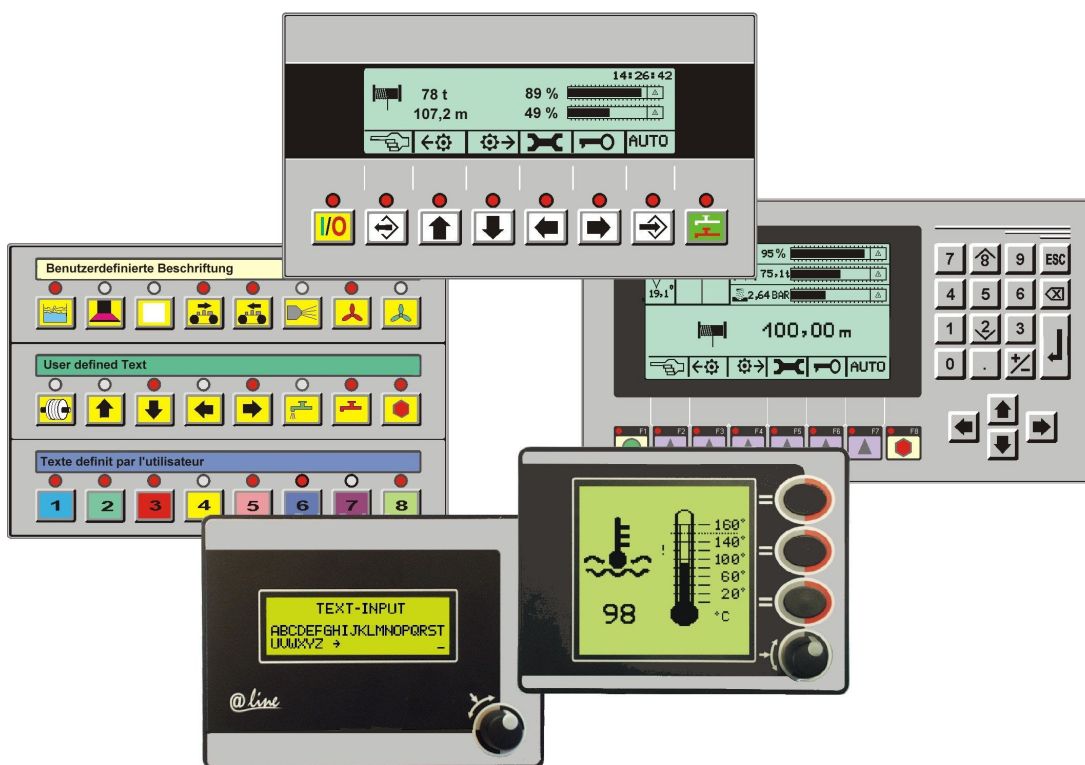




GRAF-SYTECO

Handbuch Steuern mit C



Dokument: H096A2
Status: Freigegeben
Erstellt: März 2004

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

GRAF-SYTECO GmbH & Co.KG * Kaiserstrasse 18 * D-78609 Tuningen
Tel: +49 7464 98660 * Fax: +49 7464 2550 * [http:// www.graf-syteco.de](http://www.graf-syteco.de) * eMail: info@graf-syteco.de

Handbuch Bediengeräte

1 Einführung

1.1 Integrierte Entwicklungsumgebung GCE

Mit dem Softwarepaket ITE wird zur Erstellung von Anwender-Steuerungsprogrammen in C eine einfach zu handhabende Entwicklungsumgebung mit ANSI-Code-Editor und Projektverwaltung mitgeliefert. Um die Entwicklungsumgebung nutzen zu können muß mindestens die Version „D“ des Softwarepaketes ITE installiert sein. In der kostenlosen Basisversion „C“ läuft die Entwicklungsumgebung nur im DEMO-Mode.

Grundsätzlich ist ein Bediengerät der AT-Serie Voraussetzung. Geräte der ITS-Serie unterstützen die C-Programmierung nicht.

GCE kapselt die Aufrufe von C-Compiler, Linker und Hex-Konverter intern, und zeigt Fehlermeldungen, die beim Übersetzen des Projektes auftreten an.

Die Schnittstelle zum Betriebssystem (TOS) des AT-Bediengerätes wird als C-Datei-Schablone mitgeliefert.

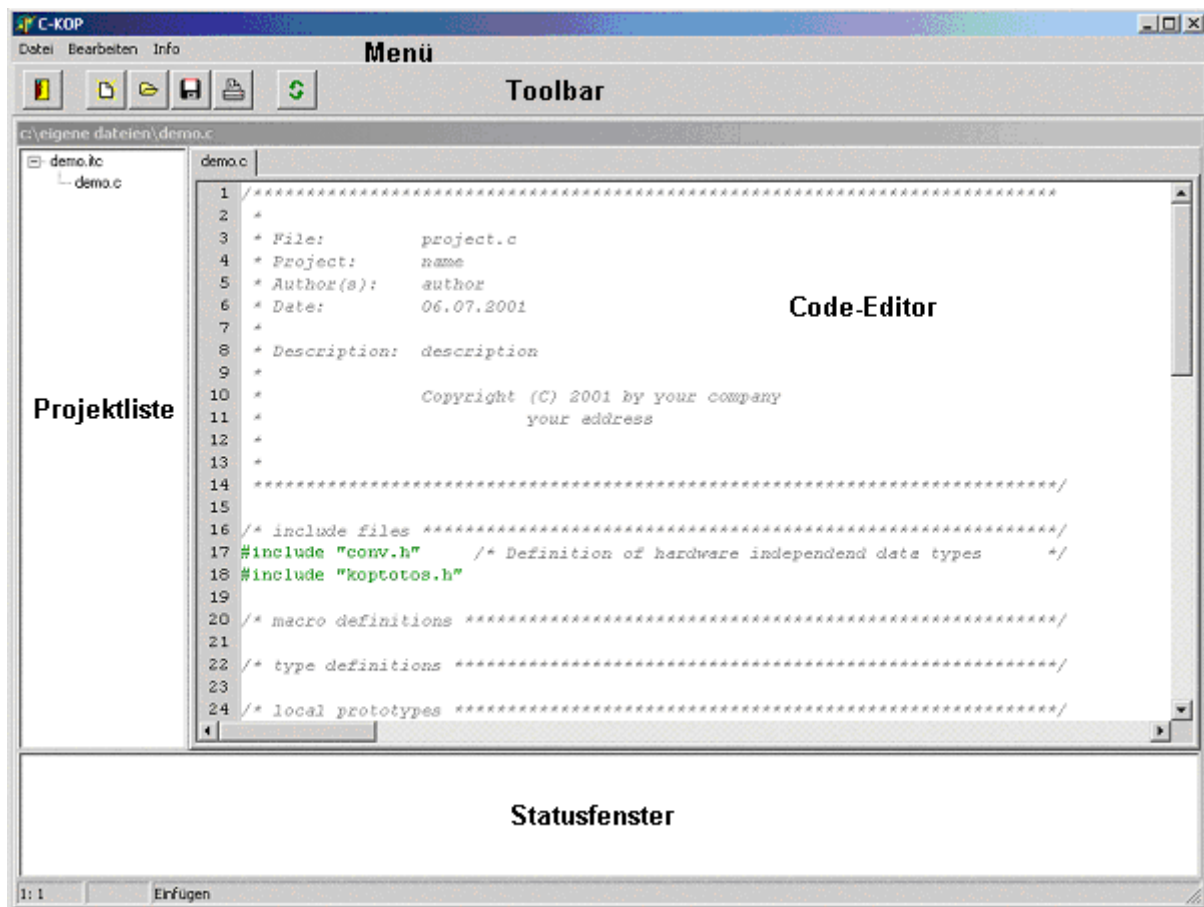
Der Start des GCE erfolgt aus dem Editor ITE heraus. Die Schaltfläche zum Starten von GCE ist nur freigegeben, wenn in der Voreinstellungsmaske ein Gerät der AT-Serie eingestellt ist.

Der aktuelle Projektname wird dabei als GCE-Projektname übernommen(z.B. ITE-Projektname ist Maschine.ITE, dann heißt das GCE-Projekt Maschine.ITC.

Existiert für das Projekt noch keine TOS-Schnittstellendatei mit dem Rahmencode für die vom TOS aufgerufenen Funktionen, dann wird diese aus einer Schablone heraus generiert. Die Schablone ist die Datei Template.C im Verzeichnis C:\Programme\Graf_ITE\bin.

Der Dateiname der automatisch generierten Datei wird auf <Projektname.c> voreingestellt (z.B. Maschine.C).

1.1.1 Oberfläche



Handbuch Bediengeräte

1.1.2 Eigenschaften des C Code-Editors

Der C Code-Editor ist ein erweiterter ANSI-Editor, speziell optimiert für die Bedürfnisse von Programmierern und bietet folgende Programmierhilfen:

- *Syntax-Highlighting*, das bedeutet, dass Kontrollstrukturen und Schlüsselwörter farbig dargestellt werden. Diese Funktion ist für Dateien mit den Erweiterungen `.c` und `.h` verfügbar.
- Zeilennummern sind automatisch eingeblendet, diese Option kann in den Voreinstellungen geändert werden.
- Zeilen und Spaltenmarkierung ist möglich.
- Ausschneiden, kopieren, einfügen und löschen von Blöcken.
- Suchen und ersetzen.
- Es können nahezu beliebig große Dateien editiert werden.

1.1.2.1 Zeilen markieren

Um Zeilen zu markieren, halten Sie die **<Shift>** Taste gedrückt, während Sie mit einer Pfeiltaste den Cursor verschieben.

1.1.2.2 Spalten markieren

Drücken Sie die Tastenkombination **<SHIFT>+<ALT>+<Pfeiltasten>** um Spalten zu markieren. Danach können Sie die markierten Spalten ausschneiden, kopieren, löschen oder verschieben. Spalten lassen sich auch markieren, indem Sie die **<Alt>**-Taste beim Klicken und Ziehen mit der Maus gedrückt halten

```
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 Alles markieren

Drücken Sie die Tastenkombination **<Strg>+<a>** um den gesamten Text zu markieren.

1.1.2.4 Block ausschneiden

Die Tastenkombination **<Strg>+<x>** schneidet den markierten Block aus und kopiert ihn in die Zwischenablage.

1.1.2.5 Block kopieren

Die Tastenkombination **<Strg>+<c>** kopiert den markierten Block in die Zwischenablage ohne ihn auszuschneiden.

1.1.2.6 Block löschen

Wurde ein Block markiert kann er mit der Taste **<Entf>** einfach gelöscht werden.

1.1.2.7 Block einfügen

Wurde ein Text in die Zwischenablage kopiert, dann kann dieser mit der Tastenkombination **<Strg>-<v>**

Handbuch Bediengeräte

an der Cursorposition eingefügt werden

1.1.2.8 Zeile löschen

Eine ganze Zeile löschen Sie mit **<Strg>-<y>**

1.1.2.9 Zeile ab Cursor bis zum Zeilenende löschen

Wenn Sie den Rest einer Zeile ab der Cursorposition löschen wollen drücken Sie **<Strg>-<t>**

1.1.2.10 Neue Zeile einfügen

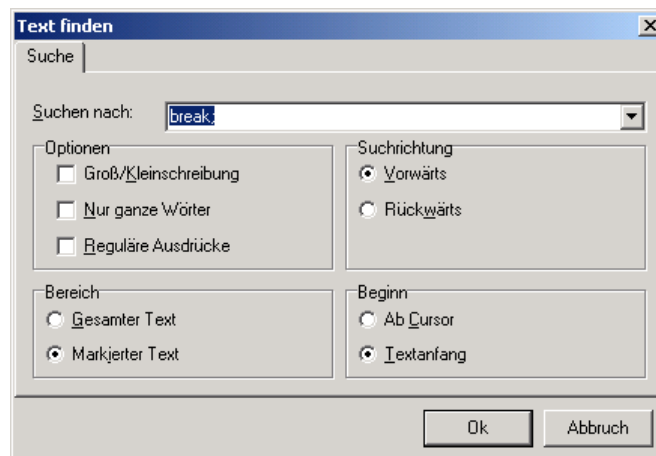
Eine neue Zeile fügen Sie mit **<Strg>-<n>** ein.

1.1.2.11 Undo-Funktion

Und wenn Sie versehentlich etwas gelöscht haben, oder sonst irgend eine Änderung rückgängig machen wollen, dann drücken Sie einfach **<Strg>-<z>**

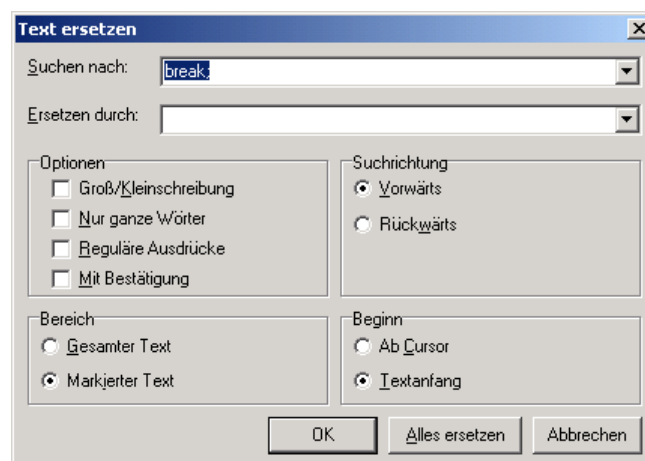
1.1.2.12 Einblenden des Such-Dialoges

Bei der Arbeit mit Editoren ist es ab und zu notwendig, im Text nach bestimmten Begriffen zu suchen. Die Tastenkombination **<CTRL>+<f>** öffnet den Suchdialog, mit dem Sie nach den gewünschten Textstellen suchen können



1.1.2.13 Einblenden des Ersetzen-Dialoges

Genauso kommt es ab und zu vor, daß ein Begriff an mehreren Stellen im Text ersetzt werden soll. Die Tastenkombination **<CTRL>+<r>** sorgt dafür, daß der Ersetzen-Dialog aufgerufen wird



Handbuch Bediengeräte

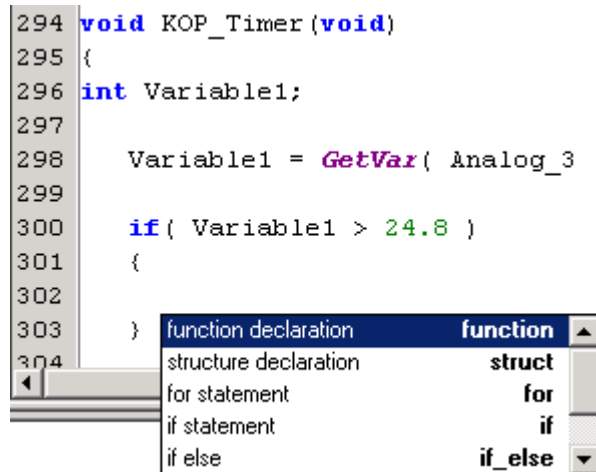
1.1.2.14 Weitersuchen / -ersetzen

Wurde zuvor nach einem Begriff gesucht oder ersetzt, und die Suche oder das Ersetzen unterbrochen, dann kann sie mit der Tastenkombination **<CTRL>+<e>** wieder aufgenommen werden.

1.1.2.15 Code-Schablonen einfügen

Machen Sie es sich bequem. Fügen Sie fertige Code-Schablonen für Ihre gängigen Kontrollstrukturen in Ihren Quellcode ein, indem Sie die Tastenkombination **<CTRL>+<j>** drücken und die gewünschte Kontrollstruktur aus der sich öffnenden Liste wählen

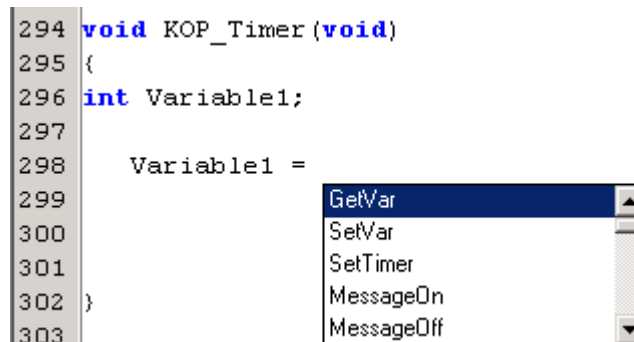
```
294 void KOP_Timer(void)
295 {
296 int Variable1;
297
298     Variable1 = GetVar( Analog_3 )
299
300     if( Variable1 > 24.8 )
301     {
302
303     }
304
```



1.1.2.16 Systemfunktionen aus einer Liste wählen

Drücken Sie die Tastenkombination **<ALT>+<j>**. Es erscheint eine Liste mit allen in der Datei KOPTOTOS.H deklarierten Systemfunktionen. Wählen Sie die gewünschte Funktion aus und drücken Sie die Enter-Taste.

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



Handbuch Bediengeräte

1.1.2.17 Variablen aus einer Liste wählen

Wie bei den Systemfunktionen, können Sie auch System- und Projektvariablen aus einer Liste auswählen und direkt in Ihren Quellcode übernehmen. Drücken Sie dazu die Tastenkombination **<ALT>+<v>**

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

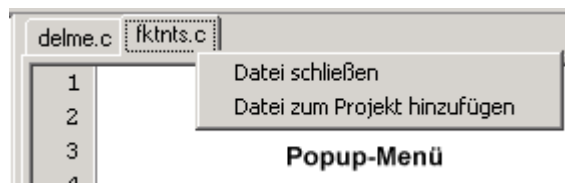
Analog_1	80
Analog_2	81
Analog_3	82
Analog_4	83
Analog_5	84

1.1.3 Eigenschaften der Projektdatensliste

In die Projektliste müssen alle zum Projekt gehörenden Dateien aufgenommen werden. Auch alle selbst erstellten und mit #include eingebundenen Headerdateien, sowie Dokumentationsdateien die nicht kompiliert werden. Zum Bearbeiten der Projektliste (z.B. zur Aufnahme existierender Dateien in die Projektliste) steht nach Drücken der *rechten* Maustaste ein POPUP-Menü zur Verfügung.



Eine bereits geöffnete Datei kann nachträglich in die Projektliste aufgenommen werden. Aktivieren Sie hierzu das entsprechende Editorfenster, positionieren Sie den Mauszeiger auf das Register mit dem Dateinamen und drücken Sie die *rechte* Maustaste.



Handbuch Bediengeräte

1.1.4 Eigenschaften des Statusfensters

Das Statusfenster zeigt den aktuellen Status der Übersetzung des Projektes an. Im Fall eines Fehlers, das zum Abbruch des Compilerprozesses führt, werden hier die Fehlermeldungen angezeigt. Durch einen Mausklick auf einen Eintrag in dieser Liste springt der Cursor automatisch an die Stelle im Quelltext, die den Fehler verursacht.

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

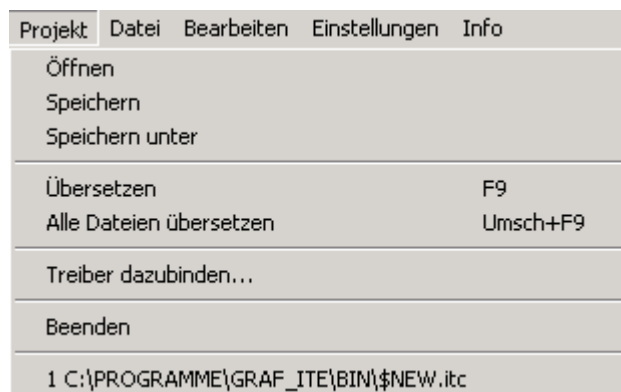
1: 1 Einfügen c:\programme\graf_ite\tos\delme2.c

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

1.2 Das Menü

1.2.1 Das Menü Projekt

Das Projektmenü dient der Verwaltung des aktuell geladenen Projektes. Sie finden hier Menüpunkte zum öffnen, speichern und übersetzen usw., sowie die Projektnamen der maximal vier zuletzt geöffneten Projekte zum direkten Öffnen.



Handbuch Bediengeräte

1.2.1.1 Öffnen

Ruft den Windows-eigenen „Datei-Öffnen-Dialog“ auf um ein GCE-Projekt zu öffnen. GCE-Projekte haben die Dateierweiterung .ITC.

Hinweis:

Dieser Befehl steht nicht zur Verfügung, wenn das GCE-Projekt aus dem Editor ITE heraus gestartet, oder wenn das Projekt durch einen Doppelklick auf die .ITC-Datei im Explorer geöffnet wurde.

1.2.1.2 Speichern

Speichert das aktuell geladene Projekt ab. Hierbei wird die .ITC-Datei überschrieben.

1.2.1.3 Speichern unter

Speichert das aktuell geladene Projekt unter einem neuen Namen ab.

Hinweis:

Dieser Befehl steht nicht zur Verfügung, wenn das GCE-Projekt aus dem Editor ITE heraus gestartet, oder wenn das Projekt durch einen Doppelklick auf die .ITC-Datei im Explorer geöffnet wurde.

1.2.1.4 Übersetzen

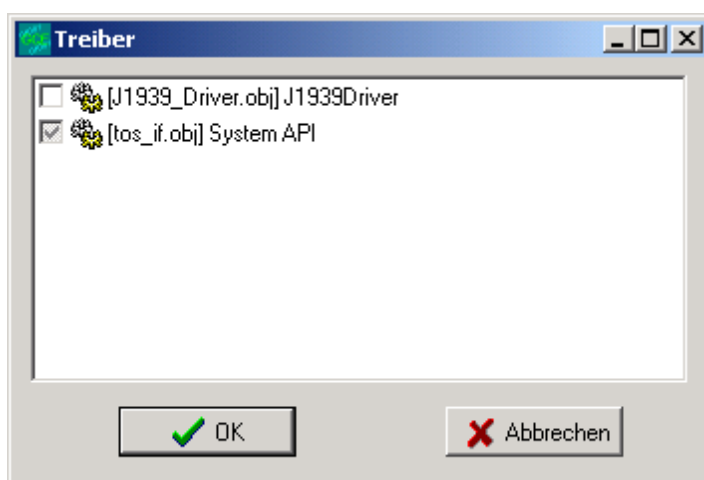
Ruft intern den C-Compiler und Linker auf, um das Projekt zu übersetzen. Wird dieser Menüpunkt ausgewählt, werden nur die Dateien mit dem Compiler übersetzt, die sich seit dem letzten Übersetzungslauf geändert haben (Make). Die Ausgabe des C-Compilers und des Linkers erscheinen dabei im Statusfenster. Sind Fehler im Quelltext, dann werden diese ebenfalls im Statusfenster angezeigt. Durch einen Doppelklick auf eine Fehlermeldung springt der Code-Editor direkt an die Fehler verursachende Stelle.

1.2.1.5 Alle Dateien übersetzen

Dieser Menüpunkt funktioniert wie der Menüpunkt „Übersetzen“, jedoch werden grundsätzlich alle Quelldateien vom C-Compiler übersetzt (Build).

1.2.1.6 Treiber dazubinden

Wenn für eine bestimmte Funktionalität im Bediengerät ein Treiber installiert ist, und man möchte Funktionen des Treibers im eigenen Anwenderprogramm aufrufen, dann muß die Objekt-Datei (xxx.obj) des Treibers zum Anwenderprogramm dazugebunden werden. Der Menüpunkt ruft das folgende Fenster auf, mit dem bestimmt werden kann, welche Treiber dazugebunden werden. Aktivieren Sie einfach das Kästchen mit dem entsprechenden Treiber.



Hinweis:

Die Einstellung ist projektabhängig. Binden Sie in Ihr Projekt nur die Treiber ein, die aus denen Sie in Ihrem Anwenderprogramm Funktionen aufrufen.

Wenn Ihr Projekt Treiber verwendet, und Sie Ihr Projekt an andere Anwender weitergeben, dann müssen Sie sicherstellen, daß dieser mit dem Projekt die Treiberdatei (.OBJ) und die zugehörige Headerda-

Handbuch Bediengeräte

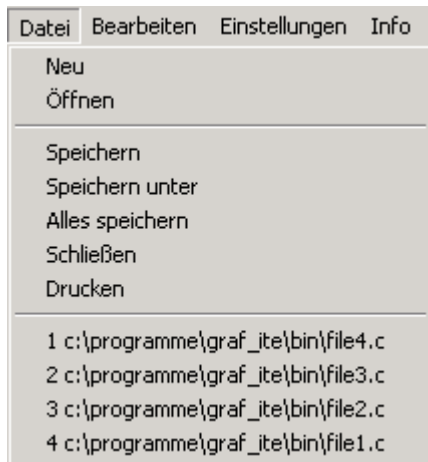
tei (.H) erhält und installiert. Ansonsten läßt sich das Projekt vom Anwender nicht „linken“. Wie Treiber installiert werden. Siehe “Treiber verwalten” auf Seite 10.

1.2.1.7 Beenden

Beendet das Programm nach Rückfrage, ob das aktuell geladene Projekt gespeichert werden soll.

1.2.2 Das Menü Datei

Das Dateimenü enthält Menüpunkte zum Verwalten von Dateien. Jeder Menüpunkt bezieht sich dabei auf die momentan aktive Datei im Code-Editor. Sind mehrere Editierfenster geöffnet, dann ist trotzdem nur eines aktiv. Außer den Menüpunkten zum Speichern, Öffnen, .. usw. finden Sie noch die Dateinamen der maximal vier zuletzt geöffneten Dateien.



1.2.2.1 Neu

Legt eine neue Datei an. In dem aufgerufenen Fenster können Sie angeben, welche Art von Datei erzeugt werden soll.



Geben Sie einen Dateinamen mit oder ohne Dateierweiterung ein. **Der Dateiname muß in jedem Fall mit einem Buchstaben (A-Z, a-z) beginnen !**

Wählen Sie den Typ der neuen Datei aus, und ob die Datei zum Projekt hinzugefügt werden soll. Wird sie hinzugefügt, dann wird sie auch beim Speichern des Projektes in einem neuen Pfad automatisch mitkopiert.

Handbuch Bediengeräte

1.2.2.2 Öffnen

Öffnet eine vorhandene Datei.

1.2.2.3 Speichern

Speichert die im aktiven Editor-Fenster geöffnete Datei ab.

1.2.2.4 Speichern unter

Speichert die im aktiven Editor-Fenster geöffnete Datei unter einem neuen Namen ab.

1.2.2.5 Alles speichern

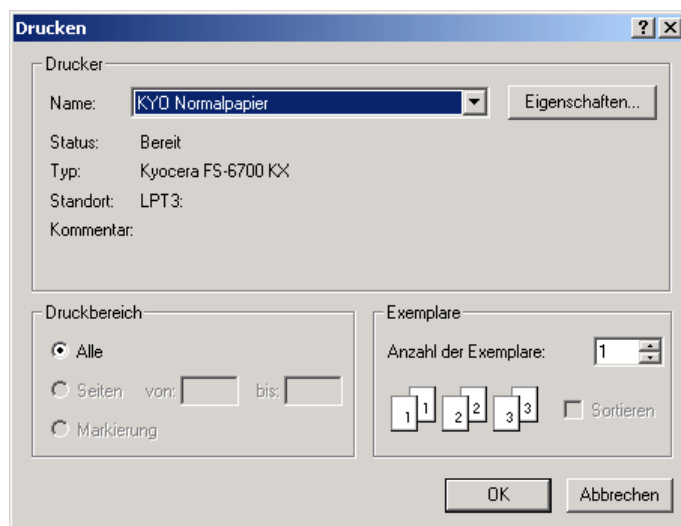
Speichert alle geöffneten Dateien aus allen Editorfenstern.

1.2.2.6 Schließen

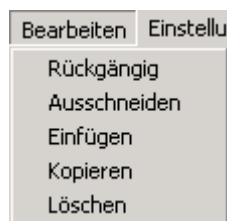
Schließt die im aktiven Editor Fenster geöffnete Datei nach Rückfrage, ob sie vor dem Schließen gespeichert werden soll. Die Datei wird dabei nicht aus dem Projekt entfernt !

1.2.2.7 Drucken

Druckt die im aktiven Editor-Fenster geöffnete Datei auf dem Drucker aus. Es wird der Standard Druck-dialog aufgerufen.



1.2.3 Das Menü Bearbeiten



1.2.3.1 Rückgängig

macht die letzten Eingaben rückgängig.

1.2.3.2 Ausschneiden

Schneidet den markierten Text aus und kopiert ihn in die Zwischenablage.

Handbuch Bediengeräte

1.2.3.3 Einfügen

Fügt den Text aus der Zwischenablage in das aktive Editorfenster an der Cursorposition ein.

1.2.3.4 Kopieren

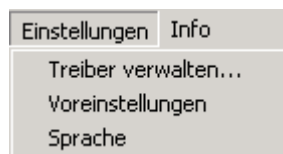
Kopiert den markierten Text in die Zwischenablage.

1.2.3.5 Löschen

Schneidet den markierten Text aus ohne ihn in die Zwischenablage zu kopieren.

1.2.3.6 Das Menü Einstellungen

Unter diesem Menü werden die Programmeinstellungen verwaltet.

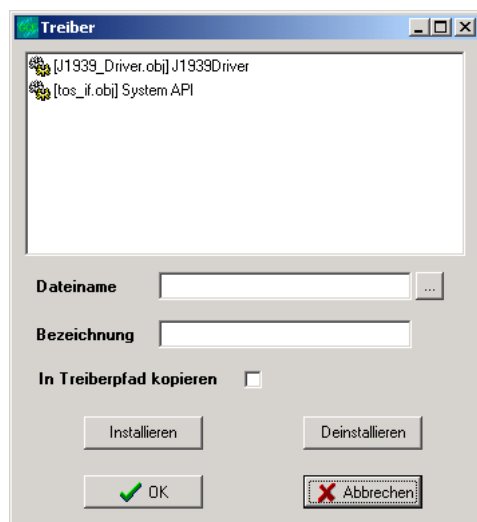


1.2.3.7 Treiber verwalten


Wenn Sie von selbst geschriebenen Programmteilen den Quelltext nicht weitergeben, aber dem Anwender dennoch Ihre Funktionen zur Verfügung stellen wollen, dann müssen Sie dem Anwender zumindest die übersetzte .OBJ Datei und eine Headerdatei (.H) zur Verfügung stellen. Die .OBJ-Datei die vom C-Compiler erzeugt wird enthält Ihren Code, die .H-Datei enthält die Prototypen der Funktionen, die der Anwender aus Ihrer Treiberdatei aufrufen kann. Diese Headerdatei muß der Anwender mit der #include -Anweisung in sein Programm einbinden, und die .OBJ-Datei mit seinem Projekt linken.

GRAF-SYTECO bietet für bestimmte Applikationen eigene Treiber an, die an dieser Stelle installiert werden können (z.B. den J1939-Treiber für die CAN-Kommunikation mit Geräten und Aggregaten, die dieses Protokoll unterstützen)

Damit Funktionen aus dem Treiber aufgerufen werden können und das Linken fehlerfrei ausgeführt wird, muß die .OBJ-Datei des Treibers zusammen mit der gleichnamigen Headerdatei im Anwendersystem installiert werden. Dies geschieht über den folgenden Dialog



Treiber installieren

Geben Sie unter „Dateiname“ den Namen und den Pfad des zu installierenden Treibers ein. Mit der Schaltfläche  können Sie dazu einen „Datei-Öffnen“ Dialog aufrufen. Voraussetzung für die Installation des Treibers ist die Existenz beider, der .OBJ **und** der zugehörigen gleichnamigen .H-Datei (z.B. J1939Driver.obj und J1939Driver.h).

Im Feld Bezeichnung geben Sie einen beliebigen Namen für den Treiber ein. Dieser Name dient nur der Übersichtlichkeit und hat weiter keine Bedeutung.

Handbuch Bediengeräte

Wenn der Treiber in das Treiberverzeichnis (in der Regel ist dies das Verzeichnis C:\Programme\Graf-IT\VC_Driver) kopiert werden soll, dann aktivieren Sie das Kästchen hinter „In Treiberpfad kopieren“.

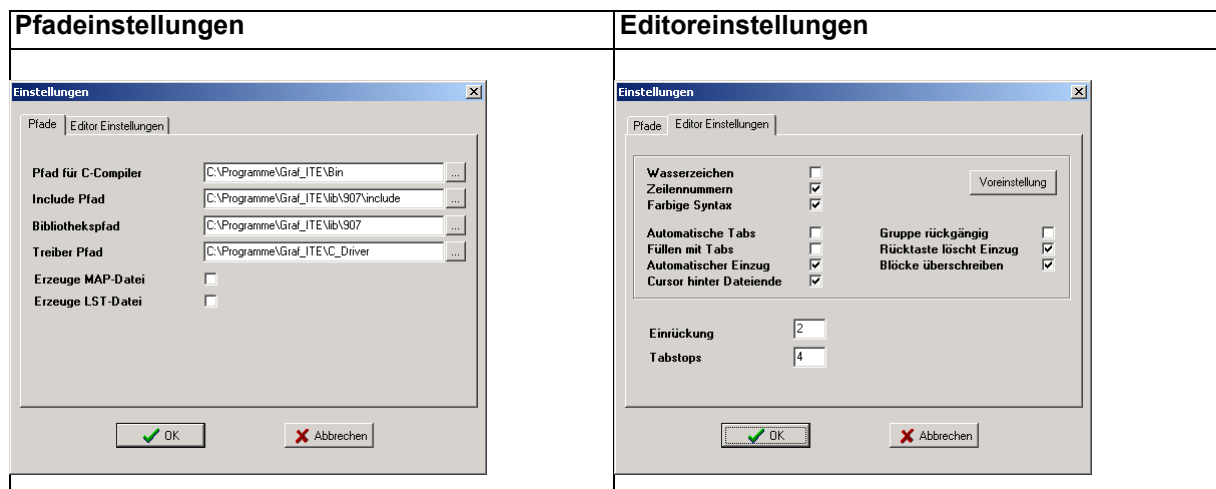
Ist „In Treiberpfad kopieren“ nicht aktiviert, dann wird der Treiber mit Drücken der Schaltfläche „Installieren“ in dem im Feld „Dateiname“ angegebenen Pfad installiert, ansonsten wird er zuerst in den Treiberpfad kopiert und dann dort installiert.

Treiber deinstallieren

Markieren Sie in der Liste den Treiber, den Sie deinstallieren wollen und drücken Sie die Schaltfläche „Deinstallieren“. Der Treiber wird aus der Liste entfernt, die zugehörige Datei wird aber nicht gelöscht. Bitte beachten Sie, daß die Deinstallation des Treibers das Risiko birgt, daß sich Projekte, die den Treiber benutzen nicht mehr Linken lassen !

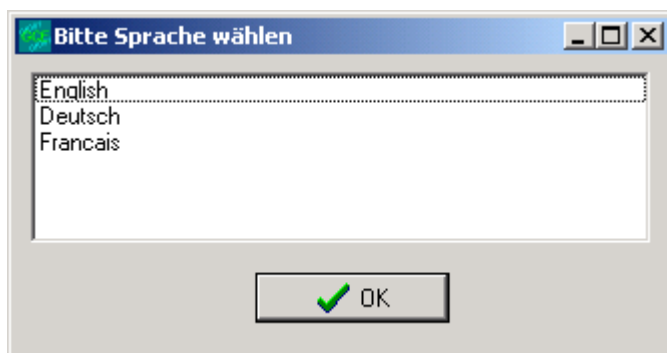
1.2.3.8 Voreinstellungen

In der Voreinstellungsmaske können Sie programmspezifische Einstellungen vornehmen. In der Regel muß hier nichts verändert werden, es sei denn Pfade würden sich verändern oder die Editor-Voreinstellungen gefallen Ihnen nicht.



1.2.3.9 Sprache

GCE ist multilingual ausgelegt. Sie können mit dem nachfolgend gezeigten Dialog die Spracheinstellung des Programms auf Ihre Sprache anpassen. Wählen Sie die gewünschte Sprache aus und drücken Sie die Schaltfläche „OK“. Das Programm schaltet sofort um.



Handbuch Bediengeräte

Hinweis:

Wenn Sie GCE aus dem Editor ITE heraus aufrufen, dann verwendet GCE die Spracheinstellung des Editors ITE. Die Änderung der Spracheinstellung dort wirkt sich unmittelbar auf alle Zusatzprogramme aus.

1.2.4 Das Menü Info

Im Info-Menü gibt es nur den Menüpunkt „Versionen“, der das Programm INFO.EXE aufruft. Dieses Zusatzprogramm zeigt Versions-Informationen über alle installierten Softwareteile an



1.3 Die Tool-Bar



Die Toolbar dient dem schnelleren Zugriff auf häufig verwendete Funktionen. Hierzu wurden für diese Funktionen die folgenden Schaltflächen eingefügt.



Programm beenden
Beendet das Programm



Datei Drucken
Druckt die Datei, die im aktuell aktiven Editorfenster geöffnet ist auf dem Drucker aus



Datei Neu
Erzeugt eine neue Quelldatei



Make
Übersetzt geänderte Quelldateien und linkt das Projekt



Datei Öffnen
Öffnet eine vorhandene Quelldatei



Build
Übersetzt alle zum Projekt gehörenden Quelldateien und linkt das Projekt.



Datei speichern
Speichert die Datei im aktuell aktiven Editorfenster

Handbuch Bediengeräte

2 Die Schnittstelle zum TOS

2.1 Systemdateien

Damit die Möglichkeit besteht, eigene, in C programmierte Steuerungsfunktionen in den Bediengeräten der AT-Serie auszuführen, ist im Betriebssystem TOS eine Schnittstelle zum Anwenderprogramm implementiert. Diese Schnittstelle besteht aus vier Funktionen, die das Betriebssystem des Bediengerätes intern aufruft. Damit diese Funktionen für den Anwender sichtbar werden, wird bei der Installation des Editorpaketes ITE im Pfad <InstPfad\bin> die Datei Template.c installiert, die die Schnittstellenfunktionen zum Betriebssystem enthält. Bei der Neuanlage eines Steuerungsprojektes in GCE wird diese Datei in das Projektverzeichnis kopiert und auf den Projektnamen umbenannt. Sie ist damit zentrale Schnittstellendatei zum Betriebssystem.

Weiterhin werden bei der Installation des Editorpaketes drei weitere wichtige Dateien in das Verzeichnis <InstPfad\C_Driver> kopiert:

2.1.1 KOP_IF.OBJ

Die Verbindung Ihres Steuerprogramms zu Funktionen im Betriebssystem des Bediengerätes wird durch die Datei KOP_IF.OBJ hergestellt. Diese Datei enthält den Code für alle im nächsten Kapitel beschriebenen Systemfunktionen und wird grundsätzlich mit Ihrem Projekt gebunden (gelinkt). KOP_IF.OBJ stellt somit das so genannte Application Interface (API) dar.

2.1.2 KOPTOTOS.H

Die Datei KOPTOTOS.H enthält die Prototypen (Deklarationen) der Systemfunktionen. Bitte schauen Sie im Zweifelsfall in dieser Datei nach, wenn Sie Informationen über Parameter und Rückgabewerte der Systemfunktionen benötigen.

2.1.3 CONV.H

Die Datei CONV.H enthält in der API verwendete Datentyp-Definitionen

2.2 Schnittstellenfunktionen

Damit benutzerspezifische Anwenderprogramme auf dem Bediengerät laufen, müssen im Anwenderprogramm vier Schnittstellenfunktionen zur Verfügung stehen, die je nach Betriebszustand des Bediengerätes aufgerufen werden. Die Namen dieser Schnittstellenfunktionen, deren Parameter und Rückgabewerte sind fest vorgegeben. Es müssen immer alle vier Funktionen im Anwenderprogramm vorhanden sein, auch wenn darin keine anwenderspezifischen Programmteile ausgeführt werden.

2.2.1 Initialisierungs-Funktion

Die Initialisierungsfunktion wird nach dem Start des TOS angerufen, bevor die zyklische bzw. die zeitgesteuerte Funktion aufgerufen wird. Damit können z.B. globale Variablen vorbesetzt, oder andere vorbereitende Aufgaben ausgeführt werden. Zu diesem Zeitpunkt ist der Zugriff auf Projektvariablen bereits möglich, so daß Sie diese auf Standardwerte initialisieren können. Auch der Zugriff auf alle Systemfunktionen ist hier bereits möglich.

Deklaration:	void KOP_Init (void)
Parameter:	keine
Rückgabewert	Der Rückgabeparameter ist vom Type integer und gibt ab, mit welchem Intervall die zeitgesteuerte Funktion aufgerufen werden soll. Die Intervallzeit ist in 10 ms - Schritten anzugeben. Wird die zeitgesteuerte Funktion nicht benötigt, muß 0 als Rückgabewert verwendet werden.

```
int KOP_Init(void)
{
    return(10); /* Zeitgesteuerte Bearbeitung alle 10*10 ms = 100 ms aufrufen */
}
```

Handbuch Bediengeräte

2.2.2 Zyklisch bearbeitetes Anwenderprogramm

Diese Funktion wird mit einem Intervall von ca. 200 ms aufgerufen. Die Intervallzeit ist nicht vorhersagbar und nicht konstant, es ist daher keine Echtzeitverarbeitung möglich.

Generell gilt, daß diese Funktion immer dann aufgerufen wird, wenn das Betriebssystem keine anderen Funktionen zu bearbeiten hat (Idle-Zustand). In diese Bearbeitung können alle komplizierten, zeitkritischen Funktionen eingebaut werden. Die Funktion muß so programmiert sein, daß sie schnellst möglich wieder verlassen wird. Es dürfen also keine sehr lang laufenden Schleifen programmiert werden, weil sonst das Betriebssystem ausgebremst wird !

Deklaration:	void KOP_Cycle (void)
Parameter:	keine
Rückgabewert	keiner

Beispiel:

```
void KOP_Cycle(void)
{
    ... /* Anwenderprogramm */
}
```

2.2.3 Zeitgesteuertes Anwenderprogramm

Diese Funktion wird immer dann verwendet, wenn die Bearbeitung des Anwenderprogramms in einem zeitlich konstanten Intervall sichergestellt sein muß.

Das Zeitintervall berechnet sich aus dem Rückgabewert der Initialisierungsfunktion KOP_Init() und ist dort in Schritten von 10 Millisekunden einstellbar.

Auch das zeitgesteuerte Anwenderprogramm muß schnellst möglich wieder verlassen werden, damit das Betriebssystem nicht ausgebremst wird. Es sollten nur die notwendigsten Funktionen ausgeführt werden, weil sonst die Performance des Geräts sinkt.

Die Bearbeitung darf nicht länger dauern, als die eingestellte Intervallzeit !.

Deklaration:	void KOP_Timer (void)
Parameter:	keine
Rückgabewert	keiner

Beispiel:

```
void KOP_Timer(void)
{
    ... /* zeitkritische Teile des Anwenderprogramms */
}
```

Handbuch Bediengeräte

2.2.4 CAN-Empfangs-Ereignis

Die Ereignisbehandlungsroutine für den Empfang von CAN-Telegrammen wird immer dann aufgerufen, wenn ein CAN Telegramm auf einer ID der Multimasterkanäle, oder der dynamisch freigeschalteten Identifier, die das Bediengerät belegt, empfangen wird.

Die dynamische Freischaltung von Identifier erfolgt in der Initialisierungsfunktion KOP_Init() durch Aufruf einer der Systemfunktionen CANEnableRxId oder CANEnableRxMask.

Auch diese Ereignisbehandlungsroutine muß schnellst möglich wieder verlassen werden, um die Performance nicht zu beeinträchtigen.

Deklaration:	void KOP_CAN_Event (...)
Parameter:	uint32 RxId Empfangs-Identifier, Identifierformat und Kanal int len Anzahl empfangener Datenbytes (0..8) uint8 D0 Datenbyte D0 uint8 D1 Datenbyte D1 uint8 D2 Datenbyte D2 uint8 D3 Datenbyte D3 uint8 D4 Datenbyte D4 uint8 D5 Datenbyte D5 uint8 D6 Datenbyte D6 uint8 D7 Datenbyte D7
Rückgabewert	keiner

Nach dem Empfang des Telegramms und dadurch bedingten Aufruf dieser Funktion stehen die über den CAN-Bus übertragenen acht Datenbytes in den Parametern D0 bis D7 für die Auswertung durch das Steuerungsprogramm zur Verfügung. Die empfangenen Daten sollten, wenn die Bearbeitung längere Zeit in Anspruch nimmt in einen Zwischenpuffer kopiert werden, der dann von der Funktion KOP_Cycle() ausgewertet wird.

Hinweis:

Im Parameter RxId ist codiert, ob es sich um einen 29-bit ID handelt und von welcher Schnittstelle das Telegramm empfangen wurde.

Handbuch Bediengeräte

Beispiel:

```
void KOP_CAN_Event(uint32 RxId, int len, uint8 D0, uint8 D1, uint8 D2, uint8 D3,  
                  uint8 D4, uint8 D5, uint8 D6, uint8 D7)
```

```
{  
  /* Abfragen auf 29 bit ID */  
  if (RxId & USE_29BIT_ID)  
  {  
    ... /* Behandlung von Telegrammen mit 29-Bit-Identifizier */  
  }  
  
  /* Abfragen, von welcher Schnittstelle das Telegramm kommt */  
  if (RxId & USE_CAN_1)  
  {  
    ... /* Telegramm kommt von 2. CAN-Schnittstelle */  
  }  
  else  
  {  
    ... /* Telegramm kommt von der Standard-CAN-Schnittstelle*/  
  }  
}
```

Um jetzt den Identifizier zu decodieren, der von Außen angesprochen wurde, müssen die Statusbits (Schnittstelle und 29-BitID-Flag) ausgeblendet werden. Erst danach kann der Identifizier ausgewertet werden.

```
/* ausblenden Statusbits */  
RxId = RxId & 0x1FFFFFFFUL;  
if (RxId == 1024)  
{  
  ... /* Auswertung für Identifizier 1024 */  
}  
}
```

Handbuch Bediengeräte

3 Die Betriebssystem-API Referenz

3.1 Übersicht

Die Funktionen der Betriebssystem-API sind in diesem Handbuch thematisch nach Anwendungsart sortiert. Zu jedem Thema gibt es eine Reihe von Funktionen. Die folgende Tabelle soll einen Überblick über die Systemfunktionen geben. In der Onlineversion dieser Dokumentation können Sie die Beschreibung jeder Funktion mit einem Mausklick auf den Funktionsnamen in der Spalte „Funktionsname“ erreichen. Die Spalte ganz rechts gibt die TOS-Version an, ab der die jeweilige Funktion im TOS implementiert ist. Grundsätzlich sind alle Funktionen im TOS vorhanden und können aufgerufen werden. Funktionen, die noch nicht implementiert sind, sind als leere Schablonen in das TOS eingebaut und in der letzten Tabellenspalte mit N/A gekennzeichnet..

Aufgabe	Funktionsname	TOS - Version
3.2.1 Funktionen für das Arbeiten mit Variablen		
3.2.1.1 Variablenwert lesen	GetVar	
3.2.1.2 Variablenwert setzen	SetVar	
3.2.1.7 Läuft eine Variable als Betriebsstundenzähler ?	IsHourCounterOn	
3.2.1.8 Variable als Betriebsstundenzähler starten	HourCounterStart	
3.2.1.9 Variable als Betriebsstundenzähler stoppen	HourCounterStop	
3.2.1.10 Projekt-Variablen an eine SPS senden	SendVarToPLC	
3.2.1.11 Projekt-Variable an anderes Bediengerät senden	SendVarToTerminal	
3.2.1.3 Skalierten Wert einer J1939 Variable lesen	GetVarJ1939Scaled	
3.2.1.4 Wert einer skalierten J1939 Variablen setzen	SetVarJ1939Scaled	
3.2.1.5 Unskalierten Wert einer J1939 Variable lesen	GetVarJ1939binary	
3.2.1.6 Wert einer unskalierten J1939 Variablen setzen	SetVarJ1939binary	
3.2.2 Funktionen für das Arbeiten mit Bildern		
3.2.2.1 Rollierzeit für Bilder lesen	GetPageAutoScrollTime	
3.2.2.2 Rollierzeit für Bilder setzen	SetPageAutoScrollTime	
3.2.2.3 Nummer des gerade angezeigten Bildes lesen	GetCurrentPageShown	
3.2.2.4 Feststellen, ob ein Bild aktiv ist	IsPageOn	
3.2.2.5 Feststellen, ob ein Prioritätsbild aktiv ist	IsPrioPageOn	
3.2.2.6 Aktivieren eines Bildes	PageOn	
3.2.2.7 Deaktivieren eines Bildes	PageOff	
3.2.2.8 Aktivieren eines Prioritäts-Bildes	PrioPageOn	
3.2.2.9 Deaktivieren eines Prioritäts-Bildes	PrioPageOff	
3.2.2.10 Aktivieren eines Menübildes	MenuPageOn	
3.2.2.11 Deaktivieren der letzten Menübilder/struktur	MenuPagesOff	N/A
3.2.3 Funktionen für das Arbeiten mit Meldungen		
3.2.3.1 Rollierzeit für Meldungen lesen	GetMessageAutoScrollTime	
3.2.3.2 Rollierzeit für Meldungen setzen	SetMessageAutoScrollTime	
3.2.3.3 Nummer der gerade angezeigten Meldung lesen	GetCurrentMessageShown	

Handbuch Bediengeräte

3.2.3.4 Feststellen, ob eine Meldung aktiv ist	IsMessageOn	
3.2.3.5 Aktivieren einer Meldung	MessageOn	
3.2.3.6 Deaktivieren einer Meldung	MessageOff	
3.2.4 Funktionen für das Arbeiten mit Tasten		
3.2.4.1 Feststellen, ob irgendeine Taste gedrückt ist	IsAnyKeyDown	
3.2.4.2 Feststellen, ob eine bestimmte Taste gedrückt ist	IsKeyDown	
3.2.4.3 Feststellen, ob eine bestimmte Taste gedrückt wurde	IsKeyPressedNew	
3.2.4.4 Menütastendruck simulieren	SimulateMenuKey	
3.2.4.5 ASCII-Eingabe simulieren	SimulateASCIIKey	
3.2.4.6 Cursor auf Eingabefeld/Menüpunkt positionieren	SetCursorPos	N/A
3.2.4.7 Cursorposition abfragen	GetCursorPos	N/A
3.2.4.8 Menüpunktindex lesen	GetCurrentMenuIndex	
3.2.4.9 Maske für die Softkey-Tasten lesen	GetSoftkeyMask	
3.2.4.10 Maske für die Softkey-Tasten setzen	SetSoftkeyMask	
3.2.4.11 Anzahl Tasten-Erweiterungsblöcke lesen	GetKeyBlockCount	
3.2.5 Funktionen für das Arbeiten mit LEDs		
3.2.5.1 Einschalten einer LED	LedOn	
3.2.5.2 Ausschalten einer LED	LedOff	
3.2.5.3 Feststellen, ob eine LED eingeschaltet ist	IsLedOn	
3.2.6 Funktionen für das Arbeiten mit Timern		
3.2.6.1 Starten/Setzen eines Timers	SetTimer	
3.2.6.2 Abfragen, ob ein Timer läuft	IsTimerOn	
3.2.6.3 Abfragen, ob ein Timer steht	IsTimerOff	
3.2.7 Funktionen für das Arbeiten mit Ein/Ausgängen		
3.2.7.1 Internen Meldeausgang einschalten	OutputOn	
3.2.7.2 Internen Meldeausgang ausschalten	OutputOff	
3.2.7.3 Meldeausgang abfragen	IsOutputOn	
3.2.7.4 Light-In Eingang abfragen	IsInputOn	
3.2.7.5 Internen digitalen Eingang abfragen	GetInternalDI	
3.2.7.6 Internen digitalen Ausgang abfragen	GetInternalDO	
3.2.7.7 Internen digitalen Ausgang schalten	SetInternalDO	
3.2.7.8 Internen analogen Eingang lesen	GetInternalAI	
3.2.7.9 Eingangsbyte 0 lesen	GetDIB0	
3.2.7.10 Eingangsbyte 1 lesen	GetDIB1	
3.2.7.11 Eingangswort 0 lesen	GetDIW0	
3.2.7.12 Ausgangsbyte 0 lesen	GetDOB0	
3.2.7.13 Ausgangsbyte 0 schreiben	SetDOB0	
3.2.7.14 Ausgangsbyte 1 lesen	GetDOB1	

Handbuch Bediengeräte

3.2.7.15 Ausgangsbyte 1 schreiben	SetDOB1	
3.2.7.16 Ausgangswort 0 lesen	GetDOW0	
3.2.7.17 Ausgangswort 0 schreiben	SetDOW0	
3.2.7.18 Kurzschlußstatus lesen	GetShortcutState	
3.2.8 Schnelle 2-kanalige Zähler		
3.2.8.1 Zählerwert 1 lesen	GetCount1	N/A
3.2.8.2 Zählerwert 1 schreiben	SetCount1	N/A
3.2.8.3 Zählerwert 2 lesen	GetCount2	N/A
3.2.8.4 Zählerwert 2 schreiben	SetCount2	N/A
3.2.8.5 Vorwahl 1 für Zähler 1 lesen	Get_C1_Preset1	N/A
3.2.8.6 Vorwahl 1 für Zähler 1 schreiben	Set_C1_Preset1	N/A
3.2.8.7 Vorwahl 2 für Zähler 1 lesen	Get_C1_Preset2	N/A
3.2.8.8 Vorwahl 2 für Zähler 1 schreiben	Set_C1_Preset2	N/A
3.2.8.9 Vorwahl 1 für Zähler 2 lesen	Get_C2_Preset1	N/A
3.2.8.10 Vorwahl 1 für Zähler 2 schreiben	Set_C2_Preset1	N/A
3.2.8.11 Vorwahl 2 für Zähler 2 lesen	Get_C2_Preset2	N/A
3.2.8.12 Vorwahl 2 für Zähler 2 schreiben	Set_C2_Preset2	N/A
3.2.9 Zugriff auf die CAN-Schnittstellen		
3.2.9.2 CAN-Busrate einstellen	SetCAN0BusrateIndex SetCAN1BusrateIndex	
3.2.9.3 CAN-Busrate lesen	GetCAN0BusrateIndex GetCAN1BusrateIndex	
3.2.9.4 Standard Sende-Identifizier lesen	GetCANTxId	
3.2.9.5 Standard Sende-Identifizier schreiben	SetCANTxId	
3.2.9.6 Standard Empfangs-Identifizier lesen	GetCANRxId0	
3.2.9.7 Standard Empfangs-Identifizier schreiben	SetCANRxId0	
3.2.9.8 Multimaster-Empfangskanäle lesen	GetCANRxId1 GetCANRxId2 GetCANRxId3 GetCANRxId4 GetCANRxId5 GetCANRxId6 GetCANRxId7	
3.2.9.9 Multimaster-Empfangskanäle schreiben	SetCANRxId1 SetCANRxId2 SetCANRxId3 SetCANRxId4 SetCANRxId5 SetCANRxId6 SetCANRxId7	
3.2.9.10 Neuinitialisierung der CAN-Schnittstellen	RestartCanSystem	
3.2.9.11 Status des CAN-Bus prüfen	Get_CAN_Status	N/A
3.2.9.12 Senden eines CAN-Telegramms	SendCANTelegram	

Handbuch Bediengeräte

3.2.9.20 Empfangsidentifizier dynamisch freischalten	CANEnableRxId	
3.2.9.21 Dynamisch zugewiesene Empfangsidentifizier sperren	CANDisableRxId	
3.2.9.22 Freigabemaske für Empfangsidentifizier setzen	CANEnableRxMask	
3.2.9.23 Freigabemaske für Empfangsidentifizier entfernen	CANDisableRxMask	
3.2.9.24 Feststellen, ob das Gerät als CAN-Master läuft	GetMasterFlag	
3.2.9.25 CAN-open Übertragung überwachen	SDOStatus	N/A
3.2.9.26 SELECAN Gerätenummer lesen	GetSELECANNodeNo	
3.2.9.27 SELECAN Gerätenummer schreiben	SetSELECANNodeNo	
3.2.10 Funktionen zur Ansteuerung von GCM-Can-Modulen		
3.2.10.1 Externen digitalen Eingang abfragen	GetGCM_DI	
3.2.10.2 Externen analogen Eingang abfragen	GetGCM_AI	
3.2.10.3 Externen digitalen Ausgang setzen	SetGCM_DO	
3.2.10.4 Externen analogen Ausgang setzen	SetGCM_AO	
3.2.10.5 Modul-Status abfragen	GetGCM_Status	N/A
3.2.11 Zugriff auf die seriellen Schnittstellen		
3.2.11.2 Baudrate lesen	GetSerial0BaudIndex GetSerial1BaudIndex	
3.2.11.3 Baudrate einstellen	SetSerial0BaudIndex SetSerial1BaudIndex	
3.2.11.4 Telegrammeinstellung lesen	GetSerial0FrameSetting GetSerial1FrameSetting	
3.2.11.5 Telegrammeinstellung setzen	SetSerial0FrameSetting SetSerial1FrameSetting	
3.2.11.6 Neuinitialisierung der seriellen Schnittstellen	RestartComSystem	
3.2.11.8 Statusabfrage	GetStatSerial0 GetStatSerial1	
3.2.11.9 Empfangenes Zeichen lesen	GetCharSerial0 GetCharSerial1	
3.2.11.10 Empfangenes Zeichen quittieren	SetStatSerial0 SetStatSerial1	
3.2.11.11 Empfangenes Zeichen überschreiben	SetCharSerial0 SetCharSerial1	
3.2.11.12 Seriell senden	SendToSerial	
3.2.12 Funktionen zur Einstellung des Displays		
3.2.12.1 Kontrast lesen	GetContrastIndex	
3.2.12.2 Kontrast setzen	SetContrastIndex	
3.2.12.3 Helligkeit lesen	GetBrightnessIndex	
3.2.12.4 Helligkeit setzen	SetBrightnessIndex	
3.2.12.5 Helligkeitswert für Tastenbeleuchtung lesen	GetELFOILState	
3.2.12.6 Helligkeitswert für Tastenbeleuchtung setzen	SetELFOILState	

Handbuch Bediengeräte

3.2.13 Grafische Funktionen		
3.2.13.4 Einzelnen Bildpunkt (Pixel) zeichnen	DrwPixel	
3.2.13.5 Rechteck ohne Füllung zeichnen	DrwRect	
3.2.13.6 Rechteck mit Füllung zeichnen	DrwRectFilled	
3.2.13.7 Eine Linie zeichnen	DrwLine	
3.2.13.8 Kreis zeichnen	DrwCircle	
3.2.13.9 Gefüllten Kreis zeichnen	DrwCircleFilled	
3.2.13.10 Gefüllte Ellipse zeichnen	DrwEllipseFilled	
3.2.13.11 Text ausgeben	DrwText	
3.2.13.12 Bereich im Display verschieben	MoveArea	
3.2.13.13 Display Hintergrundfarbe einstellen	SetDisplayBackgroundColor	
3.2.13.14 Zeigerinstrument initialisieren	DefineAnalogView	
3.2.13.15 Zeigerinstrument anzeigen / aktualisieren	ShowAnalogView	
3.2.13.16 Zeigerinstrument löschen	DeleteAnalogView	
3.2.13.17 Zeigerinstrument löschen	DrwSetSpecialScreenMode	
3.2.14 Funktionen für den Touch-Screen		
3.2.14.1 X-Position der aktuellen Druckstelle lesen	Get_X_Move	
3.2.14.2 Y-Position der aktuellen Druckstelle lesen	Get_Y_Move	
3.2.14.3 X-Position der ersten Druckstelle lesen	Get_X_Down	
3.2.14.4 Y-Position der ersten Druckstelle lesen	Get_Y_Down	
3.2.14.5 X-Position absoluter Analogwert lesen	Get_X_Value	N/A
3.2.14.6 Y-Position absoluter Analogwert lesen	Get_Y_Value	N/A
3.2.14.7 Kalibrierungswert für X lesen	Get_X_Cali	N/A
3.2.14.8 Kalibrierungswert für Y lesen	Get_Y_Cali	N/A
3.2.14.9 Toleranzwert lesen	GetTouchTol	
3.2.14.10 Toleranzwert setzen	SetTouchTol	
3.2.15 Funktionen für Flankenbewertung		
3.2.15.1 Auswertung beider Flanken	AnyEdge	
3.2.15.2 Auswertung der steigenden Flanke	RisingEdge	
3.2.15.3 Auswertung der fallenden Flanke	FallingEdge	
3.2.16 Funktionen für den Video-Eingang		
3.2.16.1 Videobild definieren	CameraWindowSet	
3.2.16.2 Videobild einschalten	CameraWindowOn	
3.2.16.3 Videobild ausschalten	CameraWindowOff	
3.2.17 Sonstige Funktionen		
3.2.17.1 Aktuellen Gerätestatus abfragen	GetCurrentDeviceStatus	
3.2.17.2 Fehlerstatus abfragen	Get_ERROR_Status	
3.2.17.3 Zeitzone abfragen	GetCurrentTimezone	

Handbuch Bediengeräte

3.2.17.4 Zeitzone setzen	SetCurrentTimezone	
3.2.17.5 Konfiguration der Druckerschnittstelle lesen	GetPrinterInterfaceIndex	
3.2.17.6 Druckerschnittstelle konfigurieren	SetPrinterInterfaceIndex	
3.2.17.7 Feststellen, welcher SPS-Treiber geladen ist	GetPLCDriverNo	
3.2.17.8 Lesen der BIOS-Version	GetBIOSVersion	
3.2.17.9 Lesen der TOS-Version	GetTOSVersion	
3.2.17.10 Lesen der Projekt-Version	GetUSERVersion	
3.2.17.11 Initialisierungsflags lesen	GetInitFlags	
3.2.17.12 Initialisierungsflags schreiben	SetInitFlags	
3.2.17.13 Setup-Daten in das Projekt-FLASH speichern	SaveToFlash	
3.2.17.14 Gerätefunktion über CAN-Aufrufkonvention aufrufen	ExecCANTelegramInternal	
3.2.17.15 Buzzer Ein/Ausschalten	SetBuzzer	
3.2.17.16 Buzzer Status abfragen	GetBuzzer	
3.2.18 Flash-Funktionen		
3.2.18.1 Testen ob Daten ins Flash gespeichert werden können.	FlashTest	
3.2.18.2 Daten aus dem Anwender-Flash lesen	FlashRead	
3.2.18.3 Daten in das Anwender-Flash schreiben	FlashWrite	

Handbuch Bediengeräte

3.2 Beschreibung der Systemfunktionen

3.2.1 Funktionen für das Arbeiten mit Variablen

3.2.1.1 Variablenwert lesen

Mit der Funktion GetVar kann eine Variable gelesen werden, die im ITE-Projekt angelegt wurde.

Deklaration:	long GetVar (uint16 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	long Wert der Variablen (32-Bit signed integer)

Beispiel:

```
VarWert = GetVar(125);
```

3.2.1.2 Variablenwert setzen

Mit der Funktion SetVar kann der Wert einer Variablen, die im ITE-Projekt angelegt wurde, gesetzt werden.

Deklaration:	void SetVar (uint16 __Handle, long __Value)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen long __Value Wert auf den die Variable gesetzt werden soll
Rückgabewert	keiner

Beispiel:

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

3.2.1.3 Skalierten Wert einer J1939 Variable lesen

Mit der Funktion GetVarJ1939Scaled kann der fertig skalierte Wert einer J1939-Variablen gelesen werden, die im ITE-Projekt angelegt wurde. Näheres zu diesem Variablentyp finden Sie im Handbuch J1939 Kommunikation.

Deklaration:	long GetVarJ1939Scaled (uint32 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	long Wert der Variablen (32-Bit signed integer)

Beispiel:

```
Wert = GetVarJ1939Scaled(Var_0_EEEF_3);
```

3.2.1.4 Wert einer skalierten J1939 Variablen setzen

Mit der Funktion SetVarJ1939Scaled kann eine J1939-Variable auf einen skalierten Wert gesetzt werden.

Deklaration:	void SetVarJ1939Scaled (uint32 __Handle, long __Value)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen long __Value Wert auf den die Variable gesetzt werden soll
Rückgabewert	keiner

Beispiel:

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

Handbuch Bediengeräte

3.2.1.5 Unskalierten Wert einer J1939 Variable lesen

Mit der Funktion `GetVarJ1939Scaled` kann der unskalierte Wert einer J1939-Variablen gelesen werden, die im ITE-Projekt angelegt wurde. Näheres zu diesem Variablentyp finden Sie im Handbuch J1939 Kommunikation.

Deklaration:	long GetVarJ1939binary (uint32 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	long Wert der Variablen (32-Bit signed integer)

Beispiel:

```
Wert = GetVarJ1939binary(Var_0_EEEF_3);
```

3.2.1.6 Wert einer unskalierten J1939 Variablen setzen

Mit der Funktion `SetVarJ1939Scaled` kann eine J1939-Variable auf einen unskalierten Wert gesetzt werden.

Deklaration:	void SetVarJ1939binary (uint32 __Handle , long __Value)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen long __Value Wert auf den die Variable gesetzt werden soll
Rückgabewert	keiner

Beispiel:

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

3.2.1.7 Lläuft eine Variable als Betriebsstundenzähler ?

Es ist möglich, eine Variable im Sekundentakt aufwärts zählen zu lassen. Mit dieser Funktion kann abgefragt werden, ob die Funktion für eine bestimmte Variable aktiviert ist.

Deklaration:	int IsHourCounterOn (uint16 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	int == 0 wenn die Variable nicht aufwärts zählt <> 0 wenn die Variable aufwärts zählt

Beispiel:

```
if (IsHourCounterOn(125))  
{  
... /* Anwenderprogramm falls die Variable aufwärts zählt */  
}
```

Handbuch Bediengeräte

3.2.1.8 Variable als Betriebsstundenzähler starten

Mit dieser Funktion können interne ITE-Projekt-Variablen als Zeitmesser im Sekundentakt verwendet werden. Wird diese Funktion aufgerufen, so zählt das Betriebssystem die angegebene Variable einmal pro Sekunde um eins hoch, bis die Funktion HourCounterStop für diese Variable aufgerufen wird. Wenn das Gerät abgeschaltet wird, bleiben diese Betriebsstundenzähler natürlich stehen. Es können mehrere Variablen gleichzeitig als Betriebsstundenzähler aktiv sein.

Deklaration:	void HourCounterStart (uint16 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	keiner

Beispiel:

```
HourCounterStart(122);
```

3.2.1.9 Variable als Betriebsstundenzähler stoppen

Mit dieser Funktion kann das Aufwärtszählen einer Variablen, die über HourCounterStart(...) als Betriebsstundenzähler gestartet wurde, wieder gestoppt werden.

Deklaration:	void HourCounterStop (uint16 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	keiner

Beispiel:

```
HourCounterStop(122);
```

3.2.1.10 Projekt-Variablen an eine SPS senden

Mit dieser Funktion kann der Wert einer Projektvariablen an eine angeschlossene SPS übermittelt werden. Dabei wird dieselbe Funktion ausgelöst wie beim Eingeben eines Sollwerts. Das Telegramm "REPORT VALUE" wird über die Standard-CAN-Schnittstelle abgesetzt.

Deklaration:	void SendVarToPLC (uint16 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	keiner

Beispiel:

```
SendVarToPLC(100);
```

3.2.1.11 Projekt-Variable an anderes Bediengerät senden

Bediengeräte können über den CAN-Bus zusammengeschaltet werden. Diese Funktion wird dazu benutzt, um Variablen auszutauschen. Auf dem CAN-Bus wird das Telegramm "SET VALUE" gesendet. Informationen über den Aufbau des Telegramms finden Sie im Handbuch Kommunikation.

Deklaration:	void SendVarToTerminal (uint16 __Handle)
Parameter:	uint16 __Handle Nummer (Handle) der Variablen
Rückgabewert	keiner

Beispiel:

```
SendVarToTerminal(10);
```

Handbuch Bediengeräte

3.2.2 Funktionen für das Arbeiten mit Bildern

3.2.2.1 Rollierzeit für Bilder lesen

Mit dieser Funktion kann die Rollierzeit von Bildern in Sekunden gelesen werden. Die Rollierzeit ist die Zeit, die abläuft, bis das nächste Bild im Bilderstapel angezeigt wird

Deklaration:	int GetPageAutoScrollTime (void)
Parameter:	keine
Rückgabewert	int Zeit in Sekunden

Beispiel:

```
iBildRollzeit = GetPageAutoScrollTime ();
```

3.2.2.2 Rollierzeit für Bilder setzen

Mit dieser Funktion kann die Rollierzeit von Bildern in Sekunden gesetzt werden. Die Rollierzeit ist die Zeit, die abläuft, bis das nächste Bild im Bilderstapel angezeigt wird

Deklaration:	void SetPageAutoScrollTime (int __Secs)
Parameter:	int __Secs Zeit in Sekunden (0=Rollieren aus)
Rückgabewert	int Zeit in Sekunden

Beispiel:

```
SetPageAutoScrollTime (2); /* rollieren alle 2 Sekunden */
```

Übernahme:

sofort. Nach Neustart aber nur, wenn SaveToFlash ausgeführt wurde.

3.2.2.3 Nummer des gerade angezeigten Bildes lesen

Die Funktion gibt Ihnen als integer die Nummer des Bildes zurück, das aktuell im Display angezeigt wird.

Deklaration:	uint16 GetCurrentPageShown (void)
Parameter:	keine
Rückgabewert	int Nummer des aktuell angezeigten Bildes

Beispiel:

```
iBildNummer = GetCurrentPageShown();
```

Handbuch Bediengeräte

3.2.2.4 Feststellen, ob ein Bild aktiv ist

Ein Bild ist aktiv, wenn es über CAN oder das Steuerprogramm aktiviert wurde. Es bedeutet nicht unbedingt, daß es im Display zu sehen ist.

Deklaration:	int IsPageOn (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes
Rückgabewert	int == 0: Bild ist nicht aktiv <> 0: Bild ist aktiv

Beispiel:

```
if (IsPageOn(25))
{
... /* Anwenderprogramm, wenn das Bild aktiv ist */
}
```

3.2.2.5 Feststellen, ob ein Prioritätsbild aktiv ist

Ein Prioritätsbild ist aktiv, wenn es über CAN oder das Programm aktiviert wurde. Es bedeutet nicht unbedingt, daß es im Display zu sehen ist.

Deklaration:	int IsPrioPageOn (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes
Rückgabewert	int == 0: Bild ist nicht aktiv <> 0: Bild ist aktiv

Beispiel:

```
if (IsPrioPageOn(25))
{
... /* Anwenderprogramm, wenn das Prioritätsbild aktiv ist */
}
```

3.2.2.6 Aktivieren eines Bildes

Über die Funktion "PageOn" kann ein Bild des Projekts aktiviert werden.

Deklaration:	void PageOn (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes, das aktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
PageOn(4); /* aktiviert Bild 4 */
```

3.2.2.7 Deaktivieren eines Bildes

Über die Funktion "PageOff" kann ein Bild des Projekts deaktiviert werden.

Deklaration:	void PageOff (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes, das deaktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
PageOff(4); /* deaktiviert Bild 4 */
```

Handbuch Bediengeräte

3.2.2.8 Aktivieren eines Prioritäts-Bildes

Über die Funktion "PrioPageOn" kann ein Bild des Projekts aktiviert werden, und zwar als Prioritätsbild. Im Unterschied zu einem "normalen" Bild wird dieses Bild auf dem Bilderstapel ganz oben abgelegt und kommt so immer sofort zur Anzeige.

Deklaration:	void PrioPageOn (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes, das aktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
PrioPageOn(5); /* aktiviert Bild 5 und zeigt das Bild sofort an */
```

3.2.2.9 Deaktivieren eines Prioritäts-Bildes

Mit dieser Funktion wird ein Prioritäts-Bild wieder deaktiviert.

Deklaration:	void PrioPageOff (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes, das deaktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
PrioPageOff(5); /* deaktiviert PrioBild 5 */
```

3.2.2.10 Aktivieren eines Menübildes

Beim Aktivieren eines Menübildes wird ein Bild aufgerufen und sofort in die Eingabe/Menüstruktur geschaltet, sofern ein aktivierbarer Menüpunkt und/oder Sollwert im Bild vorhanden ist.

Deklaration:	void MenuPageOn (int __PageNo)
Parameter:	int __PageNo Nummer des Bildes, das aktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
MenuPageOn(17); /* aktiviert Bild 17 als Menübild */
```

3.2.2.11 Deaktivieren der letzten Menübilder/struktur

Mit der hier beschriebenen Funktion können die letzten Stufen der Menüeingabe wieder deaktiviert werden. Dabei wird nicht auf die Bildnummern verwiesen, sondern es werden die letzten (Count) Menübilder deaktiviert.

Deklaration:	void MenuPagesOff (int __Count)
Parameter:	int __Count Anzahl der zu deaktivierenden Bilder
Rückgabewert	keiner

Beispiel:

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

Handbuch Bediengeräte

3.2.3 Funktionen für das Arbeiten mit Meldungen

3.2.3.1 Rollierzeit für Meldungen lesen

Mit dieser Funktion kann die Rollierzeit von Meldungen in Sekunden gelesen werden:

Deklaration:	int GetMessageAutoScrollTime (void)
Parameter:	keine
Rückgabewert	int Zeit in Sekunden

Beispiel:

```
iRollierzeit = GetMessageAutoScrollTime();
```

3.2.3.2 Rollierzeit für Meldungen setzen

Mit dieser Funktion kann die Rollierzeit von Meldungen in Sekunden geschrieben werden

Deklaration:	void SetMessageAutoScrollTime (int __Secs)
Parameter:	int __Secs Zeit in Sekunden (0=Rollieren aus)
Rückgabewert	int Zeit in Sekunden

Beispiel:

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

Übernahme:

sofort. Nach Neustart aber nur, wenn SaveToFlash ausgeführt wurde.

3.2.3.3 Nummer der gerade angezeigten Meldung lesen

Die Funktion `GetCurrentMessageShown()` gibt Ihnen als integer die Nummer der Meldung zurück, die aktuell im Display angezeigt wird. Ist keine Meldung in der Anzeige, wird 0 zurückgegeben.

Deklaration:	uint16 GetCurrentMessageShown (void)
Parameter:	keine
Rückgabewert	int Nummer der aktuell angezeigten Meldung

Beispiel:

```
iMeldeNummer = GetCurrentMessageShown();
```

3.2.3.4 Feststellen, ob eine Meldung aktiv ist

Eine Meldung ist aktiv, wenn sie über CAN oder das Programm aktiviert wurde. Es bedeutet nicht unbedingt, daß die Meldung auch im Display zu sehen ist.

Deklaration:	int IsMessageOn (int __MsgNo)
Parameter:	int __MsgNo Nummer der Meldung
Rückgabewert	int == 0: Bild ist nicht aktiv <> 0: Bild ist aktiv

Beispiel:

```
if (IsMessageOn(10))  
{  
... /* Anwenderprogramm falls die Meldung aktiv ist */  
}
```

Handbuch Bediengeräte

3.2.3.5 Aktivieren einer Meldung

Über die Funktion "MessageOn" kann eine Meldung des Projekts aktiviert werden.

Deklaration:	void MessageOn (int __MsgNo)
Parameter:	int __MsgNo Nummer der Meldung, die aktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
MessageOn(10);
```

3.2.3.6 Deaktivieren einer Meldung

Über die Funktion "MessageOff" kann eine Meldung des Projekts deaktiviert werden.

Deklaration:	void MessageOff (int __MsgNo)
Parameter:	int __MsgNo Nummer der Meldung, die deaktiviert werden soll
Rückgabewert	keiner

Beispiel:

```
MessageOff(10);
```

3.2.4 Funktionen für das Arbeiten mit Tasten

3.2.4.1 Feststellen, ob irgendeine Taste gedrückt ist

Mit dieser Funktion kann abgefragt werden, ob irgendeine oder gar keine Taste (Folientaste oder Inkremental-Poti) gedrückt ist.

Deklaration:	int IsAnyKeyDown (void)
Parameter:	keine
Rückgabewert	int == 0: keine Taste gedrückt <> 0: Tastennummer der gedrückten Taste

Beispiel:

```
switch(IsAnyKeyDown())
{
  case 0x00: /* keine Taste gedrückt */
    break;
  case 0x01: /* Taste 1 gedrückt */
    ... /* Anwenderprogramm wenn Taste 1 gedrückt */
    break;
  case 0x02: /* Taste 2 gedrückt */
    ... /* Anwenderprogramm wenn Taste 1 gedrückt */
    break;
  default:
    ... /* Anwenderprogramm wenn Taste 1 gedrückt */
    break;
}
```


Handbuch Bediengeräte

3.2.4.5 ASCII-Eingabe simulieren

Eine alphanumerische Taste lässt sich mit dieser Funktion so simulieren, als ob sie an einer extern angeschlossenen PS/2-Tastatur tatsächlich gedrückt wurde. Es wird automatisch "Taste gedrückt" und anschließend "Taste losgelassen" generiert.

Deklaration:	<code>void SimulateASCIIKey(int __KeyCode, int __ExtCode)</code>
Parameter:	<code>int __KeyCode</code> ASCII-Code der Taste <code>int __ExtCode</code> zur Zeit nicht benutzt, bitte 0 angeben
Rückgabewert	void

Beispiel:

```
SimulateASCIIKey(65,0); /* Simuliert 'A' */
```

3.2.4.6 Cursor auf Eingabefeld/Menüpunkt positionieren

Wenn ein Menübild aktiv ist, kann man über diese Funktion den Cursor auf ein Feld positionieren. Das Feld kann über X/Y-Position oder Funktion ausgewählt werden.

Die Positionierung erfolgt, wenn das Gerät im Ruhezustand(0), Menüauswahl(4) oder Sollwertauswahl(5) ist und ein Sollwert und/oder Menüpunkt im Bild vorhanden ist.

Deklaration:	<code>void SetCursorPos(int __Mode, int __X, int __Y, int __MPFunc)</code>
Parameter:	<code>int __Mode</code> 0,1Versuche auf Zeile __Y Spalte __X zu positionieren 2 Setze auf Sollwert mit Variablen Handle __X 3 Setze auf Menüpunkt mit Fkt. MPFunc und Parameter __X (Bildnr oder Menüindex) <code>int __X</code> Spalte, Variablenhandle oder Parameter (Bildnr oder Menüindex) <code>int __Y</code> Zeile <code>int __MPFunc</code> 0 Bild aufrufen (__X=Bildnummer) 1 Menü zurückgehen (__X ohne Bedeutung) 2 Sollwert-Eingabe beenden mit speichern (__X = Bildnummer) 3 Sollwert-Eingabe beenden ohne speichern (__X = Bildnummer) 4 Globaler Abbruch (__X ohne Bedeutung) 5 Menü-Indexausgabe (__X = Menüindex) 6 Indexausgabe mit globalem Abbruch (__X = Menüindex) 7 Menü-Index an Steuerprogramm geben (__X = Menüindex)
Rückgabewert	void

Beispiel:

```
SetCursorPos(3,4,0,0); /* Positioniere auf Menüpunkt, der Bildaufruf 4 enthält */
```

Handbuch Bediengeräte

3.2.4.7 Cursorposition abfragen

Wenn ein Menübild aktiv ist, kann man über diese Funktion die Position des Cursors abfragen.

Deklaration:	void GetCursorPos (int * __Mode , int * __X , int * __Y , int * __MPFunc)
Parameter:	<p>int * __Mode 0: Cursor unsichtbar (kein Menü aktiv) 1: Position in Zeile Y Spalte X 2: Gesetzt auf Sollwert mit Variablen-Handle X 3: Gesetzt auf Menüpunkt mit Funktion MPFunc und Parameter X</p> <p>int * __X Spalte oder Variablen Handle int * __Y Zeile oder Parameter</p> <p>int * __MPFunc 0 Bild aufrufen (X=Bildnummer) 1 Menü zurückgehen (X ohne Bedeutung) 2 Sollwert-Eingabe beenden mit speichern 3 Sollwert-Eingabe beenden ohne speichern 4 Globaler Abbruch 5 Menü-Indexausgabe (X = Menüindex) 6 Indexausgabe mit globalem Abbruch (X = Menüindex) 7 Menü-Index an Steuerprogramm geben (X = Menüindex)</p>
Rückgabewert	void

Beispiel:

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

GetCursorPos(&Mode,&X,&Y,&MPFunc); /* hole Werte */

if (Mode == 0)
{
.../* Anwenderprogramm wenn Cursor unsichtbar */
}
```

3.2.4.8 Menüpunktindex lesen

Ein Menüpunkt läßt sich mit der Funktion "Index an KOP geben" programmieren. Die Funktion GetCurrentMenuIndex() liefert diesen Wert, wenn der entsprechende Menüpunkt mit "Enter" gewählt wird.

Hinweis:

Der Wert steht exakt einmal im Unterprogramm "KOP_Cycle" zur Verfügung. Ein Zugriff aus der zeitgesteuerten Routine ist unzuverlässig!

Deklaration:	uint16 GetCurrentMenuIndex (void)
Parameter:	keine
Rückgabewert	int == 0: keine Taste gedrückt <> 0: Tastennummer der gedrückten Taste

Beispiel:

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

Handbuch Bediengeräte

3.2.4.9 Maske für die Softkey-Tasten lesen

Als Softkeys werden Tasten bezeichnet, die eigentlich eine Funktion für das Bediengerät haben, aber die der Anwender nur für die SPS oder für den KOP verwenden will. Die Tasten sind hierzu bitweise codiert; siehe hierzu auch die Beschreibung des CAN-Telegramms SET PARAM.

Deklaration:	int GetSoftkeyMask (void)
Parameter:	keine
Rückgabewert	int Softkey-Maske (bitweise codiert)

Beispiel:

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

3.2.4.10 Maske für die Softkey-Tasten setzen

Als Softkeys werden Tasten bezeichnet, die eigentlich eine Funktion für das Bediengerät haben, aber die der Anwender nur für die SPS oder für den KOP verwenden will. Die Tasten sind hierzu bitweise codiert; siehe hierzu auch die Beschreibung des CAN-Telegramms SET PARAM.

Deklaration:	void SetSoftkeyMask (int __State)
Parameter:	int __State Softkey-Maske (bitweise codiert)
Rückgabewert	keiner

Beispiel:

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

Übernahme:

sofort. Nach Neustart aber nur, wenn SaveToFlash ausgeführt wurde.

3.2.4.11 Anzahl Tasten-Erweiterungsblöcke lesen

Liefert die im Projekt hinterlegte Anzahl der Tastaturerweiterungen.

Deklaration:	int GetKeyBlockCount (void)
Parameter:	keine
Rückgabewert	int Anzahl Tastaturerweiterungen

Beispiel:

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

Handbuch Bediengeräte

3.2.5 Funktionen für das Arbeiten mit LEDs

3.2.5.1 Einschalten einer LED

Über diese Funktion kann eine LED eingeschaltet werden.

Deklaration:	void LedOn (int __LedNo)
Parameter:	int __LedNo Nummer der LED
Rückgabewert	keiner

Beispiel:

```
LedOn(8);
```

3.2.5.2 Ausschalten einer LED

Über diese Funktion kann eine LED ausgeschaltet werden.

Deklaration:	void LedOff (int __LedNo)
Parameter:	int __LedNo Nummer der LED
Rückgabewert	keiner

Beispiel:

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

3.2.5.3 Feststellen, ob eine LED eingeschaltet ist

Hier kann überprüft werden, ob eine LED auf der Frontseite eingeschaltet ist.
Die LED-Nummern sind geräteabhängig.

Deklaration:	int IsLedOn (int __LedNo)
Parameter:	int __LedNo Nummer der LED die abgefragt werden soll
Rückgabewert	int == 0: LED ist aus <> 0: LED ist an

Beispiel:

```
if (IsLedOn(2))  
{  
    ... /* Anwenderprogramm wenn LED an ist */  
}
```

Handbuch Bediengeräte

3.2.6 Funktionen für das Arbeiten mit Timern

3.2.6.1 Starten/Setzen eines Timers

Einen der 16 Timer starten. Es sind 16 Timer, 0-15, verfügbar. Diese können auf einen Wert gesetzt werden. Dieser Wert wird vom Betriebssystem automatisch heruntergezählt bis auf 0.

Über die Funktion "SetTimer" kann ein Timer gestartet werden. Es können mehrere Timer gleichzeitig laufen (sogar alle). Der Timerwert wird in Schritten von 10 ms angegeben..

Deklaration:	void SetTimer (int __TimerNo, long __Value)
Parameter:	int __TimerNo Nummer des Timers long __Value Timerwert
Rückgabewert	keiner

Beispiel:

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

3.2.6.2 Abfragen, ob ein Timer läuft

Mit dieser Funktion kann überprüft werden, ob der Timer noch läuft..

Deklaration:	int IsTimerOn (int __TimerNo)
Parameter:	int __TimerNo Nummer des Timers der abgefragt werden soll
Rückgabewert	int == 0: Timer steht (auf 0) <> 0: Timer läuft noch

Beispiel:

```
if (IsTimerOn(1))  
{  
... /* Anwenderprogramm solange der Timer läuft*/  
}
```

3.2.6.3 Abfragen, ob ein Timer steht

Mit dieser Funktion kann überprüft werden, ob der Timer abgelaufen ist. Diese Abfrage liefert genau das umgekehrte Ergebnis der Funktion "IsTimerOn".

Deklaration:	int IsTimerOff (int __TimerNo)
Parameter:	int __TimerNo Nummer des Timers der abgefragt werden soll
Rückgabewert	int == 0: Timer läuft noch <> 0: Timer steht (auf 0)

Beispiel:

```
if (IsTimerOff(12))  
{  
... /* Anwenderprogramm solange der Timer steht */  
}
```

Handbuch Bediengeräte

3.2.7 Funktionen für das Arbeiten mit Ein/Ausgängen

Die meisten Geräte haben entweder einen Ausgang, einen Eingang oder beides. Zusätzlich sind auch, je nach Gerätetyp, Ein/Ausgänge zum Steuern bereits in das Gerät integriert.

Mit den nachfolgenden Funktionen hat man direkten Zugriff auf die internen Ein/Ausgänge.

Außer den hier genannten Funktionen hat man noch Zugriff über die GCM-Abfragebefehle, wenn man die Knotennummer 0 benutzt.

Den Umgang mit Ein/Ausgängen mit CAN-Modulen finden Sie unter 3.2.10 Funktionen zur Ansteuerung von GCM-Can-Modulen.

3.2.7.1 Internen Meldeausgang einschalten

Mit dieser Funktion wird der Meldeausgang eingeschaltet. Zukünftig werden mehr Ausgänge schaltbar sein.

Deklaration:	void OutputOn (int __OutNo)
Parameter:	int __OutNo Nummer des Ausganges 1=Standard Meldeausgang
Rückgabewert	keiner

Beispiel:

```
OutputOn(1);
```

3.2.7.2 Internen Meldeausgang ausschalten

Mit dieser Funktion wird der Meldeausgang ausgeschaltet. Zukünftig werden mehr Ausgänge schaltbar sein.

Deklaration:	void OutputOff (int __OutNo)
Parameter:	int __OutNo Nummer des Ausganges 1=Standard Meldeausgang
Rückgabewert	keiner

Beispiel:

```
OutputOff(1);
```

3.2.7.3 Meldeausgang abfragen

Der geräteinterne Meldeausgang kann hiermit abgefragt werden. Zukünftig können über die Ausgangsnummer auch andere Ausgänge vorgesehen werden.

Deklaration:	int IsOutputOn (int __OutNo)
Parameter:	int __OutNo Nummer des Ausganges der abgefragt werden soll 1=Standard Meldeausgang
Rückgabewert	int == 0: Ausgang ist Low <> 0: Ausgang ist High

Beispiel:

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

Handbuch Bediengeräte

3.2.7.4 Light-In Eingang abfragen

Der geräteinterne LIGHT-IN Eingang kann hiermit abgefragt werden. Zukünftig können über die Eingangsnummer auch andere Eingänge vorgesehen werden.

Deklaration:	int IsInputOn (int __InNo)
Parameter:	int __InNo Nummer des Eingangs der abgefragt werden soll 1=Standard Light-In Eingang
Rückgabewert	int == 0: Eingang ist Low <> 0: Eingang ist High

Beispiel:

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

3.2.7.5 Internen digitalen Eingang abfragen

Ein digitaler Eingang auf einem Gerät mit zusätzlichen internen E/As kann hiermit abgefragt werden. Die Anzahl der Eingänge ist abhängig von der integrierten E/A-Ausführung. Die Eingangsnummern beginnen immer mit 0.

Deklaration:	int GetInternalDI (int __InNo)
Parameter:	int __InNo Nummer des Eingangs der abgefragt werden soll
Rückgabewert	int == 0: Eingang ist Low <> 0: Eingang ist High

Beispiel:

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

3.2.7.6 Internen digitalen Ausgang abfragen

Ein digitaler Ausgang auf einem Gerät mit zusätzlichen internen E/As kann hiermit abgefragt werden. Die Anzahl der Ausgänge ist abhängig von der integrierten E/A-Ausführung. Die Ausgangsnummern beginnen immer mit 0.

Deklaration:	int GetInternalDO (int __OutNo)
Parameter:	int __OutNo Nummer des Ausgangs der abgefragt werden soll
Rückgabewert	int == 0: Eingang ist Low <> 0: Eingang ist High

Beispiel:

```
if (GetInternalDO(12))  
{  
.../* Anwenderprogramm */  
}
```

Handbuch Bediengeräte

3.2.7.7 Internen digitalen Ausgang schalten

Ein digitaler Ausgang auf einem Gerät mit zusätzlichen internen E/As kann mit dieser Prozedur ein- oder ausgeschaltet werden. Die Anzahl der Ausgänge ist abhängig von der integrierten E/A-Ausführung. Die Ausgangsnummern beginnen immer mit 0.

Deklaration:	void SetInternalDO (int __OutNo, int __Value)
Parameter:	int __OutNo Nummer des Ausgangs der geschaltet werden soll int __Value Wert
Rückgabewert	keiner

Beispiel:

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

3.2.7.8 Internen analogen Eingang lesen

Ein analoger Eingang auf einem Gerät mit zusätzlichen internen E/As kann hiermit abgefragt werden. Die Anzahl der Eingänge ist abhängig von der integrierten E/A-Ausführung. Die Eingangsnummern beginnen immer mit 0.

Deklaration:	int GetInternalAI (int __InNo)
Parameter:	int __InNo Nummer des Eingangs der abgefragt werden soll
Rückgabewert	int Wert des digital gewandelten analogen Eingangssignals (unskaliert)

Beispiel:

```
SpannungDigital = GetInternalAI(1);
```

3.2.7.9 Eingangsbyte 0 lesen

Deklaration:	int GetDIB0 (void)
Parameter:	keine
Rückgabewert	int Status des Eingangsbytes 0 (Eingänge 0..7)

3.2.7.10 Eingangsbyte 1 lesen

Deklaration:	int GetDIB1 (void)
Parameter:	keine
Rückgabewert	int Status des Eingangsbytes 1 (Eingänge 8..15)

3.2.7.11 Eingangswort 0 lesen

Deklaration:	uint16 GetDIW0 (void)
Parameter:	keine
Rückgabewert	uint16 Status des Eingangswortes 0 (Eingänge 0..15)

Handbuch Bediengeräte

3.2.7.12 Ausgangsbyte 0 lesen

Deklaration:	int GetDOB0 (void)
Parameter:	keine
Rückgabewert	int Status des Ausgangsbytes 0 (Ausgänge 0..7)

3.2.7.13 Ausgangsbyte 0 schreiben

Deklaration:	void SetDOB0 (int __Value)
Parameter:	int __Value Status des Ausgangsbytes 0 (Ausgänge 0..7)
Rückgabewert	keiner

3.2.7.14 Ausgangsbyte 1 lesen

Deklaration:	int GetDOB1 (void)
Parameter:	keine
Rückgabewert	int Status des Ausgangsbytes 1 (Ausgänge 8..15)

3.2.7.15 Ausgangsbyte 1 schreiben

Deklaration:	void SetDOB1 (int __Value)
Parameter:	int __Value Status des Ausgangsbytes 1 (Ausgänge 8..15)
Rückgabewert	keiner

3.2.7.16 Ausgangswort 0 lesen

Deklaration:	uint16 GetDOW0 (void)
Parameter:	keine
Rückgabewert	uint16 Status des Ausgangswortes 0 (Ausgänge 0..15)

3.2.7.17 Ausgangswort 0 schreiben

Deklaration:	void SetDOW0 (uint16 __Value)
Parameter:	uint16 __Value Status des Ausgangswortes 0 (Ausgänge 0..15)
Rückgabewert	keiner

3.2.7.18 Kurzschlußstatus lesen

Deklaration:	uint16 GetShortcutFlags (void)
Parameter:	keine
Rückgabewert	uint16 Bitweise für Ausgänge 0..15

Handbuch Bediengeräte

3.2.8 Schnelle 2-kanalige Zähler

Die folgenden Funktionen dienen der Abfrage und der Ansteuerung der geräteinternen schnellen Zählerhardware, sofern vorhanden.

3.2.8.1 Zählerwert 1 lesen

Deklaration:	uint32 GetCount1 (void)
Parameter:	keine
Rückgabewert	uint32 Zählerwert

3.2.8.2 Zählerwert 1 schreiben

Deklaration:	void SetCount1 (uint32 __Cnt)
Parameter:	uint32 __Cnt Zählerwert auf den der Zähler gesetzt werden soll
Rückgabewert	keine

3.2.8.3 Zählerwert 2 lesen

Deklaration:	uint32 GetCount2 (void)
Parameter:	keine
Rückgabewert	uint32 Zählerwert

3.2.8.4 Zählerwert 2 schreiben

Deklaration:	void SetCount2 (uint32 __Cnt)
Parameter:	uint32 __Cnt Zählerwert auf den der Zähler gesetzt werden soll
Rückgabewert	keine

3.2.8.5 Vorwahl 1 für Zähler 1 lesen

Deklaration:	uint32 Get_C1_Preset1 (void)
Parameter:	keine
Rückgabewert	uint32 Vorwahlwert

3.2.8.6 Vorwahl 1 für Zähler 1 schreiben

Deklaration:	void Set_C1_Preset1 (uint32 __Preset)
Parameter:	uint32 __Value Wert auf den der Vorwahlwert gesetzt werden soll
Rückgabewert	keine

Handbuch Bediengeräte

3.2.8.7 Vorwahl 2 für Zähler 1 lesen

Deklaration:	uint32 Get_C1_Preset2 (void)
Parameter:	keine
Rückgabewert	uint32 Vorwahlwert

3.2.8.8 Vorwahl 2 für Zähler 1 schreiben

Deklaration:	void Set_C1_Preset2 (uint32 __Preset)
Parameter:	uint32 __Preset Wert auf den der Vorwahlwert gesetzt werden soll
Rückgabewert	keine

3.2.8.9 Vorwahl 1 für Zähler 2 lesen

Deklaration:	uint32 Get_C2_Preset1 (void)
Parameter:	keine
Rückgabewert	uint32 Vorwahlwert

3.2.8.10 Vorwahl 1 für Zähler 2 schreiben

Deklaration:	void Set_C2_Preset1 (uint32 __Preset)
Parameter:	uint32 __Preset Wert auf den der Vorwahlwert gesetzt werden soll
Rückgabewert	keine

3.2.8.11 Vorwahl 2 für Zähler 2 lesen

Deklaration:	uint32 Get_C2_Preset2 (void)
Parameter:	keine
Rückgabewert	uint32 Vorwahlwert

3.2.8.12 Vorwahl 2 für Zähler 2 schreiben

Deklaration:	void Set_C2_Preset2 (uint32 __Preset)
Parameter:	uint32 __Preset Wert auf den der Vorwahlwert gesetzt werden soll
Rückgabewert	keine

Handbuch Bediengeräte

3.2.9 Zugriff auf die CAN-Schnittstellen

3.2.9.1 Datenformat für Standard-Identifizier

Im CAN-Bus wird für die Teilnehmer eine 11-bit Adresse (Identifizier) verwendet.

Zu diesen 11 bit kommen hinzu:

1 Bit RTR (remote request)

4 Bit Datenlänge (DLC) gültige Werte 0-8

Das ergibt eine Gesamtbitzahl von 16. Die Bits in diesem 16-bit-Wort sind wie folgt belegt:

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

Um den eigentlichen Identifizier zu erhalten, muß der gelesene Wert also um 5 bit nach rechts geschoben werden. Beim Schreiben natürlich um 5 bit nach links. Dies gilt für alle folgenden Funktionen zu den Identifiern.

Wenn 29-bit Identifizier verwendet werden sollen, so muß dies über die CAN-Parametrierfunktionen erfolgen. Im Editor läßt sich zur Zeit kein 29-bit Identifizier einstellen. Damit die Funktionen mit 29-Bit Identifiern arbeiten, muß die Id bei Verwendung von 29-Bit Identifiern immer mit dem Macro USE_29BIT_ID ODER-verknüpft werden.

Soll mit Funktionen auf die 2. CAN-Schnittstelle zugegriffen werden, dann wird in diesem Fall der Identifizier mit dem Macro USE_CAN_1 ODER-verknüpft.

3.2.9.2 CAN-Busrate einstellen

Die Busrate wird nicht absolut angegeben, sondern als Indexwert aus einer Tabelle:

Deklaration:	void SetCAN0BusrateIndex (int __Index) void SetCAN1BusrateIndex (int __Index);
Parameter:	__Index:siehe Tabelle
Rückgabewert	keiner

Index	Busrate
0	10 kBit/s
1	20 kBit/s
2	50 kBit/s
3	100 kBit/s
4	125 kBit/s
5	250 kBit/s
6	500 kBit/s
7	1 Mbit/s

Da die Geräte der Serie AT als ITS-kompatible Geräte entworfen wurden, ist im Projekt nur Speicher für die Parametrierung einer CAN-Schnittstelle vorhanden. Vom Betriebssystem werden nun beide CAN-Schnittstellen mit derselben Baudrate initialisiert. Wenn man unterschiedliche Baudraten benötigt, geht man sinnvollerweise so vor:

- *Im Projekt die Baudrate für die erste CAN-Schnittstelle einstellen.*
- *Im KOP-Init die Funktion SetCAN1BusrateIndex(...) aufrufen.*
- *Danach im KOP-Init RestartCanSystem() aufrufen.*

Auf diese Weise kann auf der 2. CAN-Schnittstelle eine andere Baudrate verwendet werden.

Übernahme:

Temporär nach RestartCanSystem()

Permanent nach SaveToFlash() und RestartCanSystem() (nur CAN0-Parameter)

Handbuch Bediengeräte

3.2.9.3 CAN-Busrate lesen

Die Funktion liefert des Tabellenindex der aktuell an der jeweiligen CAN-Schnittstelle eingestellten Busrate. Die Funktion kann **nicht** dazu eingesetzt werden, die aktuell am Bus benutzte Busrate zu ermitteln. Diese muß bekannt sein !

Deklaration:	int GetCAN0BusrateIndex (void) int GetCAN1BusrateIndex (void);
Parameter:	keine
Rückgabewert	int Tabellenindex. Siehe "CAN-Busrate einstellen" auf Seite 43.

3.2.9.4 Standard Sende-Identifizier lesen

Liefert den eingestellten Sende-Identifizier incl. RTR und DLC zurück

Deklaration:	uint32 GetCANTxId (void)
Parameter:	keine
Rückgabewert	uint32 Identifizier <i>incl.</i> RTR und DLC

Beispiel:

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

3.2.9.5 Standard Sende-Identifizier schreiben

Setzt den Sender Identifizier auf den gewünschten Wert

Deklaration:	void SetCANTxId (uint32 __Id)
Parameter:	__Id Identifizier <i>incl.</i> RTR und DLC
Rückgabewert	keiner

Beispiel:

```
SetCANTxId(1000 << 5);
```

Übernahme:

Temporär nach RestartCanSystem()

Permanent nach SaveToFlash() und RestartCanSystem()

3.2.9.6 Standard Empfangs-Identifizier lesen

Liefert den eingestellten Empfangs-Identifizier incl. RTR und DLC zurück

Deklaration:	uint32 GetCANRxId0 (void)
Parameter:	keine
Rückgabewert	uint32 Identifizier <i>incl.</i> RTR und DLC

Beispiel:

```
iCanTxId = GetCANRxId0() >> 5;
```

Handbuch Bediengeräte

3.2.9.7 Standard Empfangs-Identifizierer schreiben

Setzt den Empfangs-Identifizierer auf den gewünschten Wert

Deklaration:	void SetCANRxId0 (uint32 __Id)
Parameter:	__Id Identifizierer <i>incl.</i> RTR und DLC
Rückgabewert	keiner

Beispiel:

```
SetCANRxId0(1001 << 5);
```

Übernahme:

Temporär nach RestartCanSystem()

Permanent nach SaveToFlash() und RestartCanSystem()

3.2.9.8 Multimaster-Empfangskanäle lesen

Diese Funktionen liefern den Empfangs-Identifizierer der Multi-Master Kanäle.

Deklarationen:	uint32 GetCANRxId1 (void); uint32 GetCANRxId2 (void); uint32 GetCANRxId3 (void); uint32 GetCANRxId4 (void); uint32 GetCANRxId5 (void); uint32 GetCANRxId6 (void); uint32 GetCANRxId7 (void);
Parameter:	keine
Rückgabewert	uint32 Identifizierer <i>incl.</i> RTR und DLC

Beispiel:

```
iCanRxMasterId1 = GetCANRxId1() >> 5;
```

3.2.9.9 Multimaster-Empfangskanäle schreiben

Diese Funktionen liefern den Empfangs-Identifizierer der Multi-Master Kanäle.

Deklarationen:	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 Identifizierer <i>incl.</i> RTR und DLC
Rückgabewert	keiner

Beispiel:

```
SetCANRxId4(13 << 5);
```

Übernahme:

Temporär nach RestartCanSystem()

Permanent nach SaveToFlash() und RestartCanSystem()

Handbuch Bediengeräte

3.2.9.10 Neuinitialisierung der CAN-Schnittstellen

Diese Funktion dient dazu, die Kommunikation auf dem CAN-Bus wieder zu starten, nachdem ein Fehler zu einem Zustand BUSOFF geführt hat, oder nachdem die Parameter der CAN-Schnittstelle(n) geändert wurden.

Deklarationen:	void RestartCanSystem (void);
Parameter:	keine
Rückgabewert	keiner

3.2.9.11 Status des CAN-Bus prüfen

Deklarationen:	uint16 Get_CAN_Status (void);
Parameter:	keine
Rückgabewert	uint16 Status als 16-Bit-Wert

3.2.9.12 Senden eines CAN-Telegramms

Über diese Funktion kann man ein CAN-Telegramm mit beliebigen Daten und beliebigem Identifier auf einen CAN-Bus versenden.

Deklaration:	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 und DLC) 11-Bit ID=0 ... 2047 29-Bit ID=0 ... 536870911 int __Len: Anzahl zu sendende Datenbytes char __D0: Datenbyte D0 char __D1: Datenbyte D1 char __D2: Datenbyte D2 char __D3: Datenbyte D3 char __D4: Datenbyte D4 char __D5: Datenbyte D5 char __D6: Datenbyte D6 char __D7: Datenbyte D7
Rückgabewert	keiner

Beispiele:

```
/* Sendet mit 11-bit Identifier 144, 3 Byte, auf CAN-Schnittstelle 0 */  
SendCANTelegram(144,3,0x12,0x06,0xFE,0,0,0,0,0);
```

```
/* Sendet mit dem 29-Bit Identifier 144, 3 Byte, auf CAN-Schnittstelle 0 */  
SendCANTelegram(144 | USE_29BIT_ID,3,0x12,0x06,0xFE,0,0,0,0,0);
```

```
/* Sendet mit dem 29-Bit Identifier 6122, 2 Byte, auf CAN-Schnittstelle 1 */  
SendCANTelegram(6122 | USE_29BIT_ID | USE_CAN_1,2,0x10,0x01,0,0,0,0,0,0);
```

Hinweis:

Um mit 11 Bit auf Identifier 0 senden zu können, muß man als ID 0x800 verwenden. Bei der Einstellung 11-Bit sendet ID=0 auf dem Transmit-Identifier (TxID) aus der Projekteinstellung.

3.2.9.13 Initialisieren von periodischen CAN-Telegrammen

In manchen Fällen müssen CAN-Telegramme periodisch gesendet werden. Hierzu gibt es eine Samm-

Handbuch Bediengeräte

lung von Systemaufrufen, die dies automatisch vornehmen. Bis zu (derzeit) 4 CAN-Telegramme können so periodisch abgesetzt werden. Zunächst muß aber dieses periodische Senden initialisiert werden. Hierzu dient die Funktion CANPeriodicSendInit.

Deklaration:	int CANPeriodicSendInit(void)
Parameter:	keiner
Rückgabewert	int -1 bei Fehler

Hinweise zur Verwendung von periodischen Sendungen:

Über die im folgenden beschriebenen Funktionen können Telegramme für das periodische Senden vorbereitet werden:

CANPeriodicGetSendHandle/CANPeriodicGetPGNSendHandle melden ein Telegramm zum periodischen Senden an, starten das periodische Senden jedoch noch nicht. Als Ergebnis erhält man ein Handle (Kennnummer) für das Telegramm. Dieses Handle wird dann in den darauffolgenden Funktionen benötigt.

Nach der Anmeldung sollte man über die Funktion CANPeriodicSetSendData die Daten in das Telegramm transportieren. Mit CANPeriodicSendStart wird nun das Telegramm im angegebenen Intervall automatisch gesendet. Über CANPeriodicSetSendData können die Daten jederzeit modifiziert werden, ohne das Senden zu unterbrechen. Mit CANPeriodicSendStop wird das periodische Senden des Telegramms wieder gestoppt. Man kann anschließend über CANPeriodicSendStart erneut starten oder aber das Telegramm komplett abmelden mit CANPeriodicReleaseHandle.

Ein typisches Beispiel:

```
/* Handle für ein periodisches Telegramm global definiert */
int MyPeriodicTGM;
/* Die Daten für das Telegramm */
uint8 MyTGMDData[8];

/* im KOP_Init startet das periodische Senden */
CanPeriodicSendInit();

/* Telegramm mit ID 0x300 und Datenlänge 5 */
MyPeriodicTGM = CANPeriodicGetSendHandle(0x300, 5);

/* Daten vorerst auf 0 setzen */
for (i=0; i<5; i++)
    MyTGMDData[i]=0;
/* und in das Telegramm transportieren */
CANPeriodicSetSendData(MyPeriodicTGM, MyTGMDData);
....
/* im KOP_Cycle über Tasten das Telegramm starten und stoppen: */
if (IsKeyPressedNew(1))
    CANPeriodicSendStart(MyPeriodicTGM, 10); /* alle 10*10ms=100ms senden*/

if (IsKeyPressedNew(2))
    CANPeriodicSendStop(MyPeriodicTGM); /* Telegramm nicht mehr senden */

/* neue Daten setzen */
if (IsKeyPressedNew(3))
{
    MyTGMDData[0]++;
    CANPeriodicSetSendData(MyPeriodicTGM, MyTGMDData);
}
```

Diese Funktionsvielfalt erlaubt eine hohe Flexibilität.

Handbuch Bediengeräte

3.2.9.14 Handle für periodisches Telegramm reservieren

Mit dieser Funktion wird ein Handle reserviert, das ein Telegramm mit einem bestimmten Identifier und einer festen Datenlänge kennzeichnet. Über dieses Handle wird auf das Telegramm später zugegriffen.

Deklaration:	int CANPeriodicGetSendHandle (uint32 __Id, int __Len)
Parameter:	uint32 __Id Identifier 11-Bit ID=0 ... 2047 29-Bit ID=0 ... 536870911 Die Konstanten USE_CAN_1 und USE_29BIT_ID erlauben die Zuordnung auf 29 Bit ID bzw. die CAN-Schnittstelle
Rückgabewert	int das Handle für das angemeldete Telegramm

Beispiele:

```
/* 11 Bit ID 445 Telegramm auf Kanal 0 mit Länge 8 anmelden: */  
int TGMHandle;  
TGMHandle = CANPeriodicGetSendHandle(445, 8);
```

```
/* 29 Bit ID 30223 Telegramm auf Kanal 1 mit Länge 3 anmelden: */  
int TGMHandle;  
TGMHandle = CANPeriodicGetSendHandle(30223 | USE_CAN_1 | USE_29BIT_ID, 3);
```

3.2.9.15 Handle für periodisch zu sendende PGN reservieren (J1939)

Für Steueraufgaben in J1939-Systemen können Telegramme periodisch gesendet werden. Die hier beschriebene Funktion ist eine für J1939 angepasste Funktion „CANPeriodicGetSendHandle“, in der man den Identifier nicht erst bestimmen muß, sondern direkt PGN, SA und PRIO direkt in die Funktion geben kann. Man erhält ebenfalls einen Handle als Ergebnis. Ein J1939 Telegramm wird immer als 29 Bit Identifier behandelt. Die Funktion:

Deklaration:	int CANPeriodicGetPGNSendHandle (int __Kanal, uint8 __Prio, uint16 __PGN, uint8 __SourceAddress, int __Len);
Parameter:	int __Kanal 0 = Standard-CAN-Schnittstelle, 1 = zweite CAN-Schnittstelle uint8 __Prio Bits 0-2 werden in das Feld PRIO eingefügt (Bits 26-28) uint16 __PGN PGN die periodisch gesendet werden soll (Bits 8-23 des J1939-ID) uint8 __SourceAddress Bits 0-7 des J1939 Identifiers int __Len Datenlänge des PGN-Telegramms
Rückgabewert	int das Handle für das angemeldete Telegramm

Beispiel:

```
Anmelden des TSC1-PGNs als periodisch zu sendendes Telegramm:  
int hTSC1;  
hTSC1 = CANPeriodicGetPGNSendHandle(0, 6, 0x0000, 3, 8);
```

Datenübergabe erfolgt mit CANPeriodicSetSendData, der Start des periodischen Sendens mit CANPeriodicSendStart.

3.2.9.16 Ein reserviertes Handle wieder freigeben

Da nur 4 Handles zur Verfügung stehen, muß man mit den Handles haushalten. Hat man eine Aufgabe erledigt und braucht das Handle nicht mehr, sollte man es mit CANPeriodicReleaseHandle wieder freigeben. Ist das Senden des Telegramms noch aktiv, dann wird dieses automatisch mit beendet.

Deklaration:	void CANPeriodicReleaseHandle (int __Handle)
--------------	---

Handbuch Bediengeräte

Parameter:	int __Handle Handle, das wieder freigegeben werden soll.
Rückgabewert	keiner

3.2.9.17 Periodisches Senden starten

Mit dieser Funktion wird das periodische Senden eines vorher angemeldeten Telegramms mit dem angegebenen Intervall gestartet.

Deklaration:	void CANPeriodicSendStart (int __Handle , int __Intervall)
Parameter:	int __Handle Handle des gewünschten Telegramms int __Intervall Sendeintervall in 10ms - Schritten
Rückgabewert	keiner

Beispiel:

```
int TGMHandle;  
TGMHandle = CANPeriodicGetSendHandle(445, 8);  
...  
CANPeriodicSendStart(TGMHandle,5); /* 50 ms Periode */
```

3.2.9.18 Periodisches Senden stoppen

Das periodische Senden eines Telegramms kann auch wieder gestoppt werden. Das Handle des Telegramms geht aber dabei nicht verloren (es wird nicht freigegeben. Dies muß über `CANPeriodicReleaseHandle` erfolgen). Das heißt, es kann später wieder gestartet werden.

Deklaration:	void CANPeriodicSendStop (int __Handle)
Parameter:	int __Handle Handle des gewünschten Telegramms
Rückgabewert	keiner

Beispiel:

```
int TGMHandle;  
TGMHandle = CANPeriodicGetSendHandle(445, 8);  
...  
CANPeriodicSendStart(TGMHandle,5); /* 50 ms Periode */  
...  
CANPeriodicSendStop(TGMHandle); /* nicht mehr senden */
```

3.2.9.19 Daten in das Telegramm setzen

Unabhängig vom Sendestatus kann mit dieser Funktion der Dateninhalt eines Telegramms verändert werden. Dabei wird aber der Status des Telegramms (senden oder nicht senden) nicht verändert. Dies geschieht ausschließlich über die Start- und Stop Funktionen. Der neue Inhalt wird bei der nächsten Sendeanforderung (Intervall abgelaufen) gesendet. Die Daten werden aus dem angegebenen Bereich herauskopiert; das heißt, das Datenfeld kann anschließend verändert werden, ohne dass sich der Telegramminhalt verändert.

Deklaration:	void CANPeriodicSetSendData (int __Handle , uint8 *pData)
Parameter:	int __Handle Handle des gewünschten Telegramms uint8 * pData Zeiger auf das Datenfeld (8 byte)
Rückgabewert	keiner

Beispiel:

```
int TGMHandle;  
uint8 TGMDData[8];
```

Handbuch Bediengeräte

```
TGMHandle = CANPeriodicGetSendHandle(445, 8);
...
for (i=0; i<8; i++)
    TGMData[i]=i;
CANPeriodicSetSendData(TGMHandle, TGMData);
```

3.2.9.20 Empfangsidentifizier dynamisch freischalten

Über diese Funktion kann ein Empfangs-Identifizier für eine CAN-Schnittstelle freigeschaltet werden.

Deklaration:	void CANEnableRxId (int __Channel , uint32 __Id)
Parameter:	int __Channel Kanal 0 = Standard CAN-Schnittstelle 1 = 2. CAN-Schnittstelle uint32 __Id Identifizier 11-Bit ID=0 ... 2047 29.Bit ID=0 ... 536870911
Rückgabewert	keiner

Beispiele:

```
/* 11-Bit-Identifizier 1601 auf Kanal 0 freischalten */
CANEnableRxId(0,1601);
```

```
/* 29-Bit-Identifizier 1601 auf Kanal 0 freischalten */
CANEnableRxId(0,1601 | USE_29BIT_ID);
```

```
/* 29-Bit-Identifizier 1601 auf Kanal 1 freischalten */
CANEnableRxId(1,1601 | USE_29BIT_ID);
```

3.2.9.21 Dynamisch zugewiesene Empfangsidentifizier sperren

Über diese Funktion kann ein freigeschalteter Empfangs-Identifizier einer CAN-Schnittstelle wieder gesperrt werden.

Deklaration:	void CANDisableRxId (int __Channel , uint32 __Id)
Parameter:	int __Channel Kanal 0 = Standard CAN-Schnittstelle 1 = 2. CAN-Schnittstelle uint32 __Id Identifizier 11-Bit ID=0 ... 2047 29-Bit ID=0 ... 536870911
Rückgabewert	keiner

Beispiele:

```
/* 11-Bit-Identifizier 1601 auf Kanal 0 sperren */
CANDisableRxId(0,1601);
```

```
/* 29-Bit-Identifizier 1601 auf Kanal 0 sperren */
CANDisableRxId(0,1601 | USE_29BIT_ID);
```

```
/* 29-Bit-Identifizier 1601 auf Kanal 1 sperren */
CANDisableRxId(1,1601 | USE_29BIT_ID);
```

Handbuch Bediengeräte

3.2.9.22 Freigabemaske für Empfangsidentifizier setzen

Mit dieser Funktion kann ein Identifier-Bereich freigeschaltet werden. Im Parameter `__Mask` wird angegeben, welche Bits beim Empfang relevant sein sollen. Im Parameter `__Id` wird der Bitwert angegeben. Beim Empfang wird dann eine Prüfung $(MsgId \& Mask) == (__Id \& Mask)$ durchgeführt. Ergibt die Prüfung `True`, dann wird der CAN-Event ausgelöst.

Deklaration:	<code>void CANEnableRxMask(uint32 __Channel, uint32 __Id, uint32 __Mask)</code>
Parameter:	<code>__Channel</code> CAN-Kanal <code>__Id</code> Identifier <code>__Mask</code> Relevante Bits
Rückgabewert	keiner

3.2.9.23 Freigabemaske für Empfangsidentifizier entfernen

Deklaration:	<code>void CANDisableRxMask(uint32 __Channel, uint32 __Id, uint32 __Mask)</code>
Parameter:	<code>__Channel</code> CAN-Kanal <code>__Id</code> Identifier <code>__Mask</code> Relevante Bits
Rückgabewert	keiner

3.2.9.24 Feststellen, ob das Gerät als CAN-Master läuft

Man kann feststellen, ob das Gerät als Master konfiguriert ist (SELECAN oder CAN-open). Ändern kann man es allerdings nicht...

Deklaration:	<code>int GetMasterFlag(void)</code>
Parameter:	keine
Rückgabewert	<code>int</code> = 0: Gerät läuft nicht als Master <>0: Gerät läuft als Master

Beispiel:

```
if (GetMasterFlag())  
{  
    Master-Dinge tun...  
}
```

3.2.9.25 CAN-open Übertragung überwachen

Mit dieser Funktion kann die CANopen-SDO-Übertragung überwacht werden. ACHTUNG! Diese Funktion wird vermutlich zugunsten einer vollen CANopen-Funktionalität entfallen. Es ist geplant, einen kompletten CANopen-Stack in das Gerät zu implementieren.

Deklaration:	<code>int SDOStatus(int __SDOno)</code>
Parameter:	<code>int SDOnr</code> zu überwachender SDO
Rückgabewert	<code>int</code> 0 wenn SDO nicht übertragen wurde <>0 wenn SDO übertragen wurde

Handbuch Bediengeräte

3.2.9.26 SELECAN Gerätenummer lesen

Wenn das Gerät im SELECAN-Modus betrieben wird, kann mit dieser Funktion die SELECAN-Gerätenummer ausgelesen ermittelt werden geändert werden. Dies sollten allerdings nur erfahrene Anwender tun.

Deklaration:	int GetSELECANNodeNo (void)
Parameter:	keine
Rückgabewert	int Gerätenummer

3.2.9.27 SELECAN Gerätenummer schreiben

Wenn das Gerät im SELECAN-Modus betrieben wird, kann mit dieser Funktion die SELECAN-Gerätenummer ausgelesen geändert werden. Dies sollten allerdings nur erfahrene Anwender tun.

Deklaration:	void SetSELECANNodeNo (int __NodeId)
Parameter:	int __NodeId Geräteadresse
Rückgabewert	keiner

3.2.10 Funktionen zur Ansteuerung von GCM-Can-Modulen

Die Can-Module der Serie GCM von GRAF-SYTECO können verhältnismäßig einfach angesteuert werden. Man muß diese im CAN-Konfigurationstool nur anlegen, ohne Ein/Ausgangsfunktionen zu konfigurieren.

Aus dem C-Code heraus kann man dann Eingänge lesen und Ausgänge setzen, digital wie auch analog. Die Eingänge werden laufend aktualisiert, während die Ausgänge erst am Ende des zyklischen Programmteils an die Module ausgegeben werden.

Benutzt man die Modulnummer 0, so hat man Zugriff auf die geräteinternen E/As.

3.2.10.1 Externen digitalen Eingang abfragen

Über diese Funktion kann ein digitaler Eingang abgefragt werden.

Deklaration:	int GetGCM_DI (int __NodeId, int __InNo)
Parameter:	int __NodeId Geräteadresse des GCM-Moduls int __InNo Nummer des digitalen Eingangs
Rückgabewert	int 0:Eingang ist aus <>0: Eingang ist ein

Beispiel:

```
/* Eingang 9 an Modul mit Knotennummer 2 abfragen */  
if (GetGCM_DI(2,9))
```

Handbuch Bediengeräte

3.2.10.2 Externen analogen Eingang abfragen

Über diese Funktion kann ein analoger Eingang abgefragt werden. Der Wert ist nicht skaliert und liefert exakt den binären Wert des A/D-Wandlers aus dem Modul.

Deklaration:	int GetGCM_AI (int __NodeId, int __InNo)
Parameter:	int __NodeId Geräteadresse des GCM-Moduls int __InNo Nummer des analogen Eingangs
Rückgabewert	int Wert siehe Handbuch des Moduls

Beispiel:

```
/* Eingang 2 an Modul mit Knotennummer 1 abfragen */  
Wert = GetGCM_AI(1,2);
```

3.2.10.3 Externen digitalen Ausgang setzen

Über diese Funktion kann ein digitaler Ausgang gesetzt werden.

Deklaration:	void SetGCM_DO (int __NodeId, int __OutNo, int __Value)
Parameter:	int __NodeId Geräteadresse des GCM-Moduls int __InNo Nummer des digitalen Eingangs int __Value 0 oder 1 (logisch!)
Rückgabewert	keiner

Beispiel:

```
/* Ausgang 0 an Modul mit Knotennummer 5 setzen */  
SetGCM_DO(5,0,1);
```

Anmerkung:

Ein Ausgang kann auch über die Funktion "digitalen Ausgang lesen" zurückgelesen werden. Man nimmt einfach als Eingangsnummer die Ausgangsnummer.....

3.2.10.4 Externen analogen Ausgang setzen

Über diese Funktion kann ein analoger Ausgang gesetzt werden. Der Wert ist nicht skaliert und muß exakt dem binären Wert des D/A-Wandlers aus dem Modul entsprechen.

Deklaration:	void SetGCM_AO (int __NodeId, int __OutNo, int __Value)
Parameter:	int __NodeId Geräteadresse des GCM-Moduls int __InNo Nummer des digitalen Eingangs int __Value 0 oder 1 (logisch!)
Rückgabewert	keiner

Beispiel:

```
/* Setze Ausgang 1 an Modul mit Knotennummer 1 */  
SetGCM_AO(1,1,Wert);
```

Handbuch Bediengeräte

3.2.10.5 Modul-Status abfragen

Über diese Funktion kann der Status eines Moduls abgefragt werden.

Deklaration:	uint16 GetGCM_Status (int __NodeId)
Parameter:	int __NodeId Geräteadresse des GCM-Moduls
Rückgabewert	noch nicht festgelegt

Beispiel:

```
/* Status des Moduls mit Knotennummer 1 abfragen */  
Status = GetGCM_Status(1);
```

3.2.11 Zugriff auf die seriellen Schnittstellen

3.2.11.1 Serielle Schnittstellen konfigurieren

Für die seriellen Schnittstellen sind 2 Parameter vorhanden; einerseits die Baudrate, andererseits das Datenformat. Die Baudrate ist eine Tabelle mit folgenden Werten.

Wie auch bei den CAN-Schnittstellen sind aus Kompatibilitätsgründen nur für die erste serielle Schnittstelle (Serial0) Parameter vorgesehen. Deshalb muß man, wenn man mit verschiedenen Baudraten arbeitet, dasselbe Verfahren wie beim CAN für die seriellen Schnittstellen anwenden:

- *Im Projekt Baudrate und Datenformat für die erste serielle Schnittstelle einstellen*
- *Im KOP-Init die Funktion `SetSerial1BaudrateIndex(...)` aufrufen*
- *Im KOP-Init die Funktion `SetSerial1FrameSetting(...)` aufrufen*
- *Danach im KOP-Init `RestartComSystem()` aufrufen*

3.2.11.2 Baudrate lesen

Die eingestellte Baudrate der seriellen Schnittstelle wird nicht direkt zurückgegeben, sondern als Index der nachfolgenden Tabelle. Diese Funktion kann **nicht** dazu verwendet werden, die Baudrate der seriellen Kommunikation zu ermitteln. Diese muß vorab bekannt sein ! Sie dient nur dazu, die Einstellung der Schnittstelle zu lesen.

Index	Baudrate
0	9600 Baud
1	4800 Baud
2	2400 Baud
3	1200 Baud
4	600 Baud
5	300 Baud
6	150 Baud
7	110 Baud

Deklaration:	int GetSerial0BaudIndex (void) int GetSerial1BaudIndex (void);
Parameter:	keine
Rückgabewert	int Index siehe Tabelle

Handbuch Bediengeräte

3.2.11.3 Baudrate einstellen

Stellt die Baudrate der seriellen Schnittstelle ein. Übergeben wird der Index der obigen Tabelle.

Deklaration:	void SetSerial0BaudIndex (int __Index) void SetSerial1BaudIndex (int __Index);
Parameter:	int __Index Index siehe Tabelle
Rückgabewert	keiner

3.2.11.4 Telegrammeinstellung lesen

Die Funktion liefert die Einstellung des Telegrammrahmens für die serielle Übertragung. Wie beim Lesen der Baudrate kann diese Funktion lediglich dazu benutzt werden, die Einstellung der Schnittstelle zu ermitteln, nicht jedoch zum Ermitteln der Telegrammrahmeneinstellung für die Kommunikation mit anderen Geräten. Diese muß vorher bekannt sein.

Das Datenformat der Telegrammrahmeneinstellung (PARAM) wird Bitweise beschrieben:

Bitnummer	Bedeutung
0	ohne Bedeutung
1	ohne Bedeutung
2	Anzahl Stopbits 0=1, 1=2
3	Anzahl Datenbits 0=7, 1=8
4	Parity 0=odd, 1=even
5	Parity 0=disabled 1=enabled
6	ohne Bedeutung
7	ohne Bedeutung

Deklaration:	int GetSerial0FrameSetting (void) int GetSerial1FrameSetting (void);
Parameter:	keine
Rückgabewert	int Wert siehe Tabelle

3.2.11.5 Telegrammeinstellung setzen

Stellt den Telegrammrahmen der seriellen Kommunikation ein.

Deklaration:	void SetSerial0FrameSetting (int __Frame) void SetSerial1FrameSetting (int __Frame);
Parameter:	int __Frame Wert siehe Tabelle
Rückgabewert	keiner

Beispiel für 3.2.11.3 bis 3.2.11.5:

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

Übernahme:

Temporär nach RestartComSystem()

Permanent nach SaveToFlash und RestartComSystem() (nur Serial 0)

Handbuch Bediengeräte

3.2.11.6 Neuinitialisierung der seriellen Schnittstellen

Nachdem die Parameter der seriellen Schnittstelle(n) geändert wurden, ist es erforderlich, diese neu zu Initialisieren und die geänderten Parameter zu übernehmen. Dies geschieht mittels der Funktion `RestartComSystem()`.

Deklaration:	void RestartComSystem (void)
Parameter:	keine
Rückgabewert	keiner

3.2.11.7 Serieller Empfang

Die seriellen Schnittstellen werden vom TOS interruptgesteuert verwaltet. Wenn von einer seriellen Schnittstelle Zeichen empfangen werden, schreibt das TOS diese in einen Empfangspuffer, der gelesen werden kann. Dazu gibt es eine Status- und eine Lesefunktion. Die Statusfunktion `GetStatSerialx()` liefert einen Wert ungleich 0, wenn ein Zeichen im Empfangspuffer steht. Nachdem das Zeichen mit `GetCharSerialx()` ausgelesen wurde, muß mit `SetStatSerialx(0)` das Zeichen quittiert werden.

3.2.11.8 Statusabfrage

Deklaration:	int GetStatSerial0 (void) int GetStatSerial1 (void)
Parameter:	keine
Rückgabewert	int 0=kein Zeichen empfangen <>0=Zeichen vorhanden

3.2.11.9 Empfangenes Zeichen lesen

Deklaration:	int GetCharSerial0 (void) int GetCharSerial1 (void)
Parameter:	keine
Rückgabewert	int ASCII-Code des empfangenen Zeichens

3.2.11.10 Empfangenes Zeichen quittieren

Deklaration:	void SetStatSerial0 (int __Stat) void SetStatSerial1 (int __Stat)
Parameter:	int 0 für Quittierung des empfangenen Zeichens
Rückgabewert	keiner

Handbuch Bediengeräte

3.2.11.11 Empfangenes Zeichen überschreiben

Deklaration:	void SetCharSerial0 (int __Char) void SetCharSerial1 (int __Char)
Parameter:	int __Char überschreibt eine empfangenes Zeichen mit __Char Diese Funktion macht für den Normalen Betrieb keinen Sinn !
Rückgabewert	keiner

Beispiel:

```
while (GetStatSerial0())  
{  
    Zeichen = GetCharSerial0();  
    .... Verarbeitung  
    SetStatSerial0(0);  
}
```

Anmerkung:

Mit dieser Schleife werden alle Zeichen des Empfangspuffers verarbeitet. Man erhält als Wert der Funktion GetCharSerial0() jeweils den ASCII-Code des empfangenen Zeichens, kann es also auch in eine char-Variable lesen.

Werden mehr Zeichen empfangen, als der Empfangspuffer aufnehmen kann, so gehen die zuletzt empfangenen Zeichen verloren.

3.2.11.12 Seriell senden

Diese Funktion erlaubt das Senden eines Puffers über die serielle Schnittstelle.

Deklaration:	void SendToSerial (int __Channel, char *__Buffer, int __Count, int __Timeout)
Parameter:	int __Channel 0 oder 1 je nach Schnittstelle char *__Buffer Zeiger auf erstes Zeichen in String int __Count Anzahl Zeichen int __Timeout max. Wartezeit in Millisekunden
Rückgabewert	keiner

Beispiel:

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

3.2.12 Funktionen zur Einstellung des Displays

3.2.12.1 Kontrast lesen

Die aktuelle Kontraststufe kann über diese Funktionen gelesen werden.

Deklaration:	int GetContrastIndex (void)
Parameter:	keine
Rückgabewert	int 0 .. 23 je nach eingestelltem Kontrast

Beispiel:

```
Kontrast = GetContrastIndex();
```

Handbuch Bediengeräte

3.2.12.2 Kontrast setzen

Die aktuelle Kontraststufe kann über diese Funktionen geschrieben werden.

Deklaration:	void SetContrastIndex (int __Index)
Parameter:	int __Index 0 .. 23 je nach gewünschtem Kontrast
Rückgabewert	keiner

Beispiel:

```
SetContrastIndex(Kontrast+1);
```

Übernahme:

Sofort. Die Kontraststufe wird sofort dauerhaft gespeichert.

3.2.12.3 Helligkeit lesen

Die aktuelle Helligkeitsstufe kann über diese Funktionen gelesen werden.

Deklaration:	int GetBrightnessIndex (void)
Parameter:	keine
Rückgabewert	int 0 .. 7 je nach eingestellter Helligkeit

Beispiel:

```
Helligkeit = GetBrightnessIndex();
```

3.2.12.4 Helligkeit setzen

Die aktuelle Helligkeitsstufe kann über diese Funktionen geschrieben werden.

Deklaration:	void SetBrightnessIndex (int __Index)
Parameter:	int __Index 0 .. 7 je nach gewünschter Helligkeit
Rückgabewert	keiner

Beispiel:

```
SetBrightnessIndex(Helligkeit-1);
```

Übernahme:

Sofort. Die Helligkeitsstufe wird sofort dauerhaft gespeichert.

3.2.12.5 Helligkeitswert für Tastenbeleuchtung lesen

Die aktuelle Einstellung der EL-Folie zur Tastenhinterleuchtung kann über diese Funktionen gelesen werden (geräteabhängig!).

Deklaration:	int GetELFOILState (void)
Parameter:	keine
Rückgabewert	int Helligkeitswert

Beispiel:

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

Handbuch Bediengeräte

3.2.12.6 Helligkeitswert für Tastenbeleuchtung setzen

Die aktuelle Einstellung der EL-Folie zur Tastenhinterleuchtung kann über diese Funktionen festgelegt werden (geräteabhängig !).

Deklaration:	void SetELFOILState (int __State)
Parameter:	int Helligkeitswert 0=aus 1=ein
Rückgabewert	keiner

Beispiel:

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

Übernahme:

Sofort. Die Helligkeitsstufe wird sofort dauerhaft gespeichert.

3.2.13 Grafische Funktionen

Manche Problemstellungen lassen sich nicht ohne weiteres mit dem Projektierool grafisch umsetzen, da die Darstellung variabel ist. Ein gutes Beispiel ist die Darstellung einer Messkurve.

Durch die komfortablen Möglichkeiten der C-Programmierung ist es möglich, direkt aus dem C-Code heraus auf das Display des Bediengeräts zu zeichnen.

Es gibt vier Ebenen (Layer) zum Zeichnen. Man kann den Hintergrund in eine Ebene zeichnen, den bewegten Teil in eine andere. Dann braucht man sich nicht darum zu kümmern, was vom Hintergrund neu aufgebaut werden muss.

Es gibt zum Zeichnen sogenannte direkte und indirekte Zeichenoperationen. Um das Flackern beim Bildaufbau zu minimieren, kann erst auf einen Speicherbereich gezeichnet werden, der dann auf einen Rutsch ins Display kopiert wird. Das macht den Bildaufbau sehr ruhig (indirektes Zeichnen).

Wenn man aber Bewegungen darstellen will, macht es Sinn, zunächst den Bildhintergrund im Speicher aufzubauen, ins Bild zu kopieren und dann nur noch den sich ändernden Bildteil zu zeichnen - dies dann direkt. Beim direkten Zeichnen erfolgt zusätzlich auch das indirekte Zeichnen in die zugehörige Ebene.

Es gibt insgesamt 4 Ebenen, in die gezeichnet werden kann.

Dies sind:

- Ebene für statische, nicht blinkende Elemente
- Ebene für statische, blinkende Elemente
- Ebene für dynamische, nicht blinkende Elemente
- Ebene für dynamische, blinkende Elemente

Welche der Ebenen angesprochen werden soll, wird im Attribut-Parameter bestimmt. Wenn z.B. das Blink-Bit gesetzt ist, wird automatisch in eine der Ebenen für blinkende Elemente gezeichnet, und das Betriebssystem sorgt für das Blinken. Sie müssen also nicht laufend in den Bildschirm schreiben und löschen. Wie Sie die Ebenen gezielt ansprechen, finden Sie in der Beschreibung des Attribut-Parameters.

Handbuch Bediengeräte

3.2.13.1 Aufbau des Attribut-Parameters

Auf dem Display dargestellte Element verfügen über einen Attribut-Parameter, der deren Erscheinung bestimmt. Dieser Attribut-Parameter ist bitweise aufgebaut. Die Bedeutung der einzelnen Bits entnehmen Sie bitte der folgenden Tabelle:

Bitnummer	Bedeutung
0	Element invertiert darstellen
1	Element blinkend darstellen (Ebenenauswahl)
2	Unterstrichen (nur für Zeichenausgabe)
3	Zeichen aus alternativem Zeichensatz (nur für Zeichenausgabe)
4/5	Zeichengröße (nur für Zeichenausgabe)
6	0 = indirektes Zeichnen, 1 = direktes Zeichnen
7	0 = statische Ebene, 1 = dynamische Ebene

3.2.13.2 Koordinatensystem

Sämtliche Zeichenoperationen verwenden pixelorientierte Koordinaten. Der Nullpunkt liegt links oben, die waagrechte Richtung ist die X-Koordinate und die senkrechte Richtung die Y-Koordinate. Bei manchen Funktionen wird auch Höhe und Breite oder Radius eines Objekts benötigt, deren Einheit dann ebenfalls ein Pixel ist.

Wird das AT3 verwendet, hat man noch einen Sonderfall: das Gerät läßt sich mit Tasten links oder Tasten unten einbauen. Wo sind die Nullpunktkoordinaten?

Antwort: immer links oben. Sie projizieren ja, ob die Tasten links oder unten sind. Das Grafik-Subsystem des TOS berücksichtigt diese Einstellung bei der Darstellung der Objekte. Sie müssen also keine Gedanken an Rotation oder so etwas verschwenden.

3.2.13.3 Farben

Auf Monochrom-Displays machen nur die Farben schwarz (Color = 0x000000) und weiß (Color = 0xFFFF) Sinn. Alle anderen Farben sind den Farb-Displays vorbehalten.

3.2.13.4 Einzelnen Bildpunkt (Pixel) zeichnen

Mit der Funktion `DrwPixel()` kann ein einzelner Bildpunkt gesetzt werden.

Deklaration:	<code>void DrwPixel(int __X, int __Y, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __X</code> X-Koordinate des Bildpunktes <code>int __Y</code> Y-Koordinate des Bildpunktes <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiele:

```
/* Zeichnet einen einzelnen Pixel, statische Ebene, direkte Ausgabe */  
DrwPixel(10,10,0x40,0);
```

```
/* Zeichnet einen einzelnen Pixel, dynamische Ebene, indirekte Ausgabe */  
DrwPixel(10,10,0x80,0);
```

Handbuch Bediengeräte

3.2.13.5 Rechteck ohne Füllung zeichnen

Mit der Funktion `DrwRect()` kann ein Rechteck ohne Füllung (ein Rahmen) gezeichnet werden. Die Rahmendicke ist ein Pixel. Das Rechteck verläuft parallel zu den Displaykanten.

Deklaration:	<code>void DrwRect(int __XLeft, int __YTop, int __XRight, int __YBottom, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __XLeft</code> X-Koordinate linke obere Ecke <code>int __YTop</code> Y-Koordinate linke obere Ecke <code>int __XRight</code> X-Koordinate rechte untere Ecke <code>int __YBottom</code> Y-Koordinate rechte untere Ecke <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiel:

```
/* Zeichnet einen blinkenden Rahmen, statische Ebene, direkte Ausgabe */  
DrwRect(20,20,80,30,0x42,0);
```

3.2.13.6 Rechteck mit Füllung zeichnen

Mit der Funktion `DrwRectFilled()` kann ein Rechteck mit Füllung gezeichnet werden. Die Füllung erfolgt in der Rahmenfarbe. Das Rechteck verläuft parallel zu den Displaykanten.

Deklaration:	<code>void DrwRectFilled(int __XLeft, int __YTop, int __XRight, int __YBottom, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __XLeft</code> X-Koordinate linke obere Ecke <code>int __YTop</code> Y-Koordinate linke obere Ecke <code>int __XRight</code> X-Koordinate rechte untere Ecke <code>int __YBottom</code> Y-Koordinate rechte untere Ecke <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiel:

```
/* Zeichnet ein gefülltes Rechteck, dynamische Ebene, direkte Ausgabe */  
DrwRectFilled(20,35,80,40,0xB0,0);
```

3.2.13.7 Eine Linie zeichnen

Mit der Funktion `DrwLine()` kann eine Linie gezeichnet werden. Die Liniendicke ist ein Pixel.

Deklaration:	<code>void DrwLine(int __XLeft, int __YTop, int __XRight, int __YBottom, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __XLeft</code> X-Koordinate Startpunkt <code>int __YTop</code> Y-Koordinate Startpunkt <code>int __XRight</code> X-Koordinate Endpunkt <code>int __YBottom</code> Y-Koordinate Endpunkt <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiel:

```
/* Zeichnet eine Linie, statische Ebene, indirekte Ausgabe */  
DrwLine(20,20,80,30,0x00,0);
```

Handbuch Bediengeräte

3.2.13.8 Kreis zeichnen

Mit der Funktion `DrwCircle()` kann ein Kreis um einen Mittelpunkt mit Radius gezeichnet werden. Die Liniendicke ist ein Pixel, der Kreis ist nicht gefüllt.

Deklaration:	<code>void DrwCircle(int __XMid, int __YMid, int __Radius, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __XMid</code> X-Koordinate Mittelpunkt <code>int __YMid</code> Y-Koordinate Mittelpunkt <code>int __Radius</code> Radius in Pixeln <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiel:

```
/* Zeichnet einen Kreis, statische Ebene, indirekte Ausgabe */  
DrwCircle(120,32,10,0x00,0);
```

3.2.13.9 Gefüllten Kreis zeichnen

Mit der Funktion `DrwCircleFilled()` kann ein Kreis um einen Mittelpunkt mit Radius gezeichnet werden. Der Kreis wird in der angegebenen Farbe gefüllt.

Deklaration:	<code>void DrwCircleFilled(int __XMid, int __YMid, int __Radius, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __XMid</code> X-Koordinate Mittelpunkt <code>int __YMid</code> Y-Koordinate Mittelpunkt <code>int __Radius</code> Radius in Pixeln <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiel:

```
/* Zeichnet einen gefüllten Kreis, statische Ebene, indirekte Ausgabe, Farbe weiß */  
DrwCircleFilled(120,32,5,0x00,0xFFFFFFFF);
```

3.2.13.10 Gefüllte Ellipse zeichnen

Mit der Funktion `DrwEllipseFilled()` kann eine Ellipse um einen Mittelpunkt mit Radiusangaben gezeichnet werden. Die Ellipse wird in der angegebenen Farbe gefüllt.

Deklaration:	<code>void DrwEllipseFilled(int __XMid, int __YMid, int __XRadius, int __YRadius, int __Attribute, uint32 __Color)</code>
Parameter:	<code>int __XMid</code> X-Koordinate Mittelpunkt <code>int __YMid</code> Y-Koordinate Mittelpunkt <code>int __XRadius</code> X-Radius in Pixeln <code>int __YRadius</code> Y-Radius in Pixeln <code>int __Attribute</code> Attribut siehe Tabelle <code>uint32 __Color</code> Farbe
Rückgabewert	keiner

Beispiel:

```
/* Zeichnet eine Ellipse, dynamische Ebene, indirekte Ausgabe */  
DrwEllipseFilled(120,32,20,10,0x00,0);
```

Handbuch Bediengeräte

3.2.13.11 Text ausgeben

Mit der Funktion DrwText() kann eine Zeichenkette (String) an einem beliebigen Ort im Display (pixelgenau positionierbar) dargestellt werden.

Deklaration:	void DrwText (int __X, int __Y, char * __Buf, int __len, int __Attribute, int __Size, uint32 __Color)
Parameter:	int __X X-Koordinate Textanfang (erster Buchstabe links) int __Y Y-Koordinate Textanfang (erster Buchstabe oben) char * __Buf Zeiger auf erstes Zeichen im String int __len Stringlänge, Anzahl auszugebende Zeichen int __Attribute Attribut siehe Tabelle, Bits 4 und 5 werden ersetzt durch „Size“ int __Size Textgröße 0=8x6 Pixel 1=16x12 Pixel 2=32x24 Pixel uint32 __Color Farbe
Rückgabewert	keiner

Beispiel:

```
/* Textausgabe "Hello world", unterstrichen, in 16x12 Pixel-Schrift */  
char Ausgabe[11] = {'Hello world'};  
DrwText(3,10,Ausgabe,0x04,1,0);
```

3.2.13.12 Bereich im Display verschieben

Mit der Funktion MoveArea() kann ein bereits gezeichneter Bereich des Bildinhalts verschoben werden. Anwendung: z.B. wenn bei Kurven ein Verlauf erfolgen soll, ohne alte Werte zu speichern.

Deklaration:	void MoveArea (int __XLeftOld, int __YTopOld, int __Width, int __Height, int __XLeftNew, int __YTopNew, int __MoveMode)
Parameter:	int __XLeftOld X-Koordinate alte Position int __YTopOld Y-Koordinate alte Position int __Width Breite des zu verschiebenden Bereichs int __Height Höhe des zu verschiebenden Bereichs int __XLeftNew X-Koordinate neue Position int __YTopNew Y-Koordinate neue Position int __MoveMode Bit 7: 0=indirekt 1=direkt Bits 0-6: 0 = statische Ebene, nicht blinkend 1 = statische Ebene, blinkend 2 = dynamische Ebene, nicht blinkend 3 = dynamische Ebene, blinkend
Rückgabewert	keiner

Handbuch Bediengeräte

3.2.13.13 Display Hintergrundfarbe einstellen

Diese Funktion steht nur bei Bediengeräten mit Farbdisplay zur Verfügung. Die Farbe wird unmittelbar beim Aufruf der Funktion umgeschaltet und bleibt auch bei Bildumschaltungen erhalten.

Nach dem Einschalten bzw. nach einem Reset des Bediengerätes wird die Hintergrundfarbe auf weiß eingestellt.

Deklaration:	void SetDisplayBackgroundColor (uint32 __Color)
Parameter:	uint32 __Color Der Parameter ist ein 32-Bit Wert, bei dem die 24 niederwertigen Bits die Farbe darstellen. Bits 24-31 immer 0x00 Bits 23-16 (0x00xx0000UL) Rot-Anteil (0..255)<<16 Bits 15-8 (0x0000xx00UL) Grün-Anteil(0..255)<<8 Bits 7-0 (0x000000xxUL) Blau-Anteil (0..255) Hintergrundfarbe schwarz = 0x00000000UL Hintergrundfarbe weiß = 0x00FFFFFFUL
Rückgabewert	keiner

Beispiel:

```
long aColor;
```

```
/* Hintergrundfarbe auf rot setzen */  
aColor = 0x00FF0000UL;  
SetDisplayBackgroundColor(aColor);
```

Hinweis:

Befinden sich Grafiken im Bild, dann werden die weißen Anteile der Grafik (alle Pixel mit der Farbe 0x00FFFFFFUL) transparent dargestellt, haben also die Hintergrundfarbe !

Abhilfe: Laden Sie die Grafik in ein Bildbearbeitungsprogramm. Erzeugen Sie eine Füllfarbe, die fast weiß dargestellt wird (z.B. 0x00FEFFFF) und füllen Sie weißen Flächen mit dieser Farbe.

Handbuch Bediengeräte

3.2.13.14 Textfarbe und Hintergrund einstellen

Diese Funktion steht nur bei Bediengeräten mit Farbdisplay zur Verfügung. Die Farbe wird unmittelbar beim Aufruf der Funktion umgeschaltet und bleibt auch bei Bildumschaltungen erhalten. Nach dem Einschalten bzw. nach einem Reset des Bediengerätes wird die Hintergrundfarbe auf weiß eingestellt.

Deklaration:	void SetTextColors (long __FgColor , long __BkColor)
Parameter:	long __FgColor long __BkColor Die Parameter sind 32-Bit Werte, bei dem die 24 niederwertigen Bits die Farbe darstellen. Bits 24-31 immer 0x00 Bits 23-16 (0x00xx0000UL) Rot-Anteil (0..255)<<16 Bits 15-8 (0x0000xx00UL) Grün-Anteil(0..255)<<8 Bits 7-0 (0x000000xxUL) Blau-Anteil (0..255) Hintergrundfarbe schwarz = 0x00000000UL Hintergrundfarbe weiß = 0x00FFFFFFUL BkColor kann auch einzeln mit SetDisplayBackgroundColor() gesetzt werden. Die Transparenzfarbe bei Text ist automatisch die Hintergrundfarbe, bei Farbgrafiken immer weiß.
Rückgabewert	keiner

3.2.13.15 Zeigerinstrument initialisieren

Die Funktion DefineAnalogView(...) initialisiert ein virtuelles analoges Zeigerinstrument, das in Verbindung mit einer Bitmap-Datei für die Skala auf dem Display dargestellt wird. Damit können beispielsweise Drehzahlmesser, Volt- und Amperemeter, Druckanzeiger und vieles mehr erzeugt werden.

Der Initialisierungsfunktion werden eine Reihe Parameter übergeben, damit dem Zeigerinstrument zur Aktualisierung im entsprechenden Bild nur noch der Istwert übergeben werden muß.

Rufen Sie diese Funktion in der Schnittstellenfunktion KOP_Init() auf, oder erzeugen Sie das Handle dynamisch.

Farbige Zeiger werden in dieselbe Ebene wie die Farbgrafiken gezeichnet.

Beim Aktualisieren werden die Zeiger mit der Zeiger-Hintergrundfarbe gelöscht und dann in der Zeigerfarbe neu gezeichnet. Das heißt, die Zeiger müssen über einen einfarbigen (Farbgrafik-)Bereich geführt werden, da sonst Bildteile überschrieben werden.

Schwarze Zeiger können nicht überschreibend weiterhin wie gewohnt benutzt werden. Wird beim Anlegen des Instruments das Attribut-Bit "schnelles Zeichnen" gewählt (Bit 6, 0x40) so wird der schwarze Zeiger wie ein Farbzeiger verwendet.

Vorteil: schnellerer Bildaufbau, Nachteil: überschreibt Farbgrafiken.

Zu beachten: wird als Texthintergrund die Farbe schwarz angegeben, so ist diese natürlich transparent. Ein schwarzer Zeiger muss dann als "Farbzeiger" wie oben angegeben verwendet werden..

Deklaration:	int DefineAnalogView (int __XMid , int __YMid , int __rOuter , int __rInner , int __AngleStart , int __AngleEnd , long __ValueStart , long __ValueEnd , long __FgColor , long __BkColor , int __Thickness , int __Attribute)
--------------	--

Handbuch Bediengeräte

3.2.13.18 Direktes Zeichnen in einen Screen-Buffer

Die Funktion erlaubt es, sämtliche Betriebssystemzugriffe auf den Bildschirm abzuschalten und nur die C-Zeichenfunktionen anzuzeigen und damit ungestört auf dem Display zu zeichnen.

Wird Screen 1 angezeigt, kann man in Screen 2 unsichtbar zeichnen und hat dann beim Umschalten den Screen auf einen Schlag (und umgekehrt).

Es kann auch im angezeigten Screen gezeichnet werden.

Es können 2 unabhängige Screens angelegt werden:

- *Screen 0* schaltet in den normalen Modus
- *Screen 1* über Zeichenattribut 0
- *Screen 2* über Zeichenattribut 0x80

Zu beachten:

- *Wenn Screen 1 oder Screen 2 angewählt sind, wird keine Änderung des angezeigten Bildes mehr vorgenommen.*
- *Die Funktion GetCurrentPageShown bzw auch GetCurrentMessageShown liefern dann den Wert, den sie beim Aufruf der Funktion SetSpecialScreenMode gerade hatten.*
- *Der Bildschirm wird beim Umschalten nicht gelöscht.*
- *Bei Rückkehr auf ScreenMode 0 sollte eine Bildumschaltung erzwungen werden. !*

Deklaration:	void DrwSetSpecialScreenMode (int __ScreenNo)
Parameter:	int __ScreenNo Nummer des Screen von 0 bis 2
Rückgabewert	keiner

Beispiel:

```
DrwSetSpecialScreenMode(1);
```

3.2.13.19 Rechteckigen Bereich invertieren

Mit dieser Zeichenfunktion kann ein rechteckiger Bereich der entsprechenden Zeichenebene (abhängig vom Attribut) invertiert werden.!

Deklaration:	void DrwInvertRect (int __XLeft, int __YTop, int __XRight, int __YBottom, int __Attribute)
Parameter:	int __XLeft, int __YTop, int __XRight, int __YBottom, int __Attribute
Rückgabewert	keiner

3.2.14 Funktionen für den Touch-Screen

Über die folgenden Parameter können die Touch-Screen Werte eingelesen und verändert werden.

Wenn nicht anders angegeben, wird die Pixelposition zurückgegeben, wobei die Ecke links oben die (0;0)-Koordinate hat.

Handbuch Bediengeräte

3.2.14.1 X-Position der aktuellen Druckstelle lesen

Deklaration:	int Get_X_Move (void)
Parameter:	keine
Rückgabewert	int X-Position der aktuellen Druckstelle (-1 = nicht gedrückt)

3.2.14.2 Y-Position der aktuellen Druckstelle lesen

Deklaration:	int Get_Y_Move (void)
Parameter:	keine
Rückgabewert	int Y-Position der aktuellen Druckstelle (-1 = nicht gedrückt)

3.2.14.3 X-Position der ersten Druckstelle lesen

Deklaration:	int Get_X_Down (void)
Parameter:	keine
Rückgabewert	int X-Position der ersten Druckposition

3.2.14.4 Y-Position der ersten Druckstelle lesen

Deklaration:	int Get_Y_Down (void)
Parameter:	keine
Rückgabewert	int Y-Position der ersten Druckposition

3.2.14.5 X-Position absoluter Analogwert lesen

Deklaration:	int Get_X_Value (void)
Parameter:	keine
Rückgabewert	int absoluter Analogwert X-Position

3.2.14.6 Y-Position absoluter Analogwert lesen

Deklaration:	int Get_Y_Value (void)
Parameter:	keine
Rückgabewert	int absoluter Analogwert Y-Position

Handbuch Bediengeräte

3.2.14.7 Kalibrierungswert für X lesen

Deklaration:	int Get_X_Cali (void)
Parameter:	keine
Rückgabewert	int Kalibrierungswert für X

3.2.14.8 Kalibrierungswert für Y lesen

Deklaration:	int Get_Y_Cali (void)
Parameter:	keine
Rückgabewert	int Kalibrierungswert für Y

3.2.14.9 Toleranzwert lesen

Deklaration:	int GetTouchTol (void)
Parameter:	keine
Rückgabewert	int Toleranzwert

3.2.14.10 Toleranzwert setzen

Deklaration:	void SetTouchTol (int __Value)
Parameter:	int __Value Toleranzwert
Rückgabewert	keiner

Handbuch Bediengeräte

3.2.15 Funktionen für Flankenauswertung

Es gibt viele Funktionen, bei denen eine Flankenauswertung erfolgen muß. Dem tragen wir Rechnung und bieten die folgenden Funktionen an. Jede Abfrage muß eine eindeutige Nummer "EdgeNo" erhalten, die von 0 - 2047 gehen darf.

3.2.15.1 Auswertung beider Flanken

Mit diesem Unterprogramm wird ein Eingangswert auf eine Flanke hin untersucht. Es spielt dabei keine Rolle, ob es eine steigende oder fallende Flanke ist. Logischerweise ist diese Flanke ein einziges Mal aktiv.

Deklaration:	int AnyEdge (int __Signal, int __EdgeNo)
Parameter	int __Signal abzufragendes Signal int __EdgeNo Eine eindeutige Nummer (0..2047) für die Abfrage
Rückgabewert	int 0=keine Flanke im Signalvergleich gegenüber letztem Funktionsaufruf <>0=Signal hat sich gegenüber letztem Funktionsaufruf geändert

Beispiel:

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

3.2.15.2 Auswertung der steigenden Flanke

Mit diesem Unterprogramm wird ein Eingangswert auf eine steigende Flanke hin untersucht. Logischerweise ist diese Flanke jeweils für einen Durchlauf lang aktiv.

Deklaration:	int RisingEdge (int __Signal, int __EdgeNo)
Parameter	int __Signal abzufragendes Signal int __EdgeNo Eine eindeutige Nummer (0..2047) für die Abfrage
Rückgabewert	int 0=keine Flanke im Signalvergleich gegenüber letztem Funktionsaufruf <>0=Signal hat sich gegenüber letztem Funktionsaufruf von 0 nach 1 geändert

Beispiel:

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

3.2.15.3 Auswertung der fallenden Flanke

Mit diesem Unterprogramm wird ein Eingangswert auf eine fallende Flanke hin untersucht. Logischerweise ist diese Flanke jeweils für einen Durchlauf lang aktiv.

Deklaration:	int FallingEdge (int __Signal, int __EdgeNo)
Parameter	int __Signal abzufragendes Signal int __EdgeNo Eine eindeutige Nummer (0..2047) für die Abfrage
Rückgabewert	int 0=keine Flanke im Signalvergleich gegenüber letztem Funktionsaufruf <>0=Signal hat sich gegenüber letztem Funktionsaufruf von 1 nach 0 geändert

Beispiel:

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

Handbuch Bediengeräte

3.2.16 Funktionen für den Video-Eingang

Diese Funktionen dienen dazu, das Bildsignal eines Videoeingangs (abhängig vom Gerätetyp !) nach Bedarf im Display sichtbar zu machen.

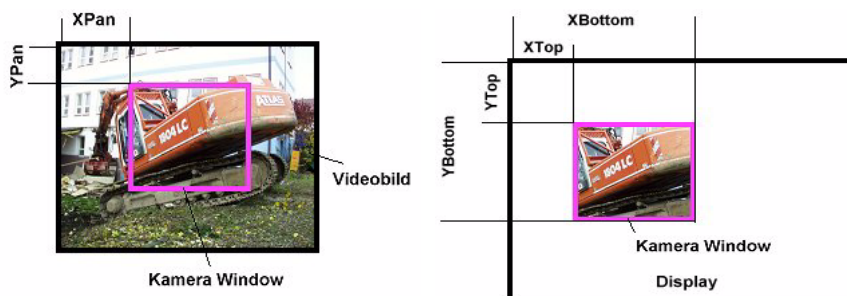
Als Anwendung in Fahrzeugen könnte beispielsweise eine Rückfahrkamera als Videoquelle dienen.

3.2.16.1 Videobild definieren

Die Funktion `CameraWindowSet(...)` bereitet das vom Video-Eingang kommende Bild mit der in den Parametern angegebenen Größe und Bildausschnittsbereich für das Display vor. Das Bild erscheint aber noch nicht !

Deklaration:	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
Rückgabewert	keiner

Die nachfolgende Grafik zeigt, wie die Einstellung der Parameter vorgenommen wird. Die __Pan Parameter geben den Punkt aus dem Videobild an, der am linken oberen Fensterpunkt (XTop, YTop) erscheinen soll..



3.2.16.2 Videobild einschalten

Mit der Funktion `CameraWindowOn` blenden Sie das Videobild in der vorab definierten Größe im Display ein.

Deklaration:	void CameraWindowOn (int __WindowPrio)
Parameter:	int __WindowPrio Ebene (Layer) in die das Videobild eingeblendet wird
Rückgabewert	keiner

3.2.16.3 Videobild ausschalten

Mit der Funktion `CameraWindowOff` wird das Videobild ausgeschaltet.

Deklaration:	void CameraWindowOff (void)
Parameter:	keine
Rückgabewert	keiner

Handbuch Bediengeräte

3.2.17 Sonstige Funktionen

3.2.17.1 Aktuellen Gerätestatus abfragen

Die Funktion `GetCurrentDeviceStatus()` liefert Ihnen den Gerätezustand als Information zurück. In welchem Status das Gerät sein kann, finden Sie im Handbuch „Kommunikation“, Kapitel Report Status Telegramm, Feld Bediengerätstatus. Diese Werte stimmen mit den Rückgabewerten dieser Funktion überein.

Deklaration:	uint16 GetCurrentDeviceStatus (void)
Parameter:	keine
Rückgabewert	int Gerätestatus

Beispiel:

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

3.2.17.2 Fehlerstatus abfragen

Die Funktion `Get_ERROR_Status()` liefert den Fehlerstatus des Gerätes zurück.

Deklaration:	uint16 Get_ERROR_Status (void)
Parameter:	keine
Rückgabewert	uint16 Fehlerstatus

Beispiel:

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

3.2.17.3 Zeitzone abfragen

Die Funktion `GetCurrentTimezone()` gibt an, welche Zeitzone gewählt ist:

Deklaration:	uint16 GetCurrentTimezone (void)
Parameter:	keine
Rückgabewert	int 0 = Winterzeit 1 = Sommerzeit 2 = Keine Zeitzone

Beispiel:

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

Handbuch Bediengeräte

3.2.17.4 Zeitzone setzen

Die Funktion `SetCurrentTimezone(...)` gibt an, welche Zeitzone gewählt werden soll:

Deklaration:	<code>void SetCurrentTimezone(uint16 __tz)</code>
Parameter:	<code>__tz</code> : 0 = Winterzeit 1 = Sommerzeit 2 = Keine Zeitzone
Rückgabewert	keiner

Wenn von Sommer- auf Winterzeit umgestellt wird, wird automatisch die Echtzeituhr des Geräts um eine Stunde vorgestellt. Umgekehrt wird bei der Umstellung von Winter- auf Sommerzeit die Uhr eine Stunde zurückgestellt. Wenn Winterzeit eingestellt ist, und es wird auf Winterzeit geschaltet, so erfolgt keine Umstellung der Uhr, ebenso bei Sommerzeit.

Beispiel:

```
SetCurrentTimezone(1); /* setzt Sommerzeit */
```

3.2.17.5 Konfiguration der Druckerschnittstelle lesen

Liefert zurück, welcher Drucker eingestellt ist

Deklaration:	<code>int GetPrinterInterfaceIndex(void)</code>
Parameter:	keiner
Rückgabewert	<code>int</code> 0 = kein Drucker 1 = Drucker seriell 2 = Drucker an CAN-Modul mit SELECAN-ID 3 = Druckdaten als ASCII-Telegramm auf eigener ID

3.2.17.6 Druckerschnittstelle konfigurieren

Mit dieser Funktion kann die Druckerschnittstelle angegeben werden.

Deklaration:	<code>void SetPrinterInterfaceIndex(int __Index)</code>
Parameter:	<code>__Index</code> : 0 = kein Drucker 1 = Drucker seriell 2 = Drucker an CAN-Modul mit SELECAN-ID 3 = Druckdaten als ASCII-Telegramm auf eigener ID
Rückgabewert	keiner

0 = kein Drucker
1 = Drucker seriell
2 = Drucker an CAN-Modul mit SELECAN-ID
3 = Druckdaten als ASCII-Telegramm auf eigener ID

Übernahme:

sofort. Nach Neustart aber nur, wenn `SaveToFlash` ausgeführt wurde.

Handbuch Bediengeräte

3.2.17.11 Initialisierungsflags lesen

Eine genaue Beschreibung folgt.

Deklaration:	uint16 GetInitFlags (void)
Parameter:	keine
Rückgabewert	uint16 Initialisierungsflags

3.2.17.12 Initialisierungsflags schreiben

Eine genaue Beschreibung folgt

Deklaration:	void SetInitFlags (uint16 __Bits)
Parameter:	uint16 __Bits :Flags für die Initialisierung
Rückgabewert	keiner

3.2.17.13 Setup-Daten in das Projekt-FLASH speichern

Viele der Konfigurationsdaten des Projekts können geändert werden. Einige der Daten werden sofort, andere erst nach dem Speichern im FLASH und nach einem Neustart des Geräts angenommen.

Nachdem SETUP-Daten geändert wurden (CAN Busrate, Identifier usw.) müssen diese im FLASH gespeichert werden, wenn sie nach dem Wiedereinschalten dieselben Werte haben sollen. Mit dieser Funktion wird das erledigt. Es wird immer der gesamte Datensatz (alle Daten) gespeichert, daher hat die Funktion keine Parameter.

Nach der Ausführung von SaveToFlash() ist, je nach dem was an Konfigurationsdaten geändert wurde, ein Neustart des Gerätes notwendig. Führen Sie dazu die Funktion **ExecCANTelegramInternal** mit D0=0x12 aus (Siehe auch Handbuch Kommunikation).

Deklaration:	void SaveToFlash (void)
Parameter:	keine
Rückgabewert	keiner

3.2.17.14 Gerätefunktion über CAN-Aufrufkonvention aufrufen

Dies ist im Gerätebetriebssystem die wohl wichtigste Funktion. Nicht alle möglichen Gerätefunktionen sind als eigene Unterprogramme verfügbar. Diese Funktion erlaubt, jede über die Schnittstelle aufrufbare Funktion auch geräteintern auszulösen. Es müssen lediglich die Parameter D0 bis D7 mit den entsprechenden Daten ausgefüllt werden (siehe Handbuch Kommunikation).

Damit sind dann auch Gerätefunktionen zugänglich, die in der System-API nicht explizit ausprogrammiert wurden.

Deklaration:	void ExecCANTelegramInternal (char __D0 , char __D1 , char __D2 , char __D3 , char __D4 , char __D5 , char __D6 , char __D7)
Parameter:	char __D0 : Datenbyte D0 char __D1 : Datenbyte D1 char __D2 : Datenbyte D2 char __D3 : Datenbyte D3 char __D4 : Datenbyte D4 char __D5 : Datenbyte D5 char __D6 : Datenbyte D6 char __D7 : Datenbyte D7
Rückgabewert	keiner

Handbuch Bediengeräte

Beispiel:

```
ExecCANTelegramInternal(0x12,0,0,0,0,0,0,0); /* Geräte-Reset */
```

3.2.17.15 Buzzer Ein/Ausschalten

Bei Geräten mit akustischem Signalgeber (Buzzer) kann mit dieser Funktion das Verhalten des Buzzers programmiert werden

Deklaration:	void SetBuzzer (uint8 __Mode, long __BuzzerOnTime, long BuzzerOffTime, long __Count)
Parameter:	<p>__Mode: 0=Buzzer aus, alle anderen Parameter werden ignoriert 1=Buzzer ein, leise 3=Buzzer ein, laut</p> <p>__BuzzerOnTime: steuert, wie lange (in 10ms-Schritten) der Buzzer eingeschaltet bleibt.</p> <p>__BuzzerOffTime: steuert, wie lange (in 10ms-Schritten) der Buzzer eingeschaltet bleibt.</p> <p>__Count: Wie oft eine Ein-Ausschaltperiode ausgeführt wird. 0=endlos</p>
Rückgabewert	keiner

Beispiele

Ist Mode gleich 0 wird der Buzzer ausgeschaltet	SetBuzzer(0,0,0,0)	Buzzer aus
Ist Mode ungleich 0 und sind BuzzerOnTime und BuzzerOffTime beide gleich 0, dann wird der Buzzer eingeschaltet	SetBuzzer(3,0,0,0)	Buzzer ein, laut
Ist Mode ungleich und ist nur BuzzerOnTime ungleich 0, dann wird der Buzzer für genau diese Zeit eingeschaltet (Single-Shot)	SetBuzzer(1,100,0,0)	Buzzer 1 Sekunde ein leise
Ist Mode ungleich und sind BuzzerOnTime und BuzzerOffTime beide ungleich 0, dann wird der Buzzer periodisch angesteuert	SetBuzzer(3,30,20,0)	300ms ein, 200ms aus fortlaufend, laut
	SetBuzzer(3,30,20,5)	300ms ein, 200ms aus 5 mal, laut

3.2.17.16 Buzzer Status abfragen

Der aktuelle Status des Buzzers kann mit dieser Funktion abgefragt werden. Der Rückgabewert ist Bitcodiert. Bit 0 = Ein/Aus, Bit 1 = Leise/Laut, Bit 7 = nicht periodisch/periodisch

Deklaration:	uint8 GetBuzzer (void)
Parameter:	keine
Rückgabewert	<p>0x00 Buzzer ist ausgeschaltet</p> <p>0x01 Buzzer ist eingeschaltet, leise</p> <p>0x03 Buzzer ist eingeschaltet, laut</p> <p>0x80 Buzzer ist gerade aus, wird aber periodisch angesteuert</p> <p>0x81 Buzzer ist eingeschaltet, leise, wird aber periodisch angesteuert</p> <p>0x83 Buzzer ist eingeschaltet, laut, wird aber periodisch angesteuert</p>

Handbuch Bediengeräte

3.2.18 Flash-Funktionen

Für den Anwender steht ein Teil des FLASH-Speichers zur Verfügung. Es wird ein Pseudo-Byte-Array mit 32000 Bytes transparent zur Verfügung gestellt.

Voraussetzung für die Nutzung der FLASH-Funktionen:

- Es muß ein Gerät mit Echtzeituhr sein
- Das Anwenderprojekt darf eine bestimmte Größe nicht überschreiten.

3.2.18.1 Testen ob Daten ins Flash gespeichert werden können.

Die Funktion FlashTest() ermittelt, ob die Voraussetzungen gegeben sind. Dann kann mit den Funktionen FlashRead bzw. FlashWrite aus dem Flash gelesen bzw. in den Flash geschrieben werden. Der Anwender muß selbst sicher stellen, dass die Parameter sinnvoll sind. Insbesondere muss er für genügend Speicher bei *__pDest sorgen. (Mindestens __Len Bytes).

Deklaration:	int FlashTest (void)
Parameter:	keiner
Rückgabewert	int <>0: Daten können im Flash gespeichert werden =0: Daten können nicht im Flash gespeichert werden

Beispiel siehe unten.

3.2.18.2 Daten aus dem Anwender-Flash lesen

Deklaration:	void FlashRead (uint8 * __pDest, long __Index, long __Len)
Parameter:	uint8 * __pDest Zeiger auf den Bereich, wohin der Anwender die Daten aus dem Flash haben möchte long __Index Index des ersten zu lesenden Bytes im Pseudo-Byte-Array (Flash) long __Len Anzahl der aus dem Flash zu lesenden Bytes
Rückgabewert	

Beispiel siehe unten.

Handbuch Bediengeräte

3.2.18.3 Daten in das Anwender-Flash schreiben

Der Flashing-Prozess verläuft für den Anwender transparent. Die Funktion FlashWrite ermittelt selbstständig die erforderlichen Schritte wie z.B. Daten zwischenspeichern, Flash löschen und neu brennen. Der Anwender muß nur den Aufruf FlashWrite tätigen.

Es wird ein Flash eingesetzt, das ca. 100000 Schreibzyklen unterstützt. Diese Funktionen sollte man also nur zum Speichern von Parameterdaten usw. benutzen, nicht aber für sich ständig ändernde Daten. Der Anwender muß selbst sicher stellen, dass die Parameter sinnvoll sind. Insbesondere muss er prüfen, ob `__Index+__Len < 32000`.

Deklaration:	void FlashWrite (long __Index , uint8 * __pSource , long __Len)	
Parameter:	long __Index	Index des ersten zu schreibenden Bytes im Pseudo-Byte-Array (Flash)
	uint8 * __pSource	Zeiger auf den Bereich, wo die zu speichernden Daten stehen
	long __Len	Anzahl der ins Flash zu schreibenden Bytes
Rückgabewert	keiner	

Beispiel:

Dieses Codesegment würde 40 Bytes ab dem Index 100 auf 200 kopieren:

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

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