module supports "node guarding". The "guard time", the "life time factor" and the CAN identifier of the guard object can be configured and are saved.

- The I/O module generates an Emergency Objekt . The COB-ID of the EMCY object can be configured.

Parameter overview

Parameter	Default value set by the manufacturer	Change saved automatically	Valid after
Node ID	32	X	immediately
Baud rate	3 (125 kBit/s)	X	Reset
COB ID SynchOobject	0x80	X	immediately
Communication Cycle	0 (Off)	X	immediately
Guard Time	0 (Off)	X	immediately
Life Time Factor	0 (Off)	X	immediately
COB ID Guarding	0x700 + Node ID	X	immediately
COB ID EMCY	0x80 + Node ID	X	immediately
Transmit Type Receive PDO1	asynchron	-	operational*
Transmit Type Receive PDO2	asynchron	-	operational*
Transmit Type Transmit PDO1	asynchron	-	operational*
Transmit Type Transmit PDO2	asynchron	-	operational*
COP ID Receive PDO1	0x200 + Node ID	-	immediately
COP ID Receive PDO2	0x300 + Node ID	-	immediately
COP ID Transmit PDO1	0x180 + Node ID	-	immediately
COP ID Transmit PDO2	0x280 + Node ID	-	immediately

* The change with PDO_RX_CONFIG und PDO_TX_CONFIG causes a CANopen Reset.
All settings that have not been saved are set to default value.
For this reason a two-step boot-up has to be carried out (see CS0015 as CANopen-Master).

Annex 1.5. Object list

Annex 1.5.1. Data range communication profile, index 1000 to 1FFF

Index	S-Idx	Name	Type	Default	Description
1000	0	device type	u32, ro	0x195	Prof. 405; programmable unit
1001	0	error register	u8, ro	0x0	Bit coded to prof. 301; supports: 0b0000 0000 no error 0b0000 000**1** generic error 0b000**1** 0000 communication error 0b**1**000 0000 manufacturer specific error
1004	0	number of PDOs	u32 ro	0x20002	2 transmitting PDOs 2 receiving PDOs
	1	number of synch PDOs	u32 ro	0x20002	All PDOs can be transferred synchronous or asynchronous.
	2	number of asynch PDOs	u32 ro	0x20002	All PDOs can be transferred synchronous or asynchronous.
1005	0	COB ID SYNC-Object	u32 rw	0x80000080	CAN identifier of the SYNC object
1006	0	Communic. Cycle	u32 rw	0x0	max. time between 2 SYNC objects in µs. resolution = 1 ms.
1007	0	synch window			is not implemented
1008	0	device name	str ro	ecomat 100	unit name: "ecomat 100"
1009	0	HW version	str ro	CS*xxxx_x*	Operating system version
100A	0	SW version	str ro	*jjmmtt*	Software date
100B	0	Node ID	u32 ro		only on request
100C	0	guard time	u16 rw	0x0	Time in ms. The unit expects a "node guarding" of the network master during this time. If value 0 is entered this function is not supported.
100D	0	life time factor	u8 rw	0x0	If no "node guarding" is received for "guard time" x "life time" the unit gives an error message COP_GUARDFAIL_ERROR
100E	0	COB ID guarding	u32 rw	0x00000700 + Node ID	CAN identifier of the node guard object
100F	0	number of SDOs			not implemented (Only the default SDO is supported)
1012	0	Time Stamp			not implemented
1013	0	high res. Time Stamp			not implemented

Index	S-Idx	Name	Type	Default	Description
1014	0	COB ID Emergcy	u32 rw	0x40000080 +Node ID	• I/O module does not respond to external EMCY Message (Bit 31 = 0) • I/O module generates EMCY Message (Bit 30 = 1) • 11 Bit ID (Bit 29 = 0) • ID = 0x80 + Node ID CAN identifier can be changed by the user.
1200	0	Server SDOs	u8 ro	0x02	number of entries
	1	COB ID Rec SDO	u32 rw	0x600+ID	• SDO is valid (Bit 31 = 0) • CAN ID of Receive SDOs
	2	COB ID Trans SDO	u32 rw	0x580 + Node ID	• SDO is valid (Bit 31 = 0) • CAN ID of Transmit SDOs
1400	0	Receive PDO 1	u8 ro	0x02	number of entries RX PDO 1
	1	COB ID	u32 rw	0x200 + Node ID	• PDO is valid (Bit 31 = 0) • CAN ID of 1st RX PDOs
	2	Trans Type	u8 rw	0xFF	• 0x00 = synch acyclic • 0x01 ... 0xF0 = synch cyclic; number of synch objects between two accesses • 0xFC not implemented • 0xFD not implemented • 0xFE = asynch manufac. specific event • 0xFF = asynch device profile event
1401	0	Receive PDO 2	u8 ro	0x02	number of entries RX PDO 2
	1	COB ID	u32 rw	0x300 + Node ID	• PDO is valid (Bit 31 = 0) • CAN ID of 2nd RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1
1402	0	Receive PDO 3	u8 ro	0x02	number of entries RX PDO 3
	1	COB ID	u32 rw	0x382	• PDO is valid (Bit 31 = 0) • CAN ID of 3rd RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1
1403	0	Receive PDO 4	u8 ro	0x02	number of entries RX PDO 4
	1	COB ID	u32 rw	0x383	• PDO is valid (Bit 31 = 0) • CAN ID of 4th RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1
1404	0	Receive PDO 5	u8 ro	0x02	number of entries RX PDO 5
	1	COB ID	u32 rw	0x384	• PDO is valid (Bit 31 = 0) • CAN ID of 5th RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1

Index	S-Idx	Name	Type	Default	Description
1405	0	Receive PDO 6	u8 ro	0x02	number of entries RX PDO 6
	1	COB ID	u32 rw	0x385	• PDO is valid (Bit 31 = 0) • CAN ID of 6th RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1
1406	0	Receive PDO 7	u8 ro	0x02	number of entries RX PDO 7
	1	COB ID	u32 rw	0x386	• PDO is valid (Bit 31 = 0) • CAN ID of 7th RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1
1407	0	Receive PDO 8	u8 ro	0x02	number of entries RX PDO 8
	1	COB ID	u32 rw	0x387	• PDO is valid (Bit 31 = 0) • CAN ID of 8th RX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for RX PDO1
1600	0	Mapping Receive PDO 1	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5800 01	in 5800 SIdx 01, 1st word received data RX PDO 1
	2	Index in object directory	u32 ro	0x5800 02	in 5800 SIdx 02, 2nd word received data RX PDO 1
	3	Index in object directory	u32 ro	0x5800 03	in 5800 SIdx 03, 3rd word received data RX PDO 1
	4	Index in object directory	u32 ro	0x5800 04	in 5800 SIdx 04, 4th word received data RX PDO 1
1601	0	Mapping Receive PDO 2	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5810 01	in 5810 SIdx 01, 1st word received data RX PDO 2
	2	Index in object directory	u32 ro	0x5810 02	in 5810 SIdx 02, 2nd word received data RX PDO 2
	3	Index in object directory	u32 ro	0x5810 03	in 5810 SIdx 03, 3rd word received data RX PDO 2
	4	Index in object directory	u32 ro	0x5810 04	in 5810 SIdx 04, 4th word received data RX PDO 2
1602	0	Mapping Receive PDO 3	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5820 01	in 5820 SIdx 01, 1st word received data RX PDO 3
	2	Index in object directory	u32 ro	0x5820 02	in 5820 SIdx 02, 2nd word received data RX PDO 3
	3	Index in object directory	u32 ro	0x5820 03	in 5820 SIdx 03, 3rd word received data RX PDO 3
	4	Index in object directory	u32 ro	0x5820 04	in 5820 SIdx 04, 4th word received data RX PDO 3

Index	S-Idx	Name	Type	Default	Description
1603	0	Mapping Receive PDO 4	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5830 01	in 5830 SIdx 01, 1st word received data RX PDO 4
	2	Index in object directory	u32 ro	0x5830 02	in 5830 SIdx 02, 2nd word received data RX PDO 4
	3	Index in object directory	u32 ro	0x5830 03	in 5830 SIdx 03, 3rd word received data RX PDO 4
	4	Index in object directory	u32 ro	0x5830 04	in 5830 SIdx 04, 4th word received data RX PDO 4
1604	0	Mapping Receive PDO 5	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5840 01	in 5840 SIdx 01, 1st word received data RX PDO 5
	2	Index in object directory	u32 ro	0x5840 02	in 5840 SIdx 02, 2nd word received data RX PDO 5
	3	Index in object directory	u32 ro	0x5840 03	in 5840 SIdx 03, 3rd word received data RX PDO 5
	4	Index in object directory	u32 ro	0x5840 04	in 5840 SIdx 04, 4th word received data RX PDO 5
1605	0	Mapping Receive PDO 6	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5850 01	in 5850 SIdx 01, 1st word received data RX PDO 6
	2	Index in object directory	u32 ro	0x5850 02	in 5850 SIdx 02, 2nd word received data RX PDO 6
	3	Index in object directory	u32 ro	0x5850 03	in 5850 SIdx 03, 3rd word received data RX PDO 6
	4	Index in object directory	u32 ro	0x5850 04	in 5850 SIdx 04, 4th word received data RX PDO 6
1606	0	Mapping Receive PDO 7	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5860 01	in 5860 SIdx 01, 1st word received data RX PDO 7
	2	Index in object directory	u32 ro	0x5860 02	in 5860 SIdx 02, 2nd word received data RX PDO 7
	3	Index in object directory	u32 ro	0x5860 03	in 5860 SIdx 03, 3rd word received data RX PDO 7
	4	Index in object directory	u32 ro	0x5860 04	in 5860 SIdx 04, 4th word received data RX PDO 7
1607	0	Mapping Receive PDO 8	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0x5870 01	in 5870 SIdx 01, 1st word received data RX PDO 8
	2	Index in object directory	u32 ro	0x5870 02	in 5870 SIdx 02, 2nd word received data RX PDO 8
	3	Index in object directory	u32 ro	0x5870 03	in 5870 SIdx 03, 3rd word received data RX PDO 8
	4	Index in object directory	u32 ro	0x5870 04	in 5870 SIdx 04, 4th word received data RX PDO 8

Index	S-Idx	Name	Type	Default	Description
1800	0	Transmit PDO 1	u8 ro	0x02	Number of entries TX PDO 1
	1	COB ID	u32 rw	0x180 + Node ID	• PDO is valid (Bit 31 = 0) • CAN ID of 1st TX PDOs
	2	Trans Type	u8 rw	0xFF	• 0x00 = synch acyclic • 0x01...0xF0 = synch cyclic; number of synch objects between two accesses • 0xFC not implemented • 0xFD not implemented • 0xFE = asynch manufac. specific event • 0xFF = asynch device profile event
1801	0	Transmit PDO 2	u8 ro	0x02	number of entries TX PDO 2
	1	COB ID	u32 rw	0x280 + Node ID	• PDO is valid (Bit 31 = 0) • CAN ID of 2nd TX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1
1802	0	Transmit PDO 3	u8 ro	0x02	number of entries TX PDO 3
	1	COB ID	u32 rw	0x38A	• PDO is valid (Bit 31 = 0) • CAN ID of 3rd TX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1
1803	0	Transmit PDO 4	u8 ro	0x02	number of entries TX PDO 4
	1	COB ID	u32 rw	0x38B	• PDO is valid (Bit 31 = 0) • CAN ID of 4th TX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1
1804	0	Transmit PDO 5	u8 ro	0x02	number of entries TX PDO 5
	1	COB ID	u32 rw	0x38C	• PDO is valid (Bit 31 = 0) • CAN ID of 5th TX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1
1805	0	Transmit PDO 6	u8 ro	0x02	number of entries TX PDO 6
	1	COB ID	u32 rw	0x38D	• PDO is valid (Bit 31 = 0) • CAN ID of 6th X PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1
1806	0	Transmit PDO 7	u8 ro	0x02	number of entries TX PDO 7
	1	COB ID	u32 rw	0x38E	• PDO is valid (Bit 31 = 0) • CAN ID of 7th TX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1
1807	0	Transmit PDO 8	u8 ro	0x02	number of entries TX PDO 8
	1	COB ID	u32 rw	0x38F	• PDO is valid (Bit 31 = 0) • CAN ID of 8th TX PDOs
	2	Trans Type	u8 rw	0xFF	permissible values as for TX PDO1

Index	S-Idx	Name	Type	Default	Description
1A00	0	Mapping Transmit PDO 1	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA100 01	in A100 SIdx 01, 1st word transmitted data TX PDO 1
	2	Index in object directory	u32 ro	0xA100 02	in A100 SIdx 02, 2nd word transmitted data TX PDO 1
	3	Index in object directory	u32 ro	0xA100 03	in A100 SIdx 03, 3rd word transmitted data TX PDO 1
	4	Index in object directory	u32 ro	0xA100 04	in A100 SIdx 04, 4th word transmitted data TX PDO 1
1A01	0	Mapping Transmit PDO 2	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA101 01	in A101 SIdx 01, 1st word transmitted data TX PDO 2
	2	Index in object directory	u32 ro	0xA101 02	in A101 SIdx 02, 2nd word transmitted data TX PDO 2
	3	Index in object directory	u32 ro	0xA101 03	in A101 SIdx 03, 3rd word transmitted data TX PDO 2
	4	Index in object directory	u32 ro	0xA101 04	in A101 SIdx 04, 4th word transmitted data TX PDO 2
1A02	0	Mapping Transmit PDO 3	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA102 01	in A102 SIdx 01, 1st word transmitted data TX PDO 3
	2	Index in object directory	u32 ro	0xA102 02	in A102 SIdx 02, 2nd word transmitted data TX PDO 3
	3	Index in object directory	u32 ro	0xA102 03	in A102 SIdx 03, 3rd word transmitted data TX PDO 3
	4	Index in object directory	u32 ro	0xA102 04	in A102 SIdx 04, 4th word transmitted data TX PDO 3
1A03	0	Mapping Transmit PDO 4	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA103 01	in A103 SIdx 01, 1st word transmitted data TX PDO 4
	2	Index in object directory	u32 ro	0xA103 02	in A103 SIdx 02, 2nd word transmitted data TX PDO 4
	3	Index in object directory	u32 ro	0xA103 03	in A103 SIdx 03, 3rd word transmitted data TX PDO 4
	4	Index in object directory	u32 ro	0xA103 04	in A103 SIdx 04, 4th word transmitted data TX PDO 4
1A04	0	Mapping Transmit PDO 4	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA104 01	in A104 SIdx 01, 1st word transmitted data TX PDO 5
	2	Index in object directory	u32 ro	0xA104 02	in A104 SIdx 02, 2nd word transmitted data TX PDO 5
	3	Index in object directory	u32 ro	0xA104 03	in A104 SIdx 03, 3rd word transmitted data TX PDO 5
	4	Index in object directory	u32 ro	0xA104 04	in A104 SIdx 04, 4th word transmitted data TX PDO 5

Index	S-Idx	Name	Type	Default	Description
1A05	0	Mapping Transmit PDO 6	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA105 01	in A105 SIdx 01. 1st word transmitted data TX PDO 6
	2	Index in object directory	u32 ro	0xA105 02	in A105 SIdx 02, 2nd word transmitted data TX PDO 6
	3	Index in object directory	u32 ro	0xA105 03	in A105 SIdx 03, 3rd word transmitted data TX PDO 6
	4	Index in object directory	u32 ro	0xA105 04	in A105 SIdx 04,4th word transmitted data TX PDO 6
1A06	0	Mapping Transmit PDO 7	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA106 01	in A106 SIdx 01,1st word transmitted data TX PDO 7
	2	Index in object directory	u32 ro	0xA106 02	in A106 SIdx 02,2nd word transmitted data TX PDO 7
	3	Index in object directory	u32 ro	0xA106 03	in A106 SIdx 03,3rd word transmitted data TX PDO 7
	4	Index in object directory	u32 ro	0xA106 04	in A106 SIdx 04,4th word transmitted data TX PDO 7
1A07	0	Mapping Transmit PDO 8	u32 ro	0x04	number of implemented application objects
	1	Index in object directory	u32 ro	0xA107 01	in A107 SIdx 01,1st word transmitted data TX PDO 8
	2	Index in object directory	u32 ro	0xA107 02	in A107 SIdx 02,2nd word transmitted data TX PDO 8
	3	Index in object directory	u32 ro	0xA107 03	in A107 SIdx 03, 3rd word transmitted data TX PDO 8
	4	Index in object directory	u32 ro	0xA107 04	in A107 SIdx 04,4th word transmitted data TX PDO 8

Annex 1.5.2. Range of manufacturer-specific data, index 2000 to 5FFF

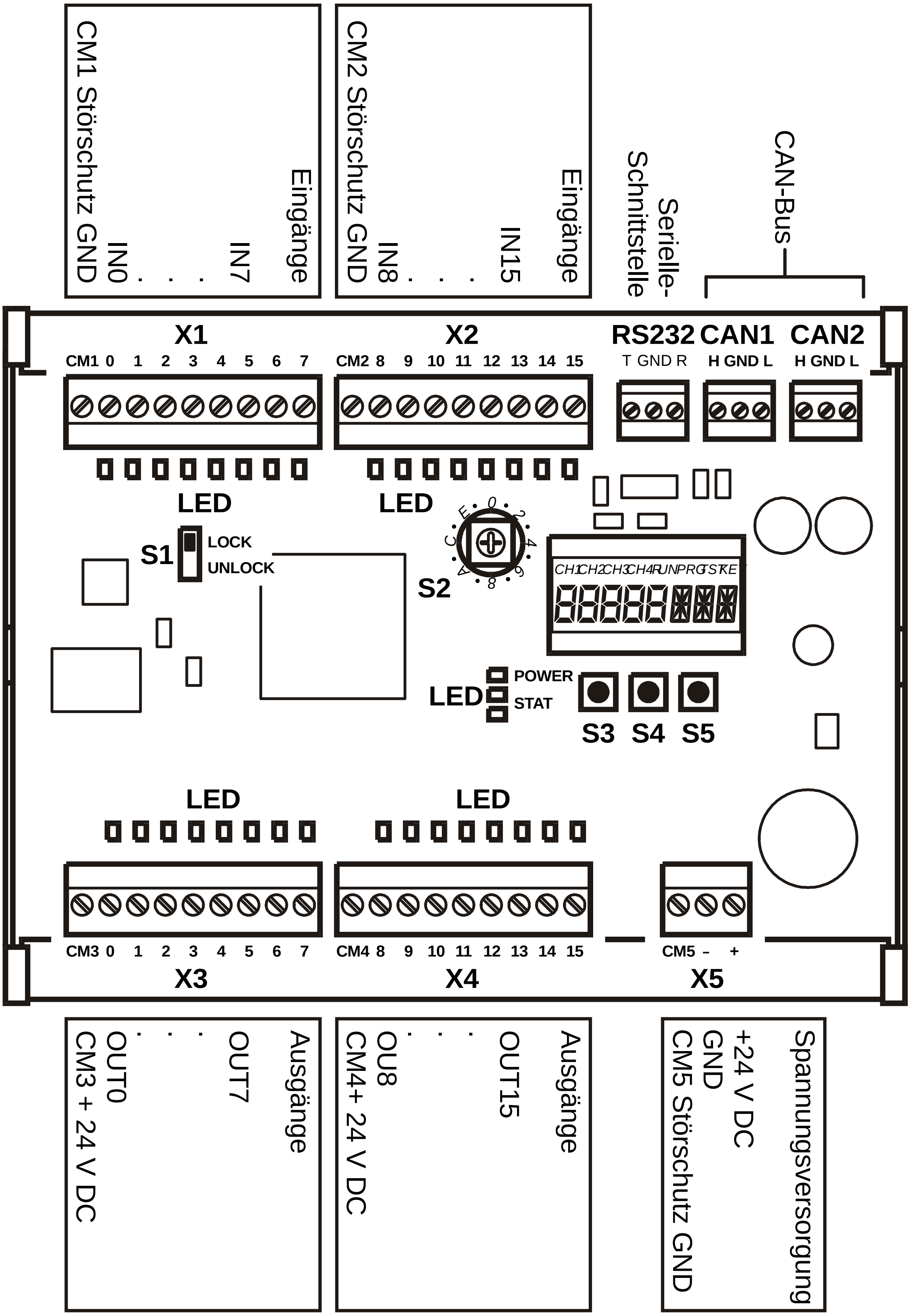
Index	S-Idx	Name	Type	Default	Description
2000	0	Retain Data	dom. rw	0x0	Maximum 256 Byte data stored in the retain marker range between %MW0 ... %MW127.
20F0	0	Setting Node ID	u8 rw	0x20	Node ID with which the I/O module in the CANopen network is addressed. A change is only taken over when the same changed value is entered in the entries 20F0 and 20F1. The change is valid immediately .
20F1	0	Setting Node ID	u8 rw	0x20	
20F2	0	Setting Baud Rate	u8 rw	0x3	Baud Rate of CAN network entry 0 => 1000 kBaud entry 1 => 500 kBaud entry 2 => 250 kBaud entry 3 => 125 kBaud entry 4 => 100 kBaud entry 5 => 50 kBaud entry 6 => 20 kBaud entry 7 => 10 kBaud A change is only taken over when the same changed value is entered in entries 20F2 and 20F3. A change only becomes valid after a reset.
20F3	0	Setting Baud Rate	u8 rw	0x3	

Annex 1.5.3. Legend to object library

Abbrev.	Name	Explanation
u8	unsigned 8	Data length 1 Byte, without sign
u16	unsigned 16	Data length 2 Byte, without sign
u32	unsigned 32	Data length 4 Byte, without sign
dom	domain	Data length variable
ro	read only	Values can only be read
rw	read write	Values can be read and written
0x....	.... Hex	Hexadezimal presentation
0b....	.... binär	Presentation as dual/binary figure

Index and sub-index (S-Idx) of the object directory are shown as hex value.

Annex 2. Wiring


CM1 Störschutz GND
IN0 . . . IN7
Eingänge
CM2 Störschutz GND
IN8 . . . IN15
Eingänge
Serielle-Schnittstelle
CAN-Bus
X1
CM1 0 1 2 3 4 5 6 7
X2
CM2 8 9 10 11 12 13 14 15
RS232
T GND R
CAN1
H GND L
CAN2
H GND L
LED
LED
S1
LOCK
UNLOCK
S2
0 2 4 6 8 A C E
CH1 CH2 CH3 CH4 RUN PRG TST KEY
LED
POWER
STAT
S3 S4 S5
LED
LED
CM3 0 1 2 3 4 5 6 7
X3
CM4 8 9 10 11 12 13 14 15
X4
CM5 – +
X5
CM3 + 24 V DC
OUT0 . . . OUT7
Ausgänge
CM4+ 24 V DC
OU8 . . . OUT15
Ausgänge
CM5 Störschutz GND
GND
+24 V DC
Spannungsversorgung