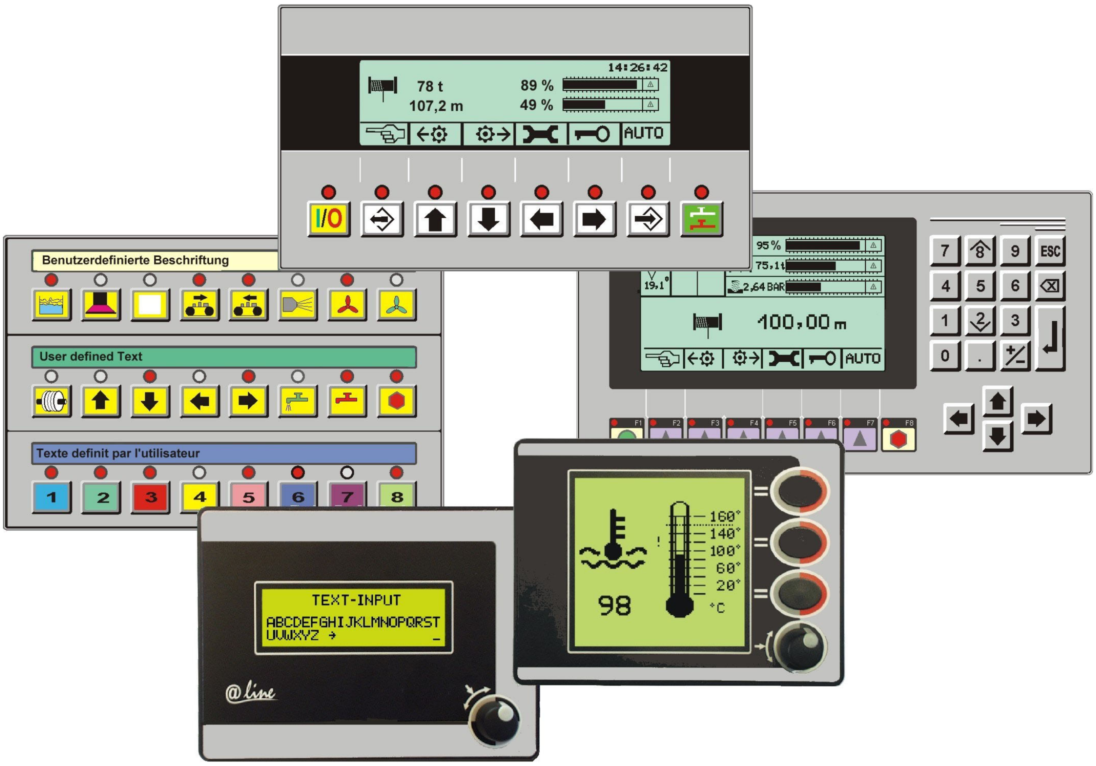
GRAF-SYTECO

Manual

Communication



Document: H127A1

Status: released

Issued: March 2004

SYsteme TEchnischer COmmunikation

GRAF-SYTECO Gmbh & Co.KG * Kaiserstrasse 18 * D-78609 Tuningen

Tel: +49 7464 98660 * Fax: +49 7464 2550 * http:// www.graf-syteco.de * eMail: info@graf-syteco.de

Manual operating panels

1 Communication

1.1 Introduction

This manual describes the manifold possibilities of the operating panels to communicate with other panels or to exchange data. The number and type of the available interfaces are different from panel to panel according to equipment and have therefore to be obtained from the documents of the appropriate panel. The panels are originally able to communicate with controls of various manufacturers directly without any programming effort. For controls which this spectrum does not cover, there exists at any time the possibility to implement their own communication drivers.

1.2 Telegram formats

This chapter describes how to control the operating panel via the serial interface or via the CAN-bus and which information the operating panel can transmit outward directly from the operating system. The transferred user data are principally identical, only the telegram frame and the number of the transferred user data bytes differ from each other.

1.2.1 Structure of the CAN telegrams

The exact structure of CAN telegrams and their telegram framework can be obtained from relevant literature to the CAN-bus. Here a schematic structure is to be shown in order to explain the function mode of the data communication.

0	1	2	3	4	5	6	7	8	9	
Frame		User data								Frame
Identifier		Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	CRC

The identifier in the telegram framework serves only for the identification of the panel at the bus. The number of the data bytes in the identifier is determined at 8 bytes, independent of the actually used number of data bytes. The data bytes marked as unused should be preallocated with 0x00 according to possibility.
With CAN telegrams all 8 user data bytes are always transferred !

1.2.2 Structure of the serial telegrams

Basically each serial telegram consists of a start-byte (STX), a data-length-code (DLN, dimension of the data area), a panel address (ID, node address) a data area (data 0-N) as well as of the checksum (CHK). The dimension of the data area can thereby vary between 1 and 8 bytes. Here the following telegram structure results:

0	1	2	3	4		3+N	4+N
Frame			User data				Frame
STX	DLN	ID	Data 0	Data 1	...	Data N	CHK

STX: Start-byte (identification 0Bhex)
DLN: Data length (from ID to DataN, inclusive)
ID: node address (its own address) = 0
Data1-N: User data
CHK: Checksum (is ascertained from the exclusive-or-linkage of the bytes "DLN" to "DataN", respectively inclusive)

With serial telegrams the necessary data bytes are transferred only partially !
In the following tables on the individual telegram types the data bytes are deposited with grey colour which are transferred via the serial interface.

1.2.3 Structure of the user data

The first byte in the user data (D0) is used for the encoding of the telegram type (TA). So maximum 7 bytes of user data can be transferred in a telegram

D0	D1	D2	D3	D4	D5	D6	D7
Telegram type (TA)	User data in dependence on the telegram type						

The operating panel works on a question-answer-basis. That means the control places an enquiry to the operating panel and receives the corresponding response.
Supplementarily there are some telegrams in which the operating panel informs the control about its status. This function can be switched off via the control.

1.3 Telegram types according to category

In the following overview the telegrams are divided into different categories and refer then to the respective description.

1.3.1 Image- and message call-up

Telegram	Function (rough)	TA=
MESSAGE_ON	Message call-up: Message into the message batch	0x04
MESSAGE_OFF	Message call-up: Message from the message batch	0x05
PAGE_ON	Image call-up: Image into the image batch	0x06
REQUEST_PRIORITY	Image call-up: Image with priority	0x08
PAGE_OFF	Image call-up: Image from the image batch	0x07
MENU_ON	The telegram activates a menu image	0x2B

1.3.2 Keys and LEDs

Telegram	Function (rough)	TA=
SET_LED	Switching on and off LED	0x16
REPORT_KEY_DATA	Key status is reported	0x17
WRITE_KEY_CODE	Key code telegram for nominal value entry	0x2A
EXECUTE_MENU	Carries out a menu function, has an effect like a key stroke	0x29

1.3.3 Variables

Telegram	Function (rough)	TA=
REQUEST_VALUE	Value of an external variable is requested	0x01
SET_VALUE	Value of an external variable is delivered	0x02
REPORT_VALUE	Nominal value is reported to the control	0x03
REQUEST_CLOCK	Read integrated real-time-clock	0x1A
REQUEST_RUNTIME	Read integrated runtime-counter	0x1B
REQUEST_INTERN_VARIABLES	Request value of an internal variable	0x1C
WRITE_CLOCK	Set the time of the internal real-time-clock	0x1D
REPORT_CLOCK	Operating panel transmits time	0x1E
REPORT_RUNTIME	Operating panel transmits runtime counter	0x1F

1.3.4 Status

Telegram	Function (rough)	TA=
REQUEST_STATUS	Request status of the operating panel	0x09
REPORT_STATUS	Operating panel reports status	0x0A
ENABLE_REPORT_STATUS	Operating panel shall report status unrequested	0x0B
DISABLE_REPORT_STATUS	Operating panel shall report status only on request	0x0C
WRITE_PARAM	Modify and guard panel parameter	0x15
REQUEST_VERSION	Requesting firmware version	0x18
REPORT_VERSION	Firmware version is delivered	0x19
REPORT_OUTPUT_STATE	The status of the message output is transmitted	0x26
REPORT_MENU_INDEX	The index of a menu entry is transmitted	0x25

1.3.5 Log/statistics

Telegram	Function (rough)	TA=
REQUEST_PROTOCOL	Requesting the log memory content	0x21
REQUEST_STATISTIC	Requesting the statistics memory content	0x22

1.3.6 Memory/text transfer

Telegram	Function (rough)	TA=
REQUEST_MEMORY_WRITE	The operating panel is informed that data follow which are to be written into the text memory.	0x0D
DISABLE_WRITE	Terminating the data transfer and RESET	0x0E
WRITE_MEMORY	User data of the data transfer to the operating panel	0x0F
REQUEST_MEMORY_READ	The operating panel is informed that it shall transmit the data from the text memory.	0x10
REPORT_READ_MEMORY	User data of the data transfer to the operating panel	0x11
REPORT_ERROR	Error at memory functions	0x14

1.3.7 Cursor positioning

Telegram	Function (rough)	TA=
REQUEST_CURSOR_POSITION	Enquires the current cursor position in the menu	0x27
WRITE_CURSOR_POSITION	Positions the cursor on a nominal value/menu point	0x28
REPORT_CURSOR_POSITION	Current cursor position is transmitted	0x2A

1.3.8 Others

Telegram	Function (rough)	TA=
RESET	Resetting the panel	0x12
CAN_INIT	New initialisation of the CAN interface	0x2C
ACKNOWLEDGE	Acknowledgement from the operating panel to various telegrams	0x13
ASCII_TELEGRAM	ASCII-data for print-out statistics, log	0x20
DRAW	Drawing of lines and rectangles	0x2E

1.4 Description of the telegram types

A detailed description of the individual telegram types follows from this section. In each description you will find here the information about the function, the direction of the telegram (operating panel -> master or master -> operating panel, whereby the master can be a PLC, a PC or e.g. also a further operating panel) and the content of the user data. The telegram format is described and you receive an example of each telegram type. The representation of all number values is hexadecimal, what is to be expressed by "0x“ placed in front of each number. Example: 0x10 = decimal 16)

1.4.1 REQUEST_VALUE (0x01)

Function:	Requests the current value of an external variables from the PLC. The PLC has to respond with the WRITE_VALUE telegram.
Direction:	Operating panel -> master
User data:	Number (handle) of the requested variable as low-byte and high-byte Binary number without preceding sign (0..65500)
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x01	Handle low-byte	Handle high-byte	not used (0x00)				

Example

0x01	0x03	0x01	0x00	0x00	0x00	0x00	0x00

The variable with handle 259 is requested in the example (1x256 + 3)

1.4.2 SET_VALUE (0x02)

Function:	Generally the answer to the telegram REQUEST_VALUE. The value of the requested variable is transferred. The telegram is also used here to set up the value of the internal variable (load variable). Project-plan the difference internal/external variable in the project planning surface. The operating panel recognizes then external/internal with the help of the variable-handle.
Direction:	Master -> operating panel
User data:	Handle: Number of the variable, binary number without preceding sign (0-65500) Byte 0: low-grade byte of the value Byte 1: : Byte 2: : Byte 3: high-grade byte of the value
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x02	Handle low-byte	Handle high-byte	not used (0x00)	Byte 0	Byte 1	Byte 2	Byte 3

Example

0x02	0x10	0x00	0x00	0x20	0x02	0x00	0x00

The variable with handle 16 (=0x10) is set on the value 544 (2x256 + 32).

Note:: The meaning and the number of the fields „byte 0“ to „byte 3“ depend on the data type of the value responded via the handle. The allocation of handle / data type takes place in the editor ITE.

1.4.3 REPORT_VALUE (0x03)

Function:	Transferring a value to the PLC. This telegram is always transmitted from the operating panel to the control if a nominal value has been entered or changed, or if the PLC has requested the value via REQUEST_INTERN_VARIABLES.
Direction:	Operating panel -> master
User data:	Handle: Number of the variable, binary number without preceding sign (0-65500) Byte 0: low-grade byte of the nominal value/variable Byte 1: : Byte 2: : Byte 3: high-grade byte of the nominal value/variable
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x03	Handle low-byte	Handle high-byte	not used 0x00	Byte 0	Byte 1	Byte 2	Byte 3

Example

0x03	0x25	0x00	0x00	0x80	0x00	0x00	0x00

In the example the value 128 for the variable with handle 37 (=0x25) is displayed.

Note:: The meaning and the number of the fields „byte 0“ to „byte 3“ depend on the data type of the value responded via the handle. The allocation of the handle / data type takes place in the editor ITE.

1.4.4 MESSAGE_ON (0x04)

Function:	Activates a message
Direction:	Master -> operating panel
User data:	Message number: 16-bit number, 1-9999, without preceding sign
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x04	Message number low-byte	Message number high-byte	not used (0x00)				

Example

0x04	0x12	0x01	0x00	0x00	0x00	0x00	0x00

In the example the message 274 (1 x 256 + 18) is activated.

1.4.5 MESSAGE_OFF (0x05)

Function:	Deactivates a message
Direction:	Master -> operating panel
User data:	Message number: 16-bit number, 1-9999, without preceding sign
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x05	Message number low-byte	Message number high-byte	not used (0x00)				

Example

0x05	0x03	0x00	0x00	0x00	0x00	0x00	0x00

In the example the message 3 is deactivated.

1.4.6 PAGE_ON (0x06)

Function:	Activates an image
Direction:	Master -> operating panel
User data:	Image number: 16-bit number, 1-9999, without preceding sign
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x04	Image number low-byte	Image number high-byte	not used (0x00)				

Example

0x06	0x04	0x00	0x00	0x00	0x00	0x00	0x00

In the example the image 1024 is activated.

1.4.7 PAGE_OFF (0x07)

Function:	Deactivates an image
Direction:	Master -> operating panel
User data:	Image number: 16-bit number, 1-9999, without preceding sign
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x07	Image number low-byte	Image number high-byte	not used (0x00)				

Example

0x07	0x12	0x07	0x00	0x00	0x00	0x00	0x00

In the example the image 7 is deactivated.

1.4.8 REQUEST_PRIORITY (0x08)

Function:	With this telegram the PLC interrupts all running activities at the operating panel and indicates an image as priority image.
Direction:	Master -> operating panel
User data:	Image number: 16-bit number, 1-9999, without preceding sign
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x08	Image number low-byte	Image number high-byte	not used (0x00)				

Example

0x08	0x68	0x00	0x00	0x00	0x00	0x00	0x00

In the example the image No. 100 with priority is activated.

1.4.9 REQUEST_STATUS (0x09)

Function:	With this telegram the PLC can enquire the current status of the operating panel. It enables the query of key-, LED- and panel status. The operating panel responds accordingly with the telegrams REPORT_KEY_DATA (key- or LED status), REPORT_STATUS (panel status) or REPORT_OUTPUT_STATE (message output)
Direction:	Master -> operating panel
User data:	Type of the status data to be queried 00 hex --> panel status (response: REPORT_STATUS) 01 hex --> keyboard status (key 1 - 32) (response: REPORT_KEY_DATA) 02 hex --> keyboard status (key 33 - 64) (response: REPORT_KEY_DATA) 03 hex --> LED status (LED 1 - 32) (response: REPORT_KEY_DATA) 04 hex --> LED status (LED 33 - 64) (response: REPORT_KEY_DATA) 05 hex --> status message outputs (response: REPORT_OUTPUT_STATE)
DLN	3

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x09	Mode	not used (0x00)					

Example

0x09	0x01	0x00	0x00	0x00	0x00	0x00	0x00

In the example the status of the keys 1-32 is queried. The telegram REPORT_KEY_DATA would be transmitted from the operating panel as response.

1.4.10 REPORT_STATUS (0x0A)

Function:	This telegram is dispatched from the operating panel to the PLC and can be used from the PLC for the detection of the operating panel status. The telegram is always dispatched by the operating panel when a status information in the operating panel changes or the telegram REQUEST_STATUS is received. If DISABLE_REPORT_STATUS has been received, then the telegram is transmitted only upon request REQUEST_STATUS.
Direction:	Operating panel -> master
User data:	Number of the image just displayed, number of the message just displayed, key- and panel status Byte 1,2: Image number: 16 bits, 1-9999, without preceding sign Byte 3,4: Message number: 16 bits, 1-9999, without preceding sign Byte 5: Key status Bit0 = not used Bit1 = ESC Bit2 = arrow on the left Bit3 = arrow on the right Bit4 = arrow downwards Bit5 = arrow upwards Bit6 = ENTER Bit7 = not used Byte 6: Operating panel status (values for bits 0 to 4, bits 5 to 7 are reserved) 0: not defined (error !) 1: Passive status. 2: Browsing images 3: Browsing batch images 4: Browsing messages 5: Selecting menu entry 6: Selecting nominal value 7: Editing nominal value with number input 8: Editing nominal value with step-value processing 9: Acknowledging message 10: Browsing the log 11: Browsing statistics 12-31: not defined (error !)
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x09	Image number low-byte	Image number high-byte	Message number low-byte	Message number high-byte	Key status	Panel status	not used

Example

0x0A	0x05	0x01	0x04	0x00	0x04	0x05	0x00

In the example the operating panel reports:
- *Image 261 is displayed,*
- *Message 4 is displayed*
- *Key 3 is pressed*
- *The panel is in the status "select menu entry", that means the cursor is on a menu point.*

1.4.11 ENABLE_REPORT_STATUS (0x0B)

Function:	The automatic transmitting when changing the status is switched on. The operating panel transmits now unrequested with each status modification a REPORT_STATUS telegram
Direction:	Master -> operating panel
User data:	none
DLN	2

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x0B	not used (0x00)						

Example

0x0B	0x00	0x00	0x00	0x00	0x00	0x00	0x00

The function REPORT_STATUS is activated in the example. The operating panel transmits immediately a REPORT_STATUS - telegram with a change of the panel status.

Note:: Changes of the key status (key pressed/released) are not reported via the REPORT_STATUS telegram, but via the telegram REPORT_KEY_DATA.
Switching off the keyboard telegrams takes place with the project planning software ITE.

1.4.12 DISABLE_REPORT_STATUS (0x0C)

Function:	The automatic transmitting when changing the status is switched off. If the status of the operating panel changes, this is not reported to the control any longer. It has to query the status of the operating panel via REQUEST_STATUS.
Direction:	Master -> operating panel
User data:	none
DLN	2

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x0C	not used (0x00)						

Example

0x0C	0x00	0x00	0x00	0x00	0x00	0x00	0x00

The function DISABLE_REPORT_STATUS is executed in the example.

Note:: Changes of the key status (key pressed/released) are not reported via the REPORT_STATUS telegram, but via the telegram REPORT_KEY_DATA.
are not affected by this and are transmitted further.

1.4.13 REQUEST_MEMORY_WRITE (0x0D)

Function:	With this telegram the write lockout of the installed text/program memory is cleared as well as address and number of the bytes to be written are announced
Direction:	Master -> operating panel
User data:	Initial address of the area to be written and number of the bytes to be written Byte 1: Bank number (memory page) of the area to be written Byte 2,3: Address bits A0-A7 and A8-A15 for area start Byte 4,5: Number of data bytes (low-byte and high-byte) Byte 6,7: Clear the write protection of the memory and have to be transferred as described below

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x0D	Bank number	Start address low-byte	Start address high-byte	Number of bytes low-byte	Number of bytes high-byte	**0xE5**	**0xAA**

IMPORTANT NOTE !
The use of this telegram requires exact knowledge about the internal memory allocation of the operating panel. If this telegram is used wrongly, then this can lead to malfunctions of the panel. The user project has then to be booted up again into the panel with the project-planning software. The operating panel is switched into the "terminal ready for upload"-status by this telegram. Users, who use the SENDDATA.DAT data generation find this telegram in the download data flow.

1.4.14 DISABLE_WRITE (0x0E)

Function:	With this telegram the write lockout of the installed text/program memory is set and the panel is reset (RESET)
Direction:	Master -> operating panel
User data:	The data bytes 1-7 have to be filled with the values specified below. This serves for the backup of the telegram, because the telegram is accepted only if all data bytes have been received in such a way.

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x0E	0xD8	0x47	0x33	0xE5	0x4C	**0xAA**	**0x29**

CAUTION !
Observe also the note on REQUEST_MEMORY_WRITE. The telegram terminates the status "terminal ready for upload" triggered by REQUEST_WRITE_MEMORY or REQUEST_MEMORY_READ.

1.4.15 WRITE_MEMORY (0x0F)

Function:	With this telegram the data to be written are transferred. Five bytes of user data are transferred per telegram.
Direction:	Master -> operating panel
User data:	Byte 1,2: Running telegram number (low- and high-byte) Bytes 3-7: 5 bytes of data to be written

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x0F	serial number low-byte	serial number high-byte	Data byte 0	Data byte 1	Data byte 2	**Data byte 3**	**Data byte 4**

CAUTION !
Observe also the notes to REQUEST_WRITE_MEMORY and DISABLE_WRITE

1.4.16 REQUEST_MEMORY_READ (0x10)

Function:	With this telegram the content of the memory in the operating panel can be read out
Direction:	Master -> operating panel
User data:	Initial address of the area to be read and number of the bytes to be read Byte 1: Bank number (memory page) of the area to be read Byte 2,3: Address bits A0-A7 and A8-A15 for area start Byte 4,5: Number of data bytes low-byte and high-byte Byte 6,7: Clear the write protection of the memory and it has to be transferred as described below

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x0E	0xD8	0x47	0x33	0xE5	0x4C	**0xAA**	**0x29**

NOTE!
The use of this telegram is only useful if the internal memory structure of the operating panel is well known. The operating panel goes to "terminal ready for upload" and transmits the requested data via REPORT_READ_MEMORY. The data transfer is terminated via the telegram „DISABLE_WRITE".

1.4.17 REPORT_READ_MEMORY (0x11)

Function:	With this telegram the operating panel responds to a read enquiry REQUEST_MEMORY_READ
Direction:	Operating panel -> master
User data:	Byte 1,2: Serial telegram number; low- and high-byte first telegram has No. 1 (!) Bytes 3-7: 5 bytes of read data

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x11	serial number low-byte	serial number high-byte	Data byte 0	Data byte 1	Data byte 2	**Data byte 3**	**Data byte 4**

1.4.18 RESET (0x12)

Function:	With this telegram the operating panel is "restarted". The firmware behaves thereby in such a way as if the voltage is switched off and on again
Direction:	Master -> operating panel
User data:	none
DLN	2

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x12	not used (0x00)						

1.4.19 ACKNOWLEDGE (0x13)

Function:	With this telegram the operating panel acknowledges the telegrams RESET, REQUEST_MEMORY_READ, REQUEST_MEMORY_WRITE, WRITE_MEMORY and DISABLE_WRITE. The ACKNOWLEDGE - telegram is, besides, used for signalling the operational standby after voltage return or watchdog error.
Direction:	Operating panel -> master
User data:	none
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x13	not used (0x00)						

1.4.20 REPORT_ERROR (0x14)

Function:	With this telegram the operating panel informs the PLC about the occurrence of an error.
Direction:	Operating panel -> master
User data:	Error code: 0, 1: Bus off (probable cable problem) 2: Communication error (communication error without data loss) 3: Overrun (data telegrams have been lost) 10: REQUEST_MEMORY_WRITE was faulty 11: DISABLE_WRITE was faulty 12: Buffer overrun (WRITE_MEMORY) 13: Write protection is active (WRITE_MEMORY) 14: WRITE_MEMORY without REQUEST_MEMORY_WRITE 15: Wrong telegram number (WRITE_MEMORY) 16: REQUEST_MEMORY_READ was faulty 50: Error when writing the project data. Probably the flash-memory is defective.
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x14	Error code	not used (0x00)					

Example

0x14	0x0A	0x00	0x00	0x00	0x00	0x00	0x00

In the example it is reported that the telegram REQUEST_MEMORY_WRITE has been received, but has an invalid content.

1.4.21 WRITE_PARAM (0x15)

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x15	PA	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5

Function:	With this telegram the PLC can guard and modify different panel parameters of the operating panel. Caution: The settings are not saved in the flash and have thus to be set always again in the initialization routine !	
Direction:	Master -> operating panel	
DLN	**PA**	**Parameteterize data dependent on field PA**
4	0	set global soft-key mask for menu keys Data 0: Bit 0 is assigned to key 0, Bit 1-Key 1, ... Bit=0: no soft-key function of the key Bit=1: key has soft-key function
4	1	Set contrast Data 0 = 0-23, whereby 23 = maximum contrast (decimal)
	2	Set background lighting. Data 0 = 0-7, whereby 7 = maximum brightness
	3	(only up to TOS IO034S00) set status of the status line Data 0 = 0: status line enabled (faded in) Data 0 = 1: Status line disabled (faded out) Data 0 = 2: Image values (local parameters defined in the image are valid)
	4	only up to TOS IO034S00) set line of the status line Data 0 = 0-7
	5	Adjust scrolling time of the active messages Data 0 = 0: no scrolling function. Data 0=1-32: scrolling time in sec.
	6	Adjust scrolling time of the active images Data 0 = 0: no scrolling function. Data 0=1-32: scrolling time in sec.
	7	Parameterize the allocation of the menu keys Data 0=Key number of the key ESC Data 1=Key number of the key "arrow on the left" Data 2=Key number of the key "arrow on the right" Data 3=Key number of the key "arrow downwards" Data 4=Key number of the key "arrow upwards" Data 5=Key number of the key Enter
	8	Set/reset message output Data 0 = 0: reset message output Data 0 = 1: activate message output
	9	Switching on/off function "automatic nominal value transmitting" in the status "nominal value processing with step function". If the function is activated, then with each key stroke of the "up/down" keys the current nominal value is transmitted to the control. Data 0 = 0: function is switched off Data 0 = 1: function is activated, with each key stroke the current nominal value is transmitted with the answer data telegram.

	10	Set keyboard layout or for optional PS/2 keyboard. Data 1=0 American allocation (US-American) Data 1=1 German allocation Data 1=2 French allocation Data 1=3 English allocation Data 1=4 Italian allocation Data 1=5 Spanish allocation Data 1=6 Swedish/Finnish allocation Data 1=7 Belgian allocation Data 1=8 Danish allocation Data 1=9 Norwegian allocation Data 1=10 Swiss/German and French allocation Data 1=11 Portuguese allocation Keyboard layouts - see manual: Operating and watching
	11	Switching over font. Data 0=Bank number in which the font is **! Do not use this function, it exists only for reason of compatibility!**
	12	Set flash cycle in 10 ms steps. D0=low-byte D1=high-byte
	13	Switch over time zone Data 0=0:Winter time Data 0=1:Summer time Data 0=2:do not use any time zone
	14	Update of the variables in the display: Data 0 = 0: edit only if its value modifies Data 0 = 1: edit again with each "set value"-telegram
	15	Buzzer On / Off Data 0=Mode Data 1=Buzzer ON-Time in 10ms Steps Data 2=Buzzer OFF-Timer in 10ms Steps Data 3=Count Mode = 0 Buzzer is switched off, On- and Off-Time and Count are ignored. Mode = 1 Buzzer is on, quiet Mode = 3 Buzzer is on, loud If Mode is <> 0 and On- and Off-Time both are = 0 the buzzer stays on If Mode is <> 0 and On-Time only is <> 0, the buzzer is on for exactly this time („Single shot“). If Mode is <> 0 and On- and Off-Time both are <> 0, the buzzer runs periodically. In this case only Count is used, and the buzzer beeps Count times

Example

0x15	0x0D	0x01	0x00	0x00	0x00	0x00	0x00

The panel is switched over to summer time in the example.

1.4.22 SET_LED (0x16)

Function:	With this telegram the PLC can control the keyboard-LEDs of the operating panel.
Direction:	Master -> operating panel
DLN 3 5 5 5 4 4	CTRL = control parameter: 0 --> reset all LEDs 1 --> LED-mask with the transferred value AND-link 2 --> LED-mask with the transferred value OR-link 3 --> set LED-mask with the transferred value 4 --> set individual LED 5 --> reset individual LED Number: LED-mask-number (0...7) for CTRL=1...3: 0=LEDs 1-8 (bit-coded) 1=LEDs 9..15 ... 7= LEDs 57-64 LED-number (1-64), for CTRL= 4+5 1=LED 1 ... 64=LED 64 Value: Mask value (bit-oriented), for CTRL = 1-3
DLN	3..5 depending upon function CTRL

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x16	CTRL	Number	Value	not used (0x00)			

Example

0x16	0x03	0x00	0x33	0x00	0x00	0x00	0x00

The LEDs 1, 2, 5 and 6 are switched on, the LEDs 3, 4, 7 and 8 are switched off in the example.

1.4.23 REPORT_KEY_DATA (0x17)

Function:	With this telegram the operating panel transmits the key or LED status to the PLC This takes place either on request of the PLC with a REQUEST_STATUS telegram, or automatically when modifying the key status, provided this function has been switched on with the ITE.
Direction:	Operating panel -> master
User data:	CTRL = control parameter: 0 --> transmitting of status and number of an individual key 1 --> transmitting the key status of the keys 1-32 (TA0-TA3) 2 --> transmitting the key status of the keys 33-64 (TA0-TA3) 3 --> transmitting the key status of the keys 1 - 32 (TA0-TA3) 4 --> transmitting the key status of the keys 1 - 32 (TA0-TA3) Number: valid for CTRL=0 contains the key number (bits 0-6), Bit 7 = status (pressed/released) Exp: number=0A hex -->key 10 has been pressed number=8A hex -->key 10 has been released TA0 to TA7:valid for CTRL=1 to 4 Status bytes of the keys or key LEDs (bit-oriented) TA0 --> Keys/LEDs 1-8 TA0 --> Keys/LEDs 9-15 ... TA0 --> Keys/LEDs 57-64
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x17	CTRL	Number	not used (0x00)	TA0 / TA4	TA1 / TA5	TA2 / TA6	TA3 / TA7

Example

0x17	0x00	0x03	0x00	0x04	0x00	0x00	0x00

It is reported in the example that key 3 has been pressed.

Subsequently a listing of the key numbers:

ITS/AT 61/67		
1-48	TA0 to TA5	corresponds to the function keys F1 - F48
65(0x41)	TA6 bit 0	not used
66(0x42)	TA6 bit 1	Key ESC
67(0x43)	TA6 bit 2	"arrow on the left" / or "number 4" (double allocation)
68(0x44)	TA6 bit 3	"arrow on the right" / or "number 6" (double allocation)
69(0x45)	TA6 bit 4	"arrow downwards" / or "number 2" (double allocation)
70(0x46)	TA6 bit 5	"arrow upwards" / or "number 8" (double allocation)
71(0x47)	TA6 bit 6	Key ENTER

72(0x48)	TA6 bit 7	not used
73(0x49)	TA7 bit 0	Key "number 0"
74(0x4A)	TA7 bit 1	Key "number 1"
75(0x4B)	TA7 bit 2	Key "number 3"
76(0x4C)	TA7 bit 3	Key "number 5"
77(0x4D)	TA7 bit 4	Key "number 7"
78(0x4E)	TA7 bit 5	Key "number 9"
79(0x4F)	TA7 bit 6	Key "decimal point"
80(0x50)	TA7 bit 7	Key "+/-"

ITS/AT 62/63/68/72/78		
1-64	TA0 to TA7	corresponds to the function keys F1 - F64
2-7	TA0 bit 1-6	possess a double function corresponding their definition as menu or function keys
Key F2-F7 defined as menu keys:		
2(0x02)	TA0 bit 1	Key ESC
3(0x03)	TA0 bit 2	"arrow on the left"
4(0x04)	TA0 bit 3	"arrow on the right"
5(0x05)	TA0 bit 4	"arrow downwards"
6(0x06)	TA0 bit 5	"arrow upwards"
7(0x07)	TA0 bit 6	Key ENTER

ITS/AT 71/77		
1-40	TA0 to TA4	corresponds to the function keys F1 - F40
41(0x29)	TA5 bit 0	not used
42(0x2A)	TA5 bit 1	Key ESC
43(0x2B)	TA5 bit 2	"arrow on the left"
44(0x2C)	TA5 bit 3	"arrow on the right"
45(0x2D)	TA5 bit 4	"arrow downwards"
46(0x2E)	TA5 bit 5	"arrow upwards"
47(0x2F)	TA5 bit 6	Key ENTER
48(0x30)	TA5 bit 7	Key BACKSPACE
49(0x31)	TA6 bit 0	Key "number 0"
50(0x32)	TA6 bit 1	Key "number 7"
51(0x33)	TA6 bit 2	Key "number 8"
52(0x34)	TA6 bit 3	Key "number 9"
53(0x35)	TA6 bit 4	Key "number 4"
54(0x36)	TA6 bit 5	Key "number 5"
55(0x37)	TA6 bit 6	Key "number 6"
56(0x38)	TA6 bit 7	Key "number 1"
57(0x39)	TA7 bit 0	not used
58(0x3A)	TA7 bit 1	Key "number 2"
59(0x3B)	TA7 bit 2	Key "number 3"
60(0x3C)	TA7 bit 3	Key "decimal point"
61(0x3D)	TA7 bit 4	Key "+/-"
62(0x3E)	TA7 bit 5	not used
63(0x3F)	TA7 bit 6	not used
64(0x40)	TA7 bit 7	not used

1.4.24 REQUEST_VERSION (0x18)

Function:	With this telegram the PLC can request the operating panel version numbers. Both firmware and user version state (field USERDATA) can be queried. The response occurs with the telegram REPORT_VERSION.
Direction:	Master -> operating panel
User data:	Control parameter CTRL: CTRL=0: Request designation of the BIOS-version CTRL=1: Request designation of the TOS-version CTRL=2+n: Request version designation of the data structure from places (field USERDATA)
DLN	3

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x18	CTRL	not used (0x00)					

Example

0x18	0x02	0x00	0x00	0x00	0x00	0x00	0x00

The field "userdata" is called from place 0 (2+0) in the example.

1.4.25 REPORT_VERSION (0x19)

Function:	With this telegram the operating panel transmits its version number to the PLC (ASCII-string, 7 digits). Is the response to the telegram REQUEST_VERSION.
Direction:	Operating panel -> master
User data:	requested version number as ASCII-string ASCII 0 .. ASCII 6 Bxxxyzz: BIOS-version Oxxxyzz: TOS-version Dnnnnnn: Data-version (USERDATA) xxx = Program state y = Special designation zz = Special number nnnnnn = arbitrary ASCII-signs
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x19	ASCII 0	ASCII 1	ASCII 2	ASCII 3	ASCII 4	ASCII 5	ASCII 6

Example

0x19	’D’	’M’	’V’	’1’	’.’	’0’	’3’

The operating panel reports in the example that the string "MV1.00" is in the field "userdata".

1.4.26 REQUEST_CLOCK (0x1A)

Function:	Requesting time and date of the real-time clock. The operating panel responds to this request with the REPORT_CLOCK - telegram. Note: If the panel is not equipped with real-time clock, then a REPORT_CLOCK telegram is, however, generated, but this contains incidental data.
Direction:	Master -> operating panel
User data:	none
DLN	2

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x1A	not used (0x00)						

1.4.27 REQUEST_RUNTIME (0x1B)

Function:	Requesting the internal time of runtime (total switching-on time of the panel). The operating panel replies with the REPORT_RUNTIME telegram.
Direction:	Master -> operating panel
User data:	none
DLN	2

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x1B	not used (0x00)						

1.4.28 REQUEST_INTERN_VARIABLES (0x1C)

Function:	Requesting an internal variable value ("internal variables" are variables saved in the operating panel). The request is answered with the REPORT_VALUE telegram.
Direction:	Master -> operating panel
User data:	Handle: Variable number (0-65500)
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x1C	Handle low-byte	Handle high-byte	not used (0x00)				

Example

0x1C	0x04	0x01	0x00	0x00	0x00	0x00	0x00

The value of the variable with handle 260 is requested in the example (4 + 256 * 1).

1.4.29 WRITE_CLOCK (0x1D)

Function:	Setting time and date of the real-time clock.
Direction:	Master -> operating panel
User data:	Date/time to be set in the BCD format (!) TT: day MM: month JJ: year HH: hour MM: minute SS: second DW: weekday 0..6
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x1D	DD	MM	JJ	HH	MM	SS	DW

Example

0x1D	0x12	0x05	0x01	0x14	0x24	0x32	0x02

Date becomes in the example: 12.05.2001, time 14:24:32, weekday 2 adjusted.

1.4.30 REPORT_CLOCK (0x1E)

Function:	Response-telegram to the "REQUEST_CLOCK" telegram. The operating panel transmits date and time of the internal real-time clock
Direction:	Operating panel -> master
User data:	Current date and time in the BCD-format (!) TT: day MM: month JJ: year HH: hour MM: minute SS: second DW: weekday 0..6
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x1E	DD	MM	JJ	HH	MM	SS	DW

Example

0x1E	0x17	0x11	0x00	0x15	0x02	0x16	0x05

The operating panel reports in the example date 17.11.2000, time 15:02:16, weekday 5.

1.4.31 REPORT_RUNTIME (0x1F)

Function:	Response-telegram to the "REQUEST_RUNTIME" telegram. The operating panel transmits its operating hours time as number of seconds.
Direction:	Operating panel -> master
User data:	Operating time in seconds Time 0: Low-byte of the operating time Time 1: ... Time 4: High-byte of the operating time
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x1F	Time 0	Time 1	Time 2	Time 3	Time 4	not used (0x00)	

Example

0x1F	0x23	0xC0	0x12	0x00	0x00	0x00	0x00

The run-time is reported in the example with 1228835 seconds (corresponds to 341 hours, 20 minutes and 35 seconds).

1.4.32 ASCII_TELEGRAM (0x20)

Function:	The telegram type ASCII_TELEGRAM is a multiplex telegram, consisting of several telegrams for transmitting larger ASCII blocks. It is used e.g. for the log- and statistics print-out.
Direction:	Operating panel -> master
User data:	Data in the ASCII format (e.g. for print-outs) LN: Number of the ASCII-signs sent in the telegram (0..6) LN=0 identifies the end of the transfer of ASCII-signs. In this case in ASCII 0 and ASCII 1 (16 bits) the number of the transferred telegrams exclusive the telegram with LN=0 is transmitted
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x20	LN	ASCII 0 / Telegrams low-byte	ASCII 1 / Telegrams high-byte	ASCII 2	ASCII 3	ASCII 4	ASCII 5

Example

0x20	0x06	’M’	’e’	’l’	’d’	’u’	’n’
0x20	0x06	’g’	’ ’	’1’	’ ’	’g’	’e’
0x20	0x04	’h’	’t’	0x0D	0x0A	0x00	0x00
0x20	0x00	0x03	0x00	0x00	0x00	0x00	0x00

The string "message 1 leaves" with line end and carriage return is transmitted in the example.

1.4.33 REQUEST_PROTOCOL (0x21)

Function:	Request to print out the log. The operating panel answers with ASCII-telegrams from the start of the print process.
Direction:	Master -> operating panel
User data:	Specifications on the format of the print-out CTRL: Log-command 0=reset log memory 1=transmit complete log 2=only entries of an image (Image No. in NR) 3=only entries of a message (Message No. in NR) 4=only entries with a certain status 5=only entries of a variable (No. in NR) 6=abort print 7=log variable (number is in NR) OUTPUT: output medium of the log print-out 0=print is transmitted to the printer determined by ITE 1=output on the serial interface 2=output on the CAN-interface FORMAT: output format of the log print-out 0=ASCII print-out (the complete text is printed out) 1=Binary print-out (only image/mess/Var-No. is printed out) Serial: number of the log entry from which the print-out is started. If 0, then it is started from the entry printed last No: image-, message number or variable handle (only valid, if CTRL = 2,3,5)
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x21	CTRL	OUTPUT	Format	LFD low-byte	LFD high-byte	NO low-byte	NO high-byte

Example

0x21	0x02	0x02	0x00	0x40	0x00	0x07	0x00

The log print-out from Entry No. 64 for the image 7 in the ASCII-format on the CAN-interface is released in the example.

1.4.34 REQUEST_STATISTIC (0x22)

Function:	Request for printing out the statistics. The operating panel answers with ASCII-telegrams from the start of the print process.
Direction:	Master -> operating panel
User data:	Specifications on the format of the print-out CTRL: Statistics command 0=reset statistics memory 1=transmit complete statistics 2=only statistics of a group (group number in NR) 3=only statistics of an image (Image number in NR) 4=only statistics of a message (message number in NR) 5=abort print OUTPUT: output medium of the log print-out 0=print is transmitted to the printer determined with ITE 1=output on the serial interface 2=output on the CAN-interface FORMAT: output format of the log print-out 0=ASCII print-out (the complete text is printed out) No: image-, message number or variable handle (only valid, if CTRL = 2,3,4)
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x22	CTRL	OUTPUT	Format	NR Low-Byte	NR High-Byte	not used (0x00)	

Examplel

0x22	0x04	0x00	0x00	0x01	0x00	0x00	0x00

The statistics print/out for message 1 in the ASCII-format is released in the example on the printer adjusted in the project.

1.4.35 REPORT_MENU_INDEX (0x25)

Function:	Transmits the menu index to the PLC if an appropriate menu point has been selected with "ENTER".
Direction:	Operating panel -> master
User data:	Index of the menu point (16 bits)
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x25	INDEX ow-byte	INDEX high-byte	not used (0x00)				

Example

0x25	0x05	0x00	0x00	0x00	0x00	0x00	0x00

The operating panel transmits the menu index 5 in the example.

1.4.36 REPORT_OUTPUT_STATE (0x26)

Function:	Transmits the status of the report output. This telegram is requested via the REQUEST_STATUS with field mode=0x05.
Direction:	Operating panel -> master
User data:	Status of the output OUTPUT=0: output is not set OUTPUT=1: output is set
DLN	3

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x26	OUTPUT	not used (0x00)					

Example

0x26	0x01	0x00	0x00	0x00	0x00	0x00	0x00

The operating panel reports in the example the output as set.

1.4.37 REQUEST_CURSOR_POSITION (0x27)

Function:	Enquires the current cursor position in the menu
Direction:	Master -> operating panel
User data:	CTRL: Status of the transmission automatic 0: The current cursor position is transmitted with the REPORT_CURSOR_POSITION tel-egram once, then not any longer. (Automatic off) 1: The current cursor position is transmitted with the REPORT_CURSOR_POSITION telegram. Additionally a transmission automatic is switched on, which outputs with each modification of the cursor position a REPORT_CURSOR_POSITION telegram.
DLN	3

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x27	CTRL	not used (0x00)					

Example

0x27	0x01	0x00	0x00	0x00	0x00	0x00	0x00

The cursor position is enquired in the example and the transmission automatic is switched on.

1.4.38 WRITE_CURSOR_POSITION (0x28)

Function:	Positions the cursor on a menu option/nominal value if a telegram of an appropriate nominal value/menu option is available. Otherwise no modification of the cursor takes place.
Direction:	Master -> operating panel
User data:	Position of the cursor is indicated by the mode MODE = 0,1: XY-positioning (line/column) Data 0:X-position (column of the cursor- character-oriented, not graphical) DATA 1:Y-position (column of the cursor- character-oriented, not graphical) MODE = 2: Positioning on variable handle (nominal value) Data 0:Variable handle low-byte DATA 1:Variable handle high-byte MODE = 3: Positioning on menu option Data 0:Image number/menu index... low-byte (corresponding to DATA 2) DATA 1:Image number/menu index... high-byte (corresponding to DATA 2) DATA 2:Function of the menu option 0= call up an image 1= go back in the menu 2= global abortion 3= terminate nominal value entry with saving, then menu image call-up 4= terminate nominal value entry with saving, then menu image call-up 5= menu index output (transmitting index value) 6= menu index output with global abortion 7= pass on menu index to KOP **Remark:** The cursor is positioned only then if the panel is in one of the statuses "passive status", "menu selection" or "nominal value entry" and a menu option and/or a nominal value is in the image.
DLN	6

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x28	MODE	DATA 0	DATA 1	DATA 2	not used (0x00)		

Example

0x28	0x03	0x0A	0x00	0x05	0x00	0x00	0x00

The cursor is positioned in the example on a menu option with function "transmitting menu index 10".

1.4.39 EXECUTE_MENU (0x29)

Function:	The telegram carries out a menu function so as if a menu key were pressed
Direction:	Master -> operating panel
User data:	CODE: number of the simulated key
DLN	3

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x29	CODE	not used (0x00)					

Example

0x29	0x47	0x0A	0x00	0x05	0x00	0x00	0x00

The key "ENTER" is carried out in the example per CAN_telegram at the operating panel series ITS/AT 61/67.

1.4.40 REPORT_CURSOR_POSITION (0x2A)

Function:	The current cursor position is transmitted
Direction:	Operating panel -> master
User data:	Position of the cursor MODE = 0: invalid, no cursor adjustable MODE = 1: Cursor position is reported as XY-value (line/column) DATA 0: X-position (column of the cursor - character-oriented, not graphical) DATA 1: Y-position (line of the cursor - character-oriented, not graphical) MODE = 2: cursor position is reported as variable handle (is on nominal value) DATA 0: Variable handle low-byte DATA 1: Variable handle high-byte MODE = 3: cursor position is reported as menu option DATA 0: Image number/menu index... low-byte (corresponding to DATA 2) DATA 1: Image number/menu index... high-byte (corresponding to DATA 2) DATA 2: Function of the menu option 0= call up an image 1= go back in the menu 2= global abortion 3= terminate nominal value entry with saving, then menu image call-up 4= terminate nominal value entry with saving, then menu image call-up 5= menu index output (transmitting index value) 6= menu index output with global abortion 7= pass on menu index to KOP
DLN	9

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x2A	MODE	DATA 0	DATA 1	DATA 2	not used (0x00)		

Example

0x2A	0x02	0x20	0x00	0x00	0x00	0x00	0x00

The operating panel reports in the example the cursor on a nominal value with Handle 32.

1.4.41 MENU_ON (0x2B)

Function:	The telegram activates a menu image, i.e. a menu option/nominal value is selected immediately in the called up image. Thus the possibility exists to influence the menu tree from externally via the CAN-interface.
Direction:	Master -> operating panel
User data:	Number of the image 16 bits value, 1-9999, without preceding sign. If here 0xFFFF (-1) is indicated, then the menu tree is completely deleted.
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x2B	Image number low-byte	Image number **high-byte**	not used (0x00)				

Example

0x2B	0x02	0x00	0x00	0x00	0x00	0x00	0x00

The image 2 is called up in the example and the menu tree is activated immediately.

1.4.42 CAN_INIT (0x2C)

Function:	The operating panel is prompted via this telegram to initialize the CAN-interface again. Thereby possibly modified parameter of the interface (baud rate, identifier) are considered.
Direction:	Master -> operating panel
User data:	EXTENDED = 0: transmit always value 0x00 ! EXTENDED = 1...255: reserved for future functions
DLN	3

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x2C	EXTENDED	not used (0x00)					

Example

0x2C	0x00	0x00	0x00	0x00	0x00	0x00	0x00

The CAN-interface is initialized again in the example.

1.4.43 SET_KEYBOARD_LAYOUT (0x15)

Function:	Key code telegram (ITS7/AT-specific telegram). With this telegram a nominal value entry of arbitrary ASCII-signs can be made (ASCII-variables !), i.e. if a nominal value is selected, the received ASCII-sign is inserted in the nominal value. This serves for to enable an ASCII-entry via function keys.
Direction:	Master -> operating panel
User data:	KEYCODE: ASCII-Code EXTENDED KEYCODE: reserved for extensions (at the moment 0x00)
DLN	4

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7

TA=0x2D	KEYCODE	EXTENDED KEYCODE	not used (0x00)

Example

0x2D	'A'	0x00	0x00	0x00	0x00	0x00	0x00

The sign A is received in the example as entry for an ASCII-variable.

1.4.44 DRAW (0x2E)

1.4.44.1 Panels of the ITS series

Function:	function to draw graphical objects
Direction:	master -> operating panel
User data:	attribute and dimension of the graphical object: ATTRIBUTE: Bit0=0: deleting a selected area (clear pixel) Bit0=1: displaying the selected area (set pixel) Bit1=1: flashing representation of the selected area Bit3=1: inverse representation of the selected area STARTPOS: X,Y: starting position in pixel (value area 0-255) ENDPOS: X, Y: End position in pixel (value area 0-255) SHAPE: 0: rectangle (contains: horizontal and vertical lines) 1: line (is implemented on request) 2: circle (is implemented on request) 3: rhombus (is implemented on request)
DLN	8

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x2E	ATTRIBUTE	STARTPOS-X	STARTPOS-Y	ENDPOS_X	ENDPOS_Y^	SHAPE	not used (0x00)

Example (draws a rectangle of (10,0)-(130,30))

0x2E	0x01	0x0A	0x00	0x82	0x1E	0x00	0x00

1.4.44.2 Panels of the AT series

Function:	function to draw graphical objects
Direction:	master -> operating panel
Used data:	attribute and dimension of the graphical object: ATTRIBUTE: Bit0=0: deleting the selected area (clear pixel) Bit0=1: displaying the selected area (set pixel) Bit1=0: normal representation Bit1=1: flashing representation of the selected area Bit3=1: Inverse representation of the selected area Bit7=0: static area Bit7=1: dynamic area X1,Y1: point 1 X2, Y2: point 2 SHAPE: **Draw functions** Bits 0-6 0: rectangle not filled. point 1=top left corner; point 2=bottom right corner 1: line from point1 to point 2 2: circle, not filled. point 1=centre point; D4=radius 3: rhombus (is implemented on request) 4: rectangle filled. point 1=top left corner; point 2=bottom right corner 5: circle, filled. point 1=centre point; D4=radius 6: image points colour = Bit0 of D1 (black/white) 7: ellipse, filled. point1=centre point; D4=radius X; D5=radius Y Bit 7 0: indirect drawing in the memory, then image restructure 1: indirect drawing in the memory and on the screen, without image restructure Difference: Method 0 is slower, overwrites, however, only in the selected drawing level. Method 1 overwrites in the screen all areas with the next image structure (modification of a variable or similar) appears but then the correct representation. Method 1 is ofered if you draw areas in which no other elements are contained (empty image area). The output occurs substantially faster. **Push operation**s Source point1 Target point 2 Width of the area D1 Height of the area D7 D6 = 8: push screen area, static, not flashing area D6 = 9: push screen area, static, flashing area D6 = 10: push screen area, dynamic, not flashing area D6 = 11: push screen area, dynamic, flashing area
DLN	8

Telegram format:

D0	D1	D2	D3	D4	D5	D6	D7
TA=0x2E	ATTRIBUTE	X1	Y1	Y2	Y2	SHAPE	not used (0x00)

Example (draws a rectangle of (10,0)-(130,30))

0x2E	0x01	0x0A	0x00	0x82	0x1E	0x00	0x00

2 CAN-bus

The CAN-bus is a bus system that stems from the automobile industry. The abbreviation means:

Controller
Area
Network

what means more or less "network between small-control devices".
The CAN-bus is very easy to handle and also very insensitive to faults, provided the regulations for the installation are to be observed. And it is low in costs. All these reasons contributed to using the CAN-bus in the operating panels.

2.1 Wiring

The CAN-bus is a bus system - as its name already says. In this system, rules apply which are to be adhered to. If this is not the case, then it cannot be forecasted whether the bus works correctly or not. This goes so far that individual devices work, however others not. But the CAN-bus has proven itself in practice to be very uncomplicated if the following regulations are adhered to:

2.1.1 Bus structure (topology)

The CAN-bus has to be set up in a bus structure whose both ends are provided each with a terminal resistance of 120 ohm.
If a star topology is required, then repeaters have to be installed which electrically decouple long stub lines from other bus segments. The relevant structure regulations for CAN-bus systems apply in this connection.

2.1.1.1 Line connections

It is recommendable to use the lines CAN-L, CAN-H and the screening CAN-SHLD as minimum wiring. Put on the screen on both sides. The lines CAN-L and CAN-H may not be crossed at the devices but have to be wired on a strait one to one basis.
It is recommended to lead these lines onto a twisted pair of wires.
You can connect the line CAN-GND still to all devices for to increase the transfer security. If you have additionally twisted wire pairs in your bus cable, please connect then both wires of a twisted pair to CAN-GND.

2.1.1.2 Bus cable

Also unscreened twisted lines can be used for test purposes in a trouble-free environment for laboratory structures with short lines. You should use a cable according to DIN/ISO 11898 in your system. The specific features of the cable can be obtained from the standard. Contact a cable manufacturer in the case of doubt who produces bus cables especially for CAN.

2.1.1.3 Line lengths

The following table gives you a guide-value for the maximum cable length of the CAN-bus:

Baud rate	Resistance	Wire cross-section	Max. cable length
10 kBit/s	< 18 mohm/m	1.00 mm2	2,000 m
20 kBit/s	< 25 mohm/m	0.80 mm2	1,000 m
50 kBit/s	< 30 mohm/m	0.65 mm2	700 m
100 kBit/s	< 40 mohm/m	0.50 mm2	500 m
125 kBit/s	< 44 mohm/m	0.45 mm2	400 m
250 kBit/s	< 50 mohm/m	0.40 mm2	200 m
500 kBit/s	< 60 mohm/m	0.34 mm2	100 m
1 MBit/s	< 70 mohm/m	0.25 mm2	40 m

With larger line lengths you should use a cable with less specific resistance - thus a thicker cable. If you should plan a system which exceeds these indicated lengths, then you can also plan 2 CAN-segments and connect these via a repeater. The repeater provides that all news is present on both segments, however, that the segments are electrically decoupled from each other.

2.1.2 Terminating

A resistance with 120 ohm must be wired on both ends of the bus between the lines CAN-L and CAN-H. The 4-pole screw-terminal strip for the CAN/bus serves at the CAN-module for this, simply insert the resistance, and tighten the screws. A switchable resistance can be found at the operating panel. A slide switch is located beside the DB-9 CAN-cable connector. If you push this to the position ON, then the internally-installed bus termination is activated.

2.1.3 Addressing

Each panel must have a single number (or a single identifier) in the bus - thus do not place a number twice. The CAN-modules have DIP-switches to adjust the addresses. The operating panels are adjusted to an address via the editor.

2.2 Logs in general

There are different log specifications also for the CAN-bus. Although the data transport is with all logs identical via the CAN-controller, the placing of the identifier (=address) and the telegram content of the different logs is dealt with separately.
The operating panels govern 3 logs in total.

2.2.1 SELECAN-log

Initially developed by the company SELECTRON, this log has achieved only a little distribution. It is based on a master (host), which takes over the configuration and monitoring of the connected slaves (modules). It must be viewed as proprietary log due to its little distribution.

2.2.1.1 Structure of the identifier

The 11-bits CAN-identifier contains 3 areas for the identification in total; 3 bits for the news priority, 5 bits node address (max. 32 devices result) and 3 bits-identification for the message direction. The user can determine the node address, whereby the master should always receive the node address 0 (because of the news priority).The SELECAN-log is only necessary if the ITS6 is to be operated together with the modules of the series GCM. The data length (DLC) amounts always to 8, the RTR-Bit (R) always to 0. The identifier-format is therefore as follows:

15-13			12-8					7-5			4	3-0			
Prio			Node address					Spec.				DLC			
x	x	x	x	x	x	x	x	x	x	x	0	1	0	0	0
Master transmit to module								1	1	1	0	x	x	x	x
Digital module to master								0	1	0	0	x	x	x	x
Analogue module to master								0	1	1	0	x	x	x	x

2.2.1.2 Identifier table in the SELECAN-log

The following are needed as identifier (the bits 15 to 5 = 11 bits are considered):

Ad	P0	P1	P2	P3	P4	P5	P6	P7
0	0-7	256-263	512-519	768-775	1024-1031	1280-1287	1536-1543	1792-1799
1	8-15	264-271	520-527	776-783	1032-1039	1288-1295	1544-1551	1800-1807
2	16-23	272-279	528-535	784-791	1040-1047	1296-1303	1552-1559	1808-1815
3	24-31	280-287	536-543	792-799	1048-1055	1304-1311	1560-1567	1816-1823
4	32-39	288-295	544-551	800-807	1056-1063	1312-1319	1568-1575	1824-1831
5	40-47	296-303	552-559	808-815	1064-1071	1320-1327	1576-1583	1832-1839
6	48-55	304-311	560-567	816-823	1072-1079	1328-1335	1584-1591	1840-1847
7	56-63	312-319	568-575	824-831	1080-1087	1336-1343	1592-1599	1848-1855
8	64-71	320-327	576-583	832-839	1088-1095	1344-1351	1600-1607	1856-1863
9	72-79	328-335	584-591	840-847	1096-1103	1352-1359	1608-1615	1864-1871
10	80-87	336-343	592-599	848-855	1104-1111	1360-1367	1616-1623	1872-1879
11	88-95	344-351	600-607	856-863	1112-1119	1368-1375	1624-1631	1880-1887
12	96-103	352-359	608-615	864-871	1120-1127	1376-1383	1632-1639	1888-1895
13	104-111	360-367	616-623	872-879	1128-1135	1384-1391	1640-1647	1896-1903
14	112-119	368-375	624-631	880-887	1136-1143	1392-1399	1648-1655	1904-1911
15	120-127	376-383	632-639	888-895	1144-1151	1400-1407	1656-1663	1912-1919
16	128-135	384-391	640-647	896-903	1152-1159	1408-1415	1664-1671	1920-1927
17	136-143	392-399	648-655	904-911	1160-1167	1416-1423	1672-1679	1928-1935
18	144-151	400-407	656-663	912-919	1168-1175	1424-1431	1680-1687	1936-1943
19	152-159	408-415	664-671	920-927	1176-1183	1432-1439	1688-1695	1944-1951
20	160-167	416-423	672-679	928-935	1184-1191	1440-1447	1696-1703	1952-1959
21	168-175	424-431	680-687	936-943	1192-1199	1448-1455	1704-1711	1960-1967
22	176-183	432-439	688-695	944-951	1200-1207	1456-1463	1712-1719	1968-1975
23	184-191	440-447	696-703	952-959	1208-1215	1464-1471	1720-1727	1976-1983
24	192-199	448-455	704-711	960-967	1216-1223	1472-1479	1728-1735	1984-1991
25	200-207	456-463	712-719	968-975	1224-1231	1480-1487	1736-1743	1992-1999
26	208-215	464-471	720-727	976-983	1232-1239	1488-1495	1744-1751	2000-2007
27	216-223	472-479	728-735	984-991	1240-1247	1496-1503	1752-1759	2008-2015
28	224-231	480-487	736-743	992-999	1248-1255	1504-1511	1760-1767	2016-2023
29	232-239	488-495	744-751	1000-1007	1256-1263	1512-1519	1768-1775	2024-2031
30	240-247	496-503	752-759	1008-1015	1264-1271	1520-1527	1776-1783	2032-2039
31	248-255	504-511	760-767	1016-1023	1272-1279	1528-1535	1784-1791	2040-2047

P0: Service, Broadcast, Host to Mod., Command
P1: Service, Host to Module, Command
P2: Service, Broadcast, Module to Host, Status
P3: Service, Module to Host, Status
P4: Data, Host to Module, Interrupt-Data
P5: Data, Host to Module
P6: Data, Module to Host, Interrupt-Data
P7: Data, Module to Host
(P0-P7: 3 bits for the news priority)

2.2.2 Telegram contents

The telegram contents are exactly specified. You receive a special manual upon request via the SELECAN-log.

2.2.3 Operating panel to SELECAN-PLC

The operating panels use only the data-channels of the SELECAN-log for the communication with the control: that means that the bit No. 10 is always set to 1. All functions can be operated via these data-channels.Thereby the telegram-length (size of the data area) is always determined to 8 bytes (maximum size). The address of the operating panel ("panel number") is adjusted in the ITE. (see chapter 10)

2.2.3.1 Control --> ITS

The identifier has the following format (11-bits):
Bit-No.:

15	14	13	12	11	10	9	8	7	6	5
1	0	0	Addr 4	Addr 3	Addr 2	Addr 1	Addr 0	1	1	1

The address works out to:
Identifier=1031 + 8 x panel no.

2.2.3.2 ITS-6000 --> control

The identifier has the following format (11-bits):

15	14	13	12	11	10	9	8	7	6	5
1	1	1	Addr 4	Addr 3	Addr 2	Addr 1	Addr 0	0	1	0

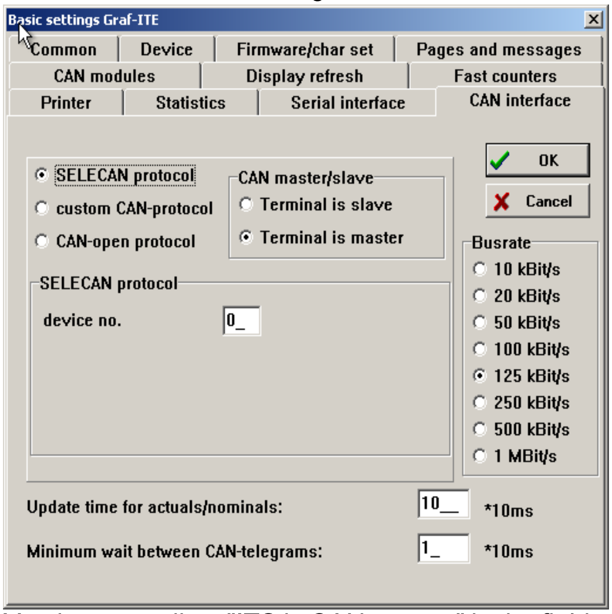
Here we receive:
Identifier=1794 + 8 x panel no.

2.2.4 Using GCM-modules

The operating panels can monitor and analyze independently CAN-modules of the series GCM. This enables in many cases a low-cost solution for to installing e.g. error-monitoring in plants or buildings. The panel can work then as independent system without external control.
You can create the control logic (PLC-function) as contact plan or in C. Simple functions like "assign input to message" or "assign counter to input" can be created via the module configuration-tool without having to write a single line program.
Only if logic functions like "if input 1 and input 2, then image 27" are necessary, then there's no alternative for a control program.

2.2.4.1 Prerequisites

At the moment the SELECAN-log is necessary for an independent system with operating panel and CAN-modules. This has to be activated in the mask "panels"/"parameterizing", register card "CAN-adjustments". Thereby the master-operating panel (there can be also several operating panels switched in the CAN-bus) has to be adjusted to the panel address 0 and has to be configurated as master:



You have to adjust "ITS is CAN-master" in the field "CAN configuration".
You can select the baud rate optionally.

2.2.4.2 Master-slave-configurations

If you want to use a second operating panel for the observation of processes, then you can use at the slave the same project as with the master. Adjust simply "operating panel is slave", and an unused panel number is unequal to 0.
If you have to control outputs from both operating stations, then the coordination must be done at the master. That means if you want to switch an output from the slave, then you should send a telegram to the master (with the internal control program), in which you e.g. send a variable which receives 0 for output off, 1 for output on. You only must query for the value of the variable in the master-control program and control the output correspondingly. This sounds much more complicate than it is; there is the function „transmit CAN-telegram“ in the control program.If you use this and structure thereby the telegram in such a way as described in the telegram description "SET VARIABLE VALUE", then you can switch up to 32 outputs via a single variable. Each variable has up to 32 bits and each bit can thereby be queried individually (Note: copy variable into a pointer double-word @MD, access then with the pointers @Mx.y.).

2.2.4.3 Adjusting the CAN-modules

The CAN-modules have a series of DIP-switches. You can adjust with these the baud rate and the node address/module number.
See the following tables for this:

Baud rate	DIP 6	DIP 7	DIP 8
10 kBit/s	off	off	off
20 kBit/s	on	off	off
50 kBit/s	off	on	off
100 kBit/s	on	on	off
125 kBit/s	off	off	on
250 kBit/s	on	off	on
500 kBit/s	off	on	on
1 MBit/s*	on*	on*	on*

* This baud rate setting is not possible at the moment. This is not supported by the bus-couplers of the modules. Please contact us if you are in need of this bus rate.

Place the module addresses as follows:

No.	DIP 1	DIP 2	DIP 3	DIP 4	DIP 5
1	on	off	off	off	off
2	off	on	off	off	off
3	on	on	off	off	off
4	off	off	on	off	off
5	on	off	on	off	off
6	off	on	on	off	off
7	on	on	on	off	off
8	off	off	off	on	off
9	on	off	off	on	off
10	off	on	off	on	off
11	on	on	off	on	off
12	off	off	on	on	off
13	on	off	on	on	off
14	off	on	on	on	off
15	on	on	on	on	off
16	off	off	off	off	on
17	on	off	off	off	on
18	off	on	off	off	on
19	on	on	off	off	on
20	off	off	on	off	on
21	on	off	on	off	on
22	off	on	on	off	on
23	on	on	on	off	on
24	off	off	off	on	on
25	on	off	off	on	on
26	off	on	off	on	on
27	on	on	off	on	on
28	off	off	on	on	on
29	on	off	on	on	on
30	off	on	on	on	on
31	on	on	on	on	on
0	Reserved for the master (operating panel)				

2.2.4.4 Assignment of panel addresses/ identifier

If you want to use several operating panels, then you must exchange possibly data between these panels. This can be achieved with the contact plan KOP via the function "send CAN telegram". But you must know which identifier you have to enter. The following table gives an assignment from the panel address to the receiving-identifier of the data channel, which you have to adjust in KOP:

Panel address	Receiving-identifier
0	1.031
1	1.047
2	1.055
3	1.063
4	1.071
5	1.079
6	1.087
7	1.095
8	1.103
9	1.111
10	1.119
11	1.127
12	1.135
13	1.143
14	1.151
15	1.159
16	1.167
17	1.175
18	1.183
19	1.191
20	1.199
21	1.207
22	1.215
23	1.223
24	1.231
25	1.239
26	1.247
27	1.255
28	1.263
29	1.271
30	1.279
31	1.287

As already said: These identifiers indicate mainly the data-receiving channel of the appropriate panel. There are also used further identifiers e.g. for status- and configuration news. These can be found in a table further above.

2.2.4.5 Actuate modules from the internal control program

You can find in the Appendix notes for the numbering of the inputs, which deviate from the numberings specified here.
KOP can actuate modules with the addresses 1-8 directly via the KOP-variables "@DIx.y" (Digital In), "@DOx.y" (Digital Out), "@AIx.y" (Analog In) and "@AOx.y" (Analog Out); x stands for the module address and y for the in input/output number.
You must generally decide whether you want to

control output functions of the modules via the control program or the outputs of the variables via the CAN-configuration-tool. You cannot go at the same time via @DO and a variable to an output. This is possible with inputs.
At modules with an address larger than 8, you have to do everything via internal variables on which the internal control program can also access.

2.2.5 Free CAN-log

The operating panel with the free CAN-log can be adjusted to any identifier both for transmitting and receiving. For the receive up to 8 receiving-identifiers are adjustable.This becomes interesting if several controls or operating panels has to communicate with each other. As this log is the most flexible, it is also used more frequently.
We do not find a identifier table since everything can be adjusted freely.

2.2.5.1 Structure of the identifier

The data length of the telegrams (when transmitting and receiving via the TOS, do not send KOP) amounts always to 8, and the RTR-Bit (R) is not considered.
The identifier-format is therefore:

15-5											4	3-0			
Identifier											R	DLC			
x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	0

2.2.5.2 Telegram contents

The telegram content is described in the Appendix. The content of the telegram is determined by the "Multiplexer-Byte D0", with whose value the following telegram content D1-D7 is analyzed.
Both transmitted and also received telegrams must keep to this telegram format.

2.2.6 CANopen-log

CANopen is a log which consists of several levels. It determines how the identifier of the devices should look like; and it divides data transfers into the 3 areas - network management (NMT), process data transfer (PDO) and parameterize data transfer (SDO). Panels which keep to these specifications can principally be operated in a CAN-open-network.
Beyond that, CANopen defines so-called "panel profiles". In these profiles, a basic standard for parameterize data and process data is defined each for the same panel types (e.g. revolution transmitter or frequency converter). Panels of manufacturers who keep to these panel profiles can be controlled in the same way (mostly only the basic functions, but nevertheless!). How the data transfer takes place is determined again by the CANopen-log.
CANopen enables the operation of 128 devices.

2.2.6.1 Basic behaviour as SLAVE

The operating panel behaves as Slave like a digital I/O-module according to the standard DS-401. There were no specifications (profile) for operating panels at the time of the driver development. The operating panel delivers or receives CANOpen-log as process data (PDO). The telegram format can be found in the Appendix (one could designate the transfer type as "multiplexed PDO"). Additionally the SDO- or NMT-services are implemented.

2.2.6.2 Basic behaviour as MASTER

As master an enlarged functionality is available: The operating panel copies independently PDOs received into variables with appropriate handle, if present. An analyzing of PDO-data is then easily possible via the control program.

But we do not go fully into details. There is a chapter totally on this via CAN/open in the Appendix.

2.2.6.3 Identifier table of CANopen

The identifier can be placed freely in CANopen via SDO-services. It is, however, recommendable to place the identifier according to the so-called „Predefined connection set" in order to receive a basic standard. CANopen works out these identifiers via simple formulas on the basis of the so-called node number „Node-ID" (all indications in decimal representation):

Type	ID =	ID-area
Network management NMT	Node-ID	0-127
Emergency EMCY	Node-ID+128	128-255
Transmitting process data (TX-PDO)	Node-ID+384	384-511
Receiving process data (RX-PDO)	Node-ID+512	512-639
Transmitting system data (TX-SDO)	Node-ID+1408	1408-1535
Receiving system data (RX-SDO)	Node-ID+1536	1536-1663
Node monitoring (GUARD)	Node-ID+1792	1792-1919

2.2.7 Mixing of logs

If you want to "drive" several logs on the same CAN-bus, then you must observe only that identifiers are not placed twice. Besides, the masters must be able to keep respective identifiers away from the monitoring. Otherwise a message could be analyzed in the wrong log and thus lead to malfunctions.

3 CAN-Open driver

CANopen is a log definition on CAN-BUS-Hardware. It concerns thereby the specification 4.0 of CANopen, which is also based on the programming of the operating panels.
The present manual is not meant to integrate the CANopen specification but to represent the features of the operating panel as CANopen device.

In the present implementation the following features are supported by CANopen.

MASTER:
Minimum network management
Transmitting and receiving of SDOs
Transmitting and receiving of PDOs

SLAVE:
Minimum boot-up behaviour
Predefined Connection Set
No PDO-mapping
Boot-up Node-Guard Frame

This documentation uses the notation employed in the CANopen literature for the telegram representation etc.

3.1 Requests

3.1.1 Operating system (TOS)

Since the CANopen functionality could not be integrated any longer into the standard operating system, its own operating system (TOS) is supplied with CANopen. This TOS has the same functionality like the standard TOS but can run only the CANopen log on the CAN-bus.

Concretely:
Standard-TOS operating panel: IO0xxSxx.hex
Standard-TOS ITS7000: IO1xxSxx.hex
CANopen-TOS operating panel: IO0xxAxx.hex
CANopen-TOS ITS7000: IO1xxAxx.hex
(x varies)

3.1.2 Firmware (BIOS)

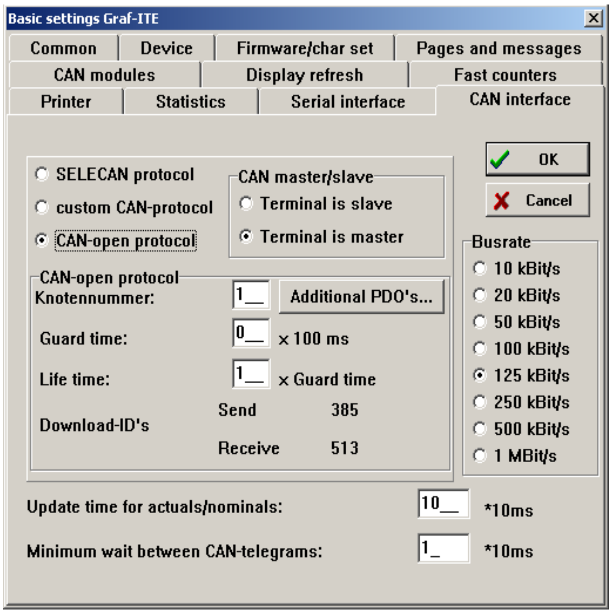
A special BIOS is not necessary. That means the CANopen software can be used also on other panels.

3.1.3 Project planning software (Editor ITE)

In order to be able to parameterize CANopen, the project planning software ITE6D16 or a newer version is necessary. This project planning software considers already the additional TOS versions for CANopen.

3.1.4 Settings in the ITE

The editor has been supplemented by the CAN-open setting possibilities. You can find these if you click on "panels"/"parameterize" in the register card "CAN settings":



These setting fields appear if you select "CAN-Open". The meaning of the fields is:

3.1.5 Field ITS-CAN configuration

Here you adjust whether the operating panel is to be used as CAN-Open Master or as Slave. Details regarding the difference "Master - Slave" can be found further back in this documentation.

3.1.6 Field node number

In this field you adjust which node number (Node-ID) the operating panel is to have. Adjust the SDO- and PDO identifier (COB-IDs) via the node number just as they are suggested by the "predefined connection set" of CANopen (from the view of the operating panel):

Emergency object	=	128 + Node-ID
TX-PDO-ID	=	384 + Node-ID
RX-PDO-ID	=	512 + Node-ID
TX-SDO-ID	=	1408 + Node-ID
RX-SDO-ID	=	1536 + Node-ID
Node Guard	=	1792 + Node-ID

3.1.7 Field guard-time

This field is only of importance for the slave. Here the node guard-time is adjusted (Node Guard-time) in multiples of 100 ms.

3.1.8 Field time window

This field is only of importance for the slave.
Here the "life cycle" (Lifetime) is adjusted. The meaning of the parameter is how often the time may expire that is adjusted under "guard-time" before the operating panel recognises and signals an error.

3.1.9 Display fields Download-ID's

In these fields the identifiers appear under which the operating panel carries out a project-download.

You need these adjustments in order to be able to carry out a transfer of the project from the editor (PC) to the operating panel. Enter in the mask "panels"/"interface" the download-ID "receiving" as transmitting identifier that is displayed here.
By the way: these are the identifiers (COB-ID's) under which the PDO-transfers take place.

3.1.10 Field interval time for call...

This field is only of importance for the slave.
Here you determine how often the operating panel demands the external variable again. See manual - Operating panels: Operating and watching.

3.1.11 Field minimum waiting time

Here you determine how much time is to pass at least between two CAN-telegrams sent by the operating panel. Thus you can restrict here the bus load for the operating panel.
This setting is ignored by the NMT-services, since time-outs could here result.

3.2 MASTER-implementation

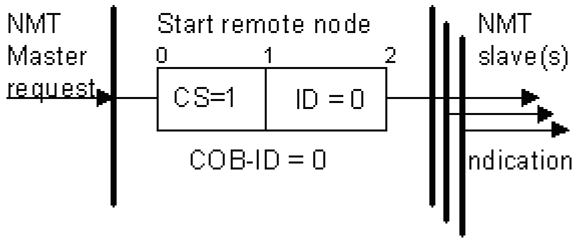
It was the aim of the MASTER-implementation to control simple CANopen-devices via the operating panel and to parameterize, if necessary. Thereby, some mechanisms have been implemented which have not to be realised only via the control program as KOP or in C.
The master is activated in the register card "CAN settings" (under "panels"/"parameterize") , field "ITS CAN configuration".

3.2.1 Minimum network management

The network management of the master contains the automatic start-up of nodes. A node-guarding (Node-Guarding) is not implemented. This has to be realised per KOP.

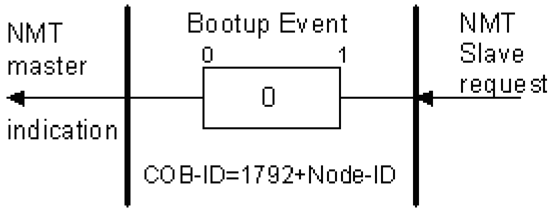
3.2.1.1 Master Boot-up

The master transmits when starting a "BROADCAST START REMOTE NODE" to all nodes:

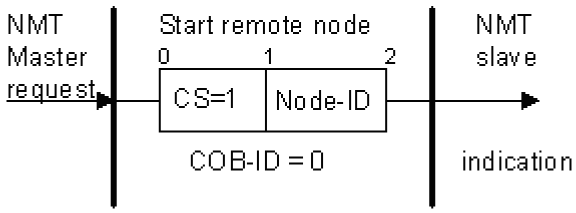


3.2.1.2 Slave Boot-up (Version 4.0)

Version 4.0 of CANopen requests that a slave sends a telegram "Boot-up Event" with DL=1 and D0=0 with the transition of "initialisation" to "pre-operational" :



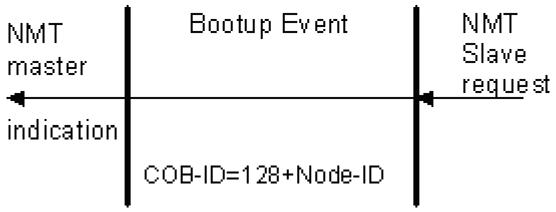
The master answers this telegram with a "Start remote node" particularly for this node:



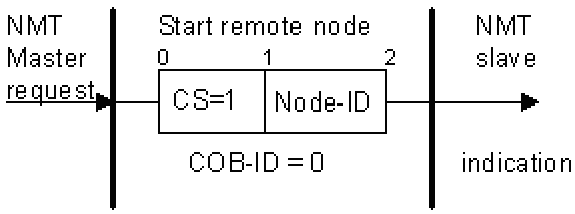
Thus nodes which have had a failure or log on again are automatically taken into operation.

3.2.1.3 Slave Boot-up (RFC to version 3.0)

In this RFC it has been requested that a slave transmits an "Emergency" without content during the transition from "initialisation" to "pre-operational" :



The master replies also to this telegram with a "Start Remote-Node":

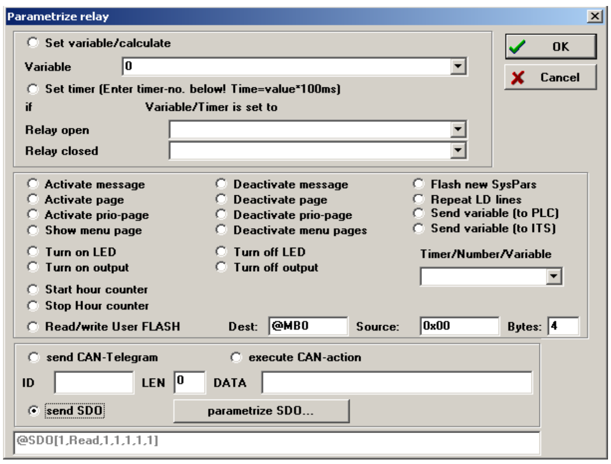


3.2.2 Transmitting and receiving of SDOs

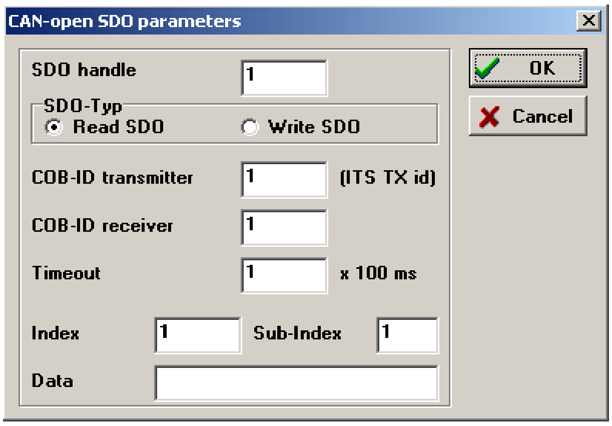
The master cannot carry out automatically transmitting and receiving of SDOs. Each application has its own data exchange.
The master monitors, however, the correct exchange of SDO-data and displays errors.
The exchange of SDO-data is realised via the control program in KOP or C.

3.2.2.1 Transmitting/requesting SDO data

The KOP starts a SCO-transmission or a SDO-query with the relay function "transmitting SDO".
Practically such a function is always started because of a condition - see further underneath for this the application example .
In the following mask the function is selected:



If you press the button "parameterize SDO...", then you receive the setting possibilities for the SDO-parameter:

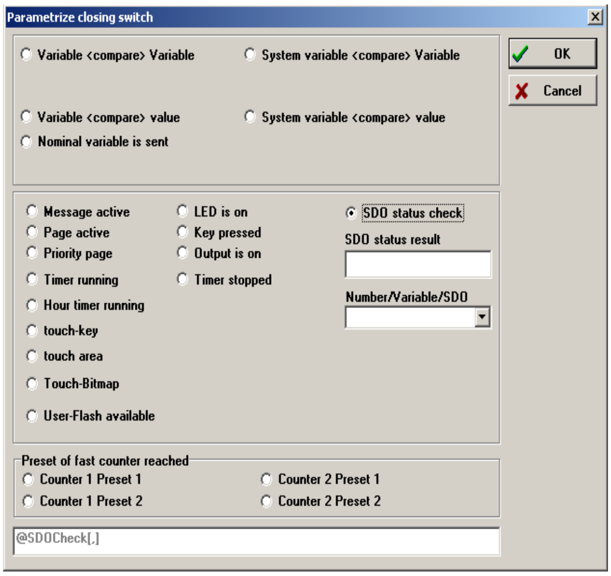


Field SDO-number
You have to enter in this field a number under which you want to make later the status query for this SDO. The operating panel can process max. 15 SDOs at the same time.
Avoid that different SDOs are provided with the same SDO number. This could lead to erroneous information.
The panel needs the SDO number only internally. This information does not appear on the CAN-bus.
Field SDO type
Here you can select whether you want to read the SDO data from another node (Read SDO) or whether you want to send SDO data to another node (Write SDO).
Field COB-ID transmitter
Enter in this field with which COB-ID the operating panel is to transmit this SDO. The entry is decimal but a variable can be also entered (as shown in the example).
Field COB-ID receiver
Here you enter under which COB-ID the reply/confirmation of the receiver is to be expected. Also here a variable can be entered.
Field Time-out
Indicate here how long the operating panel has to wait maximum for a reply/confirmation.
Field Index
Enter here the index of the SDO. The entry is decimal according to standard, but can be also indicated hexadecimal by placing "0x" in front. Example: 0x1000. Also variables are here possible.
Field Subindex
contains the subindex of the SDOs. Entries just as with the index.
Field data
You have to fill in this field only if you want to transmit SDO data to a node. The entry is again decimal, hexadecimal ("0x") or a variable.

3.2.2.2 Query SDO-answer

IMPORTANT !!!
THE STANDARD CLOSING-CONTACT WITHOUT ABORTION HAS ALWAYS TO BE USED FOR THE QUERY OF SDO REPLIES IN KOP !!!

The status query of a SDO transmission or -request occurs always with a standard closing-contact.
The branch behind the closing-contact is carried with current if a successful reply has arrived. The run occurs only once, afterwards the SDO is taken out of the internal guarding.
If a time-out or a SDO error occurs, then the branch behind the closing contact is run through without current. The run occurs also only once, afterwards the SDO is taken out of the internal guarding.
With this method the SDO data exchange can be guarded.
The parameterization of the closing-contact is carried out in the closing contact mask:

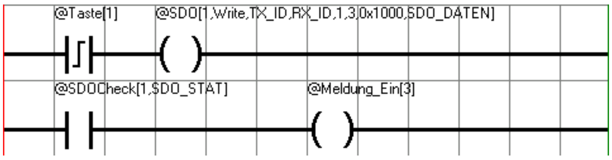


Field "number/variable/SDO"
In the field "number/variable/SDO" you enter the SDO number which you have used in the mask "SDO parameterize". Thus the KOP has the assignment which SDO it has to be guarded.
Field "SDO status result"
Enter in the field "SDO status result" where KOP has to file the result or the data. This can be a pointer word or a variable.

As already said: Run of the branch with current: SDO is ok (reply is then in the variable) or run of the branch without current: SDO error. In the variable an expanded error code is then to be found, which is explained in detail in the CANopen documentation.

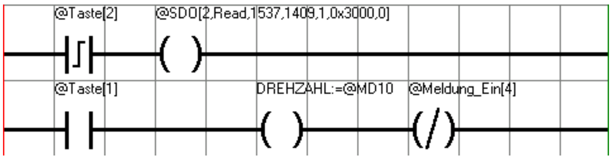
3.2.3 Example:Transmit SDO

In the following example data are transmitted for a SDO if the key 1 is pressed. The guarding activates the message 3 if a time-out has been achieved or the SDO was invalid.



3.2.4 Example:Read SDO

In the following example the requested SDO data are saved in the variable SPEED or message 4 is called up in case of error:



So reading and writing of SDOs is possible with less programming effort.

3.2.5 Transmitting and receiving of PDOs

The transmission of PDOs is substantially simpler than the SDO handling, since here replies have to be considered .

3.2.5.1 Transmitting PDOs

This function is already fulfilled for a long time by the relay function "transmitting CAN telegram". The following errors are relevant:

Field "ID"
Enter here the COB-ID under which the PDO is to be transmitted. The entry can be decimal, hexadecimal ("0x") or a variable.
Field "LEN"
Enter here the length of the PDOs (how many bytes are transmitted?).
Field "DATA"
If the field "LEN" is unequal to 0, then you can firmly enter here the data to be transmitted (2 numbers each result 1 byte: 05010206 results D0=05, D1=01, D2=02, D3=06) or transmit data via pointer: @MB10 results D0=MB10, D1=MB11 etc.

3.2.5.2 Receiving PDOs

You only need to set up the appropriate internal variables for the reception of PDOs:

Variable handle for D0-D3 = COB-ID PDO
Variable handle for D4-D7 = COB-ID PDO + 2000

The master files the received PDO data automatically in these variables if it receives data under the appropriate COB-ID.

CAUTION: The variable handles 384 to 511 and 2384 to 2511 should not be used for other purposes than for the PDO reception.

Controlling the master
If the master receives PDO (385 + Node-ID) on its own ID, then it analyses the PDO-telegram according to the description of the free CAN-driver.

3.2.6 Project download

In order to load a project into the operating panel with the ITE, the transmission identifier can be adjusted under "panels"/"interface" on 512 + Node-ID (to be found under "panels"/"parameterize", register card "CAN settings") ("select free CAN"). Then the operating panel "hears" the editor and starts with the download.

3.2.7 Object directory

The master has currently no object directory.

3.2.8 Status transitions

The MASTER goes always automatically to the status "operational". There are no other statuses.

3.3 SLAVE implementation

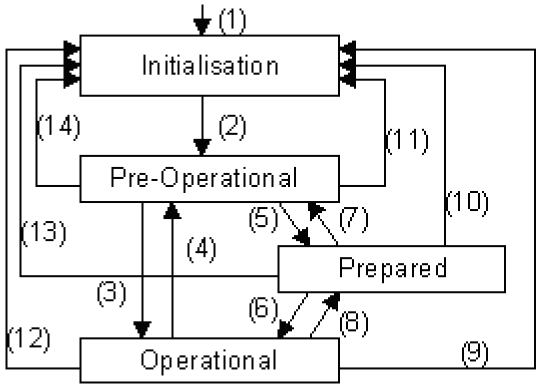
The current SLAVE implementation is restricted to the emulation of an I/O-device. Thus the telegrams are analysed as outlet data in such a way as they are also defined in the free CAN-driver. The status telegrams are delivered in the same way simply as input PDO.

Only the control of the status transitions and the object directory are added.

The slave is activated in the register card "CAN settings" (under "panels"/"parameterize"), field "ITS CAN configuration".

3.3.1 Status diagram

The slave uses the "minimum-bootup behaviour" according to the CANopen Draft standard 301 version 4.0:



3.3.1.1 Status transition table

(1)	The status "initialisation" is automatically achieved when switching on.
(2)	After initialisation is effected, the status "pre-operational " is automatically achieved and the telegram "boot-up event" is transmitted.
(3), (6)	Request "Start_Remote_Node"
(4), (7)	Request "Enter_PRE-OPERATIONAL_State"
(5), (8)	Request "Stop_Remote_Node"
(9),(10), (11)	Request "Reset_Node"
(12),(13), (14)	Request "Reset_Communication"

3.3.1.2 Description of the statuses

The following conditions apply in the individual statuses:

Status initialisation
The operating panel initialises all internal buffers and interfaces. No communication takes place during this status.

Status pre-operational
In this status the NMT-services as well as the SDO-transfer are active. PDO-services are not specified.
Exception: since a download must be possible in each status of the operating panel, the PDO with D0=0x10 (request memory read) is still executed. This PDO is used by the editor for the purpose of initialising the download.
Later versions of the editor will place a "Start_Remote_Node" request in front in order to achieve the status "operational", before the "request memory read"-PDO is transmitted.

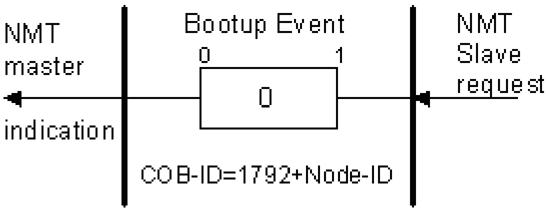
Status operational
In this status all services are active.

Status prepared
If the status "prepared" is requested, then only the network-management (NMT) is active. Neither PDO nor SDO transfers are possible.
The PDO-exception applies here also like in the status "pre-operational"

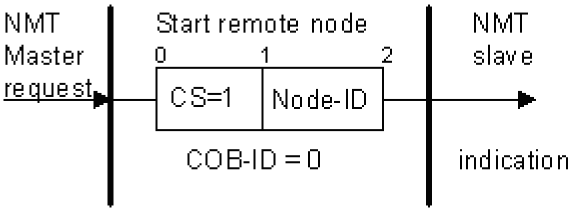
3.3.1.3 Description of the telegram communi cation

The following telegrams are responsible status change. The operating panel accepts as Node-ID its own or the 0 (broadcast = to all)

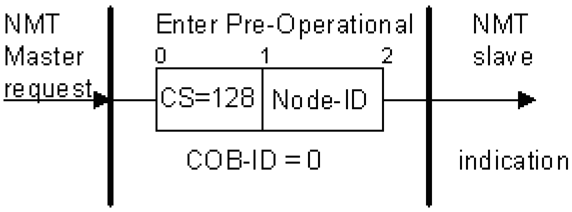
Boot-up_Event



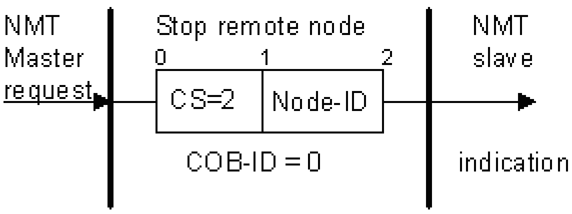
Start_Remote_Node



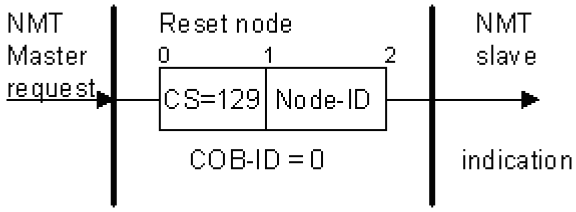
Enter_PRE-OPERATIONAL_State



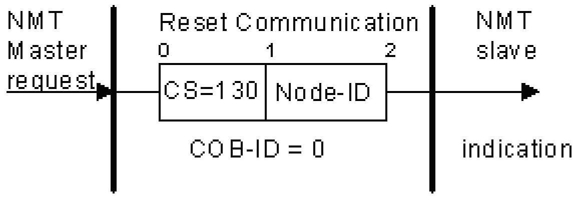
Stop_Remote_Node



Reset_Node



Reset_Communication



3.3.2 Object directory

The object directory of the slave is oriented to the DSP-401 for I/O-modules. Currently the object directory is only readable but not writeable. The following objects are defined:

3.3.2.1 Object 1000h: Panel type

Contains information over the panel type.
Object description:

INDEX	1000h
Name	Panel type
Object type	Individual value (VAR)
Data type	32 bits without preceding sign (UNSIGNED32).
Category	Mandatory

Object entry

Access	Read only
PDO-mapping	no
Value	Profile number and additional information: D4/D5 = 401d = 191h D6/D7 = 3 (inputs and outputs)

3.3.2.2 Object 1001h: Error index

Contains information on errors that have occurred
Object description:

INDEX	1001h
Name	Error index
Object type	Individual value (VAR)
Data type	8 bits without preceding sign (UNSIGNED8).

Category	Mandatory

Object entry:

Access	Read only
PDO-mapping	no
Value	Bit-coded according to DSP 301 Bit 0: Generic error (mandatory) Bit 1: Current (not used) Bit 2: Tension (n.u.) Bit 3: Temperature (n.u.) Bit 4: Communication error Bit 5: not used Bit 6: not used Bit 7: not used

3.3.2.3 Object 1004h: Number of the PDOs

This object contains information on how many PDOs are intended for the operating panel.
This object is omitted from DS 301 version 4, but is, however, still contained for reasons of compatability.

Object description:

INDEX	1004h
Name	Number of the PDOs
Object type	Field (ARRAY)
Data type	32 bits without preceding sign (UNSIGNED32). D4/D5: Number Transmit PDOs D6/D7: Number Receive PDOs
Category	Optional (V3.0)

Object entries:

Subindex	0
Description	Number of the supported PDOs
Category	Optional (V3.0)
Access	Read only
PDO-mapping	no
Value	D4/D5: 1 D6/D7: 1

Subindex	1
Description	Number of the synchronous PDOs
Category	Optional (V3.0)
Access	Read only
PDO-mapping	no
Value	D4/D5: 0D6/D7: 0

Subindex	2
Description	Number of the asynchronous PDOs
Category	Optional (V3.0)
Access	Read only
PDO-mapping	no
Value	D4/D5: 1 D6/D7: 1

3.3.2.4 Object 1008h: Manufacturer: panel name

In this object a 4-byte abbreviation is to be found for the operating panel.

Object description:

INDEX	1008h
Name	Manufacturer: panel name
Object type	Individual value (VAR)
Data type	String (4 bytes) (visible string)
Category	Optional

Object entry:

Access	Read only
PDO-mapping	no
Value	'IT61' = ITS 6100 series 'IT62' = ITS 6200 series 'IT71' = ITS 7100 series 'IT72' = ITS 7200 series

3.3.2.5 Object 1009h: Hardware version

This object contains the number of the firmware in the panel. Example: the BIOS IB055SE0 delivers ' 55'.

Object description:

INDEX	1009h
Name	Hardware version
Object type	Individual value (VAR)
Data type	String (4 bytes) (visible string)
Category	Optional

Object entry:

Access	Read only
PDO-mapping	no
Value	Dependent on the panel bios

3.3.2.6 Object 100Ah: Software version

This object contains the number of the operating system (TOS) which is booted up in the panel. Example: the TOS IO164A00 delivers ' 164'.

Object description:

INDEX	100Ah
Name	Software version
Object type	Individual value (VAR)
Data type	String (4 bytes) (visible string)
Category	Optional

Object entry:

Access	Read only
PDO-mapping	no
Value	Dependent on TOS

3.3.2.7 Object 100Bh: Node address

This object contains the parameterized node address for the operating panel.
This object is omitted from DS 301 version 4, but is, however, still contained for reasons of compatibility.

Object description:

INDEX	100Bh
Name	Node address
Object type	Individual value (VAR)
Data type	32 bits without preceding sign (UNSIGNED32).
Category	Optional

Object entry:

Access	Read only
PDO-mapping	no
Value	Node address (Node-ID)

3.3.2.8 Object 100Ch: Guard-time

Contains the adjusted Node-Guarding-Time (guard-time) of the module in milli-seconds.

Object description:

INDEX	100Ch
Name	Guard-time (Guard-time)
Object type	Individual value (VAR)
Data type	16-bits without preceding sign (UNSIGNED16)
Category	Conditional; mandatory if heart-beat-log is not supported.

Object entry:

Access	Read only
PDO-mapping	no
Value	Adjusted guard-time

3.3.2.9 Object 100Dh: Time window

Contains the adjusted factor for the node guarding (lifetime = guard-time * factor)

Object description

INDEX	100Dh
Name	Time window
Object type	Individual value (VAR)
Data type	8 bits without preceding sign (UNSIGNED8).
Category	Conditional; mandatory if heart-beat-log is not supported.

Object entry

Access	Read only
PDO-mapping	no
Value	Adjusted factor

3.3.2.10 Object 100Eh: Guard-identifier

This object contains the identifier over which the node guarding is carried out.
This object is omitted from DS 301 version 4, but is, however, still contained for reasons of compatibility.

Object description

INDEX	100Eh
Name	Guard-identifier
Object type	Individual value (VAR)
Data type	32 bits without preceding sign (UNSIGNED32).
Category	Optional (V3.0)

Object entry

Access	Read only
PDO-mapping	no
Value	Node address + 1792

3.3.2.11 Object 100Bh: Number of the SDOs

This object contains the number of the SDOs which the operating panel supports.
This object is omitted from DS 301 version 4, but is, however, still contained for reasons of compatibility.

Object description:

INDEX	100Fh

Name	Number of the supported SDOs
Object type	Individual value (VAR)
Data type	32 bits without preceding sign (UNSIGNED32). D4/D5: Number server SDOs D6/D7: Number client SD's
Category	Optional (V3.0)

Object entry:

Access	Read only
PDO-mapping	no
Value	D4/D5: 1 server SDO D6/D7: 0 client SDO

3.3.2.12 Object 1010h: Save parameter

The operating panel does not support any parameter saving. Therefore the 0 is indicated here as highest subindex, what means no saving.

Object description:

INDEX	1010h
Name	Save parameter record
Object type	Field (ARRAY)
Data type	32 bits without preceding sign (UNSIGNED32).
Category	Optional

Object entry:

Subindex	0
Description	Number of the supported parameter records
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	0

3.3.2.13 Object 1011h: Load parameter record

The operating panel does not support any parameter saving. Therefore the 0 is indicated here as highest subindex, what means no saving.

Object description:

INDEX	1011h
Name	Load parameter record
Object type	Field (ARRAY)
Data type	32 bits without preceding sign (UNSIGNED32).
Category	Optional

Object entry

Subindex	0
Description	Number of the supported parameter records
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	0

3.3.2.14 Object 1014h: Identifier Emergency

This object contains the identifier, which is used for emergency objects.

Object description:

INDEX	1014h
Name	Emergency identifier
Object type	Individual value (VAR)
Data type	32 bits without preceding sign (UNSIGNED32).
Category	Mandatory if emergency is supported.

Object entry:

Access	Read only
PDO-mapping	no
Value	Node-ID + 128

3.3.2.15 Object 1015h: Emergency waiting time

In this object the time is saved which the operating panel has to wait al least between two emergency telegrams. This entry must be writeable. Since the operating panel does not permit at the moment the writing of the object directory, this object is not yet CANopen conforming. This is made up in the near future. The object itself exists already.

Object description:

INDEX	1015h
Name	Emergency waiting time
Object type	Individual value (VAR)
Data type	16 bits without preceding sign (UNSIGNED16)
Category	Optional

Object entry:

Access	Read only
PDO-mapping	no
Value	0

3.3.2.16 Obj. 1016h:Expected heartbeat time

The panels can monitor each other by means of the heartbeat (Heartbeat). This object contains information on which panels are monitored by the operating panel. At the moment the heartbeat is not supported, therefore we find here a 0 in the first entry.

Object description:

INDEX	1016h
Name	Heartbeat time
Object type	Field (ARRAY)
Data type	32 bits without preceding sign (UNSIGNED32). D4/D5: heartbeat time (UNSIGNED16) D6: Node-ID (UNSIGNED8)
Category	Optional

Object entries:

Subindex	0
Description	Number of the entries
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	1

Subindex	1
Description	Heartbeat-time entry 1
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	0

3.3.2.17 Object 1017: Manufacturer's heartbeat

In this object the "heartbeat" rhythm (Heartbeat) of the panel is entered in ms. Since the heartbeat-log is not supported, therefore we find here a 0 in the first entry.

Object description:

INDEX	1017h
Name	Manufacturer's heartbeat (Manufacturer Heartbeat)
Object type	Individual value (VAR)
Data type	16 bits without preceding sign (UNSIGNED16)
Category	Conditional; mandatory if monitoring is not supported

Object entry:

Access	Read only
PDO-mapping	no
Value	0

3.3.2.18 Object 1018h: Identity object

This object contains data which identify clearly a CANopen-device. Here manufacturer identification, serial number etc. are generated. A manufacturer number is entered in entry 1, which is placed exclusively by CiA.

Object description:

INDEX	1018h
Name	Identification object
Object type	Data record (RECORD)
Data type	Identity
Category	Mandatory

Object entries:

Subindex	0
Description	Number of the entries
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	4

Subindex	1
Description	Manufacturer identification (Vendor ID)
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign
Value	2Dh (Vendor ID GRAF-SYTECO)

Subindex	2
Description	Product identification
Category	Optional
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32).
Value	Panel-dependent: 6100d for ITS/AT 6100 series 6200d for ITS/AT 6200 series etc.

Subindex	3
Description	Revision number
Category	Optional
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32).
Value	1

Subindex	4
Description	Serial number
Category	Optional
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign
Value	Currently. not yet supported. Data are invalid.

3.3.2.19 Object 1200h: Server SDO parameter

In this object the identifier is found which the panel uses if SDO data are inquired.

Object description:

INDEX	1200h
Name	Server SDO parameter
Object type	Data record (RECORD)
Data type	SDO parameter
Category	Conditional

Object entries:

Subindex	0
Description	Number of the entries
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	2

Subindex	1
Description	COB-ID for SDO request SDO Rx ID from view of the panel
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32).
Value	1536 + NodeID

Subindex	2
Description	COB-ID for SDO reply SDO Tx ID from view of the panel
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign
Value	1792 + NodeID

3.3.2.20 Obj. 1400h: Receive PDO param.

In this object parameters are saved which concern the PDO reception.

Object description:

INDEX	1400h
Name	Receive-PDO parameter
Object type	Data record (RECORD)
Data type	PDO CommPar
Category	Mandatory for each supported receive PDO

Object entries:

Subindex	0
Description	Number of entries (sub-signs)
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	2

Subindex	1
Description	Receive-ID (COB-ID)
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32).
Value	512 + Node-ID

Subindex	2
Description	Type of transfer
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	8 bits without preceding sign (UNSIGNED8).
Value	FEh (async PDO)

3.3.2.21 Obj. 1600h: Receive-PDO mapping

Here the allocation of the receive PDO data to the object directory is made. Logically the both object entries of the object 2000h are to be found here.

Object description:

INDEX	1600h
Name	Receive-PDO-mapping
Object type	Data record (RECORD)
Data type	PDO-mapping
Category	Mandatory for each supported receive PDO

Object entries:

Subindex	0
Description	Number of the mapped objects
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	8 bits without preceding sign
Value	2

Subindex	1
Description	1. Mapped object (Index 2000h Subidx. 1, 32 bit)
Category	Conditional; here necessary because of firm PDO-mapping
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32)
Value	20012000h

Subindex	2
Description	2. Mapped object (Index 2000h Subidx. 2, 32 bit)
Category	Conditional; here necessary because of firm PDO-mapping
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32)
Value	20022000h

3.3.2.22 Object 1800h: Transmission PDO parameter

In this object parameters are saved which concern the PDO transmission.

Object description:

INDEX	1800h
Name	Transmission PDO parameter
Object type	Data record (RECORD)
Data type	PDO CommPar
Category	Mandatory for each supported transmission PDO

Object entries:

Subindex	0
Description	Number of entries (sub-signs)
Category	Mandatory
Access	Read only
PDO-mapping	no
Value	2

Subindex	1
Description	Transmisstion-ID (COB-ID)
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32)
Value	384 + Node-ID

Subindex	2
Description	Type of transfer
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	8 bits without preceding sign (UNSIGNED8).
Value	FEh (async PDO)

3.3.2.23 Object 1A00h: Transmission-PDO-mapping

Here the allocation of the receive PDO data to the object directory is made. Logically the both object entries of the object 2001h are to be found here.

Object description:

INDEX	1A00h
Name	Transmission PDO-mapping
Type of object	Data record (RECORD)
Data type	PDO-mapping
Category	Mandatory for each supported transmission PDO

Object entries:

Subindex	0
Description	Number of the mapped objects
Category	Mandatory
Access	Read only
PDO-mapping	no
Data type	8 bits without preceding sign
Value	2

Subindex	1
Description	1. Mapped object (Index 2001h Subidx. 1, 32 bit)
Category	Conditional; here necessary because of firm PDO-mapping
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32)
Value	20012001h

Subindex	2
Description	2. Mapped object (Index 2001h Subidx. 2, 32 bit)
Category	Conditional; here necessary because of firm PDO-mapping
Access	Read only
PDO-mapping	no
Data type	32 bits without preceding sign (UNSIGNED32)
Value	20022001h

3.3.2.24 Object 2000h: Received data

This object belongs to the manufacturer-specific objects. The operating panel saves in this object the data of the PDOs received last according to the PDO-mapping entered in the object 1600h.

Object description

INDEX	2000h
Name	PDO received data
Type of object	Data record (RECORD)

Object entries:

Subindex	0
Description	Number of the entries
Access	Read only
PDO-mapping	no
Data type	8 bits without preceding sign (UNSIGNED8)
Value	2

Subindex	1
Description	Data bytes D0 to D3 of the last received PDOs
Access	Read only
PDO-mapping	yes
Data type	32 bits without preceding sign (UNSIGNED32)

Subindex	2
Description	Data bytes D4 to D7 of the last received PDOs
Access	Read only
PDO-mapping	yes
Data type	32 bits without preceding sign (UNSIGNED32)

3.3.2.25 Object 2001h: Transmitting data

This object belongs to the manufacturer-specific objects. The operating panel saves in this object the data of the last transmitted PDOs according to the PDO-mapping entered in the object 1A00h.

Object description

INDEX	2001h
Name	PDO transmission data
Type of object	Data record (RECORD)

Subindex	0
Description	Number of the entries
Access	Read only
PDO-mapping	no
Data type	8 bits without preceding sign (UNSIGNED8)
Value	2

Subindex	1
Description	Data bytes D0 to D3 of the last transmitted PDOs
Access	Read only
PDO-mapping	yes
Data type	32 bits without preceding sign (UNSIGNED32)

Subindex	2
Description	Data bytes D4 to D7 of the last transmitted PDOs
Access	Read only
PDO-mapping	yes
Data type	32 bits without preceding sign (UNSIGNED32)

4 SIMATIC S5 [1]driver

An operating panel that is equipped with the SIMATIC-S5 driver can be coupled directly to the PG-interface of a S5. This interface is installed into each S5-control.
The data transfer from the S5 to the operating panel is to be done by the driver.
It must be indicated only where the message pointers are positioned and which data modules are to be used for nominal values etc.

4.1 Principle function of the driver

The panel is connected directly to the PG-interface via the (installed) 20mA/TTY-interface. In the ITE you parameterize the data modules and pointer bytes over which the image- and message call-ups as well as variable displays are to be realised.
The S5 driver simulates now a programming panel with running function "status variable" or"control variable". It requests data from the PLC and writes data (nominal values and keys) back into the PLC. No work is necessary with respect to the operating panel except for the parameterization of the data areas. Only the data modules which are parameterized in the driver have to be created in the S5-program.
Additionally the driver functions as "Gateway" between the CAN-interface of the operating panel and the PLC. That means the PLC receives indirectly access to the CAN-bus via the operating panel.
The S5 driver contains the following functions:

- *Image call-up (also PRIO images!) via word bits*
- *Image call-up via pointers*
- *Actual value display via data modules*
- *Nominal value entry into data modules*
- *Key query via pointers*
- *Switching on and off LED via* pointers
- *Influencing panel status via data module*
- *Access to the CAN-bus which can be connected to the operating panel (via data module)*

4.2 Basic considerations

First it has to be planned where the data are createdfor the operating panel in the PLC:

- *Image- and message call-ups as well as key- and LED-functions are handled via pointers. The driver needs a related area for all these functions. Reserve thus a block of relevant pointer bytes.*
- *The sequence how the functions are converted to pointer words is always the same. You can indicate how many pointer bytes per function are to be used.*
- *It is always proceeded in pointer-byte steps per function*
- *The sequence is always as follows:*
  - pointer bytes for LED control
  - pointer bytes for image call-up
  - pointer bytes for priority images
  - pointer bytes for messages.
- *Maximum 32 pointers can be used for image/ message call-ups (sum!). If this is not sufficient, then further call-ups can be made via the "Gateway"-function.*
- *Maximum 40 pointer bytes are necessary for these functions (32 for call-up, 8 for LEDs).*
- *Nominal- and actual values (variables) are also exchanged via data modules. It is possible to parameterize one's own data modules or/and commonly used data modules for nominal values, actual values and limits.*
- *The so-called "Handle" (see chapter "variable") is used here as number of the data word. If you therefore read here something about "Handle", then this is synonymous to "data word number".*
- *All variables with a length of 1-16 bits are allocated automatically a complete data word (see chapter "variables, types")*
- *Variables with a length of 32 bits are allocated two successively data words. This is to be considered when placing handles. Example: If a Longword-variable has the handle 6, then it allocates automatically the data words DW6 and DW7 in the data module. Therefore a variable from this data module with the handle 7 should not be used.*
- *Due to driver restrictions on the PG-interface, no variables of the same type (nominal values, actual values, upper-/ lower limit) may be used within an image whose handles have more then 64 differences. Example:If the smallest handle of nominal values amounts to 10 in the image, then the largest handle of nominal values may amount only to 74 in the image. An actual value may have now again e.g. the handle 90, since it belongs to another type. The "handle difference" may, however, not again be larger than 64 within the actual values.*
- *As data index OM2 to OM255 can be selected arbitrarily.The OMs have, however, to be set up in the PLC. If a parameterized OM is missing, then no communication takes place.*
- *Images and messages have to be numbered continuously starting from 1 if image/message call-ups occurs via pointer. The editor allows, however, gaps, but when using the S5 driver you have to pay attention to a "complete" creation.*

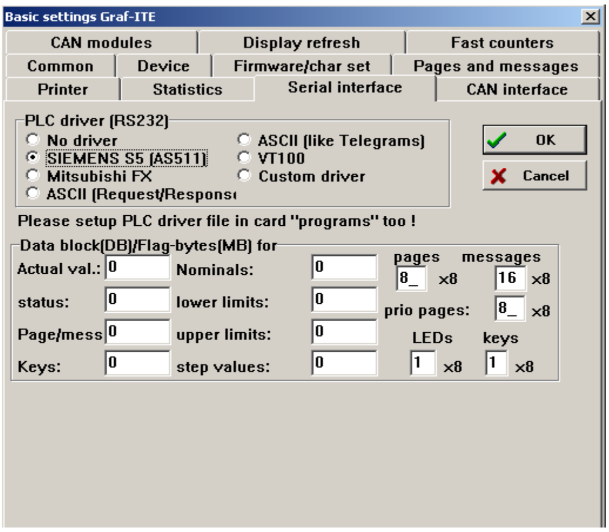
1. The terms SIMATIC and S5 are registered trademarks of the Siemens AG

- *IMPORTANT!!! All fields of the parameter mask (see below) have to be filled in. Thereby a data module must be indicated for each function, even if the function is not used. Otherwise no communication takes place.*

In practice it is shown that these rules are very simple to handle, since the parameterization of the driver and the variables is possible to do in a very comfortable manner.

4.3 Parameterization of the driver

You reach the parameter mask for the S5 driver via the menu "panel"/"parameterize", register card "serial interface". Click the button "Siemens S5 (AS511)". The following mask appears then:



4.3.1 Field actual values

Indicate here from which data module you want to deposit the actual values for the driver. Enter here e.g. OM5 if you want to use OM5 for actual values. An actual value with the handle 7 is then read from OM5, DW7.

4.3.2 Field "nominal values"

Enter into this field the designation of the data module in which the driver is to file nominal values. Nominal values are always filed then in this module if the operator has carried out a nominal value entry with saving in the operating panel.
Enter the complete designation of the data index, thus e.g. OM8.
You can indicate the same data module here as also with the actual values.
The handle of the variables is then again the number of the data word. Example: You have selected OM8 for nominal values. Then the nominal value is filed with the handle 20 in OM8 DW20.

4.3.3 Fields "lower limits", "upper limits"

With nominal value entries upper- and lower limits can be determined for the entry. Here you indicate from which data modules these limits are to be fetched if the limits are not set up absolutely as value but as variables.
Enter the complete designation of the data module, thus e.g. OM17.
You can indicate the same data module here as also with the nominal- and actual values.
The handle of the variables is then again the number of the data word. Example: You have selected OM17 for nominal values. Then the lower limit variable with the handle 12 is read from OM17 DW12.

4.3.4 Field "step values"

If you use nominal values with step-processing you can control the step values also via a variable. In this field you can now adjust from which OM you want to read the step values.
Enter the complete designation of the data module, thus e.g. OM2.
You can indicate also here the same data module as in another field.
The handle of the step value-variables is then again the number of the data word. Example: You have selected OM2 for nominal values. Then the step value with the handle 0 is read from OM2 DW0.

4.3.5 Field "status"

Enter here from which data module the operating panel is to file its current status. The driver needs this data module in any case, it should not be allocated by nominal-, actual values or limit values. Therefore indicate here another OM.
This OM is also used for the "Gateway" function; image/message call-ups are likewise settled via this OM which cannot be done via the pointer area.
The structure of the "status OM" is described still later.

4.3.6 Field "image/message"

In this field you parameterize the first pointer byte which is to be used for the function block "LEDs/ image/message call-up".
Enter the complete designation, thus e.g. MB30.
Then the pointer area starts with MB30 which is used for the LED control and image/message call-up.
How many pointer bytes are now needed depends on the parameterization of the fields "number of the images", "number of the messages" etc.

4.3.7 Fields "number of the images, messages, priority images, LEDs"

In these fields you indicate respectively how many pointer bytes you want to reserve for the individual function. Pay attention that you always work in steps of 8; you therefore cannot use 12 pointers for images, then 17 pointers for messages and 11 pointers for priority images.
If you do not use a function, enter then the number 0 for it. Then also no pointer byte is "wasted" for this function.
An example:
You use ITS6101. You want to call up 25 images and 25 messages, the 8 LEDs are likewise to be controlled. Five images as priority images should be able to appear. Keep an area free from MB50 for the panel.
For 25 images you need 4 pointer bytes (4*8=32 pointers), likewise for the messages. You can control the priority images via a pointer byte.
A pointer byte (=8 pointer) is sufficient for the LEDs. 10 pointer bytes from MB50, thus MB50-MB59 are necessary in all.
Enter the following values for this into the fields:

Field	Input value
Image/message	MB50
Number of the images	4
Number of the messages	4
Number of the priority images	1
Number of the LEDs	1

The driver reads out now MB50-MB59 (10 pointer bytes) cyclically and rates the pointers individually as call-ups. Thereby the assignment is as follows:

Pointer	LED/image/message
M50.0	LED 1
M50.1	LED 2
...	...
M50.7	LED 8
M51.0	Image 0
M51.1	Image 1
....	....
M51.7	Image 7
M52.0	Image 8
...	...
M54.7	Image 31
M55.0	Image 0, priority set
M55.1	Image 1, priority set
...	...
M55.7	Image 7, priority set
M56.0	Message 1
M56.1	Message 2
...	...
M56.7	Message 8
M57.0	Message 9
...	...
M59.7	Message 32

In order to switch on now a LED at the operating panel, the appropriate pointer in the PLC program is simply placed.

4.3.8 Field "keys"

Here you have to indicate where the panel is to mirror its keys in the pointers.
The complete designation has to be entered e.g. MB100.
Then the pointer area starts with MB100 where the panel mirrors its keys.
How many pointer bytes are now needed depends on the parameterization of the field "number of the keys".

4.3.9 Field "number of the keys"

Enter how many pointer bytes you want to reserve for the key status of the operating panel.
Always 8 pointers are reserved at the same time. Then also only key extensions in steps of 8 are offered for the panel.
Example:
You use an ITS6204 with 32 keys in total. All keys are to be mirrored in pointers. There are 4 pointer bytes necessary. You want to use MB60-MB63 as pointer area. Enter the following entries in the parameter mask:

Field	Entry
Keys	MB60
Number of the keys	4

Now the operating panel mirrors the keys into the pointers. The pointers mean now:

Pointer	= Key No.
M60.0	1 (1. row, on the left)
M60.1	2
...	...
M60.7	8 (1. row, on the right)
M61.0	9 (2. row, on the left)
....	....
M63.7	32 (4. row, on the right)

Numeric block at ITS/AT 61/67/71/77:
If the numeric keys are to be queried as well, then 8 pointer bytes have to be reserved in any case for the key query. The numeric keys are then to be found under the key numbers according to the following table:

Key	Key no.
Escape	50
"4“	51
"6“	52
"2“	53
"8“	54
Enter	55
"0“	57
"1“	58
"3“	59
"5“	60
"7“	61
"9“	62
".“	63
"+/-“	64

The pointers for the keys 49 and 56 are always placed with 0, please do not use these pointers further.
The function keys at the ITS6101 have the key numbers 1-8. If the ITS6106 is used (maximum extension), then the function keys are numbered from 1-48. Therefore the numeric keys are mirrored into the PLC from key number 50.

4.4 Status data component

The panel keeps the PLC informed concerning operator actions via the status OM. It files which image and which message are just being displayed and in which operating status it is at the moment. But also further functions of the operating panel are actuated via this status area. You have access to the CAN-bus. Besides you can influence panel parameters such as contrast and brightness via the status OM.
The indications of data words made in the following description refer always to the data module which you have entered in the field "status" of the parameter mask. The following table informs about the use of data words in the status OM:

Data words	Function
DW0-DW9	Panel status ITS
DW10-DW15	Transmission buffer for CAN-Gateway
DW16-DW21	Receive buffer for CAN-Gateway

4.4.1 Panel status information

The panel informs the PLC according to standard about the following data words with the contents specified in the table

Data word	Content of the data word
DW0	Image number of the currently displayed image
DW1	Message number of the currently indicated message (0=no message is displayed)
DW2	Panel status, see “REPORT_STATUS (0x0A)” on Page 9
DW3	Number of the active images
DW4	Number of the active messages
DW5-DW9	not allocated, reserved

4.4.2 CAN-Gateway transmission buffer

The data words DW10-DW15 of the status OMs are allocated as follows:

Data word	Function
DW10	Handshake. KH 0000: Transmission buffer free KH=FFFF: Transmission buffer allocated
DW11	CAN-identifier Here the user program must enter the addressee. Address 0 is the operating panel itself.
DW12	Telegram type/CAN user data
DW13	Function word 0/CAN user data
DW14	Function word 1/CAN user data
DW15	Function word 2/CAN user data

Handshake via DW10:
A tuning between PLC user program and the driver/CAN-bus takes place via DW10. The communication is handled with the following "FB framework":communication.

```
:A    OM   ...   Select status OM
:L    KH   0     Free identification
:L    DW   10    checking
:><F             Transmitter free?
:BEB             If not, end
:
:......          Enter here
:......          transmission data
:......          into DW11-DW15
:
```

```
:L    KH    FFFF   Enter transmission
:T    DW    10     identification
:BE
```

Thus it can be prevented that the PLC outputs data too quickly to the CAN-bus or the operating panel itself.

Telegram type and function words:
The telegram type and the function words are dependent on the addressee in D11:

DW11: KH=0000 Data for the operating panel at the PG-interface	DW11:KH=xxxx (Data are determined for the CAN-bus)
DW12 Telegram type - image call-up - delete image - message call-up - delete message - parameter commands	DW12 KH=aabb CAN user data: aa = byte0 bb = byte1
DW13 Function word 1 (allocated corresponding to the telegram type DW12, see following section)	D13 KH=ccdd CAN user data:cc = byte2 dd = byte3
DW14 Function word 2 (allocated corresponding to the telegram type DW12, see following section)	D13 KH=eeff CAN user data:ee = byte4 ff = byte5
DW15 Function word 3 (allocated corresponding to the telegram type DW12, see following section)	D15 KH=gghh CAN user data:gg = byte6 hh = byte7

Transmissions to the panel, DW11 KH=0:
The following commands can be placed with the ITS/AT via the transmission buffer (status OM):

Telegram type in DW12	Function word 1-3 DW13 - DW15
KF=+2 Transmitting variable (set value)	DW13: Handle of the variables DW14: Variable value (low word) DW15: Variable value (high word)
KF=+4 Message call-up	DW13: Number of the message being called up from KF=+1 to KF=+9999 DW14/15: not used
KF=+5 Deliver message	DW13: Number of the message being called up from KF=+1 to KF=+9999 DW14/15: not used
KF=+6 Call up image	DW13: Number of the message being called up from KF=+1 to KF=+9999 DW14/15: not used
KF=+7 Deliver image	DW13: Number of the message being called up from KF=+1 to KF=+9999 DW14/15: not used
KF=+8 Call up priority image	DW13: Number of the message being called up from KF=+1 to KF9999 DW14/15: not used

KH=15xx	Set panel parameters.
KH=1500 Place global soft-key mask	DW13: Bit mask for soft keys, KH=00xx bit 0 = not allocated bit 1 = menu key 1 bit 2 = menu key 2 ... bit 6 = menu key 6 bit 7 = not allocated If the bit of a key is placed, then the soft key function is placed for this key and the menu function is switched off. If the bit is 0, then the menu function of the key is activated. DW14/15: not used
KH=1501 Set contrast	DW13: Contrast value 0-23 (KF=+0 to KF=+23). 23 is maximum contrast DW14/15: not used
KH=1502 Brightness of the background lighting	DW13: Brightness value 0-7 (KF=+0 to KF=+7). 7 is maximum brightness. DW14/15: not used
KH=1503 Status line on/off	DW13: Status line function KF=+0 to KF=+2 0: Status line faded in 1: Status line faded out 2: as defined in the image DW14/15: not used
KH=1504 Position of the status line	DW13: Line number of the status line 0-7 (KF=+0 to +7). 0 is the topmost line. DW14/15: not used
KH=1505 Scrolling time of the messages	DW13: Scrolling time in seconds from KF=+0 to KF=+32 0 = "scrolling off" DW14/15: not used
KH=1506 Scrolling time of the images	DW13: Scrolling time in seconds from KF=+0 to KF=+32 0 = "scrolling off" DW14/15: not used
KH=1507 Key allocation of the menu keys	DW13: KH=uuvv with uu=number of the ESC key vv=number of the key "arrow on the left" DW14: KH=wwxx vv=number of the key "arrow on the right" vv=number of the key "arrow downwards" DW15: KH=yyzz vv=number of the key "arrow upwards" zz=Number of the Enter key
KH=1508 Message output	DW13: KH=0000 switching off KH=0100 switching on DW14/15: not allocated

These data are formatted in approximation to the CAN data format.“Description of the telegram types” on Page 4

4.4.3 CAN-Gateway receive buffer

Before you want to access too enthusiastically to the CAN-bus: take into account that a transfer rate of up to 1 MBit/s can be adjusted on the CAN-bus. On the PG-interface 9600 bauds are adjusted firmly of which approx. 75% for the log have to be counted. Thus there remain net approx. 2400 bauds.

If now a CAN-module relocates a telegram only 10 times per second, then the receive buffer ought to be written into the PLC 10 times per second, and at the same time the pointer-bytes and the variables to be read out - impossible!
Therefore why this whole thing?
Consider that e.g. one operating keyboard can be connected to the operating panel via the CAN-bus. Somehow you ought to be informed when a key is pressed on the operating keyboard - and this functions only via the operating panel. And to tell you the truth: so simply can you connect no other keyboards to the PLC as over the CAN-bus.
The operating keyboard transmits now each time a CAN-news to the operating panel if a key is pressed. The driver files then this telegram in the receive buffer.
Realistically seen an operator will press a key only 2-3 times per second - and the operating panel can buffer this still also, if necessary. Therefore it has a CAN-FIFO buffer with 20 telegrams depth.
If you signalise the operator via an LED that his key stroke has been registrated, then he will not begin to hammer like mad on the keyboard.
But now to the description of the data words of the receive buffer. Also the receive buffer has a handshake word available with whose help the data transfer is controlled as well as the information bytes:

Data word	Function
DW16	Handshake. KH 0000: Receive buffer empty KH=FFFF: Data in the buffer
DW17	CAN-identifier Here the user program in the PLC receives the address of the transmitter. Address 0 is the operating panel itself.
DW18	Telegram type/CAN user data
DW19	Function word 0/CAN user data
DW20	Function word 1/CAN user data
DW21	Function word 2/CAN user data

Transmissions from the panel: DW17 KH=0000
Currently no telegrams from the operating panel are defined at the PLC. All functions are handled via pointers and data modules.

Transmitting CAN module: DW17 KH=xxxx:
In this case the CAN telegram of the transmitter is written on a one-to-one basis into the receive buffer. Thus the bytes are filed as follows:

Data word	Contents
DW17	CAN identifier (Transmitter address) in the format according to 11.2.5
DW18	CAN user data KH=aabb aa = byte 0 bb = byte 1
DW19	CAN user data KH=ccdd cc = byte 2 dd = byte 3
DW20	CAN user data KH=eeff ee = byte 4 ff = byte 5
DW21	CAN user data KH=gghh gg = byte 6 hh = byte 7

The contents of the CAN telegram is dependent on the panel which has sent the telegram. Look up therefore in the manual on this panel if you have to determine the contents of the telegrams.

4.4.4 CAN identifier DW11 and DW17

The CAN-identifiers is composed of 16 bits in total. The individual bits have the following meaning:

15-5											4	3-0			
Identifier											R	DLC			
x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	0

In the bits 0-3, DLC (data length code) it is indicated how many bytes of user data the CAN telegram contains. This value can be 0 to 8. A CAN telegram can contain maximum 8 bytes of user data. The RTR-bit (R) is currently not used. Set it thus on 0.
The ID-bits 0-10 must contain the number of the panel. These are placed mostly via DIP-switch or jumper. Further details can be obtained from the manual of the respective panel.

4.4.5 " Examples for the Gateway

With the help of an interconnection of a Siemens S5, an ITS 6101 and an ITS 6303 we want to demonstrate the data transfer via the status OM. We assume that the ITS6101 and the S5 communicate via the PG interface and that the ITS6303 is connected to the ITS6101 via the CAN-bus.6101
6101 The panel address of the ITS6101 at the CAN-bus is not of importance; the ITS6303 is adjusted to the address 5.
6303 The task is now to detect if a key has been pressed on the ITS6303, likewise the LEDs on the keyboard are to be placed.

We want to do it in such a way that the LED of a key is to light up as long as until the key is pressed. Briefly a trip into the function of the ITS6303: it behaves in such a way that key activations are outputted automatically to the CAN-bus. This takes place by means of the REPORT_KEY_DATA telegram. See “REPORT_KEY_DATA (0x17)” on Page 17 . With this telegram the number of the key is communicated respectively and also whether the key has been pressed or released. You receive as additional information the status of the first 4 key rows in terms of bits.
With the telegram SET_LED we set/reset then the respective LED. “SET_LED (0x16)” on Page 16
We have to reserve 4 pointer bytes for these functions (here we remember the status of the keys and the last key stroke). Besides we need the status OM.
We use:

MB10	for keys 1-8
MB11	for keys 9-16
MB12	for keys 17-24
MB13	Number of the last key
OM10	Status module

The following PLC-program performs our task. It consists of 2 function modules:
Evaluation of the key data:
FB11 Transmitting the LED information

First we want to determine the CAN identifier for the ITS6303. The panel has the address 5 (ID = 5) and always 8 bytes of user data in the telegram. Thus DLC=8. Coded terms of bits we receive:

15-5											4	3-0			
Identifier											R	DLC			
0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0
0				1				2				8			

As CAN identifier we must therefore use KH=0128 for the ITS 6303.

Evaluation of the key data, FB10:

```
      :A     OM     10     Status OM
      :L     KH     FFFF   Status query
      :L     DW     16     with handshake
      :><F                 something there ?
      :BEB                 no.
      :L     DW     17     yes, from whom?
      :L     KH     0128   from ITS6303 ?
      :><F                 test
      :SPB   =M001         not from him
      :L     DW     19     Key number
      :SRW   8             read out
      :T     MB     13     and remember
      :L     DW     20     Keys 1-16
      :T     MW     10     remember
      :L     DW     21     Keys 17-24
      :SRW   8             standardize
      :T     MB     12     and remember
M001  :L     KH     0      make the
      :T     DW     16     receive buffer free
      :BE                  that is it done.
```

The second part of our task, setting the LED is be done with the following program:

```
      :L     MB     13     query key code
      :L     KF     +0     is one there ?
      :!=F                 let's look
      :BEB                 no nothing.
      :A     OM     10     Status OM
      :L     KH     FFFF   Look whether
      :L     DW     10     the transmission
                           buffer is free
      :!=F                 if it is allocated
      :BEB                 then there is no
                           action
      :L     KF     +4     Reception: setting
      :UN    M      13.7   Key status
      :SPB   =M001         querying: set
      :L     KF     +5     released: OFF
M001  :L     KH     1600   transmitting: LED
                           DATA
      :OW                  and function
      :T     DW     12     as user data 0, 1
      :R     M      13.7   Key code
      :L     MB     13     loading, = LED
                           No.
      :SLW   8             User data 3
      :T     DW     13     the number
      :L     KH     0128   Enter identifier
      :T     DW     11     for ITS6303
      :L     KH     FFFF   signalise: Trans-
                           mitting
      :T     DW     10     buffer filled
      :L     KF     +0     and still yet
      :T     MB     13     deleting MB13:
                           ready
      :BE
```

Now you have to call up these both FBs still cyclically from the OB1.
You see: it is really simple to use the Gateway to

the CAN-bus. Fill up simply the data words and query; that is already everything.
By the way: Do not forget to set the handshake fields DW10 and DW16 to 0 when restarting the PLC. Otherwise these FBs do not start. You have to set the MB13 likewise on 0, in order to start up no wrong LED. You should thus place these 5 program lines into OB21/OB22:

```
:A    OM    10    Status OM
:L    KF    +0    Handshakes
:T    DW    10    initialise
:T    DW    16    for run-up
:T    MB    13    and key code off
```

In the pointer bytes MB10 to MB12 you have fetched with these few program lines the status of 24 keys from the CAN-bus and controlled the appropriate LEDs. Fantastic, isn't it?

Here we can close the description of the driver for the S5.

5 Mitsubishi FX

In order to establish a simple connection possibility for the MITSUBISHI FX series, the operating panel can be supplied with a RS422-interface and be plugged directly to the PG-interface of the FX. Please order the operating panel with the appropriate interface (RS422) if you want to couple to the FX. The operating panel is then connected via an interface adapter (Order No. M232A or M232B) via the 20mA/TTY-interface to the PC.
The operating panel effects the data transfer via the interface. You only need to file the necessary pointer and variable data in the PLC - the access occurs parameterized by the operating panel. Therefore you do not need to provide a "transfer program code" in your PLC-program.

5.1 Principle function of the driver

The data areas are parameterized in the editor ITE to which the operating panel is to access. It is not necessary to assign each individual message or variable. A sort of initial address is indicated for the individual data and then the space ("offset") to this initial address is indicated via the handle number.
The operating panel provides now that a permanent data exchange takes place with the message pointers and the nominal/actual values indicated currently in the image. This is done without the help of the PLC.
The FX driver allows the following functions:

- *Image call-up (also PRIO images!) via word bits*
- *Image call-up via pointers*
- *Actual value display of data indexes*
- *Nominal value entry into data index*
- *Key query via pointers*
- *Switching-on and off LED via* pointers
- *Influencing the panel status*
- *Access to the CAN-bus which can be connected to the operating panel ("Gateway")*

5.2 Basic considerations

You must first plan where you set up the data for the operating panel in the PLC. Observe the following specifications for this:

- *Image- and message call-up as well as key- and LED-functions are handled via pointers. The operating panel needs a related area for all these functions.* T*herefore receive a block of pointer bytes (8 pointer steps) that belong together.*
- *Images and messages have to be numbered continuously starting from 1 if image/message call-ups occurs via pointer. The editor allows, however, gaps, but when using the FX-driver you have to pay attention to a "complete" creation.*
- *The sequence on how the functions are converted to pointer bytes is always the same. You can indicate how many pointer bytes per function are to be used.*
- *It is always preceded in pointer byte steps (8 pointers) per function*
- *The sequence is always as follows:*
  - pointer for LED control
  - pointer for image call-up
  - pointer for priority images
  - pointer for messages.
- *Maximum 256 pointers can be used for image/ message call-ups (sum!). If this is not sufficient then further call-ups can be made via the "Gateway"-function.*
- *Maximum 320 pointers are necessary for these functions (256 for call-up, 64 for LEDs)*
- *Nominal- and actual values (variables) are also exchanged via data indexes. It is possible to parameterize a commonly-used data index area for nominal values, actual values and limits.*
- *The so-called "Handle" (see chapter "variables") is added to the parameterized value of the data index ("offset"). If you therefore read here something about "* handle*", then this is synonymous to "data index offset". Of course, you can adjust the initial address for all data types equally. It is anyway the simplest if you adjust respectively the 0 for the data index(except status - more later on this). Because then the handle number which you adjust for variables is identical with the data index number. Thus you see immediately with the help of the handle number to which data index (or the operating panel) you access.*
- *All variables with a length of 1-16 bits are allocated automatically an entire data index. Variables with a length of 32 bits are allocated two successive data indexes. This has to be considered with the placement of handles. Example*:*If a Longword-variable has the handle 6, then it is allocated automatically two data indexes (e.g. D6 and D7). Thus no variable with the handle 7 should be used.*
- *Due to driver restrictions on the PG-interface, no variables of the same type (nominal values, actual values, upper-/ lower limit) may be used within an image whose handles have more then 32 differences. Example*:*If the smallest handle of nominal values amounts to 10 in the image, then the largest handle of nominal values may amount only to 42 in the image. An actual value may have now again e.g. the handle 90, since it belongs to another type. The "handle difference" may, however, not again be larger than 32 within the actual values.*

- *As data index, D0 to D7999 can be selected arbitrarily.*
- *IMPORTANT!!! All fields of the parameter mask (see below) have to be filled in. Thereby a data index/pointer must be indicated for each function, even if the function is not used.Otherwise no communication takes place.*

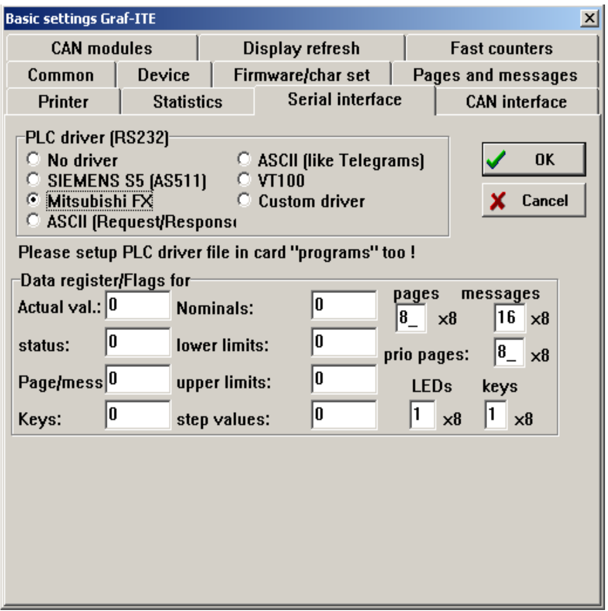
In practice it is shown that these rules are very simple to handle, since the parameterization of the driver and the variables is possible to do in a very comfortable manner.
You can/must parameterize individually the following data types:

- *Actual values (data index D0 - D7999)*
- *Nominal values (data index D0 - D7999)*
- *Lower limits (data index D0 - D7999)*
- *Upper limits (data index D0 - D7999)*
- *Step-values (data index D0 - D7999)*
- *Status information (data index D0 - D7999)*
- I*mage/message call-up, LEDs (M0 - M1536)*
- *Keys (M0 - M1536)*

With the pointers you have to consider that you may set the initial address only in steps of 8, otherwise you receive an error message during the transfer to the operating panel.

5.3 Parameterization of the driver

You reach the parameterized mask for the FX-driver via the menu "panel"/"parameterize", register card "serial interface". Click the button "Mitsubishi FX". The following mask appears then:



Enter in the fields "image/message" and "keys" each the beginning of a pointer area. Observe that the pointer number is divisible by 8; thus M0, M8, M16, M24 .....
You have to enter a data index (D0 - D999) into the fields actual values, nominal values, lower limits, upper limits, step-values and status. Enter also the letter "D", thus e.g. "D100".
In the fields "...x 8 " you have to enter the number of the "pointer bytes". The operating panel accesses always in steps of 8 pointers.

5.3.1 Field actual values

Indicate here from which data index you want to deposit the actual values for the operating panel. Enter here e.g. D5 if you want to use the data index D5 for actual values. An actual value with the handle 7 is then read from the data index D12 (basis D5 + handle 7 --> D12).

5.3.2 Field "nominal values"

Enter in this field the number of the data index from which the operating panel is to file nominal values. Nominal values are always filed then in these indexes if the operator has carried out a nominal value entry with saving in the operating panel.
Enter the complete designation of the data index, thus e.g. D8.
You can indicate the same data index here as also with the actual values.
The handle of the variables is then again the offset of the data word. Example: You have selected D8 for nominal values. Then the nominal value with the handle 20 is filed in the index D28 (basis D8 + handle 20 --> D28).

5.3.3 Fields "lower limits", "upper limits"

With nominal value entries upper- and lower limits can be determined for the entry. Here you indicate from which data indexes these limits are to be fetched if the limits are not set up absolutely as value but as variables.
Enter the complete designation of the data index, thus e.g. D17.
You can indicate the same data index here as also with the nominal- and actual values.
The handle of the variables is then again the offset of the data index. Example: You have selected D17 for lower limits. Then the lower-limit variable with the handle 12 is read out from the index D29 (basis D17 + handle 12 --> D29).

5.3.4 Field "step-values"

If you use nominal values with step processing you can control the step-values also via a variable. In this field you can now adjust from which data index you want to read the step-values.
Enter the complete designation of the data index, thus e.g. D2.
You can indicate also here the same data index as in another field.

The handle of the step-value variables is then again the offset of the data index. Example: You have selected D2 for nominal values. Then the step-value with the handle 0 is read from the index D2 (basis D2 + handle 0 --> D2).

5.3.5 Field "status"

Enter here from which data index the operating panel is to file its current status. The operating panel needs this data index in any case, it should not be allocated by nominal-, actual- or limit values. Enter therefore here another index.
This index area is also used for the "Gateway" function; image/message call-ups are likewise handled via these indexes, which cannot be done via the pointer area. Structure of the "status index-area" see below.

5.3.6 Field image/message

In this field you parameterize the first pointer which is to be used for the function block "LEDs/image/message call-up".
Enter the complete designation, thus e.g. M32. Then the pointer area starts with M32 which is used for the LED control and image/message call-up.
How many pointers are now needed depends on the parameterization of the fields "...x 8“.

5.3.7 Fields number of the images, messages, priority images, LEDs

In these fields you indicate respectively how many pointer bytes you want to reserve for the individual function. Pay attention that you always work in steps of 8; you therefore cannot use 12 pointers for images, then 17 pointers for messages and 11 pointers for priority images.
If you do not use a function, enter then the number 0 for it. Then a pointer byte is also not "wasted" for this function.
An example:
You use ITS6101. You want to call up 25 images and 25 messages, the 8 LEDs are likewise to be controlled. Five images as priority images should be able to appear. Keep an area free from M64 for the operating panel.
You need 4 pointer bytes for 25 images (4*8=32 pointers), likewise for the messages.
You can control the priority images via a pointer byte.
A pointer byte (=8 pointer) is sufficient for the LEDs. Ten pointer bytes from M64, thus M64-M143 are necessary in all.

Enter the following values for this in the fields:

Field	Input value
Image/message	M64
Number of the images	4
Number of the messages	4
Number of the priority images	1
Number of the LEDs	1

The operating panel reads out now M64-M143 (80 pointers) cyclically and rates the pointers individually as call-ups. Thereby the assignment is as follows:

Pointer	LED/image/message
M64	LED 1
M65	LED 2
...	...
M71	LED 8
M72	Image 0
M73	Image 1
....	....
M79	Image 7
M80	Image 8
...	...
M103	Image 31
M104	Image 0, priority set
M105	Image 1, priority set
...	...
M111	Image 7, priority set
M112	Message 1
M113	Message 2
...	...
M119	Message 8
M120	Message 9
...	...
M143	Message 32

In order to place e.g. an LED, you simply place the pointer in your PLC-program (just like an output) - and the LED lights up on the operating panel. It does not function any simpler.

5.3.8 Field keys

Here you have to indicate where the operating panel is to mirror its keys in the pointers.
Enter the full designation, thus e.g. M160 (divisible by 8 !!!).
Then the pointer area starts with M160 where the operating panel mirrors its keys.
How many pointers are now needed depends on the parameterization of the field "number of the keys".

5.3.9 Field number of the keys

Enter how many pointer bytes you want to reserve for the key status of the operating panel.
Always 8 pointers are reserved at the same time. For in the operating panel only key extension in steps of 8 are offered likewise.

Example:
You use an ITS6204 with 32 keys in total. You want to have mirrored all keys in the PLC as pointers.
Thus you have to reserve 4 pointer bytes. You want to use M160-M191 as pointer area. Enter the following entries into the parameter mask:

Field	Entry
Keys	M160
Number of the keys	4

Now the operating panel mirrors the keys into the pointers. The pointers mean now:

Pointer	= key No.
M160	1 (1. row, on the left)
M161	2
...	...
M167	8 (1. row, on the right)
M168	9 (2. row, on the left)
....	....
M191	32 (4. row, on the right)

Numeric block at ITS/AT 6100:
If the numeric keys of the ITS6100 are to be queried as well, then 8 pointer words (=64 pointers) have to be reserved in any case for the key query. The first 48 pointers are always assigned to the function keys. The numeric block is transferred from the 49. pointer.
The numeric keys are then to be found under the key numbers according to the following table (Example: M0 as basis):.

Key	Key no.	for basis M0
Escape	50	M49
"4	"51	M50
"6	"52	M51
"2	"53	M52
"8	"54	M53
Enter	55	M54
"0	"57	M56
"1	"58	M57
"3	"59	M58
"5	"60	M59
"7	"61	M60
"9	"62	M61
".	"63	M62
"+/-	"64	M63

The pointers for the keys 49 (M48.) and 56 (M55) are always placed with 0, please do not use these pointers further.
The function keys at the ITS6101 have the key numbers 1-8. If the ITS6106 is used (maximum extension), then the function keys are numbered from 1-48. Therefore the numeric keys are mirrored into the PLC from key number 50.

5.4 Status data index

The operating panel keeps the PLC informed regarding operator actions via the status data-index. It files which image and which message are currently being displayed and in which operating status it is at the moment.
But also further functions are actuated via this status area. You have access to the CAN-bus which can be connected at the operating panel. Besides you can influence panel parameters such as contrast and brightness via the status-area.
The indication of data indexes which have been made in the following description refer always as offset to the basic index which you have entered in the field "status" of the parameter mask (Example: If D2 stands in the text and you have indicated in the field "status" D10, then it is the "real" index D12). The following table informs about the use of data words in the status area:

Data index	Function
D0-D9	Panel status of the operating panel
D10-D15	Transmission buffer for CAN-Gateway
D16-D21	Receive buffer for CAN-Gateway

5.4.1 Panel status information

The operating panel informs the PLC according to standard about the following data words with the contents specified in the table

Data index	Contents of the data index
D0	Image number of the currently displayed image
D1	Message number of the currently displayed message (0=no message is displayed)
D2	Panel status. “REPORT_STATUS (0x0A)” on page 9
D3	Number of the active images
D4	Number of the active messages
D5-D9	not allocated, reserved

5.4.2 CAN-Gateway transmission buffer

The data words DW10-DW15 of the status DBs are allocated as follows:

Data word	Function
D10	Handshake. K0: transmission buffer free otherwise transmission buffer allocated
D11	CAN-identifier Here the user program must enter the addressee. Address 0 is the operating panel itself.
D12	Telegram type/CAN user data
D13	Function word 0/CAN user data
D14	Function word 1/CAN user data
D15	Function word 2/CAN user data

Handshake via D10:
A tuning between PLC user program and the operating panel/CAN-bus takes place via D10. The communication is handled with the following "AWL framework":

```
LD      M8000
        CMP
                K0
                D10
                M0
```

If the pointer M1 is now placed, then the transmission buffer is free and the data may be written into the transmission buffer.
Thus it can be prevented that the PLC outputs too quickly data to the CAN-bus (or the operating panel itself).

Telegram type and function words:
The telegram type and the function words are dependent on the addressee in D11:

D11=K0 Data for the operating panel at the PG-interface	D11<>K0 Data are determined for the CAN-bus
D12 Telegram type - image call-up - delete image - message call-up - delete message - parameterize commands	D12 KH=aabb CAN user data: aa = byte0 bb = byte1
D13 Function word 1 (allocated corresponding to the telegram type D12, see following section)	D13 KH=ccdd CAN user data: cc = byte2 dd = byte3
D14 Function word 2 (allocated corresponding to the telegram type D12, see following section)	D14 KH=eeff CAN user data: ee = byte4 ff = byte5
D15 Function word 3 (allocated corresponding to the telegram type D12, see following section)	D15 KH=gghh CAN user data: gg = byte6 hh = byte7

Transmissions to the operating panel, D11=K0:
The following commands can be conveyed to the operating panel via the transmission buffer (status area):

Telegram type in D12	Function word 1-3 D13 - D15
K2 Transmit variables (set value)	D13: Handle of the variables D14: Variable value (low word) D15: Variable value (high word)
K4 Message call-up	D13: Number of the message being called up from K1 to K9999 D14/15: not used
K5 Deliver message	D13: Number of the message being delivered from K1 to K9999 D14/15: not used
K6 Call up image	D13: Number of the image being called up from K1 to K9999 D14/15: not used
K7 Deliver image	D13: Number of the image being delivered from K1 to K9999 D14/15: not used
K8 Call up priority image	D13: Number of the priority image being called up from K1 to K9999 D14/15: not used
K5376 to K5384	Place panel parameters. Compare “WRITE_PARAM (0x15)” on page 14
K5376 Place global soft-key mask	D13: Bit mask for soft keys, KH=00xx bit 0 = not allocated bit 1 = menu key 1 bit 2 = menu key 2 .....bit 6 = menu key 6 bit 7 = not allocated If the bit of a key is placed, then the soft-key function is placed for this key and the menu function is switched off. If the bit is 0, then the menu function of the key is activated. D14/15: not used
K5377 Set contrast	D13: contrast value 0-23 (K0 to K23). 23 is maximum contrast D14/15: not used
K5378 Brightness of the background lighting	D13: brightness value 0-7 (K0 to K7). 7 is maximum brightness. D14/15: not used
K5379 Status line on/off	D13: Status line function K0 to K2 0: Status line faded in 1: Status line faded out 2: as defined in the image D14/15: not used
K5380 Position of the status line	D13: line number of the status line 0-7 (K0 to K7). 0 is the topmost line. D14/15: not used

K5381 Scrolling time of the messages	D13: scrolling time in seconds from K0 to K32. 0 = "scrolling off" D14/15: not used
K5382 Scrolling time of the images	D13: scrolling time in seconds from K0 to K32. 0 = "scrolling off" D14/15: not used
K5383 Key allocation of the menu keys	D13: KH=uuvv with uu=number of the ESC key vv=number of the key "arrow on the left" D14: KH=wwxx with vv=number of the key "arrow on the right" vv=number of the key "arrow downwards" D15: KH=yyzz with vv=number of the key "arrow upwards" zz=Number of the Enter key
K5384 Message output	D13: switch off K0 switch on K256 D14/15: not allocated

These data are formatted in approximation to the CAN data format. “Description of the telegram types” on page 4

5.4.3 CAN-Gateway receive buffer

Before you want to access too enthusiastically to the CAN-bus: take into account that a transfer rate of up to 1 MBit/s can be adjusted on the CAN-bus. On the PG-interface 9600 bauds are adjusted firmly of which approx. 75% for the log have to be estimated. Thus there remain net approx. 2400 bauds.
If now a CAN-module deposits a telegram only 10 times per second, the receive buffer ought to be written into the PLC 10 times per second, and at the same time the pointer-bytes and the variables to be read out - impossible!
Therefore why the whole thing?
Consider that e.g. one operating keyboard ITS6300 can be connected to the operating panel via the CAN-bus. Somehow you ought to be informed when a key is pressed on the ITS6300 - and this functions only via the operating panel. And to tell you the truth: so simply you can connect no other keyboard to the PLC as over the CAN-bus.
The ITS6300 transmits now each time a CAN-news to the operating panel if a key is pressed. The operating panel files then this telegram in the receive buffer.
Realistically seen, an operator will press a key only 2-3 times per second. The operating panel can buffer key entries possibly in a FIFO-buffer with 20 telegrams depth.
If you signalise the operator via a LED that his key stroke has been registered, then he will not begin to hammer like mad on the keyboard.
But now to the description of the data index of the receive buffer. Also the receive buffer has a handshake index available with whose help the data transfer is controlled as well as the information bytes:

Data index	Function
D16	Handshake. K0: Receive buffer empty otherwise data in the buffer
D17	CAN-identifier Here the user program in the PLC receives the address of the transmitter. Address 0 is the operating panel itself.
D18	Telegram type/CAN user data
D19	Function word 0/CAN user data
D20	Function word 1/CAN user data
D21	Function word 2/CAN user data

Transmissions from the operating panel: D17=K0
Currently no telegrams from the operating panel are defined at the PLC. All functions are handled via pointer and data components.

Transmitting CAN module: D17<>K0:
In this case the CAN telegram of the transmitter is written on a one-to-one basis into the receive buffer. Thus the bytes are filed as follows:

Data index	Contents
D17	CAN-identifier (transmitter address)
D18	CAN user data KH=aabb aa = byte 0 bb = byte 1
D19	CAN user data KH=ccdd cc = byte 2 dd = byte 3
D20	CAN user data KH=eeff ee = byte 4 ff = byte 5
D21	CAN user data KH=gghh gg = byte 6 hh = byte 7

The contents of the CAN telegram is dependent on the panel which has sent the telegram. Look up therefore in the manual relating to this panel if you have to determine the contents of the telegrams.

CAN identifier D11 and D17
The CAN-identifier composes of 16 bits in total. The individual bits have the following meaning:

15-5											4	3-0			
Identifier											R	DLC			
x	x	x	x	x	x	x	x	x	x	x	0	1	0	0	0

In the bits 0-3, DLC (data length code) it is indicated how many bytes of user data the CAN telegram contains. This value can be 0 to 8. A CAN telegram can contain maximum 8 bytes user data.
The RTR-bit (R) is currently not used. Place it thus on 0.

The ID-bits 0-10 must contain the number of the panel. These are placed mostly via DIP-switch or jumper. Further details can be obtained from the manual of the respective panel.

5.5 " Examples for the Gateway

With the help of the interconnection of a MITSUBISHI FX, a ITS6101 and a ITS6303, we want to demonstrate the data transfer via the status indexes. We assume that the ITS6101 and the FX communicate via the PG-interface and that the ITS6303 is connected to the ITS6101 via the CAN-bus.
The panel address of the ITS6101 at the CAN-bus is not of importance; the ITS6303 is adjusted to the address 5.
The task is now to detect if a key has been pressed on the ITS6303, likewise the LEDs on the keyboard are to be placed.
We want to do it in such a way that the LED of a key shall light up as the key is held pressed.
Briefly a trip into the function of the ITS6303: it behaves in such a way that key activation's are outputted automatically to the CAN-bus. This takes place by means of the REPORT_KEY_DATA telegram (see “REPORT_KEY_DATA (0x17)” on page 17). Here the number of the key is transferred respectively and also whether the key has been pressed or released. You receive as additional information the status of the first 4 key rows bit by bit.
With the telegram SET_LED we set/reset then the respective LED (see “SET_LED (0x16)” on page 16).
We have to reserve 4 pointer bytes for these functions (here we remember the status of the keys and the last key stroke). Besides we need the status indexes.

We use:

M0-M7	for relational operations
M80-M87	for keys 1-8
M88-M95	for keys 9-16
M96-M103	for keys 17-24
M104-M119	temporary pointers
D30	Number of the last key
D0	Status component

The following PLC-program performs our task.

First we want to determine the CAN identifier for the ITS6303. The panel has the address 5 (ID = 5) and always 8-byte user data in the telegram. Thus we receive DLC=8. coded bit-by-bit:

Bit No.

15-5											4	3-0			
Identifier											R	DLC			
0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0
0				1				2				8			

As CAN identifier we must therefore use KH=0128 for the ITS6303. This corresponds to the decimal number K296 accordingly.

Evaluation of the key data:

```
LD   M8000
CMP             Status DB
     K0         Status query
     D16        with handshake
     M0
LD   M1         what is there ?
CMP             yes, from whom?
     K296       from ITS6303 ?
     D11        test
     M0
LD   M1         from ITS6303
MOV             Keys 1-16
     D20        load into the pointers
     K4M80
MOV             Keys 17-32
     D21
     K4M96
MOV             Key number
     D19        at first
     D30        saving
MOV             transmission buffer
     K0         set free
     D10        that is it done.
```

Simple, isn't it? Nearly like querying an input module. Think over that you have connected 24 keys quasi free-of-charge additionally to the PLC and can query in the pointers M80-M103.
The second part of our task, the placement of the LED is be done with the following program:

```
LD   M8000
CMP             query key code
     D30        is there one there ?
     K0         let's look
     M0
LDI  M1         if not 0: Key
CMP
```

	K0	transmission buffer free ?
	D10	
	M4	Yes --> M5 set
LD	M8000	Key code in pointers
MOV		
	D30	
	K4M104	104-119
LD	M5	transmission buffer free ?
ANI	M119	and pressed ?
MOV		
	K5636	KH 1604: SET LED
	D12	as command
LD	M5	transmission buffer free ?
AND	M119	and pressed ?
MOV		
	K5637	KH 1605: RESET LED
	D12	as command
LDI	M8000	now still the Key No.
OUT	M119	without Bit 7
LD	M5	transmission buffer free ?
MOV		yes, key code=LED
	K4M104	
	D13	in user data 3
MOV		
	K296	Identifier ITS6303
	D11	enter
MOV		
	K-1	Transmission buffer
	D10	Transmitting
MOV		
	K0	Key code
	D30	deleting

You see: it is really simple to use the Gateway to the CAN-bus. Fill in simply the data index and query; that is already everything.
By the way: Do not forget to set the handshake indexes D10 and D16 to 0 when restarting the PLC. Otherwise these functions do not start. You have to set the D30 likewise to 0 in order not to trigger off any wrong LED. Therefore you should use the following program lines as initialisation program:

LD	M8002	Start-up pointer
MOV		
	K0	
	D10	Transmission buffer free
MOV		
	K0	
	D16	Receive buffer free
MOV		
	K0	
	D30	No key is there

In the pointers M80 to M103 you have fetched with these couple of program lines the status of 24 keys from the CAN-bus and controlled the appropriate LEDs. Great, isn't it?

Here we can close the description of the driver for the MITSUBISHI FX.

6 Request/response driver

In order to provide for a simple connection possibility for user-defined controls, the operating panel can be actuated via an RS232 interface (optional RS422).
The operating panel effects the data transfer via the interface. You only have to configure the required basis addresses for variables, page and message call-ups, LEDs and keys in the ITE editor - the operating panel accesses in accordance with the parameterisation the basis address adjusted by you. You therefore only have to respond to requests of the operating panel, which considerably simplifies the programming efforts.

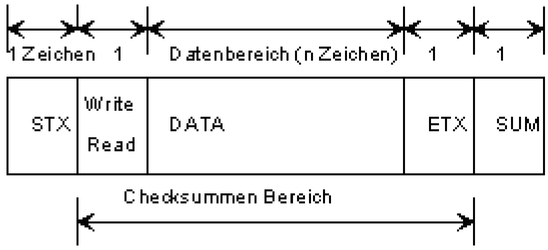
6.1 Interface description:

Interface:	RS232C standard
Protocol:	described in the following
Baud rate:	300 - 9600 bauds
Parity:	no, even, odd
Stop bits:	1,2
Data bits:	8

6.1.1 Interface commands:

Charac-ters	Hex code	Description
ENQ	05 hex	Enquiry: enquiry of the operating panel
ACK	06 hex	Acknowledge: response to an ENQ
NACK	15 hex	Negative Ack: response in case of a transmission error
STX	02 hex	Start of text: start byte of a telegram
ETX	03 hex	End of text: end code of a telegram
WRITE	31 hex	Operating panel transmits data to the control
READ	30 hex	Operating panel requests data from the control

Transmission format:
ENQ, ACK and NACK are transmitted as individual commands. All other commands (STX,ETX, Write,Read) are transmitted within a transmission frame:



The check sum is transmitted as single character (byte) according to the ETX code.

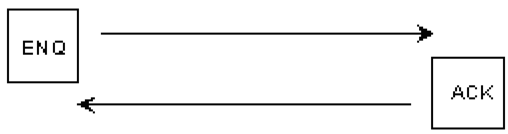
Example:

STX	CMD	High	Low	DLN	ETX	Check sum
	READ	TOP AD-DRESS				
02h	30h	10h	00h	04h	03h	47h
	30h + 10h + 00h + 04h + 03h = 47h					

All hex bytes following the byte CMD to ETX are added up to the check sum.
DLN: Data length code (number of bytes that are requested)

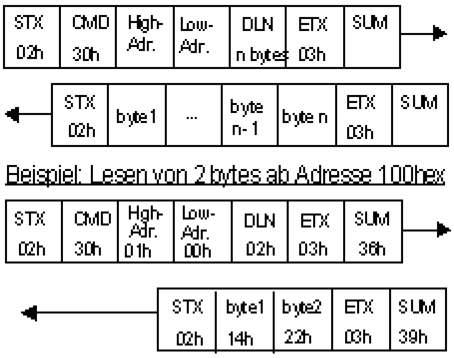
6.1.1.1 Communication telegrams:

Initialisation check
Operating panel <> control

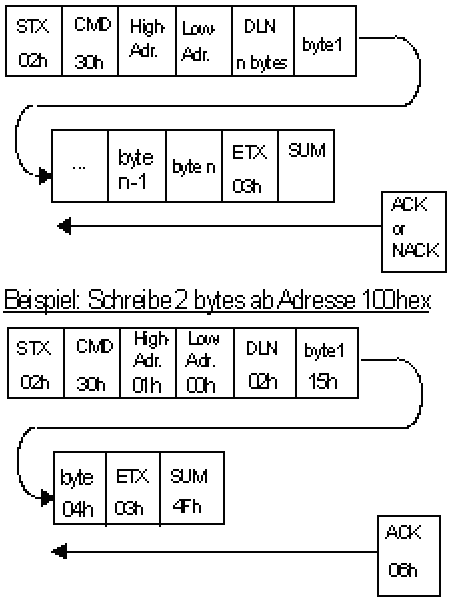


Max. delay = 500 msec between request and response

6.1.1.2 Read telegram



6.1.1.3 Write telegram



6.2 General driver function

The communication model of the "request/response" driver is reproduced in accordance with the data access and the memory organisation of a PLC.
The operating panel reads from/writes into data and flag areas of the control. Address and size of the data blocks to be transferred are contained in the "read/write" telegrams.
The control behaves passively and only responds to the "enquiries" (read/write) of the operating panel. This considerably simplifies the programming efforts for the communication driver on the control side.

The data areas which are to be accessed by the operating panel are parameterised in the ITE editor. It is not necessary to assign each message or variable individually. A kind of initial address is specified for the individual data and the offset to this initial address is indicated via the handle number.

The operating panel now ensures that a permanent data exchange with the message flags and the nominal/actual values currently being displayed on the page is performed. This is done without the help of the control.
The "free request/response" driver incorporates the following functions:

- *Page call-up (also PRIO pages!) via flags.*
- *Message call-up via flags.*
- *Actual value display from data indexes.*
- *Nominal value entry into data indexes.*
- *Key request via flags.*
- *Switching LED on and off via flags.*
- *Influencing the device status.*
- *Access to the CAN bus which can be connected to the operating panel ("gateway").*

6.2.1 Name agreements

In the following description on the driver configuration, terms from the PLC environment such as flag, flag byte, data index etc. are used. Within this context, the following agreement applies:

Flag	M	Bit 0..7 within a flag byte	Bit position within a data byte, addressable via address in the read/write telegram.
Flag byte	MB	1 byte	Addressable via address in the read/write telegram.
Data word	D	2 bytes	Addressable via address in the read/write telegram.
Examples:			
Basis address = A102			
Flag M102.5 = bit 5, within the data bytes under address 102hex.			
Data word D102 = 16bit (2 bytes) Data under address 102hex (occupies address 102hex and 103hex) --> the next addressable data word is D104 (address 104hex).			
Flag byte M102 = 1 byte date under address 102hex.			

6.2.2 Basic considerations

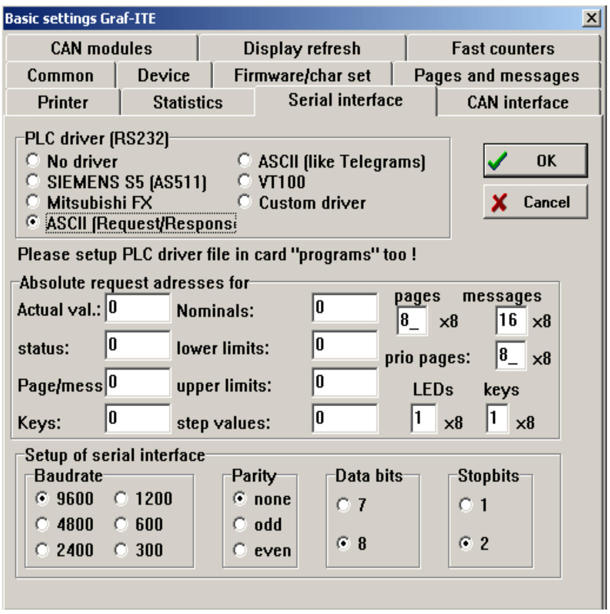
You first have to plan where you store the data for the operating panel in the control. Please observe the following specifications within this context:

- *Page and message call-ups as well as key and LED functions are processed via flags (bit-oriented). The operating panel needs a related area for all these functions. For this reason, please reserve a block of related flag bytes.*

- *Pages and messages should be numbered consecutively starting from 0. Based on the basis address adjusted in the ITE editor, the page/message number is assigned to the respective "bit".*
  Example: PageNo.=9,
  adjusted basis addr. = 100hex --> bit1 of the addr. 101hex is assigned to page 9.
- *The transcription of functions to flag bytes is always executed in the same sequence. You can specify how many flag bytes are to be used per function.*
- *The transcription of each function is always executed in flag bytes steps (8 flags).*
- *The sequence is always as follows:*
  - Flags for LED actuation
  - Flags for page call-up
  - Flags for priority pages
  - Flags for messages.
- *A maximum of 256 flags can be used for page/ message call-ups (total!). If this is not sufficient, further call-ups can be made via the "gateway" function.*
- *A maximum of 320 flags is required for these functions (256 for call-ups, 64 for LEDs).*
- *Nominal and actual values (variables) are exchanged via data indexes. It is possible to parameterise a commonly used data index area for nominal values, actual values and limits.*
- *The handle is added to the parameterised value of the data index ("offset"). Whenever you come across the term "handle", it is synonymously used to "data index offset". Of course you can adjust the same initial address for all data types. The easiest thing would be anyway to set 0 for each data index (except status - more details later on). Then, the handle number which you adjust for variables is identical with the data index number. Thanks to the handle number you then immediately realise which data index you (or the operating panel) access.*
- *All variables with a length of 1-16 bit automatically occupy an entire data index.*
- *Variables with a length of 32 bit occupy two successive data indexes. This fact has to be considered when assigning handles. Example: If a longword variable has handle 6, it automatically occupies two data indexes (e.g. D6 and D7). This is why no variable with handle 7 should be used.*
- *Due to driver restrictions on the serial interface, no variables of the same type (nominal values, actual values, upper/lower limit) must be used within a page whose handles are characterised by a difference of more than 64. Example: If the smallest handle of nominal values amounts to 10 on the page, the largest handle of nominal values may only amount to 74 on the page. In contrast, an actual value may have, e.g., handle 90, as it belongs to another type. The "handle difference" within the actual values must, however, not exceed 64.*
- *As data index, D0 to D9999 can be selected arbitrarily.*
- *IMPORTANT!!! All fields of the parameterisation mask (see below) must be filled in. Here, a data index/flag must be indicated for each function, even if the function is not used. Otherwise no communication takes place.*
- *Practical applications have proven that these rules are very easy to handle due to the comfortable parameterisation of the driver and the variables.*
- *You can/must parameterise the following areas (16 bit HEX addresses) individually:*
  Actual values (address A0 - AFFFF)
  Nominal values (data index A0 - AFFFF)
  Lower limits (data index A0 - AFFFF)
  Upper limits (data index A0 - AFFFF)
  Step values (data index A0 - AFFFF)
  Status information (data index A0 - AFFFF)
  Page/message call-up, LEDs (A0 - AFFFF, flag-oriented)
- *Keys (A0 - AFFFF, flag-oriented)*

6.3 Driver parameterisation

The parameterisation mask for the "request/response" driver can be accessed via the "Device"/ "Parametrize" menu, in the "Serial interface" index card. Click the "request/response" button. The following mask shows up:



Enter the respective beginning of a flag area in the

"Page/message" and "Keys" fields, e.g. A1F0 for address 1F0hex. .....
Enter a basis address in HEX format (A0 - AFFFF) into the actual values, nominal values, lower limit, upper limit, step values and status fields. Also enter the letter "A", e.g. "A5DC0" for the address 5DC0hex.
Into the "...x 8 " fields, you have to enter the number of "flag bytes". The operating panel always accesses in steps of 8 flags.

6.3.1 "Actual values" field

Indicate in this field the address with which you want to start depositing the actual values for the operating panel. Enter here, e.g. A6 if you want to use the address 0006hex for actual values. An actual value with handle 7 is then requested by means of the address 0014hex (basis addr. = 0006hex + handle 7 * 2bytes [2 bytes are reserved for each variable handle] --> 0014hex).

6.3.2 "Nominal values" field

Enter the HEX address with which the operating panel is to start storing the nominal values. Nominal values are stored under this address whenever the operator has carried out a nominal value entry with saving in the operating panel.
Enter the complete designation of the basis address, e.g. A100.
You can indicate the basis address which you used for the actual values.
The handle of the variables is again the offset of the adjusted address (2 bytes offset per variable handle number).
Example:
You have selected A100 for nominal values. Then, the nominal value with handle 20 is stored under the address A128hex (basis A100 + handle 20 [2 bytes per handle] --> 128hex).

6.3.3 "Lower limits", "Upper limits" and "Step values" fields

When entering nominal values, upper and lower limits can be determined for the entry. In this field, you determine the addresses with which the operating panel is to start adopting these limits if the limits are not specified as absolute values but as variables.
Enter the complete designation of the HEX address, e.g. A1017.
You can indicate the address which you used for nominal and actual values.
The handle of the variables is again the offset of the basis address. Example: You have selected A1017 for lower limits. Then, the lower-limit variable with handle 12 is requested with the address 1028h (basis 1017hex + handle 12 [2 bytes per handle] --> 1028hex).

6.3.4 "Status" field

Please enter the address with which the operating panel is to start storing its current status. The operating panel requires this address area in any case; it should not be occupied by nominal, actual or limit values. For this reason, please enter another basis address here.
This address area is also used for the "gateway" function. Additionally, page/message call-ups are settled via this address which cannot be realised via the flag area.
For the structure of the "status data area" please refer to below.

6.3.5 "Page/message" field

Parameterise the address area (is evaluated in a flag (bit)-oriented way) which is to be used for the "LEDs/page-message call-up" function block.
Enter the complete designation, e.g. A50.
Then, the address area which is used for the LED control and page/message call-up starts with address 50hex.
The number of flags (bits) required depends on the parameterisation of the "...x 8“ fields.

6.3.6 "...x 8" fields

Specify the number of flag bytes you want to reserve for the individual function in this field.
If you do not use a function, enter 0 for the number of flag bytes. This way, no address is "wasted" for this function.

An example:
You use ITS6101. You want to be able to call up 25 pages and 25 messages, the 8 LEDs should be actuated as well. Five pages are to be designed in a way which enables their availability as priority pages. They reserve an area deletion address A50 for the operating panel.
For 25 pages you need 4 flag bytes (4*8=32 flags), the same applies to messages.
The priority pages can be controlled via a flag byte (8 flags).
For the LEDs only one flag byte (=8 flag) is sufficient. A total of 10 flag bytes starting with address A50, i.e. A50 - A5A (50 hex to 5A hex) is required.
Enter the following values into the fields:

Field	Input value
Page/message	A50
Number of pages	4
Number of messages	4
Number of priority pages	1
Number of LEDs	1

The operating panel now cyclically requests the flag area by means of which a read telegram with the address 0050hex and a data length of 10 bytes and classifies the flags individually as call-ups (address 50hex - 59hex =80 flags).
The respective allocation is as follows:

Flag	LED/page/message
A50.0	LED 1
A50.1	LED 2
...	...
A50.7	LED 8
A51.0	Page 0
A51.1	Page 1
....	....
A51.7	Page 7
A51.8	Page 8
...	...
A54.7	Page 31
A55.0	Page 0, priorised
A55.1	Page 1, priorised
...	...
A55.7	Page 7, priorised
A56.0	Message 1
A56.1	Message 2
...	...
A56.7	Message 8
A57.0	Message 9
...	...
A59.7	Message 32

In order to set, e.g., a LED, simply set the flag in your control program in flag byte, which is assigned to the address A50 - and the LED lights up on the operating panel. There is no easier way.

6.3.7 "Keys" field

Please indicate where the device is to map its keys to flags.
Enter the complete designation, e.g. A160.
Then, the flag area to which the device maps its keys starts with A160.
The number of flags required depends on the parameterisation of the "Number of keys" field.

6.3.8 "Number of keys" field

Enter the number of flag bytes you want to reserve for the key status of the operating panel.
As a principle, 8 flags at a time are reserved, as the key extension for the operating panel is also only possible in steps of 8.

Example:
You use an ITS6204 with a total of 32 keys. All keys in the control are to be mapped to flags.
Therefore, you need to reserve 4 flag bytes. You want to use A160-A163 as address area.
Enter the following entries into the parameterisation mask:

Field	Entry
Keys	A160
Number of the keys	4

Now, the operating panel transmits the key status as write telegram with the address 0160hex and the data length 4 bytes (for the telegram format, please refer to Section 1.2.1). The flag bytes (flags) under the address A160 have now the following meaning:

Address (flags)	= Key No.
A160.0	1 (1st row, on the left)
A160.1	2
...	...
A160.7	8 (1st row, on the right)
A161.0	9 (2nd row, on the left)
....	....
A163.7	32 (4th row, on the right)

Number pad with ITS/AT 6100:
If the number keys of the ITS/AT 6100 are to be called up as well, 8 flag words (=64 flags) must be reserved in any case for the key call-up. The first 48 flags are always assigned to the function keys. The number pad is transferred starting with the 49th flag. The number keys can then be found under the key numbers contained in the following table (Example: M0 as basis):

Key	Key no.	For basis A0
Escape	50	A6.1
"4	"51	A6.2
"6	"52	A6.3
"2	"53	A6.4
"8	"54	A6.5
Enter	55	A6.6
"0	"57	A7.0
"1	"58	A7.1
"3	"59	A7.2
"5	"60	A7.3
"7	"61	A7.4
"9	"62	A7.5
".	"63	A7.6
"+/-	"64	A7.7

The flags for the keys 49 (A6.0) and 56 (A6.7) are always set to 0, please do not use these flags for further purposes.
The function keys of the ITS6101 have the key numbers 1-8. If the ITS6106 is used (maximum extension), the function keys are numbered from 1-48. Therefore, the number keys are mapped to the control starting with key number 50.

6.3.9 Status data indexes

By means of the status data indexes, the operating panel keeps the control informed on operator actions. It stores the page and message which is currently being displayed as well as the current operating status.
But also further functions of the operating panel are addressed via this status area. You have access to the CAN bus which can be connected to the operating panel. Furthermore, you can influence device parameters such as contrast and brightness via the status area.
The status data indexes (data words) specifications included in the following description always refer as offset to the basis address you have entered in the "Status" field of the parameterisation mask (Example: If D2 stands in the text and you have indicated A200 in the "Status" field, the "real" address of the status byte is A204 [2 bytes are reserved per data index). The following table describes the use of the data indexes in the status area:

Data index	Function
D0-D9	Device status of the operating panel
D10-D15	Send buffer for CAN gateway
D16-D21	Receive buffer for CAN gateway

6.3.10 Device status information

As a standard, the device informs the PLC via the following data words with the contents specified in the table:

Data index	Contents of the data index
D0	Page number of the currently displayed page
D1	Message number of the currently displayed message 0=no message is displayed
D2	Device status
D3	Number of active pages
D4	Number of active messages
D5-D9	Not assigned, reserved

6.3.11 CAN gateway send buffer

The data words D10-D15 of the status area are assigned the following functions:

Word	Function
D10	Handshake. K0: Send buffer free otherwise send buffer occupied
D11	CAN identifier Here, the user program must enter the addressee. Address 0 is the operating panel.
D12	Telegram type/CAN user data
D13	Function word 0/CAN user data
D14	Function word 1/CAN user data
D15	Function word 2/CAN user data

Handshake via D10:
The user program and the operating panel/CAN bus are matched to each other via the D10.
If the data index D0 contains the value 0, the send buffer is free and the data may be written into the send buffer.
Herewith, the PLC can be prevented from transmitting data too quickly to the CAN bus or the operating panel itself.

Telegram type and function words:
The telegram type and the function words depend on the addressee in D11:

D11=0 Data for the operating panel at the PG interface	**D11<>0** Data are determined for the CAN bus
D12 Telegram type - Page call-up - Delete page - Message call-up - Delete message - Parameterisation commands	D12 KH=aabb CAN user data: aa = byte0 bb = byte1
D13 Function word 1 (occupied in accordance with telegram type D12, refer to the following section)	D13 KH=ccdd CAN user data: cc = byte2 dd = byte3
D14 Function word 2 (occupied in accordance with telegram type D12, refer to the following section)	D14 KH=eeff CAN user data: ee = byte4 ff = byte5
D15 Function word 3 (occupied in accordance with telegram type D12, refer to the following section)	D15 KH=gghh CAN user data: gg = byte6 hh = byte7

Transmissions to the operating panel, D11=0:
The following commands can be issued to the ITS/AT via the send buffer (status area):

Telegram type in D12	Function words 1-3 D13 - D15
2 Transmitting variables (set value)	D13: Variable handle D14: Variable value (low word) D15: Variable value (high word)
4 Message call-up	D13: Number of the message to be called up, from 1 to 9999 D14/15: Not used
5 Remove message	D13: Number of the message to be removed, from 1 to 9999 D14/15: Not used
6 Call up page	D13: Number of the page to be called up, from 1 to 9999 D14/15: Not used
7 Remove page	D13: Number of the page to be removed, from 1 to 9999 D14/15: Not used
8 Call up priority page	D13: Number of the priority page to be called up, from K1 to K9999 D14/15: Not used
5376 to 5384	Set device parameter. “WRITE_PARAM (0x15)” o13n page
5376 Set global soft-key mask	D13: Bit mask for soft-keys, KH=00xx bit 0 = not assigned bit 1 = menu key 1 bit 2 = menu key 2 ... bit 6 = menu key 6 bit 7 = not assigned If the bit of a key has been set, the soft key function is set for this key and the menu function is switched off. If the bit is 0, the menu function of the key is activated. D14/15: Not used
5377 Set contrast	D13: Contrast value 0-15 (0 to 23). 23 is maximum contrast D14/15: Not used
5378 Brightness of the background lighting	D13: Contrast value 0-7 (0 to 7). 7 is maximum brightness. D14/15: Not used
5379 Status line on/off	D13: Status line function 0 to 2 0: Status line displayed 1: Status line blanked out 2: As defined on the page D14/15: Not used
5380 Position of the status line	D13: Line number of the status line (0 to 7). 0 is the top line. D14/15: Not used
5381 Alternating time of messages	D13: Alternating time in seconds, from 0 to 32. 0 = "alternating off" D14/15: Not used
5382 Alternating time of pages	D13: Alternating time in seconds, from 0 to 32. 0 = "alternating off" D14/15: Not used
5383 Key mapping of the menu keys	D13: KH=uuvv with uu=number of the ESC key vv=number of the "arrow left" key D14: KH=wwxx vv=number of the "arrow right" key vv=number of the "arrow down" key D15: KH=yyzz vv=number of the "arrow up" key zz=number of the Enter key
K5384 Message output	D13: 0 switching off 256 switching on D14/15: Not assigned

These data are formatted following the CAN data format (refer to the “Description of telegram types” o4n page).

6.3.12 CAN gateway receive buffer

Before you snap too enthusiastically at the CAN bus, please consider that a transmission rate of up to 1 MBit/s can be adjusted for the CAN bus. A maximum of 9600 bauds can be adjusted for the RS232 interface.
If a CAN module transmits a telegram only 10 times per second, the receive buffer would have to be written 10 times per second into the control and at the same time the flag bytes and the variables would have to be read out - impossible!
What's the point in it?
Consider that, e.g., an ITS6300 operating keyboard can be connected to the operating panel via the CAN bus. Somehow you must be informed when a key is pressed on the ITS6300 - and this can only be realised via the operating panel. Frankly, there is no easier way to connect other keyboards to the PLC than via the CAN bus.
The ITS6300 now transmits a CAN message to the operating panel each time a key is pressed. The operating panel then files this telegram in the receive buffer.
Realistically speaking, an operator presses a key only 2-3 times per second - and, if required, the operating panel can buffer these key entries in a CAN-FIFO buffer with a 20 telegram capacity.
If you signalise the operator via LED that his key stroke has been registered, he will not begin to hammer like mad on the keyboard.
But now, we will concentrate on the description of data index of the receive buffer. Also the receive buffer provides a handshake index on the basis of which the data transfer as well as the information bytes are controlled:

Data index	**Function**
D16	Handshake. 0: Receive buffer empty otherwise data in the buffer

D17	CAN identifier Here, the user program in the PLC receives the transmitter address. Address 0 is the operating panel.
D18	Telegram type/CAN user data
D19	Function word 0/CAN user data
D20	Function word 1/CAN user data
D21	Function word 2/CAN user data

Operating panel telegrams: D17=0
Currently no telegrams from the operating panel to the PLC are defined. All functions are realised via flags and data indexes.
CAN module telegrams: D17<>0:
In this case, the CAN telegram of the transmitter is written on a one-to-one basis into the receive buffer. Within this context, the bytes are stored as follows:

Data index	**Contents**
D17	CAN identifier (transmitter address)
D18	CAN user data KH=aabb aa = byte 0 bb = byte 1
D19	CAN user data KH=ccdd cc = byte 2 dd = byte 3
D20	CAN user data KH=eeff ee = byte 4 ff = byte 5
D21	CAN user data KH=gghh gg = byte 6 hh = byte 7

The contents of the CAN telegram depend on the panel which has sent the telegram. For this reason, please refer to the respective device manual if you need to determine the contents of the telegrams.

6.4 CAN identifier D11 and D17

The CAN identifier consists of a total of 16 bits. The individual bits have the following meaning:

15-5											4	3-0			
Identifier											R	DLC			
0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0
0				1				2				8			

The bits 0-3, DLC (data length code) indicate how many bytes of user data are included in the CAN telegram. This value can be 0 to 8. A CAN telegram can contain a maximum of 8 bytes of user data.
The RTR bit (R) is currently not used. Therefore, set it to 0.
The ID bits 0-10 must contain the device number, which is usually set via a DIP switch or jumper. Further details can be obtained from the manual of the respective device.

7 VT100 driver

Using the VT100 driver, the operating panel can be actuated like a VT100 terminal. Also XOn-XOff control sequences have been implemented.
Pages can be freely created and page contents subsequently modified via the VT100 control sequences. You may also start with a completely empty page and transfer all information.

7.1 Configuration

With the ITE, the project file VT100.txt is delivered, which represents an empty screen and initialises the VT100 driver. This project contains the following presettings:

- *"Page 0" is defined as empty page, i.e. neither texts, graphics nor variables are available on this screen page.*
- *("Page 0" is the displayed screen page after a "power-up" of the operating panel.)*
- *In the "Device/Parametrize/Programs" folder, the current TOS and driver versions IO042S00.hex (TOS) and VT042S00.drv (VT100 driver) have been selected.*
- *In the "Device/Parametrize/Device" folder, the device version ITS6100 has been selected.*
- *In the "Device/Parametrize/Serial interface" folder, the setting "VT100 driver" and the VT option "0" ("0"=standard key codes, "1"=additional codes) have been selected. The interface parameters have been set to 9600 bauds, 2 stop bits, 8 data bits, no parity.*

If required, you may now modify the adjusted values and transfer the project to the operating panel. For this purpose, connect the operating panel to your PC using the ITK100 adapter and transfer your project into the device.

After switching on the operating panel, a publicity page appears for approx. 2 sec. (can be created in the ITE editor, refer to the Operating panel manual). Lines 7 and 8 of the publicity page display the respective versions of BIOS (firmware), TOS (operating system), driver as well as the project name (UserData).
If the VT100 driver has been loaded, "DRV:VT" appears in line 7 of the publicity page. The blinking of the "VT" driver identification indicates that the driver and the operating system (TOS) are incompatible. Ensure that the version numbers of the driver and the TOS are identical (Device/Parametrize/Programs folder, e.g. IO042S00.hex and VT042S00.drv for version 042S00).

After the start-up time (approx. 2 sec) has been expired, the configured empty page (page 0) is displayed. The communication via VT100-interface can start (refer to the following protocol description).

7.2 Description of the VT-100 control sequences

The following sections describe all Escape and control sequences of the VT100 driver of the operating panel series. The sequences have been taken from the VT101 terminal manual of the "Digital Equipment Corporation (DEC)".

7.2.1 Control character (receive)

Name	Mnemonic	HEX	Description
Zero	NUL	0x00	Filling sign (is ignored)
Enquire	ENQ	0x05	Transmits response
Bell	BEL	0x07	Alarm tone (message output is activated for 500 msec)
BackSpace	BS	0x08	Cursor left
LineFeed	LF	0x0A	Line down/column pos. remains unchanged
VerticalTab	VT	0x0B	As LF
FormFeed	FF	0x0C	As LF
CarriageReturn	CR	0x0D	Position cursor on the beginning of the current line
XON	DC1	0x11	XON: Terminal transmission release
XOFF	DC3	0x13	XOFF: Terminal must not transmit any further signs
Cancel	CAN	0x18	Received sequence is deleted, substitution character is displayed
Substitute	SUB	0x1A	As cancel (CAN)
Escape	ESC	0x1B	Start byte of a sequence

7.2.2 ESC sequences (receive)

Name	Mne-monic	ASCII /HEX sequence	Description
Cursor positioning			
Cursor up	CUU	ESC [n A (1B 5B n 41)	Move cursor upwards by "n"-lines with n = '0'-'99' (ASCII)
Cursor down	CUD	ESC [n B (1B 5B n 42)	Move cursor downwards by "n"-lines with n = '0'-'99' (ASCII)
Cursor right	CUF	ESC [n C (1B 5B k 43)	Move cursor to the right by "n"-lines with n = '0'-'99' (ASCII)
Cursor left	CUB	ESC [n D (1B 5B n 44)	Move cursor to the left by "n"-lines with n = '0'-'99' (ASCII entry)
Cursor position	CUP	ESC [n ; m H (1B 5B n 3D m 48)	Cursor on line "n" and column "m" with n,m = '0'-'99' (ASCII entry)
Cursor-PosHome	CUP	ESC [H (1B 5B 48)	Cursor home position: "top left"
INDEX	IND	ESC D (1B 44)	Cursor 1 line downwards column position remains (LF)
Reverse index	RI	ESC M (1B 4D)	Cursor 1 line upwards column position remains
Next Line	NEL	ESC E (1B 45)	Cursor to 1st column in the next line
Character output			
Character		ASCII code 20hex - FFhex	All ASCII characters as of 20hex can be displayed (control characters 0 - 1F hex are not displayed)
Erasing			
Erase in line	EL	ESC [K (1B 5B 4B)	Erase from the cursor position to the end of the line
Erase in line	EL	ESC [0 K (1B 5B 30 4B)	Erase from the cursor position to the end of the line (see above)
Erase in line	EL	ESC [1 K (1B 5B 31 4B)	Erase from the beginning of the line to the cursor position
Erase in line	EL	ESC [2 K (1B 5B 32 4B)	Erase entire line (cursor line)
Erase in display	ED	ESC [J (1B 5B 4A)	Erase from the cursor position to the end of the screen
Erase in display	ED	ESC [0J (1B 5B 30 4A)	Erase from the cursor position to the end of the screen (as above)
Erase in display	ED	ESC [1J (1B 5B 31 4A)	Erase from the beginning of the screen to the cursor position
Erase in display	ED	ESC [2J (1B 5B 32 4A)	Erase entire display
Reports			
Status report	DSR	ESC [5 n (1B 5B 35 6E)	Request status. Reply: ESC [0 n (terminal OK) ESC [3 n (terminal not OK)
Status report	DSR	ESC [6 n (1B 5B 36 6E)	Request cursor position. Reply: ESC [n ; m R (n=line, m=column) with n,m = '0' - '99' (ASCII format)
Terminal reset			

Reset	RIS	ESC c (1B 63)	Initiate terminal reset
LED control			
Load LEDs	DE- CLL	ESC [q (1B 5B 71)	All LEDs off
Load LEDs	DE- CLL	ESC [0 q (1B 5B 30 71)	All LEDs off (as above)
Load LEDs	DE- CLL	ESC [n q (1B 5B n 71)	LED "n" on (n = LED number) with n = '0'-'99' (ASCII)
Character attributes			
Attribute reset	SGR	ESC [m (1B 5B 6D)	Switch off attributes: Font size 8 pixels Normal representation (no blinking)
Attribute reset	SGR	ESC [0 m (1B 5B 30 6D)	Switch off attributes: (as above)
Underline	SGR	ESC [4 m (1B 5B 34 6D)	Underlined characters
Font size 16	SGR	ESC [5 m (1B 5B 35 6D)	Font size 16 pixels
Font size 32	SGR	ESC [6 m (1B 5B 36 6D)	Font size 32 pixels
Reverse	SGR	ESC [7 m (1B 5B 37 6D)	Reverse characters
Blinking "ON“	"SGR	ESC [8 m (1B 5B 38 6D)	Blinking characters
Blinking "OFF“	"SGR	ESC [9 m (1B 5B 39 6D)	"Blinking" function is deactivated
Special telegrams (operating panel extensions)			
Signalling output OFF	EXT	ESC [0 x (1B 5B 30 78)	Deactivate signalling output
Signalling output ON	EXT	ESC [1 x (1B 5B 31 78)	Activate signalling output
Cursor OFF	EXT	ESC [2 x (1B 5B 32 78)	Cursor is no longer displayed
Cursor ON (block)	EXT	ESC [3 x (1B 5B 33 78)	Cursor is displayed as block
Cursor ON (underline)	EXT	ESC [4 x (1B 5B 34 78)	Cursor is displayed as "underline"
Adjust display brightness	EXT	ESC [n l (1B 5B n 6C)	Set brightness (n) of the display with n = '0'-'7' (ASCII)
Adjust display contrast	EXT	ESC [n C (1B 5B n 63)	Set contrast value (n) for the display with n = '0'-'23' (ASCII)

7.2.3 Key codes

(Terminal transmitted characters)

Name	Sequence / Code	Description
Cursor keys (only ITS7000 series)		
Up	ESC [A	Cursor up is pressed
Down	ESC [B	Cursor down is pressed
Right	ESC [C	Cursor right is pressed
Left	ESC [D	Cursor left is pressed
Number keys (hex format)		
0	0x30	
1	0x31	
2	0x32	
3	0x33	
4	0x34	
5	0x35	
6	0x36	
7	0x37	
8	0x38	
9	0x39	
.	0x2C	Comma
+/-	0x2D	Minus
Enter	0x0D	Enter
ESC	0x1B	Escape
BS	0x08	Back Space
Function keys		
F1	'A' (0x41)	F1 transmits an ASCII 'A'
F2	'B' (0x42)	F2 transmits an ASCII 'B'
F3	'C' (0x43)	F3 transmits an ASCII 'C'
F4	'D' (0x44)	F4 transmits an ASCII 'D'
F5	'E' (0x45)	F5 transmits an ASCII 'E'
F6	'F' (0x46)	F6 transmits an ASCII 'F'
F7	'G' (0x47)	F7 transmits an ASCII 'G'
F8	'H' (0x48)	F8 transmits an ASCII 'H'

- *Keys only transmit their code with a positive edge ("key released" is not reported).*
- *Answer-back telegram (= answer to an ENQ of the control) has been implemented. The answer-back telegram is an identification string with a maximum of 20 characters (in the operating panel series, the answer-back string is determined as an 'operating panel' constant).*

7.3 VT100 driver extensions

In the "option = 1" setting, the following additional control sequences are available:

7.3.1 ESC sequences (receive)

Name	Sequence	Description
Cursor positioning	ESC=XY	X: line position 1 to 8 Y: column pos. 1 to 40 X-value is between 0x20 and 0x27 (line 1 to 8) Y-value is between 0x20 and 0x66 (column 1 to 40) Example: 1B 3D 21 26 (line 2, column 7)
Line erasing	ESC T (1B 54)	The entire line of the cursor is erased
Display erasing	ESC + (1B 2B)	The entire display is erased
Blinking start	ESC j (1B 6A)	The following transmitted characters are represented in the "blinking" mode
Blinking stop	ESC k (1B 6B)	The "blinking" function is switched off

7.3.2 Transmitted key codes

Key pressed	ASCII	HEX code
Function key F1	'A'	0x41
Function key F2	'B'	0x42
Function key F3	'G'	0x47
Function key F4	'I'	0x49
Function key F5	'K'	0x4B
Function key F6	'C'	0x43
Function key F7	'F'	0x46
Function key F8	'J'	0x4A
ESC combined with number 1	'D'	0x44
ESC combined with number 2	'H'	0x48
ESC combined with number 3	'L'	0x4C
ESC combined with number 4	'E'	0x45
ESC combined with number 5	'M'	0x4D
ESC combined with number 6	'N'	0x4E
ESC combined with number 7	'O'	0x4F
ESC combined with number 8	'P'	0x50
Number key '0'	'0'	0x30
Number key '1'	'1'	0x31
Number key '2'	'2'	0x32
Number key '3'	'3'	0x33
Number key '4'	'4'	0x34
Number key '5'	'5'	0x35
Number key '6'	'6'	0x36
Number key '7'	'7'	0x37
Number key '8'	'8'	0x38
Number key '9'	'9'	0x39
Decimal point key	'.'	0x2E
Minus key	' '	0x20
ESC combined with minus key	'.'	0x2E
ENTER key	CR	0x0D

8 Intercontrol DIGSYplus

The Intercontrol DIGSYplus control can be connected to the operating panel via the COM-SP interface. The operating panel affects the data transfer via the interface. You only need to file the necessary pointer and variable data in the PLC - the access occurs parameterized from the operating panel. Therefore you do not need to provide a "transfer program code" in your PLC-program.

8.1 Principle function of the driver

The data areas are parameterized in the editor ITE to which the operating panel is to access. It is not necessary to assign each individual message or variable. A kind of initial address is indicated for the individual data and then the space ("offset") to this initial address is indicated via the handle number.
The operating panel provides now that a permanent data exchange takes place with the message pointers ("word bits") and the nominal/actual values indicated currently in the image. This is done without the help of the PLC.
The driver allows the following functions:

- *Image call-up (also PRIO images!) via word bits*
- *Message call-up via word bits*
- *Actual value display from pointer words*
- *Nominal value entry in pointer words*
- *Key and touch screen query via word bits*
- *Switching on and off LED via word bits*
- *Influencing the panel status*
- *Access to the CAN-bus which can be connected to the operating panel ("Gateway")*

8.2 Basic considerations

You must first plan where you set up the data for the operating panel in the PLC. Observe the following specifications for this:

- *Image and message call-ups as well as key- and LED-functions are handled via word bits (in pointer words). The operating panel needs a related area for all these functions. Reserve thus a block of relevant pointer words.*
- *Images and messages have to be numbered continuously starting from 1 if image/message call-ups occurs via pointer. The editor allows, however, gaps, but when using the Intercontrol-driver you have to pay attention to a "complete" creation.*
- *The sequence on how the functions are converted to pointer words is always the same. You can indicate how many pointer bytes per function are to be used.*
- *It is always preceded in pointer byte steps (8 pointers) per function*
- *The sequence is always as follows:*
  - pointer for LED control
  - pointer for image call-up
  - pointer for priority images
  - pointer for messages
- *Maximum 256 pointers (=16 pointer words) for image/message call-ups can be used (sum!). If this is not sufficient then further call-ups can be made via the "Gateway"-function.*
- *Maximum 320 pointers are needed for these functions (256 for image/message call, 64 for LED control)*
- *Nominal- and actual values (variables) are also exchanged via pointer words. It is possible to parameterize a commonly used pointer word area for nominal values, actual values and limits.*
- *The handle is added to the parameterized value of the data index ("offset"). If you therefore read here something about handle, then this is synonymous to "pointer word-offset". Of course you can adjust the initial address for all data types equally. It is anyway the simplest if you adjust respectively the address 3FE for the pointer word (except status - more later on this). Because then the handle number which you adjust for variables is identical with the pointer word number. Thus you see immediately with the help of the handle number to which pointer word (or the operating panel) you access.*
- *All variables with a length of 1-16 bits are allocated automatically an entire pointer word*
- *Variables with a length of 32 bits are allocated two successively pointer words. This has to be considered with the placement of handles. Example:If a Longword-variable has the handle 6, then it is allocated automatically two pointer words (e.g. MW6 and MW7). Thus no variable with the handle 7 should be used.*
- *Due to driver restrictions on the SP-interface no variables of the same type (nominal values, actual values, upper-/ lower limit) may be used within an image whose handles have more then 32 differences. Example:If the smallest handle of nominal values amounts to 10 in the image, then the largest handle of nominal values may amount only to 42 in the image. An actual value may have now again e.g. the handle 90, since it belongs to another type. The "handle difference" may, however, not again be larger than 32 within the actual values.*

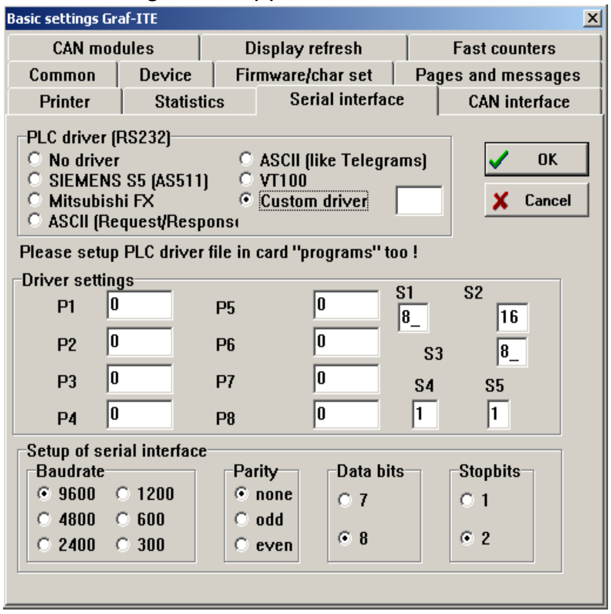
IMPORTANT !!! All fields of the parameter mask must be filled in. Thereby a pointer word address must be indicated for each function, even if the function is not used. Otherwise no communication takes place.

In practice it is shown that these rules are very simple to handle, since the parameterization of the driver and the variables is possible to do in a very comfortable manner.
You can/must parameterize individually the following data types:

- *Actual values (P1 in the register card)*
- *Nominal values (P5 in the register card)*
- *Lower limits (P6 in the register card)*
- *Upper limits (P7 in the register card)*
- *Step-values (P8 in the register card)*
- *Status information (P2 in the register card)*
- *Image/message call-ups, LEDs (P3 in the register card)*
- *Keys (P4 in the register card)*

8.3 Parameterization of the driver

You reach the parameter mask for the driver via the menu "panel"/"parameterize", register card "serial interface". Click the button "other driver". The following mask appears then:



Enter IC" (for InterControl) in the field behind the selection "other driver".
The fields P1 to P8 are fed with absolute addresses (indication hexadecimal) for different task areas. The addresses computes to:

MW address = 1024 + (pointer word number-1)*2

Example: the pointer word 1 has the address 1024+(1-1)*2 = 1024 = 400hex

Example: the pointer word 50 has the address 1024+(50-1)*2 = 1122 = 462hex

The fields mean:

8.3.1 P1:Basis address for actual values

This setting indicates from which pointer word the actual value variables from the PLC are read. The handle of the variable is each added to this address. So up to 256 actual values can be "addressed". Example: if you have entered as basis address 400(hex), then the variable with handle 0 corresponds to MW1, variable with handle 1 to MW2 etc.
Tip: If you use 3FE (hex) as initial address, then the handle number corresponds exactly to the MW number. But you may not use then the handle 0. MW0 does not finally exist....

8.3.2 P2:Basis address for status info

Here you indicate the address of the pointer word from which the status information of the operating panel is filed. 22 pointer words are needed for this. Also the Gateway-buffer belongs to this area.

8.3.3 P3:Basis address for LED and calls

This field must contain the initial address for the LED-pointer, image- and message call pointer ("word bits"). The number of the necessary pointer words is dependent on the parameterization of the fields S1 to S4.

8.3.4 P4:Basis address for key pointer

The operating panel announces pressed keys bit for bit into the PLC. Here you can indicate the address of the pointer word from which the key pointers are filed. You parameterize the number of the pointers via the field S5.

8.3.5 P5:Basis address for nominal values

Parameterization like P1

8.3.6 P6:Basis address for lower limits

Parameterization like P1

8.3.7 P7:Basis address for upper limits

Parameterization like P1

8.3.8 P8: Basis address for step values

Parameterization like P1

8.3.9 Field S1

In this field you parameterize how many pointer words you want to use for the image call-up. It must be observed here that the entry has to be done decimally and is expected in double steps. If you want to thus parameterize 2 pointer words = 32 image-pointers (32 word bits), then enter please a 4. Note: this corresponds to the images 1-32.

8.3.10 Field S2

In this field you parameterize how many pointer words you want to use for the message call-up. It must be observed here that the entry has to be done decimally and is expected in double steps. If you want to thus parameterize 1 pointer word = 16 message-pointers (16 word bits), then enter please a 2. Note: this corresponds to the messages 1-16.

8.3.11 Field S3

In this field you parameterize how many pointer words you want to use for the image call-up. It must be observed here that the entry has to be done decimally and is expected in double steps. If you want to thus parameterize 1 pointer word = 16 image-pointers (16 word bits), then enter please a 2. Note: this corresponds to the images 1-16.

8.3.12 Field S4

In this field you parameterize how many pointer words you want to use for the control of the LEDs of the operating panel. It must be observed here that the entry has to be done decimally and is expected in double steps. If you want to thus parameterize 1 pointer word = 16 LED-pointers (16 word bits), then enter please a 2. Note: this corresponds to the LEDs 1-16.

An example of this:
If you have indicated the pointer word MW 1 (address 400 hex) as basis for image/message call-up and fill in the fields S1 to S4 as in the table, then the word bits are as follows:

Field /value	Pointer word/ word bits	Bits correspond to
S1/2	MW1/ WB1.1 to WB1.16	LEDs 1 - 16
S2/4	MW2, MW3/ WB2.1 to WB3.16	Images 1 - 32
S4/2	MW4/ WB4.1 to WB4.16	Prio-images 1 - 16
S3/4	MW5, MW6/ WB5.1 to WB6.16	Messages 1 - 32

The operating panel reads out these pointer words cyclically and classifies the pointers individually as call-ups.

In order to display e.g. an image, you simply place the appropriate pointer in your PLC-program (just like an output) - and the image appears on the operating panel. It does not function any simpler.

8.3.13 Field S5

Here you have to indicate how many keys are to be mirrored in the pointer.
An indication in double steps is to be expected. A 2 means that 16 key pointers / 1 pointer word are used. Additionally 4 pointer words are hanged up in which you can bring in free arbitrary data from the operating panel. And, namely, the internal variables with handles 1000 and 1001 have been expanded for this task.
The values of this variable appear in the 4 pointer words which follow after the pointer words indicated for the keys.
An example: If you have indicated MW32 as pointer word for keys (address 440 hex) and indicate 2 in the field S2, then the keys appear as word bits in MW32; in MW33/MW34 the value of the internal variable with handle 1000 and in MW35/MW36 the value of the internal variable with handle 1001.
So you have the possibility e.g. when using the touch-screen to use individual bits for individual touch keys.

Note:
These pointer words are described with coincidental values if the internal variables 1000 and 1001 are not project-planned. Thus do not use them under any circumstances.

Another example:
You use an ITS6204 with 32 keys in total. You want to have mirrored all keys in the PLC as pointers.
Thus you have to reserve 4 pointer bytes. You want to use WB10.1 to WB11.16 as pointer area.
Enter the following entries into the parameter mask:

Field	Entry
P4	414
S5	4

Observe that MW12/13 and MW14/15 are reserved for the variable values of the variables 1000 and 1001.

Now the operating panel mirrors the keys into the pointers. The pointers mean now:

Pointer	= Key No.
WB10.1	1 (1. row, on the left)
WB10.2	2
...	...
WB10.8	8 (1. row, on the right)
WB10.9	9 (2. row, on the left)
....	....
WB11.16	32 (4. row, on the right)

Numeric block at ITS6100:
If the numerical keys of the ITS6100 are to be queried as well, then 4 pointer words (=64 pointers) have to be reserved for the key query. The first 48 pointers are always assigned to the function keys. The numeric block is transferred from the 49. pointer.
The numeric keys are then to be found under the key numbers according to the following table (Example: MW17/addr. 420 hex as basis):

Key	Key no.	for basis M0
Escape	50	WB20.2
"4“	51	WB20.3
"6“	52	WB20.4
"2“	53	WB20.5
"8“	54	WB20.6
Enter	55	WB20.7
"0“	57	WB20.9
"1“	58	WB20.10
"3“	59	WB20.11
"5“	60	WB20.12
"7“	61	WB20.13
"9“	62	WB20.14
".“	63	WB20.15
"+/-“	64	WB20.16

The pointers for the keys 49 (WB20.1) and 56 (WB20.8) are always placed with 0, please do not use these pointers further.
The function keys at the ITS6101 have the key numbers 1-8. If the ITS6106 is used (maximum extension), then the function keys are numbered from 1-48. Therefore the numeric keys are mirrored into the PLC from key number 50.

8.4 Status data index

The operating panel keeps the PLC informed concerning operator actions via the status data index. It files which image and which message are just being displayed and in which operating status it is at the moment.
But also further functions are actuated via this status area. You have access to the CAN-bus which can be connected at the operating panel. Besides, you can influence panel parameters such as contrast and brightness via the status area.

The indication of pointer words which have been made in the following description refer always as offset to the basic index which you have entered in the field "P2" of the parameter mask (Example: If MW+2 stands in the text and you have indicated in the field "status" MW10, then it is the "real" index MW10+2). The following table inform about the use of data words in the status area:

Data index	Function
MW+0 to MW+9	Panel status of the operating panel
MW+10 to MW+15	Transmission buffer for CAN-Gateway
MW+16 to MW+21	Receive buffer for CAN-Gateway

8.4.1 Panel status information

The operating panel informs the PLC according to standard about the following data words with the contents specified in the table (reference field P2):

Data index	Contents of the data index
MW+0	Image number of the currently displayed image
MW+1	Message number of the currently displayed message (0=no message is displayed)
MW+2	Panel status. See list under TA=0x0A
MW+3	Number of the active images
MW+4	Number of the active messages
MW+5	Seconds and minute, BCD-encoded
MW+6	Hours and weekday, BCD-encoded
MW+7	Day and month, BCD-encoded
MW+8	Year 4-digit, BCD-encoded
MW+9	not allocated, reserved

8.4.2 CAN-Gateway transmission buffer

The data words DW+10 to DW+15 of the status DBs are allocated as follows (reference field P2):

Data word	Function
MW+10	Handshake.0: Transmission buffer free otherwise transmission buffer allocated
MW+11	CAN-identifier Here the user program must enter the addressee. Address 0 is the operating panel itself.
MW+12	CAN user data D0/D1
MW+13	CAN user data D2/D3
MW+14	CAN user data D4/D5
MW+15	CAN user data D6/D7

Handshake via MW+10:
A tuning between PLC user program and the operating panel/CAN-bus takes place via MW+10. First it has to be checked whether MW+10=0. Then the data are entered in MW+11 to MW+15 and afterwards (!) MW+10 is set to 1.
Thereby it can be prevented that the PLC outputs too quickly data to the CAN-bus (or the operating panel itself).

Telegram type and function words:
The telegram type and the function words are dependent on the addressee in MW+11:

MW+11=0 Data for the operating panel at the PG-interface	MW+11<>0 (Data are determined for the CAN-bus)
MW+12 CAN user data corresponding to the description of CAN telegrams	MW+12 KH=bbaa aa = D0 bb = D1
MW+13 CAN user data corresponding to the description of CAN telegrams	MW+13 KH=ddcc cc = D2 dd = D3
MW+14 CAN user data corresponding to the description of CAN telegrams	MW+14 KH=ffee ee = D4 ff = D5
MW+15 CAN user data corresponding to the description of CAN telegrams	MW+15 KH=gghh gg = byte6 hh = byte7

8.4.3 CAN-Gateway receive buffer

Before you want to access too enthusiastically to the CAN-bus: take into account that a transfer rate of up to 1 MBit/s can be adjusted on the CAN-bus.

On the PG-interface 9600 bauds are adjusted firmly of which approx. 75% for the log have to be counted. Thus there remain net approx. 2400 bauds.
If now a CAN-module relocates a telegram only 10 times per second, then the receive buffer ought to be written into the PLC 10 times per second, and at the same time the pointer-bytes and the variables to be read out - impossible!
Therefore why the whole thing?
Consider that e.g. one operating keyboard ITS6300 can be connected to the operating panel via the CAN-bus. Somehow you ought to be informed when a key is pressed on the ITS6300 - and this functions only via the operating panel. And to tell you the truth: so simply you can connect no other keyboards to the PLC as over the CAN-bus.
The ITS6300 transmits now each time a CAN-news to the operating panel if a key is pressed. The operating panel files then this telegram in the receive buffer.
Realistically seen, an operator will press a key only 2-3 times per second. The operating panel can buffer key entries possibly in a FIFO-buffer with 20 telegrams depth.
If you signalise the operator via a LED that his key stroke has been registrated, then he will not begin to hammer like mad on the keyboard.
But now to the description of the data index of the receive buffer. Also the receive buffer has a handshake index available with whose help the data transfer is controlled as well as the information bytes:

Data index	Function
MW+16	Handshake. 0: Receive buffer empty otherwise data in the buffer
MW+17	CAN-identifier Here the user program in the PLC receives the address of the transmitter. Address 0 is the operating panel itself.
MW+18	CAN user data D0/D1
MW+19	CAN user data D2/D3
MW+20	CAN user data D4/D5
MW21	CAN user data D6/D7

Transmissions from the operating panel: MW+17=0
Currently no telegrams from the operating panel are defined at the PLC. All functions are handled via pointer and data components.

Transmission from the CAN-bus: MW+17<>0:
In this case the CAN telegram of the transmitter is written on a one to one basis into the receive buffer. Thus the bytes are filed as follows:

Data index	Contents
MW+17	CAN-identifier (transmitter address)
MW+18	CAN user data KH=bbaa aa = D0 bb = D1
MW+19	CAN user data KH=ddcc cc = D2 dd = D3
MW+20	CAN user data KH=ffee ee = D4 ff = D5
MW+21	CAN user data KH=hhgg gg = D6 hh = D7

The contents of the CAN telegram is dependent on the panel which has sent the telegram. Look up therefore in the manual regarding this panel if you have to determine the contents of the telegrams.

8.4.4 CAN-identifier MW+11 and MW+17

The CAN-identifier composes of totally 16 bits. The individual bits have the following meaning:

15-5											4	3-0			
Identifier											R	DLC			
x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	0

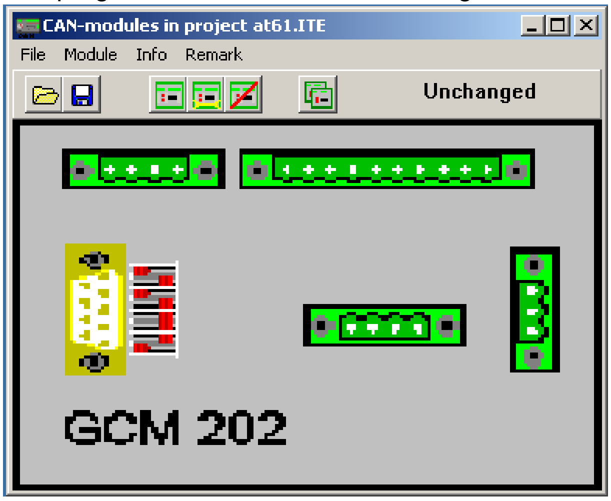
In the bits 0-3, DLC (data length code) it is indicated how many bytes of user data the CAN telegram contain. This value can be 0 to 8. A CAN telegram can contain maximum 8 bytes of user data.
The RTR-bit (R) is currently not used. Place it thus on 0.
The ID-bits 0-10 must contain the number of the panel. These are placed mostly via DIP-switch or jumper. Further details can be obtained from the manual of the respective panel.

9 Configurating CAN-modules

9.1 Starting the module-configurator

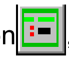
The integration of CAN-modules is carried out with a configuration program which you can call up from the editor.

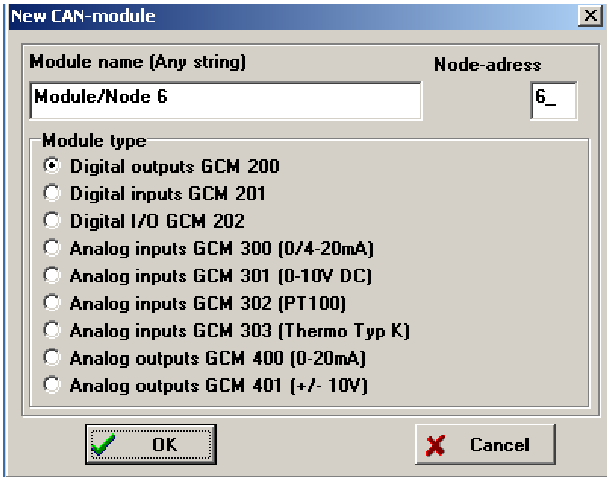
Use the button  or the menu option "configurating programs"/"CAN-modules" for this.
But you must have saved the started project already beforehand.
The program answers with the following window:



Now you can set up, configurate and delete CAN-modules in this program. You can define likewise input and output functions. You cannot carry out logical links - use the control program for this.

9.2 Creating CAN-module

After you have started the CAN-configuration program, first you will have
not any modules yet. If you call-up the menu option "module"/"new", press the button , then you receive a list of the modules which
are known at the moment:



It involves here the basic module-mask. If you insert extension modules of the type GCM 205 or GCM 206: these are adjusted later. Select here the basic module.

9.2.1 Field "Name of the module"

You may enter into the field "name of the module" what you want. Do not leave the field empty, but allocate a respective name like e.g.: "temperature measurement cold-store" or "message coverage control-cabinet". It is easier to find the modules then again.

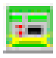
9.2.2 Field "Node-No.:"

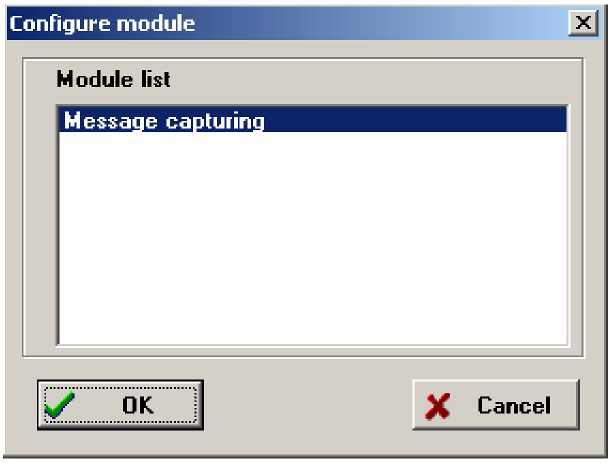
Enter the node number of the module into this field. The node number is adjusted at the module via the DIP-switches 1-5. (See above)

9.2.3 Field "Type of the module"

Select here the type of the basic module which you insert as module at this address.
To finish off, press "OK", and then the module is taken over into the project.

9.3 Configurating a CAN-module

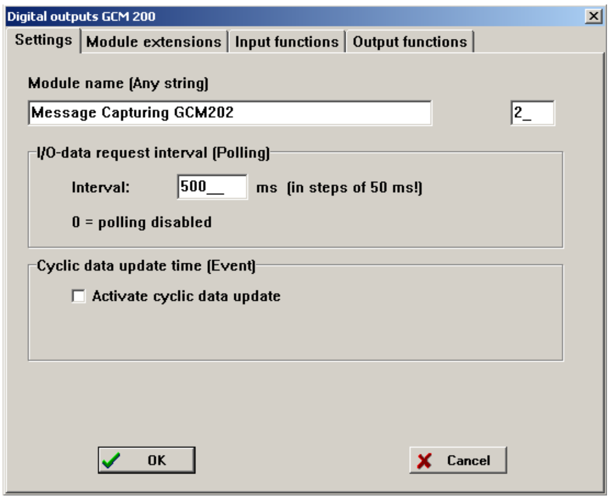
If you want to configurate a module it has first to be created via "module new" - see previous section.
If a module is created in the project, then further buttons appear in the editor and the
menu option "configurating module" is selectable.
The button  also calls up the following selection mask like "configurating module":



In this selection mask the names are displayed which you have given when creating the modules (therefore you should indicate another name!). Mark the module which you want to process and press "OK". Dependent on the module type, you receive then a configuration mask with different possibilities.

9.3.1 Module series GCM 200

Here you receive a configuration mask with 4 register cards:



The register cards have the following content.

9.3.1.1 Register card "settings"

Here the basic settings of the module are carried out. The fields have the following meaning:

Module name
You have seen this one already when creating the module. Here you can correct it, if necessary.

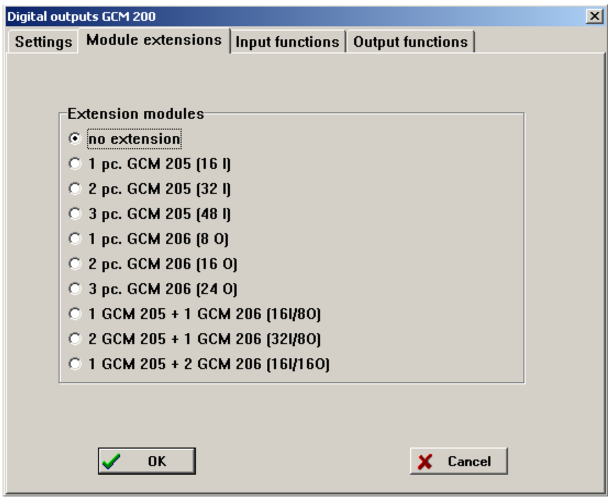
Node-address
Here you enter the node number (address) of the module.

Query interval...(poll-time)
There are different settings for when and how a module announces its I/O-data on the bus. With this poll-time you can instruct the master (thus the operating panel) to request cyclically in ms the I/O-data from the module. Here the master is the one that releases the data transfer. If you enter 0, then the data is not requested from the master in intervals, but the module must transmit its data itself.
In the most cases, you can maintain the setting of 500 ms.
The poll-time is therefore also used to carry out a refresh of the outputs of the module. The outputs, however, are always immediately written if they are changing. But a module could have wrong output data with power failure or a fault through EMC. However, it is achieved through the poll-time that after this time at the latest all outputs are positioned how they have to.

Cycle time (event)
You determine with this setting whether the module shall transmit independently its entire data within certain intervals. If you mark the field "activating automatic transmitting" with a cross, then you can adjust the requested interval in the field "cycle time" which appears then.
You can use both settings poll-time and cycle time at the same time; this is however only useful if the module has outputs (output-refresh!). We recommend using only one of the possibilities, whereby the poll-time should be used with output modules. But you can treat each module differently.

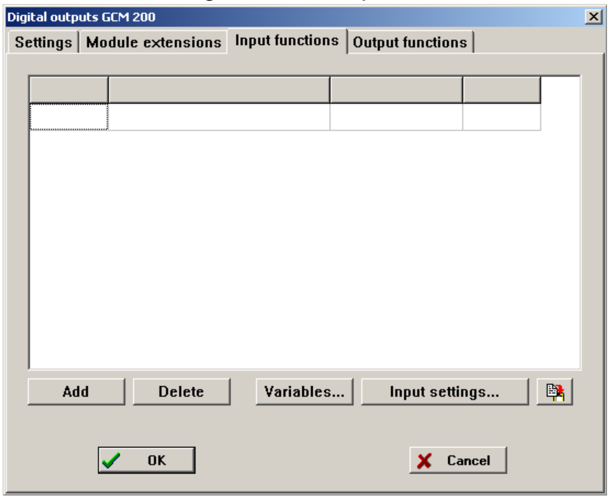
9.3.1.2 Register card "module removal"

Here you must indicate which type of expansion modules and how many of them are present at the basic module. You have here the complete selection:



9.3.1.3 Register card "input function"

In this register card you see a table which contains the input functions of the I/O-module. You can assign several functions to each input. Thus you could e.g. call up a message and count up a variable when a positive flank arrives and deliver the message again if a negative flank arrives. This function assignments are contained in the table of this register card, sorted according to input. And this is how the register card is presented:



The first column of the table contains the input number. Counting begins with 0. In the second column the function in a number is displayed encoded and column 3 contains information dependent upon column 2.
The numbers mean:

No./function column 2		Meaning column 3
0	Call up message	Message number
1	Deliver message	Message number
2	Call up image	Image number
3	Call up priority image	Image number
4	Deliver image	Image number
5	Display variables (input is transferred to variable)	Handle
6	Increment variable	Handle
7	Decrement variable	Handle
8	Delete variable	Handle

In column 4 it is indicated when the function is to be carried out:

Column 4	Function is ...
P	... carried out with positive flank
N	... carried out with negative flank
B	... carried out with both flanks

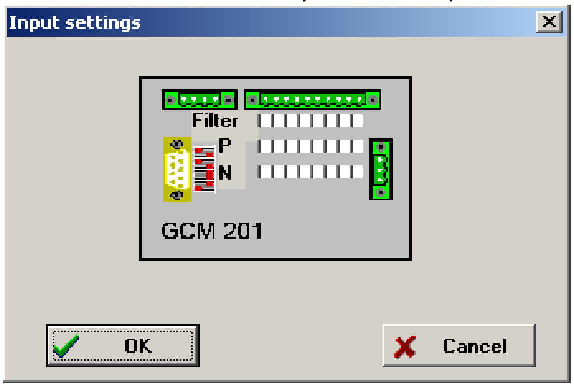
The indications in the table can be modified directly in the table or by double-click in the respective line. You receive then a setting-mask for the appropriate function (see "possible input functions" further below).

Button "add"
With this button you can call up an input mask with which you can determine the input function interactively. See "possible input functions"

Button "deleting"
With this button you can delete the function line marked currently in the table.

Button "variables"
This one calls up the table for processing the variables. This is described in more detail in the chapter "variables".

Button "input behaviour"
With this button you receive an input mask which shows the module with inputs and outputs:



For each input there are 3 options:

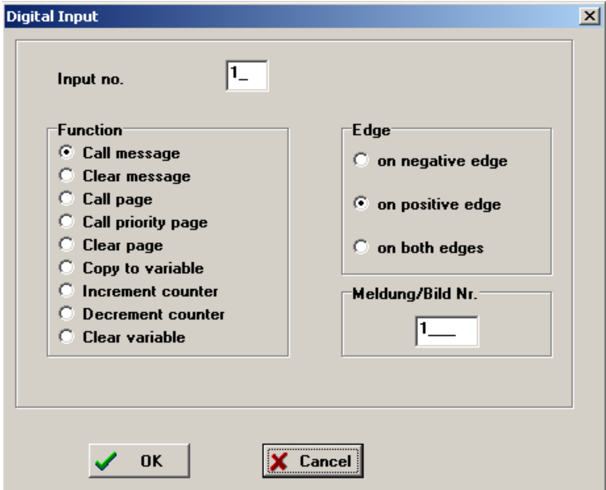
Filter (appears sometimes only as F)
Only signals are announced which are pending longer than 6 ms (de-buffer)

P (positive flank)
If a positive flank is recognized, then the module transmits immediately a data news to the bus, independent of the settings "poll-time" and "cycle-time" in the register "settings". Always all inputs are transferred.

N (negative flank)
If a negative flank is recognized then the module transmits immediately a data news to the bus, independent of the settings "poll-time" and "cycle-time" in the register "settings". Always all inputs are transferred.
We recommend that you activate all 3 options with the used inputs. Thus you receive the quickest possible signal transfer.

Button "copy input function" 
Imagine you have an input module with 40 inputs. You want to put now the functions "call up message at positive flank" and "deliver message at negative flank" on each input. 80 functions result.
This button enables you to transfer all functions defined on one input to other inputs; and thus also the number of the message (of the message or the variable handle) is increased automatically for the copy. This facilitates the work with extensive projects.

Possible input functions
If you double-click in the table or press the button "add", then you receive the entry mask for the input function. In this mask, you can determine very comfortably a function for an input. You can determine also several functions per input, this must however be done one after the other. The mask has the following setting possibilities:



Select first the input number in the field "input number". In the field "function", you adjust the function. Dependent on the function, the fields "flank" and "message/image-no." can modify. This can be seen when using it; this function supports you with the input.

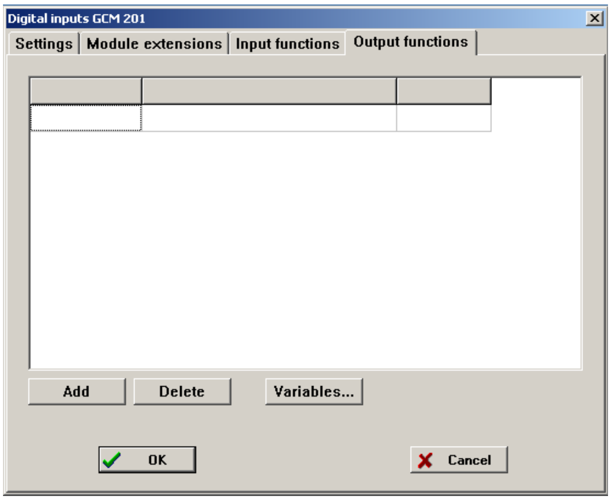
Numbering of the inputs
Counting is done starting from "0" (!), whereby "0" is the first input on the first module part starting from left.
It continues then with the expansion modules, where first the upper, then the lower terminal is counted at each module.

Example:
You have at a GCM 201 with a GCM 205 the inputs 0-7 at the basic module GCM 201, the inputs 8-15 at the upper terminal of the GCM 205 and the inputs 16-23 at the lower terminal of the GCM 205. The numbering is respectively from the left to the right.

For consideration
If you want to achieve that an image or a message is to be active as long as until the input is pending, then you have to parameterize two functions: One which calls up the message or image with positive flank and delivers again with negative flank.

9.3.1.4 Card "output functions"

The functions are displayed tabularly in this register card, which are assigned to the outputs of the module:

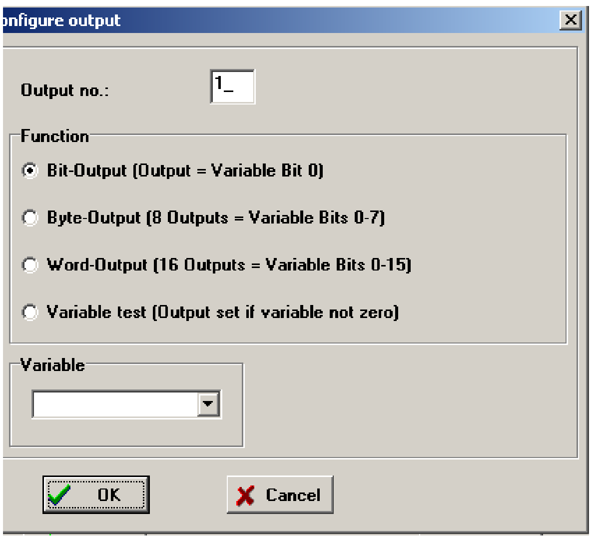


The output number is displayed in the first column. Column 2 contains the function number and column 3 the variable handle.

The following functions are available:

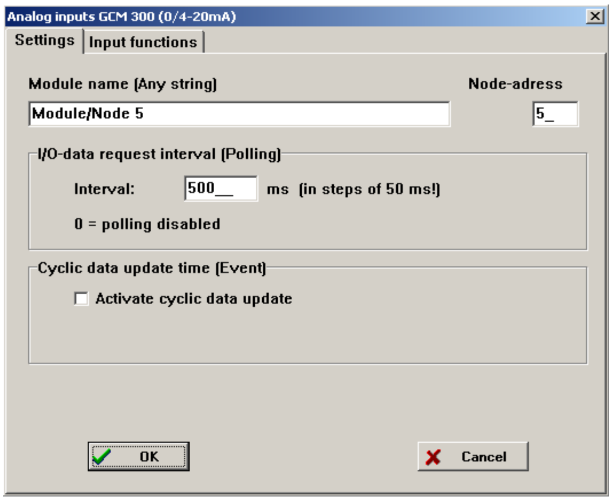
No.	Function
0	The bit No. 0 of the specified variable is copied into the output. That means the output is switched on if the variable has an uneven value.
1	The lowest-grade 8 bits of a variable are transferred into 8 outputs. Use for this function only the output numbers 0, 8, 16 and 24.
2	The lowest-grade 16 bits of a variable are transferred into the 16 outputs. Use only the output numbers 0, 8 and 16 for this.
3	The output is switched on as long as until the variable is unequal to 0.

You achieve the input mask for output functions with the button "add" or by double-click on a table line:



9.3.2 Module series GCM 300

This module series can process 4 equivalent inputs per module. If you configurate such a module it is done via the following mask:



9.3.2.1 Register card "settings"

These settings are the same as with the digital I/O-modules of the series GCM 200. Look up further above in this chapter.

9.3.2.2 Card "input functions"

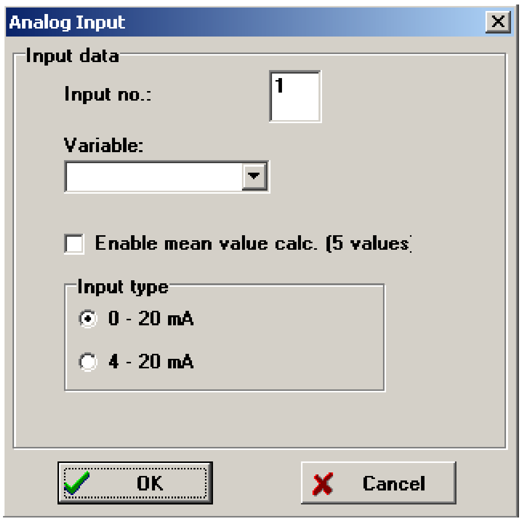
Here the input functions created by you are listed tabularly. You receive the input-parameterize mask "analogue input" by double-clicking into a function line in the table. But you don't have much selection; there are only few options. Only "transfer input in variable" is available as input function; you can only indicate to which variable the input value is to be transferred. You must carry out a scale in KOP, you cannot do it here. Also the monitoring of limit values can be made solely in the control program.
The inputs are numbered from 0-3 from left to right at the input terminal.

Button "add"
With this button you add a further input function. You receive the input-parameterize mask "analogue input" as shown further below.

Button "deleting"
The function line marked in the table is deleted.

Button "variables"
This button calls up the variable table that is generally known and described in chapter 7.

Parameterize mask "analogue input"
The configuration mask has - dependent upon the type of the module - the following image:



Depending upon type of the input module, you can select whether you want to activate an automatic average-value formation via the last 5 measuring values and whether you want to connect a 0-20 mA or a 4 - 20 mA transducer.
A scaling of the input values must be made via KOP. The following indications can be consulted for this:

Range of the inputs
Modules of **the series GCM300** deliver the following area as variable value:

0 = 0 µA; 1 = 5 µA; ...; 4000 = 20 mA
Therefore: **Iin= variable value x 5 µA**

Modules of **the series GCM300** deliver the following area as variable value:

0 = 0 mV; 1 = 2,5 mV; ...; 4000 = 10 V
Therefore: **Vin= variable value x 2,5 mV**

Modules of **the series GCM302** deliver the following area as variable value:

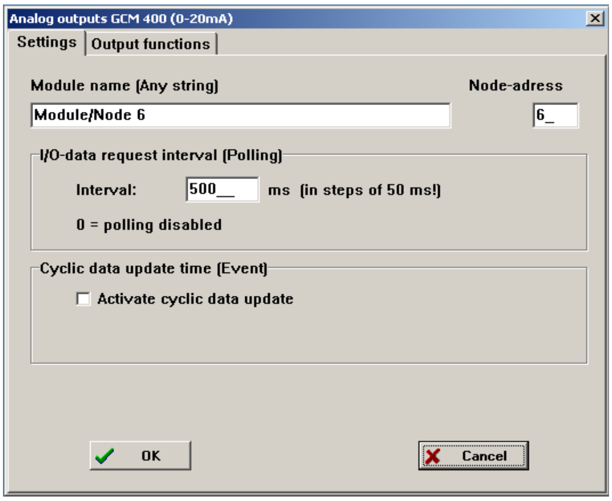
0 = -50°C; 1=-49,75°C; ...; 4000 = 950°C
Therefore: **Tin= variable value x 0,25°C - 50°C**

Modules of **the series GCM303** deliver as variable value

1600 = 0°C; 1601 = 0,5°C; ...; 3600 = 1000°C
Therefore: **Tin= (variable value-1600) x 0.5°C**

9.3.2.3 Module series GCM 400

There are 4 similar analogue outputs 0-20 mA or 0-10V each at the module series GCM 400.. These modules are configurated via the following mask:



The outputs of the module are always written at that time when they are changing. But you can determine additionally via the poll-time how often the outputs are to be written independent from modifications.

9.3.2.4 Register card "settings"

The settings in this register card are identical as with the modules GCM 200 and GCM 300. But there are still specific features:

Poll-time:
The poll-time is used here as output-refresh-interval. That means this time indicates in which intervals the output values of the module are transmitted from the master of the output module. The output module transmits as confirmation the output values as data telegram so that an operating panel-slave can likewise display the output values.

Cycle time
Only if an operating panel is connected as slave can the output data be of interest and then be continuously updated via the cycle time.

9.3.2.5 Card "output functions"

The output functions are listed in tabular form in this register card. Normally you have one function per output; namely the data output of a variable value to the output.
In the first column stands the output number, in the second the function number (here always 2, that means the data output of the lower-grade 16 bit of a variable) and in the third the handle of the variable. A double-click on a table line opens the configuration window for the appropriate line (see below)

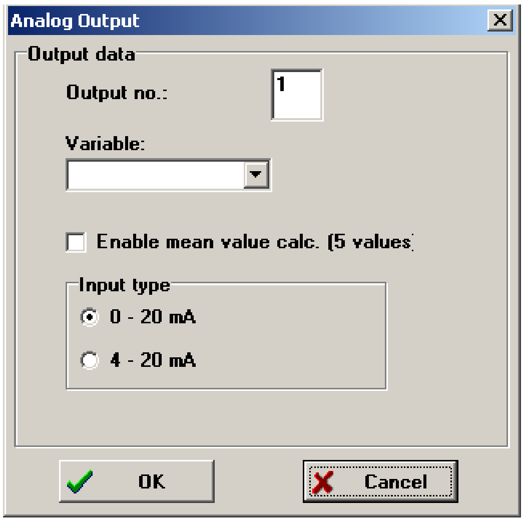
Manual operating panels

Button "add"
If you press this button you must add a new output function. You receive the mask "analogue output".

Button "deleting"
The function line marked in the table is deleted (and thus the assignment of a variable to the output).

Button "variables"
Here you arrive again in the known variable table.

Mask "analogue output"
Here you can assign a variable to an output:



A scaling of the data must be made with KOP. The information necessary for this is:

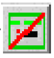
Range of the outputs
Modules of **the series GCM400** require the following area as variable value:

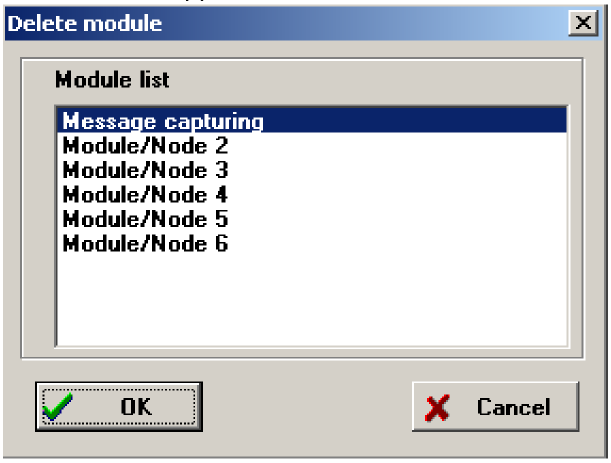
0 = 0 µA; 1 = 5 µA; ...; 4000 = 20 mA
Therefore:**Iout= variable value x 5 µA**

Modules of **the series GCM401** require the following area as variable value:

48 = -10 V; ...; 2048 = 0V; ...; 4048 = 10 V
Therefore: **Vout= (variable value-2048)x2,5 mV-10,24V**

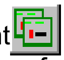
9.4 Removing CAN-module

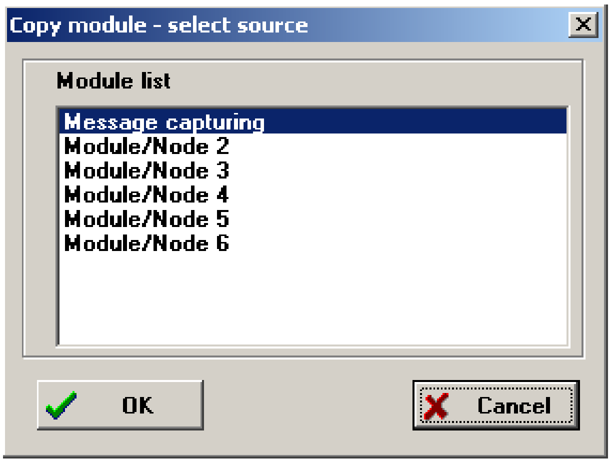
If you want to remove a module, then use the button or the menu entry  "Deleting module". Mark the module in the selection list which appears now



and press"OK".

9.5 Copying module

If you have several modules where you want to set up the same function (e.g. registration call ups or image call ups), then you can copy a module completely.
Select the button or the menu point  "copying module" for this. Then a list appears from which you can select the module that is to be copied:



Mark the requested module and click on "OK". Then a further mask is displayed. This corresponds to the configuration mask for the indicated module. Here the I/O-functions of the source module are already indicated in the tables and the input behaviour has also been taken over. You have now only to still adapt image or registration numbers and variables.