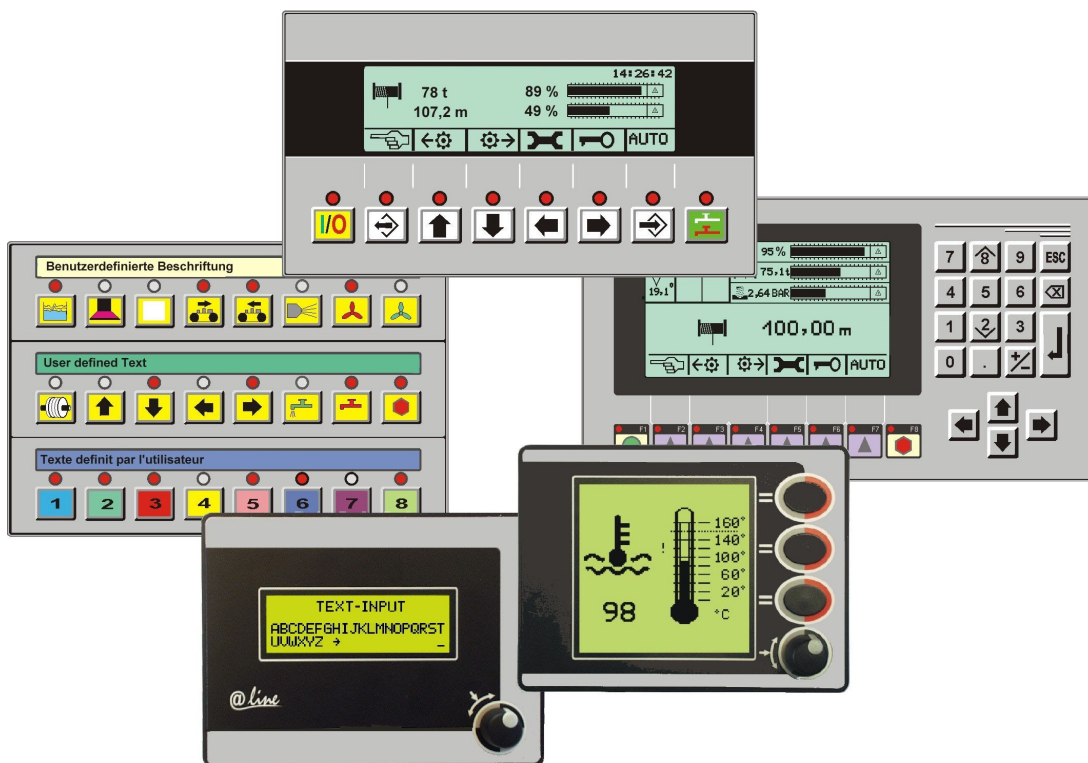




GRAF-SYTECO

Manual

Controlling with C



Document: H129A0
Status: released
Issued: June 2003

SYsteme **TE**chnischer **CO**mmunikation

Operating panel manual

1 Introduction

1.1 Integrated development environment GCE

For the generation of user control programs in C, an easy-to-manage development environment with ANSI-code editor and project management is supplied with the ITE software package.

To be able to use the development environment, version "D" or a higher version of the ITE software package must be installed. With the free basic version "C", the development environment only runs in the DEMO mode.

For the application of the integrated development environment, an operating panel of the AT series must be available. Devices of the ITS series do not support C programming.

GCE internally encapsulates the call-ups of the C compiler, linker and hex converter and displays fault messages which occur during project translation.

The interface to the operating system (TOS) of the AT operating panel is included in the scope of delivery as C-file template.

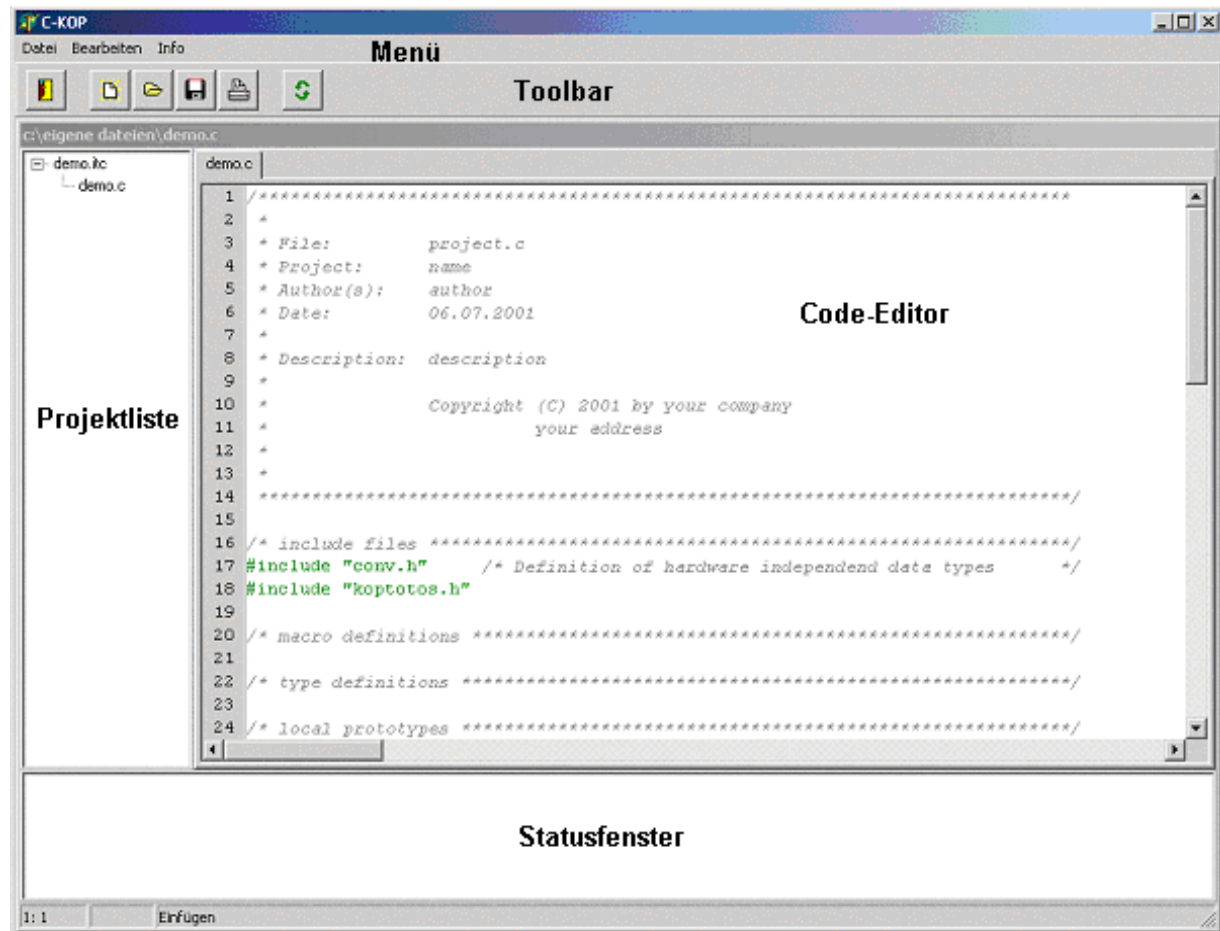
The GCE is started in the ITE editor. The start button for the GCE is only enabled if a device of the AT series is set in the preferences mask.

The current project name is used as GCE project name (e.g. if the ITE project name is machine.ITE, the GCE project name is machine.ITC).

If the project has not yet access to a TOS interface file including the frame code for the functions called up by the TOS, this file is generated on the basis of the Template.C file in the C:\programs\Graf_ITE\bin directory.

The file name of the automatically generated file is preset to <project name.c > (e.g. machine.C).

1.1.1 Interface



Operating panel manual

1.1.2 Features of the C-code editor

The C-code editor is an extended ANSI editor, which has been specially optimised for the requirements of programmers and which offers the following programming aids:

- *Syntax highlighting, i.e. control structures and key words are displayed in colour. This function is available for files with .c and .h extensions.*
- *Line numbers are automatically displayed. This option can be modified in the "Preferences" menu item.*
- *Possible line and column selection.*
- *Cutting, copying, inserting and deleting blocks.*
- *Searching and replacing.*
- *The size of the files to be edited is virtually unlimited.*

1.1.2.1 Line selection

To select lines, keep the **<Shift>** key pressed while you move the cursor with an arrow key.

1.1.2.2 Column selection

Press the **<SHIFT>+<ALT>+<arrow keys>** key combination to select columns. Now, you can cut, copy, delete or shift the selected columns.

Columns can also be selected by keeping the **<Alt>** key pressed while clicking and dragging the mouse.

```
54 void KOP_Cycle(void)
55 {
56 static int State;
57
58 switch(State)
59 {
60     case 1: break;
61     case 2: break;
62     case 3: break;
63     case 4: break;
64
65 }
66 }
```

Spaltenmarkierung

1.1.2.3 Select all

Press the **<Strg>+<a>** key combination to select the entire text.

1.1.2.4 Cut block

The **<Strg>+<x>** key combination cuts the selected block and copies it into the clipboard.

1.1.2.5 Copy block

The **<Strg>+<c>** key combination copies the selected block into the clipboard without cutting it.

1.1.2.6 Delete block

If a block has been selected, it can be deleted by simply pressing the **** key.

1.1.2.7 Insert block

If a text has been copied into the clipboard, it can be inserted at the cursor position using the **<Strg>-<v>** key combination.

1.1.2.8 Delete line

An entire line can be deleted by pressing the **<Strg>-<y>** key combination.

Operating panel manual

1.1.2.9 Delete line from the cursor position to the end of the line

If you want to delete the rest of a line starting from the cursor position, press **<Strg>-<t>**.

1.1.2.10 Insert new line

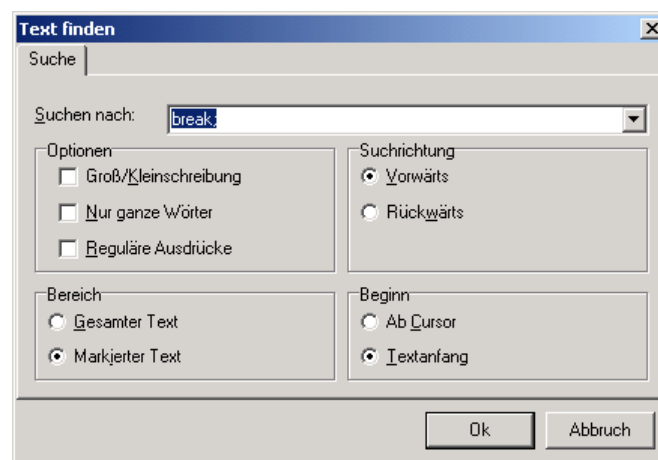
Insert a new line with **<Strg>-<n>**.

1.1.2.11 Undo function

If you have deleted something unintentionally or if you want to undo a modification, simply press **<Strg>-<z>**.

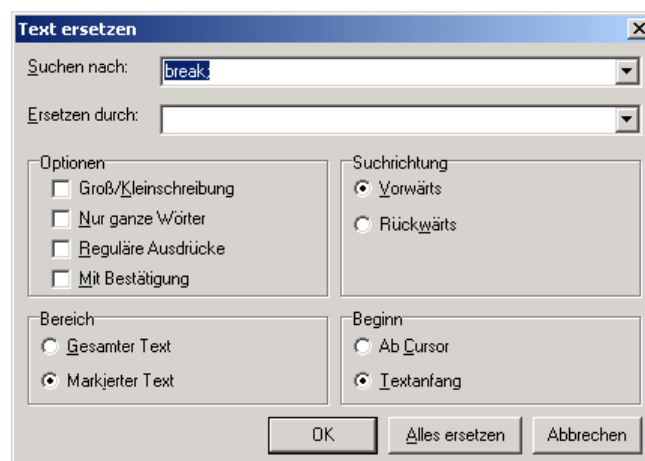
1.1.2.12 Open "Find text" dialogue

When working with editors, it is sometimes necessary to search the text for certain terms. The **<CTRL>+<f>** key combination opens the "Find text" dialogue by means of which you can search for the desired text passages.



1.1.2.13 Open "Replace text" dialogue

Sometimes you may want to replace a term throughout the entire text. For this purpose, press the **<CTRL>+<r>** key combination to call up the "Replace text" dialogue.



1.1.2.14 Find/replace next

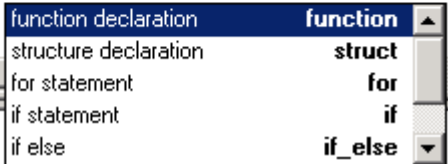
If you have started searching or replacing a particular term and the search or replacement process has been interrupted, you can continue with the **<CTRL>+<e>** key combination.

Operating panel manual

1.1.2.15 Insert code templates

Take the easy way out. Insert finished code templates for your conventional control structures in your source code by pressing the <CTRL>+<j> key combination and selecting the desired control structure from the list that shows up.

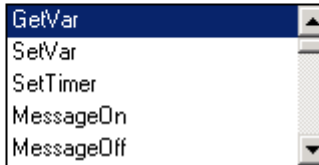
```
294 void KOP_Timer(void)
295 {
296 int Variable1;
297
298     Variable1 = GetVar( Analog_3 )
299
300     if( Variable1 > 24.8 )
301     {
302
303     }
304
```



1.1.2.16 Select system function from a list

Press the <ALT>+<j> key combination. A list appears including all system functions specified in the KOPTOTOS.H file. Select the desired function and press the Enter key.

```
294 void KOP_Timer(void)
295 {
296 int Variable1;
297
298     Variable1 =
299
300
301
302 }
303
```



Operating panel manual

1.1.2.17 Select variables from a list

As with the system functions, you can also select system and project variables from a list and directly integrate them in your source code. For this purpose, press the **<ALT>+<v>** key combination.

```
294 void KOP_Timer(void)
295 {
296 int Variable1;
297
298     Variable1 = GetVar(
299
300
301
302 }
303
```

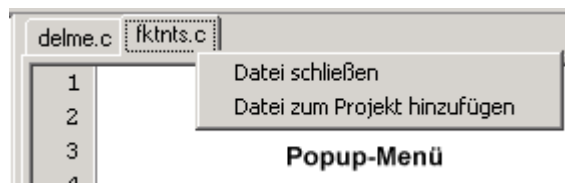
Analog_1	80
Analog_2	81
Analog_3	82
Analog_4	83
Analog_5	84

1.1.3 Features of the project file list

All project-related files must be included in the project list. This also refers to all customer-created and header files integrated by #include as well as to documentation files which are not compiled. When pressing the right mouse button, a POPUP menu shows up for editing the project list (e.g. for including existing files into the project list).



You also have the possibility to subsequently include an already open file in the project list. For this purpose, activate the respective editor window, position the mouse cursor on the index with the file name and press the *right* mouse key.



Operating panel manual

1.1.4 Features of the status window

The status window indicates the status of the current project translation. In the case of a fault which leads to the interruption of the compiler process, the respective fault messages are displayed in this status window. By clicking the mouse on an entry in this list, the cursor automatically jumps to the position in the source text which has caused the fault.

```
*** Übersetze c:\programme\graf_ite\tos\delme2.c...
*** Projekt wird gelinkt...
*** HEX-Datei wird erzeugt...
*** Erzeuge ATX-Datei(en)...
```

1: 1 Einfügen c:\programme\graf_ite\tos\delme2.c

```
*** Übersetze c:\programme\graf_ite\tos\delme2.c...
*** c:\programme\graf_ite\tos\delme2.c(62) E4062C: syntax error near `case'
```

1.2 Menu

1.2.1 Project menu

The project menu serves for the management of the currently loaded project. This menu includes menu items for opening, saving and compiling etc. as well as a direct opening function for the project names of the projects which have been opened last (maximum four).

Projekt	Datei	Bearbeiten	Einstellungen	Info
Öffnen				
Speichern				
Speichern unter				
Übersetzen				F9
Alle Dateien übersetzen				Umsch+F9
Treiber dazubinden...				
Beenden				
1 C:\PROGRAMME\GRAF_ITE\BIN\NEW.itc				

Operating panel manual

1.2.1.1 Open

Calls up the Windows "File-Open-Dialogue" to open a GCE project. GCE projects have the file extension .ITC.

Note:

This command is not available if the GCE project is started in the ITE editor or if the project has been opened by double clicking the .ITC file in the explorer.

1.2.1.2 Save

Saves the currently loaded project and overwrites the .ITC file.

1.2.1.3 Save as

Saves the currently loaded project under a new name.

Note:

This command is not available if the GCE project is started in the ITE editor or if the project has been opened by double clicking the .ITC-file in the explorer.

1.2.1.4 Compile

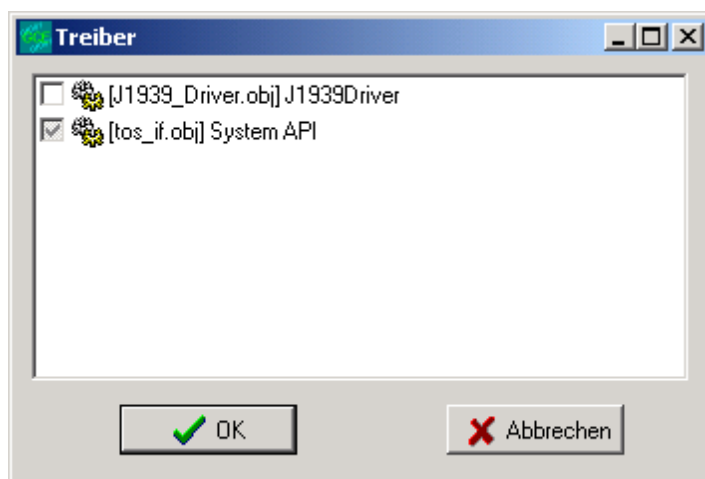
This function internally calls up the C compiler and the linker for the project compilation. If this menu item has been selected, only files which have been changed since the last compilation process (make) are compiled by the compiler. The C-compiler and the linker version are displayed in the status window. Faults that have been detected in the source text are also displayed in the status window. By double clicking a fault message, the code editor directly jumps to the position where the fault originates from.

1.2.1.5 Build all

This menu item functions like the "Compile" menu item. Here, however, all source files are compiled by the C-compiler (build).

1.2.1.6 Drivers to link

If a driver has been installed for a certain functionality in the operating panel and you would like to call up driver functions in your own user program, the object file (xxx.obj) of the driver must be integrated in the user program. This menu item calls up the following window where you can select the drivers to be linked. Simply tick the box with the respective driver.



Note:

The setting is project-dependent. Only link drivers in your project from which you call up functions in your user program.

If your project uses drivers and you want to forward your project to other users, you must ensure that the driver file (.OBJ) and the respective header file (.H) are forwarded together with the project. Otherwise the project cannot be "linked" by the user. For the installation of drivers, please refer to the See "Driver administration" on page 10.

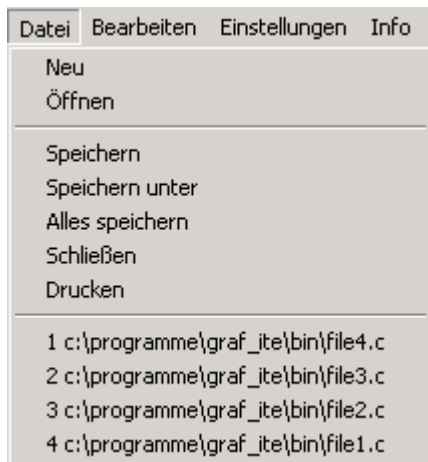
Operating panel manual

1.2.1.7 Exit

This menu item exits the program after you have been requested whether the currently loaded project should be saved.

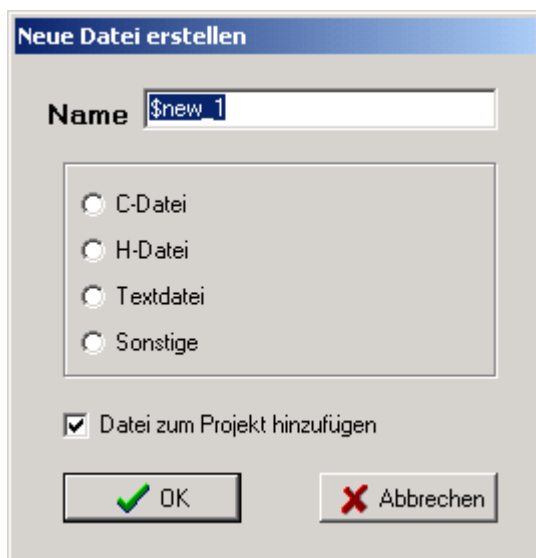
1.2.2 File menu

The file name contains menu items for the file management. Each menu item refers to the currently active file in the code editor. Even if several editing windows are open, only one is active. Apart from the menu items for saving, opening, etc. this menu also displays the file names of the files opened last (maximum four).



1.2.2.1 New

Creates a new file. You can select your desired file type in the displayed window.



Enter a file name with or without file extension. **The file name must always start with a letter (A-Z, a-z)!** Select the type of the new file and also whether the file is to be added to the project. If the file is added and the project is saved in a new path, the file is automatically copied to the new location.

Operating panel manual

1.2.2.2 Open

Opens an existing file.

1.2.2.3 Save

Saves the file which has been opened in the active editor window.

1.2.2.4 Save as

Saves the file which has been opened in the active editor window under a new name.

1.2.2.5 Save all

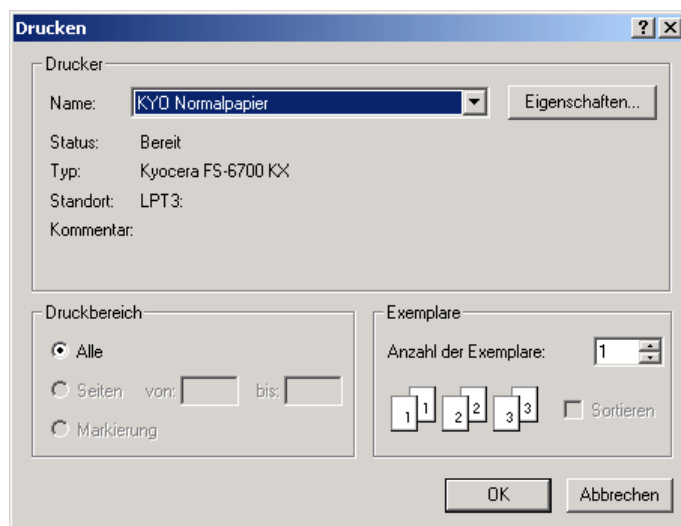
Saves all open files from all editor windows.

1.2.2.6 Close

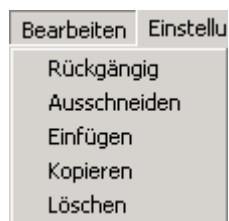
Closes the file which has been opened in the active editor window after you have been requested whether the file should be saved before closing. The file is not removed from the project!

1.2.2.7 Print

Prints the file which has been opened in the active editor window on the printer. The standard dialogue is called up.



1.2.3 Edit menu



1.2.3.1 Redo

Redoes the last entries.

1.2.3.2 Cut

Cuts the selected text and copies it into the clipboard.

Operating panel manual

1.2.3.3 Paste

Pastes the text from the clipboard into the active editor window at the cursor position.

1.2.3.4 Copy

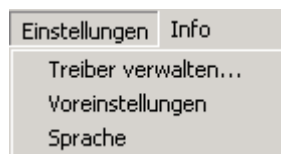
Copies the selected text into the clipboard.

1.2.3.5 Delete

Cuts the selected text without copying it into the clipboard.

1.2.3.6 Setup menu

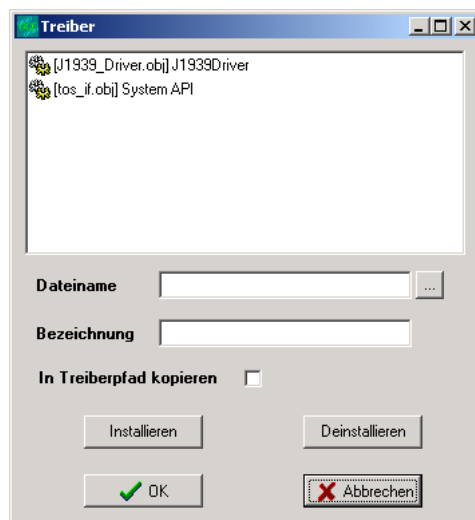
This menu contains the program settings.




1.2.3.7 Driver administration

If you do not forward the source text of program components written by you, but still want to make your functions available for the user, you must at least provide the compiled .OBJ file and the header file (.H) for the user. The .OBJ file which is generated by the C-compiler contains your code and the .H-file contains the prototypes of the functions which can be called up from your driver file by the user. The user must integrate this header file in his program and link the .OBJ file to his project via the #include command.

GRAF-SYTECO has developed specific drivers for certain applications which can be installed within this context (e.g. the J1939 driver for CAN communication with devices and aggregates which support this protocol). To ensure that functions can be called up from the driver and the linking process is executed without faults, the .OBJ file as well as the identically named header file must be installed in the user system. This process is carried out via the following dialogue:



Install driver

Enter the name and the path of the driver to be installed under "file name". The  button calls up the "File-Open" dialogue. As a prerequisite of the driver installation, both the .OBJ and the identically-named .H-file (e.g. J1939Driver.obj and J1939Driver.h) must be available.

Enter an arbitrary driver name in the designation field. This name only serves for clarity and has no further significance.

If the driver is to be copied into the driver directory (usually the directory C:\programs\Graf-ITE\C_Driver), tick the box behind the "Copy in driver path" option.

If "Copy in driver path" has not been activated, the driver is installed into the path indicated in the "File name"

Operating panel manual

field by pressing the "Installing" button. Otherwise it is first copied into the driver path and then installed.

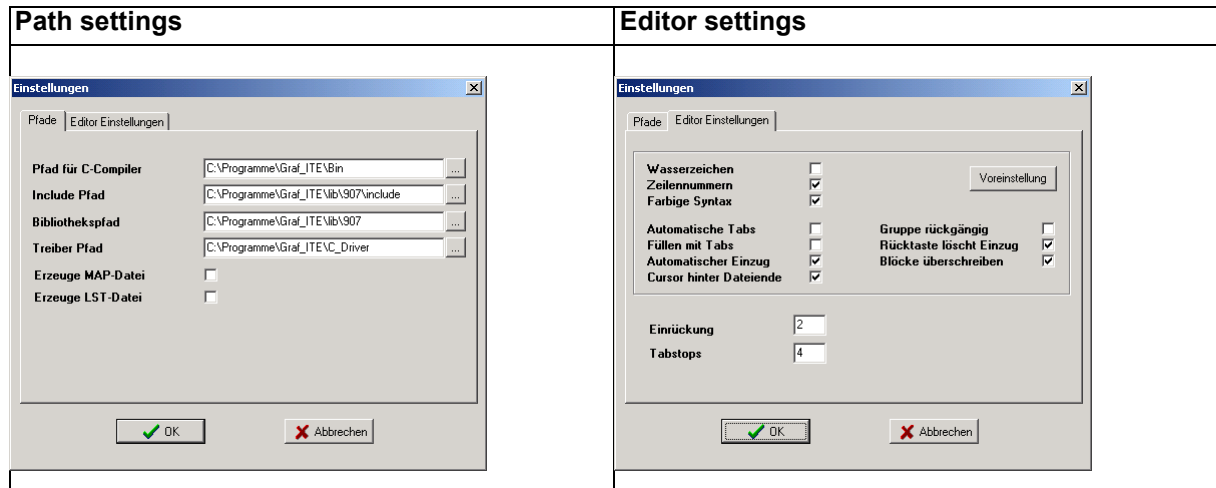
Uninstall driver

Select the driver you want to uninstall from the list and press the "Uninstall" button to remove the driver from the list. The related file is not deleted.

Please observe that projects using the driver cannot be linked any longer when the driver has been uninstalled!

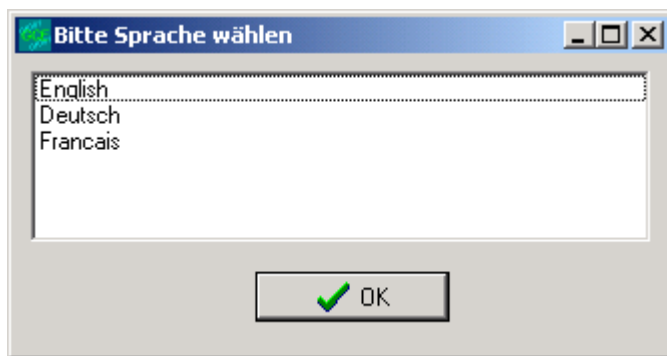
1.2.3.8 Presettings:

The setting mask allows you to make program-specific settings. Usually there is no need to change the settings unless paths are changed or you are not satisfied with the editor presettings.



1.2.3.9 Language

GCE has a multilingual design. In the below illustrated dialogue, you can select your desired language. Select the desired language and press the "OK" button. The program immediately switches to the new language.



Operating panel manual

Note:

If you call up the GCE in the ITE editor, GCE uses the language setting of the ITE editor. There, the modification of language settings has an immediate effect on all additional programs.

1.2.4 Information menu

The information menu only comprises the "Version" menu item which calls up the INFO.EXE program. This additional program displays version information concerning all software components installed.



1.3 Tool bar



The tool bar offers quick access to frequently used functions. The following buttons have been inserted for these functions.



Exit program
Exits the program.



Print file
Prints the file which is opened in the currently active editor window on the printer.



New file
Creates a new source file.



Make
Compiles modified source files and links the project.



Open file
Opens an existing source file.



Build
Compiles all project-related source files and links the project.



Save file
Saves the file in the currently active editor window.

Operating panel manual

2 The interface to the TOS

2.1 System files

To be able to execute customised C programmed control functions with the operating panels of the AT series, an interface to the user program has been implemented in the TOS operating system. This interface consists of four functions which are internally called up by the operating system of the operating panel. To visualise these functions for the user, the Template.c file, which contains the interface functions to the operating system, is installed in the <InstPfad\bin> path during the installation of the ITE editor package. If a new control project is created in the GCE, this file is copied into the project directory and renamed to the project name. This way, it becomes the central interface file for the operating system. In addition to this, three further important files are copied into the <InstPfad\C_Driver> directory when the editor package is installed:

2.1.1 KOP_IF.OBJ

Your control program is connected to the functions in the operating system of the operating panel by the KOP_IF.OBJ file. This file contains the code for all system functions described in the next chapter and is linked to your project. KOP_IF.OBJ thus represents the so-called application interface (API).

2.1.2 KOPTOTOS.H

The KOPTOTOS.H file contains the prototypes (declarations) of the system functions. In cases of doubt, please refer to this file if you require information about parameters and return values of the system functions.

2.1.3 CONV.H

The CONV.H file contains the data type definitions used in the API.

2.2 Interface functions

To enable user-specific user programs to run on the operating panel, four interface functions must be available in the user program, which are called up depending on the operating status of the operating panel. The names of these interface functions as well as their parameters and return values are predefined. All four functions must always be available in the user program, even if no user-specific program components are executed.

2.2.1 Initialisation function

The initialisation function is called up after the TOS has been started and before the cyclic or time-controlled function is activated. This way, e.g., global variables can be preassigned or other preparation tasks can be executed. Project variables can now be accessed and initialised to standard values. The access to all system functions is also possible at this stage.

Declaration:	void KOP_Init (void)
Parameter:	none
Return value	The return parameter is an integer type and indicates the interval during which the time-controlled function is to be called up. The interval time must be specified in steps of 10 msec. If the time-controlled function is not required, 0 must be used as return value.

```
int KOP_Init(void)
{
    return(10); /* call up time-controlled processing every 10*10 msec = 100 msec */
}
```

Operating panel manual

2.2.2 Cyclically-processed user program

This function is called up in intervals of approx. 200 msec. As the interval time is neither forecastable nor constant, real-time processing is not possible.

Generally, this function is always called up if the operating system has no other functions to process (idle state). All complicated, non-time-critical functions can be integrated into this process. The function must be programmed in a way which ensures that it can be left as quickly as possible. Therefore, no long-time loops must be programmed as otherwise the operating system is slowed down!

Declaration:	void KOP_Cycle (void)
Parameter:	none
Return value	none

Example:

```
void KOP_Cycle(void)
{
    .../* User program */
}
```

2.2.3 Time-controlled user program

This function is used whenever the processing of the user program has to be ensured within a temporally constant interval.

The time calculation of the interval is based on the return value of the KOP_Init() initialisation function and can be set in steps of 10 milliseconds.

The time-controlled user program must also be left as quickly as possible to ensure that the operating system is not slowed down. Only the most important functions should be executed, as otherwise the device performance is reduced.

The processing must not take longer than the adjusted interval time!

Declaration:	void KOP_Timer (void)
Parameter:	none
Return value	none

Example:

```
void KOP_Timer(void)
{
    ... /* time-critical components of the user program */
}
```

Operating panel manual

2.2.4 CAN receive event

The event handling routine for the reception of CAN telegrams is called up whenever a CAN telegram is received on an ID of the multimaster channels or on one of the dynamically enabled identifiers, which are allocated by the operating panel.

Identifiers are dynamically enabled in the KOP_Init() initialisation function by calling up one of the system functions CANEnableRxId or CANEnableRxMask.

This event handling routine must also be left as quickly as possible in order not to impair the device performance.

Declaration:	void KOP_CAN_Event (...)
Parameter:	uint32 RxId receive identifier, identifier format and channel int len number of received data bytes (0..8) uint8 D0 data byte D0 uint8 D1 data byte D1 uint8 D2 data byte D2 uint8 D3 data byte D3 uint8 D4 data byte D4 uint8 D5 data byte D5 uint8 D6 data byte D6 uint8 D7 data byte D7
Return value	None

After reception of the telegram and the following call-up of this function, the eight data bytes transferred via the CAN bus are available in the parameters D0 to D7 for a control program evaluation. If a longer period of time is required for the processing, the received data should be copied into an intermediate buffer which is then evaluated by the KOP_Cycle() function.

Note:

The RxId parameter contains a code which indicates whether the telegram has a 29-bit ID and which specifies the interface that transmitted the telegram.

Operating panel manual

Example:

```
void KOP_CAN_Event(uint32 RxId, int len, uint8 D0, uint8 D1, uint8 D2, uint8 D3,
                  uint8 D4, uint8 D5, uint8 D6, uint8 D7)
{
    /* request for 29 bit ID */
    if (RxId & USE_29BIT_ID)
    {
        ... /* handling of telegrams with 29-bit identifier */
    }

    /* request the interface which transmitted the telegram*/
    if (RxId & USE_CAN_1)
    {
        ... /* telegram was transmitted by the second CAN interface */
    }
    else
    {
        ... /* telegram was transmitted by the standard CAN interface*/
    }
}
```

In order to decode the identifier which has been externally addressed, the status bits (interface and 29-bitID flag) must be blanked out. Only then can the identifier be evaluated.

```
/* blanking out status bits */
RxId = RxId & 0x1FFFFFFFUL;
if (RxId == 1024)
{
    ... /* evaluation for identifier 1024 */
}
}
```

Operating panel manual

3 The API operating system reference

3.1 Overview

This manual thematically structures the functions of the API operating system in accordance with the types of application. For each type of application, various functions are available. The following table is to give you an overview of the system functions. In the online version of this documentation, a description of the individual functions can be obtained by clicking the function name in the "Function name" column. The very right column indicates the TOS version from which the respective function is implemented in the TOS. Basically all functions are available in the TOS and can be called up. Functions which have not yet been implemented are installed as empty templates in the TOS and marked in the last table column with N/A.

Task	Function name	TOS version
3.2.1 Functions for working with variables		
3.2.1.1 Getting variable value	GetVar	
3.2.1.2 Setting variable value	SetVar	
3.2.1.7 Does a variable run as hour-counter?	IsHourCounterOn	
3.2.1.8 Starting variable as hour-counter	HourCounterStart	
3.2.1.9 Stopping variable as hour-counter	HourCounterStop	
3.2.1.10 Transmitting project variables to a PLC	SendVarToPLC	
3.2.1.11 Transmitting project variable to a different operating panel	SendVarToTerminal	
3.2.1.3 Getting scaled value of a J1939 variable	GetVarJ1939Scaled	
3.2.1.4 Setting value of a scaled J1939 variable	SetVarJ1939Scaled	
3.2.1.5 Getting unscaled value of a J1939 variable	GetVarJ1939binary	
3.2.1.6 Setting value of an unscaled J1939 variable	SetVarJ1939binary	
3.2.2 Functions for working with pages		
3.2.2.1 Getting page scrolling time	GetPageAutoScrollTime	
3.2.2.2 Setting page scrolling time	SetPageAutoScrollTime	
3.2.2.3 Getting number of the currently displayed page	GetCurrentPageShown	
3.2.2.4 Determining whether a page is active	IsPageOn	
3.2.2.5 Determining whether a priority page is active	IsPrioPageOn	
3.2.2.6 Activating a page	PageOn	
3.2.2.7 Deactivating a page	PageOff	
3.2.2.8 Activating a priority page	PrioPageOn	
3.2.2.9 Deactivating a priority page	PrioPageOff	
3.2.2.10 Activating a menu page	MenuPageOn	
3.2.2.11 Deactivating the last menu pages/structure	MenuPagesOff	N/A
3.2.3 Functions for working with messages		
3.2.3.1 Getting message scrolling time	GetMessageAutoScrollTime	
3.2.3.2 Setting message scrolling time	SetMessageAutoScrollTime	

Operating panel manual

3.2.3.3 Getting number of the currently displayed message	GetCurrentMessageShown	
3.2.3.4 Determining whether a message is active	IsMessageOn	
3.2.3.5 Activating a message	MessageOn	
3.2.3.6 Deactivating a message	MessageOff	
3.2.4 Functions for working with keys		
3.2.4.1 Checking whether any key is pressed	IsAnyKeyDown	
3.2.4.2 Checking whether a particular key is pressed	IsKeyDown	
3.2.4.3 Checking whether a particular key has been pressed	IsKeyPressedNew	
3.2.4.4 Simulating a menu key stroke	SimulateMenuKey	
3.2.4.5 Simulating an ASCII input	SimulateASCIIKey	
3.2.4.6 Positioning cursor on the entry field/menu item	SetCursorPos	N/A
3.2.4.7 Requesting cursor position	GetCursorPos	N/A
3.2.4.8 Getting menu item index	GetCurrentMenuIndex	
3.2.4.9 Getting "soft key" mask	GetSoftkeyMask	
3.2.4.10 Setting "soft key" mask	SetSoftkeyMask	
3.2.4.11 Getting number of the key extension blocks	GetKeyBlockCount	
3.2.5 Functions for working with LEDs		
3.2.5.1 Switching on LEDs	LedOn	
3.2.5.2 Switching off LEDs	LedOff	
3.2.5.3 Checking whether a LED is switched on	IsLedOn	
3.2.6 Functions for working with timers		
3.2.6.1 Starting/setting a timer	SetTimer	
3.2.6.2 Requesting whether a timer is operating	IsTimerOn	
3.2.6.3 Requesting whether a timer has expired	IsTimerOff	
3.2.7 Functions for working with inputs/outputs		
3.2.7.1 Switching internal message output on	OutputOn	
3.2.7.2 Switching internal message output off	OutputOff	
3.2.7.3 Requesting message output	IsOutputOn	
3.2.7.4 Requesting LIGHT-IN input	IsInputOn	
3.2.7.5 Requesting internal digital input	GetInternalDI	
3.2.7.6 Requesting internal digital output	GetInternalDO	
3.2.7.7 Switching internal digital output	SetInternalDO	
3.2.7.8 Getting internal analogue input	GetInternalAI	
3.2.7.9 Getting input byte 0	GetDIB0	
3.2.7.10 Getting input byte 1	GetDIB1	
3.2.7.11 Getting input word 0	GetDIW0	
3.2.7.12 Getting output byte 0	GetDOB0	

Operating panel manual

3.2.7.13 Writing output byte 0	SetDOB0	
3.2.7.14 Getting output byte 1	GetDOB1	
3.2.7.15 Writing output byte 1	SetDOB1	
3.2.7.16 Getting output word 0	GetDOW0	
3.2.7.17 Writing output word 0	SetDOW0	
3.2.7.18 Getting short-circuit status	GetShortcutState	
3.2.8 Fast 2-channel counters		
3.2.8.1 Getting counter value 1	GetCount1	N/A
3.2.8.2 Writing counter value 1	SetCount1	N/A
3.2.8.3 Getting counter value 2	GetCount2	N/A
3.2.8.4 Writing counter value 2	SetCount2	N/A
3.2.8.5 Getting pre-selection 1 for counter 1	Get_C1_Preset1	N/A
3.2.8.6 Writing pre-selection 1 for counter 1	Set_C1_Preset1	N/A
3.2.8.7 Getting pre-selection 2 for counter 1	Get_C1_Preset2	N/A
3.2.8.8 Writing pre-selection 2 for counter 1	Set_C1_Preset2	N/A
3.2.8.9 Getting pre-selection 1 for counter 2	Get_C2_Preset1	N/A
3.2.8.10 Writing pre-selection 1 for counter 2	Set_C2_Preset1	N/A
3.2.8.11 Getting pre-selection 2 for counter 2	Get_C2_Preset2	N/A
3.2.8.12 Writing pre-selection 2 for counter 2	Set_C2_Preset2	N/A
3.2.9 Access to the CAN interfaces		
3.2.9.2 Setting CAN-bus rate	SetCAN0BusrateIndex SetCAN1BusrateIndex	
3.2.9.3 Getting CAN-bus rate	GetCAN0BusrateIndex GetCAN1BusrateIndex	
3.2.9.4 Getting standard transmission identifier	GetCanTxId	
3.2.9.5 Writing standard transmission identifier	SetCanTxId	
3.2.9.6 Getting standard receive identifier	GetCanRxId0	
3.2.9.7 Writing standard receive identifier	SetCanRxId0	
3.2.9.8 Getting multimaster receive channels	GetCanRxId1 GetCanRxId2 GetCanRxId3 GetCanRxId4 GetCanRxId5 GetCanRxId6 GetCanRxId7	
3.2.9.9 Writing multimaster receive channels	SetCanRxId1 SetCanRxId2 SetCanRxId3 SetCanRxId4 SetCanRxId5 SetCanRxId6 SetCanRxId7	
3.2.9.10 New initialisation of the CAN interfaces	RestartCanSystem	

Operating panel manual

3.2.9.11 Checking CAN-bus status	Get_CAN_Status	N/A
3.2.9.12 Transmitting a CAN telegram	SendCANTelegram	
3.2.9.13 Dynamically enabling a receive identifier	CANEnableRxId	
3.2.9.14 Disabling dynamically-assigned receive identifiers	CANDisableRxId	
3.2.9.15 Setting enabling mask for receive identifier	CANEnableRxMask	
3.2.9.16 Removing enabling mask for receive identifier	CANDisableRxMask	
3.2.9.17 Checking whether the device runs as CAN master	GetMASTERFlag	
3.2.9.18 Monitoring the CANopen transmission	SDOStatus	N/A
3.2.9.19 Getting SELECAN device number	GetSELECANNodeNo	
3.2.9.20 Writing SELECAN device number	SetSELECANNodeNo	
3.2.10 Functions for actuating GCM-Can modules		
3.2.10.1 Requesting external digital input	GetGCM_DI	
3.2.10.2 Requesting external analogue input	GetGCM_AI	
3.2.10.3 Setting external digital output	SetGCM_DO	
3.2.10.4 Setting external analogue output	SetGCM_AO	
3.2.10.5 Requesting module status	GetGCM_Status	N/A
3.2.11 Access to the serial interfaces		
3.2.11.2 Getting baud rate	GetSerial0BaudIndex GetSerial1BaudIndex	
3.2.11.3 Setting baud rate	SetSerial0BaudIndex SetSerial1BaudIndex	
3.2.11.4 Getting telegram setting	GetSerial0FrameSetting GetSerial1FrameSetting	
3.2.11.5 Making telegram setting	SetSerial0FrameSetting SetSerial1FrameSetting	
3.2.11.6 New initialisation of the serial interfaces	RestartComSystem	
3.2.11.8 Status request	GetStatSerial0 GetStatSerial1	
3.2.11.9 Getting received characters	GetCharSerial0 GetCharSerial1	
3.2.11.10 Acknowledging received character	SetStatSerial0 SetStatSerial1	
3.2.11.11 Overwriting received character	SetCharSerial0 SetCharSerial1	
3.2.11.12 Serial transmission	SendToSerial	
3.2.12 Functions for display settings		
3.2.12.1 Getting contrast	GetContrastIndex	
3.2.12.2 Setting contrast	SetContrastIndex	
3.2.12.3 Getting brightness	GetBrightnessIndex	
3.2.12.4 Setting brightness	SetBrightnessIndex	

Operating panel manual

3.2.12.5 Getting brightness value for key illumination	GetELFOILState	
3.2.12.6 Setting brightness value for key illumination	SetELFOILState	
3.2.13 Graphical functions		
3.2.13.4 Drawing an individual pixel	DrwPixel	
3.2.13.5 Drawing a rectangle without filling	DrwRect	
3.2.13.6 Drawing a rectangle with filling	DrwRectFilled	
3.2.13.7 Drawing a line	DrwLine	
3.2.13.8 Drawing a circle	DrwCircle	
3.2.13.9 Drawing a filled circle	DrwCircleFilled	
3.2.13.10 Drawing a filled ellipse	DrwEllipseFilled	
3.2.13.11 Output text	DrwText	
3.2.13.12 Moving area on the display	MoveArea	
3.2.13.13 Initialising pointer instrument	DefineAnalogView	
3.2.13.14 Displaying/updating pointer instrument	ShowAnalogView	
3.2.13.15 Deleting pointer instrument	DeleteAnalogView	
3.2.14 Functions for the touch screen		
3.2.14.1 Getting X position of the currently touched position	Get_X_Move	
3.2.14.2 Getting Y position of the currently touched position	Get_Y_Move	
3.2.14.3 Getting X position of the first touched position	Get_X_Down	
3.2.14.4 Getting Y position of the first touched position	Get_Y_Down	
3.2.14.5 Getting absolute analogue value of X position	Get_X_Value	N/A
3.2.14.6 Getting absolute analogue value of Y position	Get_Y_Value	N/A
3.2.14.7 Getting calibration value for X	Get_X_Cali	N/A
3.2.14.8 Getting calibration value for Y	Get_Y_Cali	N/A
3.2.14.9 Getting tolerance value	GetTouchTol	
3.2.14.10 Setting tolerance value	SetTouchTol	
3.2.15 Functions for edge evaluation		
3.2.15.1 Evaluation of both edges	AnyEdge	
3.2.15.2 Evaluation of the rising edge	RisingEdge	
3.2.15.3 Evaluation of the falling edge	FallingEdge	
3.2.16 Functions for the video input		
3.2.16.1 Defining camera window	CameraWindowSet	
3.2.16.2 Switching on camera window	CameraWindowOn	
3.2.16.3 Switching off camera window	CameraWindowOff	
3.2.17 Other functions		
3.2.17.1 Requesting current device status	GetCurrentDeviceStatus	
3.2.17.2 Requesting error status	Get_ERROR_Status	

Operating panel manual

3.2.17.3 Requesting time zone	GetCurrentTimezone	
3.2.17.4 Setting time zone	SetCurrentTimezone	
3.2.17.5 Getting configuration of the printer interface	GetPrinterInterfaceIndex	
3.2.17.6 Configuring printer interface	SetPrinterInterfaceIndex	
3.2.17.7 Checking which PLC driver has been loaded	GetPLCDriverNo	
3.2.17.8 Getting BIOS version	GetBIOSVersion	
3.2.17.9 Getting TOS version	GetTOSVersion	
3.2.17.10 Getting project version	GetUSERVersion	
3.2.17.11 Getting initialisation flags	GetInitFlags	
3.2.17.12 Writing initialisation flags	SetInitFlags	
3.2.17.13 Saving setup data in the project FLASH	SaveToFlash	
3.2.18 Calling up device function via CAN call-up convention	ExecCANTelegramInternal	
3.2.19 Flash functions		
3.2.19.1 Testing whether data can be saved in the Flash memory.	FlashTest	
3.2.19.2 Getting data from the user Flash	FlashRead	
3.2.19.3 Writing data into the user Flash	FlashWrite	

Operating panel manual

3.2 Description of the system functions

3.2.1 Functions for working with variables

3.2.1.1 Getting variable value

With the GetVar function, a variable can be retrieved which was created in the ITE project.

Declaration:	long GetVar (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	long value of the variable (32-bit signed integer)

Example:

```
VarValue = GetVar(125);
```

3.2.1.2 Setting variable value

With the SetVar function, the value of a variable which was created in the ITE project can be retrieved.

Declaration:	void SetVar (uint16 __Handle, long __value)
Parameter:	uint16 __Handle number (handle) of the variable long __Value value to which the variable is to be set
Return value	none

Example:

```
SetVar(100, VarValue+2);
```

3.2.1.3 Getting scaled value of a J1939 variable

With the GetVarJ1939Scaled function, the scaled value of a J1939 variable which was created in the ITE project can be retrieved. Further details on this variable type are available in the J1939 Communication manual.

Declaration:	long GetVarJ1939Scaled (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	long value of the variable (32-bit signed integer)

Example:

```
Value = GetVarJ1939Scaled(Var_0_EEEF_3);
```

3.2.1.4 Setting value of a scaled J1939 variable

With the SetVarJ1939Scaled function, a J1939 variable can be set to a scaled value.

Declaration:	void SetVarJ1939Scaled (uint16 __Handle, long __value)
Parameter:	uint16 __Handle number (handle) of the variable long __Value value to which the variable is to be set
Return value	none

Example:

```
SetVarJ1939Scaled(Var_0_F004_3, 1000)
```

Operating panel manual

3.2.1.5 Getting unscaled value of a J1939 variable

With the GetVarJ1939Scaled function, the unscaled value of a J1939 variable which was created in the ITE project can be retrieved. Further details on this variable type are available in the J1939 Communication manual.

Declaration:	long GetVarJ1939binary (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	long value of the variable (32-bit signed integer)

Example:

```
Value = GetVarJ1939binary(Var_0_EEEF_3);
```

3.2.1.6 Setting value of an unscaled J1939 variable

With the SetVarJ1939Scaled function, a J1939 variable can be set to an unscaled value.

Declaration:	void SetVarJ1939binary (uint16 __Handle, long __value)
Parameter:	uint16 __Handle number (handle) of the variable long __Value value to which the variable is to be set
Return value	none

Example:

```
SetVarJ1939binary(Var_0_F004_3, 1000)
```

3.2.1.7 Does a variable run as hour-counter?

It is possible to make a variable count up second by second. It can be requested with this function whether the function is activated for a particular variable.

Declaration:	int IsHourCounterOn (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	int == 0 if the variable does not count up <> 0 if the variable counts up

Example:

```
if (IsHourCounterOn(125))  
{  
... /* user program in case the variable counts up */  
}
```

Operating panel manual

3.2.1.8 Starting variable as hour-counter

With this function, internal ITE project variables can be used as chronometers with a second-based operating mode. If this function is called up, the operating system counts up the indicated variable by one every second until the HourCounterStop function is activated for this variable.

If the device is switched off, these hour-counters, of course, stop operating.

Several variables may be simultaneously active as hour-counters.

Declaration:	void HourCounterStart (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	none

Example:

```
HourCounterStart(122);
```

3.2.1.9 Stopping variable as hour-counter

With this function, a variable which has been activated as hour-counter via HourCounterStart(...) can be stopped from counting up.

Declaration:	void HourCounterStop (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	none

Example:

```
HourCounterStop(122);
```

3.2.1.10 Transmitting project variables to a PLC

With this function, the value of a project variable can be transmitted to a connected PLC. The same function is enabled as with the entry of a nominal value.

The "REPORT VALUE" telegram is transmitted via the standard CAN interface.

Declaration:	void SendVarToPLC (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	none

Example:

```
SendVarToPLC(100);
```

3.2.1.11 Transmitting project variable to a different operating panel

Operating panels can be interconnected via the CAN bus.

This function is used to exchange variables. The CAN bus transmits the "SET VALUE" telegram. Information on the telegram structure are available in the Communication manual.

Declaration:	void SendVarToTerminal (uint16 __Handle)
Parameter:	uint16 __Handle number (handle) of the variable
Return value	none

Example:

```
SendVarToTerminal(10);
```

Operating panel manual

3.2.2 Functions for working with pages

3.2.2.1 Getting page scrolling time

With this function, the page scrolling time can be retrieved in seconds. The scrolling time indicates the time which expires until the operating panel displays the next page of the page stack.

Declaration:	int GetPageAutoScrollTime (void)
Parameter:	none
Return value	int time in seconds

Example:

```
iPageScrollTime = GetPageAutoScrollTime ();
```

3.2.2.2 Setting page scrolling time

With this function, the page scrolling time can be set in seconds. The scrolling time indicates the time which expires until the operating panel displays the next page of the page stack.

Declaration:	void SetPageAutoScrollTime (int __Secs)
Parameter:	int __Secs time in seconds (0=scrolling off)
Return value	int time in seconds

Example:

```
SetPageAutoScrollTime (2); /* scrolling every 2 seconds */
```

Accept:

Immediately. After a re-start only if SaveToFlash has been executed.

3.2.2.3 Getting number of the currently displayed page

The function indicates the number of the currently displayed page as integer.

Declaration:	int GetCurrentPageShown (void)
Parameter:	none
Return value	int number of the currently displayed page

Example:

```
iPageNumber = GetCurrentPageShown();
```

Operating panel manual

3.2.2.4 Determining whether a page is active

A page is active if it has been activated via the CAN or the control program. This does not necessarily imply that it is visible on the display.

Declaration:	int IsPageOn (int __PageNo)
Parameter:	int __PageNo number of the page
Return value	int == 0: page is not active <> 0: page is active

Example:

```
if (IsPageOn(25))
{
... /* user program if the page is active */
}
```

3.2.2.5 Determining whether a priority page is active

A priority page is active if it has been activated via the CAN or the control program. This does not necessarily imply that it is visible on the display.

Declaration:	int IsPrioPageOn (int __PageNo)
Parameter:	int __PageNo number of the page
Return value	int == 0: page is not active <> 0: page is active

Example:

```
if (IsPrioPageOn(25))
{
... /* user program if the priority page is active */
}
```

3.2.2.6 Activating a page

A project page can be activated via the "PageOn" function.

Declaration:	void PageOn (int __PageNo)
Parameter:	int __PageNo number of page to be activated
Return value	none

Example:

```
PageOn(4); /* activates page 4 */
```

3.2.2.7 Deactivating a page

A project page can be deactivated via the "PageOff" function.

Declaration:	void PageOff (int __PageNo)
Parameter:	int __PageNo number of page to be deactivated
Return value	none

Example:

```
PageOff(4); /* deactivates page 4 */
```

Operating panel manual

3.2.2.8 Activating a priority page

A project page can be activated as priority page via the "PrioPageOn" function. As opposed to a "normal" page, the priority page is stored on top of the page stack and is therefore displayed immediately.

Declaration:	void PrioPageOn (int __PageNo)
Parameter:	int __PageNo number of page to be activated
Return value	none

Example:

```
PrioPageOn(5); /* activates page 5 and displays the page immediately */
```

3.2.2.9 Deactivating a priority page

This function deactivates a priority page.

Declaration:	void PrioPageOff (int __PageNo)
Parameter:	int __PageNo number of page to be deactivated
Return value	none

Example:

```
PrioPageOff(5); /* deactivates PrioPage 5 */
```

3.2.2.10 Activating a menu page

If a menu page is activated, a page is called up and the device immediately switches to the entry/menu structure, provided a menu item and/or nominal value with activation option is available on the page.

Declaration:	void MenuPageOn (int __PageNo)
Parameter:	int __PageNo number of page to be activated
Return value	none

Example:

```
MenuPageOn(17); /* activates page 17 as menu page */
```

3.2.2.11 Deactivating the last menu pages/structure

This function deactivates the last steps of the menu entry. Within this context, the program does not refer to the page numbers, but deactivates the last (count) menu pages.

Declaration:	void MenuPagesOff (int __Count)
Parameter:	int __Count number of pages to be deactivated
Return value	none

Example:

```
MenuPageOn(18);  
MenuPageOn(17);  
MenuPageOn(16);  
MenuPagesOff(2); /* deactivates 16, 17 */
```

Operating panel manual

3.2.3 Functions for working with messages

3.2.3.1 Getting message scrolling time

With this function, the message scrolling time can be retrieved in seconds.

Declaration:	int GetMessageAutoScrollTime (void)
Parameter:	none
Return value	int time in seconds

Example:

```
iScrollTime = GetMessageAutoScrollTime();
```

3.2.3.2 Setting message scrolling time

With this function, the message scrolling time can be written in seconds.

Declaration:	void SetMessageAutoScrollTime (int __Secs)
Parameter:	int __Secs time in seconds (0=scrolling off)
Return value	int time in seconds

Example:

```
SetMessageAutoScrollTime(5); /* 5 seconds */
```

Accept:

Immediately. After a re-start only if SaveToFlash has been executed.

3.2.3.3 Getting number of the currently displayed message

The `GetCurrentMessageShown()` function indicates the number of the currently displayed message as integer. If no message is displayed, 0 is indicated.

Declaration:	int GetCurrentMessageShown (void)
Parameter:	none
Return value	int number of the currently displayed message

Example:

```
iMessageNumber = GetCurrentMessageShown();
```

3.2.3.4 Determining whether a message is active

A message is active if it has been activated via the CAN or the program. This does not necessarily imply that the message is visible on the display.

Declaration:	int IsMessageOn (int __MsgNo)
Parameter:	int __MsgNo number of the message
Return value	int == 0: page is not active <> 0: page is active

Example:

```
if (IsMessageOn(10))  
{  
  ... /* user program if the message is active*/  
}
```

Operating panel manual

3.2.3.5 Activating a message

A project message can be activated via the "MessageOn" function.

Declaration:	void MessageOn (int __MsgNo)
Parameter:	int __MsgNo number of message to be activated
Return value	none

Example:

```
MessageOn(10);
```

3.2.3.6 Deactivating a message

A project message can be deactivated via the "MessageOff" function.

Declaration:	void MessageOff (int __MsgNo)
Parameter:	int __MsgNo number of message to be deactivated
Return value	none

Example:

```
MessageOff(10);
```

3.2.4 Functions for working with keys

3.2.4.1 Checking whether any key is pressed

With this function, you can check whether any or no key (foil key or incremental poti) is pressed.

Declaration:	int IsAnyKeyDown (void)
Parameter:	none
Return value	int == 0: no key is pressed <> 0: key number of the pressed key

Example:

```
switch(IsAnyKeyDown())
{
  case 0x00: /* no key is pressed*/
    break;
  case 0x01: /* key 1 is pressed */
    ... /* user program if key 1 is pressed */
    break;
  case 0x02: /* key 2 is pressed */
    ... /* user program if key 1 is pressed */
    break;
  default:
    ... /* user program if key 1 is pressed */
    break;
}
```


Operating panel manual

3.2.4.5 Simulating an ASCII input

Activating this function, an alphanumeric key can be simulated in a way as if it had been really pressed at an externally connected PS/2 keyboard. The device automatically generates "Key pressed" and subsequently "Key released".

Declaration:	<code>void SimulateASCIIKey(int __KeyCode, int ExtCode)</code>
Parameter:	<code>int __KeyCode</code> ASCII code of the key <code>int __ExtCode</code> currently not used, please indicate 0
Return value	void

Example:

```
SimulateASCIIKey(65.0); /* Simulates A */
```

3.2.4.6 Positioning cursor on the entry field/menu item

If a menu page is active, this function allows the cursor to be positioned on a particular field. The field can be selected via the X/Y position or function.

The cursor is positioned if the device is in the idle state (0), menu selection (4) or nominal value selection (5) and a nominal value and/or menu item is available on the page.

Declaration:	<code>void SetCursorPos(int __Mode, int __X, int __Y, int MPFunc)</code>
Parameter:	<code>int __Mode</code> 0,1try to position cursor on line Y, column X 2 set to nominal value with variable handle X 3 set to menu item with function MPFunc and parameter Y <code>int __X</code> column or variable handle <code>int __Y</code> line or parameter <code>int __MPFunc</code> 0 call up page (X=page number) 1 go back menu (X insignificant) 2 exit nominal value input with saving 3 exit nominal value input without saving 4 global abortion 5 menu index output (X = menu index) 6 index output with global abortion (X = menu index) 7 transfer menu index to control program (X = menu index)
Return value	void

Example:

```
SetCursorPos(3,4,0,0); /* Position cursor on the menu item which contains page call-up 4 */
```

Operating panel manual

3.2.4.7 Requesting cursor position

If a menu page is active, the cursor position can be requested via this function.

Declaration:	void GetCursorPos (int * __Mode , int * __X , int * __Y , int * MPFunc)
Parameter:	<p>int * __Mode 0: cursor is invisible (no menu is active) 1: position in line Y, column X 2: set to nominal value with variable handle X 3: set to menu item with function MPFunc and parameter X</p> <p>int * __X column or variable handle</p> <p>int * __Y line or parameter</p> <p>int MPFunc 0 call up page (X=page number) go back1 menu (X insignificant) 2 exit nominal value input with saving 3 exit nominal value input without saving 4 global abortion 5 menu index output (X = menu index) 6 index output with global abortion (X = menu index) 7 transfer menu index to control program (X = menu index)</p>
Return value	void

Example:

```
int Mode, X, Y, MPFunc;

GetCursorPos(&Mode,&X,&Y,&MPFunc); /* get values */

if (mode == 0)
{
... /* user program if cursor is invisible */
}
```

3.2.4.8 Getting menu item index

A menu item can be programmed via the "transfer index to LD" function. The GetCurrentMenuIndex() function provides this value if the respective menu item is selected by pressing "Enter".

Note:

The value is available exactly once in the "LD_Cycle" sub-program. The access via the time-controlled routine is not reliable!

Declaration:	uint16 GetCurrentMenuIndex (void)
Parameter:	none
Return value	int == 0: no key is pressed <> 0: key number of the pressed key

Example:

```
iMenuIndex = GetCurrentMenuIndex();
```

Operating panel manual

3.2.4.9 Getting "soft key" mask

Soft keys are keys which actually have a function for the operating panel, but the user only wants to use them for the PLC or the LD. For this purpose, the keys are coded in terms of bits. For more information, please also refer to the description of the SET PARAM CAN-telegram.

Declaration:	int GetSoftkeyMask (void)
Parameter:	none
Return value	int soft key mask (coded in terms of bits)

Example:

```
iSoftmask = GetSoftkeyMask();
```

3.2.4.10 Setting "soft key" mask

Soft keys are keys which actually have a function for the operating panel, but the user only wants to use them for the PLC or the LD. For this purpose, the keys are coded in terms of bits. For more information, please refer to the description of the SET PARAM CAN telegram.

Declaration:	void SetSoftkeyMask (int __State)
Parameter:	int soft key mask (coded in terms of bits)
Return value	none

Example:

```
iSoftmask = GetSoftkeyMask();  
iSoftmask = iSoftmask | 0x02;  
SetSoftkeyMask(iSoftmask);
```

Accept:

Immediately. After a re-start only if SaveToFlash has been executed.

3.2.4.11 Getting number of the key extension blocks

Provides the number of the project-related key extensions.

Declaration:	int GetKeyBlockCount (void)
Parameter:	none
Return value	int number of key extensions

Example:

```
number_F_keys = 8 * GetKeyBlockCount();
```

Operating panel manual

3.2.5 Functions for working with LEDs

3.2.5.1 Switching on LEDs

Via this function, a LED can be switched on.

Declaration:	void LedOn (int __LedNo)
Parameter:	int __LedNo number of the LED
Return value	none

Example:

```
LedOn(8);
```

3.2.5.2 Switching off LEDs

Via this function, a LED can be switched off.

Declaration:	void LedOff (int __LedNo)
Parameter:	int __LedNo number of the LED
Return value	none

Example:

```
for (i=1; i<9; i++)  
  LedOff(i);
```

3.2.5.3 Checking whether a LED is switched on

Via this function, you can check whether a front LED is switched on.
The LED numbers are device-dependent.

Declaration:	int IsLedOn (int __LedNo)
Parameter:	int __KeyNo number of the LED to be requested
Return value	int == 0: LED is off <> 0: LED is on

Example:

```
if (IsLedOn(2))  
{  
  ... /* user program if LED is on */  
}
```

Operating panel manual

3.2.6 Functions for working with timers

3.2.6.1 Starting/setting a timer

Start one of the timers. 16 timers, 0-15, are available. All these timers can be set to a particular value which is automatically counted down to 0 by the operating system.

A timer can be started via the "SetTimer" function. Several timers can be operated simultaneously (even all of them). The timer value is indicated in steps of 10 msec.

Declaration:	void SetTimer (int __TimerNo, long __value)
Parameter:	int __TimerNo number of the timer int __value timer value
Return value	none

Example:

```
SetTimer(3,100); /* 1 sec. */
```

3.2.6.2 Requesting whether a timer is operating

With this function, you can check whether the timer is still operating.

Declaration:	int IsTimerOn (int __TimerNo)
Parameter:	int __TimerNo number of timer to be requested
Return value	int == 0: timer is out of operation (at 0) <> 0: timer is still in operation

Example:

```
if (IsTimerOn(1))  
{  
... /* user program as long as the timer is in operation*/  
}
```

3.2.6.3 Requesting whether a timer has expired

With this function, you can check whether the timer has expired. This request provides the reverse result of the "IsTimerOn" function.

Declaration:	int IsTimerOff (int __TimerNo)
Parameter:	int __TimerNo number of timer to be requested
Return value	int == 0: timer is still in operation <> 0: timer is out of operation (at 0)

Example:

```
if (IsTimerOff(12))  
{  
... /* user program as long as the timer is out of operation*/  
}
```

Operating panel manual

3.2.7 Functions for working with inputs/outputs

Most devices either have an output or an input or both. Depending on the device type, additional inputs/outputs have already been integrated for control purposes.

The following functions allow for a direct access to the internal inputs/outputs.

Apart from the already mentioned functions, you also have access via the GCM request commands when using node number 0.

Further information on the handling of inputs/outputs with CAN modules are available under 3.2.10 Functions for actuating GCM-Can modules.

3.2.7.1 Switching internal message output on

With this function, the message output is switched on. In the future, more outputs will be switchable.

Declaration:	void OutputOn (int __OutNo)
Parameter:	int __OutNo number of the output 1=standard message output
Return value	none

Example:

```
OutputOn(1);
```

3.2.7.2 Switching internal message output off

With this function, the message output is switched off. In the future, more outputs will be switchable.

Declaration:	void OutputOff (int __OutNo)
Parameter:	int __OutNo number of the output 1=standard message output
Return value	none

Example:

```
OutputOff(1);
```

3.2.7.3 Requesting message output

With this function, the device internal message output can be requested. In the future, further outputs can be provided via the output number.

Declaration:	int IsOutputOn (int __OutNo)
Parameter:	int __OutNo number of output to be called up 1=standard message output
Return value	int == 0: output is low <> 0: output is high

Example:

```
if (IsOutputOn(1))
```

Operating panel manual

3.2.7.4 Requesting LIGHT-IN input

With this function, the device internal LIGHT-IN input can be requested. In the future, further inputs can be provided via the input number.

Declaration:	int IsInputOn (int __InNo)
Parameter:	int __InNo number of input to be requested 1=standard LIGHT-IN input
Return value	int == 0: input is low <> 0: input is high

Example:

```
if (IsInputOn(1))
```

3.2.7.5 Requesting internal digital input

With this function, a digital input of a device with additional internal I/Os can be requested. The number of the inputs depends on the integrated I/O structure. The input numbers always start at 0.

Declaration:	int GetInternalDI (int __InNo)
Parameter:	int __InNo number of input to be requested
Return value	int == 0: input is low <> 0: input is high

Example:

```
if (GetInternalDI(1))
```

3.2.7.6 Requesting internal digital output

With this function, a digital output of a device with additional internal I/Os can be requested. The number of outputs depends on the integrated I/O structure. The output numbers always start at 0.

Declaration:	int GetInternalDO (int __OutNo)
Parameter:	int __OutNo number of output to be requested
Return value	int == 0: input is low <> 0: input is high

Example:

```
if (GetInternalDO(12))  
{  
.../* user program */  
}
```

Operating panel manual

3.2.7.7 Switching internal digital output

By means of this procedure, you can switch a digital output of a device with additional internal I/Os on or off. The number of outputs depends on the integrated I/O structure. The output numbers always start at 0.

Declaration:	void SetInternalDO (int __OutNo, int __Value)
Parameter:	int __OutNo number of output to be switched int __Value value
Return value	none

Example:

```
SetInternalDO(12,1);
```

3.2.7.8 Getting internal analogue input

With this function, you can request an analogue input of a device with additional internal I/Os. The number of inputs depends on the integrated I/O structure. The input numbers always start at 0.

Declaration:	int GetInternalAI (int __InNo)
Parameter:	int __InNo number of input to be requested
Return value	int value of the digitally converted analogue input signal (unscaled)

Example:

```
VoltageDigital = GetInternalAI(1);
```

3.2.7.9 Getting input byte 0

Declaration:	int GetDIB0 (void)
Parameter:	none
Return value	int status of input byte 0 (inputs 0..7)

3.2.7.10 Getting input byte 1

Declaration:	int GetDIB1 (void)
Parameter:	none
Return value	int status of input byte 1 (inputs 8..15)

3.2.7.11 Getting input word 0

Declaration:	uint16 GetDIW0 (void)
Parameter:	none
Return value	uint16 status of input word 0 (inputs 0..15)

Operating panel manual

3.2.7.12 Getting output byte 0

Declaration:	int GetDOB0 (void)
Parameter:	none
Return value	int status of output byte 0 (outputs 0..7)

3.2.7.13 Writing output byte 0

Declaration:	void SetDOB0 (int __Value)
Parameter:	int __Value status of output byte 0 (outputs 0..7)
Return value	none

3.2.7.14 Getting output byte 1

Declaration:	int GetDOB1 (void)
Parameter:	none
Return value	int status of output byte 1 (outputs 8..15)

3.2.7.15 Writing output byte 1

Declaration:	void SetDOB1 (int __Value)
Parameter:	int __Value status of output byte 1 (outputs 8..15)
Return value	none

3.2.7.16 Getting output word 0

Declaration:	uint16 GetDOW0 (void)
Parameter:	none
Return value	uint16 status of output word 0 (outputs 0..15)

3.2.7.17 Writing output word 0

Declaration:	void SetDOW0 (uint16 __Value)
Parameter:	int __Value status of output byte 0 (outputs 0..15)
Return value	none

3.2.7.18 Getting short-circuit status

Declaration:	uint16 GetShortcutState (void)
Parameter:	none
Return value	uint16 in terms of bits for outputs 0..15

Operating panel manual

3.2.8 Fast 2-channel counters

The following functions serve for requesting and actuating the device-internal fast counter hardware, if available.

3.2.8.1 Getting counter value 1

Declaration:	uint32 GetCount1 (void)
Parameter:	none
Return value	uint32 counter value

3.2.8.2 Writing counter value 1

Declaration:	void SetCount1 (uint32 __Value)
Parameter:	uint32 __Value counter value to which the counter is to be set
Return value	none

3.2.8.3 Getting counter value 2

Declaration:	uint32 GetCount2 (void)
Parameter:	none
Return value	uint32 counter value

3.2.8.4 Writing counter value 2

Declaration:	void SetCount2 (uint32 __Value)
Parameter:	uint32 __Value counter value to which the counter is to be set
Return value	none

3.2.8.5 Getting pre-selection 1 for counter 1

Declaration:	uint32 Get_C1_Preset1 (void)
Parameter:	none
Return value	uint32 pre-selection value

3.2.8.6 Writing pre-selection 1 for counter 1

Declaration:	void Set_C1_Preset1 (uint32 __Value)
Parameter:	uint32 __Value value to which the pre-selection value is to be set
Return value	none

Operating panel manual

3.2.8.7 Getting pre-selection 2 for counter 1

Declaration:	uint32 Get_C1_Preset2 (void)
Parameter:	none
Return value	uint32 pre-selection value

3.2.8.8 Writing pre-selection 2 for counter 1

Declaration:	void Set_C1_Preset2 (uint32 __Value)
Parameter:	uint32 __Value value to which the pre-selection value is to be set
Return value	none

3.2.8.9 Getting pre-selection 1 for counter 2

Declaration:	uint32 Get_C2_Preset1 (void)
Parameter:	none
Return value	uint32 pre-selection value

3.2.8.10 Writing pre-selection 1 for counter 2

Declaration:	void Set_C2_Preset1 (uint32 __Value)
Parameter:	uint32 __Value value to which the pre-selection value is to be set
Return value	none

3.2.8.11 Getting pre-selection 2 for counter 2

Declaration:	uint32 Get_C2_Preset2 (void)
Parameter:	none
Return value	uint32 pre-selection value

3.2.8.12 Writing pre-selection 2 for counter 2

Declaration:	void Set_C2_Preset2 (uint32 __Value)
Parameter:	uint32 __Value value to which the pre-selection value is to be set
Return value	none

Operating panel manual

3.2.9 Access to the CAN interfaces

3.2.9.1 Data format for standard identifiers

In the CAN bus, an 11-bit address (identifier) is used for the stations.

These 11 bits are supplemented by:

1 bit RTR (remote request)

4 bit data length (DLC) valid values 0-8)

The total bit number resulting thereof is 16. The bits in this 16-bit word are allocated as follows:

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

In order to receive the actual identifier, the retrieved value must be moved 5 bits to the right. When writing, the value must logically be moved 5 bits to the left. This applies to all identifier-related functions described in the following section.

If 29-bit identifiers are to be used, the CAN parameterisation functions must be applied. Currently, no 29-bit identifier can be installed in the editor. To allow the functions to work with 29-bit identifiers, the Id must always be linked with the USE_29BIT_ID OR- macro.

If certain functions are to access the second CAN interface, the identifier is linked with the USE_CAN_1 OR-macro.

3.2.9.2 Setting CAN-bus rate

The bus rate is not indicated as an absolute but as an index value from a table:

Declaration:	void SetCAN0BusrateIndex (int Index) void SetCAN1BusrateIndex (int Index);
Parameter:	Index: see table
Return value	none

INDEX	Bus rate
0	10 kBit/sec
1	20 kBit/sec
2	50 kBit/sec
3	100 kBit/sec
4	125 kBit/sec
5	250 kBit/sec
6	500 kBit/sec
7	1 MBit/sec

As the devices of the AT series have been designed as ITS compatible devices, the project offers memory capacities for the parameterisation of one CAN interface. The operating system initialises both CAN interfaces with the same baud rate. If you require different baud rates, you are recommended to proceed as follows:

- Set the baud rate for the first CAN interface in the project.
- Call up the *SetCAN1BusRateIndex(...)* function in the LD init.
- Call up the *RestartCanSystem()* function in the LD init.

This way, a different baud rate can be used for the second CAN interface.

Accept:

Temporarily after *RestartCanSystem()*

Permanently after *SaveToFlash()* and *RestartCanSystem()* (only CAN0 parameters)

Operating panel manual

3.2.9.3 Getting CAN-bus rate

This function provides the table index of the bus rate currently set for the respective CAN interface. The function **cannot** be used to determine the currently set bus rate. This bus rate must be available!

Declaration:	int GetCAN0BusrateIndex (void) int GetCAN1BusrateIndex (void);
Parameter:	none
Return value	int table index. See "Setting CAN-bus rate" on page43.

3.2.9.4 Getting standard transmission identifier

Provides the set transmission identifier incl. RTR and DLC.

Declaration:	uint32 GetCanTxId (void)
Parameter:	none
Return value	uint32 identifier <i>incl.</i> RTR and DLC

Example:

```
iCanTxId = GetCanTxId() >> 5;
```

3.2.9.5 Writing standard transmission identifier

Sets the transmission identifier to the desired value.

Declaration:	void SetCanTxId (uint32 Id)
Parameter:	Id identifier <i>incl.</i> RTR and DLC
Return value	none

Example:

```
SetCanTxId(1000 << 5);
```

Accept:

Temporarily after RestartCanSystem()

Permanently after SaveToFlash() and RestartCanSystem()

3.2.9.6 Getting standard receive identifier

Provides the set receive identifier incl. RTR and DLC.

Declaration:	uint32 GetCanRxId0 (void)
Parameter:	none
Return value	uint32 identifier <i>incl.</i> RTR and DLC

Example:

```
iCanTxId = GetCanRxId0() >> 5;
```

Operating panel manual

3.2.9.7 Writing standard receive identifier

Sets the receive identifier to the desired value.

Declaration:	void SetCanRxId0 (uint32 Id)
Parameter:	Id identifier <i>incl.</i> RTR and DLC
Return value	none

Example:

```
SetCanRxId0(1001 << 5);
```

Accept:

Temporarily after RestartCanSystem()

Permanently after SaveToFlash() and RestartCanSystem()

3.2.9.8 Getting multimaster receive channels

These functions provide the receive identifier of the multimaster channels.

Declarations:	uint32 GetCanRxId1 (void); uint32 GetCanRxId2 (void); uint32 GetCanRxId3 (void); uint32 GetCanRxId4 (void); uint32 GetCanRxId5 (void); uint32 GetCanRxId6 (void); uint32 GetCanRxId7 (void);
Parameter:	none
Return value	uint32 identifier <i>incl.</i> RTR and DLC

Example:

```
iCanRxMasterId1 = GetCanRxId1() >> 5;
```

3.2.9.9 Writing multimaster receive channels

These functions provide the receive identifier of the multimaster channels.

Declarations:	void SetCanRxId1 (uint32 Id) void SetCanRxId2 (uint32 Id) void SetCanRxId3 (uint32 Id) void SetCanRxId4 (uint32 Id) void SetCanRxId5 (uint32 Id) void SetCanRxId6 (uint32 Id) void SetCanRxId7 (uint32 Id)
Parameter:	Id identifier <i>incl.</i> RTR and DLC
Return value	none

Example:

```
SetCanRxId4(13 << 5);
```

Accept:

Temporarily after RestartCanSystem()

Permanently after SaveToFlash() and RestartCanSystem()

Operating panel manual

3.2.9.10 New initialisation of the CAN interfaces

This function is used to re-start the communication on the CAN bus after an error has led to the BUSOFF state or after the CAN-interface(s) parameters have been changed.

Declarations:	void RestartCanSystem (void);
Parameter:	none
Return value	none

3.2.9.11 Checking CAN-bus status

Declarations:	int Get_CAN_Status (void);
Parameter:	none
Return value	int status as 16-bit value

3.2.9.12 Transmitting a CAN telegram

Via this function, a CAN telegram with arbitrary data and an arbitrary identifier can be transmitted to a CAN bus.

Declaration:	void SendCANTelegram (int ID , int Len , char D0 , char D1 , char D2 , char D3 , char D4 , char D5 , char D6 , char D7);
Parameter:	int Id : identifier (excl. RTR and DLC) 11-bit ID=0 ... 2047 29-bit ID=0 ... 536870911 int Len : number of data bytes to be transmitted char D0 : data byte D0 char D1 : data byte D1 char D2 : data byte D2 char D3 : data byte D3 char D4 : data byte D4 char D5 : data byte D5 char D6 : data byte D6 char D7 : data byte D7
Return value	none

Examples:

```
/* transmits with 11-bit identifier 144, 3 bytes, to CAN interface 0 */  
SendCANTelegram(144,3,0x12,0x06,0xFE,0,0,0,0,0);
```

```
/* transmits with 29-bit identifier 144, 3 bytes, to CAN interface 0 */  
SendCANTelegram(144 | USE_29BIT_ID,3,0x12,0x06,0xFE,0,0,0,0,0);
```

```
/* transmits with 29-bit identifier 6122, 2 bytes, to CAN interface 1 */  
SendCANTelegram(6122 | USE_29BIT_ID | USE_CAN_1,2,0x10,0x01,0,0,0,0,0,0);
```

Note:

In order to be able to transmit with 11 bits to identifier 0, you have to use ID 0x800. With the 11-bit setting, ID=0 transmits on the transmit identifier (TxID) from the project setting.

Operating panel manual

3.2.9.13 Dynamically enabling a receive identifier

Via this function, a receive identifier can be enabled for a CAN interface.

Declaration:	void CANEnableRxId (int Channel , uint32 Id)
Parameter:	int Channel channel 0 = standard CAN interface 1 = second CAN interface uint32 Id identifier 11-bit ID=0 ... 2047 29.Bit ID=0 ... 536870911
Return value	none

Examples:

```
/* enable 11-bit-identifier 1601 on channel 0 */  
CANEnableRxId(0,1601);
```

```
/* enable 29-bit-identifier 1601 on channel 0 */  
CANEnableRxId(0,1601 | USE_29BIT_ID);
```

```
/* enable 29-bit-identifier 1601 on channel 1 */  
CANEnableRxId(1,1601 | USE_29BIT_ID);
```

3.2.9.14 Disabling dynamically-assigned receive identifiers

Via this function, the enabled receive identifier of a CAN interface can be disabled.

Declaration:	void CANDisableRxId (int Channel , uint32 Id)
Parameter:	int Channel channel 0 = standard CAN interface 1 = second. CAN interface uint32 Id identifier 11-bit ID=0 ... 2047 29-bit ID=0 ... 536870911
Return value	none

Examples:

```
/* disable 11-bit-identifier 1601 on channel 0 */  
CANDisableRxId(0,1601);
```

```
/* disable 29-bit-identifier 1601 on channel 0 */  
CANDisableRxId(0,1601 | USE_29BIT_ID);
```

```
/* disable 29-bit-identifier 1601 on channel 1 */  
CANDisableRxId(1,1601 | USE_29BIT_ID);
```

Operating panel manual

3.2.9.15 Setting enabling mask for receive identifier

With this function, an identifier area can be enabled. The parameter `__mask` contains the bits to be relevant for reception. The bit value is to be indicated in the parameter `__Id`. Upon reception, a $(MsgId \& Mask) == (__Id \& Mask)$ check is carried out. If the check is completed as True, the CAN event is released.

Declaration:	void CANEnableRxMask (uint32 __Channel , uint32 __Id , uint32 __Mask)
Parameter:	__Channel __Id Identifier __Mask
Return value	none

3.2.9.16 Removing enabling mask for receive identifier

Declaration:	void CANDisableRxMask (uint32 __Channel , uint32 __Id , uint32 __Mask)
Parameter:	__Channel __Id identifier __Mask
Return value	none

3.2.9.17 Checking whether the device runs as CAN master

You can check whether the device is configured as master (SELECAN or CANopen), but you cannot modify this setting...

Declaration:	void GetMASTERFlag (void)
Parameter:	none
Return value	none

Example:

```
if (GetMASTERFlag())
{
    perform master functions ...
}
```

3.2.9.18 Monitoring the CANopen transmission

With this function, the CANopen-SDO transmission can be monitored.

CAUTION! This function will presumably be abandoned in favour of a complete CANopen functionality. We are planning to implement a complete CANopen stack into the device.

Declaration:	int SDOStatus (int SDOno)
Parameter:	int SDOno of SDO to be monitored
Return value	int 0 if SDO has not been transmitted <>0 if SDO has been transmitted

Operating panel manual

3.2.9.19 Getting SELECAN device number

If the device is operated in the SELECAN mode, this function enables you to retrieve, determine and modify the SELECAN device number. These actions should be carried out by experienced users only.

Declaration:	int GetSELECANNodeNo (void)
Parameter:	none
Return value	int device number

3.2.9.20 Writing SELECAN device number

If the device is operated in the SELECAN mode, this function enables you to retrieve, determine and modify the SELECAN device number. These actions should be carried out by experienced users only.

Declaration:	void SetSELECANNodeNo (int NodeId)
Parameter:	int NodeId device address
Return value	none

3.2.10 Functions for actuating GCM-Can modules

The CAN modules of the GRAF-SYTECO GCM series can be actuated relatively easily. Simply create these modules in the CAN configuration tool without configuring any input/output functions.

By means of the C-code, you have the possibility to retrieve inputs or set outputs both digitally and analogously.

The inputs are permanently updated whereas the outputs are not output to the modules until the cyclic program part has been completed.

If you use module number 0, you have access to the device-internal I/Os.

3.2.10.1 Requesting external digital input

This function enables you to request a digital input.

Declaration:	void GetGCM_DI (int NodeId , int InNo)
Parameter:	int NodeId device address of the GCM module int InNo number of the digital input
Return value	int 0:input is off <>0: input is on

Example:

```
/* request module input 9 with node number 2 */  
if (GetGCM_DI(2,9))
```

Operating panel manual

3.2.10.2 Requesting external analogue input

This function enables you to request an analogue input. The value is not scaled and provides the exact binary value of the A/D converter of the module.

Declaration:	void GetGCM_AI (int NodeId , int InNo)
Parameter:	int NodeId device address of the GCM module int InNo number of the analogue input
Return value	int for the value, refer to the module manual

Example:

```
/* request module input 2 with node number 1 */  
Value = GetGCM_AI(1,2);
```

3.2.10.3 Setting external digital output

This function enables you to set a digital output.

Declaration:	void SetGCM_DO (int NodeId , int OutNo , int Value)
Parameter:	int NodeId device address of the GCM module int InNo number of the digital input int Value 0 or 1 (logical!)
Return value	none

Example:

```
/* set output 0 at the module with node number 5 */  
SetGCM_DO(5,0,1);
```

Note::

An output may also be read back via the "get digital output" function. Simply take the output number as input number.....

3.2.10.4 Setting external analogue output

This function enables you to set an analogue output. The value is not scaled and must correspond exactly to the binary value of the D/A converter of the module.

Declaration:	void SetGCM_AO (int NodeId , int OutNo , int Value)
Parameter:	int NodeId device address of the GCM module int InNo number of the digital input int Value 0 or 1 (logical!)
Return value	none

Example:

```
/* set output 1 at the module with node number 1 */  
SetGCM_AO(1,1,value);
```

Operating panel manual

3.2.10.5 Requesting module status

Via this function, the module status can be requested.

Declaration:	void GetGCM_Status (int NodeId)
Parameter:	int NodeId device address of the GCM module
Return value	not yet determined

Example:

```
/* request module status with node number 1 */
Status = GetGCM_Status(1);
```

3.2.11 Access to the serial interfaces

3.2.11.1 Configuring serial interfaces

Two parameters, the baud rate and the data format, are available for serial interfaces. The baud rate is a table which contains the following values.

As with CAN interfaces and due to compatibility reasons, parameters are only provided for the first serial interface (Serial0). If you would like to work with different baud rates you must therefore proceed as with CAN interfaces:

- *Set the baud rate and the data format for the first serial interface in the project.*
- *Call up the `SetSerial1BaudrateIndex(...)` function in the LD init.*
- *Call up the `SetSerial1FrameSetting(...)` function in the LD init.*
- *Call up the `RestartCanSystem()` LD init.*

3.2.11.2 Getting baud rate

The set baud rate of the serial interface is not returned directly but as index of the following table. This function **cannot** be used to determine the baud rate of the serial communication. This rate must be available before! It only serves to retrieve the interface setting.

INDEX	Baud rate
0	9600 bauds
1	4800 bauds
2	2400 bauds
3	1200 bauds
4	600 bauds
5	300 bauds
6	150 bauds
7	110 bauds

Declaration:	int GetSerial0BaudIndex (void) int GetSerial1BaudIndex (void);
Parameter:	none
Return value	int for the index, refer to the table

3.2.11.3 Setting baud rate

This function sets the baud rate of the serial interface. Function parameter is the index of the table above.

Declaration:	void SetSerial0BaudIndex (int value) void SetSerial1BaudIndex (int value);
Parameter:	int for the index, refer to the table

Operating panel manual

Return value	none
--------------	------

3.2.11.4 Getting telegram setting

This function provides the setting of the telegram frame for the serial transmission. As with getting the baud rate, this function can only be used to determine the interface setting and not the telegram-frame setting for the communication with other devices. The setting must be available before. The data format of the telegram-frame setting (PARAM) is described in terms of bits:

Bit number	Meaning
0	meaningless
1	meaningless
2	number of stop bits 0=1, 1=2
3	number of data bits 0=7, 1=8
4	parity 0=odd, 1=even
5	parity 0=disabled 1=enabled
6	meaningless
7	meaningless

Declaration:	int GetSerial0FrameSetting (void) int GetSerial1FrameSetting (void);
Parameter:	none
Return value	int for the value, refer to the table

3.2.11.5 Making telegram setting

Sets the telegram frame of the serial communication.

Declaration:	void SetSerial0FrameSetting (int value) void SetSerial1FrameSetting (int value);
Parameter:	int Value for the value, refer to the table
Return value	none

Example for 3.2.11.3 to 3.2.11.5:

```
FrameSetting = GetSerial0FrameSetting();
```

Accept:

Temporarily after RestartComSystem()

Permanently after SaveToFlash and RestartComSystem() (only serial 0)

Operating panel manual

3.2.11.6 New initialisation of the serial interfaces

After the parameters of the serial interface(s) have been modified, you are required to initialise them again to accept the modified parameters. This can be realised by means of the `RestartComSystem()` function.

Declaration:	void RestartComSystem (void)
Parameter:	none
Return value	none

3.2.11.7 Serial reception

The serial interfaces are administered in an interrupt-controlled way by the TOS. The TOS writes characters which are received by a serial interface into a readable receive buffer. Furthermore, a status and a retrieve function are available. The `GetStatSerialx()` status function provides a value unequal 0 if a character is pending in the receive buffer. After the character has been read out with `GetCharSerialx()`, the character must be acknowledged with `SetStatSerialx(0)`.

3.2.11.8 Status request

Declaration:	void GetStatSerial0 (void) void GetStatSerial1 (void)
Parameter:	none
Return value	int 0=no characters received <>0=characters present

3.2.11.9 Getting received characters

Declaration:	int GetCharSerial0 (void) int GetCharSerial1 (void)
Parameter:	none
Return value	int ASCII code of the received character

3.2.11.10 Acknowledging received character

Declaration:	void SetStatSerial0 (int __Stat) void SetStatSerial1 (int __Stat)
Parameter:	int 0 only for the acknowledgement of the received character
Return value	none

Operating panel manual

3.2.11.11 Overwriting received character

Declaration:	void SetCharSerial0 (int __Char) void SetCharSerial1 (int __Char)
Parameter:	int __Char overwrites a received character with __Char This function is useless for normal operation!
Return value	none

Example:

```
while (GetStatSerial0())  
{  
    Characters = GetCharSerial0();  
    .... processing  
    SetStatSerial0(0);  
}
```

Note:

All characters of the receive buffer are processed by means of this loop. The respective value of the GetCharSerial0() function is the ASCII code of the received character. Therefore, it can also be read into a char-variable.

If more characters are received than the receive buffer is able to take up, the last received characters get lost.

3.2.11.12 Serial transmission

This function allows for a buffer transmission via the serial interface.

Declaration:	void SendToSerial (int __Channel, char *__Buffer, int __Count, int __Timeout)
Parameter:	int __Channel 0 or 1 depending on the interface char *__Buffer pointer on first character in string int __Count number of characters int __Timeout max. waiting time in milliseconds
Return value	none

Example:

```
char SendBuffer[10];  
...  
SendToSerial(0, SendBuffer, 5, 1000);
```

3.2.12 Functions for display settings

3.2.12.1 Getting contrast

Via these functions, you can get the current contrast level.

Declaration:	void GetContrastIndex (void)
Parameter:	none
Return value	int 0 .. 23 depending on the set contrast

Example:

```
Contrast = GetContrastIndex();
```

Operating panel manual

3.2.12.2 Setting contrast

Via these functions, the current contrast level can be written.

Declaration:	void SetContrastIndex (int __Index)
Parameter:	int __Index 0 .. 23 depending on the desired contrast
Return value	none

Example:

```
SetContrastIndex(contrast+1);
```

Accept:

Immediately. The contrast level is permanently saved.

3.2.12.3 Getting brightness

Via these functions, the you can get the current brightness level.

Declaration:	int GetBrightnessIndex (void)
Parameter:	none
Return value	int 0 .. 7 depending on the set brightness

Example:

```
Brightness = GetBrightnessIndex();
```

3.2.12.4 Setting brightness

These functions enable you to set the current brightness level.

Declaration:	void SetBrightnessIndex (int __Index)
Parameter:	int __Index 0 .. 7 depending on the desired brightness
Return value	none

Example:

```
SetBrightnessIndex(brightness-1);
```

Accept:

Immediately. The brightness level is permanently saved.

3.2.12.5 Getting brightness value for key illumination

Via these functions, you can get the current EL foil setting for key illumination (device-dependent!).

Declaration:	int GetELFOILState (void)
Parameter:	none
Return value	int brightness value

Example:

```
FoilState = GetELFOILState();
```

Operating panel manual

3.2.12.6 Setting brightness value for key illumination

Via these functions, the current EL foil setting for key illumination can be specified (device-dependending!).

Declaration:	void SetELFOILState (int State)
Parameter:	int brightness value 0=off 1=on
Return value	none

Example:

```
SetELFOILState(1-FoilState);
```

Accept:

Immediately. The brightness level is permanently saved.

3.2.13 Graphical functions

With some tasks, a graphical realisation by means of the configuration tool is rather complicated due to the dynamic representation. Within this context, the representation of a calibration curve offers a good example.

Thanks to the comfortable C-programming functions, you have the possibility to draw directly from the C code level on the operating panel display.

There are four drawing layers. You may draw the background onto one layer and dynamic components onto another one. You therefore do not need to care about the parts which have to be built up again by the background.

For drawing tasks, so-called direct and indirect drawing operations are available. In order to minimise flickering during the page build-up, you may first draw onto a memory field which is then completely copied into the display. This procedure allows for a smooth page build-up (indirect drawing).

If you would like to represent movements, it is recommendable to initially build up the page background in the memory, to copy it into the page and to finally draw the dynamic page components directly onto the page. With the direct drawing method you are additionally required to indirectly draw onto the respective layer.

A total of four layers is available for your drawings.

These layers are specified in the following:

- *Layer for static, non-blinking elements*
- *Layer for static, blinking elements*
- *Layer for dynamic, non-blinking elements*
- *Layer for dynamic, blinking elements*

The layer to be addressed is specified in the attribute parameter. If, e.g., the blinking bit is set, one of the layers for blinking elements is selected for drawing and the operating system enables the blinking. You therefore do not need to permanently write into and delete from the screen. For the targeted addressing of individual layers, please refer to the description of the attribute parameter.

Operating panel manual

3.2.13.1 Structure of the attribute parameter

Elements visualised on the display are provided with an attribute parameter, which determines their appearance. This attribute parameter is structured in terms of bits. The meaning of the individual bits can be taken from the following table:

Bit number	Meaning
0	Displaying reverted element
1	Displaying blinking element (layer selection)
2	Underlined (only for character output)
3	Characters of alternative character set (only for character output)
4/5	Character size (only for character output)
6	0 = indirect drawing, 1 = direct drawing
7	0 = static layer, 1 = dynamic layer

3.2.13.2 Coordinate system

All drawing operations are based on pixel-oriented coordinates. The zero point is located in the top left corner, the horizontal direction is determined by the X coordinate and the vertical direction by the Y coordinate. For some functions, also the height and width or radius of an object are required. These dimensions are indicated in pixels as well.

The AT3 is a special case: The device can be installed with the keys on the left or at the bottom. Where are the zero point coordinates in this case?

Answer: always in the top left corner. After all, it depends on your configuration whether the keys are on the left or at the bottom. The graphic-sub-system of the TOS adopts this setting when representing objects. Consequently, you need not think about rotation or similar things.

3.2.13.3 Colours

On monochrome displays, only the colours black (colour = 0x000000) and white (colour = 0xFFFFFFFF) make sense. All other colours are reserved for colour displays.

3.2.13.4 Drawing an individual pixel

With the `DrwPixel()` function, an individual pixel can be set.

Declaration:	<code>void DrwPixel(int __X, int __Y, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __X</code> X coordinate of the pixel <code>int __Y</code> Y coordinate of the pixel <code>int __attributes</code> for attributes, refer to the table <code>uint32 __Color</code> colour
Return value	none

Examples:

```
/* draws an individual pixel, static layer, direct output */  
DrwPixel(10,10,0x40,0);
```

```
/* draws an individual pixel, dynamic layer, indirect output */  
DrwPixel(10,10,0x80,0);
```

Operating panel manual

3.2.13.5 Drawing a rectangle without filling

With the `DrwRect()` function, you can draw a rectangle without filling (a frame). The frame thickness is one pixel. The rectangle is located parallelly to the display edges.

Declaration:	void DrwRect (int __XLeft , int __YTop , int __XRight , int __YBottom , int Attribute , uint32 __Color)
Parameter:	int __XLeft X coordinate top-left corner int __YTop Y coordinate top-left corner int __XRight X coordinate bottom-right corner int __YBottom Y coordinate bottom-right corner int __attributes for the attribute, refer to the table uint32 __Color colour
Return value	none

Example:

```
/* draws a blinking frame, static layer, direct output */  
DrwRect(20,20,80,30,0x42,0);
```

3.2.13.6 Drawing a rectangle with filling

With the `DrwRectFilled()` function, you can draw a rectangle with filling. The filling appears in the frame colour. The rectangle is located parallelly to the display edges.

Declaration:	void DrwRectFilled (int __XLeft , int __YTop , int __XRight , int __YBottom , int attributes , uint32 __Color)
Parameter:	int __XLeft X coordinate top-left corner int __YTop Y coordinate top-left corner int __XRight X coordinate bottom-right corner int __YBottom Y coordinate bottom-right corner int __attributes for the attribute, refer to the table uint32 __Color colour
Return value	none

Example:

```
/* draws a filled rectangle, dynamic layer, direct output */  
DrwRectFilled(20,35,80,40,0xB0,0);
```

3.2.13.7 Drawing a line

With the `DrwLine()` function, you can draw a line. The line thickness is one pixel.

Declaration:	void DrwLine (int __XLeft , int __YTop , int __XRight , int __YBottom , int attributes , uint32 __Color)
Parameter:	int __XLeft X coordinate start point int __YTop Y coordinate start point int __XRight X coordinate end point int __YBottom Y coordinate end point int __attributes for the attribute, refer to the table uint32 __Color colour
Return value	none

Example:

```
/* draws a line, static layer, indirect output */  
DrwLine(20,20,80,30,0x00,0);
```

Operating panel manual

3.2.13.8 Drawing a circle

With the `DrwCircle()` function, you can draw a circle around a centre point with radius. The line thickness is one pixel, the circle is not filled.

Declaration:	<code>void DrwCircle(int __XMid, int __YMid, int __Radius, int attributes, uint32 __Color)</code>
Parameter:	<code>int __XMid</code> X coordinate centre point <code>int __YMid</code> Y coordinate centre point <code>int __Radius</code> radius in pixels <code>int __attributes</code> for the attribute, refer to the table <code>uint32 __Color</code> colour
Return value	none

Example:

```
/* draws a circle, static layer, indirect output */  
DrwCircle(120,32,10,0x00,0);
```

3.2.13.9 Drawing a filled circle

With the `DrwCircle()` function, you can draw a circle around a centre point with radius. The circle is filled with the indicated colour.

Declaration:	<code>void DrwCircleFilled(int __XMid, int __YMid, int __Radius, int attributes, uint32 __Color)</code>
Parameter:	<code>int __XMid</code> X coordinate centre point <code>int __YMid</code> Y coordinate centre point <code>int __Radius</code> radius in pixels <code>int __attributes</code> for the attribute, refer to the table <code>uint32 __Color</code> colour
Return value	none

Example:

```
/* draws a filled circle, static layer, indirect output, colour white */  
DrwCircleFilled(120,32,5,0x00,0xFFFFFFFF);
```

3.2.13.10 Drawing a filled ellipse

With the `DrwCircle()` function, you can draw an ellipse around a centre point in a specified radius. The ellipse is filled with the indicated colour.

Declaration:	<code>void DrwEllipseFilled(int __XMid, int __YMid, int __XRadius, int __YRadius, int attributes, uint32 __Color)</code>
Parameter:	<code>int __XMid</code> X coordinate centre point <code>int __YMid</code> Y coordinate centre point <code>int __XRadius</code> X radius in pixels <code>int __YRadius</code> Y radius in pixels <code>int __attributes</code> for attributes, refer to the table <code>uint32 __Color</code> colour
Return value	none

Example:

```
/* draws an ellipse, dynamic layer, indirect output */  
DrwEllipseFilled(120,32,20,10,0x00,0);
```

Operating panel manual

3.2.13.11 Output text

With the DrwText() function, you can display a string at any position on the display (pixel-level layout control).

Declaration:	void DrwText (int __X , int __Y , char * __Buf , int __len , int __attributes , int __Size , uint32 __Color)
Parameter:	<p>int __X X coordinate beginning of the text (first letter on the left)</p> <p>int __Y Y coordinate beginning of the text (first letter top)</p> <p>char * __Buf pointer on first character in string</p> <p>int __len string length, numbers of characters to be output</p> <p>int __attributes for the attribute, refer to the table; bits 4 and 5 are replaced by "size"</p> <p>int __size text size</p> <p>0=8x6 pixels</p> <p>1=16x6 pixels</p> <p>2=32x6 pixels</p> <p>uint32 __Color colour</p>
Return value	none

Example:

```
/* Text output "Hello world", underlined, in 16x12 pixel font*/
char output[11] = {'Hello world'};
DrwText(3,10,output,0x04,1,0);
```

3.2.13.12 Moving area on the display

With the MoveArea() function, you can move an already drawn area of the page content. Application: e.g. if a curve progression is required without saving the old values.

Declaration:	void MoveArea (int __XLeftOld , int __YTopOld , int __Width , int __Height , int __XLeftNew , int __YTopNew , int Movemode)
Parameter:	<p>int __XLeftOld X coordinate old position</p> <p>int __YTopOld Y coordinate old position</p> <p>int __Width width of the area to be moved</p> <p>int __Height height of the area to be moved</p> <p>int __XLeftNew X coordinate new position</p> <p>int __YTopNew Y coordinate new position</p> <p>int __MoveMode Bit 7: 0=indirect 1=direct</p> <p>Bits 0-6:</p> <p>0 = static layer, non-blinking</p> <p>1 = static layer, blinking</p> <p>2 = dynamic layer, non-blinking</p> <p>3 = dynamic layer, blinking</p>
Return value	none

Operating panel manual

3.2.13.15 Deleting pointer instrument

The function described within this section is able to delete an analogue instrument which has been dynamically generated during the cycle.

After this function has been called up, the ShowAnalogView(...) function must not be called up again for this instrument !

Declaration:	void DeleteAnalogView (int __InstrNo)
Parameter:	int __InstrNo handle of the pointer instrument
Return value	none

Example:

```
DeleteAnalogView(speedmeter);
```

3.2.14 Functions for the touch screen

The touch screen values can be retrieved and modified via the following parameters. Unless otherwise indicated, the pixel position is returned, whereby the top left corner is assigned the (0;0) coordinate.

3.2.14.1 Getting X position of the currently touched position

Declaration:	int Get_X_Move (void)
Parameter:	none
Return value	int X position of the currently touched position (-1 = not touched)

3.2.14.2 Getting Y position of the currently touched position

Declaration:	int Get_Y_Move (void)
Parameter:	none
Return value	int Y position of the currently touched position (-1 = not touched)

3.2.14.3 Getting X position of the first touched position

Declaration:	int Get_X_Down (void)
Parameter:	none
Return value	int X position of the first touched position

3.2.14.4 Getting Y position of the first touched position

Declaration:	int Get_Y_Down (void)
Parameter:	none
Return value	int Y position of the first touched position

Operating panel manual

3.2.14.5 Getting absolute analogue value of X position

Declaration:	int Get_X_Value (void)
Parameter:	none
Return value	int absolute analogue value X position

3.2.14.6 Getting absolute analogue value of Y position

Declaration:	int Get_Y_Value (void)
Parameter:	none
Return value	int absolute analogue value Y position

3.2.14.7 Getting calibration value for X

Declaration:	int Get_X_Cali (void)
Parameter:	none
Return value	int calibration value for X

3.2.14.8 Getting calibration value for Y

Declaration:	int Get_Y_Cali (void)
Parameter:	none
Return value	int calibration value for Y

3.2.14.9 Getting tolerance value

Declaration:	int GetTouchTol (void)
Parameter:	none
Return value	int tolerance value

3.2.14.10 Setting tolerance value

Declaration:	void SetTouchTol (int __Value)
Parameter:	int __Value tolerance value
Return value	none

Operating panel manual

3.2.15 Functions for edge evaluation

Many functions require an edge evaluation. We have taken this fact into consideration and offer the following functions. Each request must obtain a unique "EdgeNo" number within a range from 0 to 2047.

3.2.15.1 Evaluation of both edges

This sub-program checks an input value for an edge, no matter whether it is a rising or falling edge. Logically, this edge is active only once.

Declaration:	int AnyEdge (int Signal , int EdgeNo)
Parameter	int Signal signal to be requested int EdgeNo a clear number (0..2047) for the request
Return value	int 0=no edge in the signal comparison compared to the last function call-up <>0=signal has changed compared to the last function call-up

Example:

```
if (AnyEdge(IsMessageOn(5),3))
```

3.2.15.2 Evaluation of the rising edge

This sub-program checks an input value for a rising edge. Logically, this edge is active during one cycle.

Declaration:	int RisingEdge (int Signal , int EdgeNo)
Parameter	int Signal signal to be requested int EdgeNo a clear number (0..2047) for the request
Return value	int 0=no edge in the signal comparison compared to the last function call-up <>0=signal has changed from 0 to 1 compared to the last function call-up

Example:

```
if (RisingEdge(IsKeyDown(2),10))  
  PageOn(5);
```

3.2.15.3 Evaluation of the falling edge

This sub-program checks an input value for a falling edge. Logically, this edge is active during one cycle.

Declaration:	int FallingEdge (int Signal , int EdgeNo)
Parameter	int Signal signal to be requested int EdgeNo a clear number (0..2047) for the request
Return value	int 0=no edge in the signal comparison compared to the last function call-up <>0=signal has changed from 1 to 0 compared to the last function call-up

Example:

```
if (FallingEdge(IsKeyTouched(2),10))  
  PageOff(5);
```

3.2.16 Functions for the video input

These functions are used to visualise, the video signal of a video input (device-dependent!) on the display.

For vehicle applications, for example, a back-up camera could be used as video source.

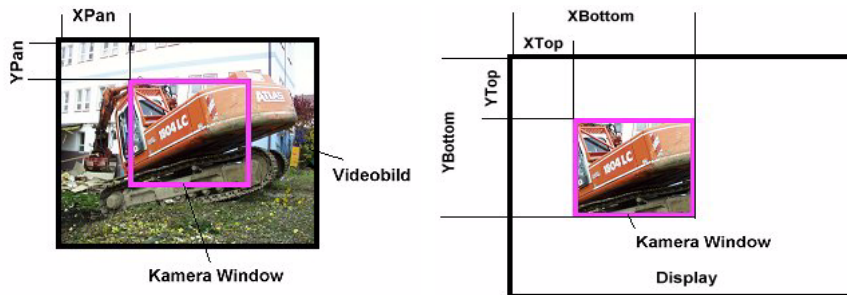
Operating panel manual

3.2.16.1 Defining camera window

The CameraWindowSet(...) function processes the image transferred by the video input in accordance with the size and page cut-out area specified in the display parameters. The page does not yet appear!

Declaration:	void CameraWindowSet (int __XTop, int __YTop, int __XBottom, int __YBottom, int __XPan, int __YPan)
Parameter:	int __XTop int __YTop int __XBottom int __YBottom int __XPan int __YPan
Return value	none

The following graphic shows the setting of parameters. The __Pan parameters indicate the point on the camera window which is to appear at the top-left window point (XTop, YTop).



3.2.16.2 Switching on camera window

With the CameraWindowOn function, the camera window can be displayed in the pre-defined size.

Declaration:	void CameraWindowOn (int __WindowPrio)
Parameter:	int __WindowPrio layer onto which the camera window is displayed
Return value	none

3.2.16.3 Switching off camera window

With the CameraWindowOff function, the camera window can be switched off.

Declaration:	void CameraWindowOff (void)
Parameter:	none
Return value	none

3.2.17 Other functions

3.2.17.1 Requesting current device status

The GetCurrentDeviceStatus() function provides you with the current device status. The diverse device states are listed in the "Communication" manual, chapter Report Status Telegram, Operating Panel Status.

Operating panel manual

These values correspond to the return values of this function.

Declaration:	int GetCurrentDeviceStatus (void)
Parameter:	none
Return value	int device status

Example:

```
Status = GetCurrentDeviceStatus();
```

3.2.17.2 Requesting error status

The `Get_ERROR_Status()` function provides the error status of the device.

Declaration:	int Get_ERROR_Status (void)
Parameter:	none
Return value	uint16 error status

Example:

```
Error = Get_ERROR_Status();
```

3.2.17.3 Requesting time zone

The `GetCurrentTimezone()` function indicates the selected time zone:

Declaration:	int GetCurrentTimezone (void)
Parameter:	none
Return value	int 0 = winter time 1 = summer time 2 = no time zone

Example:

```
Timezone = GetCurrentTimezone();
```

Operating panel manual

3.2.17.4 Setting time zone

The SetCurrentTimezone(...) function indicates the time zone to be selected:

Declaration:	void SetCurrentTimezone (uint16 tz)
Parameter:	tz: 0 = winter time 1 = summer time 2 = no time zone
Return value	none

When switching over from summer to winter time, the real-time clock of the device is automatically put forward by one hour. Respectively, the clock is put back by one hour if the winter time is set to summer time. If winter time has been set and the device is switched over to winter time, the clock setting remains unchanged; the same applies to the summer time.

Example:

```
SetCurrentTimezone(1); /* sets summer time */
```

3.2.17.5 Getting configuration of the printer interface

Outputs the currently set printer

Declaration:	int GetPrinterInterfaceIndex (void)
Parameter:	none
Return value	0=no printer 1 = serial printer 2 = printer to CAN module with SELECAN-ID 3 = print data as ASCII telegram on individual ID

3.2.17.6 Configuring printer interface

This function serves to indicate the printer interface.

Declaration:	void SetPrinterInterfaceIndex (uint16 value)
Parameter:	Value: 0 = no printer 1 = serial printer 2 = printer to CAN module with SELECAN-ID 3 = print data as ASCII telegram on individual ID
Return value	none

0=no printer
1 = serial printer
2 = printer to CAN module with SELECAN-ID
3 = print data as ASCII telegram on individual ID

Accept:

Immediately. After a re-start only if SaveToFlash has been executed.

Operating panel manual

3.2.17.7 Checking which PLC driver has been loaded

This function is normally negligible. Usually, you know which settings you have made. A modification is not possible anyway.

Declaration:	int GetPLCDriverNo (void)
Parameter:	none
Return value	int 0 = no driver

3.2.17.8 Getting BIOS version

With the GetBIOSVersion function, the BIOS version of the device can be retrieved.

Declaration:	void GetBIOSVersion (char * Buf , int Len)
Parameter:	*Buf pointer on string with BIOS version ITB-nnnSmm. nnn identifies the BIOS status and mm the version (usually00) Len length of the string
Return value	none

Example:

```
char BV[10];  
GetBIOSVersion(&BV[0],10);
```

3.2.17.9 Getting TOS version

With the GetTOSVersion function, the TOS version of the device can be retrieved.

Declaration:	void GetTOSVersion (char * Buf , int Len)
Parameter:	*Buf pointer on string with TOS version ITB-nnnSmm. nnn identifies the TOS status and mm the version (usually 00) Len length of the string
Return value	none

Example:

```
char TV[10];  
GetTOSVersion(&TV[0],10);
```

3.2.17.10 Getting project version

With the GetUSERVersion function, the project version which has been entered in the USERDATA field can be retrieved.

Declaration:	void GetUSERVersion (char * Buf , int Len)
Parameter:	*Buf pointer on string with USER version The string has no fixed structure. Len length of the string
Return value	none

Example:

```
char UV[15];  
GetUSERVersion(&UV[0],15);
```

Operating panel manual

3.2.17.11 Getting initialisation flags

A detailed description follows.

Declaration:	uint16 GetInitFlags (void)
Parameter:	none
Return value	uint16 initialisation flags

3.2.17.12 Writing initialisation flags

A detailed description follows.

Declaration:	void SetInitFlags (uint16 Bits)
Parameter:	Bits: flags for initialisation
Return value	none

3.2.17.13 Saving setup data in the project FLASH

Many of the project configuration data can be modified. Some of the data are immediately accepted, others only after they have been saved in the FLASH and the device has been re-started.

If SETUP data should keep their values after a re-start, they must be saved in the FLASH after they have been modified (CAN bus rate, identifier etc.). As the entire data record (all data) is saved, the function has no parameters.

Depending on which configuration data have been changed, the device must be re-started after the SaveToFlash() function has been performed. For this purpose, activate the **ExecCANTelegramInternal** function with D0=0x12 (also refer to the Communication manual).

Declaration:	void SaveToFlash (void)
Parameter:	none
Return value	none

3.2.17.14 Calling up device function via CAN call-up convention

This function is certainly the most important function within the device operating system. Not all device functions are available as independent sub-programs. This functions allows to internally execute each device function which can be called up via the interface. Only the parameters D0 to D7 must be filled in with the respective data (refer to the Communication manual).

This way, even device functions are accessible which have not been explicitly programmed in the system API.

Declaration:	void ExecCANTelegramInternal (char D0 , char D1 , char D2 , char D3 , char D4 , char D5 , char D6 , char D7)
Parameter:	char D0 : data byte D0 char D1 : data byte D1 char D2 : data byte D2 char D3 : data byte D3 char D4 : data byte D4 char D5 : data byte D5 char D6 : data byte D6 char D7 : data byte D7
Return value	none

Example:

Operating panel manual

ExecCANTelegramInternal (0x12,0,0,0,0,0,0,0); /* device reset */

3.2.17.15 Buzzer Ein/Ausschalten

Using this function for devices with Buzzer, the behavior of the buzzer can be programmed,

Declaration:	void SetBuzzer (uint8 __Mode, long __BuzzerOnTime, long BuzzerOffTime, long __Count)
Parameter:	<p>__Mode: 0=Buzzer off, all other parameters are ignored 1=Buzzer on, quiet 3=Buzzer on, loud</p> <p>__BuzzerOnTime:determines, how long the buzzer is on (in 10ms-Steps) __BuzzerOffTime:determines, how long the buzzer is off (in 10ms-Steps) __Count: determines how often a On-Off period is 0=endless</p>
Return value	none

Examples

If Mode is zero then the buzzer is switched off	SetBuzzer(0,0,0,0)	Buzzer off
If Mode is not zero and BuzzerOnTime and BuzzerOffTime are both zero, then the buzzer is switched on	SetBuzzer(3,0,0,0)	Buzzer on, loud
If Mode is not zero and BuzzerOnTime only is not zero, then the buzzer is switched on for exactly this (Single-Beep)	SetBuzzer(1,100,0,0)	Buzzer on for 1 Second, quiet
If Mode is not zero and BuzzerOnTime and BuzzerOffTime are both not then the buzzer is used periodically	SetBuzzer(3,30,20,0) SetBuzzer(3,30,20,5)	300ms on, 200ms off endless, loud 300ms on, 200ms off 5 times, loud

3.2.17.16 Examine buzzer status

The buzzer's current status can be examined using this function. The return value is coded in bits. Bit 0 = On/Off, Bit 1 = Quiet/loud, Bit 7 = not periodically/periodically

Declaration:	uint8 GetBuzzer (void)
Parameter:	none
Return value	<p>0x00 Buzzer is off 0x01 Buzzer is on, quiet 0x03 Buzzer is on, loud 0x80 Buzzer is off at the moment, but is used periodically 0x81 Buzzer is quiet on, but is used periodically 0x83 Buzzer is loud on, but is used periodically</p>

Operating panel manual

3.2.18 Flash functions

A part of the FLASH memory is reserved for the user.

A pseudo-byte-array with 32000 bytes is made available in a transparent way.

Prerequisite for using FLASH functions:

- The device must be equipped with a real-time clock.
- The user project must not exceed a certain size.

3.2.18.1 Testing whether data can be saved in the Flash memory.

The FlashTest() function determines whether the respective requirements are met. Only then can you read from or written into the Flash via the FlashRead or FlashWrite functions.

The user himself must ensure that the parameters are useful. He must particularly provide for sufficient memory capacities at *__pDest. (at least __Len bytes).

Declaration:	int FlashTest (void)
Parameter:	none
Return value	int <>0: data can be saved in the Flash =0: data cannot be saved in the Flash

For an example, see below.

3.2.18.2 Getting data from the user Flash

Declaration:	void FlashRead (uint8 *__pDest, long __Index, long __Len)
Parameter:	uint8 *__pDest pointer on the area where the user wants to save the data from the flash long __Index index of the first byte to be read in the pseudo-byte-array (Flash) long __Len number of bytes to be retrieved from the flash
Return value	

For an example, see below.

Operating panel manual

3.2.18.3 Writing data into the user Flash

For the user, the flashing process is carried out transparently. The FlashWrite function autonomously determines the required steps such as buffering data, deleting flash and burning.

The user must only activate the FlashWrite call-up.

A flash is used which supports approx. 100000 writing cycles. These functions should therefore only be used for saving parameter data etc., but not for data which are subject to a permanent modification. The user himself must ensure that the parameters are useful. He must particularly check whether `__Index+__Len < 32000`.

Declaration:	void FlashWrite (long __Index , uint8 * __pSource , long __Len)	
Parameter:	long __Index	index of the first byte to be written in the pseudo-byte-array (Flash)
	uint8 * __pSource	pointer on the area where the data to be saved are located
	long __Len	number of bytes to be written into the flash
Return value	none	

Example:

This code segment would copy 40 bytes from index 100 to 200:

```
uint8 UserData[40];
```

```
if (FlashTest())  
{  
    FlashRead(UserData, 100, sizeof(UserData));  
    FlashWrite(200, UserData, sizeof(UserData));  
}
```