SIEMENS

SIMATIC

Systemsoftware für M7-300/400 System- und Standardfunktionen, Band 1

Referenzhandbuch
Dieses Handbuch ist Bestandteil des Doku–mentationspaketes mit der Bestellnummer:

6ES7802–0FA14–8AA0

C79000–G7000–C852–02

Sicherheitstechnische Hinweise

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise sind durch ein Warndreieck hervorgehoben und je nach Gefährdungsgrad folgendermaßen dargestellt:

Gefahr

bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **werden**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Warnung

bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **können**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht

bedeutet, daß eine leichte Körperverletzung oder ein Sachschaden eintreten können, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Hinweis

ist eine wichtige Information über das Produkt, die Handhabung des Produktes oder den jeweiligen Teil der Dokumentation, auf den besonders aufmerksam gemacht werden soll.

Bestimmungsgemäßer Gebrauch

Beachten Sie folgendes:

Warnung

Das Gerät darf nur für die im Katalog und in der technischen Beschreibung vorgesehenen Einsatzfälle und nur in Verbindung mit von Siemens empfohlenen bzw. zugelassenen Fremdgeräten und -Komponenten verwendet werden.

Der einwandfreie und sichere Betrieb des Produktes setzt sachgemäßen Transport. sachgemäße Lagerung, Aufstellung und Montage sowie sorgfältige Bedienung und Instandhaltung voraus.

Warenzeichen

SIMATIC®, SIMATIC HMI® und SIMATIC NET® sind eingetragene Warenzeichen der SIEMENS AG.

Die übrigen Bezeichnungen in dieser Schrift können Warenzeichen sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen können.

Siemens AG
Bereich Automatisierungs-und Antriebstechnik
Geschäftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D- 90327 Nürnberg

Haftungsausschluß

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so daß wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, und notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Siemens Aktiengesellschaft

Buch-Nr. C79000-G7000-C852

Vorwort

Zweck des Handbuchs

Dieses Handbuch unterstützt Sie bei der C–Programmierung der Automatisierungsrechnerfamilie M7 300 und M7 400 unter dem Betriebssystem M7 RMOS32. Es versorgt Sie mit detaillierten Funktionsumfang der Aufrufschnittstelle von M7 RMOS32. Sie erhalten Informationen über

- Notation und Datentypen
- Funktionelle Einteilung der verschiedenen Aufrufe
- Verwendete Datenstukturen
- Fehlercodes und Meldungen
- Detailinformationen zu den Funktionsaufrufen

Leserkreis

Dieses Handbuch wendet sich in erster Linie an C–Programmierer der Automatisierungrechner M7 300 und M7 400.

Gültigkeitsbereich des Handbuchs

Dieses Handbuch ist gültig für die Automatisierungsrechner M7 300 und M7 400 mit der Systemsoftware M7–SYS RT V 4.0.

Einordnung in die Informationslandschaft

Die Systemsoftware für Automatisierungsrechnern M7–300 und M7–400 mit M7 RMOS32 wird in mehreren Handbüchern dokumentiert, die unabhängig vom jeweiligen Produkt bestellt werden können. Die Handbücher sind in der folgenden Tabelle aufgelistet.

Handbuch	**Inhalt**
Systemsoftware für M7–300/400 Installieren und Bedienen, Benutzerhandbuch	Installation und Bedienung von Automatisierungsrechnern M7–300/400.
Systemsoftware für M7–300/400 Programmentwurf, Programmierhandbuch	Entwerfen und Erstellen von C/C++–Programmen.
Systemsoftware für M7–300/400 System– und Standardfunktionen, Referenzhandbuch	Detailinformationen zur Programmierung mit M7 RMOS32.
Systemsoftware für M7–300/400 Ladbare Treiber erstellen, elektronisches Handbuch M7LDRV1A.PDF	Entwerfen und Erstellen von ladbaren Treibern für M7 RMOS32, Programmieranleitung und Referenz.

Wegweiser

Dieses Referenzhandbuch unterstützt Sie vor allem beim Programmieren von Anwendungen für M7 RMOS32. Es ist das zentrale Nachschlagewerk beim Programmieren, Testen und kontrollieren des Quellcodes. Das Handbuch ist in zwei Bänd mit folgendem Inhalt aufgeteilt:

BAND 1:

Funktionsgruppen

Kapitel 1 gibt Ihnen einen Leitfaden und stellt die programmierbaren Funktionen im logischen Zusammenhang dar. Wenn Sie eine bestimmte Aufgabe lösen wollen und die passende Funktion suchen, können Sie hier die gesuchte Funktion identifizieren.

Für die einzelnen Gruppen von Aufrufen sind in diesem Kapitel auch die Randbedingungen für die Verwendung beschrieben. Eine detaillierte Beschreibung der einzelnen Funktionen finden Sie in Band 1, Kapitel 5 und 6 und in Band 2, Kapitel 1 bis 3.

Typkennzeichen

Das 2. Kapitel enthält die wichtigsten Typkennzeichen, die bei der Programmierung verwendet werden. Es werden die Kennungen für die Systemnachrichten, S7–Objekte und die benutzten Datentypen aufgelistet.

Datenstrukturen

Das 3. Kapitel beschreibt die Datenstrukturen, die bei den RMOS–API–, M7–API– und Socket–Aufrufen verwendet werden.

Fehlercodes und Meldungen

Das 4. Kapitel erläutert die Fehlercodes und Meldungen, die vom M7 RMOS32–Kernel bzw. von den einzelnen Funktionsaufrufen zurückgeliefert werden.

Beschreibung der Funktionsaufrufe

Kapitel 5, und 6 enthalten jeweils in alphabetischer Reihenfolge detaillierte Beschreibungen der M7–API– und RMOS–API–Aufrufe

BAND 2:

Bibliotheken

Kapitel 1, 2 und 3 enthalten jeweils in alphabetischer Reihenfolge detaillierte Beschreibungen der Aufrufe der C–Laufzeitbibliothek, der Socket-Schnittstelle und weiterer Aufrufe.

Index

Jeder Band enthält einen Gesamtindex, der Ihnen hilft, Textstellen zu wichtigen Stichwörtern schnell zu finden.

Handbuch und Online–Hilfe

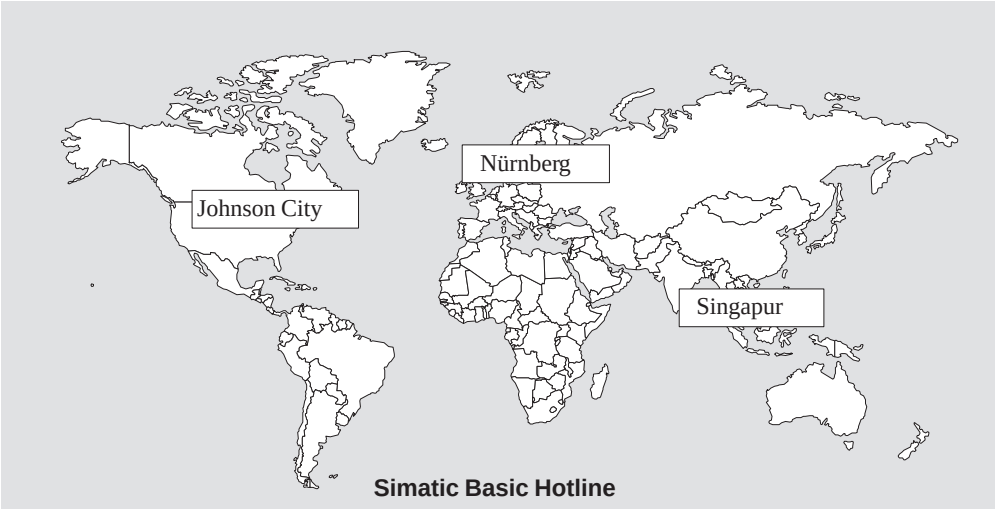
Das vorliegende Handbuch ist sowohl auf Papier als auch in elektronischer Form als Bestandteil der SIMATIC Standsammlung erhältlich. Zudem liegt der Inhalt auch als Online-Hilfedatei M7SYS40A.HLP im Verzeichnis S7BIN von STEP 7 vor. Diese Datei können Sie in den Suchbereich der Hilfe-Funktion OpenHelp der Borland-IDE einbinden, um kontextsensitive Unterstützung bei der Programmierung zu erhalten.

Rückmeldung zur Dokumentation

Um Ihnen und zukünftigen M7-SYS RT Anwendern eine optimale Dokumentation anbieten zu können, bitten wir Sie, uns hierbei zu unterstützen. Bei Anmerkungen zum vorliegenden *Handbuch* oder zur *Online–Hilfe* füllen Sie bitte den Fragebogen am Ende des Handbuchs aus und schicken Sie ihn an die dort angegebene Adresse. Bitte tragen Sie dort auch Ihre persönliche Bewertung ein.

SIMATIC Customer Support Hotline

Weltweit erreichbar zu jeder Tageszeit:



Simatic Basic Hotline

Nürnberg

SIMATIC BASIC Hotline

Ortszeit: Mo.-Fr. 8:00 bis 18:00
Telefon: +49 (911) 895-7000
Fax: +49 (911) 895-7002
E-Mail: simatic.support@nbgm.siemens.de

SIMATIC Premium Hotline

(kostenpflichtig, nur mit SIMATIC Card)
Zeit: Mo.-Fr. 0:00 bis 24:00
Telefon: +49 (911) 895-7777
Fax: +49 (911) 895-7001

Johnson City

SIMATIC BASIC Hotline

Ortszeit: Mo.-Fr. 8:00 bis 17:00
Telefon: +1 423 461-2522
Fax: +1 423 461-2231
E-Mail: simatic.hotline@sea.siemens.com

Singapur

SIMATIC BASIC Hotline

Ortszeit: Mo.-Fr. 8:30 bis 17:30
Telefon: +65 740-7000
Fax: +65 740-7001
E-Mail: simatic@singnet.com.sg

SIMATIC Customer Support Online-Dienste

Das SIMATIC Customer Support bietet Ihnen über die Online–Dienste umfangreiche zusätzliche Informationen zu den SIMATIC–Produkten:

- Allgemeine aktuelle Informationen erhalten Sie
  - im **Internet** unter `http://www.ad.siemens.de/simatic`
  - über **Fax-Polling** Nr. 08765-93 02 77 95 00
- Aktuelle Produkt–Informationen und Downloads, die beim Einsatz nützlich sein können:
  - im **Internet** unter `http://www.ad.siemens.de/support/html–00/`
  - über das **Bulletin Board System** (BBS) in Nürnberg (*SIMATIC Customer Support Mailbox)* unter der Nummer +49 (911) 895-7100.

    Verwenden Sie zur Anwahl der Mailbox ein Modem mit bis zu V.34 (28,8 kBaud), dessen Parameter Sie wie folgt einstellen: 8, N, 1, ANSI, oder wählen Sie sich per ISDN (x.75, 64 kBit) ein.

SIMATIC Trainingcenter

Um Ihnen den Einstieg in die Automatisierungssysteme SIMATIC S7 und M7 zu erleichtern, bieten wir entsprechende Kurse an. Wenden Sie sich bitte an Ihr regionales Trainingscenter oder an das zentrale Trainingscenter in D-90327 Nürnberg. Tel. 0911/895 3154.

Weitere Unterstützung

Bei weiteren Fragen zu den SIMATIC Produkten wenden Sie sich bitte an Ihre Siemens-Ansprechpartner in den für Sie zuständigen Vertretungen und Geschäftsstellen. Die Adressen finden Sie in Katalogen und in Compuserve (`go autforum`).

Inhaltsverzeichnis

Tabellen

Funktionsgruppen

1

Kapitelübersicht

1.1 Überblick

Was beschreibt dieses Kapitel?

Dieses Kapitel beschreibt alle zur Verfügung stehenden Funktionsaufrufe. Die einzelnen Aufrufe sind in logische Funktionsgruppen untergliedert.

Übersicht: Bibliotheken und Header–Dateien

Wenn Funktionen einer Gruppe in M7 RMOS32–Tasks verwendet werden sollen, ist bei der Programmerstellung die zugehörige Header–Datei zu inkludieren und beim Binden die entsprechende Bibliothek zur Verfügung zu stellen:

Tabelle 1-1 Übersicht der Funktionsgruppen

Funktionsgruppe	**Header–Datei**	**Bibliothek**
RMOS–API–Funktionen	RMAPI.H	RMFHLI.LIB
M7–API–Funktionen	M7API.H	M7APIBL.LIB
MS–DOS–Schnittstellenfunktionen	RM3DOS.H	RMFDOSIB.LIB
C–Bibliotheksfunktionen	nach ANSI	RMFCRIFB.LIB
Funktionen der Socket–Schnittstelle	SOCKET.H	RMFSK2IB.LIB
Funktionen für serielle Schnittstellen	SERIAL.H	RMFSER.LIB
Sonstige Funktionen	MISC86.H	RM3BCC.LIB

1.2 RMOS–API–Funktionen

1.2.1 Hinweise zu den RMOS-API-Funktionen

Allgemeine Hinweise

M7 RMOS32 bietet zum Zugriff auf Dienste des M7 RMOS32–Kernels eine reine Funktionsschnittstelle, wobei der Rückgabewert (Fehlercode) der Funktion Auskunft über Erfolg oder Mißerfolg der Funktionsausführung gibt. Bei speziellen Aufrufen werden auch Hinweise zurückgemeldet.

Als Header–Datei mit den Prototypes für das API wird **RMAPI.H** inkludiert. Bei der Erstellung von M7 RMOS32–Anwendungen in der integrierten Entwicklungsumgebung wird die Datei automatisch inkludiert. RMAPI.H inkludiert wiederum die Dateien **RMTYPES.H** (RMOS–API-spezifische Typdefinitionen) und **RMDEF.H** (Generelle Definitionen wie Fehlercodes usw.).

M7 RMOS32–Anwendungen werden im Speichermodell FLAT erstellt, d.h. alle Zeiger bestehen lediglich aus einem 32 Bit Offset.

Hinweis

M7 RMOS32–Anwendungen werden im Speichermodell FLAT erstellt, d.h. alle Zeiger bestehen lediglich aus einem 32 Bit Offset.

Im FLAT–Speichermodell gibt es keinen Speicherschutz für Adreßbereiche von Anwendertasks. Der Systembereich des M7 RMOS32–Kernels ist mit folgender Ausnahme gegen Überschreiben geschützt: wird einer Systemfunktion ein Pointer übergeben, wird die referenzierte Adresse überschrieben, auch wenn sie im Systembereich liegt.

Hinweise zur Programmierung in C

Die RMOS–API-Aufrufe werden anhand von Codebeispielen in C erläutert.

Die C-Schnittstelle wird durch die RMAPI.H im Verzeichnis INC beschrieben. Dort sind alle Funktionsprototypen des RMOS–API enthalten. Zusätzlich werden die Dateien RMDEF.H und RMTYPES.H inkludiert. In RMDEF.H sind die Define-Konstanten und in RMTYPES.H die Datentypen und Strukturen für die Programmierung der Systemaufrufe enthalten.

Um Schwierigkeiten durch Parameterfehler auszuschließen, sollten die definierten Konstanten aus RMDEF.H verwendet werden.

Die Parameter werden immer über den Stack übergeben, der Rückgabewert enthält den Fehlercode des RMOS–API–Aufrufs. Tritt kein Fehler auf, wird RM_OK (= 0) zurückgegeben. Im Fehlerfall erhält man einen Wert größer 0. Bestimmte RMOS–API–Aufrufe haben auch negative Rückgabewerte; diese dienen als zusätzliche Hinweise. Beispielsweise liefert `RmSetFlag` RM_FLAG_ALREADY_SET wenn das Flag bereits gesetzt war.

Beispiel eines RMOS-API-Aufrufs

Dieser Aufruf allokiert einen Speicherbereich von 1000 Byte, der nicht automatisch freigegeben wird; er wird somit nicht taskspezifisch zugeordnet. Ist nicht genügend freier Speicher vorhanden, wird nicht auf die Zuteilung gewartet.

```
main()
  {
     int        Error;
     void       *Pointer;
     ...
     Error = RmAlloc( RM_CONTINUE, RM_NOAUTOFREE, 1000ul, &Poin-
ter)
     ...
  }
```

Allgemeine Datentypen

Die folgende Datentypen können bei der Programmierung von RMOS–API–Aufrufen verwendet werden.

Tabelle 1-2 Allgemeine Datentypen von C

Datentyp	Bedeutung
char	Character : 8 Bit
short	Ganzzahl: 16 Bit
int	Ganzahl: 32 Bit
long	Ganzahl: 32 Bit
void *	Pointer (FLAT): 32 Bit
enum	Aufzählungstyp: 32 Bit
float	Gleitkommazahl: 32 Bit
double	Gleitkommazahl: 64 Bit

Neben den allgemeinen C–Datentypen sollten für RMOS–API–Aufrufe die RMOS–API–spezifischen Datentypen (vgl. Abschnitt 3-1) verwendet werden. Diese Datentypen sind in der Header–Datei RMTYPES.H definiert.

Interruptnummern

Bei allen RMOS–API–Aufrufen zum Abfragen, Installieren und Deinstallieren von Interrupthandlern kann die Interruptnummer auf zwei verschieden Arten angegeben werden:

1. Nummer zwischen 0 und 255
   Der Interrupt wird als Softwareinterrupt behandelt.
2. IRQ<n>
   Die Nummer <n> wird direkt eingegeben, z.B. IRQ1, IRQ2. Der Interrupt wird als Hardwareinterrupt behandelt.
   Die Werte IRQ1, IRQ2 usw. sind in einer Inkludedatei festgelegt. Mit dem Makro IRQ(x) kann die IRQ-Nummer in einer Variablen übergeben werden. Der Wert von (x) kann zwischen 0 und dem maximal verfügbaren Interrupt liegen. Auf dem PC ist der Wertebereich 0..15 gültig.

Hinweise zur Timer–Programmierung

Arbeiten Sie in Ihrem Programm mit Timeout–Werten, werden diese entsprechend ihrer Ablaufzeit (geordnet nach Timerticks) in der Timerqueue eingetragen. Liegen für einen Timertick mehrere Timeout–Anforderungen vor, werden diese nach dem *Last In First Out* Prinzip abgearbeitet.

Liegt zwischen zwei gleich langen Timeout–Anforderungen ein Timertick, werden diese Anforderungen auf verschiedene Timerticks verteilt.

Beispiel:
Timeout–Anforderung 1, 2, 3 innerhalb eines Timerticks, Timeout–Anforderung 4, 5, 6 innerhalb des nächsten Timerticks. In der Timerqueue ergibt sich die Reihenfolge 3, 2, 1, 6, 5, 4

Damit gewährleistet werden kann, daß alle Timeraufrufe innerhalb eines Timerticks geschehen, muß wie folgt vorgegangen werden.

1. Priorität der Task sehr hoch einstellen (Höchste Systempriorität), damit sie nicht von anderen Tasks unterbrochen wird.
2. Pauseaufruf mit 0 zur Synchronisation auf den nächsten Timertick.
3. Timeout–Anforderungen absetzen.
4. Priorität der Task auf Ausgangswert zurücksetzen.

Dabei muß darauf geachtet werden, daß die gesamte Bearbeitung innerhalb eines Timerticks abgeschlossen sein muß.

1.2.2 Kurzbeschreibung der RMOS-API-Funktionen

Überblick

Das RMOS–API bietet in Form einer C–Schnittstelle den M7 RMOS32–Anwendungen alle notwendigen Funktionen zur Realisierung eines Multitasking–Systems. RMOS–API–Funktionen bilden die Schnittstelle zum M7 RMOS32– Kernel.

Eine detaillierte Beschreibung der Funktionen finden Sie im Kapitel 6.

Speicher-verwaltung

In folgender Tabelle sind alle Funktionen der Speicherverwaltung mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-3 Funktionen für der Speicherverwaltung

Funktion	Kurzbeschreibung
RmAlloc	Speicher aus dem Heap anfordern
RmCreateMemPool	Speicherpool aus dem Heap erzeugen
RmDeleteMemPool	Speicherpool löschen
RmFree	Speicherbereich freigeben
RmFreeAll	Sämtliche Speicherbereiche einer Task freigeben
RmGetMemPoolInfo	Information über Speicherpool einholen
RmGetSize	Länge eines Speicherbereichs ermitteln
RmMapMemory	Physikalischen Speicher einblenden
RmMemPoolAlloc	Speicherbereich aus Speicherpool anfordern
RmReAlloc	Speicherbereich vergrößern

Task–Steuerung

In folgender Tabelle sind alle Funktionen, die Sie für die Task–Steuerung anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-4 Funktionen für die Task–Steuerung

Funktion	Kurzbeschreibung
RmActivateTask	Task in Zustand BEREIT versetzen
RmCreateTask	Task erzeugen
RmCreateTaskEx	Task erzeugen
RmCreateChildTask	Kinder–Task erzeugen
RmDeleteTask	Aufrufende Task beenden (mit löschen)
RmDisableScheduler	Scheduler anhalten
RmEnableScheduler	Scheduler wieder starten
RmEndTask	Aufrufende Task beenden (ohne löschen)
RmGetTaskID	ID (Kennzeichen) einer Task ermitteln

Tabelle 1-4 Funktionen für die Task–Steuerung

Funktion	Kurzbeschreibung
RmGetTaskPriority	Priorität einer Task ermitteln
RmGetTaskState	Zustand einer Task ermitteln
RmKillTask	Task in Zustand RUHEND oder NICHT EXISTENT versetzen
RmPauseTask	Aufrufende Task verzögern
RmQueueStartTask	Task in Warteschlange einreihen. Die Task wird gestartet, sobald sie sich im Zustand RUHEND befindet.
RmRestartTask	Aufrufende Task beenden und nach einer gewissen Zeitspanne automatisch starten
RmResumeTask	Taskbearbeitung nach Ablauf eines mit `RmPauseTask` eingeleiteten Intervalls fortsetzen
RmSetTaskPriority	Priorität einer Task ändern
RmStartTask	Task-Start-Anforderung für Tasks, die sich im Zustand RUHEND befinden.
RmSuspendTask	Task in Zustand WARTEND setzen

Zeit und Betriebsmittel verwalten

In folgender Tabelle sind alle Funktionen, die Sie zur Verwaltung von Betriebsmitteln (BM) anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-5 Funktionen für die Zeitverwaltung und Katalogisierung von Betriebsmitteln (BM)

Funktion	Kurzbeschreibung
RmCatalog	Eintragen von Betriebsmitteln im BM–Katalog
RmGetEntry	Eintrag (ID) im BM–Katalog ermitteln
RmGetName	Eintrag (Name) im BM–Katalog ermitteln
RmList	Auflisten von Einträgen des BM–Katalogs
RmUncatalog	Löschen von Einträgen im BM–Katalog
RmGetAbsTime	Absolute Systemzeit abfragen

Nachrichtenaustausch

In folgender Tabelle sind alle Funktionen, die Sie für den Nachrichtenaustausch anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-6 Funktionen für den Nachrichtenaustausch

Funktion	Kurzbeschreibung
RmCreateMessageQueue	Nachrichtenwarteschlangeerzeugen
RmDeleteMessageQueue	Nachrichtenwarteschlangelöschen
RmReadMessage	Nachricht aus Nachrichtenwarteschlange lesen
RmSendMessage	Nachricht in Nachrichtenwarteschlange ablegen
RmSetMessageQueueSize	Länge der Nachrichtenwarteschlange begrenzen

Mailboxen

In folgender Tabelle sind alle Funktionen, die Sie für den Nachrichtenaustausch über Mailboxen anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-7 Funktionen für den Nachrichtenaustausch mit Mailboxen

Funktion	Kurzbeschreibung
RmCreateMailbox	Mailbox erzeugen
RmDeleteMailbox	Mailbox löschen
RmReceiveMail	Nachricht aus Mailbox lesen
RmSendMail	Nachricht in Mailbox ablegen
RmSendMailCancel	Verzögertes Nachrichtenablegen abbrechen
RmSendMailDelayed	Nachricht in Mailbox verzögert ablegen
RmSetMailboxSize	Länge der Mailbox begrenzen

Ereignisflags

In folgender Tabelle sind alle Funktionen, die Sie für die Koordination mit Hilfe von Ereignisflags anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-8 Funktionen für die Koordination mit Ereignisflags

Funktion	Kurzbeschreibung
RmCreateFlagGrp	Flaggruppe erzeugen
RmDeleteFlagGrp	Flaggruppe löschen
RmGetFlag	Bit in Flaggruppe testen
RmResetFlag	Bit in Flaggruppe zurücksetzen
RmSetFlag	Bit in Flaggruppe setzen
RmSetFlagDelayed	Bit in Flaggruppe nach Intervall setzen

Semaphor-bearbeitung

In folgender Tabelle sind alle Funktionen, die Sie für die Bearbeitung von Semaphoren anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-9 Funktionen für Semaphorbearbeitung

Funktion	Kurzbeschreibung
RmCreateBinSemaphore	Semaphor erzeugen
RmDeleteBinSemaphore	Semaphor löschen
RmGetBinSemaphore	Semaphor belegen
RmReleaseBinSemaphore	Semaphor freigeben

Interrupt-bearbeitung

In folgender Tabelle sind alle Funktionen der Interruptbearbeitung mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-10 Funktionen der Interruptbearbeitung

Funktion	Kurzbeschreibung
RmGetIntHandler	Momentanen Interrupt–Handler ermitteln
RmSetIntDefHandler	Interrupt–Handler deinstallieren
RmSetIntISHandler	Interrupt–Handler für I– und S–Zustand installieren
RmSetIntMailboxHandler	Mailbox-Interrupt-Handler installieren
RmSetIntTaskHandler	Interrupt–Handler für Taskstart installieren

Ladbare Treiber

In folgender Tabelle sind alle Funktionen für ladbare Treiber mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-11 Funktionen für ladbare Treiber

Funktion	Kurzbeschreibung
RmIOClose	Unit schließen
RmIOControl	Steuerfunktionen für ladbare Treiber
RmIOOpen	Unit öffnen
RmIORead	Von Unit lesen
RmIOWrite	Auf Unit schreiben
RmLoadDevice	Treiber laden

Sonstige Aufrufe

In folgender Tabelle sind alle sonstigen RMOS–API–Aufrufe mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-12 Sonstige Funktionen

Funktion	Kurzbeschreibung
get2ndparm	Startparameter EBX der Task auslesen
getdword	Startparameter der Task als long auslesen
getparm	Startparameter der Task als Zeiger auslesen

1.2.3 RMOS-API-Aufrufe in MS-DOS-Anwendungen

Allgemeine Hinweise

RMOS bietet ebenfalls ein API, das von MS–DOS–Anwendungen genutzt werden kann. Somit können DOS–Anwendungen ebenfalls Systemaufrufe an den RMOS–Kernel absetzen (nicht an M7 Server!), um z.B. eine RMOS–Task zu starten oder Nachrichten in die Message Queue einer Task oder an eine Mailbox schicken.

Das RMOS–API für MS–DOS–Anwendungen ist nicht für zukünftige Entwicklungen vorgesehen!

Header–Dateien und Konventionen

MS–DOS–Programme, die die Schnittstelle verwenden, müssen die Prototypen der Header–Datei **RMAPI.H** inkludieren.

In MS–DOS–Programmen kann unter M7 RMOS32 nur die 16 Bit Real–Mode Aufrufschnittstelle verwendet werden. Datenformate, Typen und Strukturen sind konform zur Real–Mode–Programmierung von MS–DOS definiert.

C–Prototypes und Makros der RMOS–API–Schnittstelle sind in der Datei RMAPI.H bzw. in den Dateien **RMDEF.H** und **RMTYPES.H** definiert. Über den Schalter **RM3** wird ausgewählt, ob die Datei für M7 RMOS32 oder für MS–DOS–Anwendungen verwendet wird.

Inkludierung bei DOS-Programmen

In einem MS–DOS–Programm müssen deshalb die Schalter vor der Include–Anweisung für RMAPI.H nach folgendem Muster gesetzt werden:

```
#define RM3 0
#include "RMAPI.H"
```

Bibliotheken

Außerdem muß eine entsprechende Schnittstellenbibliothek mit in die Bindeanweisung für das Programm aufgenommen werden. Bei MS–DOS–Programmen ist dies die Bibliothek **DOSHLIB.LIB**.

Konvertierung von Daten

M7 RMOS32 wandelt intern bei einem RMOS–API–Aufruf aus DOS–Programmen die Parameter in das M7 RMOS32–Format um.

Datentypen, die eine Datenbreite von 16 Bit und bei M7 RMOS32 32 Bit haben, werden ”zero extended”, d.h Bit 31 ... Bit 16 werden auf 0 gesetzt, und an den RMOS–Kernel weitergeleitet.

Interrupt–Nummer

Ein RMOS-API–Aufruf wird aus einem MS–DOS–Programm über einen Software-Interrupt ausgelöst. Der dabei verwendete Interrupt-Vektor ist fest auf **79H** konfiguriert. Dieser Interrupt darf deshalb von MS–DOS–Anwendungen nicht anderweitig belegt werden..

Nicht unterstützte RMOS-API-Aufrufe

Die folgende Tabelle gibt eine Aufstellung der RMOS–API–Aufrufe, die in MS–DOS–Programmen nicht verwendet werden dürfen. Werden Sie trotzdem eingesetzt, so kehrt der Aufruf mit Fehlermeldung zurück.

Tabelle 1-13 Nicht unterstützte RMOS–API–Aufrufe

RMOS–API–Aufrufe	Ursache
RmAlloc, RmMemPoolAlloc, RmFree, RmFreeAll, RmReAllocMem, RmMapMemory	Es ist nicht zulässig, daß RMOS einen Speicherpool innerhalb des von MS–DOS adressierten Speicherbereichsverwaltet. Jeder RMOS–Speicherpool liegt deshalb bei M7 RMOS32 immer oberhalb des von MS–DOS adressierbarenBereichs.
RmSetISHandler, RmSetIntTaskHandler, RmSetIntMailboxHandler, RmSetIntDefHandler	Mit Hilfe der RMOS–API–Aufrufe zur Interruptverwaltung werden Interruptvektoren in der RMOS–Umgebung gesetzt. Um in der MS–DOS–Umgebung einen Interruptvektor zu ändern oder einzutragen, müssen die unter MS–DOS vorhandenen Funktionen verwendet werden.
RmEndTask, RmRestartTask	Ein MS–DOS–Programm kann sich durch diese Aufruf nicht beenden.
RmCreateTask	Ein MS–DOS–Programm kann keine andere Task kreieren, da innerhalb des von MS–DOS verwalteten Speichers keine weitere Task angelegt werden kann.
RmReadMessage, RmSendMessage, RmCreateMessageQueue, RmDeleteMessageQueue,	Bei diesen Aufrufen werden betriebssystemspezifische Zeiger übergeben, die nicht automatisch konvertiert werden können. Statt dessen kann die Kommunikation mit Mailboxen durchgeführt werden (siehe Tabelle 1-7).

Besonderheiten bei RMOS-API-Aufrufen

Die folgende Tabelle zeigt, welche Besonderheiten bei RMOS–API–Aufrufen in einer MS–DOS–Umgebung zu berücksichtigen sind. Werden sie nicht beachtet, so führt dies zu einem Systemfehlverhalten.

Tabelle 1-14 Besonderheiten bei RMOS–API–Aufrufen

RMOS–API–Aufruf	Ursache
RmDeleteTask	Ein MS–DOS–Programm kann zwar mit diesem Aufruf eine andere RMOS–Task löschen, jedoch nicht sich selbst. Ein Aufruf mit Task_ID gleich RM_OWN_TASK ist nicht zulässig.
RmSetTaskPriority	Ein MS–DOS–Programm kann durch den Aufruf die Priorität einer anderen RMOS–Task ändern, jedoch nicht die eigene. Ein Aufruf mit Task_ID gleich RM_OWN_TASK ist nicht zulässig.

Kommunikation über Mailbox–Dienste

Bei der Kommunikation zwischen RMOS– und MS–DOS–Programmen über Mailboxen (siehe Tabelle 1-7) müssen Sie außerdem folgende Besonderheiten beachten:

Bei dem Mailbox–Aufruf `RmSendMail` wird der Inhalt eines ”3–Wort–Puffers” (Botschaft) transferiert. Dieser Puffer hat bei M7 RMOS32 eine Länge von 12 Bytes.

Wird der Aufruf `RmSendMail` aus einer RMOS–Task unter M7 RMOS32 abgesetzt, so wird auch intern ein 12 Byte großer Datenbereich in die adressierte Mailbox übertragen.

Liest nun ein MS–DOS–Programm die Botschaft aus der Mailbox, so werden ebenfalls 12 Byte in den Speicherbereich des MS–DOS–Programm übertragen. In einem MS–DOS–Programm müssen Sie deshalb dafür sorgen, das der ”3 Wort–Puffer” ebenfalls 12 Byte lang ist.

Ein Zeiger in einer Botschaft wird vom RMOS–Kernel nicht umgesetzt, d.h. ein Flat–Pointer (lineare Adresse unter M7 RMOS32) wird nicht in einen Real–Mode–Pointer (physikalische Adresse unter MS–DOS) konvertiert.

Hinweis

Wird innerhalb des MS–DOS–Programms ein blockierender RMOS–API–Aufruf abgesetzt, so wird die DOS–Task, d.h. die gesamte DOS–Maschine blockiert (Taskzustand: WARTEND).

1.3 M7-API-Funktionen

1.3.1 Hinweise zu den M7-API-Funktionen

Konventionen und Header–Dateien für M7 RMOS32–Anwendungen

Als Header–Datei für die Prototypen der Funktionen muß von den M7 RMOS32–Programmen die Header–Datei **M7API.H** inkludiert werden.

Desweiteren finden Sie in M7API.H alle Datentyp– und Strukturdefinitionen, sowie die Fehlercodes.

Allg. Datentypen des M7–APIs

Um Programme in der Zukunft leichter auf andere Systeme portieren zu können, werden auch in der M7–API–Umgebung anstelle maschinenspezifischer Datentypbezeichner wie *int* oder *long* eigene Typdefinitionen verwendet. Die entsprechenden Datentypen sind in der Header–Datei M7API.H definiert.

1.3.2 Kurzbeschreibung der M7-API-Funktionen

Überblick

Das M7–API bietet in Form einer C–Schnittstelle den M7 RMOS32–Anwendungen alle notwendigen Funktionen zur Lösung einer Automatisierungsaufgabe.

Neben dem Zugriff auf die Prozeßperipherie bietet das M7–API Funktionen für die Verwaltung von internen S7–Objekten, Aufrufe für die Kommunikation mit anderen Automatisierungskomponenten, sowie weitere Funktionen zur transparenten Einbindung Ihres M7–Automatisierungsrechners in ein S7 Automatisierungssystem.

Eine detaillierte Beschreibung dieser Funktionen finden Sie im Kapitel 5.

Initialisierung

In folgender Tabelle ist die Funktion für die taskspezifische Initialisierung des M7–API aufgelistet.

Tabelle 1-15 Funktion für die Initialisierung

Funktion	Kurzbeschreibung
M7InitAPI	M7–API initialisieren

Zugriff auf die Prozeßperipherie

In folgender Tabelle sind alle Funktionen, die Sie für den Zugriff auf die Prozeßperipherie anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-16 Funktionen für den Zugriff auf die Prozeßperipherie

Funktion	Kurzbeschreibung
M7ClearPI	Prozeßabbild rücksetzen
M7LoadBit	Bit aus Prozeßabbild laden
M7LoadByte	Byte aus Prozeßabbild laden
M7LoadDWord	Doppelwort aus Prozeßabbild laden
M7LoadDirect	Daten direkt von Peripheriebereich lesen
M7LoadDirectByte	Byte direkt von Peripherie lesen
M7LoadDirectDWord	Doppelwort direkt von Peripherie lesen
M7LoadDirectWord	Wort direkt von Peripherie lesen
M7LoadISAByte	Byte aus ISA–Bus–Peripherie lesen
M7LoadISADWord	Doppelwort aus ISA–Bus–Peripherie lesen
M7LoadISAWord	Wort aus ISA–Bus–Peripherie lesen
M7LoadPII	Aktualisieren des Prozeßabbildes der Eingänge
M7LoadRecord	Datensatz aus Signalbaugruppe lesen
M7LoadRecordEx	Datensatz aus Signalbaugruppe lesen
M7LoadWord	Wort aus Prozeßabbild laden
M7StoreBit	Bit im Prozeßabbild überschreiben
M7StoreByte	Byte im Prozeßabbild überschreiben
M7StoreDWord	Doppelwort im Prozeßabbild überschreiben
M7StoreDirect	Daten direkt zum Peripherierbereich übertragen
M7StoreDirectByte	Byte direkt in die Peripherie schreiben
M7StoreDirectDWord	Doppelwort direkt in die Peripherie schreiben
M7StoreDirectWord	Wort direkt in die Peripherie schreiben
M7StoreISAByte	Byte in ISA–Bus–Peripherie schreiben
M7StoreISAWord	Wort in ISA–Bus–Peripherie schreiben
M7StoreISADWord	Doppelwort in ISA–Bus–Peripherie schreiben
M7StorePIQ	Aktualisieren der Peripherie aus Prozeßabbild
M7StoreRecord	Datensatz zu einer Signalbaugruppe übertragen
M7StoreWord	Wort im Prozeßabbild überschreiben

FRB–Bearbeitung

In folgender Tabelle sind die Aufrufe zur allgemeinen Bearbeitung von FRBs (Function Request Block) aufgelistet.

Tabelle 1-17 Funktionen für die FRB–Bearbeitung

Funktion	Kurzbeschreibung
M7GetFRBErrCode	Errorcode aus FRB–Header ermitteln
M7GetFRBTag	Tag aus FRB–Header ermitteln
M7SetFRBTag	Tag in FRB–Header setzen

Alarmbearbeitung (Slave–Funktionen)

In folgender Tabelle sind alle Funktionen zum Senden von Alarmen bzw. zum Abfragen des Alarmbearbeitungsstatus mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-18 Funktionen zur Alarmverarbeitung (Slave–Funktionen)

Funktion	Kurzbeschreibung
M7GetDiagAlarmBusy	Status eines Diagnosealarms abfragen
M7GetIOAlarmBusy	Status eines Prozeßalarms abfragen
M7SendDiagAlarm	Diagnosealarm an CPU senden
M7SendIOAlarm	Prozeßalarm an CPU senden

Verwaltung von S7-Objekten

In folgender Tabelle sind alle Aufrufe, die Sie für die Verwaltung von S7-Objekten anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-19 Funktionen für die Verwaltung von S7-Objekten

Funktion	Kurzbeschreibung
M7CreateObject	S7-Objekt erzeugen
M7DeleteObject	S7-Objekt aus Arbeitsspeicher **und** "Permanentem Ladespeicher" löschen
M7GetFlags	Zugriffstyp auf S7–Objekt aus OBJFRB ermitteln
M7GetObjectInfo	Information über Datenstruktur eines S7-Objektes lesen
M7GetObjType	Typkennzeichen des S7–Objekts aus OBJFRB ermitteln
M7GetPart	Teilbereichsnummer des S7–Objekts aus OBJFRB ermitteln
M7LinkDataAccess	OBJFRB für Zugriff auf S7–Objekt anmelden
M7LocateObject	S7-Objekt im Arbeitsspeicher verschieben
M7Read	S7-Datenbereich lesen
M7ReadBit	Bit aus S7-Objekt lesen

Tabelle 1-19 Funktionen für die Verwaltung von S7-Objekten, Fortsetzung

Funktion	Kurzbeschreibung
M7ReadByte	Byte aus S7-Objekt lesen
M7ReadWord	Wort aus S7-Objekt lesen
M7ReadDWord	Doppelwort aus S7-Objekt lesen
M7ReadReal	Gleitpunktzahl aus S7-Objekt lesen
M7RelocateObject	S7-Objekt an Objekt–Server übergeben
M7RemoveObject	S7-Objekt aus "Festwert–" **oder** "Permanentem Ladespeicher" löschen
M7StoreObject	S7-Objekt im "Festwert–" bzw. "Permanenten Ladespeicher" speichern
M7UnLinkDataAccess	OBJFRB für Zugriff auf S7–Objekt abmelden
M7Write	Anwenderdaten in S7-Datenbereich kopieren
M7WriteBit	Bit in S7-Objekt überschreiben
M7WriteByte	Byte in S7-Objekt überschreiben
M7WriteWord	Wort in S7-Objekt überschreiben
M7WriteDWord	Doppelwort in S7-Objekt überschreiben
M7WriteReal	Gleitpunktzahl in S7-Objekt überschreiben

Aufruf der Callbackfunktion bei S7–Objektzugriff

In folgender Tabelle sind alle Aufrufe, die Sie für die Anmeldung von Callback–Funktionen und zur Auswertung der Zugriffsinformation innerhalb der Callback–Funktion anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-20 Aufrufe für die Verwaltung von Callback–Funktionen

Funktion	Kurzbeschreibung
M7GetCBBitOffset	Bit Offset aus CBFRB ermitteln
M7GetCBBuffer	Lese– bzw. Schreibpuffer aus CBFRB ermitteln
M7GetCBByteOffset	Byte Offset aus CBFRB ermitteln
M7GetCBCount	Anzahl der Elemente aus CBFRB ermitteln
M7GetCBDataType	Datentyp aus CBFRB ermitteln
M7GetCBFlags	Zugriffstyp aus CBFRB ermitteln
M7GetCBObjType	Typkennzeichen des S7–Objekts aus CBFRB ermitteln
M7GetCBPart	Teilbereichsnummer des S7–Objeks aus CBFRB ermitteln
M7LinkDataAccessCB	Callback–Funktion für S7–Objektzugriff anmelden
M7UnLinkDataAccessCB	Callback–Funktion für S7–Objektzugriff abmelden

Alarmbearbeitung (Master–Funktionen)

In folgender Tabelle sind alle Funktionen, die Sie für die Bearbeitung von Alarmen als Master anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-21 Funktionen für die Alarmbearbeitung

Funktion	Kurzbeschreibung
M7ConfirmDiagAlarm	Diagnosealarm quittieren
M7ConfirmIOAlarm	Prozeßalarm quittieren
M7ConfirmZSAlarm	Ziehen/Stecken–Alarm quittieren
M7DPNormDiagnose	DP–Normdiagnose einer DP–Station ermitteln
M7GetDiagAlarmAddr	Basisadresse der Baugruppe aus DIAGFRB ermitteln
M7GetDiagAlarmInfo	Alarminformation aus DIAGFRB ermitteln
M7GetDiagAlarmPType	I/O–Typ der Baugruppe aus DIAGFRB ermitteln
M7GetIOAlarmAddr	Basisadresse der Baugruppe aus IOFRB ermitteln
M7GetIOAlarmMask	Alarmmaske aus IOFRB ermitteln
M7GetIOAlarmState	Alarminformation aus IOFRB ermitteln
M7GetIOAlarmPType	I/O–Typ der Baugruppe aus IOFRB ermitteln
M7GetPIErrorAddr	Adresse des Prozeßabbilds mit Transferfehler ermitteln
M7GetPIErrorPIType	Typ des Prozeßabbilds mit Transferfehlerermitteln
M7GetZSAlarmAddr	Basisadresse der Baugruppe aus ZSFRB ermitteln
M7GetZSAlarmIdent	Identifikationskennung einer Baugruppe ermitteln
M7GetZSAlarmIMRBaddr	Basisadresse der IMR–Baugruppe, für die ein Ziehen/Stecken–Alarm angemeldet war, ermitteln
M7GetZSAlarmMode	Modus der Baugruppe aus ZSFRB ermitteln
M7GetZSAlarmPType	I/O–Typ der Baugruppe aus ZSFRB ermitteln
M7LinkDiagAlarm	Diagnosealarm zur Bearbeitung anmelden
M7LinkIOAlarm	Prozeßalarm zur Bearbeitung anmelden
M7LinkPIError	FRB für Prozeßabbildtransfer initialisieren
M7LinkZSAlarm	ZS–Alarm zur Bearbeitung anmelden
M7UnLinkDiagAlarm	Diagnosealarm von Bearbeitung abmelden
M7UnLinkIOAlarm	Prozeßalarm von Bearbeitung abmelden
M7UnLinkPIError	FRB für Prozeßabbildtransfer abmelden
M7UnlinkZSAlarm	ZS–Alarm von Bearbeitung abmelden

Zeitbearbeitung

In folgender Tabelle sind alle Funktionen, die Sie für die Zeitbearbeitung anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-22 Funktionen für die Zeitbearbeitung

Funktion	Kurzbeschreibung
M7ConfirmPeriodicTimer	Periodisches Zeitsignal quittieren
M7GetLostPeriods	Verlorene periodische Zeitnachrichten abfragen
M7GetPeriod	Vielfaches der Zeitbasis aus TFRB ermitteln
M7GetTime	Datum/Uhrzeit auslesen
M7GetTimeBase	Zeitbasis aus TFRB ermitteln
M7LinkDate	Uhrzeitgesteuerte Zeitnachricht anmelden
M7LinkOneShotTimer	Singuläre Zeitnachricht anmelden
M7LinkPeriodicTimer	Periodische Zeitnachricht anmelden
M7SetTime	Datum/Uhrzeit stellen
M7UnLinkDate	Uhrzeitgesteuerte Zeitnachricht abmelden
M7UnLinkOneShotTimer	Singuläre Zeitnachricht abmelden
M7UnLinkPeriodTimer	Periodische Zeitnachricht abmelden

Betriebszustands-bearbeitung

In folgender Tabelle sind alle Funktionen, die Sie für die Überwachung des Betriebszustandes anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-23 Funktionen zur Bearbeitung des Betriebszustandes

Funktion	Kurzbeschreibung
M7ConfirmTransition	Quittieren einer Benachrichtigung über Betriebszustandsübergang
M7GetState	Betriebszustand abfragen
M7GetTSReason	Grund für den Übergang aus TSFRB ermitteln
M7GetTSType	Betriebszustand aus TSFRB ermitteln
M7LinkBatteryFailure	Einen BAFFRB für Batteriealarm anmelden
M7LinkState	Nachricht bei bestimmtem Betriebszustand anfordern
M7LinkTransition	Nachricht bei bestimmtem Betriebszustandsübergang anfordern
M7RequestState	Betriebszustandswechsel anfordern
M7UnLinkBatteryFailure	BAFFRB für Batteriealarm abmelden
M7UnLinkState	Abmelden eines TSFRB, der mit `M7LinkState` angemeldet wurde
M7UnLinkTransition	Abmelden eines TSFRB, der mit `M7LinkTransition` angemeldet wurde

Freier Zyklus

In folgender Tabelle sind alle Funktionen, die Sie zur An– und Abmeldung für den Anlauf, Systemkontrollpunkt, den ”Freien Zyklus” und Zykluszeitüberschreitung anwenden können, mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-24 Funktionen für Systemkontrollpunkt und ”Freien Zyklus”

Funktion	Kurzbeschreibung
M7ConfirmCycle	Quittieren einer Benachrichtigung
M7GetFSCTyp	Typ der Nachricht aus FSCFRB ermitteln
M7LinkCycle	Benachrichtigung für Anlauf, Systemkontrollpunkt, für den ”Freien Zyklus” bzw. Zykluszeitüberschreitung anfordern
M7RetriggerCycle	Zyklusüberwachung nachtriggern
M7UnLinkCycle	Abmelden der Benachrichtigung für Anlauf, Systemkontrollpunkt, den ”Freien Zyklus” bzw. Zykluszeitüberschreitung

Anwender-LED steuern

In folgender Tabelle ist die Funktion zum Ansteuern der Anwender–LED auf dem M7 aufgelistet:

Tabelle 1-25 Funktionen zum Ansteuern der Anwender–LED

Funktion	Kurzbeschreibung
M7SetUserLED	Anwender–LED ansteuern

Applikationsbeziehungs-Mangement

In folgender Tabelle sind die Funktionen zum Auf–, Abbau und zum Legitimieren einer K–BUS–Applikationsbeziehung mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-26 Funktionen fürs Applikationsbeziehungs–Management

Funktion	Kurzbeschreibung
M7GetConnState	Zustand einer Applikationsbeziehung abfragen
M7KAbort	Applikationsbeziehung zum Kommunikationspartner abbauen
M7KInitiate	Applikationsbeziehung zum Kommunikationspartner aufbauen
M7KPassWord	Applikationsbeziehung zum Kommunikationspartner legitimieren
M7GetPduSize	PDU–Größe ermitteln

Kommunikations-funktionen

In folgender Tabelle sind die Kommunikationsfunktionen mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-27 Kommunikationsfunktionen

Funktion	Kurzbeschreibung
M7PBKBrcv	Daten vom Partner empfangen (zweiseitige Kommunikationsfunktion)
M7PBKBsend	Daten zum Partner senden (zweiseitige Kommunikationsfunktion)
M7PBKCancel	`M7PBKBsend` bzw. `M7PBKBrcv`–Auftrag abbrechen
M7PBKGet	Daten vom Partner anfordern (einseitige Kommunikationsfunktion)
M7PBKIAbort	Schließen einer Applikationsbeziehung
M7PBKIGet	Asynchrones Lesen einer Variablen starten
M7PBKIPut	Asynchrones Schreiben einer Variablen starten
M7PBKPrint	Senden von Daten mit Formatbeschreibung
M7PBKPut	Daten zum Partner senden (einseitige Kommunikationsfunktion)
M7PBKResume	WIEDERANLAUF–Anforderung an Kommunikationspartner senden
M7PBKStart	NEUSTART–Anforderung an Kommunikationspartner senden
M7PBKStatus	Status des Kommunikationspartner ermitteln
M7PBKStop	STOP–Anforderung an Kommunikationspartner senden
M7PBKUrcv	Unkoordiniertes Empfangen über projektierte Verbindungen
M7PBKUsend	Unkoordiniertes Senden über projektierte Verbindungen
M7PBKXAbort	Schließen einer Applikationsbeziehung
M7PBKXCancel	Laufenden Empfangsauftrag von `M7PBKXrv` abbrechen.
M7PBKXGet	Asynchrones Lesen einer Variablen starten
M7PBKXPut	Asynchrones Schreiben einer Variablen starten

BuB-Funktionen

In folgender Tabelle sind die BuB–Funktionen mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-28 BuB–Funktionen

Funktion	Kurzbeschreibung
M7BUBCycRead	BuB–Auftrag zum zyklischen Lesen einrichten
M7BUBCycReadDelete	BuB–Auftrag zum zyklischen Lesen löschen
M7BUBCycReadStart	BuB–Auftrag zum zyklischen Lesen starten
M7BUBCycReadStop	Zyklisches Lesen stoppen
M7BUBRead	Einmaliges BuB–Lesen von Variablen
M7BUBWrite	Einmaliges BuB–Schreiben von Variablen

OVS-Funktionen

In folgender Tabelle sind die Funktionen an das Objektverwaltungssystem (OVS) mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-29 OVS–Funktionen

Funktion	Kurzbeschreibung
M7OVSCompress	Ladespeicher komprimieren
M7OVSDelete	Löschen eines Bausteins
M7OVSFindFirst	Ersten Eintrag aus dem Bausteinverzeichnis auslesen
M7OVSFindNext	Nächsten Eintrag aus dem Bausteinverzeichnis auslesen
M7OVSLinkln	Einketten eines Bausteins
M7OVSMemMode	Speichermodus einstellen
M7OVSRead	Hochladen eines Bausteins
M7OVSSetObjectHeader	Setzen eines S7–Objekt–Headers
M7OVSWrite	Kopieren eines Bausteins

Uhrzeit-Funktionen

In folgender Tabelle sind die Funktionen zum Lesen und Setzen der Uhrzeit mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-30 Funktionen zum Lesen/Setzen der Uhrzeit

Funktion	Kurzbeschreibung
M7KReadTime	Lesen der Uhrzeit über K–BUS
M7KWriteTime	Stellen der Uhrzeit über K–BUS

Diagnose–Server

In folgender Tabelle sind die Funktionen an den Diagnose–Server mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-31 Funktionen an den Diagnose–Server

Funktion	Kurzbeschreibung
M7DiagMode	Fürs Versenden von Diagnoseereignissen über K–BUS anmelden
M7SZLRead	Systemzustandsliste über K–BUS auslesen
M7WriteDiagnose	Anwendereintrag in lokalen Diagnose–Server schreiben

Sonstige Funktionen

In folgender Tabelle sind die sonstigen Funktionen mit einer Kurzbeschreibung aufgelistet.

Tabelle 1-32 Sonstige Funktionen

Funktion	Kurzbeschreibung
M7GetCommRequest	Auftragsnummer aus COMMFRB ermitteln
M7GetCommStatus	Status einer Datenübertragung aus COMMFRB ermitteln
M7KEvent	Daten nach einer Benachrichtigung abholen

1.4 DOS-Schnittstellenfunktionen

Einleitung

Für den schnellen Austausch von größeren Datenmengen bietet sich ein von M7 RMOS32 und MS–DOS gemeinsam nutzbarer Speicherbereich an. Allerdings gilt es dabei, die Speicheraufteilung zwischen den Betriebssystemen MS–DOS und M7 RMOS32 und die verschiedenen Adreßinterpretationen von Pointern (Real–Mode versus Flat) zu beachten.

Die DOS–Schnittstellenfunktionen sind nicht für zukünftige Entwicklungen vorgesehen!

Speicherverwaltung

Da MS–DOS–Anwendungen generell nur auf den Adreßbereich unterhalb von 1 MByte zugreifen können, andererseits aber der private Speicherbereich von M7 RMOS32–Tasks immer über der 1 MByte–Schwelle liegt, bietet M7 RMOS32 hierfür eine spezielle Speicherverwaltung.

Durch das TSR–Programm RM3_TSR wird ein Transferpuffer unterhalb von 1 MByte angelegt, aus dem RMOS–Tasks Speicherbereiche allokieren bzw. freigeben können.

Header–Dateien und Bibliotheken

Um die Funktionen der Speicherverwaltung des Transferpuffers in M7 RMOS32–Anwendungen nutzen zu können, müssen Sie in Ihren C–Programmen die Header–Datei **RM3DOS.H** inkludieren.

Außerdem müssen Sie in der Bindeanweisung des Linkers die zugehörige Bibliothek **RMFDOSIB.LIB** mit aufnehmen.

Kurzbeschreibung der Funktionen

In der folgenden Tabelle sind alle Funktionen, die von M7 RMOS32–Tasks für die Kommunikation mit MS–DOS–Anwendungen genutzt werden können, mit einer Kurzbeschreibung aufgelistet.

Eine detaillierte Beschreibung dieser Funktionen finden Sie im Kapitel 6.

Tabelle 1-33 Funktionen zur DOS–Kommunikation

Funktion	Kurzbeschreibung
x_dos_cpyin	Einen Speicherbereich aus dem Transferpuffer allokieren und Daten hineinkopieren.
x_dos_cpyout	Daten aus einem zuvor allokierten Bereich im Transferpuffer kopieren und anschließend den Bereich freigeben.

1.5 Funktionen der C–Laufzeitbibliothek

1.5.1 Übersicht

Einleitung

Die vorkonfigurierte C–Laufzeitunterstützung stellt alle Funktionen gemäß dem ANSI–Standard Draft International Standard ISO/IEC DIS 9899 (herausgegeben 1990) zur Verfügung.

Anforderung an die Speicherverwaltung

Für jede Task, die C–Laufzeitunterstützung anfordert, werden folgende Speicheranforderungen gestellt:

- ca. 1,3 KByte beim Aufruf der Initialisierungsfunktion `xinitt`. Diese Anforderung erfolgt auch implizit, wenn eine Task `xinitt` nicht aufruft, trotzdem aber C–Funktionen benutzt.
- ca. 1 KByte für jeden geöffneten Stream, falls die Größe des Stream–Puffers für diesen Stream nicht mit den Funktionen `setvbuf` oder `setbuf` redimensioniert wurde.

  Die für die Initialisierung und die Stream–Puffer benötigten Speicherbereiche werden dem **Heap** entnommen.
- Außerdem benötigt jede Task, die C–Funktionen der Laufzeitbibliothek benutzt, einen zusätzlichen Stackbereich von ca. 1 KByte.

Initialisierung der C–Laufzeitunterstützung

Um taskspezifische Daten zu initialisieren, muß die Funktion `xinitt` zu Beginn jeder Task aufgerufen werden. Erst danach stehen die eigentlichen Funktionen der C–Bibliothek zur Verfügung.

Hinweis

Fehlt der Aufruf `xinitt`, wird eine automatische Initialisierung vorgenommen.

Funktionen der C–Bibliothek

Die C-Bibliothek umfaßt Funktionen und Makros nach folgenden Ordnungskriterien bzw. Funktionsklassen (Diese Funktionsklassen sind weitgehend mit denen, in der gängigen Fachliteratur verwendeten, identisch.):

- Ein–/Ausgabe–Operationen, z.B. Festplatte, Terminal, Drucker, etc.
- Zeichenverwaltung
- String–Operationen
- Speicher–Operationen
- Speicherzuweisung
- mathematische Funktionen
- Zeit– und Datumsfunktionen
- Steuerfunktionen
- Fehlerbehandlung
- sonstige Funktionen

1.5.2 Ein–/Ausgabe–Operationen

Einleitung

Die umfangreichste Funktionsklasse der C–Bibliothek wird von den Ein–/Ausgabeoperationen gebildet. Sie enthält Funktionen, mit denen die Ein– und Ausgabe von C–Programmen aus durchgeführt werden können.

Ebenso enthält sie Funktionen zur Überprüfung und Formatierung von Ein–/Ausgaben und zur Dateiverwaltung. Die Funktionen sind in den Header–Dateien **IO.H** und **STDIO.H** deklariert.

Current Working Directory

Bei den Funktionen zum Öffnen, Umbenennen und Löschen von Dateien ist die Angabe eines Datei– oder Verzeichnisnamen erforderlich.

Dieser Name bezieht sich immer auf ein sog. Current Working Directory (CWD), das taskspezifisch zugeordnet ist. Zunächst ist aber das CWD einer Task nicht initialisiert. Das Initialisieren des CWD erfolgt mit Hilfe der Funktion `chdir`.

Regeln für Datei- und Verzeichnisnamen

Bei der Angabe eines Datei– oder Verzeichnisnamens gelten folgende Regeln:

- Der Doppelpunkt ':' dient als Trennzeichen zwischen dem Laufwerksnamen und dem Datei– bzw. Verzeichnisnamen. Er darf nur an zweiter oder dritter Stelle eines Pfadnamens stehen, an allen anderen Stellen ist er verboten. Dies bedeutet auch, daß nur Laufwerksnamen mit einer Länge von ein oder zwei Zeichen erlaubt sind.
  Beispiel: `R:TEST`
- Die Zeichen '\' und '/' sind Trennzeichen zwischen einzelnen Verzeichnisnamen oder zwischen Verzeichnis– und Dateinamen.
  Beispiel: `R:TEST\DIR1/DIR2\DATEI`
- Pfadnamen, die mit einem Laufwerksnamen beginnen, d.h. das zweite oder dritte Zeichen ist ein Doppelpunkt '**:**' (links davon steht der Name des Laufwerks), sind absolute Pfadnamen.
  Beispiel: `R:TEST\DIR1\DIR2\DATEI`
- Pfadnamen, die mit einem '\' oder '/' beginnen sind eine Sonderform des absoluten Pfadnamens. In diesem Fall wird nur der Laufwerksbuchstabe aus dem CWD übernommen und vor den angegebenen Pfadnamen gesetzt.
  Voraussetzung: bei der Verwendung dieser Art von Pfadnamen muß das CWD bereits initialisiert sein.
  Beispiel:

  ```
  R:TEST                    (CWD)
  \TEST2\DIR1\DIR2\DATEI    (angegebener Pfadname)
  R:TEST2\DIR1\DIR2\DATEI   (resultierender Pfadname)
  ```

  Eine Variante ist die Pfadangabe "\" bzw. "/". Dabei wird das Stammverzeichnis des im CWD angegebenen Laufwerks angesprochen. Eine Anwendung hierfür ist die Funktion `chdir("\\")` bzw. `chdir("/")`.
- Pfadnamen, die weder mit einem Laufwerksbuchstaben noch mit '\' oder '/' beginnen sind relative Pfadnamen, die sich auf das CWD beziehen.
  Beispiel:

  ```
  R:TEST                    (CWD)
  DIR2\DATEI                (angegebener Pfadname)
  R:TEST\DIR2\DATEI         (resultierender Pfadname)
  ```
- Pfadnamen. die mit ..<Trennzeichen> beginnen, sind eine Sonderform des relativen Pfadnamens. Die Pfadangabe bezieht sich dann auf das Vaterverzeichnis des CWD.
  Beispiel:

  ```
  R:TEST\DIR1               (CWD)
  ..\DIR2\DATEI             (angegebener Pfadname)
  R:TEST\DIR2\DATEI         (resultierender Pfadname)
  ```

  Eine Variante ist die Pfadangabe "`..`". Dabei wird das Verzeichnis angesprochen, das dem Laufwerksnamen um eine Ebene näher ist als das CWD. Eine Anwendung hierfür ist die Funktion `chdir("..")`.

Hinweis

Wurde das CWD einer Task noch nicht initialisiert, so müssen absolute Pfadnamen verwendet werden.

Eine Unterscheidung zwischen Klein– und Großbuchstaben ist, wie beim MS–DOS Dateisystem nicht erforderlich.

Textmodus – Binärmodus

Bei den Funktionen `fopen`, `fduopen`, `freopen`, `fdureopen` und `open` wird angegeben, ob ein Stream bzw. ein Handle im Text– oder im Binärmodus geöffnet werden soll.

Wird ein Stream bzw. Handle im Textmodus geöffnet, so werden bei Schreibzugriffen alle '\n' (New Line) in '\r\n' (Carriage Return – New Line) umgewandelt, bei Leseoperationen wird umgekehrt verfahren, d.h. alle '\r\n' werden in '\n' umgewandelt.

Bei Streams bzw. Handles, die im Binärmodus geöffnet wurden, erfolgt keine Konvertierung.

NUL–Datei

Es existiert eine NUL–Datei, die zwar geöffnet werden kann, aber physikalisch nirgendwo vorhanden ist. Nach dem Öffnen der NUL–Datei sind alle Operationen möglich, die auch mit normalen Dateien erlaubt sind.

Der Unterschied liegt darin, daß Schreib– und Leseaufrufe sofort beendet werden, ohne daß Ein–/Ausgabeoperationen vorgenommen werden.

Alle Schreiboperationen auf die NUL–Datei werden so beendet, daß kein Fehler (**errno**, **errno2**, etc.) signalisiert wird. Lese–Operationen melden immer **EOF** (End of File).

Wird für Datei- oder Pfadnamen **NUL** (in beliebiger Kombination von Klein- und Großbuchstaben) angegeben, wird damit die NUL-Datei angesprochen (z.B. *`fopen("NUL","w")`*).

Tabelle 1-34 Ein–/Ausgabe–Operationen

Aufruf	Bedeutung	Header–Datei
access	prüft Datei–Zugriffsrechte des Benutzers	IO.H
changevib	Beschreibungsblock eines Datenträgers ändern	IO.H
chdir	wechselt das CWD	DIRECT.H
checkpoint	Rückschreiben der (HSFS) Zwischenpuffer einer Datei	IO.H
chmod	Attribute einer Datei ändern	IO.H
clearerr	löscht den Fehler–Status eines Streams	STDIO.H
close	Offene Datei, Unit eines ladbaren Treibers oder Socket schließen	IO.H

Tabelle 1-34 Ein–/Ausgabe–Operationen

Aufruf	Bedeutung	Header–Datei
createvib	Neuen Beschreibungsblock eines Datenträgers erzeugen	IO.H
dismount	meldet ein HSFS–Gerät ab	IO.H
duread	liest Zeichen über RMOS–Treiber	IO.H
duwrite	gibt Zeichen über RMOS–Treiber aus	IO.H
efsstop	Verbindung zwischen Netz–Unit und Server–Unit abbauen.	IO.H
efsuse	Verbindung zwischen Netz–Unit und Server–Unit aufbauen.	IO.H
fclose	schließt einen Stream	STDIO.H
fduopen	öffnet einen Stream über RMOS–Treiber	STDIO.H
fdureopen	lenkt Stream auf RMOS–Treiber um	STDIO.H
feof	überprüft, ob Dateiende erreicht ist	STDIO.H
ferror	fragt Stream–Status ab	STDIO.H
fflush	leert den Puffer eines Streams	STDIO.H
fgetc	liest Zeichen aus einem Stream	STDIO.H
fgetpos	ermittelt Position in der Datei	STDIO.H
fgets	liest String aus einem Stream	STDIO.H
fileno	liefert den Dateideskriptor, der dem angegebenen Stream zugeordnet ist	STDIO.H
fopen	eröffnet Stream	STDIO.H
fprintf	schreibt formatierte Angabe in einen Stream	STDIO.H
fputc	schreibt ein Zeichen in einen Stream	STDIO.H
fputs	schreibt String in einen Stream	STDIO.H
fread	liest aus einem Stream	STDIO.H
freopen	tauscht die einem Stream zugeordnete Datei aus	STDIO.H
fscanf	liest formatierte Eingabe aus einem Stream	STDIO.H
fseek	positioniert Dateizeiger in einem Stream	STDIO.H
fsetpos	setzt Position in einer Datei	STDIO.H
ftell	gibt die Distanz des Dateizeigers zum Dateianfang zurück	STDIO.H
fwrite	schreibt in einen Stream	STDIO.H
getc	liest ein Zeichen aus einem Stream	STDIO.H
getchar	liest ein Zeichen von stdin	STDIO.H
getcwd	ermittelt CWD	DIRECT.H
gets	liest einen String aus einem Stream	STDIO.H
getvolumestatus	Statusinformation eines Datenträgers ermitteln	IO.H

Tabelle 1-34 Ein–/Ausgabe–Operationen

Aufruf	Bedeutung	Header–Datei
getw	liest ein Wort aus einem Stream	STDIO.H
ioctl	Steuerfunktion für Socket oder Unit eines ladbaren Treibers ausführen	IO.H
lseek	positioniert Dateizeiger	IO.H
mkdir	Verzeichnis erzeugen	DIRECT.H
mount	meldet HSFS–Gerät an	IO.H
open	öffnet Datei zum Lesen und/oder Schreiben	IO.H
printf	schreibt formatierte Ausgabe nach stdout	STDIO.H
putc	schreibt ein Zeichen in einen Stream	STDIO.H
putchar	schreibt ein Zeichen nach stdout	STDIO.H
puts	schreibt einen String in einen Stream	STDIO.H
putw	schreibt ein Wort in einen Stream	STDIO.H
read	liest aus einer Datei	IO.H
remap	formatiert einen Datenträger	IO.H
remove	löscht eine Datei	STDIO.H
rename	ändert den Namen einer Datei	STDIO.H
rewind	positioniert Dateizeiger an den Anfang	STDIO.H
rmdir	Verzeichnis löschen	DIRECT.H
scanf	liest formatierte Eingabe von stdin	STDIO.H
search	sucht Dateien	IO.H
setbuf	weist Puffer für Stream zu	STDIO.H
setvbuf	weist Puffer für Stream zu	STDIO.H
sprintf	schreibt formatierte Ausgabe in einen String	STDIO.H
sscanf	liest formatierte Eingabe aus einem String	STDIO.H
tmpfile	legt temporäre Datei an	STDIO.H
tmpnam	erzeugt Namen für temporäre Datei	STDIO.H
ungetc	schreibt Zeichen in Stream zurück	STDIO.H
unlink	löscht Datei	IO.H
vfprintf	gibt varargs–Argument–Liste formatiert aus	STDIO.H
vprintf	gibt varargs–Argument–Liste formatiert aus	STDIO.H
vsprintf	gibt varargs–Argument–Liste formatiert aus	STDIO.H
write	schreibt in eine Datei	IO.H

1.5.3 Funktionen der Zeichenverwaltung

Die Zeichenverwaltung beinhaltet Funktionen zur Konvertierung und Klassifizierung von Zeichentypen. Sie sind in der Header–Datei **CTYPE.H** deklariert.

Tabelle 1-35 Funktionen der Zeichenverwaltung

Aufruf	**Bedeutung**	**Header–Datei**
_tolower	wandelt Groß– in Kleinbuchstaben um	CTYPE.H
_toupper	wandelt Klein– in Großbuchstaben um	CTYPE.H
isalnum	bestimmt Zeichentyp (alphanumerisch)	CTYPE.H
isalpha	bestimmt Zeichentyp (Alphabetzeichen)	CTYPE.H
isascii	bestimmt Zeichentyp (ASCII–Code 0 – 127)	CTYPE.H
iscntrl	bestimmt Zeichentyp (ASCII–Code > 127 oder < 32)	CTYPE.H
isdigit	bestimmt Zeichentyp (Dezimalzahl 0 – 9)	CTYPE.H
isgraph	bestimmt Zeichentyp (druckbares Zeichen, kein Leerzeichen)	CTYPE.H
islower	bestimmt Zeichentyp (Kleinbuchstabe)	CTYPE.H
isprint	bestimmt Zeichentyp (ASCII–Code 32 – 126)	CTYPE.H
ispunct	bestimmt Zeichentyp (Satzzeichen)	CTYPE.H
isspace	bestimmt Zeichentyp (Leerzeichen, Tabulator,...)	CTYPE.H
isupper	bestimmt Zeichentyp (Großbuchstabe)	CTYPE.H
isxdigit	bestimmt Zeichentyp (Hexadezimalzahl 0 – 9, A – F, a – f)	CTYPE.H
toascii	blendet alle Nicht–ASCII–Bits aus	CTYPE.H
tolower	wandelt Groß– in Kleinbuchstaben um	CTYPE.H
toupper	wandelt Klein– in Großbuchstaben um	CTYPE.H

1.5.4 String–Operationen

Mit String–Operationen lassen sich Zeichen bzw. Byte–Ketten überprüfen sowie be– und verarbeiten. Sie sind in den Header–Dateien **STRING.H** und **STDLIB.H** deklariert.

Tabelle 1-36 String–Operationen

Aufruf	Bedeutung	Header–Datei
atof	wandelt String in eine Double–Zahl um	STDLIB.H
atoi	wandelt String in Integer–Zahl um	STDLIB.H
atol	wandelt String in Long–Zahl um	STDLIB.H
strcat	hängt einen String an einen anderen an	STRING.H
strchr	ermittelt ein Zeichen in einem String	STRING.H
strcmp	vergleicht zwei Strings	STRING.H
strcpy	kopiert einen String in einen anderen	STRING.H
strcspn	gibt an, wie weit ein String mit einem anderen übereinstimmt	STRING.H
strlen	gibt die Anzahl der Zeichen in einem String zurück	STRING.H
strncat	hängt maximal n Zeichen von einem String an einen anderen String an	STRING.H
strncmp	vergleicht maximal n Zeichen zweier Strings	STRING.H
strncpy	kopiert einen String in einen anderen, maximal n Zeichen	STRING.H
strpbrk	sucht einen String nach dem ersten Auftreten eines Zeichens ab	STRING.H
strrchr	sucht einen String nach dem letzten Auftreten eines Zeichens ab	STRING.H
strspn	liefert die Länge des Teilstrings von String 1 zurück, der ausschließlich aus in String 2 angegebenen Zeichen besteht	STRING.H
strstr	gibt die erste Übereinstimmung von String 2 mit String 1 an	STRING.H
strtod	wandelt String in eine Double–Zahl um	STDLIB.H
strtok	sucht einen String nach der ersten von mehreren Zeichenfolgen ab	STRING.H
strtol	wandelt String in eine Long–Zahl um	STDLIB.H
strtoul	wandelt String in eine Unsigned–Long–Zahl um	STDLIB.H

1.5.5 Speicher–Operationen

Mit Speicher–Operationen können Zeichen kopiert, sowie Speicherbereiche verglichen oder beschrieben werden. Die Speicheroperationen sind in den Header–Dateien **STRING.H** und **MEMORY.H** deklariert

Tabelle 1-37 Speicher–Operationen

Aufruf	Bedeutung	Header–Datei
memccpy	kopiert Zeichen vom Quellbereich in den Zielbereich	STRING.H, MEMORY.H
memchr	sucht in einem Speicherbereich nach einem Zeichen	STRING.H, MEMORY.H
memcmp	vergleicht zwei Speicherbereiche	STRING.H, MEMORY.H
memcpy	kopiert Zeichen vom Quellbereich in den Zielbereich	STRING.H, MEMORY.H
memmove	verschiebt Zeichen vom Quellbereich in den Zielbereich	STRING.H, MEMORY.H
memset	schreibt n mal ein Zeichen in einen Speicherbereich	STRING.H, MEMORY.H

1.5.6 Speicherzuweisungen

Mit Hilfe dieser Funktionen kann Speicher aus dem Heap allokiert werden. Die Deklarationen der Funktionen finden sich in den Header–Dateien **MALLOC.H** und auch in **STDLIB.H**.

Tabelle 1-38 Speicherzuweisungs–Operationen

Aufruf	Bedeutung	Header–Datei
calloc	fordert Speicherplatz für eine Anzahl von n Elementen einer bestimmten Größe an	MALLOC.C, STDLIB.H
free	gibt Speicherplatz frei	MALLOC.C, STDLIB.H
malloc	fordert Speicherplatz an	MALLOC.C, STDLIB.H
realloc	ändert die Größe eines zuvor angeforderten Speicherbereichs	MALLOC.C, STDLIB.H

1.5.7 Mathematische Funktionen

Diejenigen Funktionen, die in der Header–Datei **STDLIB.H** deklariert sind, können nur auf ganzzahlige Werte angewendet werden. Gleitkomma–Funktionen sind dagegen in der Header–Datei **MATH.H** deklariert.

Tabelle 1-39 Mathematische Funktionen

Aufruf	Bedeutung	Header–Datei
abs	ermittelt Absolutwert einer Integer–Zahl	STDLIB.H
acos	berechnet Arcuscosinus aus einer Double–Zahl	MATH.H
asin	berechnet Arcussinus aus einer Double–Zahl	MATH.H
atan	berechnet Arcustangens aus einer Double–Zahl	MATH.H
atan2	berechnet den Arcustangens aus zwei Double–Zahlen unter Berücksichtigung aller vier Quadranten	MATH.H
ceil	rundet auf die nächst größere ganze Double–Zahl auf	MATH.H
cos	berechnet den Cosinus aus einer Double–Zahl	MATH.H
cosh	berechnet den hyperbolischen Cosinus aus einer Double–Zahl	MATH.H
div	dividiert zwei ganzzahlige Werte	STDLIB.H
exp	berechnet e^x von einer Double–Zahl	MATH.H
fabs	berechnet den Absolutwert aus einer Double–Zahl	MATH.H
floor	rundet auf die nächst kleinere ganze Double–Zahl ab	MATH.H
fmod	berechnet den Divisionsrest aus zwei Double–Zahlen	MATH.H
frexp	liefert die Mantisse und den binären Exponenten	MATH.H
labs	ermittelt den Absolutwert einer Long–Zahl	STDLIB.H
ldexp	berechnet Double–Zahl*2 $^{\text{Integer–Zahl}}$	MATH.H
ldiv	dividiert zwei ganzzahlige Werte	STDLIB.H
log	berechnet den natürlichen Logarithmus aus einer Double–Zahl	MATH.H
log10	berechnet den Logarithmus zur Basis 10 aus einer Double–Zahl	MATH.H
matherr	anwenderspezifische Funktion zur Fehlerbehandlung in numerischen Funktionen	MATH.H
modf	teilt eine Double–Zahl in Mantisse und Exponent auf	MATH.H

Tabelle 1-39 Mathematische Funktionen

Aufruf	Bedeutung	Header–Datei
pow	Berechnet die Potenz aus zwei Double–Zahlen	MATH.H
rand	erzeugt Integer–Zufallszahl	STDLIB.H
sin	berechnet den Sinus aus einer Double–Zahl	MATH.H
sinh	berechnet den hyperbolischen Sinus aus einer Double–Zahl	MATH.H
sqrt	berechnet die Quadratwurzel aus einer Double–Zahl	MATH.H
srand	initialisierungswert für Pseudozufallszahlen	STDLIB.H
tan	berechnet den Tangens aus einer Double–Zahl	MATH.H
tanh	berechnet den hyperbolischen Tangens aus einer Double–Zahl	MATH.H

1.5.8 Zeit– und Datumsfunktionen

Mit diesen Funktionen können Zeit– und Datumsangaben z.B. konvertiert und nach Zeitzonen angepaßt werden. Die Funktionen sind in der Header–Datei **TIME.H** deklariert.

Tabelle 1-40 Zeit– und Datumsfunktionen

Aufruf	Bedeutung	Header–Datei
asctime	wandelt eine Zeitangabe in einen String um	TIME.H
ctime	wandelt Datum und Uhrzeit in einen String um	TIME.H
difftime	bildet die Differenz zwischen zwei Zeitangaben	TIME.H
gmtime	wandelt Zeitangabe in Greenwich Mean Time (GMT) um	TIME.H
localtime	korrigiert Zeitangaben nach Zeitzonen–Differenzen	TIME.H
mktime	konvertiert Zeitangabe	TIME.H
strftime	formatiert Ausgabe von Datum und Uhrzeit	TIME.H
time	ermittelt Systemzeit	TIME.H
tzset	berechnet die Zeitzonenanpassung	TIME.H

1.5.9 Steuerfunktionen

Die Steuerfunktionen werden zum Beenden von Tasks benötigt. Sie sind in der Header–Datei **STDLIB.H** deklariert.

Tabelle 1-41 Steuerfunktionen

Aufruf	Bedeutung	Header–Datei
abort	sendet Signal SIGABRT zur aufrufenden Task	STDLIB.H
assert	prüft eine Bedingung und bricht bei Nichterfüllung die Task ab	ASSERT.H
atexit	legt Routinen fest, die am Ende einer Task aufgerufen werden sollen	STDLIB.H
exit	Task bereinigen und mit festgesetztem Status beenden	STDLIB.H
x_cr_killtsk	Task löschen	TASK.H

1.5.10 Fehlerbehandlung

Neben **errno** steht zusätzlich das Makro **errno2** (RMOS–Erweiterung) zur Verfügung.

Tabelle 1-42 Fehlerbehandlungs–Funktionen

Aufruf	Bedeutung	Header–Datei
errno, errno2	Fehlernummer	ERRNO.H
perror	gibt Betriebssystem–Fehlermeldungen aus	STDIO.H
strerror	liefert einen Zeiger auf einen Fehlertext	STRING.H
sys_nerr	ist die Anzahl der Fehlermeldungen in sys_errlist	ERRNO.H
sys_errlist	ist ein String–Array mit Fehlermeldungen	ERRNO.H

1.5.11 Sonstige Funktionen

Hier sind eine Reihe von Funktionen zusammengefaßt, die sich nicht eindeutig einer Klasse zuordnen lassen

Tabelle 1-43 Sonstige Funktionen

Aufruf	Bedeutung	Header–Datei
bsearch	binäres Suchen in einer sortierten Tabelle	STDLIB.H
getenv	ermittelt den Inhalt einer Umgebungs–Variablen	STDLIB.H
longjmp	führt einen nicht lokalen Sprung aus	SETJMP.H
putenv	ändert eine Umgebungs–Variable oder fügt eine neue hinzu	STDLIB.H
qsort	sortiert Datenelemente nach der angegebenen Reihenfolge	STDLIB.H
raise	übergibt die Kontrolle an einen Signal–Handler	SIGNAL.H
setjmp	setzt setzt die Marke für einen späteren nicht lokalen Sprung	SETJMP.H
signal	Installiert einen Signal–Handler zur Exception–Behandlung	SIGNAL.H
sleep	hält Task für bestimmte Zeit an	TIME.H
x_cr_gettaskid	ID der aufrufenden Task ermitteln	TASK.H
x_cr_gettaskparam	`stdin`, `stdout`, `stderr` und Taskumgebung ermitteln	TASK.H
x_cr_initenv	Taskumgebung initialisieren	TASK.H
x_cr_setexit	Taskspezifischen Exit–Handler setzen	TASK.H
xinitc	initialisiert C –Bibliothek	TASK.H
xinitt	Taskspezifische Initialisierung der C–Bibliothek vornehmen	TASK.H

1.6 Funktionen der Socket–Schnittstelle

Allgemeine Hinweise

M7–SYS RT enthält die Socket-Schnittstelle für die Kommunikation über TCP/IP. Für die Prototypen der Funktionen muß von den M7 RMOS32–Programmen die Header–Datei **SOCKET.H** inkludiert werden.

Zusätzlich muß beim Linken die Bibliothek RMFSK2IB.LIB eingebunden werden.

Tabelle 1-44 Funktionen der Socket- Schnittstelle

Aufruf	Bedeutung
accept	Verbindung über Socket annehmen
bind	Name an Socket binden
connect	Verbindung über Socket anfordern
endhostent	HOSTS-Datei schließen
endnet	Taskbezogene Ressourcen der Sockets freigeben
endservent	SERVICES-Datei schließen
gethostbyaddr	Eintrag eines Kommunikationsteilnehmers aus HOSTS-Datei lesen
gethostbyname	Eintrageines Kommunikationsteilnehmers aus HOSTS-Datei lesen
gethostent	Eintrag aus HOSTS-Datei lesen
getpeername	Namen des mit dem Socket verbundenen Peer lesen
getservbyname	Eintrag eines Kommunikationsdienstes aus SERVICES-Datei lesen
getservbyport	Eintrag eines Kommunikationsdienstes aus SERVICES-Datei lesen
getservent	Eintrag aus SERVICES-Datei lesen
getsockname	Socket-Namen lesen
getsockopt	Socket-Optionen lesen
htons	Wert von Host Byte Order in Network Byte Order umwandeln
listen	Socket für passiven Verbindungsaufbau vorbereiten
nselect	Gleichzeitiges Warten auf Ereignisse an mehreren Sockets
ntohs	Wert von Network Byte Order in Host Byte Order umwandeln
recv	Nachricht von einem Socket empfangen
recvfrom	Datagramm-Nachricht empfangen
send	Nachricht an verbundenen Socket senden

Tabelle 1-44 Funktionen der Socket- Schnittstelle

Aufruf	Bedeutung
sendto	Nachricht an einen Socket mit einer bestimmter Adresse senden
sethostent	HOSTS-Datei öffnen
setservent	SERVICES-Datei öffnen
setsockopt	Socket-Optionen setzen
shutdown	Das Senden von Nachrichten unterbinden
socket	Endpunkt für Kommunikation erzeugen

1.7 Funktionen für serielle Schnittstellen

Allgemeine Hinweise

RMOS bietet ein API für serielle Schnittstellen. Als Header–Datei für die Prototypen der Funktionen muß von den M7 RMOS32–Programmen die Header–Datei **SERIAL.H** inkludiert werden.

Zusätzlich muß beim Linken die Bibliothek RMFSER.LIB eingebunden werden.

Tabelle 1-45 Funktionen für serielle Schnittstellen

Aufruf	Bedeutung
SerialCheckChar	Einzelnes Zeichen von Unit einlesen
SerialCheckString	String von Unit einlesen
SerialClose	Verbindung zu einer Unit eines Treibers schließen
SerialGetChar	Einzelnes Zeichen von Unit einlesen
SerialGetString	String von Unit einlesen
SerialInit	Unit initialisieren
SerialInitEx	Erweiterte Initialisierung der Unit
SerialOpen	Verbindung zu einer Unit eines Treibers herstellen
SerialPutChar	Einzelnes Zeichen auf Unit schreiben
SerialPutString	Zeichen auf Unit schreiben

1.8 Sonstige Funktionen

Allgemeine Hinweise

RMOS bietet weitere Funktionen für hardware–orientierte I/O–Operationen und die Interruptbearbeitung. Als Header–Datei für die Prototypen der Funktionen muß von den M7 RMOS32–Programmen die Header–Datei **MISC86.H** inkludiert werden.

1.8.1 Funktionen zur Interruptbearbeitung

Folgende Funktionen stehen zur Interruptbearbeitung zur Verfügung.

Tabelle 1-46 Funktionen zur Interruptbearbeitung

Aufruf	Bedeutung
causeinterrupt	Software–Interrupt erzeugen
geninterrupt	Software–Interrupt erzeugen

1.8.2 Funktionen für hardware–orientierte I/O–Operationen

Folgende Funktionen stehen für hardware–orientierte I/O–Operationen zur Verfügung.

Tabelle 1-47 Funktionen für hardware–orientierte I/O–Operationen

Aufruf	Bedeutung
disable	Hardware–Interrupts deaktivieren
enable	Hardware–Interrupts aktivieren
inbyte	Byte von einer Schnittstelle lesen
inp	Byte von einer Schnittstelle lesen
inport	Wort von Schnittstelle lesen
inport b	Byte von Schnittstelle lesen
inpw	Wort von Hardware–Schnittstelle lesen
inword	Wort von Schnittstelle lesen
outbyte	Byte an I/O–Adresse schreiben
outp	Byte an eine I/O–Adresse schreiben
outport	Wort an I/O–Adresse schreiben
outportb	Byte an I/O–Adresse schreiben
outpw	Wort an eine I/O–Adresse schreiben
outword	Wort an eine I/O–Adresse schreiben

Typkennzeichen

2

Kapitelübersicht

2.1 Systemnachrichten der M7–Server

Hinweise

Im folgenden werden die Kennungen der Systemnachrichten der M7–Server in aufsteigender numerischer Reihenfolge aufgeführt. Die M7 RMOS32–Tasks können sich bei den M7–Servern anmelden, um beim Eintreten des zugehörigen Ereignisses benachrichtigt zu werden.

Die M7–Server schicken die Nachrichten mit zugehöriger Kennung an die Nachrichtenwarteschlange der Tasks. Diese können mit Hilfe der Funktion `RmReadMessage` die Nachricht auslesen und das Nachrichtenkennzeichen, das in der Variablen *Message* übergeben wird, beispielsweise mit einer ”switch–Anweisung” auswerten.

Alle Nachrichten enthalten zusätzlich im Parameter *pMessageParam* die Adresse des bei der Anmeldung über den `M7Link...`–Aufrufs referenzierten FRBs.

Die im folgenden aufgelisteten Konstanten für die Nachrichtenkennungen sind in der Datei **M7API.H** definiert. Alle numerischen Konstanten werden in der Headerdatei explizit in den C–Typ *unsigned int* ”gecastet”.

BZÜ-Server

Die folgende Tabelle zeigt die im Parameter *Message* übergebenen Nachrichtenkennzeichen, der vom BZÜ–Server an M7 RMOS32–Tasks verschickten Nachricht.

Tabelle 2-1 Nachrichten des BZÜ–Servers

Kennung	Beschreibung
M7MSG_TRANSITION	Benachrichtigung durch den BZÜ-Server **vor** dem Übergang in einen neuen Betriebszustand. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkTransition`übergebenen **M7TSFRB**.
M7MSG_STATE	Benachrichtigung durch den BZÜ-Server unmittelbar **nach** dem Übergang in einen neuen Betriebszustand. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkState`übergebenen **M7TSFRB**.
M7MSG_REQ_FINISHED	Benachrichtigung durch den BZÜ-Server unmittelbar **nach** dem Übergang in den angeforderten Betriebszustand bzw. dessen Ablehnung. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7Request-State` übergebenen **M7TSFRB**.
M7MSG_BATTERY_FAILURE	Benachrichtigung durch den BZÜ-Servers unmittelbar **nach** dem Unterschreiten des Grenzwerts für die Batteriespannung. Die Variable *pMessage-Param* referenziert den bei der Anmeldung über `M7LinkBatteryFailure` übergebenen **M7BAFFRB**.

S7–Objekt–Server

Die folgende Tabelle zeigt die Nachrichtenkennzeichen, die vom S7–Objekt–Server an M7 RMOS32–Tasks verschickt werden.

Tabelle 2-2 Nachrichten des S7–Objekt–Servers

Kennung	Beschreibung
M7MSG_DATA_ACCESS_R	Benachrichtigung durch den S7–Objekt-Server unmittelbar **nach** dem lesenden Zugriff auf ein S7–Objekt. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkDataAccess` übergebenen **M7OBJFRB**.
M7MSG_DATA_ACCESS_W	Benachrichtigung durch den S7–Objekt-Server unmittelbar **nach** dem schreibenden Zugriff auf ein S7–Objekt. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkDataAccess` übergebenen **M7OBJFRB**.
M7MSG_DATA_ACCESS_CREATE	Benachrichtigung durch den S7–Objekt-Server unmittelbar **nach** dem Erzeugen eines neuen S7–Objekts. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkDataAccess` übergebenen **M7OBJFRB**.
M7MSG_DATA_ACCESS_DEL	Benachrichtigung durch den S7–Objekt-Server unmittelbar **nach** dem Löschen eines S7–Objekts. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkDataAccess`übergebenen **M7OBJFRB**.
M7MSG_DATA_ACCESS_LINK	Benachrichtigung durch den S7–Objekt-Server unmittelbar **nach** dem Einketten eines S7–Objekts. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkDataAccess`übergebenen **M7OBJFRB**.

Time–Server

Die folgende Tabelle zeigt das Nachrichtenkennzeichen, das vom Time–Server an M7 RMOS32–Tasks verschickt wird.

Tabelle 2-3 Nachricht des Time–Servers

Kennung	Beschreibung
M7MSG_TIMESERVER	Benachrichtigung durch den Time-Server unmittelbar **nach** dem Zeitereignis. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7Link..` übergebenen **M7TFRB**.

FZ–Server

Die folgende Tabelle zeigt das Nachrichtenkennzeichen, das vom FZ–Server an M7 RMOS32–Tasks verschickt wird.

Tabelle 2-4 Nachricht des FZ–Servers

Kennung	Beschreibung
M7MSG_CYCLE	Benachrichtigung durch den FZ-Server zu Beginn eines Zustandes (STARTUP, FREECYCLE, ZKP).
M7MSG_PI_ERROR	Benachrichtigung durch den FZ-Server nach Auftreten eines Prozeßabbildtransferfehlers.

Alarm–Server

Die folgende Tabelle zeigt die Nachrichtenkennzeichen, die vom Alarm–Server an M7 RMOS32–Tasks verschickt werden.

Tabelle 2-5 Nachrichten des Alarm–Servers

Kennung	Beschreibung
M7MSG_IO_ALARM	Benachrichtigung durch den Alarm-Server unmittelbar **nach** dem Melden eines I/O–Alarms durch die entsprechende Baugruppe. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkIOAlarm`übergebenen **M7IOFRB**.
M7MSG_DIAG_ALARM	Benachrichtigung durch den Alarm-Server unmittelbar **nach** dem Melden eines Diagnosealarms durch die entsprechende Baugruppe. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkDiagAlarm`übergebenen **M7DIAGFRB**.
M7MSG_ZS_ALARM	Benachrichtigung durch den Alarm-Server unmittelbar **nach** dem Melden eines Ziehen/Stecken–Alarms durch die entsprechende Baugruppe. Die Variable *pMessageParam* referenziert den bei der Anmeldung über `M7LinkZSAlarm` übergebenen **M7ZSFRB**.

K-BUS-Subsystem

Die folgende Auflistung zeigt die Nachrichtenkennzeichen, die vom K_BUS–Subsystem an M7 RMOS32–Tasks verschickt werden.

Tabelle 2-6 Nachrichten des K–BUS–Subsystems

Kennung	Beschreibung
M7MSG_DIAG_MSG	Die Benachrichtigung durch das K–BUS–Subsystem signalisiert den Erhalt einer Diagnosemeldung, die über den Aufruf `M7KEvent` von der M7 RMOS32–Task ausgelesen werden kann.
M7MSG_BUB_NDR	Die Benachrichtigung durch das K–BUS–Subsystem signalisiert den Erhalt neuer BuB–Daten, die über den Aufruf `M7KEvent` von der M7 RMOS32–Task ausgelesen werden können.
M7MSG_PBK_NDR	Die Benachrichtigung durch das K–BUS–Subsystem signalisiert den Erhalt neuer Daten nach einem `M7PBKBrcv`–Aufruf.
M7MSG_PBK_DONE	Die Benachrichtigung durch das K–BUS–Subsystem signalisiert den Abschluß eines `M7PBKBsend`–Aufrufs.

2.2 Kennungen für S7-Objekte und Datentypen

Typkennzeichen

Die in der folgenden Tabelle aufgelisteten S7-Objekte werden vom S7–Objekt–Server auf einem M7 Automatisierungsrechner unterstützt. Die aufgeführten Typkennzeichen sind in der Header–Datei **M7API.H** definiert und werden in den entsprechenden M7 API-Funktionsaufrufen zum Ansprechen der S7-Objekte benötigt.

Die zugehörigen numerischen Werte werden in **M7API.H** in den M7–Datentyp UBYTE gecastet.

Tabelle 2-7 Auf dem M7 unterstützte S7-Objekte

S7–Objekt	Typkennzeichen	wird eingerichtet durch
Peripheriebereich	M7D_IO	automatisch
Prozeßabbild der Eingänge	M7D_PII	automatisch
Prozeßabbild der Ausgänge	M7D_PIQ	automatisch
Merkerbereich	M7D_M	C-Anwenderprogramm
Datenbaustein	M7D_DB	C-Anwenderprogramm
Datensätze, Lesen * (Bei Kommunikation nur BuB–Funktionen)	M7D_PAR_READ	C-Anwenderprogramm
Datensätze, Schreiben * (Bei Kommunikation nur BuB–Funktionen)	M7D_PAR_WRITE	C-Anwenderprogramm

*Auf einer FM sind die Attribute “Lesen” und “Schreiben” für Datensätze aus Sicht der CPU zu betrachten.

Die FM liest die Datensätze – z.B. Parameterdatensätze – die von der CPU geschrieben wurden (Typkennzeichen M7D_PAR_WRITE). Andererseits schreibt die FM Datensätze – z.B. Diagnosedatensätze – die von der CPU gelesen werden sollen (Typkennzeichen M7D_PAR_READ).

Teilbereichs-nummer

Die folgende Tabelle listet die zu den einzelnen S7–Objekten gehörigen Teilbereichsnummern auf. Die aufgeführten Teilbereichsnummern werden in den entsprechenden M7 API-Funktionsaufrufen zum Ansprechen der S7-Objekte einer S7–CPU bzw. eines M7s benötigt.

Tabelle 2-8 Zu den S7-Objekten gehörige Teilbereichsnummern

S7–Objekt	Typkennzeichen	Teilbereichsnummer	Wertebereich
Peripheriebereich	M7D_IO	0	0 ... 0xFFFF
Prozeßabbild der Eingänge	M7D_PII	0	0 ... 255 bzw. 511
Prozeßabbild der Ausgänge	M7D_PIQ	0	0 ... 255 bzw. 511
Merkerbereich	M7D_M	0	0 ... 65 535
Datenbaustein	M7D_DB	DB–Nummer	0 ... 65 535 bei M7

Tabelle 2-8 Zu den S7-Objekten gehörige Teilbereichsnummern,Fortsetzung

S7–Objekt	Typkennzeichen	Teilbereichsnummer	Wertebereich
Datensätze, Lesen (Bei Kommunikation nur BuB–Funktionen)	M7D_PAR_READ	Nr. des Datensatzes	0 ... 255 bei M7
Datensätze, Schreiben (Bei Kommunikation nur BuB–Funktionen)	M7D_PAR_WRITE	Nr. des Datensatzes	0 ... 255 bei M7

Datentyp-kennzeichen

Die Kennzeichen in der folgenden Tabelle spezifizieren die möglichen Datentypen von Variablen innerhalb von S7–Objekten. Die Kennzeichen werden in allen M7–Aufrufen verwendet, die auf einen Variablenbereich innerhalb eines S7–Objekts zugreifen.

Die zugehörigen M7–Datentypen sind in der folgenden Tabelle aufgelistet.

Tabelle 2-9 Datentypkennzeichen bei Zugriff auf S7–Objekten

M7–Datentyp	Typkennzeichen
BOOL	M7DT_BOOL
UBYTE	M7DT_BYTE
UBYTE	M7DT_CHAR
UWORD	M7DT_WORD
SWORD	M7DT_INT
UDWORD	M7DT_DWORD
SDWORD	M7DT_DINT
REAL	M7DT_REAL
UBYTE	M7DT_OCTET

Bausteinyp-kennzeichen

Die Kennzeichen in der Tabelle spezifizieren die möglichen Bausteintypen die im Arbeitsspeicher einer S7–CPU bzw eines M7 abgelegt werden können. Die Kennzeichen werden vom den M7–Aufrufen an das OVS (Objektverwaltungssystem) verwendet.

Tabelle 2-10 Bausteintypkennzeichen

Bausteintyp	Typkennzeichen	Bemerkung
Organisationsbaustein OB	M7BLKTYP_OB	nur S7–CPU
Datenbaustein	M7BLKTYP_DB	M7 und S7–CPU
Funktionsaufruf	M7BLKTYP_FC	nur S7–CPU

Tabelle 2-10 Bausteintypkennzeichen

Bausteintyp	**Typkennzeichen**	**Bemerkung**
Systemfunktionsaufruf	M7BLKTYP_SFC	nur S7–CPU
Funktionsbaustein	M7BLKTYP_FB	nur S7–CPU
Systemfunktionsbaustein	M7BLKTYP_SFB	nur S7–CPU

Datenstrukturen

3

Kapitelübersicht

3.1 Datentypen des RMOS-API

Hinweise

In der Header–Datei **RMTYPES.H** des RMOS–API werden die folgenden allgemeinen Datentypen definiert. Diese Datentypen sollten bei den entsprechenden RMOS–API–Aufrufen anstelle der allgemeinen C–Datentypen verwendet werden.

Tabelle 3-1 Allgemeine Datentyp–Definitionen des RMOS–API

Bezeichner	Typdefinition	Bedeutung
uchar	unsigned char	vorzeichenloser Character (Wertebereich: 0 ... 255)
ushort	unsigned short	vorzeichenlose 16 Bit Ganzzahl (Wertebereich: 0 ... 65 535)
uint	unsigned int	vorzeichenlose 32 Bit Ganzzahl (Wertebereich: 0 ... 2^32 – 1)
ulong	unsigned long	vorzeichenlose 32 Bit Ganzzahl (Wertebereich: 0 ... 2^32 – 1)
rmproc	void(*rmproc)(void)	Zeiger auf Funktion ohne Übergabe– und Rückgabeparameter

3.2 Datenstrukturen des RMOS-API

Hinweise

In der Header–Datei **RMTYPES.H** des RMOS–API werden die folgenden allgemeinen Datenstrukturen definiert. Diese Datenstrukturen finden bei den entsprechenden RMOS–API–Aufrufen Verwendung.

Rm3964InitStruct

Syntax

```
#include <drv3964.h>
typedef struct tagRm3964InitStruct
 {
   ushort irq;
   ushort base;
   ulong mode_baud;
   uchar mode_parity;
   uchar mode_data;
   uchar mode_stop;
   uchar mode_fill;
   int prot3964r;
   int master;
 } Rm3964InitStruct;
```

Beschreibung

Die Struktur `Rm3964InitStruct` enthält die Konfigurationsdaten um eine Unit für eine 3964(R)–Kommunikation zu initialisieren. Die Konfiguration wird mit der `RmIOControl` Steuerfunktion RM_IOCTL_INIT durchgeführt.

Feld	Typ	Bedeutung
irq	ushort	IRQ–Nummer der Schnittstelle (z.B. 4 für COM1) Die Angabe von IRQ nur bei der ersten Initialisierung der Unit ausgewertet. Bei weiteren Aufrufen der Steuerfunktion RM_IOCTL_INIT wird sie ignoriert.
base	ushort	I/O Basisadresse des 8250–Bausteins (z.B. 0x3F8 für COM1) Die Basisadresse wird nur bei der ersten Initialisierung der Unit ausgewertet. Bei weiteren Aufrufen der Steuerfunktion RM_IOCTL_INIT wird sie ignoriert.
mode_baud	ulong	Baudrate (Zahlenwert, z.B. 19200)
mode_parity	uchar	Steuerung des Parity–Bit. Folgende Angaben sind zulässig: RM_IOCTL_MODE_PARITYNONE – keine Paritäts–Prüfung RM_IOCTL_MODE_PARITYEVEN – gerade Parität RM_IOCTL_MODE_PARITYODD – ungerade Parität RM_IOCTL_MODE_PARITY0 – Paritäts–Bit fest auf 0 RM_IOCTL_MODE_PARITY1 – Paritäts–Bit fest auf 1
mode_data	uchar	Anzahl der Daten–Bits (mögliche Werte: 5,6,7,8)

Feld	Typ	Bedeutung
mode_stop	uchar	Anzahl der Stop–Bits. RM_IOCTL_MODE_STOP1 1 Stop–Bit RM_IOCTL_MODE_STOP2 2 Stop–Bit RM_IOCTL_MODE_STOP15 1,5 Stop–Bit
mode_fill	uchar	reserviert
prot3964r	int	Protokollauswahl 0 3964–Protokoll 1 3964R–Protokoll
master	int	Festlegung ob Master 0 Slave 1 Master

Beispiel

```
int                                 iostatus;
int                                 status;
Rm3964InitStruct    parameter;
parameter.irq       = 4;
parameter.base      = 0x3F8;
parameter.mode_baud                 = 19200;
parameter.mode_parity               = RM_IOCTL_MODE_PARITY-
NONE;
parameter.mode_data                 = 8;
parameter.mode_stop                 = RM_IOCTL_MODE_STOP1;
parameter.prot3964r                 = 1;
parameter.master    = 1;
status = RmIOControl(               RM_WAIT, 0, handle,
RM_IOCTL_INIT,
                                    &parameter, &iostatus);
```

Siehe auch

RmIOControl

RmAbsTimeStruct

Syntax

```
#include <rmtypes.h>
typedef struct _RmAbsTimeStruct
{
     ulong lotime;
    ulong hitime;
}RmAbsTimeStruct;
```

Beschreibung

Enthält die absolute Systemzeit seit dem letzten Neustart in Millisekunden und wird von `RmGetAbsTime` benutzt.

Feld	Typ	Bedeutung
`lotime`	ulong	Niederwertiger Teil der absoluten Zeit
`hitime`	ulong	Höherwertiger Teil der absoluten Zeit

Siehe auch

RmGetAbsTime

RmEntryStruct

Syntax

```
#include <rmtypes.h>
typedef struct _RmEntryStruct
 {
   uchar slen;
   char string[16];
   uchar type;
   ulong ide;
   ushort id;
 }RmEntryStruct;
```

Beschreibung

Die Struktur **`RmEntryStruct`** wird bei den RMOS–API–Aufrufen `RmList` und `RmGetEntry` verwendet, um Einträge aus dem Betriebsmittelkatalog auszulesen.

Feld	Typ	Bedeutung
slen	uchar	Länge der nachfolgenden Zeichenkette.
string	char[16]	Zeichenkette, die den Namen einer Ressource enthält.
type	uchar	Gibt den Typ der Ressource an. Folgende Werte sind möglich: Wert Define — Bedeutung 0 RM_CATALOG_TASK — Task 1 RM_CATALOG_DEVICE — Gerätetreiber (Device) 2 RM_CATALOG_POOL — Speicherpool 3 RM_CATALOG_SEMAPHORE — Semaphor 4 RM_CATALOG_EVENTFLAG — Globales Ereignisflag 5 RM_CATALOG_CNTRL — Überwachter Programmzugang 6 RM_CATALOG_LOCALMAILBOX — Lokale Mailbox 7 RM_CATALOG_MISC — Reserviert 8 RM_CATALOG_USER — Anwenderdefinierter Typ 10 RM_CATALOG_UNIT — Gerät (Unit) 11 RM_CATALOG_MESSAGE — Messages 255 RM_CATALOG_ALL

Feld	Typ	Bedeutung
ide	ulong	Gibt die erweiterte ID der Resource an. Der Wertebereich hängt vom Typ und von den konfigurierten Maximalwerten ab.
id	ushort	Gibt die ID der Resource an. Der Wertebereich hängt vom Typ und von den konfigurierten Maximalwerten ab.

Hinweis

Der Ressource-Typ RM_CATALOG_USER ist nicht für bestimmte RMOS-Ressourcen reserviert und kann vom Anwender für beliebige Zwecke verwendet werden. Beispielsweise könnte die Verfügbarkeit bestimmter Bibliotheksmodule durch Katalogisieren unter dem Bibliotheksnamen und dem Typ RM_CATALOG_USER angezeigt werden.

Siehe auch

RmCatalog, RmGetEntry, RmGetName, RmList

RmIntrhandMailStruct

Syntax

```
#include <rmtypes.h>
typedef struct _RmIntrhandMailStruct
 {
   uint int_no ;
   uint int_vec :8 ;
   uint int_kind :1 ;
   uint lost_int_overflow :1 ;
   uint dummy_2 :22 ;
   ushort lost_int;
   ushort dummy_3;
 }RmIntrhandMailStruct;
```

Beschreibung

Mit dem RMOS–API–Aufruf `RmSetIntMailboxHandler` können Interrupthandler zum Senden einer Botschaft an eine Mailbox definiert werden. Die Struktur **`RmIntrhandMailStruct`** definiert das Format dieser Botschaft, die nach Auslösen des zugehörigen Interrupts in der Mailbox abgelegt wird. Die Struktur umfaßt insgesamt drei 32-Bit-Worte.

Feld	Typ	Bedeutung
int_no	uint	Bezeichnet die laufende Nummer des Interrupts.
int_vec	8 bit	Gibt den zugehörigen Interruptvektor an.
int_kind	1 bit	Kennzeichnet die Art des Interrupts: Wert Bedeutung 0 Hardwareinterrupt 1 Softwareinterrupt
lost_int_ overflow	1 bit	Dieses Bit ist gesetzt (= 1), wenn Interrupts verloren gingen.
dummy_2	22 bit	Reserviert
lost_int	ushort	Gibt die Anzahl der verlorenen Interrupts an.
dummy_3	ushort	Reserviert

Siehe auch

RmSetIntMailboxHandler

RmIOCTLModeSerialStruct

Syntax

```
#include <rmapi.h>
typedef struct tagRmIOCTLModeSerialStruct
 {
   ulong baud;
   uchar parity;
   uchar data;
   uchar stop;
 }RmIOCTLModeSerialStruct;
```

Beschreibung

Die Struktur `RmIOCTLModeSerialStruct` enthält die Konfigurationsdaten für Treiber für serielle Schnittstellen (z.B. 8250). Sie wird bei der Steuerfunktion RM_IOCTL_MODE benötigt, um die Unit neu zu konfigurieren.

Feld	Typ	Bedeutung
baud	ulong	Übertragungsrate (Zahlenwert, z.B. 19200)
parity	uchar	Steuerung des Parity–Bit. Folgende Angaben sind zulässig: RM_IOCTL_MODE_PARITYNONE keine Paritäts–Prüfung RM_IOCTL_MODE_PARITYEVEN gerade Parität RM_IOCTL_MODE_PARITYODD ungerade Parität RM_IOCTL_MODE_PARITY0 Paritäts–Bit fest auf 0 RM_IOCTL_MODE_PARITY1 Paritäts–Bit fest auf 1
data	uchar	Anzahl der Daten–Bits (Zahlenwert, z.B. 8)
stop	uchar	Anzahl der Stop–Bits. Folgende Angaben sind zulässig: RM_IOCTL_MODE_STOP1 1 Stop–Bit RM_IOCTL_MODE_STOP2 2 Stop–Bit RM_IOCTL_MODE_STOP15 1,5 Stop–Bit

Beispiel

```
int                 iostatus;
int                 status;
RmIOCTLModeSerialStruct         param;
param.baud          = 19200ul;
param.parity        = RM_IOCTL_MODE_PARITYNONE;
param.data          = 8;
param.stop          = RM_IOCTL_MODE_STOP1;
status = RmIOControl(RM_WAIT, 0, handle, RM_IOCTL_MODE,
                     (void *) &param,  &iostatus);
```

Siehe auch

RmIOControl

RmIOCTLPropertiesStruct

Syntax

```
#include <rmapi.h>
typedef struct tagRmIOCTLPropertiesStruct
 {
   uint block_device : 1;
   uint convert : 1;
   uint protocol : 1;
   uint terminal : 1;
   uint hsfs : 1;
   uint serial : 1;
   uint buffer : 1;
   uint reserved1 : 9;
   uint reserved2 : 16;
   uint ioctl_lock : 1;
   uint ioctl_get_status : 1;
   uint ioctl_verify : 1;
   uint ioctl_linemode : 1;
   uint ioctl_readterm : 1;
   uint ioctl_writeterm : 1;
   uint ioctl_readstop : 1;
   uint ioctl_writestop : 1:
   uint ioctl_readtout : 1:
   uint ioctl_writetout : 1;
   uint ioctl_echo : 1;
   uint ioctl_line_feed : 1;
   uint ioctl_form_feed : 1;
   uint ioctl_abortchar : 1;
   uint ioctl_terminal : 1;
   uint reserved3 : 1;
   uint reserved4 : 16;
   ulong block_size;
   ulong number_of_blocks;
   ulong reserved5;
   ulong reserved6;
   ulong reserved7;
 }RmIOCTLPropertiesStruct ;
```

Beschreibung

Die Struktur `RmIOCTLPropertiesStruct` enthält Informationen über den Funktionsumfang des ladbaren Treibers.

Feld	Typ	Bedeutung
block_device	1 Bit	Typ des Treibers 0: Zeichenorientierter Treiber 1: Blockorientierter Treiber
convert	1 Bit	reserviert

Feld	Typ	Bedeutung
protocol	1 Bit	Protokolltreiber (z.B. 3964R) 1 = ja, 0 = nein
terminal	1 Bit	Terminaltreiber 1 = ja, 0 = nein
hsfs	1 Bit	Massenspeichertreiber (z.B. für Hard–Disk) 1 = ja, 0 = nein
serial	1 Bit	Treiber für serielle Schnittstelle 1 = ja, 0 = nein
buffer	1 Bit	Hintergrundpuffer vorhanden? 1 = ja, 0 = nein
reserved1	9 Bit	reserviert
reserved2	16 Bit	reserviert
ioctl_lock	1 Bit	Lock–Funktion (RM_IOCTL_LOCK) vorhanden 1 = ja, 0 = nein
ioctl_get_status	1 Bit	RM_IOCTL_GET_STATUS vorhanden 1 = ja, 0 = nein
ioctl_verify	1 Bit	Verify–Funktion (RM_IOCTL_VERIFY_ON / OFF) 1 = ja, 0 = nein
ioctl_linemode	1 Bit	Zeilenorientiertes Lesen (RM_IOCTL_LINEMODE_ON / OFF) 1 = ja, 0 = nein
ioctl_readterm	1 Bit	Terminatorzeichen für Lesen (RM_IOCTL_READTERM_ON / OFF) 1 = ja, 0 = nein
ioctl_writeterm	1 Bit	Terminatorzeichen für Schreiben (RM_IOCTL_WRITETERM_ON / OFF) 1 = ja, 0 = nein
ioctl_readstop	1 Bit	Stopzeichen für Lesen (RM_IOCTL_READSTOP) und maximale Anzahl Zeichen (RM_IOCTL_READLEN) 1 = ja, 0 = nein
ioctl_writestop	1 Bit	Stopzeichen für Schreiben (RM_IOCTL_WRITESTOP) 1 = ja, 0 = nein
ioctl_readtout	1 Bit	Timeout für Lesen (RM_IOCTL_READTIMEOUT) 1 = ja, 0 = nein
ioctl_writetout	1 Bit	Delay für Schreiben (RM_IOCTL_WRITEDELAY) 1 = ja, 0 = nein
ioctl_echo	1 Bit	Echo–Funktion ein/ausschalten (RM_IOCTL_ECHO_ON / OFF) 1 = ja, 0 = nein

Feld	Typ	Bedeutung
ioctl_line_feed	1 Bit	Zeilenvorschub (RM_IOCTL_LINE_FEED) 1 = ja, 0 = nein
ioctl_form_feed	1 Bit	Seitenvorschub (RM_IOCTL_FORM_FEED) 1 = ja, 0 = nein
ioctl_abort_char	1 Bit	Abbruchzeichen (RM_IOCTL_ABORTCHAR_ON / OFF) 1 = ja, 0 = nein
ioctl_terminal	1 Bit	Umschalten Terminal–/Transparent–Mode (RM_IOCTL_TERMINAL_ON / OFF) 1 = ja, 0 = nein
reserved3	1 Bit	reserviert
reserved4	16 Bit	reserviert
block_size	ulong	Blockgröße (in Byte) bei blockorientierten Treibern
number_of_blocks	ulong	Anzahl der Blöcke bei blockorientierten Treibern
reserved5	ulong	reserviert
reserved6	ulong	reserviert
reserved7	ulong	reserviert

Siehe auch **RmIOControl**

RmIOCTLVersionStruct

Syntax

```
#include <rmapi.h>
typedef struct tagRmIOCTLVersionStruct
 {
   int MajorVersion;
   int MinorVersion;
   int DriverInfo1;
   int DriverInfo2;
   char Name[RM_MAXCATALOGLEN+1];
 }RmIOCTLVersionStruct;
```

Beschreibung

Die Struktur `RmIOCTLVersionStruct` dient zur Ermittlung der Version eines ladbaren Treibers.

Feld	Typ	Bedeutung
MajorVersion	int	Version des Treibers (Wert vor dem Punkt) z.B. für Version 1.0 ist MajorVersion 1
MinorVersion	int	Version des Treibers (Wert nach dem Punkt) z.B. für Version 1.0 ist MinorVersion 0
DriverInfo1	int	Treiberabhängige Informationen (Bei SER8250.DRV und 3964.DRV immer 0)
DriverInfo2	int	Treiberabhängige Informationen (Bei SER8250.DRV und 3964.DRV immer 0)
Name	char Array	Name des Treibers, mit dem er im Betriebsmittelkatalog eingetragen ist (SER8250 bzw. 3964).

Siehe auch

RmIOControl

RmMailboxStruct

Syntax

```
#include <rmtypes.h>
typedef struct _RmMailboxStruct
 {
   void *adr;
   ushort adr_res;
   ushort pad;
   uint len;
 }RmMailboxStruct;
```

Beschreibung

`RmMailboxStruct` wird verwendet, um eine Nachricht per Mailbox indirekt zu senden, indem in der Mailbox nicht die Nachricht selbst, sondern die Speicheradresse und Länge der Nachricht übergeben werden.

Feld	Typ	Bedeutung
adr	void *	Enthält einen Pointer auf die Speicheradresse der Nachricht.
adr_res	ushort	Füllwort bei FLAT-Modell
pad	ushort	Wird auf 64 Bit aufgefüllt.
len	uint	Gibt die Länge der Nachricht an.

Siehe auch

RmReceiveMail, RmSendMail

RmMailIDStruct

Syntax

```
#include <rmtypes.h>
typedef struct _RmMailIDStruct
 {
   ulong low;
   ulong high;
 }RmMailIDStruct;
```

Beschreibung

Rückgabewert der Funktion `RmSendMailDelayed`. Dieser Rückgabewert wird beispielsweise zum Löschen einer verzögert abgeschickten Mail benötigt.

Feld	Typ	Bedeutung
low	ulong	Niederwertiger Teil der Mail-ID
high	ulong	Höherwertiger Teil der Mail-ID

Siehe auch

RmSendMailCancel, RmSendMailDelayed

RmMemPoolInfoStruct

Syntax

```
#include <rmtypes.h>
typedef struct _RmMemPoolInfoStruct
 {
   ulong pool_size;
   ulong avail_mem_size;
   ulong max_block_size;
   ulong reserved[5]
 }RmMemPoolInfoStruct;
```

Beschreibung

Rückgabewert der Funktion `RmGetMemPoolInfo`. Der Rückgabewert enthält Informationen zum angegebenen Speicherpool.

Feld	Typ	Bedeutung
pool_size	ulong	Gesamtgröße des Speicherpools
avail_mem_size	ulong	Gesamtgröße des verfügbaren Speichers
max_block_size	ulong	Größe des größten verfügbaren Speicherblocks (immer –1)

Siehe auch

RmGetMemPoolInfo

Ser8250InitStruct

Syntax

```
#include <ser8250.h>
typedef struct tagSer8250InitStruct
 {
   ushort irq;
   ushort base;
   ulong mode_baud;
   uchar mode_parity;
   uchar mode_data;
   uchar mode_stop;
   uchar mode_fill;
   ulong buffer_size;
 } Ser8250InitStruct;
```

Beschreibung

Die Struktur `Ser8250InitStruct` enthält die Konfigurationsdaten, um eine Unit für den Treiber einer seriellen Schnittstelle zu initialisieren. Die Konfiguration wird mit der `RmIOControl` Steuerfunktion RM_IOCTL_INIT durchgeführt.

Feld	Typ	Bedeutung
irq	ushort	IRQ–Nummer der Schnittstelle (z.B. 4 für COM1) Die Angabe von IRQ nur bei der ersten Initialisierung der Unit ausgewertet. Bei weiteren Aufrufen der Steuerfunktion RM_IOCTL_INIT wird sie ignoriert.
base	ushort	I/O Basisadresse des 8250–Bausteins (z.B. 0x3F8 für COM1) Die Basisadresse wird nur bei der ersten Initialisierung der Unit ausgewertet. Bei weiteren Aufrufen der Steuerfunktion RM_IOCTL_INIT wird sie ignoriert.
mode_baud	ulong	Baudrate (Zahlenwert, z.B. 19200)
mode_parity	uchar	Steuerung des Parity–Bit. Folgende Angaben sind zulässig: RM_IOCTL_MODE_PARITYNONE keine Paritäts–Prüfung RM_IOCTL_MODE_PARITYEVEN gerade Parität RM_IOCTL_MODE_PARITYODD ungerade Parität RM_IOCTL_MODE_PARITY0 Paritäts–Bit fest auf 0 RM_IOCTL_MODE_PARITY1 Paritäts–Bit fest auf 1
mode_data	uchar	Anzahl Daten–Bits (Mögliche Werte 5,6,7,8)
mode_stop	uchar	Anzahl der Stop–Bits. RM_IOCTL_MODE_STOP1 1 Stop–Bit RM_IOCTL_MODE_STOP2 2 Stop–Bit RM_IOCTL_MODE_STOP15 1,5 Stop–Bit

Feld	Typ	Bedeutung
mode_fill	uchar	wird ignoriert
buffer_size	ulong	Größe des Hintergrundpuffers des Treibers (Anzahl Zeichen)

Beispiel

```
int                 iostatus;
int                 status;
Ser8250InitStruct   parameter;
parameter.irq       = 4;
parameter.base      = 0x3F8;
parameter.mode_baud                 = 19200;
parameter.mode_parity               = RM_IOCTL_MODE_PARITY-
NONE;
parameter.mode_data                 = 8;
parameter.mode_stop                 = RM_IOCTL_MODE_STOP1;
parameter.buffer_size               = 256;
status = RmIOControl(               RM_WAIT, 0, handle,
RM_IOCTL_INIT,
                                    &parameter, &iostatus);
```

Siehe auch

RmIOControl

STDSTRUCT

Syntax

```
#include <task.h>
struct              std_struct
 {
   int    stdin_dev;
   int    stdin_unit;
   int    stdout_dev;
   int    stdout_unit;
   int    stderr_dev;
   int    stderr_unit;
   char   *stdin_fname;
   unsigned short  stdin_fill;
   char   *stdout_fname;
   unsigned short  stdout_fill;
   char   *stderr_fname;
   unsigned short  stderr_fill;
   char   *tmp_path;
   unsigned short  tmp_fill;
  };

typedef struct std_struct  STDSTRUCT;
```

Beschreibung

Die Struktur **STDSTRUCT** definiert die Ein- und Ausgabekanäle **stdin**, **stdout** und den Fehler-Ausgabekanal **stderr** eines Programms. Ein Kanal kann entweder durch die Angabe einer Device/Unit Nummernkombination oder durch einen Dateinamen bestimmt werden.

Feld	Typ	Bedeutung
stdxx_dev	int	Ein Wert >= 0 bestimmt die Nummer eines E/A-Treibers (Device-Nummer). Werte < 0 haben folgende Bedeutung: Wert — Bedeutung -1 — Es wird der in stdxx_fname angegebene Dateiname verwendet. Im Fall von stdout und stderr wird eine neue Datei angelegt. stdxx_unit wird nicht verwendet. -2 — Es wird der in stdxx_fname angegebene Dateiname verwendet. Im Fall von stdout und stderr werden die Ausgaben am Ende der Datei angefügt, falls sie schon existiert. stdxx_unit wird nicht verwendet. -3 — Anwender sollten diesen Wert genauso wie den Wert stdxx_dev = -2 behandeln, da er lediglich für das interaktive CLI-Kommando START folgende Bedeutung hat: Die Ausgabedatei wurde vom aufrufenden Job geerbt und darf nicht weitervererbt werden.

Feld	Typ	Bedeutung
stdxx_unit	int	Wenn stdxx_dev einen Wert >= 0 enthält, bestimmt stdxx_unit die Nummer eines E/A-Geräts (Unit-Nummer). Hat stdxx_dev einen Wert < 0, ist stdxx_unit bedeutungslos.
stdxx_fname	char *	Zeigt auf eine Dateinamen-Zeichenkette. Die mit dem Dateinamen bezeichnete Datei wird verwendet, wenn stdxx_dev wie oben beschrieben einen Wert <0 enthält.
stdxx_fill	unsigned short	Reserviert, Füllwort bei FLAT–Modell
tmp_path	char *	Zeigt auf eine Dateinamen-Zeichenkette, die eine Datei für temporäre Daten bestimmt.
tmp_fill	unsigned short	Reserviert, Füllwort bei FLAT–Modell

Hinweis

Die oben beschriebenen Werte -2 und -3 für **`stdxx_dev`** haben nur für den CLI eine Bedeutung. Die Funktion `x_cr_gettaskparam` liefert immer Werte **`stdxx_dev`** >= -1.

Der durch das Feld **`tmp_path`** bestimmte Dateiname ist identisch mit dem bei der Funktion `xinitt` angegebenen Namen für die temporäre Datei.

3.3 Datentypen des M7-API

3.3.1 Allgemeine Datentypen des M7-API

Hinweise

In der Header–Datei **M7API.H** des M7–API werden die folgenden allgemeinen Datentypen definiert. Diese Datentypen sollten bei den entsprechenden M7–API–Aufrufen anstelle der allgemeinen C–Datentypen verwendet werden.

Die folgende Tabelle listet die Bezeichner der M7–Basisdatentypen auf, die in der M7–API–Umgebung verwendet werden. Ihre Definitionen finden sich in der Headerdatei **M7API.H**.

Tabelle 3-2 Allgemeine Datentypen des M7–API

Bezeichner	Typdefinition	Bedeutung
UBYTE	unsigned char	vorzeichenloserCharacter (Wertebereich: 0 ... 255)
UWORD	unsigned short	vorzeichenlose 16 Bit Ganzzahl (Wertebereich: 0 .. 65 535)
UDWORD	unsigned long	vorzeichenlose 32 Bit Ganzzahl (Wertebereich: 0.. 2^32 – 1)
SBYTE	signed char	vorzeichenbehafteter Character (Wertebereich: –128..127)
SWORD	signed short	vorzeichenbehaftete 16 Bit Ganzzahl (Wertebereich: –32 768...32 767)
SDWORD	signed long	vorzeichenbehaftete 32 Bit Ganzzahl (Wertebereich: –2^31..2^31 – 1)
BOOL	unsigned int	Boolscher Wert
REAL	float	32 Bit Gleitpunktzahl
BYTE	UBYTE	vorzeichenloser Character (Wertebereich: 0...255)
UBYTE_PTR	UBYTE *	Zeiger auf UBYTE
WORD	UWORD	vorzeichenlose 16 Bit Ganzzahl (Wertebereich: 0...32 767)
DWORD	UDWORD	vorzeichenlose 32 Bit Ganzzahl (Wertebereich: 0... 2^32 – 1)
M7ERR_CODE	int	Fehlerrückgabewert

Tabelle 3-2 Allgemeine Datentypen des M7–API

Bezeichner	Typdefinition	Bedeutung
M7ERR_CODE_PTR	M7ERR_CODE *	Zeiger auf M7ERR_CODE–Variable
M7IO_LOGADDR	UWORD	Logische Adresse eines Signals
M7IO_BASEADDR	UWORD	Basisadresse einer I/O–Baugruppe
M7CONNID	UWORD	ID einer Applikations–beziehung

3.3.2 FRB-Datentypen der M7-Server

Hinweise

In der Header–Datei **M7API.H** des M7–API werden die folgenden FRB–Strukturen (Function Request Block) definiert. Die FRBs werden beim Anmelden bei den entsprechenden M7–Servern benötigt. Die folgende Tabelle listet die FRB–Strukturen nebst den zugehörigen Pointer–Definitionen.

Der Zugriff auf Informationen der entsprechenden FRBs erfolgt ausschließlich über Makros, die ebenfalls in der Header–Datei **M7API.H** definiert sind.

Tabelle 3-3 FRB–Definitionen des M7–API

Typdefinition	Bedeutung
M7FRBHEADER	Header eines jeden FRBs. Enthält allgemeine Verwaltungsinformationen
M7FRBHEADER_PTR	Zeiger auf einen FRB–Header
M7CBFRB	FRB zum Anmelden einer Callback–Funktion beim S7–Objektserver
M7CBFRB_PTR	Zeiger auf einen FRB vom Typ M7CFRB
M7OBJFRB	FRB zum Anmelden der Zugriffsbe–nachrichtigung durch den S7–Objekt–Server
M7OBJFRB_PTR	Zeiger auf einen FRB vom Typ M7OBJFRB
M7IOALARM_FRB	FRB zur Anmeldung der Benachrichti–gung bei I/O–Alarm durch den Alarm–Server
M7IOALARM_FRB_PTR	Zeiger auf einen FRB vom Typ M7IOALARM_FRB
M7DIAGALARM_FRB	FRB zur Anmeldung der Benachrichti–gung bei Diagnosealarm durch den Alarm–Server
M7DIAGALARM_FRB_PTR	Zeiger auf einen FRB vom Typ M7SDIAGALARM_FRB

Tabelle 3-3 FRB–Definitionen des M7–API

Typdefinition	Bedeutung
M7ZSALARM_FRB	FRB zur Anmeldung der Benachrichti gung bei Ziehen/Stecken–Alarm durch den Alarm–Server
M7ZSALARM_FRB_PTR	Zeiger auf einen FRB vom Typ M7ZSA-LARM_FRB
M7TFRB	FRB zur Anmeldung der Benachrichtigung bei Zeitereignissen durch den Time–Server
M7TFRB_PTR	Zeiger auf einen FRB vom Typ M7TFRB
M7TSFRB	FRB zur Anmeldung der Benachrichtigung bei neuen Betriebszuständen bzw. Betriebszustands-übergangen durch den BZ–Server
M7TSFRB_PTR	Zeiger auf einen FRB vom Typ M7TSFRB
M7FSCFRB	FRB zur Anmeldung der Benachrichtigung: Freier Zyklus, Zykluskontrollpunt, ANLAUF und Zy-kluszeitüberwachung durch den FZ–Server
M7FSCFRB_PTR	Zeiger auf einen FRB vom Typ M7FSCFRB
M7COMMFRB	Wird benötigt beim Aufruf einseitiger PBK–Funk-tionen
M7COMMFRB_PTR	Zeiger auf einen FRB vom Typ M7COMMFRB

3.3.3 Sonstige Datentypen der M7-Server

Hinweise

In der Tabelle sind weitere Datentypen des M7–API aufgelistet. Die darunterliegende Struktur wird nicht näher erläutert, da der Zugriff auf die einzelnen Einträge ausschließlich über Makros erfolgt.

Tabelle 3-4 Sonstige Datentypen des M7–API

Typdefinition	Bedeutung
M7IO_DESC	Datenstruktur zur Aufnahme der Deskriptorinformationen für den Zugriff auf ISA–Module
M7IO_DESC_PTR	Zeiger auf einen ISA–Modul–Deskriptor

3.4 Datenstrukturen des M7-API

Hinweise

In der Header–Datei M7API.H des M7–API werden die folgenden allgemeinen Datenstrukturen definiert. Diese Datenstrukturen finden bei den entsprechenden M7–API–Aufrufen Verwendung.

M7BLKINFO

Syntax

```
#include <m7api.h>
typedef struct tagM7BlkInfo
 {
   UWORD Language
   UWORD Blktyp;
   UWORD Blknum;
   UBYTE Bitmap;
   UBYTE filler;
 }M7BLKINFO;

typedef M7BLKINFO * M7BLKINFO_PTR
```

Beschreibung

Die Struktur **M7BLKINFO** wird von OVS–Funktionen beim Auslesen des Bausteinverzeichnisses einer S7–CPU oder eines M7 verwendet. In der Struktur wird vom Aufruf Information über einen Baustein zurückgegeben.

Feld	Typ	Bedeutung
Language	UWORD	Das Feld liefert die Kennung der Sprache, in der ein Baustein erstellt worden ist, aus dem Kopf eines Bausteins.
Blktyp	UWORD	Bausteintyp: Die Kennungen der möglichen Bausteintypen sind in Tabelle 2-10 aufgelistet
Blknum	UWORD	Nummer des Bausteins
Bitmap	UBYTE	Die einzelnen Bits können mit Hilfe vorderfinierten Konstanten ”verundet” und auf ungleich Null geprüft werden. M7BLKINFO_PASSIV Baustein ist kopiert (passiv), d.h. im temporären Ladespeicher M7BLKINFO_ACTIVE Baustein ist eingekettet (aktiv), d.h. im Arbeitsspeicher M7BLKINFO_RAM Baustein ist im RAM–Speicher bzw. im RAM–Modus M7BLKINFO_EPROM Baustein ist im EPROM bzw. im EPROM–Modus M7BLKINFO_BESY Baustein ist im Betriebssystem
filler	UBYTE	reserviert

Siehe auch

M7OVSFindFirst, M7OVSFindNext

M7BLKLIST

Syntax

```
#include <m7api.h>
typedef struct tagM7BlkList
 {
   UWORD Blktyp;
   UWORD Blknum;
 }M7BLKLIST;

typedef M7BLKLIST * M7BLKLIST_PTR
```

Beschreibung

Die Struktur **M7BLKLIST** wird von OVS–Funktionen zum simultanen Einketten oder Löschen von mehreren Bausteinen verwendet.

Feld	Typ	Bedeutung
Blktyp	UWORD	Typ des Bausteins. Die Kennungen der möglichen Bausteintypen sind in Tabelle 2-10 aufgelistet
Blknum	UWORD	Nummer des Bausteins.

Siehe auch

M7OVSDelete, M7OVSLinkIn

M7CBRet

Syntax

```
#include <m7api.h>
typedef struct tagM7CBRet
 {
   UBYTE process;
   UBYTE result;
   UBYTE errcls;
   UBYTE errcode;
 }M7CBRet;
```

Beschreibung

Die Struktur **M7CBRet** muß von einer Callback–Funktion, die über den Aufruf `M7LinkDataAccessCB` von einer Task angemeldet wurde, im Return–Wert an das M7–API zurückgegeben werden.

In diesem Return–Wert bestimmt die Callback–Funktion, ob eine weitere Bearbeitung durch den S7–Objekt–Server gewünscht wird oder nicht.

Feld	Typ	Bedeutung
process	UBYTE	TRUE: Objektserver führt weitere Arbeit durch FALSE: Arbeit durch die Callback–Funktion erfüllt
result	UBYTE	Fehlernummer im Falle von `process`=FALSE
errcls	UBYTE	nicht relevant
errcode	UBYTE	nicht relevant

Hinweis

Abarbeitung durch den ObjektServer erfolgt sowohl bei `process`=FALSE als auch bei result ≠ 0.

Siehe auch

M7LinkDataAccessCB

M7KTIME

Syntax

```
#include <m7api.h>
typedef struct tagM7KTime
 {
   UWORD TimeState;
   UBYTE Year;
   UBYTE Month;
   UBYTE Day;
   UBYTE Hour;
   UBYTE Minute;
   UBYTE Second;
   unsigned int m_sec_10:4;
   unsigned int m_sec_100:4;
   unsigned int Weekday:4;
   unsigned int m_sec_1:4;
 }M7KTIME;

typedef M7KTIME * M7KTIME_PTR
```

Beschreibung

Die Struktur **M7KTIME** wird von den zugehörigen M7–API–Funktionen zum Auslesen und Setzen der Zeit am K–BUS verwendet.

Feld	Typ	Bedeutung
TimeState	UWORD	Status der Uhrzeit. Verundung von `TimeState` mit Hilfe der folgenden vordefinierten Konstanten und Auswertung auf ungleich Null ergibt folgende Zustandswerte: M7KTIME_SYA Synchronisation der Uhrzeit ist erfolgt M7KTIME_ESY Ersatzsynchronisation der Uhrzeit ist am LAN erfolgt M7KTIME_UZS Uhrzeitsprung ist erfolgt M7KTIME_ZNA Zeitwert ist nicht aktuell M7KTIME_KMASK Maske für den Korrekturwert für Sommer–, Winter– und Weltzeit in 1/2 Stunden
		Verundung von `TimeState` mit Hilfe der Maske M7KTIME_UA_MASK und anschließendem Vergleich auf Gleichheit mit den folgenden Konstanten, ergibt die Auflösung der Uhrzeit:

Feld	Typ	Bedeutung
		M7KTIME_UA_M_SEC_1 Auflösung 1 msec M7KTIME_UA_M_SEC_10 Auflösung 10 msec M7KTIME_UA_M_SEC_100 Auflösung 100 msec M7KTIME_UA_SECOND Auflösung 1 sec
Year	UBYTE	Angabe des Jahrs: 00 ... 99 (BCD–Zahl)
Month	UBYTE	Angabe des Monats: 01 ... 12 (BCD–Zahl)
Day	UBYTE	Angabe des Tags: 01 ... 31 (BCD–Zahl)
Hour	UBYTE	Angabe der Stunden: 00 ... 23 (BCD–Zahl)
Minute	UBYTE	Angabe der Minuten: 00 ... 59 (BCD–Zahl)
Second	UBYTE	Angabe der Sekunden: 00 ... 59 (BCD–Zahl)
m_sec_10	unsigned int	Angabe der 1/100–Sekunden: 0 ... 9. Nur beim Lesen der Zeit, beim Schreiben = 0
m_sec_100	unsigned int	Angabe der 1/10–Sekunden: 0 ... 9. Nur beim Lesen der Zeit, beim Schreiben = 0
Weekday	unsigned int	Angabe des Wochentags: 1: Sonntag 2: Montag 3: Dienstag 4: Mittwoch 5: Donnerstag 6: Freitag 7: Samstag
m_sec_1	unsigned int	Angabe der 1/1000–Sekunden: 0 ... 9 Nur beim Lesen der Zeit, beim Schreiben = 0

Siehe auch **M7KReadTime, M7KWriteTime**

M7OBJ_INFO

Syntax

```
#include <m7api.h>
typedef struct tagM7ObjInfo
 {
   UWORD Size;
   UWORD Attrib;
   unsigned long Data;
   UBYTE External;
 }M7OBJ_INFO;

typedef M7OBJ_INFO * M7OBJ_INFO_PTR
```

Beschreibung

Die Struktur **M7OBJ_INFO** wird beim Aufruf `M7GetObjectInfo` verwendet, um Information über ein S7–Objekt zu ermitteln.

Feld	Typ	Bedeutung
Size	UWORD	Länge des S7–Objekts in Bytes.
Attrib	UWORD	Objekteigenschaften 0x00 Objekt vom Anwender allokiert 0x01 Objekt vom Objekt–Server allokiert 0x02 Objekt im SRAM 0x10 Objekt im RAM–Mode 0x20 Objekt im ROM–Mode 0x40 Objekt im BESY–Mode Der Wert von Attrib kann sich auch aus einer Kombination dieser einzelnen Wertezusammensetzen. Beispielsweise bedeutet ein Wert von 0x11, daß sich das S7–Objekt im RAM–Mode befindet und vom Objekt–Server verwaltet wird.
Data	unsigned long	Zeiger auf die Daten des S7–Objekts. Das Strukturelement muß vom Anwender auf den benötigten Pointertyp gecastet werden.
External	UBYTE	TRUE: Speicher für das S7–Objekt wurde von der M7 RMOS32–Task allokiert. FALSE: Speicher fürs S7–Objekt wurde vom S7–Objekt–Server allokiert.

Siehe auch

M7GetObjectInfo

M7PBKSTATUS

Syntax

```
#include <m7api.h>
typedef struct tagM7PBKStatus
 {
   UBYTE LogicalState;
   UBYTE PhysicalState;
   UBYTE LocalSupplement[16];
 }M7PBKSTATUS;

typedef M7PBKSTATUS * M7PBKSTATUS_PTR
```

Beschreibung

Die Struktur wird von der Funktion `M7PBKStatus` zur Ermittlung des Status des Kommunikationspartners benutzt.

Feld	Typ	Bedeutung
LogicalState	UBYTE	Angabe des logischen Status des Kommunikationspartners. Folgende logische Zustände sind möglich: M7LSTATE_OK Betriebszustandsänderungen sind erlaubt
PhysicalState	UBYTE	Angabe des physikalischen Status des Kommunikationspartners. Folgende physikalischen Zustände sind möglich: M7PSTATE_OPERATIONAL Gerät arbeitet M7PSTATE_NEED_SERVICE Gerät benötigt Service
LocalSupplement	UBYTE	Zusatzinformation. Im Byte 0 der Zusatzinformation werden folgende Zustandsdaten übermittelt: M7LSUPPL_STOP: Gerät ist im Betriebszustand STOP M7LSUPPL_START Gerät ist im Betriebszustand NEUSTART M7LSUPPL_RUN Gerät ist im Betriebszustand RUN M7LSUPPL_RESTART Gerät ist im Betriebszustand WIEDERANLAUF M7LSUPPL_HALT Gerät ist im Betriebszustand HALT M7LSUPPL_DEFECT Gerät ist ist außer Betrieb

Siehe auch

M7PBKStatus

M7TIME_DATE

Syntax

```
#include <m7api.h>
typedef struct tagM7Time_Date
 {
   UBYTE Hour;
   UBYTE Minute;
   UBYTE Second;
   UBYTE HSecond;
   UBYTE Day;
   UBYTE Month;
   UWORDYear;
   UBYTE DayOfWeek;
 }M7TIME_DATE;

typedef M7TIME_DATE * M7TIME_DATE_PTR
```

Beschreibung

Die Struktur **M7TIME_DATE** wird von den zugehörigen M7–API–Funktionen zum Auslesen und Setzen der systeminternen Zeit verwendet.

Feld	Typ	Bedeutung
Hour	UBYTE	Angabe der Stunden: 0 ... 23
Minute	UBYTE	Angabe der Minuten: 0 ... 59
Second	UBYTE	Angabe der Sekunden: 0 ... 59
HSecond	UBYTE	Angabe der Sekunden: 0 ... 99 Nur beim Lesen der Zeit
Day	UBYTE	Angabe des Tags: 1 ... 31
Month	UBYTE	Angabe des Monats: 1 ... 12
Year	UWORD	Angabe des Jahrs z.B. 1997
DayOfWeek	UBYTE	Angabe des Wochentags: 0: Sonntag 1: Montag 2: Dienstag 3: Mittwoch 4: Donnerstag 5: Freitag 6: Samstag

Siehe auch

M7GetTime, M7SetTime

M7VARADDR

Syntax

```
#include <m7api.h>
typedef struct tagM7VarAddr
 {
   UBYTE Syntax;
   UBYTE DataType;
   UWORD Length;
   UWORD Part;
   UBYTE Area;
   UBYTE filler;
   UDWORD Offset;
 }M7VARADDR;

typedef M7VARADDR * M7VARADDR_PTR
```

Beschreibung

Die Struktur **M7VARADDR** wird von PBK– und BuB–Funktionen verwendet, um eine fortlaufende Anzahl von Items innerhalb eines S7–Objekts zu adressieren.

Feld	Typ	Bedeutung
Syntax	UBYTE	muß für diese Datenstruktur immer auf Wert: 0x10 gesetzt sein
DataType	UBYTE	Spezifiziert den Datentyp eines Items innerhalb des adressierten S7–Objekts. Die Kennungen für die möglichen M7–Datentypen sind in Tabelle 2-9 aufgelistet.
Length	UWORD	Anzahl der Items. Beim Datentyp M7DT_BOOL ist für den Parameter LENGTH nur 1 zulässig.
Part	UWORD	Spezifiziert die Teilbereichsnummer(DB–Nummer, usw.) eines S7–Objekts. Die möglichen Teilbereichsnummern für die einzelnen S7–Objekte sind in Tabelle 2-8 aufgelistet.
Area	UBYTE	Spezifiziert das Typkennzeichen des S7–Objekts. Die möglichen Typkennzeichen sind in Tabelle 2-7 aufgelistet.
filler	UBYTE	reserviert, muß mit 0x00 besetzt werden.
Offset	UDWORD	Spezifiziert den Adreß–Offset des ersten Items innerhalb des S7–Objekts. Der Adreß–Offset muß immer ein Vielfaches der Bitlänge des spezifizierten Datentyps (siehe: `DataType`) sein. Bei Datensätzen ist im Intel Format in Byte 0 und 1 die logische Baugruppenadresse anzugeben, Byte 2 spezifiziert, ob Eingangs– oder Ausgangsadresse (0 für Eingang, 1 für Ausgang).

Siehe auch

M7BUBCycRead, M7BUBRead, M7BUBWrite, M7PBKBrcv, M7PBKBsend, M7PBKGet, M7PBKPut,

M7VARDATA

Syntax

```
#include <m7api.h>
typedef struct tagM7VarData
 {
   UBYTE_PTR Buffer;
   UDWORD Length;
   UBYTE AccessResult;
   UBYTE DataType;
 }M7VARDATA;

typedef M7VARDATA * M7VARDATA_PTR
```

Beschreibung

Die Struktur **M7VARDATA** wird von BuB–Funktionen verwendet, um einen Puffer zu spezifizieren.

Der spezifizierte Puffer dient entweder für die Aufnahme (lesender Zugriff) der Werte der adressierten Variablen oder für die Daten (schreibender Zugriff), mit denen adressierte Variablen überschrieben werden.

Feld	Typ	Bedeutung
Buffer	UBYTE_PTR	Zeiger auf den eigentlichen Puffer. Der Puffer muß vom Anwenderprogramm entweder im globalen Datenbereich oder auf dem Heap (sonstigem Speicher–Pool) allokiert werden
Length	UDWORD	Länge des Datenpuffers in Anzahl Items
AccessResult	UBYTE	Spezifiziert nach dem Zugriff (Lesen oder Schreiben) das Resultat. Mögliche Fehlerkennungen sind: M7RES_SUCCESS: Transfer erfolgreich durchgeführt M7RES_HWERROR: Hardwarefehler M7RES_NOACCESS: keine Zugriffserlaubnis auf das Objekt M7RES_INVADDR: ungültiges Item im S7–Objekt adressiert M7RES_INVDTYP: ungültiger Datentyp M7RES_NOOBJECT: kein solches Objekt oder ungültige Länge
DataType	UBYTE	Spezifiziert den Datentyp eines Items. Die möglichen Datentypen finden Sie in Tabelle 2-9.

Siehe auch

M7BUBCycRead, M7BUBRead, M7BUBWrite, RmIOControl

3.5 Datenstrukturen der Socket–Schnittstelle

Hinweise

In der Header–Datei **SOCKET.H** werden die folgenden Datenstrukturen definiert. Diese Datenstrukturen finden bei den entsprechenden Aufrufen der Socket–Schnittstelle Verwendung.

HOSTENT

Syntax

```
#include <socket.h>
typedef struct hostent
 {
   char        *h_name;
   char        **h_aliases;
   short       h_addrtype;
   short       h_length;
   char        *h_addr;
 } HOSTENT;
```

Beschreibung

Die Struktur HOSTENT wird in den Aufrufen `gethostent`, `gethostbyname`, `getservbyaddr` verwendet, um Einträge der Datei HOSTS abzufragen. Sie enthält einzelne Felder der Datei HOSTS. Die Felder haben folgende Bedeutung:

Feld	Typ	Bedeutung
h_name	char *	Offizieller Name des Teilnehmers.
h_aliases	char **	Feld mit Alternativ- (Alias-) Namen für den Teilnehmer (Abschluß mit NULL).
h_addrtype	short	Adreßtyp des Teilnehmers; immer AF_INET.
h_length	short	Adreßlänge in Bytes.
h_addr	char *	Internet (IP)-Adresse des Teilnehmers; (wird in Netzwerk-Bytefolge angegeben).

Siehe auch

gethostent, gethostbyname, gethostbyaddr

SERVENT

Syntax

```
#include <socket.h>
typedef struct servent
 {
   char        *s_name;
   char        **s_aliases;
   int         s_port;
   char        *s_proto;
 } SERVENT;
```

Beschreibung

Die Struktur SERVENT wird in den Aufrufen `getservent`, `getservbyname`, `getservbyport` verwendet, um Einträge der Datei SERVICES abzufragen. Sie enthält einzelne Felder der Datei SERVICES. Die Felder haben folgende Bedeutung:

Feld	Typ	Bedeutung
s_name	char *	Offizieller Name des Dienstes (Service-Name).
s_aliases	char **	Feld mit Alternativ- (Alias-) Namen für den Service (Abschluß mit NULL).
s_port	int	Nummer des Ports, über den der Service zu erreichen ist.
s_proto	char *	Protokoll, über das der Service angesprochen werden muß.

Die Portnummer s_port wird in Host Byte Order dargestellt; sie muß gegebenenfalls mit `htons` in Network Byte Order umgewandelt werden.

Siehe auch

getservent, getservbyname, getservbyport

SOCKADDR

Syntax

```
#include <socket.h>
typedef struct sockaddr
 {
   short        sa_family;
   short        sin_port;
   char         sin_addr[4];
 } SOCKADDR;
```

Beschreibung

Die Struktur SOCKADDR wird in den Aufrufen der Socket-Schnittstelle verwendet, um Adressen der Kommunikationsteilnehmer anzugeben oder abzufragen. Die Felder haben folgende Bedeutung:

Feld	Typ	Bedeutung
sa_family	short	Adressfamilie
sin_port	short	Internet-Portnummer
sin_addr	char [4]	Internet(IP)-Adresse

Siehe auch

accept, bind, connect, getpeername, getsockname, recvfrom, sendto

SOCKSEL

Syntax

```
#include <socket.h>
typedef struct socksel
{
 unsigned short  se_inflags;
 unsigned short  se_outflags;
 int             se_fd;
 int             se_1reserved;
 unsigned long   se_user;
 unsigned long   se_2reserved;
} SOCKSEL;
```

Beschreibung

Die Struktur SOCKSEL wird im Aufruf `nselect` verwendet, um Ereignisse an einem bestimmten Socket abzufragen. Die Felder haben folgende Bedeutung:

Feld	Typ	Bedeutung
se_inflags	unsigned short	Input/Request Flags
se_outflags	unsigned short	Output/Reply Flags
se_fd	int	Socket-Deskriptor
se_1reserved	int	Reserviert
se_user	unsigned long	Frei für den Anwender
se_2reserved	unsigned long	Reserviert

Siehe auch

nselect

3.6 Parameterdatensätze der Schnittstellenmodule IF 961-AIO/DIO

Parametrier-möglichkeiten

Sie haben zwei Möglichkeiten, die Schnittstellenmodule zu parametrieren:

1. Über STEP 7
2. über den Aufruf der Funktion *M7StoreRecord* im Anwendungsprogramm

Analog-Ein-/ Ausgabemodul IF 961-AIO

Die nachfolgende Tabelle enthält die Parameter, die Sie für das Schnittstellenmodul IF 961-AIO einstellen können. Das Schnittstellenmodul hat:

- 4 Eingangskanäle und
- 2 Ausgangskanäle

Tabelle 3-5 Parameter für Schnittstellenmodul IF 961-AIO

Parameter	Datentyp	Wertebereich	Codierung	Default–wert	Byte ADR	Bit ADR
Datensatz DS0, 2 Byte lang						
Konvertierungszeit (Zykluszeit)	FIELD3	{5,7 ms | 2,8 ms | 1,3 ms | 0,6 ms | 0,185 ms }	{0|1|2|3|4}	0	0	0
Alarmgenerierung	FIELD1	{nein | ja }	{0|1}	0	0	6
Analogwandlung (Art der Abtastung der Analogkanäle)	FIELD1	{ Einzel | Zyklisch }	{0|1}	0	0	7
...	BIT[3]	...	...	0	0	3
Alarmauswahl	FIELD2	{keine | Prozeß | Prozeß + Diagnose }	{0|1|2}	0	1	0
...	BIT[6]	...	...	0	1	2

Prozeß– und Diagnosealarme

Wenn das Schnittstellenmodul IF 961-AIO für zyklische Wandlung parametriert wurde (Analogwandlung = 1), besteht die Möglichkeit, Prozeßalarme bei Zyklusende auszulösen. Weiterhin ist es möglich einen Diagnosealarm bei Verlust eines Prozeßalarms auszulösen.

Digital-Ein-/ Ausgabemodul IF 961-DIO

Die nachfolgende Tabelle enthält die Parameter, die Sie für das Schnittstellenmodul IF 961-DIO einstellen können.

Die Tabelle 3-6 zeigt Ihnen den Aufbau des Datensatzes 1 der Parameter des Schnittstellenmoduls IF 961-DIO.

Tabelle 3-6 Parameter für Schnittstellenmodul IF 961-DIO

Parameter	Datentyp	Wertebereich	Codierung	Default–wert	Byte ADR	Bit ADR
Datensatz DS0, 2 Byte lang						
Eingangsverzögerung	FIELD1	{ 3 ms | 0,5 ms }	{0|1}	0	0	0
Datensatz DS1, 4 Byte lang						
Interruptfreigabe (Prozeßalarmfreigabe)	FIELD1	{ NEIN | JA }	{ 0|1 }	0	0	7
Alarmfreigabe bei positiver Flanke	FIELD1	{ NEIN | JA }	{0|1}	0	1	0+EK
Alarmfreigabe bei negativer Flanke	FIELD1	{ NEIN | JA }	{0|1}	0	2	0+EK

EK = Eingangskanal: [0 .. 7]

Aufbau Datensatz 1

Sie aktivieren einen Parameter, indem Sie das entsprechende Bit auf ”1” setzen. Eine ”1” in Byte 1 und 2 bedeutet, daß der Prozeßalarm freigegeben ist.

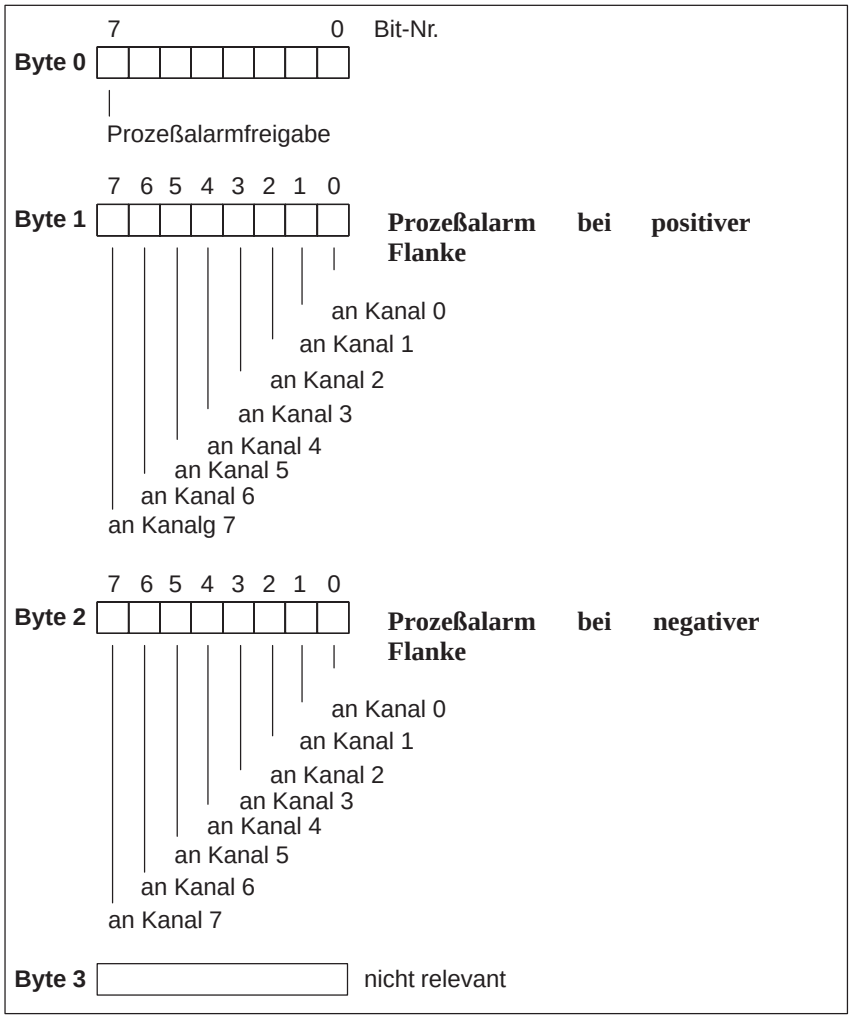


Bild 3-1 Parameterdatensatz 1 der des Schnittstellenmoduls IF 961-DIO

Fehlercodes und -meldungen

4

Kapitelübersicht

4.1 Fehlermeldungen des M7 RMOS32–Kernels

Der M7 RMOS32–Kernel (Nukleus) gibt seine Fehlermeldungen auf der Systemkonsole aus. Die Systemkonsole ist defaultmäßig nicht konfiguriert, kann aber hinzukonfiguriert werden (siehe Benutzerhandbuch).

Fehlende Systemressourcen

Der M7 RMOS32–Kernel benötigt für die Verwaltung der Ressourcen Systemspeicherblöcke, die dynamisch aus dem Heap angefordert bzw. wieder freigegeben werden.

Bei unzureichenden Systemressourcen können folgende Fehlermeldungen ausgegeben werden:

***** nuc: <datum> <uhrzeit> no SRBS, SYSTEM HALTED**
Dem Betriebssystem stehen keine Systemanforderungsblöcke (SRB) mehr zur Verfügung.

***** nuc: <datum> <uhrzeit> no SMRS, SYSTEM HALTED**
Dem Betriebssystem stehen keine Systemspeicherblöcke (SMR) mehr zur Verfügung (z.B. Treiber fordert SMR an).

***** nuc: <datum> <uhrzeit> SMRS increased**
Anzahl der Systemspeicherblöcke (SMR) wurde vom Kernel um 50 erhöht.

***** nuc: <datum> <uhrzeit> SMRS reached 0**
Anzahl der Systemspeicherblöcke (SMR) konnte nicht mehr erhöht werden; der RMOS–API–Aufruf wurde verzögert. Dieser Zustand tritt nur auf, wenn kein Speicher im Heap verfügbar ist oder das Datensegment des Kernels durch eine ungünstige Fragmentierung des Heaps nicht mehr vergrößert werden kann.

Es werden nur die Tasks gesperrt, die indirekt SMRS anfordern z.B. durch RMOS–API–Aufrufe. Andere Tasks – auch mit kleinerer Priorität – laufen weiter. Gesperrte Tasks werden fortgesetzt, sobald wieder SMRS verfügbar sind.

Exception–Interrupthandler

Der Exception-Interrupthandler protokolliert die Prozessor-Exceptions des 80x86-Prozessors und die unerwarteten Interrupts.

Die Protokollausgabe der Prozessorexception-Interrupts gibt in der ersten Zeile Zeitpunkt und Art des Interrupts an. Die zweite Zeile gibt bei Exceptions–Interrupts 8, 10, 11, 12, 13, 14 und 17 den vom Prozessor auf den Stack gelegten Errorcode aus. Die vierte Zeile macht genauere Angaben über den ”Auslöser”. Anschließend werden die aktuellen Registerwerte angezeigt. In der letzten Zeile wird das decodierte Flagregister angezeigt.

Wurde beispielsweise ein Exception-Interrupt von einer Task im A–Zustand ausgelöst, hat die Ausgabe folgendes Aussehen.

```
*** nuc: 02–JAN–1980 10:39:44, GENERAL PROTECTION AT ADDRESS:
0270:0000027A
0270:0000027A   64C60000          MOV    BYTE PTR FS:[EAX],00
   error code: 0
   caused by task id: 0x21: 'exep prot'
   eax: FFFFFFFF, ebx: 00000000, ecx: 00000280, edx: 00000068
   esi: AA55AA55, edi: 000002B8, ebp: FFFFFF78, esp: FFFFFF64
   ss: 0278, ds: 0280, es: 0280, fs: 0000, gs: 0228
   cr0: 7FFFFFE3, cr2: 00000000, cr3: 0000C000
   eflag: 00010282 ( SIGN INTERRUPT IOPL(0) RESUME )
```

Wurde der Exception-Interrupt von einer Interrupthandlerroutine im I-Zustand ausgelöst, ändert sich die vierte Zeile und hat dann folgendes Aussehen:

```
   caused by interrupt handler in i state, SYSTEM HALTED
```

Wurde der Exception-Interrupt von einer Interrupthandlerroutine im S-Zustand ausgelöst, hat die vierte Zeile folgendes Aussehen:

```
   caused by interrupt handler in s state, SYSTEM HALTED
```

In den beiden letzten Fällen hält der Exception–Interrupthandler das System an.

<Exception–Text> ist abhängig vom Exception-Interrupt und steht dabei für folgende Zeichenfolgen:

INT-NUM	ZEICHENFOLGE
INT 0:	DIVIDE ERROR AT ADDRESS:
INT 1:	DEBUG EXCEPTION NEAR ADDRESS:
INT 3:	BREAKPOINT EXCEPTION NEAR ADDRESS:
INT 4:	OVERFLOW EXCEPTION NEAR ADDRESS:
INT 5:	BOUNDS CHECK NEAR ADDRESS:
INT 6:	INVALID OPCODE AT ADDRESS:
INT 7:	NO COPROCESSOR AVAILABLE AT ADDRESS:
INT 8:	DOUBLE FAULT EXCEPTION AT ADDRESS:
INT 9:	NPX SEGMENT OVERRUN NEAR ADDRESS:
INT 10:	INVALID TSS AT ADDRESS:
INT 11:	SEGMENT NOT PRESENT AT ADDRESS:
INT 12:	STACK FAULT AT ADDRESS:
INT 13:	GENERAL PROTECTION AT ADDRESS:
INT 14:	PAGE FAULT AT ADDRESS:
INT 16:	FLOATING-POINT ERROR NEAR ADDRESS:
INT 17:	ALIGNMENT CHECK NEAR ADDRESS:

Abhängig davon, ob das EIP-Register nach dem Exception-Interrupt die Adresse des auslösenden Befehls oder des nächsten Befehls enthält, wird entweder `AT ADDRESS` oder `NEAR ADDRESS` ausgegeben.

NMI–Interrupt

Bei NMI-Interrupt (INT 2) wird folgende Zeichenfolge ausgegeben:

***** nuc: <datum> <uhrzeit> NMI INTERRUPT**

Unerwartete Interrupts

Bei unerwarteten Interrupts wird folgende Meldung ausgegeben:

***** nuc: <datum> <uhrzeit> UNEXPECTED INTERRUPT**

4.2 M7 RMOS32–Exceptionhandler

Ein Exceptionhandler protokolliert alle RMOS–API–Aufrufe, die mit einem Fehler beendet werden, an der Systemkonsole. In der Default–Einstellung ist er nicht aktiviert (siehe Benutzerhandbuch, Systemsoftware für M7–300/400, Installieren und Bedienen):

***** nuc: <datum> <uhrzeit>, svc <name> <zustandstext>**
failed: <fehlernummer> (<fehlertext>)

Dabei bedeuten:

`<name>`	Name des decodierten RMOS–API–Aufrufs, z.B. `RmGetFlag`
`<zustandstext>`	Je nach Systemzustand wird bei Aufruf des RMOS–Exceptionhandlers einer der folgenden Texte eingefügt. 1. from task: <name> id: 0xXX 2. during system startup 3. in monitor mode 4. in s–state 5. in i–state
`<fehlernummer>`	Fehlernummer
`<fehlertext>`	DecodierterFehlertext

***** nuc: 14–FEB–1995 16:20:57, svc RmGetEntry from task: RUN id: 0x29**
failed: 36 (Invalid ID)

4.3 Fehlercodes der RMOS–API–Aufrufe

Return–Werte

Unter bestimmten Umständen kann ein Aufruf des RMOS–APIs fehlschlagen. Deshalb werden von allen Funktionen des RMOS–APIs ***Fehlercodes*** im Return–Wert zurückgeliefert, aus dem Sie dann Erfolg oder Mißerfolg einer Funktionsausführung ableiten können. Der Datentyp des Return–Werts ist ***int***.

Die fehlerfreie Ausführung eines RMOS–API–Aufrufs wird durch den Return–Wert ***RM_OK*** (=0) signalisiert.

RM_OK:
Es ist kein Fehler aufgetreten.

Bei gewissen RMOS–API–Aufrufen können Return–Werte zurückgeliefert werden, die keinen Fehler signalisieren sondern als Mitteilung an den Aufrufer dienen. Diese Mitteilungen haben immer einen ***negativen*** Integerwert (< 0).

Fehlgeschlagene RMOS–API–Aufrufe enthalten Fehlercodes, deren Integerwert echt ***positiv*** (> 0) ist.

Übersicht: Mitteilungen

Folgende Return–Werte sind keine Fehlernummern, sondern Mitteilungen. Sie haben negative Werte.

RM_ENTRY_REMOVED: (–263)
Der Eintrag konnte aus dem Katalog entfernt werden.

RM_ERROR_OUT_OF_RANGE: (–265)
Ungültige Fehlernummer

RM_FLAG_ALREADY_SET: (–258)
Ein Flag war bereits gesetzt.

RM_FLAG_RESET: (–260)
Ein Flag wurde zurückgesetzt.

RM_FLAG_SET: (–259)
Ein Flag wurde gesetzt.

RM_PRI_NOT_CHANGED: (–261)
Die Priorität wurde nicht geändert.

RM_TASK_RESUMED: (–256)
Die Task wurde fortgesetzt.

RM_TASK_WAITING: (–262)
Die Task mußte auf die Ausführung warten (bei WAIT Modus).

Die folgende Auflistung enthält die möglichen von RMOS–API–Aufrufen zurückgelieferten Fehlercodes.

RM_ALL_DEBUGREGISTERS_USED: (45)
Es werden bereits alle Debugregister benutzt.

RM_BOUND_REACHED: (27)
Die mittels `RmSetMailboxSize` eingetragene Grenze wurde überschritten.

RM_BREAKPOINT_ALREADY_SET: (29)
Für die angegebenene Adresse wurde bereits ein Breakpoint gesetzt.

RM_BREAKPOINT_ID_ALREADY_USED: (28)
Die angegebene Breakpoint ID wurde bereits verwendet.

RM_CATALOG_EXCEEDED: (100)
Die konfigurierte Anzahl der möglichen Katalogeinträge wurde überschritten.

RM_GOT_TIMEOUT: (4)
Ein RMOS–API–Aufruf wurde nach der eingestellten Timeout–Zeit abgebrochen.

RM_HEAP_NOT_REDEFINEABLE: (14)
Der Heap ist bereits definiert.

RM_INVALID_DESCRIPTOR: (5)
Es wurde ein ungültiger Deskriptor verwendet.

RM_INVALID_FUNCTION: (44)
Es wurde eine ungültige bzw. nicht unterstützte Funktionsnummer übergeben.

RM_INVALID_ID: (36)
Es wurde eine ungültige ID übergeben.

RM_INVALID_INTERRUPT_NUMBER: (56)
Die Interrupt Nummer war nicht im gültigen Bereich (0–255)

RM_INVALID_IRQ_NUMBER: (41)
Es wurde eine IRQ Nummer für einen nicht definierten PIC verwendet.

RM_INVALID_MEMORYBLOCK: (17)
Es wurde versucht, einen ungültigen Speicherbereich freizugeben.

RM_INVALID_NULLPOINTER: (10)
An dieser Stelle ist kein Null Pointer erlaubt.

RM_INVALID_OFFSET: (39)
Der Offset war außerhalb des gültigen Bereichs.

RM_INVALID_POINTER: (42)
Ein Zeiger war ungültig.

RM_INVALID_SEGMENTLENGTH: (6)
Es wurde eine ungültige Segmentlänge angegeben.

RM_INVALID_SELECTOR: (21)
Es wurde ein ungültiger Selektor verwendet.

RM_INVALID_SIZE: (38)
Eine Größenangabe war ungültig.

RM_INVALID_STRING: (37)
Ein String ist nicht innerhalb der vorgegebenen Größe

RM_INVALID_TASK_ENTRY: (60)
Task Entry ungültig

RM_INVALID_TASK_STATE: (22)
Es wurde ein nicht erlaubter `RmKillTask` Aufruf abgesetzt.

RM_INVALID_TYPE: (35)
Es wurde ein ungültiger Parameter (*mode*, *type*, *pri_type*, etc.) übergeben.

RM_IS_ALREADY_CATALOGED: (47)
Der zu katalogisierende String ist bereits eingetragen.

RM_IS_NOT_CATALOGED: (48)
Der String ist nicht katalogisiert.

RM_MEMORY_ALREADY_USED: (25)
Der zu reservierende Speicherblock ist bereits belegt.

RM_NO_MESSAGE: (43)
Die Mailbox (Message Queue) enthält keine Nachricht (Botschaft).

RM_NOT_HALTABLE: (46)
Die Task konnte nicht angehalten werden.

RM_OUT_OF_FLAGGROUPS: (12)
Die konfigurierte Anzahl Event Flags ist überschritten.

RM_OUT_OF_MAILBOXES: (15)
Die konfigurierte Anzahl Mailboxes ist überschritten.

RM_OUT_OF_MEMORY: (3)
Es steht kein Speicherbereich ausreichender Größe zur Verfügung.

RM_OUT_OF_MEMORYPOOLS: (13)
Die konfigurierte Anzahl Speicherpools ist überschritten.

RM_OUT_OF_SEMAPHORES: (16)
Die konfigurierte Anzahl Semaphores ist überschritten.

RM_PARAMETER_ERROR: (2)
Ein wurde mit falschen Parametern versorgt.

RM_QUEUE_EXIST: (59)
Die Message Queue ist bereits vorhanden.

RM_QUEUE_NOT_EXIST: (58)
Keine Message Queue vorhanden.

RM_RESOURCE_BUSY: (18)
Das zu löschende Betriebsmittel ist belegt.

RM_RESOURCE_NOT_AVAILABLE: (23)
Das gewünschte Betriebsmittel ist nicht verfügbar.

RM_SVC_NOT_CONFIGURED: (33)
Es wurde versucht, einen nicht konfigurierten RMOS–API–Aufruf auszuführen. Um welchen RMOS–API–Aufruf es sich hierbei handelt können Sie den Ausgaben des RMOS–Exceptionhandlers entnehmen.

RM_TASK_DORMANT: (7)
Die Task ist im Zustand RUHEND.

RM_TASK_KILLED: (49)
Die Task wurde mit dem RMOS–API–Aufruf `RmKillTask` gelöscht.

RM_TASK_NOT_DORMANT: (20)
Es wurde versucht, eine nicht im Zustand RUHEND befindliche Task zu löschen oder zu starten.

RM_TASK_NOT_IN_BP_CONTEXT: (31)
Die Task wurde nicht durch einen Breakpoint unterbrochen.

RM_TASK_NOT_IN_RTE_HALT: (32)
Die Task wurde nicht durch einen Laufzeitfehler unterbrochen.

RM_TASK_NOT_PAUSED: (26)
Die mittels `RmResumeTask` fortzusetzende Task wurde nicht durch `RmPauseTask` angehalten.

RM_TEST_NOT_OK: (57)
Ein Test wurde nicht erfolgreich abgeschlossen.

RM_TASK_NOT_READY: (30)
Es wurde versucht, eine Task anzuhalten, die sich nicht im Zustand BEREIT befindet.

4.4 Fehlercodes der M7–API–Aufrufe

Hinweise

Von den Funktionen des M7–APIs werden Fehlercodes entweder im Return–Wert der Funktion oder – im Unterschied zum RMOS–API – über eine Pointer–Variable zurückgegeben. Der Datentyp des zurückgelieferten Fehlercodes ist M7ERR_CODE und ist in der Datei **M7API.H** definiert.

Da die Funktionalität des M7–APIs von einzelnen M7–Server bereitgestellt wird, sind die Fehlercodes gemäß dieser Aufteilung untergliedert.

Allg. Fehler

Die folgende Auflistung zeigt die möglichen von M7–API–Aufrufen zurückgelieferten allgemeinen Fehlercodes. Sämtliche Konstanten sind in der Headerdatei **M7API.H** definiert.

M7SUCCESS: (0)
Funktion wurde erfolgreich ausgeführt, kein Fehler ist aufgetreten.

M7E_NO_MEM: (–1)
Funktion muß zur Ausführung dynamisch Speicher allokieren, kein Speicher vorhanden

M7E_PAR: (–100)
Beim Funktionsaufruf wurde ein falscher Parameter übergeben

M7E_PRIO: (–3)
Die beim Funktionsaufruf übergebene Priorität ist außerhalb des gültigen Bereichs.

M7E_RESSOURCE_LIMIT: (–2)
Keine Ressourcen vorhanden

PSUB–Interface

Die folgende Auflistung zeigt die vom P–BUS–Peripherietreiber zurückgelieferten Fehlercodes.

M7E_ALARM_GEN_DISABLED: (–121)
Alarmgenerierung wurde im Datensatz 0 gesperrt

M7E_ALARM_PENDING: (–128)
Es steht noch ein Alarm an, der zuerst quittiert werden muß

M7E_BSY: (–104)
Lokalbus ist busy

M7E_CMD: (–105)
Lokalbus mit Kommandofehler

M7E_COM_ERROR: (–110)
Baugruppe hat Protokoll abgebrochen

M7E_D_ALARM_BUSY: (–117)
Diagnosealarm wurde noch nicht durch die CPU quittiert

M7E_D_ALARM_GEN_DISABLED: (–119)
Diagnosealarm im Datensatz 0 gesperrt

M7E_DP_SLAVE_STATE: (–123)
Aktion im aktuellen Slave–Status nicht möglich

M7E_DPX2_FAULT: (–124)
DPX2–Anforderung gescheitert

M7E_GL_ALARM_DISABLED (–122)
Alle Alarme sind gesperrt

M7E_HWFAULT: (–101)
Allgemeiner Hardware–Fehler

M7E_INVAL_DEV: (–126)
Parameterfehler

M7E_IO_DESC: (–109)
Falscher I/O–Deskriptor

M7E_NORM_DIAG: (–127)
Keine Diagnosedaten vorhanden

M7E_ODIS: (–120)
CPU hat das Signal ODIS (Output Disabled) ausgelöst

M7E_P_ALARM_BUSY: (–116)
Prozeßalarm wurde noch nicht durch die CPU quittiert

M7E_P_ALARM_GEN_DISABLED: (–118)
Prozeßalarm im Datensatz 0 gesperrt

M7E_PARITY: (– 106)
Lokalbus mit Parityfehler

M7E_PEU: (–102)
Fehler im Ausbau der Peripherie

M7E_QVZ: (–103)
Lokalbus mit Quittungsverzug

M7E_REC_LENGTH: (–111)
Falsche Datensatzlänge

M7E_REC_NUMBER: (–112)
Falsche Datensatznummer

M7E_SLAVE_TYPE: (–125)
Kein S7–Slave

M7E_WRONG_STATE: (–113)
Aktion im aktuellen Betriebszustand nicht erlaubt

S7–Objekt–Server

Die folgende Auflistung zeigt die vom S7–Objekt–Server zurückgelieferten Fehlercodes.

M7E_BIT_OFFSET: (–203)
Der innerhalb eines Bytes angegebene Bit–Offset ist falsch

M7E_BLOCK_ROMDIR: (–211)
Baustein im Verzeichnis ROMDIR nicht lesbar

M7E_LENGTH: (–208)
Die beim Lesen, Schreiben oder Erzeugen angegebene Länge ist 0

M7E_LINK_PAR: (–214)
Übergebene Parameter bei den Aufrufen `M7LinkDataAccess` oder `M7LinkDataAccessCB` sind falsch

M7E_NODIR: (–209)
Das Verzeichnis der S7–Objekte ist nicht vorhanden oder nicht lesbar

M7E_NOT_LOCATED: (–217)
Objekt wurde nicht mit `M7LocateObject` an eine Anwendertask übergeben

M7E_OBJ: (–200)
Objekttyp wird durch den S7–Objekt–Server nicht unterstützt

M7E_OBJ_EXISTS: (–205)
Das S7–Objekt existiert bereits

M7E_OFFSET: (–202)
Der im S7–Objekt angegebene Offset ist falsch

M7E_OVS_WRONG_STATE: (–216)
Aktion im aktuellen Betriebszustand nicht erlaubt

M7E_PART: (–201)
Der zum Objekttyp spezifizierte Teilbereich ist nicht vorhanden

M7E_PART_INVALID: (–206)
Angegebene Teilbereichsnummer ist ungültig

M7E_PER_BITS: (–213)
Bitadressierung im Peripheriebereich verboten

M7E_REM_OBJ: (–215)
Aktion für remanente Objekte nicht erlaubt

M7E_SIZE: (–212)
Längeninformation im Bausteinkopf und Dateilänge sind unterschiedlich

M7E_TYPE: (–207)
Angegebener Datentyp wird nicht unterstützt

M7E_WRITE_PROTECT: (–204)
Das S7–Objekt ist schreibgeschützt

BZÜ–Server

Die folgende Auflistung zeigt die vom BZÜ–Server (Betriebszustandsübergang) zurückgelieferten Fehlercodes.

M7E_OST_CPU_IN_STOP: (–306)
CPU ist im Betriebszustand STOP

M7E_OST_DENIED: (–308)
Angeforderter Betriebszustandsübergang wurde von mindestens einer Task abgebrochen

M7E_OST_ILLEGAL_PARAM_CPU: (–305)
Ungültige CPU–Parametrierung

M7E_OST_MODE_SW_IN_STOP: (–304)
Betriebsartenschalter der Baugruppe ist auf STOP

M7E_OST_NO_SUCH_FRB: (–301)
Angegebener TSFRB ist nicht in Bearbeitung

M7E_OST_NO_SUCH_STATE: (–302)
Unbekannter Betriebszustand

M7E_OST_NO_SUCH_TRANSITION: (–300)
Unbekannter Betriebszustandsübergang

M7E_OST_TIMEOUT: (–307)
Angeforderter Betriebszustandsübergang wurde mit Timeout abgebrochen

M7E_OST_WRONG_STATE: (–303)
Betriebszustandsübergang ist aus dem augenblicklichen Betriebszustand nicht möglich

FZ–Server

Die folgende Auflistung zeigt die vom FZ–Server (Freier Zyklus) zurückgelieferten Fehlercodes.

M7E_FSC_NO_SUCH_CYCLE: (–400)
Unbekannter Zustand

M7E_FSC_NO_SUCH_FRB: (–401)
Angegebener FSCFRB ist nicht in Bearbeitung

Diagnose–Server

Die folgende Auflistung zeigt die vom Diagnose–Server zurückgelieferten Fehlercodes.

M7E_DIAG_NUMBER: (–500)
Falsche Ereignisklasse (nur 0x0a oder 0x0b erlaubt)

M7E_DIAG_STATE: (–501)
Falscher Betriebszustand

K–Bus–Interface

Die folgende Auflistung zeigt die vom K–BUS–Kommunikationsfunktionen zurückgelieferten Fehlercodes.

M7E_KSUB_BLOCK_TOO_LARGE: (–604)
Angegebener Puffer hat keine ausreichende Größe

M7E_KSUB_CONN_ACTIVE: (–609)
Die Verbindung ist zur Zeit aktiv und kann nicht geschlossen werden

M7E_KSUB_CONN_CLOSED: (–602)
Angegebene Verbindung wurde bereits geschlossen

M7E_KSUB_EOF: (–607)
End of File bzw. Verzeichnisende

M7E_KSUB_FILEIO: (–606)
Fehler in der Dateibearbeitung

M7E_KSUB_NO_SRV: (–603)
K–BUS ist nicht verfügbar

M7E_KSUB_NO_SUCH_CONN: (–601)
Angegebene Verbindungs–ID ist ungültig

M7E_KSUB_NO_SUCH_FRB: (–605)
Angegebener COMMFRB ist nicht in Bearbeitung

M7E_KSUB_PARAM: (–600)
Angegebene Parameter sind falsch

M7E_KSUB_REMOTE: (–608)
Ausführungsfehler beim entfernten Server

M7E_KSUB_SDB_WAS_DELETED: (–611)
Verbindung in STEP7 gelöscht,die Verbindung ist nicht mehr aktiv

FRB–Bearbeitung

Die folgende Auflistung zeigt die bei der allgemeinen Bearbeitung von FRBs auftretenden Fehlercodes. Der Fehlercode kann aus dem Header des FRBs über das Makro `M7GetFRBErr` ausgelesen werden.

M7E_FRB_NOT_BUSY: (–700)
Angegebener FRB ist nicht in Bearbeitung

M7E_FRB_NOT_IN_LIST: (–701)
Angegebener FRB befindet sich nicht in der verketteten, internen FRB–Liste

M7E_FRB_ALREADY IN LIST: (–702)
FRB ist bereits eingekettet

Interne Fehler

Die folgende Auflistung zeigt die bei der internen Bearbeitung auftretenden Fehlercodes.

M7E_INTERNAL_ERROR: (–9901)
Interner Fehler ist aufgetreten

M7E_NOT_IMPLEMENTED: (–9900)
Server existiert nicht

4.5 Fehlercodes ladbarer Treiber

Dieser Abschnitt beschreibt die Fehlercodes, die von den Aufrufen für ladbare Treiber zurückgegeben werden können. Neben dem Define ist der entsprechende Zahlenwert und eine kurze Erklärung der Fehlermeldung angegeben.

Fehlercodes

Folgende Fehlercodes können bei allen ladbaren Treiber (SER8250.DRV, 3964.DRV) auftreten.

RM_EIO_PARAMETER 0×0401
Parameterfehler

RM_EIO_INVALID_CONTROL 0×0402
Die angegebene Steuerfunktion wird nicht unterstützt

RM_EIO_INVALID_ACCESS 0×0403
Deskriptor nicht für benutzten Zugriff (Read/Write) geöffnet

RM_EIO_UNIT_RESERVED 0×0404
Unit ist bereits reserviert bzw. Unit war nicht von der aufrufenden Task reserviert.

RM_EIO_CANCEL 0×0405
Auftrag wurde durch RM_IOCTL_CANCEL abgebrochen

RM_EIO_LOCKED 0×0406
Die Unit ist durch RM_IOCTL_LOCK ”verriegelt”

RM_EIO_IO_ERROR 0×0407
Auftrag ist aufgrund I/O–Fehler abgebrochen

RM_EIO_PARITY_ERROR 0×0408
Auftrag wegen Parity–Fehler abgebrochen

RM_EIO_OVERRUN_ERROR 0×0409
Auftrag wegen Überlauffehler abgebrochen

RM_EIO_TIMEOUT 0×040A
Auftrag mit Timeout abgebrochen

RM_EIO_INVALID_STATE 0×040B
Bei der Statusüberprüfung des Controllers (z.B. Parity) ist ein Fehler aufgetreten

RM_EIO_NO_HARDWARE 0×040C
Hardware nicht vorhanden oder defekt

RM_EIO_INIT_FAILED 0×040D
Die Initialisierung der Unit war nicht möglich

RM_EIO_UNIT_RESET 0×040E
Auftrag wurde durch RM_IOCTL_RESET abgebrochen

Hinweise

Folgende Hinweise können ebenfalls als Rückgabewert auftreten

RM_IO_QUEUED × − 1024
Auftrag in die Warteschlange eingehängt

RM_IO_IN_PROGRESS − 1025
Auftrag befindet sich in Bearbeitung

RM_IO_NO_DATA × − 1026
Keine Daten vorhanden

Fehlercodes 3964(R)–Treiber

Beim 3964(R)–Treiber (3964.DRV) können zusätzlich folgende Fehler auftreten:

RM_EIO_3964_NO_TIMER 0x480
Es konnte kein Timer gestartet werden

RM_EIO_3964_BUFFER_OVERFLOW 0x481
Es wurden mehr Daten empfangen als beim Leseauftrag angegeben

RM_EIO_3964_UNEXPECTED_CHARACTER 0x482
Unerwartetes Zeichen empfangen

RM_EIO_3964_CHECKSUM_ERROR 0x483
Fehler in der Prüfsumme (bei 3964R Protokoll)

RM_EIO_3964_REQUEST_SUSPENDED 0x484
Der Auftrag wurde wegen eines Initiierungskonflikts (Master und Slave senden gleichzeitig) beendet

RM_EIO_3964_CONNECTION_REFUSED 0x485
reserviert

RM_EIO_3964_TRANSFER_ABORT 0x486
Die Gegenseite hat die Übertragung (Senden oder Empfang) mit NAK abgebrochen.

RM_EIO_3964_READ_CANCELED 0x487
Leseauftrag mit RM_IOCTL_CANCEL abgebrochen

RM_EIO_3964_WRITE_CANCELED 0x488
Schreibauftrag mit RM_IOCTL_CANCEL abgebrochen

4.6 Fehlercodes C–Laufzeitbibliothek

Aufbau der Fehlermeldung

Fehlermeldungen der C–Laufzeitbibliothek (CRUN) werden folgendermaßen ausgegeben:

***** crun: <datum> <uhrzeit>, <fehlermeldung>**
caused by task id: <taskid>: '<taskname>'

`<datum>`	Datum, an dem der Fehler auftrat
`<uhrzeit>`	Uhrzeit, zu der der Fehler auftrat
`<fehlermeldung>`	Eigentliche Fehlermeldung
`<taskid>`	ID der Task, die den Fehler verursacht hat
`<taskname>`	String, mit dem die Task, die den Fehler verursacht hat, im Betriebsmittel–Katalog eingetragen ist

Beispiel:

***** crun: 20–OCT–94 17:32:20, sin not configured – task aborted**
caused by task id: 0x23: 'FLTTEST'

Die Fehlermeldungen werden ebenfalls auf der Systemkonsole ausgegeben.

Fehlermeldungen

Fehlermeldungen der C–Laufzeitbibliothek (CRUN)

<function>: cannot allocate memory
In der CRUN Funktion `<function>` konnte für interne Operationen kein Speicher mehr angefordert werden.

<function> not configured – task aborted
Die Funktion `<function>` wurde von einer nachladbaren Task aufgerufen, ist aber für die Schnittstelle für nachladbare Tasks nicht konfiguriert. Die aufrufende Task wurde durch `exit` beendet.

<function>: unknown hsfs return value xxxx
In der CRUN Funktion `<function>` wurde ein HSFS Aufruf mit dem (unerwarteten) Fehlercode `xxxx` beendet.

automatic xinitc failed – task aborted
Die automatische Initialisierung der CRUN (siehe auch `xinitc`) schlug fehl. Die Task, die die automatische CRUN–Initialisierung verursacht hat, wurde durch `exit` abgebrochen.

automatic xinitt failed – task aborted
Die automatische Initialisierung einer Task innerhalb CRUN (siehe auch `xinitt`) schlug fehl. Die Task, die die automatische Initialisierung verursacht hat, wurde durch `exit` abgebrochen.

catalog entry "ERRLOG" not found
Der Eintrag ERRLOG wurde im Betriebsmittel–Katalog nicht gefunden. Daher kann die CRUN nicht die Errorlogger–Task für Fehlerausgaben benutzen, sondern gibt Fehlermeldungen über den BYT–Treiber an der Systemkonsole aus.

fclose: cannot delete temporary file
Eine mit `tmpfile` erstellte temporäre Datei konnte beim Schließen durch `fclose` nicht gelöscht werden.

illegal function code *xxxx* – task aborted
Der Schnittstelle für nachladbare Tasks wurde der ungültige Funktionscode `xxxx` übergeben. Die aufrufende Task wurde durch `exit` beendet.

reserved function code *xxxx* – task aborted
Der Schnittstelle für nachladbare Tasks wurde der reservierte Funktionscode `xxxx` übergeben. Die aufrufende Task wurde durch `exit` beendet.

4.7 Fehlercodes der Socket–Schnittstelle

Dieser Abschnitt beschreibt die Fehlercodes, die von den Aufrufen der Sokket–Schnittstelle zurückgegeben werden können. Neben dem Define ist der entsprechende Zahlenwert und eine kurze Erklärung der Fehlermeldung angegeben. Zusätzlich können Standard–Fehlercodes der C–Laufzeitbibliothek in *errno* gesetzt werden (siehe Beschreibung von ***errno***).

EWOULDBLOCK 61
Der Socket ist im nicht–blockierenden Modus und die Funktion kann nicht ausgeführt werden

EINPROGRESS 62
Der Aufruf wird gerade bearbeitet.

EALREADY 63
Der Aufruf wird bereits bearbeitet.

EDESTADDRREQ 64
Es wird ein Zieladresse benötigt

EMSGSIZE 65
Die Nachricht ist zu lang

EPROTOTYPE 66
Falsches Protokoll für diesen Socket

ENOPROTOOPT 67
Die angegebene Protokolloption ist ungültig.

EPROTONOSUPPORT 68
Das angegebene Protokoll ist ungültig.

ESOCKNOSUPPORT 69
Der angegebene Socket–Typ ist ungültig.

EOPNOTSUPP 70
Die Funktion wird an diesem Socket–Typ nicht unterstützt.

EPFNOSUPPORT 71
Die Protokollfamilie wird nicht unterstützt

EAFNOSUPPORT 72
Die angegebene Adressfamilie ist ungültig.

EADDRINUSE 73
Die Portnummer oder Adresse wird bereits verwendet.

EADDRNOTAVAIL 74
Falsche IP–Adresse

ENETDOWN 75
Der Treiber ist nicht korrekt initialisiert.

ENETUNREACH 76
Das Netzwerk ist nicht erreichbar

ENETRESET 77
Das Netwerk wurde zurückgesetzt und die Verbindung wurde abgebaut

ECONNABORTED 78
Die Verbindung ist abgebaut.

ECONNRESET 79
Der Partner hat die Verbindung abgebaut.

ENOBUFS 80
Es ist kein Speicherplatz für einen weiteren Socket oder für weitere Verbindungsanforderungen vorhanden.

EISCONN 81
Es existiert bereits eine Verbindung über diesen Socket.

ENOTCONN 82
An dem Socket ist noch keine Verbindung aufgebaut

ESHUTDOWN 83
shutdown wurde aufgerufen. Es können keine weiteren Daten gesendet werden.

ETOOMANYREFS 84
Zu viele Referenzen

ETIMEDOUT 85
Die Verbindung zum Partner konnte nicht aufgebaut werden.

ECONNREFUSED 86
Der Verbindungsaufbau wurde vom Partner zurückgewiesen.

EBUFTOOSMALL 87
Der Puffer ist für diese Operation zu klein

ESMODEXISTS 88
Socket–Modul ist bereits vorhanden

ENOTSOCK 89
Der Socket–Deskriptor *sd* referenziert keinen Socket.

EDEADLOCK 90
Völliger Stillstand

EHOSTDOWN 91
Kommunikationspartner ist nicht aktiv

EHOSTUNREACH 92
Kommunikationspartner ist nicht erreichbar

ENOURGENTDATA 93
Es sind keine hochprioren Daten vorhanden.

EMAYBEISO 95
Ungültiges Protokoll beim Partner.

M7–API

5

Kapitelübersicht

M7_SWAP_DWORD

Syntax

#include <m7api.h>
UDWORD M7_SWAP_DWORD(UDWORD *x*);

Parameter

Parametername	Bedeutung
x	Doppelwort (M7–Datentyp DWORD, 32 Bit) in Intel bzw. SIMATIC–Darstellung

Beschreibung

Die Funktion wandelt ein Doppelwort (M7–Datentyp: DWORD) aus der Intel–Darstellung in ein Doppelwort in SIMATIC–Darstellung (Motorola–Format) um und umgekehrt.

Der Aufruf ist als Makro realisiert. Es wird keine Typprüfung des Eingangsparameters vorgenommen.

Rückgabewert

Doppelwort in Intel–Darstellung, wenn Eingangsparameter in SIMATIC–Darstellung

Doppelwort in SIMATIC–Darstellung, wenn Eingangsparameter in Intel–Darstellung

Siehe auch

M7_SWAP_WORD

M7_SWAP_WORD

Funktion

Wort von Intel– in SIMATIC–Darstellung umwandeln und umgekehrt

Syntax

#include <m7api.h>
UWORD M7_SWAP_WORD(UWORD *x*);

Parameter

Parametername	Bedeutung
x	Wort (M7–Datentyp WORD, 16 Bit) in Intel– bzw. SIMATIC–Darstellung

Beschreibung

Die Funktion wandelt ein Wort (M7–Datentyp WORD) aus der Intel– in ein Wort in SIMATIC–Darstellung (Motorola–Format) um und umgekehrt.

Der Aufruf ist als Makro realisiert. Es wird keine Typprüfung des Eingangsparameters vorgenommen.

Rückgabewert

Wort in Intel–Darstellung, wenn Eingangsparameter in SIMATIC–Darstellung

Wort in SIMATIC–Darstellung, wenn Eingangsparameter in Intel–Darstellung

Siehe auch

M7_SWAP_DWORD

M7BUBCycRead

Funktion **Auftrag für Zyklisches Lesen einrichten**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7BUBCycRead(
                UDWORD flags,
                M7CONNID ConnID,
                M7COMMFRB_PTR pCommFRB,
                UBYTE nVars,
                M7VARADDR_PTR pAddrBuffer,
                M7VARDATA_PTR pDataBuffer,
                UDWORD CycTime,
                UDWORD *pnRequest
                unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
flags	Flags A_IMMEDIATE: Falls A_IMMEDIATE gesetzt ist, wird der Auftrag sofort gestartet, ansonsten muß der angemeldete Auftrag explizit über `M7BUBCycReadStart` gestartet werden. A_ZERO_FLAG: Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
pCommFRB	Zeiger auf Function Request Block für asynchrone Benachrichtigung
nVars	Anzahl der zu lesenden Variablen, d.h. Einträge im Adreßpuffer.
pAddrBuffer	Zeiger auf ein Array mit nVars Elementen. Jedes Element ist vom Typ **M7VARADDR** und spezifiziert einen fortlaufenden Bereich von Items innerhalb eines S7–Objekts(vgl. Kapitel 3).
pDataBuffer	Zeiger auf ein Array mit nVars Elementen. Jedes Element ist vom Typ **M7VARDATA** und spezifiziert einen Puffer (Adresse, Größe, usw.) zur Aufnahme einer Variablen (vgl. Kapitel 3) . Die einzelnen Puffer müssen vor Absetzen des obigen Aufrufs in den globalen Daten oder auf dem Heap eingerichtet worden sein.
CycTime	Zykluszeit in ms. Es sind folgende Zykluszeiten möglich: 0.1s, 0.2s, 0.3s, 0.4s, 0.5s, 0.6s, 0.7s, 0.8s, 0.9s, 1s, 2s, 3s, 4s, 5s, 6s, 7s, 8s, 9s, 10s, 20s, 30s, 40s, 50s, 60s, 70s, 80s, 90s.

Parametername	Bedeutung
pnRequest	Zeiger auf die zurückgegebene Auftragsnummer
MPrio	Priorität, mit der die Message verschickt wird (0–255).

Beschreibung

Die Funktion `M7BUBCycRead` richtet einen BuB–Auftrag zum zyklischen Lesen ein. Die Variablen–Spezifikation wird im Adreßpuffer mitgegeben und entspricht derjenigen in `M7BUBRead`. Die Daten fallen asynchron zur Anwendung an.

Beim Aufruf `M7BUBCycRead` gelten folgende Bedingungen für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize - 28$$

and

$$0 \quad maxpdusize - 26 - 12 * nVars$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit M7KInitiate eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Die Anwendung wird mit Hilfe der Nachricht M7MSG_BUB_NDR über neue Daten informiert und kann diese mit `M7KEvent` abholen.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt (siehe Hinweis).
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Hinweis

Der Rückgabewert M7SUCCESS stellt nicht sicher, daß der gesamte Lesevorgang erfolgreich durchgeführt wurde. Zusatzinformationen über den Erfolg der einzelnen Datentransfers sind im Element AccessResult der Struktur M7VARDATA enthalten.

Fehlercodes

Fehlercodes	Bedeutung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_BLOCK_TOO_LARGE	Puffer nicht ausreichend
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden

Fehlercodes	Bedeutung
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch **M7BUBCycReadDelete, M7BUBCycReadStart, M7BUBCycReadStop**

M7BUBCycReadDelete

Funktion

Auftrag für Zyklisches Lesen löschen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7BUBCycReadDelete(
                  M7CONNID ConnID,
                  UDWORD nRequest);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nRequest	Auftragsnummer von `M7BUBCycRead`

Beschreibung

Die Funktion `M7BUBCycReadDelete` löscht einen BuB–Auftrag für zyklisches Lesen, der mit `M7BUBCycRead` eingerichtet wurde.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7BUBCycRead, M7BUBCycReadStart, M7BUBCycReadStop

M7BUBCycReadStart

Funktion

Auftrag für Zyklisches Lesen starten

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7BUBCycReadStart(
                  M7CONNID ConnID,
                  UDWORD nRequest);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`– Aufruf.
nRequest	Auftragsnummer von `M7BUBCycRead`

Beschreibung

Die Funktion `M7BUBCycReadStart` startet einen BuB–Auftrag zum zyklischen Lesen, der mit `M7BUBCycRead` eingerichtet wurde.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7BUBCycRead, M7BUBCycReadDelete, M7BUBCycReadStop

M7BUBCycReadStop

Funktion

Auftrag für Zyklisches Lesen stoppen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7BUBCycReadStop(
                  M7CONNID ConnID,
                  UDWORD nRequest);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`– Aufruf.
nRequest	Auftragsnummer von `M7BUBCycRead`

Beschreibung

Die Funktion `M7BUBCycReadStop` stoppt einen BuB–Auftrag zum zyklischen Lesen, der mit `M7BUBCycRead` oder `M7BUBCycReadStart` gestartet wurde.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7BUBCycRead, M7BUBCycReadDelete, M7BUBCycReadStart

M7BUBRead

Funktion

BUB–Variablen lesen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7BUBRead(
                  M7CONNID ConnID,
                  UBYTE nVars,
                  M7VARADDR_PTR pAddrBuffer,
                  M7VARDATA_PTR pDataBuffer,
                  UDWORD*pnBytes);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nVars	Anzahl der zu lesenden Variablen, d.h. Einträge im Adreßpuffer.
pAddrBuffer	Zeiger auf ein Array mit nVars Elementen. Jedes Element ist vom Typ **M7VARADDR** und spezifiziert einen fortlaufenden Bereich von Items innerhalb eines S7–Objekts (vgl. Kapitel 3).
pDataBuffer	Zeiger auf ein Array mit nVars Elementen. Jedes Element ist vom Typ **M7VARDATA** und spezifiziert einen Puffer (Adresse, Größe, usw.) zur Aufnahme einer Variablen (vgl. Kapitel 3). Die einzelnen Puffer müssen vor Absetzen des obigen Aufrufs in den globalen Daten oder auf dem Heap eingerichtet worden sein.
pnBytes	Zeiger auf Variable. In dieser Variablen liefert der Aufruf die Anzahl der tatsächlich gelesenen Bytes zurück

Beschreibung

Die Funktion `M7BUBRead` startet einen synchronen Aufruf zum Lesen der im Adreß–Array *pAddrBuffer* angegebenen Variablen in den, über das Daten–Array *pDataBuffer* spezifizierten Datenpuffer ein.

Beim Aufruf `M7BUBRead` gelten folgende Bedingungen für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize - 14$$

and

$$0 \quad maxpdusize - 12 * (nVars \quad 1)$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit M7KInitiate eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt (siehe Hinweis).
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Hinweis

Der Rückgabewert M7SUCCESS stellt nicht sicher, daß der gesamte Lesevorgang erfolgreich durchgeführt wurde. Zusatzinformationen über das Ergebnis der einzelnen Datentransfers sind im Element `AccessResult` der Struktur M7VARDATA enthalten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_BLOCK_TOO_LARGE	Puffer nicht ausreichend
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7BUBWrite

M7BUBWrite

Funktion

BUB–Variablen schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7BUBWrite(
                  M7CONNID ConnID,
                  UBYTE nVars,
                  M7VARADDR_PTR pAddrBuffer,
                  M7VARDATA_PTR pDataBuffer);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nVars	Anzahl der zu schreibenden Variablen.
pAddrBuffer	Zeiger auf ein Array mit nVars Elementen. Jedes Element ist vom Typ **M7VARADDR** (vgl. Kapitel 3) und spezifiziert den Datentyp, den Bausteintyp, die Bausteinnummer und den Startoffset der zu überschreibenden Variablen im Datenbereich des S7–Objekt–Servers (M7) bzw. im S7–CPU–Datenbereich.
pDataBuffer	Zeiger auf ein Array mit nVars Elementen. Jedes Element ist vom Typ **M7VARDATA**(vgl. Kapitel 3)und spezifiziert einen Puffer (Adresse, Größe, usw.) zur Aufnahme eines Wertes, mit dem die zugehörige Variable im Datenbereich des S7–Objekt–Servers (M7) bzw. im S7–CPU–Datenbereich überschrieben wird.

Beschreibung

Die Funktion `M7BUBWrite` startet einen synchronen Aufruf zum Überschreiben der im Adreß–Array *pAddrBuffer* angegebenen Variablen mit den im Daten–Array *pDataBuffer* indirekt angegebenen Werten.

Die Adreß– und Datenspezifikationen entsprechen denen von `M7BUBRead`.

Für den Aufruf `M7BUBWrite` gilt folgende Bedingung für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize - 12 * (nVars \quad 1)$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit M7KInitiate eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt (siehe Hinweis).
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Hinweis

Der Rückgabewert M7SUCCESS stellt nicht sicher, daß der gesamte Schreibvorgang erfolgreich durchgeführt wurde. Zusatzinformationen über das Ergebnis der einzelnen Datentransfers sind im Element `AccessResult` der Struktur M7VARDATA enthalten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7BUBRead

M7CheckResource

Funktion

Batterie und SRAM prüfen

Syntax

#include <m7api.h>
M7ERR_CODE M7CheckResource (UWORD *pFlags);

Parameter

Parametername	Bedeutung	
pFlags	Zeiger auf Flags.	
	M7SRAM_OK	SRAM is fehlerfrei
	M7BATTERY_OK	Batteriepufferung ist sichergestellt
	M7BATTERY_CHARGE_OK	Alle Pufferbatterien sind fehlerfrei
	Falls eines der Bits nicht gesetzt ist, ist die zugehörige Ressource fehlerhaft.	

Beschreibung

Die Funktion `M7CheckResource` dient zur Überprüfung des SRAM und der Batterie.

Die Batteriepufferung für eine M7 300 CPU bzw. FM wird durch eine Batterie auf der Baugruppe sichergestellt, die Pufferung für eine M7 400 CPU stellen zwei Batterien auf der Stromversorgung des Zentralbaugruppenträgers sicher.

Hinweis

M7VARDATADie Funktion `M7CheckResource` wird für eine FM 456–4 nicht unterstützt. `M7CheckResource` liefert auf einer FM 456–4 immer BATTERY_OK zurück.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

M7ClearPI

Funktion

Prozeßabbild rücksetzen

Syntax

#include <m7api.h>
M7ERR_CODE M7ClearPI(UWORD *PIType***);**

Parameter

Parametername	Bedeutung
PIType	Kennzeichen der Prozeßabbilder: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge

Beschreibung

Die Funktion setzt das gesamte Prozeßabbild, das mit dem Parameter *PIType* angegeben wird, auf ’0’ zurück.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType*

Siehe auch

M7LoadPII, M7StorePIQ

M7ConfirmCycle

Funktion

Nachricht des FZ–Server quittieren

Syntax

#include <m7api.h>
M7ERR_CODE M7ConfirmCycle(
M7FSCFRB_PTR *pFSCFRB***);**

Parameter

Parametername	Bedeutung
pFSCFRB	Zeiger auf den FRB, der quittiert werden soll.

Beschreibung

Die Funktion quittiert eine Nachricht des Typs M7MSG_CYCLE. Der FZ–Server wartet auf die Quittungen aller angemeldeten FRBs.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FSC_NO_SUCH_CYCLE	Unbekannter Zustand
M7E_FSC_NO_SUCH_FRB	FSCFRB ist nicht angemeldet
M7E_FRB_NOT_BUSY	Angegebener FRB ist nicht in Bearbeitung

Siehe auch

M7LinkCycle, M7UnLinkCycle

M7ConfirmDiagAlarm

Funktion

Diagnosealarm quittieren

Syntax

#include <m7api.h>
M7ERR_CODE M7ConfirmDiagAlarm(
M7DIAGALARM_FRB_PTR *pDAFRB***);**

Parameter

Parametername	Bedeutung
pDAFRB	Zeiger auf den FRB des Diagnosealarms, der quittiert werden soll.

Beschreibung

Die Funktion quittiert einen Diagnosealarm.

Nach dem Eintreffen eines Diagnosealarms kann von der auslösenden Baugruppe erst dann wieder ein neuer Diagnosealarm empfangen werden, wenn der gerade gemeldete Diagnosealarm quittiert wurde. Zwischenzeitlich eingetroffene Diagnoseereignisse werden auf der Baugruppe gespeichert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden bzw. hat keinen Alarm ausgelöst
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DPX2_FAULT	Fehler beim DP–Auftrag für die Alarmquittierung
M7E_SLAVE_TYPE	Alarme von DP–Normslaves müssen nicht quittiert werden
M7E_DP_SLAVE_STATE	DP–Slave ist nicht in DATA Zustand
M7E_INVAL_DEV	Baugruppe eines DP–Slaves ist nicht vorhanden

Zusätzliche Fehleranzeigen im FRB

Im FRB des angemeldeten Diagnosealarms können weitere Fehleranzeigen stehen. Diese können Sie mit folgendem C–Makro auslesen:

error = M7GetFRBErrCode(pDiagFrb);

Die Variable *error* muß dabei vom Typ *M7ERR_CODE* sein.

Die Bedeutung der FRB-Fehleranzeigen wird in der nachfolgenden Tabelle aufgeführt.

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug

Siehe auch

M7LinkDiagAlarm, M7GetDiagAlarmAddr, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo, M7GetDiagAlarmPType, M7UnlinkDiagAlarm

M7ConfirmIOAlarm

Funktion

Prozeßalarm quittieren

Syntax

#include <m7api.h>
M7ERR_CODE M7ConfirmIOAlarm(
M7IOALARM_FRB_PTR *pPAFRB***);**

Parameter

Parametername	Bedeutung
pPAFRB	Zeiger auf den FRB des Alarms, der quittiert werden soll.

Beschreibung

Die Funktion quittiert einen Prozeßalarm.

Nach dem Eintreffen eines Prozeßalarms kann erst dann wieder ein neuer Prozeßalarm derselben Baugruppe empfangen werden, wenn der gerade gemeldete Prozeßalarm quittiert wurde. Zwischenzeitlich eingetroffene Prozeßalarme werden auf der Baugruppe gespeichert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden bzw. hat keinen Alarm ausgelöst
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DPX2_FAULT	Fehler beim DP–Auftrag für die Alarmquittierung
M7E_DP_SLAVE_STATE	DP–Slave ist nicht in DATA Zustand
M7E_INVAL_DEV	Baugruppe eines DP–Slaves ist nicht vorhanden

Zusätzliche Fehleranzeigen im FRB

Im FRB des angemeldeten Prozeßalarms können weitere Fehleranzeigen stehen. Diese können Sie mit folgendem C-Makro auslesen:

error = M7GetFRBErrCode(pIOFrb);

Die Variable *error* muß dabei vom Typ *M7ERR_CODE* sein.

Die Bedeutung der FRB-Fehleranzeigen wird in der nachfolgenden Tabelle aufgeführt.

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PARITY	Lokalbus Parityfehler
M7E_QVZ	Lokalbus Quittungsverzug

Siehe auch **M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPType, M7UnLinkIOAlarm**

M7ConfirmPeriodicTimer

Funktion

Periodische Zeitnachricht quittieren

Syntax

#include <m7api.h>

VOID M7ConfirmPeriodicTimer(M7TFRB_PTR *pTFRB***);**

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den FRB, mit dem die periodische Zeitnachricht angemeldet wurde.

Beschreibung

Der Aufruf quittiert eine periodische Zeitnachricht. Wird beim Anmelden eines FRBs für periodische Zeitnachrichten eine Quittierung vereinbart, so sendet der Time–Server erst dann eine neue Zeitnachricht, wenn die vorhergehende Nachricht quittiert worden ist.

Der Aufruf ist als C-Makro realisiert. Es erfolgt keine weitere Überprüfung, ob der Pointer *pTFRB* einen gültigen FRB referenziert.

Die Anzahl der verlorenen Zeitnachrichten kann über die Funktion `M7GetLostPeriods` ermittelt werden.

Siehe auch

M7LinkPeriodicTimer, M7UnLinkPeriodicTimer, M7GetLostPeriods

M7ConfirmTransition

Funktion

Nachricht eines Betriebszustandsübergangs quittieren

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7ConfirmTransition(
                  M7TSFRB_PTR pTSFRB,
                  BOOL AllowTransition);
```

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger auf den FRB, der quittiert werden soll.
AllowTransition	Mit diesem Flag kann der Übergang nach ANLAUF oder RUN verhindert werden. Soll der Übergang nach ANLAUF oder RUN unterbleiben, so müssen Sie *FALSE* übergeben, ansonsten *TRUE*

Beschreibung

Die Funktion quittiert eine Nachricht des Typs M7MSG_TRANSITION.

Der BZÜ-Server wechselt erst dann in den neuen Betriebszustand, wenn alle Tasks, die einen FRB für diesen Betriebszustandsübergang angemeldet haben, ihre FRBs quittiert haben.

Bei der Anforderung aller Betriebszustände außer ANLAUF und RUN erfolgt der BZ–Übergang unabhängig davon, ob im Parameter *AllowTransition* *TRUE* oder *FALSE* angegeben wurde. Eine Quittung muß aber immer erfolgen.

Hinweis

Wenn der Übergang von STOP nach ANLAUF abgelehnt wird (***M7ConfirmTransition(.. AllowTransition=FALSE***), so wird das Erreichen des Betriebszustands ANLAUF nicht mit einer M7MSG_STATE-Nachricht gemeldet.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_OST_NO_SUCH_TRANSITION	Betriebszustandsübergang im FRB unbekannt
M7E_OST_NO_SUCH_FRB	FRB ist nicht in Bearbeitung

Siehe auch

M7GetTSReason, M7GetTSType, M7LinkTransition, M7UnLinkTransition

M7ConfirmZSAlarm

Funktion

Ziehen/Stecken–Alarm quittieren

Syntax

#include <m7api.h>
M7ERR_CODE M7ConfirmZSAlarm(
M7ZSALARM_FRB_PTR *pZSFRB***);**

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den Ziehen/Stecken FRB

Beschreibung

`M7ConfirmZSAlarm` quittiert einen Ziehen/Stecken–Alarm.

Die Funktion `M7ConfirmZSAlarm` muß nach der Auswertung der Ziehen/Stecken–Information vom Anwender aufgerufen werden, damit der vom System allokierte FRB mit der Ziehen/Stecken–Information wieder freigegeben wird.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB war nicht angemeldet

Siehe auch

M7GetZSAlarmAddr, M7GetZSAlarmIdent, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7LinkZSAlarm, M7UnLinkZSAlarm

M7CreateObject

Funktion

Ein S7-Objekt erzeugen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7CreateObject(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD Count,
                  VOID_PTR Ptr);
```

Parameter

Parametername	Bedeutung
ObjType	Kennzeichen des S7-Objekts. Die Kennzeichen der möglichen S7–Objekte, die vom Anwenderprogramm auf einem M7 eingerichtet werden können sind in der Tabelle 2-7 aufgelistet.
Part	Teilbereichsnummer. Die zulässigen Werte sind in der Tabelle 2-8 aufgelistet.
Count	Länge des S7–Objekts in Bytes; dieser Wert muß immer geradzahlig sein.
Ptr	Zeiger auf den Speicherbereich für den ablaufrelevanten Teil des Objektes. Wenn für Ptr der Wert NULL angegeben wird, so allokiert der Objektserver selbständig den Speicherplatz für das Objekt.

Beschreibung

Die Funktion erzeugt ein S7-Objekt, das durch die o. g. Parameter beschrieben wird. Danach wird das Objekt automatisch eingekettet.

Den Speicherplatz für das Objekt können Sie selbst bestimmen oder die Speicherplatzzuweisung dem Objektserver überlassen. Wenn Sie selbst den Speicherplatz bestimmen, so müssen Sie dafür sorgen, daß er für das gewünschte Objekt groß genug ist.

Hinweis

Wenn Sie einen Datenbaustein erzeugen, so können Sie dafür die Nummern (Parameter *part*) 0 bis 65535 benutzen. Der Bereich der Nummern wird **nicht** durch den bei der S7–CPU zulässigen Nummernbereich eingeschränkt.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch oder Anzahl der Bytes ist ungerade.
M7E_NO_MEM	Arbeitsspeicher belegt oder Fehler bei Speicheranforderung.
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_OBJ_EXISTS	Baustein bereits vorhanden.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PART	Teilbereich nicht vorhanden.
M7E_RESOURCE_LIMIT	Ressourcen überschritten
M7E_REM_OBJ	Aktion für remanente Objekte nicht erlaubt.

Siehe auch

M7StoreObject, M7DeleteObject, M7RemoveObject, M7LocateObject

M7DeleteObject

Funktion

S7-Objekt aus Arbeitsspeicher und BACKDIR löschen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7DeleteObject(
                  UBYTE ObjType,
                  UWORD Part);
```

Parameter

Parametername	Bedeutung
ObjType	Kennzeichen des S7-Objekts. Die Kennzeichen der möglichen S7–Objekte sind in der Tabelle 2-7 aufgelistet.
Part	Nummer des Teilbereichs. Die Teilbereichsnummern der S7–Objekte sind in der Tabelle 2-8 aufgelistet.

Beschreibung

Die Funktion löscht ein S7-Objekt, das durch die Parameter *ObjType* und *Part* beschrieben wird, aus dem Arbeitsspeicher **und** aus dem Katalog BACKDIR.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PART	Teilbereich nicht vorhanden.
M7E_REM_OBJ	Aktion für remanente Objekte nicht erlaubt.
M7E_WRITE_PROTECT	Objekt schreibgeschützt.

Siehe auch

M7CreateObject, M7LocateObject, M7RemoveObject, M7StoreObject

M7DiagMode

Funktion

Diagnose an– bzw. abmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7DiagMode(
                  UDWORD flags,
                  M7CONNID ConnID,
                  M7COMMFRB_PTR pCommFRB,
                  UBYTE_PTR pszUserName
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_BESYMSG	Betriebssystemmeldungen
	A_SYSMSG	System–Diagnosemeldungen
	A_USERMSG	Anwender–Diagnosemeldungen
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.	
pCommFRB	Function Request Block für asynchrone Benachrichtigung	
pszUserName	Mit diesem String (max. 8 Byte) identifiziert sich die Applikation gegenüber dem Server.	
MPrio	Priorität, mit der die Message verschickt wird (0–255).	

Beschreibung

Mit der Funktion `M7DiagMode` wird der Diagnosefilter des Benutzers neu eingestellt. Mit den aufzusummierenden Flags A_BESYMSG, A_SYSMSG und A_USERMSG kann sich eine Applikation für die entsprechenden Diagnosemeldungen anmelden. Nicht gesetzte Flags bedeuten eine Abmeldung.

Eintreffende Meldungen werden durch M7MSG_DIAG_MSG Messages angezeigt.

Nach Erhalt einer M7MSG_DIAG_MSG kann mit `M7GetCommRequest` die Auftragsnummer für die aktuelle Meldung abgefragt werden.

Folgende Auftragsnummern sind möglich:

Betriebssystemmeldungen haben die Auftragsnummer DIAG_BESYMSG.

System–Diagnosemeldungen haben die Auftragsnummer DIAG_SYSMSG.

Anwender–Diagnosemeldungen haben die Auftragsnummer DIAG_USERMSG.

Werden sowohl System– und Anwendermeldungen empfangen, ist die Auftragsnummer DIAG_SYS_USER_MSG.

Die Meldung selbst muß mit dem Aufruf `M7KEvent` ausgelesen werden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_PRIO	Falsche Priorität
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_NO_SUCH_FRB	*M7COMMFRB nicht in Bearbeitung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KEvent

M7DPNormDiagnose

Funktion

Normdiagnose eines DP–Slaves ermitteln

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7DPNormDiagnose(
                  M7IO_BASEADDR Baddr,
                  VOID_PTR pBuffer);
```

Parameter

Parametername	Bedeutung
Baddr	Basisadresse der ET–ER
pBuffer	Zeiger auf den Datenpuffer für das Normdiagnosetelegramm

Beschreibung

Die Funktion liefert die nach DP–Norm kodierte Diagnose eines DP–Slaves.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falsche Basisadresse
M7E_NORM_DIAG	Für die Baugruppe liegen keine Diagnosedaten vor.
M7E_NOT_IMPLEMENTED	L2–DP–Server nicht gestartet

Siehe auch

M7GetDiagAlarmInfo

M7GetCBBitOffset

Funktion

Ermittlung des Bit–Offsets innerhalb einer Callback–Funktion

Syntax

#include <m7api.h>
UBYTE M7GetCBBitOffset(
M7CBFRB_PTR *pCBFRB*
);

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB den Bit–Offset der Variablen, auf die eine weitere Anwendung mittels des S7–Objekt–Servers zuzugreifen versucht.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird der Bit–Offset übergeben.

Siehe auch

M7GetCBBuffer, M7GetCBByteOffset, M7GetCBCount, M7GetCBDataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBBuffer

Funktion

Ermittlung der Pufferadresse innerhalb einer Callback–Funktion

Syntax

#include <m7api.h>
VOID_PTR M7GetCBBuffer(
M7CBFRB_PTR *pCBFRB*
);

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktion übergebenen CBFRB die Adresse des Datenpuffers.

Hat sich die Task für einen schreibenden Zugriff mit einer Callback–Funktion angemeldet, so enthält der Puffer die Daten, mit denen Variablen des S7–Objekt–Servers überschrieben werden sollen.

Bei einem lesenden Zugriff dient er zur Aufnahme der zu lesenden Variablen des S7–Objekt–Severs.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf ein Zeiger auf den Puffer übergeben.

Siehe auch

M7GetCBBitOffset, M7GetCBByteOffset, M7GetCBCount, M7GetCBDataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBByteOffset

Funktion

Ermittlung des Byte–Offsets innerhalb einer Callback–Funktion

Syntax

```
#include <m7api.h>
UDWORD          M7GetCBByteOffset(
                  M7CBFRB_PTR pCBFRB);
```

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB den Byte–Offset der Variablen, auf die eine weitere Anwendung mittels des S7–Objekt–Servers zuzugreifen versucht.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird der Byte–Offset übergeben.

Siehe auch

M7GetCBBitOffset, M7GetCBBuffer, M7GetCBCount, M7GetCBDataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBCount

Funktion

Ermittlung der Anzahl der Elemente innerhalb einer Callback–Funktion

Syntax

```
#include <m7api.h>
UWORD          M7GetCBCount(
                 M7CBFRB_PTR pCBFRB);
```

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB die Anzahl der Elemente, auf die eine weitere Anwendung mittels des S7–Objekt–Servers zuzugreifen versucht.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird die Anzahl der Elemente übergeben.

Siehe auch

M7GetCBBitOffset,M7GetCBBuffer, M7GetCBByteOffset, M7GetCBDataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBDataType

Funktion

Ermittlung des Datentyps innerhalb einer Callback–Funktion

Syntax

#include <m7api.h>
UBYTE M7GetCBDataType(
M7CBFRB_PTR *pCBFRB*
);

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB den Datentyp der Variablen, auf die eine weitere Anwendung mit Hilfe des S7–Objekt–Servers zugreifen will.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf der Datentyp zurückgeliefert

Die möglichen Datentypen sind in der Tabelle 2-9 aufgelistet:

Siehe auch

M7GetCBBitOffset, M7GetCBBuffer, M7GetCBByteOffset, M7GetCBCount, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBFlags

Funktion

Ermittlung des Zugriffstyps innerhalb einer Callback–Funktion

Syntax

```
#include <m7api.h>
UWORD          M7GetCBFlags(
                 M7CBFRB_PTR pCBFRB);
```

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB den tatsächlichen Zugriffstyp (lesend, schreibend, löschend, usw.), mit dem eine weitere Anwendung versucht auf Variablen des S7–Objekt–Servers zuzugreifen.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf der tatsächliche Zugriffstyp zurückgeliefert

Die möglichen Zugriffstypen sind in der folgenden Tabelle aufgelistet:

Zugriffstyp	Typkennzeichen
S7–Objektvariable lesen	M7READ_ACCESS
S7–Objektvariablen schreiben	M7WRITE_ACCESS
S7–Objektvariablen erzeugen	M7CREATE_ACCESS
S7–Objektvariablen löschen	M7DELETE_ACCESS
S7–Objekt einketten	M7LINK_ACCESS

Siehe auch

M7GetCBBitOffset, M7GetCBBuffer,M7GetCBByteOffset, M7GetCBCount, M7GetCBDataType, M7GetCBObjType, M7GetCBPart

M7GetCBObjType

Funktion

Ermittlung des Typkennzeichen des S7–Objeks innerhalb einer Callback–Funktion

Syntax

#include <m7api.h>

UBYTE M7GetCBObjType(
M7CBFRB_PTR *pCBFRB***);**

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den CBFRB, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB das Typkennzeichen des S7–Objekts, auf den eine weitere Anwendung zuzugreifen versucht.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf das Typkennzeichen des S7–Objekttyps zurückgeliefert (siehe Tabelle 2-7).

Siehe auch

M7GetCBBitOffset, M7GetCBBuffer,M7GetCBByteOffset, M7GetCBCount, M7GetCBDataType, M7GetCBFlags, M7GetCBPart

M7GetCBPart

Funktion

Ermittlung der Teilbereichsnummer des S7–Objeks innerhalb einer Callback–Funktion

Syntax

```
#include <m7api.h>
UWORD          M7GetCBPart(
                 M7CBFRB_PTR pCBFRB);
```

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den *CBFRB*, der beim Aufruf der Callback–Funktion vom M7–API übergeben wird.

Beschreibung

Der Aufruf ermittelt aus dem beim Aufruf einer Callback–Funktionen übergebenen CBFRB die Teilbereichsnummer des S7–Objekts, auf den eine weitere Anwendung zuzugreifen versucht.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf das Typkennzeichen des S7–Objekttyps zurückgeliefert (siehe Tabelle 2-7).

Siehe auch

M7GetCBBitOffset, M7GetCBBuffer, M7GetCBByteOffset, M7GetCBCount, M7GetCBDataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCommRcvLen

Funktion

Länge der Empfangsdaten nach `M7PBKBrcv`**–Aufruf ermitteln**

Syntax

```
#include <m7api.h>
UDWORD          M7GetCommRcvLen(
                  M7COMMFRB_PTR pFRB);
```

Parameter

Parametername	Bedeutung
pFRB	Zeiger des FRB, aus dem die Empfangslänge gelesen werden soll.

Beschreibung

Der Aufruf `M7GetCommRcvLen` ermittelt aus dem FRB die Anzahl der empfangenen Bytes nach einem `M7PBKBrcv` Aufruf.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Die Anzahl der durch `M7PBKBrcv` empfangenen Bytes wird zurückgegeben

Siehe auch

M7PBKBrcv, M7GetCommStatus

M7GetCommRequest

Funktion

Auftragsnummer ermitteln

Syntax

#include <m7api.h>

UDWORD M7GetCommRequest(
M7COMMFRB_PTR *pFRB*);

Parameter

Parametername	Bedeutung
pFRB	Zeiger des FRB, aus dem die Auftragsnummer gelesen werden soll.

Beschreibung

Der Aufruf `M7GetCommRequest` ermittelt nach dem Empfang einer M7MSG_PBK_DONE–, M7MSG_PBK_NDR–, M7MSG_BUB_NDR– oder M7MSG_DIAG_MSG–Nachricht aus dem über *pFRB* referenzierten FRB die zugehörige Auftragsnummer.

Die Nachrichten werden von den PBK–, BuB–, und Diagnose–Aufrufen gesendet.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Die Auftragsnummer wird zurückgegeben

Siehe auch

M7PBKBrcv, M7PBKBsend, M7PBKGet, M7PBKPut, M7BUBCycRead, M7DiagMode, M7GetCommStatus

M7GetCommStatus

Funktion

Return–Status Abfrage der Applikationsbeziehung

Syntax

```
#include <m7api.h>
UWORD          M7GetCommStatus(
                 M7COMMFRB_PTR pFRB);
```

Parameter

Parametername	Bedeutung
pFRB	Zeiger des FRB, aus dem der Status gelesen werden soll.

Beschreibung

Der Aufruf `M7GetCommStatus` wertet den *pFRB* aus, nachdem eine M7MSG_PBK_DONE oder M7MSG_PBK_NDR Message empfangen wurde. Diese werden von den Aufrufen `M7PBKPut`, `M7PBKGet`, `M7PBKBsend` oder `M7PBKBrcv` gesendet.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Mögliche Ergebnisse sind in der folgenden Tabelle aufgeführt:

Status	Bedeutung
M7COMMSTATE_OK	Auftrag ohne Fehler beendet
M7COMMSTATE_NO_CONN	Kommunikationsprobleme
M7COMMSTATE_NACK	negative Quittung, Funktion nicht ausführbar
M7COMMSTATE_RID_ UNKNOWN	R_ID ist nicht bekannt bzw. es war kein Receive aufgerufen.
M7COMMSTATE_WRONG_ DATA	Anzahl der Datenbereiche oder einzelne Datentypen stimmen nicht überein
M7COMMSTATE_RES_REQ	Resetanforderung eingetroffen
M7COMMSTATE_REM_ BLCK_DISABLED	remoter Baustein DISABLED
M7COMMSTATE_REM_ WRONG_STATE	remoter Partner im falschen Zustand
M7COMMSTATE_REM_ ACCESS_DENIED	Zugriffsfehler bei remotem Partner
M7COMMSTATE_OVERRUN	Empfangsdaten wurden von neueren Daten überschrieben
M7COMMSTATE_MEM_ ACCESS_DENIED	Zugriff auf lokaler Anwenderspeicher nicht möglich

Status	Bedeutung
M7COMMSTATE_NOT_ FINISHED	Vorangegangener Auftrag noch nicht abgeschlossen
M7COMMSTATE_TERM_ BY_USER	Auftrag wurde durch den Benutzer abgebrochen.

Siehe auch **M7PBKBrcv, M7PBKBsend, M7PBKGet, M7PBKPut, M7GetCommRequest**

M7GetConnStatus

Funktion

Zustand einer Applikationsbeziehung abfragen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7GetConnStatus(
                  M7CONNID ConnID,
                  M7_CONN_STATE_PTR pConnState);
```

Beschreibung

Die Funktion `M7GetConnStatus` erlaubt es, den Zustand einer mit *ConnID* angegebenen Applikationsbeziehung zu ermitteln.

Folgene Zustände sind definiert (M7_CONN_STATE):

M7_CNST_CLOSED	Die Applikationsbeziehung ist abgebaut
M7_CNST_CONNECTING	Die Applikationsbeziehung wird gerade aufgebaut
M7_CNST_CONNECTED	Die Applikationsbeziehung ist aufgebaut
M7_CNST_DISCONNECTING	Applikationsbeziehung wird gerade abgebaut

Die K–Bus–Funktionen `M7KAbort` und `M7GetConnStatus` können unabhängig vom Zustand einer Applikationsbeziehung mit einer gültigen *ConnID* aufgerufen werden.

Alle anderen für Applikationsbeziehungen spezifischen K–Bus–Funktionen werden nur im Zustand M7_CNST_CONNECTED bearbeitet. Bei anderen Zuständen werden diese Aufrufe mit M7E_KSUB_CONN_CLOSED abgelehnt.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung

Siehe auch

M7KAbort, M7KInitiate

M7GetDiagAlarmAddr

Funktion

Logische Basisadresse zu Diagnosealarm aus FRB lesen

Syntax

#include <m7api.h>
M7IO_BASEADDR M7GetDiagAlarmAddr(
M7DIAGALARM_FRB_PTR *pDiagFrb***);**

Parameter

Parametername	Bedeutung
pDiagFrb	Zeiger des FRB, aus dem die Adresse gelesen werden soll.

Beschreibung

Der Aufruf liefert zu einem Diagnosealarm die logische Basisadresse der alarmauslösenden Baugruppe aus dem über *pDiagFrb* referenzierten FRB.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf die logische Basisadresse der Baugruppe übergeben, die den Diagnosealarm ausgelöst hat.

Siehe auch

M7LinkDiagAlarm, M7UnLinkDiagAlarm, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo, M7GetDiagAlarmPType, M7ConfirmDiagAlarm

M7GetDiagAlarmBusy

Funktion

Status eines Diagnosealarms von M7/S7-CPU abfragen

Syntax

#include <m7api.h>
BOOL M7GetDiagAlarmBusy(
M7ERR_CODE_PTR *pError***);**

Parameter

Parametername	Bedeutung
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE.

Beschreibung

Die Funktion ermittelt, ob ein an die M7/S7-CPU gesendeter Diagnosealarm von der M7/S7-CPU quittiert wurde.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert eine Kennung des aktuellen Alarmzustandes übergeben. Die Bedeutung der Zustandskennungen können Sie der nachfolgenden Tabelle entnehmen.

Zustandskennung	Bedeutung
TRUE	Der Alarm steht noch zur Bearbeitung an.
FALSE	Der Alarm wurde von der S7/M7-CPU erkannt und bearbeitet.

Fehlercodes

**pError* ist immer 'M7SUCCESS'

Siehe auch

M7SendDiagAlarm

M7GetDiagAlarmInfo

Funktion

Diagnoseinformation aus FRB lesen

Syntax

#include <m7api.h>

void **M7GetDiagAlarmInfo(**
M7DIAGALARM_FRB_PTR *pDiagFrb,*
UBYTE_PTR **Info*);**

Parameter

Parametername	Bedeutung
pDiagFrb	Zeiger des FRB, aus dem die Diagnoseinformation gelesen werden soll.
Info	Zeiger auf einen Puffer, in den die 4 Bytes mit der Diagnoseinformation hinterlegt werden sollen.

Beschreibung

Der Aufruf liefert zu einem Diagnosealarm die 4 Bytes mit der Diagnoseinformation aus dem über *pDiagFrb* referenzierten FRB. Die Diagnoseinformation ist baugruppenspezifisch.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Die Diagnoseinformation wird von der Funktion in dem mit *Info* referenzierten Puffer hinterlegt.

Siehe auch

M7LinkDiagAlarm, M7UnLinkDiagAlarm, M7GetDiagAlarmBusy, M7GetDiagAlarmAddr, M7GetDiagAlarmPType, M7ConfirmDiagAlarm

M7GetDiagAlarmPType

Funktion

Kennzeichen der Signalbaugruppe eines Diagnosealarms aus FRB lesen

Syntax

```
#include <m7api.h>
UBYTE          M7GetDiagAlarmPType(
                 M7DIAGALARM_FRB_PTR pDiagFrb);
```

Parameter

Parametername	Bedeutung
pDiagFrb	Zeiger des FRB, aus dem das Kennzeichen gelesen werden soll.

Beschreibung

Der Aufruf liefert zu einem Diagnosealarm das Kennzeichen der Signalbaugruppe aus dem über *pDiagFrb* referenzierten FRB, das beim Aufruf der Funktion `M7LinkDiagAlarm` mit dem Parameter *pType* angegeben wurde.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf das Kennzeichen für den Typ der Baugruppe übergeben.

Peripherietyp	Bedeutung
M7IO_IN	Baugruppe ist Eingabebaugruppe
M7IO_OUT	Baugruppe ist Ausgabebaugruppe

Siehe auch

M7LinkDiagAlarm, M7UnLinkDiagAlarm, M7GetDiagAlarmBusy, M7GetDiagAlarmAddr, M7GetDiagAlarmInfo, M7ConfirmDiagAlarm

M7GetFlags

Funktion

Angemeldete Zugriffsart aus FRB lesen

Syntax

#include <m7api.h>
UWORD M7GetFlags(M7OBJFRB_PTR *pOBJFRB***);**

Parameter

Parametername	Bedeutung
pOBJFRB	Zeiger auf den OBJFRB, der bei der Anmeldung für eine Benachrichtigung bei S7–Objektzugriff übergeben wurde.

Beschreibung

Der Aufruf liefert den Parameter *flags* aus dem bei der Anmeldung mittels `M7LinkDataAccess` referenzierten OBJFRBs.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf die Parameter *flags* übergeben. Der Parameter *flags* repräsentiert die bei der Anmeldung spezifizierte Zugriffsart.

Die möglichen Zugriffsarten sind in der folgenden Tabelle aufgelistet:

Art des Zugriffs	Kennzeichen
S7–Objekte lesen	M7READ_ACCESS
S7–Objekte schreiben	M7WRITE_ACCESS
S7–Objekte erzeugen	M7CREATE_ACCESS
S7–Objekte löschen	M7DELETE_ACCESS
S7–Objekt einketten	M7LINK_ACCESS

Siehe auch

M7LinkDataAccess, M7UnLinkDataAcess, M7GetObjType, M7GetPart

M7GetFRBErrCode

Funktion **FRBs lesen**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7GetFRBErrCode(
                M7FRBHEADER_PTR pFRBHeader);
```

Parameter

Parametername	Bedeutung
pFrbHeader	Zeiger des FRB–Headers, dessen Fehlerkennzeichen gelesen werden soll.

Beschreibung

Der Aufruf liefert das Fehlerkennzeichen des über *pFrbHeader* referenzierten FRBs. Das Fehlerkennzeichen gibt den allgemeinen Fehlercode der beim Handling des FRBs auftreten kann.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird von der Funktion das Fehlerkennzeichen des referenzierten FRBs übergeben.

Die möglichen Fehlerkennzeichen sind abhängig vom jeweiligen Typ des FRBs.

Siehe auch **M7GetFRBTag, M7SetFRBTag**

M7GetFRBTag

Funktion

Kennzeichen eines FRBs lesen

Syntax

#include <m7api.h>

UWORD M7GetFRBTag(
M7FRBHEADER_PTR *pFRBHeader***);**

Parameter

Parametername	Bedeutung
pFRBHeader	Zeiger des FRB, dessen Kennzeichen gelesen werden soll.

Beschreibung

Der Aufruf liefert das Kennzeichen des über den Parameter *pFRBHeader* referenzierten FRBs.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird von der Funktion das Kennzeichen des referenzierten FRBs übergeben.

Siehe auch

M7SetFRBTag, M7GetFRBErrCode

M7GetFSCType

Funktion

Typ der Benachrichtigung durch FZ–Server aus FRB lesen

Syntax

```
#include <m7api.h>
UWORD          M7GetFSCType(
                 M7FSCFRB_PTR pFSCFRB);
```

Parameter

Parametername	Bedeutung
pFSCFRB	Zeiger des FRB, aus dem die Adresse gelesen werden soll.

Beschreibung

Mit Hilfe des Aufrufs kann aus einer Nachricht des FZ–Servers der Dienst (Systemkontrollpunkt, Freier Zyklus, usw.) ermittelt werden, für den sich die Anwendung beim FZ–Server angemeldet hat. Alle vom FZ–Server gesendeten Nachrichten haben das Nachrichtenkennzeichen M7MSG_CYCLE.

Dier Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird der Typ des Dienstes zurückgegeben.

Die möglichen Dienste des FZ–Servers sind in der folgenden Tabelle aufgelistet:

Dienste des FZ–Servers	Kennzeichen
Systemkontrollpunkt	M7S_CYCLECONTROLPOINT
Freier Zyklus	M7S_FREECYCLE
ANLAUF	M7S_STARTUPCYCLE
Zykluszeitüberlauf	M7S_CYCLEOVERFLOW

Siehe auch

M7LinkCycle, M7ConfirmCycle, M7UnLinkCycle

M7GetIOAlarmAddr

Funktion

Logische Basisadresse zu Prozeßalarm aus FRB lesen

Syntax

#include <m7api.h>
M7IO_BASEADDR M7GetIOAlarmAddr(
M7IOALARM_FRB_PTR *pIOFrb***);**

Parameter

Parametername	Bedeutung
pIOFrb	Zeiger des FRB, aus dem die Adresse gelesen werden soll.

Beschreibung

Der Aufruf liefert zu einem Prozeßalarm die logische Basisadresse der alarmauslösenden Baugruppe aus dem über *pIOFrb* referenzierten FRB.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird von der Funktion die logische Basisadresse der Baugruppe übergeben, die den Prozeßalarm ausgelöst hat.

Siehe auch

M7LinkIOAlarm, M7UnLinkIoAlarm, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPType, M7ConfirmIOAlarm

M7GetIOAlarmBusy

Funktion

Status eines Prozeßalarms von M7/S7-CPU abfragen

Syntax

#include <m7api.h>
BOOL M7GetIOAlarmBusy(
M7ERR_CODE_PTR *pError***);**

Parameter

Parametername	Bedeutung
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE.

Beschreibung

Die Funktion ermittelt, ob ein an die M7/S7-CPU gesendeter Prozeßalarm von der M7/S7-CPU quittiert wurde.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert eine Kennung des aktuellen Alarmzustandes übergeben. Die Bedeutung der Zustandskennungen können Sie der nachfolgenden Tabelle entnehmen.

Zustandskennung	Bedeutung
TRUE	Der Alarm steht noch zur Bearbeitung an.
FALSE	Der Alarm wurde von der S7-CPU erkannt und bearbeitet.

Fehlercodes

**pError* ist immer 'M7SUCCESS'

Siehe auch

M7SendIOAlarm

M7GetIOAlarmMask

Funktion

Alarmmaske zu einem Prozeßalarm aus FRB lesen

Syntax

#include <m7api.h>
UDWORD M7GetIOAlarmMask(
M7IOALARM_FRB_PTR *pIOFrb***);**

Parameter

Parametername	Bedeutung
pIOFrb	Zeiger des FRB, aus dem die Alarmmaske gelesen werden soll.

Beschreibung

Der Aufruf liefert zu einem Prozeßalarm die Alarmmaske aus dem über *pIOFrb* referenzierten FRB.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf die Alarmmaske aus dem FRB übergeben.

Siehe auch

M7ConfirmIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmPType, M7GetIOAlarmState, M7LinkIOAlarm, M7UnLinkIOAlarm,

M7GetIOAlarmState

Funktion

Zusatzinformation eines Prozeßalarms aus FRB lesen

Syntax

#include <m7api.h>
UDWORD M7GetIOAlarmState(
M7IOALARM_FRB_PTR *pIOFrb***);**

Parameter

Parametername	Bedeutung
pIOFrb	Zeiger des FRB, aus dem die Zusatzinformation gelesen werden soll.

Beschreibung

Der Aufruf liefert zu einem Prozeßalarm die Zusatzinformation aus dem über *pIOFrb* referenzierten FRB. Die Zusatzinformation ist baugruppenspezifisch und wird im Intel–Format geliefert.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf die Zusatzinformation aus dem FRB übergeben.

Siehe auch

M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7UnLinkIOAlarm, M7GetIOAlarmPType, M7ConfirmIOAlarm

M7GetIOAlarmPType

Funktion

Kennzeichen der Signalbaugruppe eines Prozeßalarms aus FRB lesen

Syntax

#include <m7api.h>
UWORD M7GetIOAlarmPType(
M7IOALARM_FRB_PTR *pIOFrb***);**

Parameter

Parametername	Bedeutung
pIOFrb	Zeiger des FRB, aus dem das Kennzeichen gelesen werden soll.

Beschreibung

Der Aufruf liefert zu einem Prozeßalarm das Kennzeichen der Signalbaugruppe aus dem über *pIOFrb* referenzierten FRB, das beim Aufruf der Funktion `M7LinkIOAlarm` mit dem Parameter *pType* angegeben wurde.

Der Aufruf ist als C–Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf das Kennzeichen für den Peripherietyp übergeben.

Peripherietyp	Bedeutung
M7IO_IN	Baugruppe ist Eingabebaugruppe
M7IO_OUT	Baugruppe ist Ausgabebaugruppe

Siehe auch

M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7GetIOAlarmState, M7UnLinkIOAlarm, M7ConfirmIOAlarm

M7GetLostPeriods

Funktion

Anzahl verlorener periodische Zeitnachrichten abfragen

Syntax

#include <m7api.h>
UDWORD M7GetLostPeriods(M7TFRB_PTR *pTFRB***);**

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den FRB, mit dem die periodischen Zeitnachrichten angemeldet wurden.

Beschreibung

Mit dieser Funktion wird die Anzahl der auf Grund fehlender Quittung nicht gesendeten periodischen Zeitnachrichten ermittelt. Anschließend wird der systeminterne Zähler für die verlorenen periodischen Zeitnachrichten gelöscht.

Rückgabewert

Als Rückgabewert wird von der Funktion die Anzahl der verlorenen periodischen Zeitnachrichten übergeben.

Siehe auch

M7LinkPeriodicTimer, M7ConfirmPeriodicTimer, M7UnLinkPeriodicTimer

M7GetObjectInfo

Funktion

Information über Datenstruktur eines S7-Objektes lesen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7GetObjectInfo(
                  UBYTE ObjType,
                  UWORD Part,
                  M7OBJ_INFO_PTR pObjInfo);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objektes (vgl. Tabelle 2-7.)
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig.
pObjInfo	Zeiger auf einen Speicherbereich mit der Datenstruktur **M7OBJ_INFO**, in der die Information über das S7-Objekt hinterlegt wird.

Beschreibung

Die Funktion liefert alle Informationen über die Datenstruktur eines S7-Objektes, das durch die Parameter *ObjType* und *Part* beschrieben wird. Der Speicherplatz für die Informationen muß vom aufrufenden Programm zur Verfügung gestellt werden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PART	Teilbereich nicht vorhanden.
M7E_OBJ	Objekttyp nicht unterstützt.

Siehe auch

M7CreateObject, M7DeleteObject, M7RemoveObject, M7LocateObject, M7StoreObject

M7GetObjType

Funktion

Ermittlung des Typkennzeichen bei S7–Objektzugriff

Syntax

```
#include <m7api.h>
UBYTE          M7GetObjType(
                 M7OBJFRB_PTR pOBJFRB);
```

Parameter

Parametername	Bedeutung
pOBJFRB	Zeiger auf den OBJFRB, der bei der Anmeldung für S7–Objektzugriff referenziert wurde.

Beschreibung

Der Aufruf ermittelt aus dem bei der Benachrichtigung durch den S7–Objekt–Server referenzierten OBJFRB das Typkennzeichen des Objekts auf das zugegriffen wurde.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf das Typkennzeichen des S7–Objekttyps zurückgeliefert

Die möglichen Typkennnzeichen der adressierbaren S7–Objekte finden Sie in der Tabelle 2-7.

Siehe auch

M7LinkDataAccess, M7UnLinkDataAccess, M7GetPart, M7GetFlags

M7GetPart

Funktion

Ermittlung der Teilbereichsnummer bei S7–Objektzugriff

Syntax

#include <m7api.h>
UBYTE M7GetPart(
M7OBJFRB_PTR *pOBJFRB***);**

Parameter

Parametername	Bedeutung
pOBJFRB	Zeiger auf den OBJFRB, der bei der Anmeldung für S7–Objektzugriff referenziert wurde.

Beschreibung

Der Aufruf ermittelt aus dem bei der Benachrichtigung durch den S7–Objekt–Server referenzierten OBJFRB die Teilbereichsnummer des Objekts auf das zugegriffen wurde.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf die Teilbereichsnummer des S7–Objekttyps zurückgeliefert

Die möglichen Teilbereichsnummern der adressierbaren S7–Objekte finden Sie folgender Tabelle:

S7-Objekt	Typkennzeichen	Teilbereichsnummer
Datenbaustein	M7D_DB	DB–Nummer
Parameterdatensätze,Lesen	M7D_PAR_READ	DS–Nummer
Parameterdatensätze,Schreiben	M7D_PAR_WRITE	DS–Nummer

Siehe auch

M7LinkDataAccess, M7UnLinkDataAccess, M7GetObjType, M7GetFlags

M7GetPduSize

Funktion

Maximale PDU–Größe abfragen

Syntax

#include <m7api.h>
M7ERR_CODE M7GetPduSize (
M7CONNID *ConnID*,
UDWORD **pnPduSize*);

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
pnPduSize	Puffer für PDU Größe.

Beschreibung

Die Funktion liefert die maximale PDU–Größe für eine Verbindung.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7PBKGet, M7PBKPut, M7BUBRead, M7BUBWrite

M7GetPeriod

Funktion

Vielfaches der Zeitbasis aus TFRB ermitteln

Syntax

```
#include <m7api.h>
UDWORD          M7GetPeriod
                  M7TFRB_PTR pTFRB);
```

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf FRB, aus dem das Vielfache(Parameter: *TimeBase*) der Zeitbasis gelesen werden soll.

Beschreibung

Der Aufruf ermittelt aus dem von einer periodischen oder singulären Zeitnachricht referenzierten TFRB den Parameter *Period*. Der Parameter *Period* wird beim Anmelden des FRBS spezifiziert.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf der Parameter *Period* aus dem referenzierten TFRB übergeben.

Siehe auch

M7LinkPeriodicTimer, M7LinkOneShotTimer, M7GetTimeBase

M7GetPIErrorAddr

Funktion

Adresse des Prozeßabbildes mit Transferfehler ermitteln

Syntax

```
#include <m7api.h>
                    M7GetPIErrorAddr(
                      void *PIErrMsgBuf,
                      M7IO_LOGADDR Addr);
```

Parameter

Parametername	Bedeutung
PIErrMsgBuf	ProzeßabbildtransferfehlerMessage Puffer
Addr	Adresse des Prozeßabbildes, bei dem ein Transferfehler aufgetreten ist.

Beschreibung

Der Aufruf ermittelt aus der Prozeßabbildtransferfehler Message die Adresse, bei der ein Transferfehler aufgetreten ist und liefert sie in der Variablen *Addr* zurück.

Der Aufruf ist als C–Makro realisiert.

Siehe auch

M7GetPIErrorPIType, M7LinkPIError, M7UnLinkPIError

M7GetPIErrorPIType

Funktion

Typ des Prozeßabbildes mit Transferfehler ermitteln

Syntax

```
#include <m7api.h>
                    M7GetPIErrorPIType(
                      void *PIErrMsgBuf
                      UBYTE PIType);
```

Parameter

Parametername	Bedeutung
PIType	Typ des Prozeßabbilds, bei dem der Fehler aufgetreten ist.
	M7IO_PII Prozeßabbild der Eingänge
	M7IO_PIQ Prozeßabbild der Ausgänge

Beschreibung

Der Aufruf ermittelt aus der Prozeßabbildtransferfehler Message den Typ des Prozeßabbildes, bei dem ein Transferfehler aufgetreten ist und liefert ihn in der Variablen *PIType* zurück.

Der Aufruf ist als C–Makro realisiert.

Siehe auch

M7GetPIErrorAddr, M7LinkPIError, M7UnLinkPIError

M7GetResetCause

Funktion **Reset–Ursache abfragen**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7GetResetCause(
                  UDWORD          *pState);
```

Parameter

Parametername	Bedeutung
pState	Zeiger auf den Zustand. Wenn eines der folgenden Bits gesetzt ist, dann gilt der dazugehörige Zustand. Es können auch mehrere Bits gleichzeitig gesetzt sein: M7WD_RESET – Das System wurde das letzte Mal durch den Watchdog zurückgesetzt. M7KEY_RESET – Das System wurde das letzte Mal durch den Schlüsselschalter zurückgesetzt. Wenn keines der oben genannten Bits gesetzt ist, dann wurde das System durch einen Netz–Aus zurückgesetzt.

Beschreibung Die Funktion liefert der Applikation Informationen darüber, aus welchem Grund das System das letzte Mal gestartet wurde.

Rückgabewert = M7SUCCESS: Die Funktion wurde erfolgreich ausgeführt.

M7GetState

Funktion **Betriebszustand abfragen**

Syntax

```
#include <m7api.h>
UWORD          M7GetState(void);
```

Beschreibung Die Funktion liefert den aktuellen Betriebszustand zurück.

Rückgabewert Die Funktion liefert im Rückgabewert eine Kennung für den augenblicklichen Betriebszustand zurück. Die Bedeutung der Zustandskennungen können Sie der nachfolgenden Tabelle entnehmen.

Zustandskennung	Bedeutung
M7STATE_STOP	Betriebszustand STOP
M7STATE_STARTUP	Betriebszustand ANLAUF
M7STATE_RUN	Betriebszustand RUN
M7STATE_HALT	Betriebszustand HALT
M7STATE_RESET	Betriebszustand URLÖSCHEN

Siehe auch **M7LinkState, M7UnLinkState, M7RequestState**

M7GetTime

Funktion **Datum/Uhrzeit auslesen**

Syntax

#include <m7api.h>
M7ERR_CODE M7GetTime(M7TIME_DATE_PTR *pDateTime***);**

Parameter

Parametername	Bedeutung
pDateTime	Zeiger auf Speicherbereich mit Datum-Zeit-Struktur

Beschreibung

Die Funktion liest die systeminterne Zeit und das systeminterne Datum und speichert beide in dem mit *pDateTime* referenzierten Speicherbereich ab .

Zum Aufbau der Struktur M7TIME_DATE siehe Kapitel 3.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Siehe auch **M7SetTime**

M7GetTimeBase

Funktion

Zeitbasis aus TFRB ermitteln

Syntax

```
#include <m7api.h>
UWORD          M7GetITimeBase(
                 M7TFRB_PTR pTFRB);
```

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf FRB, aus dem die Zeitbasis (Parameter: *TimeBase*) gelesen werden soll.

Beschreibung

Der Aufruf ermittelt aus dem von einer periodischen oder singulären Zeitnachricht referenzierten TFRB den Parameter *TimeBase*. Der Parameter *TimeBase* wird beim anmelden des FRBS spezifiziert.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf der Parameter *TimeBase* aus dem TFRB übergeben. Mögliche Werte für *TimeBase* sind:

Rückgabewert	Bedeutung	
TimeBase	Wert für die Zeitbasis:	
	M7TB_1MS:	1 ms
	M7TB_10MS:	10 ms
	M7TB_100MS:	100 ms
	M7TB_1S:	1s

Siehe auch

M7LinkPeriodicTimer, M7LinkOneShotTimer, M7GetPeriod

M7GetTSReason

Funktion

Grund für Betriebszustand/-übergang aus FRB lesen

Syntax

#include <m7api.h>
UWORD M7GetTSReason(M7TSFRB_PTR *pTSFRB***);**

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger des FRB, aus dem der Grund für den Betriebszustand bzw. Betriebszustandsübergang gelesen werden soll.

Beschreibung

Mit dem Makro `M7GetTSReason` kann nach Erreichen eines Zustandes abgefragt werden, warum der Wechsel in diesen Zustand durch `M7RequestState` gegeben wurde. Es wird der Wert ausgewertet, der beim Aufruf von `M7RequestState` im Parameter *Reason* angegeben wurde.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Als Rückgabewert wird vom Aufruf der Grund aus dem FRB übergeben.

Siehe auch

M7LinkTransition, M7UnLinkTransition, M7GetTSType, M7ConfirmTransition

M7GetTSType

Funktion

Betriebszustand aus einem FRB lesen

Syntax

#include <m7api.h>
UWORD M7GetTSType(M7TSFRB_PTR *pTSFRB***);**

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger des FRB, aus dem der Betriebszustand gelesen werden soll.

Beschreibung

Der Aufruf liefert aus einem TSFRB des BZ–Servers eine Kennung für den Betriebszustand bzw. Betriebszustandsübergang.

Der Aufruf ist als C-Makro realisiert.

Rückgabewert

Bei Erhalt eines Nachricht vom Typ M7MSG_STATE (Anmelden über `M7LinkState`) bzw. M7MSG_REQ_FINISHED (Anforderung über `M7RequestState`) sind die folgenden Kennungen im referenzierten TSFRB möglich:

Kennung	Bedeutung
M7STATE_STOP	M7 befindet sich im Betriebszustand STOP
M7STATE_STARTUP	M7 befindet sich im Betriebszustand ANLAUF
M7STATE_RUN	M7 befindet sich im Betriebszustand RUN
M7STATE_HALT	M7 befindet sich im Betriebszustand HALT
M7STATE_RESET	M7 befindet sich im Betriebszustand URLÖSCHEN

Bei Erhalt einer Nachricht vom Typ M7MSG_TRANSITION (Anmelden über `M7LinkTransition`) sind die folgenden Kennungen im referenzierten TSFRB möglich:

Kennung	Bedeutung
M7TRANS_STOPSTARTUP	Betriebszustanssübergang von STOP nach ANLAUF angefordert
M7TRANS_STOPRESET	Betriebszustandsübergang von STOP nach URLÖSCHEN angefordert
M7TRANS_STARTUPSTOP	Betriebszustandsübergang von ANLAUF nach STOP angefordert

Kennung	Bedeutung
M7TRANS_STARTUPRUN	Betriebszustandsübergang von ANLAUF nach RUN angefordert
M7TRANS_STARTUPHALT	Betriebszustandsübergang von ANLAUF nach HALT angefordert
M7TRANS_RUNSTOP	Betriebszustandsübergang von RUN nach STOP angefordert
M7TRANS_RUNHALT	Betriebszustandsübergang von RUN nach HALT angefordert
M7TRANS_HALTSTOP	Betriebszustandsübergang von HALT nach STOP angefordert
M7TRANS_HALTSTARTUP	Betriebszustandsübergang von HALT nach ANLAUF angefordert
M7TRANS_HALTRUN	Betriebszustandsübergang von HALT nach RUN angefordert
M7TRANS_RESETSTOP	Betriebszustandsübergang von URLÖSCHEN nach STOP angefordert

Siehe auch

M7LinkState, M7UnLinkState, M7RequestState, M7GetTSReason, M7LinkTransition, M7UnLinkTransition, M7ConfirmTransition

M7GetZSAlarmAddr

Funktion

Basisadresse einer I/O–Baugruppe ermitteln

Syntax

```
#include <m7api.h>
M7IO_BASEADDR  M7GetZSAlarmAddr(
                   M7ZSALARM_FRB_PTR pZSFRB,
                   UWORD SlotNum);
```

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den ZSFRB, aus dem die Basisadresse der I/O–Baugruppe ermittelt wird.
SlotNum	Nummer des Einsteckplatzes, auf dem die Baugruppe steckt. Die Nummer des Steckplatzes muß innerhalb des Bereichs 1 ... MAX_SLOT_400 sein. Die Konstante MAX_SLOT_400 kennzeichnet die maximale Anzahl von Steckplätzen im System S7–400.

Beschreibung

Der Aufruf liefert zu einem Ziehen/Stecken Alarm die Basisadresse der Baugruppe auf dem Steckplatz mit Nummer *SlotNum*.

Der Aufruf ist als C-Makro realisiert.

Die Funktion wird nur im System SIMATIC S7–400 unterstützt.

Rückgabewert

Als Rückgabewert wird vom Aufruf die Basisadresse zurückgegeben.

Siehe auch

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7GetZSAlarmIdent

M7GetZSAlarmIdent

Funktion

Identifikationskennung einer I/O–Baugruppe ermitteln

Syntax

#include <m7api.h>

UBYTE **M7GetZSAlarmIdent(**
M7ZSALARM_FRB_PTR *pZSFRB,*
UWORD *SlotNum*);

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den ZSFRB, aus dem die Identifikationsnummer der I/O–Baugruppe ermittelt wird.
SlotNum	Nummer des Einsteckplatzes auf dem die Baugruppe steckt. Die Nummer muß innerhalb des Bereichs 1 ... MAX_SLOT_400 sein. Die Konstante MAX_SLOT_400 kennzeichnet die maximale Anzahl von Steckplätzen im System S7–400.

Beschreibung

Der Aufruf liefert zu einem Ziehen/Stecken Alarm die Identifikationsnummer der Baugruppe auf dem Steckplatz mit Nummer *SlotNum*.

Der Aufruf ist als C-Makro realisiert.

Die Funktion wird nur im System SIMATIC S7–400 unterstützt.

Rückgabewert

Als Rückgabewert wird vom Aufruf die Identifikationskennung zurückgegeben. Die Identifikationskennung einer Baugruppe ist in der zugehörigen Hardware–Beschreibung erörtert.

Siehe auch

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmAddr, M7GetZSAlarmPType, M7GetZSAlarmMode

M7GetZSAlarmIMRBaddr

Funktion

Basisadresse der IM–Baugruppe, für die ein Ziehen/Stecken–Alarm gemeldet wurde, bestimmen

Syntax

#include <m7api.h>
M7IO_BASEADDR M7GetZSAlarmIMRBaddr(
M7ZSALARM_FRB_PTR *pZSFRB***);**

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf einen Ziehen/Stecken FRB

Beschreibung

Der Aufruf liefert zu einem Ziehen/Stecken Alarm Information über die Basisadresse der IM Baugruppe, die in einem Baugruppenträger bzw. S7–Slave steckt, in dem das Ereignis aufgetreten ist (CR_BADDR für den Zentralbaugruppenträger).

Der Aufruf ist als C-Makro realisiert.

Die Funktion wird nur im System SIMATIC S7–400 unterstützt.

Rückgabewert

Als Rückgabewert wird die Basisadresse der IM–Baugruppe zurückgegeben.

Siehe auch

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmPType, M7GetZSAlarmAddr, M7GetZSAlarmMode, M7GetZSAlarmIdent

M7GetZSAlarmMode

Funktion

Modus einer I/O–Baugruppe ermitteln

Syntax

```
#include <m7api.h>
UBYTE          M7GetZSAlarmMode(
                 M7ZSALARM_FRB_PTR pZSFRB,
                 UWORD SlotNum);
```

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den ZSFRB, aus dem der Modus der I/O–Baugruppe ermittelt wird.
SlotNum	Nummer des Einsteckplatzes auf dem die Baugruppe steckt. Die Nummer muß innerhalb des Bereichs 1 ... MAX_SLOT_400 sein. Die Konstante MAX_SLOT_400 kennzeichnet die maximale Anzahl von Steckplätzen im System S7–400.

Beschreibung

Der Aufruf liefert zu einem Ziehen/Stecken Alarm den Modus der Baugruppe auf dem Steckplatz mit Nummer *SlotNum*.

Der Aufruf ist als C-Makro realisiert.

Die Funktion wird nur im System SIMATIC S7–400 unterstützt.

Rückgabewert

Als Rückgabewert wird vom Aufruf eine Kennung für den Modus zurückgegeben. Die möglichen Werte sind in der folgenden Tabelle aufgelistet:

Kennung	Bedeutung
M7DEV_OK	Baugruppe ist in Ordnung
M7DEV_REM	Baugruppe wurde gezogen
M7DEV_PUT	Baugruppe wurde gesteckt

Siehe auch

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmAddr, M7GetZSAlarmPType, M7GetZSAlarmIdent

M7GetZSAlarmPType

Funktion

Peripherietyp einer I/O–Baugruppe bestimmen

Syntax

```
#include <m7api.h>
UBYTE          M7GetZSAlarmPType(
                 M7ZSALARM_FRB_PTR pZSFRB,
                 UWORD SlotNum);
```

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den ZSFRB, aus dem der Typ der I/O–Baugruppe ermittelt wird.
SlotNum	Nummer des Einsteckplatzes auf dem die Baugruppe steckt. Die Nummer muß innerhalb des Bereichs 1 ... MAX_SLOT_400 sein. Die Konstante MAX_SLOT_400 kennzeichnet die maximale Anzahl von Steckplätzen im System S7–400.

Beschreibung

Der Aufruf liefert zu einem Ziehen/Stecken Alarm den Peripherietyp der Baugruppe auf dem Steckplatz mit Nummer *SlotNum*.

Der Aufruf ist als C-Makro realisiert.

Die Funktion wird nur im System SIMATIC S7–400 unterstützt.

Rückgabewert

Als Rückgabewert wird vom Aufruf der Peripherietyp zurückgegeben. Die möglichen Werte sind in der folgenden Tabelle aufgelistet:

Peripherietyp	Bedeutung
M7IO_IN	Baugruppe ist Eingabebaugruppe
M7IO_OUT	Baugruppe ist Ausgabebaugruppe

Siehe auch

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmAddr, M7GetZSAlarmMode, M7GetZSAlarmIdent

M7InitAPI

Funktion

M7-API initialisieren

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7InitAPI(void);
```

Beschreibung

Die Funktion initialisiert das M7-API. Sie muß unmittelbar am Anfang eines C-Anwenderprogramms im Main-Teil aufgerufen werden.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NOT_IMPLEMENTED	M7 Server sind nicht gestartet

M7InitISADesc

Funktion

I/O Deskriptor aus logischer Adresse erzeugen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7InitISADesc(
                  M7IO_LOGADDR Addr,
                  UBYTE PType,
                  UWORD Len,
                  M7IO_DESC_PTR pIODesc);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse
PType	Peripherietyp M7IO_IN M7IO_OUT
Len	Länge des geplanten Zugriffes. Folgende Kennungen sind möglich: M7PBYTE: Deskriptor für ein Byte M7PWORD: Deskriptor für ein Wort M7PDWORD: Deskriptor für ein Doppelwort
pIODesc	Zeiger auf bereitgestellten I/O–Deskriptor. Der Speicherbereich für den I/O–Deskriptor muß vom Anwenderprogramm im globalen Datenbereich oder auf dem Heap bereitgestelltwerden.

Beschreibung

Die Funktion erzeugt aus der logischen Adresse einen I/O–Deskriptor. Der I/O–Deskriptor dient zum schnellen Zugriff auf ISA–Bus Peripherie.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Angegebene Adresse bezeichnet kein Interface–Modul, Länge oder Peripherietyp falsch

Siehe auch

M7StoreISAByte, M7StoreISAWord, M7StoreISADWord, M7LoadISAByte, M7LoadISAWord, M7LoadISADWord

M7KAbort

Funktion

Schließen einer Applikationsbeziehung

Syntax

#include <m7api.h>
M7ERR_CODE M7KAbort(M7CONNID *ConnID***);**

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.

Beschreibung

Die Funktion `M7KAbort` schließt eine Applikationsbeziehung zwischen Client und Server. Alle asynchronen Aufträge für die Verbindung werden gelöscht.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut

Siehe auch

M7KInitiate

M7KEvent

Funktion

Daten asynchroner Meldungen abholen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7KEvent(
                  M7CONNID ConnID,
                  UDWORD nRequest,
                  UBYTE_PTR pBuffer,
                  UDWORD nBufsiz,
                  UDWORD *pnBytes);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`– Aufruf.
nRequest	Auftragsnummer. Die Auftragsnummer kann aus dem in der Nachricht referenzierten FRB über den Aufruf `M7GetCommRequest` ausgelesen werden
pBuffer	Zeiger auf den Ergebnispuffer. Der Ergebnispuffer muß vom Anwenderprogramm bereitgestellt werden.
nBufsiz	Länge des Ergebnispuffers.
pnBytes	Anzahl gelesener Bytes.

Beschreibung

Die durch Zyklisches Lesen und Diagnosemeldungen anfallenden Daten müssen mit der Funktion `M7KEvent` vom Treiber abgeholt werden.

Die nächste Meldung mit der Auftragsnummer *nRequest* für die Verbindungsreferenz *ConnID* wird in den Ergebnispuffer kopiert und im Treiber gelöscht. Die Anzahl übertragener Bytes wird in **pnBytes* vermerkt.

Falls der Ergebnispuffer zuklein ist, um die gesamten Daten einer Message aufzunehmen, werden soviele Daten wie möglich kopiert und ein entsprechender Errorcode gesetzt. Falls keine passende Meldung vorliegt, kehrt der Aufruf ohne Fehler mit **pnBytes* gleich 0 zurück.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut

Fehlercode	Bedeutung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch **M7BUBCycRead, M7DiagMode**

M7KInitiate

Funktion

Applikationsbeziehung für Kommunikation über K–Bus/MPI einrichten

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7KInitiate(
                  M7CONNID *pConnID,
                  UBYTE_PTR pHostAddr);
```

Parameter

Parametername	Bedeutung
**pConnID*	Zeiger auf die Verbindungsreferenz für weitere M7–Kommunikationsaufrufe.
pHostAddr	Adresse des Zielrechners.

Beschreibung

Die Funktion `M7KInitiate` eröffnet eine Applikationsbeziehung zu einem Server via MPI oder K–Bus. Die Hostadresse des remoten Partners wird als String angegeben. *pHostAddr* enthält die Verbindungsnummer aus der Verbindungsprojektierung. Die Verbindungsnummer kann sowohl dezimal als auch hexadezimal eingegeben werden. Beispiel: 0x1d0. Es ist egal, ob die Buchstaben groß oder klein geschrieben sind. Durch die Übergabe des Strings ”local” wird eine einseitige Applikationsbeziehung zur eigenen CPU/FM eingerichtet.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_RESOURCE_LIMIT	Ressourcen überschritten
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server

Siehe auch

M7KAbort

M7KPassword

Funktion

Für Funktionen mit besonderer Schutzstufe anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7KPassword(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UBYTE_PTR pszPassword);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	SET_PASSWORD	Falls SET_PASSWORD gesetzt ist und das richtige Paßwort eingegeben wird, wird die Verbindung legitimiert; d.h. es sind danach alle Funktionen verfügbar.
	A_ZERO_FLAG	Falls A_ZERO_FLAG gesetzt ist, wird die Verbindung freigegeben; d.h. es sind danach nur Funktionen entsprechend der Schutzstufe verfügbar. Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.	
pszPassword	Zeiger auf ein 8–Byte Paßwort.	

Beschreibung

Die M7/S7–CPU hat im SDB0 ein Paßwort und eine Schutzstufe eingetragen. Nach einem `M7KInitiate`–Aufruf kann die Applikation nur Funktionen der aktuellen Schutzstufe ausführen. Um alle Funktionen ausführen zu können, muß sich die Applikation mit dem richtigen Paßwort legitimieren.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut

Fehlercode	Bedeutung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch **M7KInitiate**

M7KReadTime

Funktion

Uhrzeit lesen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7KReadTime(
                  M7CONNID ConnID,
                  M7KTIME_PTR pBuffer,
                  UDWORD nBufsize,
                  UDWORD *pnBytes);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`– Aufruf.
pBuffer	Zeiger auf eine Datenstruktur vom Typ **M7KTIME**. Die Datenstruktur zur Aufnahme der K–BUS–Uhrzeit muß vom Anwenderprogramm in den globalen Daten oder auf dem Heap bereitgestelltwerden.
nBufsize	Länge der **M7KTIME**–Struktur.
pnBytes	Zeiger auf die Anzahl gelesener Bytes.

Beschreibung

Die Funktion `M7KReadTime` liest die Uhrzeit des Serverrechners in die bereitgestellte Datenstruktur. In **pnBytes* wird die Anzahl gelesener Bytes eingetragen.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7KWriteTime

M7KWriteTime

Funktion **Uhrzeit stellen**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7KWriteTime(
                  M7CONNID ConnID,
                  M7KTIME_PTR pBuffer,
                  UDWORD nBufsize);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`– Aufruf.
pBuffer	Zeiger auf Datenstruktur vom Typ **M7KTIME** mit der zu setzenden Uhrzeit.
nBufsize	Länge der **M7KTIME**–Struktur.

Beschreibung Die Funktion `M7KWriteTime` stellt die Uhrzeit des Zielrechners auf den in *pBuffer* angegebenen Wert.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch **M7KReadTime, M7KInitiate**

M7LinkBatteryFailure

Funktion

FRB für Batterieüberwachung initialisieren und beim BZÜ–Server anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkBatteryFailure(
                  M7BAFFRB_PTR pBAFFRB,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pBAFFRB	Zeiger auf den FRB der für die Bearbeitung der Anmeldung vorgesehen ist. Der FRB muß vom Anwenderprogramm in den globalen Daten oder auf dem Heap bereitgestellt werden.
MPrio	Priorität der zu versendenden M7MSG_BATTERY_FAILURE Message (0–255).

Beschreibung

Die Funktion `M7LinkBatteryFailure` initialisiert einen FRB und meldet den FRB beim BZÜ–Server zur Bearbeitung an.

Sinkt vor oder während der Bearbeitung des FRBs die Batteriespannung unter den Schwellwert, so erhält die Task eine Nachricht vom Typ M7MSG_BATTERY_FAILURE mit der Message Priorität *MPrio*.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PRIO	Falsche Priorität

Siehe auch

M7UnLinkBatteryFailure

M7LinkCycle

Funktion

FRB initialisieren und beim FZ–Server anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkCycle(
                  M7FSCFRB_PTR pFSCFRB,
                  UWORD Cycle,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pFSCFRB	Zeiger auf den FRB der für die Benachrichtigung beim FZ–Server angemeldet wird.
Cycle	Angabe des Zustands, bei dem die Benachrichtigung erfolgen soll. M7S_CYCLECONTROLPOINT Benachrichtigung am Systemkontrollpunkt M7S_FREECYCLE Benachrchtigung beim Start des Freien Zyklusses M7S_STARTUPCYCLE Benachrichtigung für Zustand: ANLAUF M7S_CYCLEOVERFLOW Benachrichtigung bei Zykluszeitüberschreitung
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Die Funktion `M7LinkCycle` initialisiert einen FRB und meldet den FRB beim FZ–Server zur Bearbeitung an. Wenn der mit *Cycle* gewünschte Zustand im eingenommen ist, erhält die Task eine Nachricht vom Typ M7MSG_M_CYCLE.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Unbekannter Zustand
M7E_PRIO	Falsche Priorität

Siehe auch

M7UnLinkCycle, M7ConfirmCycle

M7LinkDataAccess

Funktion

S7-Objekt für Zugriffsinformation über Nachricht anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkDataAccess(
                  M7OBJFRB_PTR pOBJFRB,
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD Flags,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pOBJFRB	Zeiger auf den FRB, der für die Anmeldung vorgesehen ist
ObjType	Typkennzeichen des S7-Objektes, für das die Zugriffe gemeldet werden sollen (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw. vgl. Tabelle 2-8)
Flags	Maske für Auswahl, welcher Zugriff gemeldet werden soll: M7READ_ACCESS: nur lesende M7WRITE_ACCESS: nur schreibende M7CREATE_ACCESS: Meldung bei Objekt–Erzeugung M7DELETE_ACCESS: Meldung beim Objekt–Löschen M7LINK_ACCESS: Meldung beim Objekt–Einketten
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Mit der Funktion gibt eine Task dem Objektserver den Auftrag, Zugriffe auf das referenzierte S7-Objekt über eine Nachricht an die Task zu melden.

Über *Flags* kann die aufrufende Task bestimmen, **welche** Zugriffart gemeldet werden soll z. B. schreibender Zugriff. Die *Flags* dürfen nicht verodert werden, es ist nur eine Zugriffsart erlaubt.

Wird nach erfolgreich durchgeführter Funktion von einer anderen Task oder über Kommunikation von außen auf das angemeldete S7-Objekt zugegriffen, so sendet der Objektserver – in Abhängigkeit von der angegebenen Zugriffsart – nach Durchführen des Zugriffs eine der in der nachfolgenden Tabelle aufgeführten Nachrichten.

Zugriff	Meldung
Lese-Zugriff	M7MSG_DATA_ACCESS_R
Schreib-Zugriff	M7MSG_DATA_ACCESS_W

Zugriff	Meldung
S7-Objekt wird gelöscht	M7MSG_DATA_ACCESS_DEL
S7-Objekt wird erzeugt	M7MSG_DATA_ACCESS_CREATE

Rückgabewert

= M7SUCCESS Wird vom Aufruf immer zurückgegeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_ALREADY_IN_LIST	FRB ist bereits eingekettet
M7E_LINK_PAR	Parameterfehler
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_PAR	Parameterfehler
M7E_PRIO	Falsche Priorität

Siehe auch

M7SetFRBTag, M7GetFRBTag, M7GetObjType, M7Getflags, M7GetPart

M7LinkDataAccessCB

Funktion

Callback–Funktion bei S7–Zugriff anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkDataAccessCB(
                  M7CBFRB_PTR pCBFRB,
                  UDWORD (*pCallback)(M7CBFRB_PTR pCBFRB),
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD Flags);
```

Parameter

Parametername	Bedeutung
pOBJFRB	Zeiger auf den FRB, der für die Anmeldung vorgesehen ist
**pCallback*	Zeiger auf die Callback–Funktion
ObjType	Typkennzeichen des S7-Objektes, für das die Zugriffe gemeldet werden sollen (vgl. Tabelle 2-7).
Part	Teilbereichsnummer (DB-Nummer usw., vgl. Tabelle 2-8)
Flags	Maske für Auswahl, bei welchen Zugriffsarten die Callback–Funktion aufgerufen werden soll: M7READ_ACCESS: lesender Zugriff M7WRITE_ACCESS: schreibender Zugriff M7CREATE_ACCESS: Meldung bei Objekt–Erzeugung M7DELETE_ACCESS: Meldung beim Objekt-Löschen M7LINK_ACCESS: Meldung beim Objekt–Einketten

Beschreibung

Mit der Funktion gibt eine Task dem Objektserver den Auftrag vor einem WRITE–, CREATE–, oder LINK–Zugriff oder nach einem READ–Zugriff auf das angegebene S7–Objekt die Callback–Funktion aufzurufen.

Über Flags kann die aufrufende Task bestimmen, bei **welcher** Zugriffart die Callback–Funktion aufgerufen werden soll, z. B. nur schreibende Zugriffe.

Rückgabewert

= M7SUCCESS Wird vom Aufruf immer zurückgegeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_ALREADY_IN_LIST	FRB ist bereits eingekettet
M7E_LINK_PAR	Parameterfehler

Fehlercode	Bedeutung
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_PAR	Parameterfehler

Siehe auch **M7GetCBBitOffset, M7GetCBBuffer, M7GetCBByteOffset, M7GetCBDataType, M7GetCBCount, M7GetCBFlags, M7GetCBObjType, M7GetCBPart, M7UnlinkDataAccessCB**

M7LinkDate

Funktion

Uhrzeitgesteuerte Zeitnachricht anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkDate(
                  M7TFRB_PTR pTFRB,
                  M7TIME_DATE_PTR pDateTime,
                  BOOL Periodic,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den zugehörigen Time-Server-FRB
pDateTime	Zeiger auf Speicherbereich mit Datum-Zeit-Struktur , in dem die Zeitparameter für die Funktion abgelegt sind (vgl. Kapitel 3).
Periodic	Auswahl für ”einmalig” oder ”täglich”: M7ONCE einmalige Nachricht M7DAILY tägliche Nachricht (Datum = Startdatum)
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Mit der Funktion wird ein FRB für eine uhrzeitgesteuerte Bearbeitung beim Time-Server angemeldet. Wenn die in **pDateTime* angegebene Uhrzeit bzw. das angegebene Datum erreicht sind, sendet der Time-Server eine Nachricht vom Typ M7MSG_TIMESERVER an die aufrufende Task.

Die Nachricht wird im Betriebszustand RUN sekundengenau gesendet (Auflösung = 1Sekunde). Wenn die angegebene Uhrzeit erreicht wird und das System nicht im Betriebszustand RUN läuft, wird die Nachricht bis zum nächsten Zustandsübergang in den Betriebszustand RUN verzögert. Ist eine Task gleichzeitig für Betriebszustand–Nachrichten angemeldet, so ist beim Übergang in den Zustand RUN die Reihenfolge des Empfangs der uhrzeitgesteuerten Nachrichten und der Betriebszustand–Nachrichten unbestimmt.

Bei nicht–periodischem Betrieb löscht der Time–Server den zugehörigen FRB nach dem Senden der uhrzeitgesteuerten Nachricht.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_PRIO	Falsche Priorität
M7E_RESOURCE_LIMIT	Zu viele Timer FRBs in Bearbeitung

Siehe auch

M7UnLinkDate

M7LinkDiagAlarm

Funktion

Diagnosealarm zur Bearbeitung anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkDiagAlarm(
                  M7DIAGALARM_FRB_PTR pDiagFrb,
                  UBYTE PType,
                  M7IO_BASEADDR Addr,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pDiagFrb	Zeiger auf den FRB, der für die Anmeldung vorgesehen ist. Der FRB muß vom Anwenderprogramm in den globalen Daten oder auf dem Heap bereitgestellt werden.
PType	Kennzeichen für Ein- oder Ausgabebaugruppe: M7IO_IN Eingabebaugruppe M7IO_OUT Ausgabebaugruppe
Addr	Logische Basisadresse der Baugruppe, die Diagnosealarme sendet.
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Die Funktion initialisiert einen FRB-Header und meldet den FRB beim Alarmserver zur Bearbeitung an.

Wenn die mit *Addr* angegebene Peripheriebaugruppe einen **Diagnosealarm** meldet, erhält die aufrufende Task eine Nachricht vom Typ M7MSG_DIAG_ALARM.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Angesprochene Baugruppe nicht vorhanden.
M7E_INVAL_DEV	Diagnosealarm kann bei DP–Normslaves nur von der ET–ER gemeldet werden.

Siehe auch

M7UnLinkDiagAlarm, M7GetDiagAlarmAddr, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo, M7GetDiagAlarmPType

M7LinkIOAlarm

Funktion

Prozeßalarm zur Bearbeitung anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkIOAlarm(
                  M7IOALARM_FRB_PTR pIOFrb,
                  UBYTE PType,
                  M7IO_BASEADDR Addr,
                  UDWORD AlarmMask,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pIOFrb	Zeiger auf den FRB, der für die Anmeldung vorgesehen ist
PType	Kennzeichen für Ein- oder Ausgabebaugruppe: M7IO_IN Eingabebaugruppe M7IO_OUT Ausgabebaugruppe
Addr	Logische Basisadresse der Baugruppe, deren Prozeßalarme gemeldet werden sollen.
AlarmMask	Alarm-Maske: Mit dem Parameter AlarmMask können 32 Kanäle selektiert werden. Dem Bit 2^0 ist Kanal 0, dem Bit 2^1 der Kanal 1 zugeordnet usw. Masken-Bit = 1 bedeutet, daß der Kanal **nicht** bearbeitet wird; Masken-Bit = 0 bedeutet, daß der Kanal bearbeitet wird.
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Die Funktion initialisiert einen FRB-Header und meldet den FRB beim Alarmserver zur Bearbeitung an.

Wenn die mit *Addr* angegebene Peripheriebaugruppe einen **Prozeßalarm** meldet, erhält die aufrufende Task eine Nachricht vom Typ M7MSG_IO_ALARM.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Angesprochene Baugruppe nicht vorhanden.
M7E_SLAVE_TYPE	Prozeßalarme können nur von DP–S7–Slave Baugruppen gemeldet werden.
M7E_INVAL_DEV	Prozeßalarme können nur von Peripheriebaugruppen und nicht von der ET–ER generiert werden.

Siehe auch

M7UnLinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmBusy, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPType

M7LinkOneShotTimer

Funktion

Singuläre Zeitnachricht anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkOneShotTimer(
                  M7TFRB_PTR pTFRB,
                  UWORD TimeBase,
                  UDWORD Time,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den zugehörigen Time-Server-FRB
TimeBase	Wert für die Zeitbasis: M7TB_1MS: 1 ms M7TB_10MS: 10 ms M7TB_100MS: 100 ms M7TB_1S: 1 s
Time	Zeit (Vielfaches von *TimeBase*, max. 4 198 404)
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Mit der Funktion wird ein FRB für die Bearbeitung einer singulären Zeitnachricht beim Time-Server angemeldet. Wenn die angegebene Zeit abgelaufen ist, sendet der Time-Server eine Nachricht an die aufrufende Task und löscht den zugehörigen FRB. Die Zeitnachrichten werden nur im Betriebszustand RUN gesendet.

Hinweis

Wählen Sie die Parameter *TimeBase* und *Time* so aus, daß für das gewünschte Zeitintervall der Parameter *TimeBase* den größtmöglichen Wert erhält. Dadurch minimieren Sie die Systembelastung durch den Time-Server.

Beispiel:
Ihre Task soll nach einer Zeit von 4 s eine einmalige Zeitnachricht vom Time-Server erhalten. Wählen Sie in diesem Falle für *TimeBase* den Wert ’M7TB_1S’ und für *Time* den Wert ’4’ (nicht: ’M7TB_100MS’ für *TimeBase* und ’40’ für *Time*!).

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	falscher Wert für *TimeBase*
M7E_PRIO	Falsche Priorität
M7E_RESOURCE_LIMIT	Zu viele Timer FRBs in Bearbeitung

Siehe auch

M7UnLinkOneShotTimer

M7LinkPeriodicTimer

Funktion

Periodische Zeitnachricht anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkPeriodicTimer(
                  M7TFRB_PTR pTFRB,
                  UWORD TimeBase,
                  UDWORD Period,
                  BOOL Handshake,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den zugehörigen Time-Server-FRB
TimeBase	Wert für die Zeitbasis: M7TB_1MS: 1 ms M7TB_10MS: 10 ms M7TB_100MS: 100 ms M7TB_1S: 1s
Period	Dauer der Perioden (Vielfaches von *TimeBase*, max. 4 198 404)
Handshake	Auswahl der Betriebsart: M7WITH_HANDSHAKE quittunggesteuerter Betrieb M7NO_HANDSHAKE kein quittunggesteuerter Betrieb
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Mit der Funktion wird ein FRB für die Bearbeitung einer periodischen Zeitnachricht beim Time-Server angemeldet.

Nachdem der FRB angemeldet ist, sendet der Time-Server periodische Zeitnachrichten an die aufrufende Task. Die Zeitnachrichten werden nur im Betriebszustand RUN gesendet.

Bei quittungsgesteuertem Betrieb (*Handshake* = M7WITH_HANDSHAKE) muß jede periodische Zeitnachricht von der empfangenen Task mit der Funktion `M7ConfirmPeriodicTimer` quittiert werden.

Pro M7–CPU/FM können maximal 10 FRBs angemeldet werden.

Hinweis

Wählen Sie die Parameter *TimeBase* und *Period* so aus, daß für das gewünschte Zeitintervall der Parameter *TimeBase* den größtmöglichen Wert erhält. Dadurch minimieren Sie die Systembelastung durch den Time-Server.

Beispiel:
Ihre Task soll nach einer Zeit von 4 s eine einmalige Zeitnachricht vom Time-Server erhalten. Wählen Sie in diesem Falle für *TimeBase* den Wert 'M7TB_1S' und M7für *Time* den Wert '4' (nicht: 'M7TB_100MS' für *TimeBase* und '40' für *Time*!).

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	falscher Wert für *TimeBase*
M7E_PRIO	Falsche Priorität
M7E_RESOURCE_LIMIT	Zu viele Timer FRBs in Bearbeitung

Siehe auch **M7UnLinkPeriodicTimer, M7ConfirmPeriodicTimer, M7GetLostPeriods**

M7LinkPIError

Funktion

FRB für Prozeßabbildtransferfehler initialisieren

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkPIError(
                  M7FRBHEADER_PTR pPIEFRB
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pPIEFRB	Zeiger auf den FRB, der für die Anmeldung vorgesehen ist
MPrio	Priorität der M7MSG_PI_ERROR Nachricht (0–255)

Beschreibung

Die Funktion `M7LinkPIError` initialisiert einen FRB zur Bearbeitung von Prozeßabbildtransferfehler, die im Freien Zyklus aufgetreten sind.

Erkennt der Freie Zyklus Server einen PA–Transferfehler, verschickt er an jede angemeldete Task die Nachricht M7MSG_PI_ERROR. Die Nachricht enthält den Prozeßabbildtyp und die Prozeßabbildadresse, bei der der Transferfehler aufgetreten ist.

Mit dem Parameter *MPrio* kann die Priorität der M7MSG_PI_ERROR Nachricht vorgegeben werden.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PRIO	Falsche Priorität

Siehe auch

M7UnLinkPIError

M7LinkState

Funktion

Nachricht bei bestimmten Betriebszustand anfordern

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkState(
                M7TSFRB_PTR pTSFRB,
                UWORD State,
                unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger auf den FRB, der für die Bearbeitung der Anmeldung vorgesehen ist. Der FRB muß vom Anwenderprogramm in den globalen Daten oder auf dem Heap bereitgestellt werden.
State	Angabe des Betriebszustandes, bei dem die Benachrichtigung erfolgen soll. Eine Task kann sich mit einem FRB nur für einen BZ anmelden. Es können folgende Werte angegeben werden: M7STATE_STOP Betriebszustand STOP erreicht M7STATE_STARTUP Betriebszustand ANLAUF erreicht M7STATE_RUN Betriebszustand RUN erreicht M7STATE_HALT Betriebszustand HALT erreicht M7STATE_RESET Betriebszustand URLÖSCHEN erreicht
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Die Funktion initialisiert einen FRB-Header und meldet den FRB beim BZÜ-Server zur Bearbeitung an.

Wenn der mit dem Parameter *State* angegebene Betriebszustand eingetreten ist, wird die aufrufende Task mit einer Nachricht vom Typ *M7MSG_STATE* darüber informiert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_PRIO	Falsche Priorität

Siehe auch

M7UnLinkState, M7GetState, M7RequestState, M7GetTSType, M7GetTSReason

M7LinkTransition

Funktion

Nachricht bei bestimmten Betriebszustandsübergang anfordern

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkTransition(
                M7TSFRB_PTR pTSFRB,
                UWORD Transition,
                unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger auf den FRB, der für die Bearbeitung der Anmeldung vorgesehen ist. Der FRB muß vom Anwenderprogramm in den globalen Daten oder auf dem Heap bereitgestellt werden.
Transition	Angabe des Betriebszustandsübergang, bei dem die Benachrichtigung erfolgen soll. Eine Tasks kann sich mit einem FRB nur für einen BZ–Übergang anmelden Es können folgende Werte angegeben werden: M7TRANS_STOPSTARTUP STOP nach ANLAUF M7TRANS_STOPRESET STOP nach URLÖSCHEN M7TRANS_STARTUPSTOP ANLAUF nach STOP M7TRANS_STARTUPRUN ANLAUF nach RUN M7TRANS_STARTUPHALT ANLAUF nach HALT M7TRANS_RUNSTOP RUN nach STOP M7TRANS_RUNHALT RUN nach HALT M7TRANS_HALTSTOP HALT nach STOP M7TRANS_HALTSTARTUP HALT nach ANLAUF M7TRANS_HALTRUN HALT nach RUN M7TRANS_RESETSTOP URLÖSCHEN nach STOP
MPrio	Priorität, mit der eine Nachricht gesendet werden soll (0–255).

Beschreibung

Die Funktion initialisiert einen FRB-Header und meldet den FRB beim BZÜ-Server zur Bearbeitung an.

Bevor der mit dem Parameter *Transition* angegebene Betriebszustandsübergang erfolgt, wird die aufrufende Task mit der Meldung vom Typ M7MSG_TRANSITION darüber informiert. Die Task muß diesen Betriebszustandsübergang quittieren.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_PRIO	Falsche Priorität

Siehe auch

M7UnLinkTransition, M7GetTSReason, M7GetTSType

M7LinkZSAlarm

Funktion

FRB für Ziehen/Stecken–Ereignis anmelden

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkZSAlarm(
                  M7ZSALARM_FRB_PTR pZSFRB,
                  M7IO_BASEADDR IMRBaddr,
                  unsigned int Prio);
```

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den FRB, der für die Anmeldung vorgesehen ist
IMRBaddr	Logische Basisadresse einer IMR–Baugruppe oder *M7CR_BADDR* für den Zentralbaugruppenträger
MPrio	Priorität der M7MSG_ZS_ALARM Nachricht (0–255).

Beschreibung

Die Funktion initialisiert einen FRB zur Ziehen/Stecken–Alarmbearbeitung und meldet den FRB beim Alarmserver an.

Wenn ein Ziehen/Stecken–Ereignis in dem Baugruppenträger bzw. in dem S7–Slave, in dem die IM–Baugruppe mit der Basisadresse *IMRBaddr* steckt, eingetreten ist, erhält die Task die Nachricht M7MSG_ZS_ALARM.

Für den Zentralbaugruppenträger muß die Anmeldung für die Basisadresse *M7CR_BADDR* erfolgen.

Mit *MPrio* kann die Priorität der Nachricht vorgegeben werden.

Im Nachrichtenpuffer wird dem Anwender die Adresse des Ziehen/Stecken FRBs mit der Ziehen/Stecken–Information übergeben. **Dieser FRB ist nicht der FRB, mit dem sich der Anwender angemeldet hat, sondern ein im System allokierter FRB**.

Nach der Alarmauswertung muß der Anwender den Ziehen/Stecken–Alarm mit `M7ConfirmZSAlarm` quittieren, damit diese Systemressource wieder verfügbar gemacht werden kann.

Die Funktion wird nur im System SIMATIC S7–400 unterstützt.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Ungültiger Wert für *IMRBADDR*
M7E_PRIO	Falsche Priorität
M7E_NOT_IMPLEMENTED	Funktion wird in S7–300 nicht unterstützt

Siehe auch **M7ConfirmZSAlarm, M7GetZSAlarmAddr, M7GetZSAlarmIdent, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7UnLinkZSAlarm**

M7LoadBit

Funktion

Bit aus Prozeßabbild laden

Syntax

```
#include <m7api.h>
BOOL        M7LoadBit(
              UWORD PIType,
              UWORD ByteOffset,
              UBYTE BitOffset,
              M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalbytes
BitOffset	Bit-Offset innerhalb des Signalbytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion adressiert ein Bit im Prozeßabbild, das mit *PIType* bestimmt wird, und liefert als Rückgabewert den Zustand des Bits.

Rückgabewert

Als Rückgabewert wird von der Funktion der Zustand des adressierten Bits übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType*, *ByteOffset* oder *BitOffset*

Siehe auch

M7LoadByte, M7LoadDWord, M7LoadWord

M7LoadByte

Funktion

Byte aus Prozeßabbild laden

Syntax

```
#include <m7api.h>
UBYTE          M7LoadByte(
                 UWORD PIType,
                 UWORD ByteOffset,
                 M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalbytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion adressiert ein Byte im Prozeßabbild, das mit *PIType* bestimmt wird, und liefert als Rückgabewert den Inhalt des adressierten Bytes.

Rückgabewert

Als Rückgabewert wird von der Funktion der Inhalt des adressierten Bytes übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType* oder *ByteOffset*

Siehe auch

M7LoadBit, M7LoadDWord, M7LoadWord

M7LoadDirect

Funktion

Peripheriebereich direkt lesen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LoadDirect(
                  VOID_PTR pBuffer,
                  UWORD SizeOfItem,
                  UWORD Count,
                  M7IO_LOGADDR Addr);
```

Parameter

Parametername	Bedeutung
pBuffer	Zeiger auf den Zielpuffer
SizeOfItem	Größe eines Elementes in Byte. Folgende Konstanten sind vordefiniert: M7PBYTE Zugriff auf Elemente des Typs BYTES M7PWORD Zugriff auf Elemente des Typs WORD M7PDWORD Zugriff auf Elemente des Typs DWORD
Count	Anzahl der Elemente
Addr	logische Adresse des ersten Elementes

Beschreibung

Die Funktion führt einen direkten Zugriff auf die Prozeßperipherie durch. Quelle, Größe, Anzahl und Ziel der gelesenen Daten werden durch die Aufruf-Parameter bestimmt.

Die Funktion führt keine Konvertierung der Zahlendarstellung (*SIMATIC/Intel*) durch.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit für den Datenaustausch

Siehe auch **M7LoadDirectByte, M7LoadDirectDWord, M7LoadDirectWord**

M7LoadDirectByte

Funktion

Byte direkt aus Peripherie lesen

Syntax

```
#include <m7api.h>
UBYTE          M7LoadDirectByte(
                 M7IO_LOGADDR Addr,
                 M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse des Peripheriebytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion führt einen direkten Zugriff auf die Prozeßperipherie durch und liest ein Byte.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert das gelesene Byte aus der Prozeßperipherie.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit für den Datenaustausch

Siehe auch

M7LoadDirect, M7LoadDirectDWord, M7LoadDirectWord

M7LoadDirectDWord

Funktion

Doppelwort direkt aus Peripherie lesen

Syntax

```
#include <m7api.h>
UDWORD      M7LoadDirectDWord(
              M7IO_LOGADDR Addr,
              M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse des Peripheriedoppelwortes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion führt einen direkten Zugriff auf die Prozeßperipherie durch und liest ein Doppelwort.

Der Inhalt des Doppelwortes wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert das gelesene Doppelwort aus der Prozeßperipherie in *Intel*-Zahlendarstellung.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit für den Datenaustausch

Siehe auch

M7LoadDirect, M7LoadDirectByte, M7LoadDirectWord

M7LoadDirectWord

Funktion

Wort direkt aus Peripherie lesen

Syntax

```
#include <m7api.h>
UWORD        M7LoadDirectWord(
               M7IO_LOGADDR Addr,
               M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse des Peripheriewortes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion führt einen direkten Zugriff auf die Prozeßperipherie durch und liest ein Wort.

Der Inhalt des Wortes wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert das gelesene Wort aus der Prozeßperipherie in *Intel*-Zahlendarstellung.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit für den Datenaustausch

Siehe auch

M7LoadDirect, M7LoadDirectByte, M7LoadDirectDWord

M7LoadDWord

Funktion

Doppelwort aus Prozeßabbild laden

Syntax

```
#include <m7api.h>
UDWORD      M7LoadDWord(
              UWORD PIType,
              UWORD ByteOffset,
              M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalbytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion adressiert ein Doppelwort im Prozeßabbild, das mit *PIType* bestimmt wird, und liefert als Rückgabewert den Inhalt des adressierten Doppelwortes.

Der Inhalt des Doppelwortes wird zuvor aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Als Rückgabewert wird von der Funktion der Inhalt des adressierten Doppelwortes in *Intel*-Zahlendarstellung übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType* oder *ByteOffset*

Siehe auch

M7LoadBit, M7LoadByte, M7LoadWord

M7LoadISAByte

Funktion

Byte direkt aus ISA–Bus–Peripherie lesen

Syntax

```
#include <m7api.h>
UBYTE          M7LoadISAByte(
                 M7IO_DESC_PTR pIODesc,
                 M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
pIODesc	Zeiger auf I/O–Deskriptor, der mit `M7InitISADesc` eingerichtet wurde
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion führt als Makro einen direkten Zugriff auf die ISA–Bus–Prozeßperipherie mit Hilfe eines mit `M7InitISADesc` generierten I/O–Deskriptors durch und liest ein Byte ein.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert das gelesene Byte aus der ISA–Bus–Prozeßperipherie.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Datenzugriff auf ISA–Bus ist größer (in Bytes) als in `M7InitISADesc` spezifiziert.

Siehe auch

M7LoadISAWord, M7LoadISADWord, M7InitISADesc

M7LoadISADWord

Funktion

Doppelwort direkt aus ISA–Bus–Peripherie lesen

Syntax

```
#include <m7api.h>
UDWORD      M7LoadISADWord(
              M7IO_DESC_PTR pIODesc,
              M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
pIODesc	Zeiger auf I/O–Deskriptor, der mit `M7InitISADesc` eingerichtet wurde
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion führt als Makro einen direkten Zugriff auf die ISA–Bus–Prozeßperipherie mit Hilfe eines mit `M7InitISADesc` generierten I/O–Deskriptors durch und liest ein Doppelwort (32 Bit) in Inteldarstellung ein.

Der Inhalt des Doppelworts wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert das gelesene Doppelwort (32 Bit) in Intel–Darstellung aus der ISA–Bus–Prozeßperipherie.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Datenzugriff auf ISA–Bus ist größer (in Bytes) als in `M7InitISADesc` spezifiziert.

Siehe auch

M7LoadISAByte, M7LoadISAWord, M7InitISADesc

M7LoadISAWord

Funktion

Wort direkt aus ISA–Bus–Peripherie lesen

Syntax

```
#include <m7api.h>
UWORD        M7LoadISAWord(
               M7IO_DESC_PTR pIODesc,
               M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
pIODesc	Zeiger auf I/O–Deskriptor, der mit `M7InitISADesc` eingerichtet wurde
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion führt als Makro einen direkten Zugriff auf die ISA–Bus–Prozeßperipherie mit Hilfe eines mit `M7InitISADesc` generierten I/O–Deskriptors durch und liest ein Wort (16 Bit) in Inteldarstellung ein.

Der Inhalt des Wortes wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert das gelesene Wort (16 Bit) aus der ISA–Bus–Prozeßperipherie.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Datenzugriff auf ISA–Bus ist größer (in Bytes) als in `M7InitISADesc` spezifiziert.

Siehe auch

M7LoadISAByte, M7LoadISADWord, M7InitISADesc

M7LoadPII

Funktion

Aktualisieren des Prozeßabbildes der Eingänge

Syntax

#include <m7api.h>
M7ERR_CODE M7LoadPII(UWORD *PIINo***);**

Parameter

Parametername	Bedeutung
PIINo	Nummer des Teilprozeßabbilds bei M7–400. M7–400: 0 gesamtes Prozeßabbild 1 ... 8 Teilprozeßabbild M7–300: 0 gesamtes Prozeßabbild Teilprozeßabbilder werden nicht unterstützt

Beschreibung

Die Funktion aktualisiert das gesamte Prozeßabbild bzw. das angegebene Teilprozeßabbild der Eingänge.

Teilprozeßabbilder werden nur im System S7–400 unterstützt.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Falsche *PIINo*
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug

Siehe auch

M7StorePIQ, M7ClearPI

M7LoadRecord

Funktion

Datensatz aus Signalbaugruppe lesen

Syntax

```
#include      <m7api.h>
M7ERR_CODE    M7LoadRecord(
                UBYTE RecordNum,
                VOID_PTR pBuffer,
                UBYTE Size,
                UBYTE PType,
                M7IO_BASEADDR Addr);
```

Parameter

Parametername	Bedeutung
RecordNum	Datensatznummer Bereich: 0 bis 255
pBuffer	Zeiger auf einen Puffer im Arbeitsspeicher, in den der Datensatz übertragen werden soll.
Size	Länge des Datensatzes
PType	Kennzeichen des Peripheriebereichs: M7IO_IN Peripheriebereich der Eingänge M7IO_OUT Peripheriebereich der Ausgänge Handelt es sich um eine Mischbaugruppe, ist die Bereichskennung der niedrigsten Adresse anzugeben. Bei gleichen Adressen ist M7IO_IN anzugeben.
Addr	I/O–Basisadresse der Baugruppe

Beschreibung

Die Funktion überträgt einen Datensatz aus einer Peripheriebaugruppe in einen Puffer, der mit dem Aufrufparameter *pBuffer* referenziert wird.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_COM_ERROR	Fehler bei Abarbeitung des Transferprotokolls
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug

Fehlercode	Bedeutung
M7E_REC_LENGTH	Baugruppe meldet falsche Datensatzlänge
M7E_REC_NUMBER	Baugruppe meldet falsche Datensatznummer
M7E_DPX2_FAULT	Fehler beim DP–Auftrag für den Datensatz–Transfer
M7E_DP_SLAVE_STATE	DP–Slave ist nicht in DATA Zustand
M7E_INVAL_DEV	Baugruppe eines DP–Slaves ist nicht vorhanden

Siehe auch **M7StoreRecord**

M7LoadRecordEx

Funktion

Datensatz aus Signalbaugruppe lesen

Syntax

```
#include <m7api.h>
long        M7LoadRecordEx(
              UBYTE RecordNum
              VOID_PTR pBuffer
              UBYTE Size
              UBYTE PType
              M7IO_BASEADDR Addr);
```

Parameter

Parametername	Bedeutung
RecordNum	Datensatznummer Bereich: 0 bis 255
pBuffer	Zeiger auf einen Puffer im Arbeitsspeicher, in den der Datensatz übertragen werden soll.
Size	Länge des Datenpuffers
PType	Kennzeichen des Peripheriebereichs: M7IO_IN Peripheriebereich der Eingänge M7IO_OUT Peripheriebereich der Ausgänge Handelt es sich um eine Mischbaugruppe, ist die Bereichskennung der niedrigsten Adresse anzugeben. Bei gleichen Adressen ist M7IO_IN anzugeben.
Addr	I/O–Basisadresse der Baugruppe

Beschreibung

Die Funktion überträgt einen Datensatz aus einer Peripheriebaugruppe in einen Puffer, der mit dem Aufrufparameter *pBuffer* referenziert wird.

Im Gegensatz zur Funktion `M7LoadRecord` erlaubt `M7LoadRecordEx` einen Datensatzzugriff ohne genaue Angabe der zu lesenden Bytes. Wird im Parameter *Size* mit 240 die maximale Datensatzlänge angegeben, werden die gültigen Bytes des Datensatzes *RecordNum* gelesen und nach *pBuffer* übertragen.

Der Rückgabewert enthält die Anzahl der im Datenpuffer gültigen Bytes (siehe unten).

Rückgabewert

> M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt. Der Rückgabewert enthält die Anzahl der im Datenpuffer gültigen Bytes, d.h. Datensatzlänge, wenn der Datenpuffer ≥ Datensatz Pufferlänge, wenn Datenpuffer < Datensatz.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_COM_ERROR	Fehler bei Abarbeitung des Transferprotokolls
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Angesprochene Baugruppe nicht vorhanden
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_REC_LENGTH	Baugruppe meldet falsche Datensatzlänge
M7E_REC_NUMBER	Baugruppe meldet falsche Datensatznummer
M7E_DPX2_FAULT	Fehler beim DP–Auftrag für den Datensatz–Transfer
M7E_DP_SLAVE_STATE	DP–Slave ist nicht in DATA Zustand
M7E_INVAL_DEV	Baugruppe eines DP–Slaves ist nicht vorhanden

Siehe auch

M7LoadRecord, M7Store Record

M7LoadWord

Funktion

Wort aus Prozeßabbild laden

Syntax

```
#include <m7api.h>
UWORD        M7LoadWord(
               UWORD PIType,
               UWORD ByteOffset,
               M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalbytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion adressiert ein Wort im Prozeßabbild, das mit *PIType* bestimmt wird, und liefert als Rückgabewert den Inhalt des adressierten Wortes.

Der Inhalt des Wortes wird zuvor aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Als Rückgabewert wird von der Funktion der Inhalt des adressierten Wortes in *Intel*-Zahlendarstellung übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType* oder *ByteOffset*

Siehe auch

M7LoadBit, M7LoadByte, M7LoadDWord

M7LocateObject

Funktion

Anfangsadresse des Nutzdatenbereichs eines S7-Objektes ändern

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LocateObject(
                  UBYTE ObjType,
                  UWORD Part,
                  VOID_PTR Ptr
                  BOOL Copy);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
Ptr	neue Anfangsadresse des S7-Objektes
Copy	Behandlung des neuen Speicherbereichs TRUE Die Nutzdaten des Objekts werden in den neuen Speicherbereich kopiert. FALSE Die Nutzdaten des Objekts werden nicht übertragen.

Beschreibung

Die Funktion ändert die Anfangsadresse des Nutzdatenbereichs eines S7-Objektes, das durch die o. g. Parameter beschrieben ist. Für Objekte im SRAM (remanent) kann die Funktion nicht verwendet werden.

Gemäß Parameter *Copy* werden die Nutzdaten des Objekts in den neuen Bereich übertragen oder nicht.

Beim Aufruf der Funktion müssen Sie sicherstellen, daß ab der neuen Anfangsadresse genügend Speicherplatz zur Verfügung steht.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_PART	Teilbereich nicht vorhanden.
M7E_REM_OBJ	Aktion für remanente Objekte nicht erlaubt.

Siehe auch **M7CreateObject, M7DeleteObject, M7RemoveObject, M7StoreObject**

M7OVSCompress

Funktion

OVS komprimieren

Syntax

#include <m7api.h>
M7ERR_CODE M7OVSCompress(M7CONNID *ConnID***);**

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.

Beschreibung

Mit der Funktion `M7OVSCompress` wird eine Speicherkomprimierung einer S7–CPU angefordert (OVS Komprimieren).

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Hinweis

Die Funktion `M7OVSCompress` ist nur für S7–CPUs verfügbar.

Siehe auch

M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite

M7OVSDelete

Funktion

Bausteine über OVS löschen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSDelete(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UBYTE nBlks,
                  M7BLKLIST_PTR pBlkList);
```

Parameter

Parametername	Bedeutung
flags	Es müssen eines oder beide der folgenden Flags gesetzt sein: A_PASSIV: Löschen der passiven Bausteine. A_LINKED_IN: Löschen der eingeketteten Bausteine. Sind in der Blockliste nur Blöcke eines Blocktyps vorhanden, aber beide Flags gesetzt, wird der Auftrag komplett abgewiesen.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nBlks	Anzahl der Einträge in der Blockliste. Ist *nBlks* gleich 0, werden alle Bausteine im RAM–Speicher gelöscht.
pBlkList	Zeiger auf die Blockliste mit den zu löschenden Bausteinen. Die Blockliste besteht aus Einträgen der Struktur **M7BLKLIST** Die Beschreibung von **M7BLKLIST** finden Sie in Kapitel 3.

Beschreibung

Mit der Funktion `M7OVSDelete` werden die in der Blockliste angegebenen Bausteine gemeinsam gelöscht. Es besteht die Möglichkeit sowohl kopierte, als auch eingekettete Bausteine zu löschen. Die Bausteine werden nur dann gelöscht, wenn alle angegebenen Bausteine vorhanden sind.

Die maximale Anzahl der zu löschenden Bausteine ist in Abhängigkeit von der maximalen PDU–Größe (siehe `M7GetPduSize`) durch folgenden Wert festgelegt:

```
max_anzahl = (maxpdusize – 28)/8
```

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung

Fehlercode	Bedeutung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch **M7OVSCompress, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite**

M7OVSFindFirst

Funktion **Erster Eintrag aus OVS Directory auslesen**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSFindFirst (
                UDWORD flags,
                M7CONNID ConnID,
                UWORD BlkTyp,
                UWORD Language,
                M7BLKINFO_PTR pFFBlkInfo);
```

Parameter

Parametername	Bedeutung
flags	Es müssen eines oder beide der folgenden Flags gesetzt sein: A_PASSIV: Anzeigen von passiven Bausteinen. A_LINKED_IN: Anzeigen von eingeketteten Bausteinen. Zusätzlich können noch folgende Flags gesetzt werden: A_DIRECTORY Anzeigen von Bausteinen des Blocktyps mit der niedrigsten Typnummer A_LANGUAGE Anzeigen von Bausteinen der angegebenen Programmiersprache
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
BlkTyp	Falls A_Directory nicht spezifiziert wurde, enthält der Parameter den Bausteintyp: M7BLKTYP_OB Organisationsbaustein M7BLKTYP_DB Datenbaustein M7BLKTYP_FC Funktionsaufruf M7BLKTYP_SFC Systemfunktionsaufruf M7BLKTYP_FB Funktionsbaustein M7BLKTYP_SFB Systemfunktionsbaustein
Language	Falls A_LANGUAGE spezifiziert wurde, enthält Language die Programmiersprache des zu suchenden Bausteins: M7LANGTYP_HUELSE Container für SFCs und SFBs M7LANGTYP_AWL Baustein in AWL (Anweisungsliste)erstellt M7LANGTYP_KOP Baustein in KOP (Kontaktplan)erstellt M7LANGTYP_FUP Baustein in FUP (Funktionsplan)erstellt M7LANGTYP_SCL Baustein in SCL erstellt M7LANGTYP_DB Baustein mit Baustein–Editor erstellt M7LANGTYP_GRAPH Baustein mit Graph 5 erstellt

Parametername	Bedeutung	
	M7LANGTYP_SDB	Baustein mit Systemdaten–Baustein–Editor erstellt
	M7LANGTYP_CPU	Baustein wurde durch die CPU dynamisch kreiert.
pFFBlkInfo	Zeiger auf eine Find–First–Block–Info–Struktur des Typs **M7BLKINFO** (vgl. Kapitel 3), in die ein gefundener Baustein eingetragen wird.	

Beschreibung

`M7OVSFindFirst` liefert den ersten Verzeichnis–Eintrag in **pFFBlkInfo* gemäß den angegebenen Parametern und startet eine Suchsequenz mit diesen Parametern, die mit `M7OVSFindNext` fortgesetzt werden kann.

Es muß mindestens eines der beiden Flags A_PASSIV und A_LINKED_IN angegeben werden. Wenn A_PASSIV angegeben ist, werden passive Bausteine angezeigt. Wenn A_LINKED_IN angegeben ist, werden eingekettete Bausteine angezeigt.

Wenn A_DIRECTORY spezifiziert ist, wird nach Bausteinen des Blocktyps mit der niedrigsten Typnummer gesucht. BlkTyp braucht dann nicht angegeben zu werden.

Wenn A_LANGUAGE angegeben ist, wird nach Bausteinen der angegebenen Programmiersprache gesucht.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_EOF	Dateiende bzw. Verzeichnisende erreicht
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7OVSCompress, M7OVSDelete, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite

M7OVSFindNext

Funktion

Lesen des OVS Directory fortsetzen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSFindNext (
                  UDWORD flags,
                  M7CONNID ConnID,
                  M7BLKINFO_PTR pFFBlkInfo);
```

Parameter

Parametername	Bedeutung
flags	Es müssen dieselben Flags spezifiziert werden wie in `M7OVSFindFirst`.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
pFFBlkInfo	Zeiger auf eine Find–First–Block–Info–Struktur, in die ein gefundener Baustein eingetragen wird (siehe `M7OVSFindFirst`).

Beschreibung

Es müssen dieselben Flags angegeben werden, wie beim vorausgegangenen `M7OVSFindFirst`–Aufruf. `M7OVSFindNext` liefert den nächsten Verzeichnis–Eintrag in *pFFBlkInfo* in dieser Suchsequenz.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_EOF	Dateiende bzw. Verzeichnisende erreicht
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite

M7OVSLinkIn

Funktion **OVS Einketten**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSLinkIn(
                  M7CONNID ConnID,
                  UBYTE nBlks,
                  M7BLKLIST_PTR pBlkList);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nBlks	Anzahl der Einträge in der Blockliste. Ist *nBlks* gleich 0, werden alle kopierten Bausteine eingekettet.
pBlkList	Zeiger auf die Blockliste mit den einzuketten den Bausteinen. Die Blockliste besteht aus Einträgen der Struktur **M7BLKLIST**.Die Beschreibung von **M7BLKLIST** finden Sie in Kapitel 3.

Beschreibung

Mit der Funktion `M7OVSLinkIn` wird die Anzahl *nBlks* der auf der CPU befindlichen Bausteine gemeinsam aktiviert.

Die maximale Anzahl der einzukettenden Bausteine ist in Abhängigkeit von der maximalen PDU–Größe (siehe `M7GetPduSize`) durch folgenden Wert festgelegt :

```
max_anzahl = (maxpdusize – 28)/8
```

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch **M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSMemMode, M7OVSRead, M7OVSWrite**

M7OVSMemMode

Funktion

OVS Speichermodus einstellen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSMemMode(
                  UDWORD flags,
                  M7CONNID ConnID);
```

Parameter

Parametername	Bedeutung
flags	A_PLC_RAM: Speichermodus auf RAM stellen. A_PLC_EPROM: Speichermodus auf EPROM stellen. Es muß genau eines der beiden Flags gesetzt werden.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.

Beschreibung

Mit der Funktion `M7OVSMemMode` kann der M7/S7–CPU Speicher in RAM oder EPROM–Mode geschalten werden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSRead, M7OVSWrite

M7OVSRead

Funktion **OVS Hochladen**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSRead (
                UDWORD flags,
                M7CONNID ConnID,
                UBYTE_PTR pBitmap,
                UBYTE_PTR pBuffer,
                UDWORD nBufsiz,
                UWORD BlkTyp,
                UWORD BlkNum,
                UDWORD *pnBytes);
```

Parameter

Parametername	Bedeutung
flags	A_PASSIV: Hochladen eines passiven Bausteins. A_LINKED_IN: Hochladen eines eingeketteten Bausteins. Mindestens eines der beiden obigen Flags muß gesetzt sein. Wenn beide Flags gesetzt sind, muß auch A_HEADER gesetzt sein. A_SSB: Nur die Schnittstellenbeschreibung lesen. A_HEADER: Nur der Baustein–Header wird gelesen. A_FILE Wenn A_FILE gesetzt ist, gibt *pBuffer* den Namen der Datei an, in der der Baustein abgelegt wird. Sonst erfolgt die Ablage im Speicher.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
pBitmap	Ein–Byte Bitmap; falls A_HEADER–Mode spezifiziert ist, wird hier der Ablageort des Objekts zurückgeliefert. Das zurückgelieferte Bitmap kann mit den folgenden Kennzeichen logisch verknüpft werden: M7BLKINFO_PASSIV Baustein ist im Ladespeicher (kopiert) M7BLKINFO_ACTIV Baustein ist im Arbeitsspeicher (eingekettet) M7BLKINFO_RAM Baustein ist im RAM bzw. RAM–Modus M7BLKINFO_EPROM Baustein ist im EPRO bzw. EPROM–Modus M7BLKINFO_BESY Baustein ist Bestandteil des Betriebssystems
pBuffer	Empfangspuffer. Wenn A_FILE gesetzt ist, gibt *pBuffer* den Namen der Datei an, in der der Baustein abgelegt wird.
nBufsiz	Größe des Empfangspuffers. Wenn A_FILE gesetzt ist, ist *nBufsiz* ohne Bedeutung.

Parametername	Bedeutung
BlkTyp	Bausteintypen: M7BLKTYP_OB Organisationsbaustein M7BLKTYP_DB Datenbaustein M7BLKTYP_FC Funktionsaufruf M7BLKTYP_SFC Systemfunktionsaufruf M7BLKTYP_FB Funktionsbaustein M7BLKTYP_SFB Systemfunktionsbaustein
BlkNum	Nummer des Bausteins
pnBytes	Zeiger auf die Anzahl gelesener Bytes bzw. 0, wenn A_FILE gesetzt ist und der Datenbaustein in eine Datei abgelegt wird.

Beschreibung

Diese Funktion lädt einen Baustein der M7/S7–CPU in einen Pufferbereich bzw. als Datei auf die Festplatte des M7 hoch.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich durchgeführt.

< M7SUCCESS Ein Fehler ist aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_BLOCK_TOO_LARGE	Puffer nicht ausreichend
M7E_KSUB_FILEIO	Fehler in der Dateibearbeitung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSWrite

M7OVSSetObjectHeader

Funktion **Setzen eines S7–Objekt–Header**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSSetObjectHeader(
                  UBYTE_PTR ptr,
                  UWORD BlkNum,
                  UDWORD nLength,
                  UBYTE Language,
                  UBYTE Type,
                  UBYTE Attribute,
                  UBYTE ProtectionLevel);
```

Parameter

Parametername	Bedeutung
ptr	Zeiger auf den Speicherbereich, in den der S7–Objekt–Header abgelegt wird. Der Speicherbereich muß mindestens S7_OBJECT_HEADER_LENGTH Bytes groß sein.
BlkNum	Bausteinnummer
nLength	Gesamtlänge des Bausteins in Bytes
Language	Sprache, in der der Baustein erstellt wurde: M7LANGTYP_HUELSE – Container für SFCs und SFBs M7LANGTYP_AWL – Baustein in AWL (Anweisungsliste) erstellt M7LANGTYP_KOP – Baustein in KOP (Kopplungsplan) erstellt M7LANGTYP_FUP – Baustein in FUP (Funktionsplan) erstellt M7LANGTYP_SCL – Baustein in SCL erstellt M7LANGTYP_DB – Baustein mit Baustein–Editor erstellt M7LANGTYP_GRAPH – Baustein mit Graph 5 erstellt M7LANGTYP_SDB – Baustein mit Systemdatenbaustein–Editor erstellt M7LANGTYP_CPU – Baustein wurde durch die CPU dynamisch kreiert.
Type	Bausteintypen: M7BLKTYP_OB – Organisationsbaustein M7BLKTYP_DB – Datenbaustein M7BLKTYP_FC – Funktionsaufruf M7BLKTYP_SFC – Systemfunktionsaufruf M7BLKTYP_FB – Funktionsbaustein M7BLKTYP_SFB – Systemfunktionsbaustein

Parametername	Bedeutung
Attribute	Reserviert, muß mit 0 belegt werden
WriteProtect	Erlaubte Zugriffe: 0 Lesen und schreiben 1 Nur lesen 2 Lesen und Schreiben nicht erlaubt 3 Know–How Protection

Beschreibung

Die Funktion `M7OVSSetObjectHeader` setzt den Header für einen mit der Funktion `M7OVSWrite` zu schreibenden Baustein. Die Gesamtlänge des Bausteins muß mindestens S7_OBJECT_HEADER_LENGTH sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_PARAM	Parameterfehler

Siehe auch

M7OVSWrite

M7OVSWrite

Funktion **OVS Kopieren**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSWrite(
                UDWORD flags,
                M7CONNID ConnID,
                UBYTE_PTR pBuffer,
                UDWORD nBufsiz,
                UWORD BlkTyp,
                UWORD BlkNum);
```

Parameter

Parametername	Bedeutung
flags	Flags A_UNCONDITIONAL Falls nicht gesetzt, wird ein vorhandener Baustein gleichen Typs und mit der gleichen Nummer nicht überschrieben. Falls A_UNCONDITIONAL gesetzt ist, wird ein vorhandener Baustein gleichen Typs und mit der gleichen Nummer überschrieben. A_FILE Wenn A_FILE gesetzt ist, zeigt *pBuffer* auf einen String mit einem Filenamen. Die spezifizierte Datei enthält den Baustein. A_ZERO_FLAG Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
pBuffer	Datenpuffer mit den Daten des Bausteins. Wenn A_FILE gesetzt ist, zeigt *pBuffer* auf einen String mit einem Filenamen. Die spezifizierte Datei enthält den Baustein.
nBufsiz	Länge des Datenpuffers. Wenn A_FILE gesetzt ist ohne Bedeutung.
BlkTyp	Bausteintypen: M7BLKTYP_OB Organisationsbaustein M7BLKTYP_DB Datenbaustein M7BLKTYP_FC Funktionsaufruf M7BLKTYP_SFC Systemfunktionsaufruf M7BLKTYP_FB Funktionsbaustein M7BLKTYP_SFB Systemfunktionsbaustein
BlkNum	Nummer des Bausteins

Beschreibung

Die Funktion `M7OVSWrite` kopiert den angegebenen Baustein aus dem angegebenen Puffer oder Datei in den Speicher einer entfernten S7–CPU oder eines M7.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_FILEIO	Fehler in der Dateibearbeitung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Hinweis

Bei M7 ist kein Wiederanlauf möglich.

Siehe auch

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSSetObject-Header

M7PBKBrcv

Funktion

Blockorientierter Datenempfang über projektierte Verbindungen

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKBrcv(
                UDWORD flags,
                M7CONNID ConnID,
                UDWORD R_ID,
                M7VARADDR_PTR pDstVar,
                UDWORD nLength,
                M7COMMFRB_PTR pCommFRB
                unsigned int MPrio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	A_USER	Das A_USER Flag dient zur Steuerung des Parameters *pDstVar* (siehe unten).
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.	
R_ID	Bausteinkennung für den entfernten B_SEND– bzw. `M7PBKBsend`–Aufruf .	
pDstVar	Zeiger auf den Empfangspuffer. A_USER nicht gesetzt Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert einen zusammenhängenden Bereich an Items eines lokalen S7–Objekts, in den die empfangenen Daten kopiert werden. A_USER gesetzt Zeiger auf einen Puffer, in den die empfangenen Daten geschrieben werden.	
nLength	Gesamtlänge des Puffers in Bytes.	
pCommFRB	Zeiger auf den Function Request Block.	
MPrio	Priorität der verschickten Nachricht (0–255).	

Beschreibung

`M7PBKBrcv` startet den asynchronen Empfangsauftrag für einen Puffer von *nLength* Bytes über die Verbindung *ConnID* von einem BSEND–Baustein bzw. `M7PBKBsend`–Aufruf mit Kennung *R_ID*. Die Daten werden in Abhängigkeit der angegebenen *flags* in einen Puffer im Adreßbereich der Task (Flags=A_USER) oder in den Datenbereich des S7–Objekt–Servers (flags=0) geschrieben.

Wenn das Flag A_USER nicht gesetzt ist**,** wird der Parameter *nLength* nicht ausgewertet, sondern die Pufferlänge wird aus der Struktur, auf welche der Parameter *pDstVar* zeigt, ermittelt. In diesem Fall kann ein beliebiger Wert für *nLength* angegeben werden. Ist jedoch das Flag A_USER gesetzt, muß in *nLength* weiterhin die Pufferlänge angegeben werden.

Wenn die Daten von der lokalen Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_NDR Nachricht erzeugt. In der Zeit zwischen dem `M7PBKBrcv`–Aufruf und dem Empfang der M7MSG_PBK_NDR Message darf der FRB nicht anderweitig verwendet werden.

Nach dem Empfang einer M7MSG_PBK_NDR Message kann über den `M7GetCommRcvLen`–Aufruf die Anzahl der empfangenen Bytes ermittelt werden.

`M7PBKBrcv`–Aufrufe können mit `M7PBKCancel` abgebrochen werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7GetCommRcvLen, M7PBKBsend, M7PBKCancel

M7PBKBsend

Funktion

Blockorientiertes Senden über projektierte Verbindungen

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKBsend(
                UDWORD flags,
                M7CONNID ConnID,
                UDWORD R_ID,
                M7VARADDR_PTR pSrcVar,
                UDWORD nLength,
                M7COMMFRB_PTR pCommFRB
                unsigned int MPrio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	A_USER	Das A_USER Flag dient zur Steuerung des Parameters *pSrcVar* (siehe unten).
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.	
R_ID	Bausteinkennung für den entfernten BRCV–Baustein bzw. `M7PBKBrcv`–Aufruf.	
pSrcVar	Zeiger auf die zu sendenden Daten. A_USER nicht gesetzt Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert einen zusammenhängenden Bereich an Items in einem lokalen S7–Objekt. A_USER gesetzt Zeiger auf einen Puffer mit den zu versendenden Daten.	
nLength	Gesamtlänge des Puffers in Bytes.	
pCommFRB	Zeiger auf den Function Request Block.	
MPrio	Priorität der verschickten Nachricht (0–255).	

Beschreibung

`M7PBKBsend` startet das asynchrone Senden eines Datenbereichs der Länge *nLength* über die Verbindung *ConnID* an den mit Kennung *R_ID* spezifizierten BRCV–Baustein bzw. `M7PBKBrcv`–Aufruf der entfernten Station.

Wenn flags=A_USER, beginnen die zu sendenden Daten an der mit *pSrcVar* spezifizierten Adresse.

Ist flags=0, spezifiziert *pSrcVar* die Adresse der zu sendenden Variable im Adreßbereich des S7–Objekt–Servers.

Wenn das Flag A_USER nicht gesetzt ist**,** wird der Parameter *nLength* nicht ausgewertet, sondern die Pufferlänge wird aus der Struktur, auf welche der Parameter *pSrcVar* zeigt, ermittelt. In diesem Fall kann ein beliebiger Wert für *nLength* angegeben werden. Ist jedoch das Flag A_USER gesetzt, muß in *nLength* weiterhin die Pufferlänge angegeben werden.

Wenn die Daten von der entfernten Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_DONE–Message erzeugt. In der Zeit zwischen dem `M7PBKBsend`–Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

`M7PBKBsend`–Aufrufe können mit `M7PBKCancel` abgebrochen werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch **M7PBKBrcv, M7PBKCancel**

M7PBKCancel

Funktion

Laufenden Sende– bzw. Empfangsauftrag über projektierte Verbindungen abbrechen

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKCancel(
M7CONNID *ConnID*,
M7COMMFRB_PTR *pCommFRB*);

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
pCommFRB	Zeiger auf den Function Request Block.

Beschreibung

`M7PBKCancel` bricht einen laufenden `M7PBKBsend`, `M7PBKBrcv` oder `M7PBKURcv` Auftrag ab. Der abzubrechende Sende– bzw. Empfangsauftrag wird durch die Parameter *ConnID* und *pCommFRB* angegeben (siehe `M7PBKBsend` bzw. `M7PBKBrcv`).

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_NO_SUCH_FRB	*M7COMMFRB nicht in Bearbeitung
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7PBKBsend, M7PBKBrcv

M7PBKGet

Funktion

Asynchrones Lesen von Variablen über projektierte Verbindungen starten

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKGet(
                  M7CONNID ConnID,
                  UBYTE nVars,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pDstVar,
                  M7COMMFRB_PTR pCommFRB
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nVars	Anzahl der zu lesenden Variablen.
pRemoteVar	Array mit den Adreßspezifikationen **(M7VARADDR)**. Sie spezifiziert die zu lesenden Variablen der entfernten Station.
pDstVar	Array mit den Adreßspezifikationen **(M7VARADDR)**. Sie spezifiziert für den Datenempfang die Variablen des S7–Objekt–Servers in der lokalen Station.
pCommFRB	Zeiger auf den Function Request Block.
MPrio	Priorität der verschickten Nachricht (0–255).

Beschreibung

`M7PBKGet` startet den asynchronen Einlesevorgang von *nVars* Variablen aus dem Variablenbereich des S7–Objekt–Servers bzw. S7–CPU–Datenbereich der entfernten Station in den Variablenbereich des S7–Objekt–Servers der lokalen Station.

Für den Aufruf `M7PBKGet` gelten folgende Bedingungen für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize - 14$$

and

$$0 \quad maxpdusize - 12 * (nVars \quad 1)$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit `M7KInitiate` eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

pRemoteVar und *pDstVar* sind Zeiger auf Arrays mit jeweils *nVars* Elementen. Jedes Element spezifiziert einen zusammenhängenden Bereich an Items im S7–Objekt–Server bzw. im S7–CPU–Datenbereich (siehe `M7BUBRead`).

Wenn die Daten in den über *pDstVar* spezifizierten Datenbereich abgelegt sind, wird für *pCommFRB* eine M7MSG_PBK_NDR Message erzeugt. In der Zeit zwischen dem `M7PBKGet`–Aufruf und dem Empfang der M7MSG_PBK_NDR Message darf der FRB nicht anderweitig verwendet werden.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7KInitiate, M7PBKPut, M7BUBRead

M7PBKIAbort

Funktion

Schließen einer Applikationsbeziehung (für SIMATIC stationsinterne Kommunikation über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKIAbort(
                  UBYTE IOID,
                  UWORD LADDR);
```

Parameter

Parametername	Bedeutung
IOID	Ein– oder Ausgangsadreßbereich (M7KIOID_IN, M7KIOID_OUT)
LADDR	E/A–Anfangsadresse der entfernten Station (0–MAX_LOG_ADDR)

Beschreibung

Die Funktion `M7PBKIAbort` schließt eine Applikationsbeziehung zwischen Client und Server, die mit den Funktionen `M7PBKIPut` oder `M7PBKIGet` eingerichtet wurden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zur Station *LADDR* ist zur Zeit aktiv, sie kann nicht geschlossen werden.

Siehe auch

M7PBKIGet, M7PBKIPut

M7PBKIGet

Funktion

Asynchrones Lesen einer Variablen starten (für SIMATIC stationsinterne Kommunikation über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKIGet(
                  UDWORD flags,
                  UBYTE IOID,
                  UWORD LADDR,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pDstVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int Mprio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	CONT	Ist CONT gesetzt, bleibt die durch die Datenübertragung eingerichtete Applikationsbeziehung erhalten. Ist CONT nicht gesetzt, wird die durch die Datenübertragung eingerichtete Applikationsbeziehung nach der Datenübertragung geschlossen.
IOID	Ein– oder Ausgangsadreßbereich (M7KIOID_IN, M7KIOID_OUT)	
LADDR	E/A–Anfangsadresse der entfernten Station (0–MAX_LOG_ADDR)	
pRemoteVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert einen zusammenhängenden Bereich an Items eines S7–Objekts in der entfernten Station.	
pDstVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert für den Datenempfang eine Variable des S7–Objekts in der lokalen Station.	
pCommFRB	Zeiger auf den Function Request Block	
MPrio	Priorität der verschickten Nachricht (0-255)	

Beschreibung

`M7PBKIGet` startet den asynchronen Einlesevorgang von einer Variablen aus dem Variablenbereich des S7-Objekt-Servers der entfernten Station *LADDR* in den Variablenbereich des S7-Objekt-Servers der lokalen Station.

Falls zu der entfernten Station noch keine Applikationsbeziehung besteht, wird sie eingerichtet. Ist das Flag `CONT` gesetzt, bleibt diese Beziehung auch nach Ende der Datenübertragung bestehen. Wird die Applikationsbeziehung nicht mehr benötigt, muß sie mit dem Aufruf `M7PBKIAbort` geschlossen werden. Ist des Flag `CONT` nicht gesetzt, wird nach Beendigung der Datenübertragung die Applikationsbeziehung automatisch wieder geschlossen.

pRemoteVar und *pDstVar* sind Zeiger auf Elemente, die einen zusammenhängenden Bereich an Items im S7-Objekt-Server spezifizieren (siehe `M7BUBRead`).

Wenn die Daten in den über *pDstVa*r spezifizierten Datenbereich abgelegt sind, wird für *pCommFRB* eine M7MSG_PBK_NDR Message erzeugt. Zwischen dem `M7PBKIGet`-Aufruf und dem Empfang der M7MSG_PBK_NDR Message darf der FRB nicht anderweitig verwendet werden.

Hinweis

Die Nutzdatenlänge beträgt 76 Byte.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zur Station *LADDR* ist zur Zeit aktiv. Es können keine Daten übertragen werden.
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7BUBRead, M7GetCommStatus, M7PBKIAbort, M7PBKIPut

M7PBKIPut

Funktion

Asynchrones Schreiben einer Variablen starten (für SIMATIC stationsinterne Kommunikation über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKIPut(
                  UDWORD flags,
                  UBYTE IOID,
                  UWORD LADDR,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pSrcVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int Mprio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	CONT	Ist CONT gesetzt, bleibt die durch die Datenübertragung eingerichtete Applikationsbeziehung erhalten. Ist CONT nicht gesetzt, wird die durch die Datenübertragung eingerichtete Applikationsbeziehung nach der Datenübertragung geschlossen.
IOID	Ein– oder Ausgangsadreßbereich (M7KIOID_IN, M7KIOID_OUT)	
LADDR	E/A–Anfangsadresse der entfernten Station (0–MAX_LOG_ADDR)	
pRemoteVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert die zu überschreibenden Variablen im S7–Objekt–Server bzw. S7–CPU–Datenbereich der entfernten Station.	
pSrcVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert die zu sendenden Variablen im S7-Objekt-Servers der lokalen Station	
pCommFRB	Zeiger auf den Function Request Block	
MPrio	Priorität der verschickten Nachricht (0-255)	

Beschreibung

`M7PBKIPut` startet das asynchrone Überschreiben von einer Variablen im S7-Objekt-Server bzw. S7-CPU-Datenbereich der entfernten Station *LADDR* mit den Werten einer lokalen Variable des S7-Objekt-Servers.

Falls zu der entfernten Station noch keine Applikationsbeziehung besteht, wird sie eingerichtet. Ist das Flag `CONT` gesetzt, bleibt diese Beziehung auch nach Ende der Datenübertragung bestehen. Wird die Applikationsbeziehung nicht mehr benötigt, muß sie mit dem Aufruf `M7PBKIAbort` geschlossen werden. Ist des Flag `CONT` nicht gesetzt, wird nach Beendigung der Datenübertragung die Applikationsbeziehung automatisch wieder geschlossen.

pRemoteVar und *pSrcVar* sind Zeiger auf die Adreßspezifikationen der betreffenden entfernten bzw. lokalen Variable im S7-Objekt-Server/S7-CPU-Datenbereich.

Wenn die Daten im entfernten Rechner abgelegt sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_DONE Message erzeugt. In der Zeit zwischen dem `M7PBKIPut`-Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

Hinweis

Die Nutzdatenlänge beträgt 76 Byte.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zur Station *LADDR* ist zur Zeit aktiv. Es können keine Daten übertragen werden.
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7BUBWrite, M7GetCommStatus, M7PBKIAbort, M7PBKIGet

M7PBKPrint

Funktion

Senden von Daten mit einer Formatbeschreibung

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKPrint(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UBYTE printerID,
                  UBYTE * fmt,
                  UBYTE nVars,
                  M7VARDATA_PTR pSrcVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
flags	Flags (A_ZERO_FLAG)
ConnID	Verbindungs–ID
printerID	Printer–ID
fmt	Formatstring(null–terminiert)
nVars	Anzahl Sendeparameter
pSrcVar	Sendeparameter
pCommFRB	Zeiger auf den Function Request Block
MPrio	Priorität der verschickten Nachricht (0–255)

Beschreibung

`M7PBKPrint` startet das asynchrone Senden mehrerer Datenbereiche und einem Formatstring über die Verbindung *ConnID* an die entfernten Station.

Der Parameter *nVars* spezifiziert die Anzahl der zu übertragenden Datenbereiche. *pSrcVar* zeigt auf ein Array von M7VARDATA Objekten. Diese Objekte enthalten jeweils einen zu sendenden Datenbereich.

Der Parameter *fmt* zeigt auf einen null–terminierten Formatstring.

Wenn die Daten von der entfernten Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_DONE-Message erzeugt. In der Zeit zwischen dem `M7PBKPrint`-Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Für den Aufruf `M7PBKPrint` gilt folgende Bedingung für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize \quad 26 \quad längefmt \quad 4 * nVars$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit `M7KInitiate` eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_PRIO	Falsche Priorität
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	STEP7–Verbindungsbeschreibung gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate

M7PBKPut

Funktion

Asynchrones Schreiben von Variablen über projektierte Verbindungen starten

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKPut(
                  M7CONNID ConnID,
                  UBYTE nVars,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pSrcVar,
                  M7COMMFRB_PTR pCommFRB
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.
nVars	Anzahl der zu schreibenden Variablen.
pRemoteVar	Array mit den Adreßspezifikationen **(M7VARADDR)**. Sie spezifiziert die zu überschreibenden Variablen im S7–Objekt–Server bzw. S7–CPU–Datenbereich der entfernten Station .
pSrcVar	Array mit den Adreßspezifikationen **(M7VARADDR)**. Sie spezifiziert die zu senden Variablen im S7–Objekt–Servers der lokalen Station
pCommFRB	Zeiger auf den Function Request Block.
MPrio	Priorität, mit der die Message verschickt wird (0–255).

Beschreibung

`M7PBKPut` startet das asynchrone Überschreiben von *nVars* Variablen im S7–Objekt–Server bzw. S7–CPU–Datenbereich der entfernten Station mit den Werten lokaler Variablen des S7–Objekt–Servers.

*pRemoteVa*r und *pSrcVar* sind Zeiger auf *nVars*–elementige Arrays mit den Adreßspezifikationen der betreffenden entfernten bzw. lokalen Variablen im S7–Objekt–Server/S7–CPU–Datenbereich.

Wenn die Daten im entfernten Rechner abgelegt sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_DONE Message erzeugt. In der Zeit zwischen dem `M7PBKPut`–Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

Für den Aufruf `M7PBKPut` gilt folgende Bedingung für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \qquad maxpdusize - 12 * (nVars \quad 1)$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit `M7KInitiate` eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7KInitiate, M7BKGet, M7BUBWrite, M7GetCommStatus

M7PBKResume

Funktion **Wiederanlauf–Anforderung an Kommunikationspartner senden**

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKResume(M7CONNID *ConnID***);**

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.

Beschreibung

`M7PBKResume` sendet eine WIEDERANLAUF–Anforderung an den entfernten Rechner.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Hinweis

Bei M7 ist kein Wiederanlauf möglich.

Siehe auch

M7KInitiate, M7PBKStart, M7PBKStop, M7PBKStatus

M7PBKStart

Funktion

Neustart–Anforderung an Kommunikationspartner senden

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKStart(M7CONNID *ConnID***);**

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.

Beschreibung

Die Funktion `M7PBKStart` sendet an den Zielrechner eine NEUSTART–Anforderung für alle Anwenderprogramme.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7PBKResume, M7PBKStop, M7PBKStatus

M7PBKStatus

Funktion

Status des Kommunikationspartners ermitteln

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKStatus (
                  M7CONNID ConnID,
                  M7PBKSTATUS_PTR pPBKStatus,
                  UDWORD nPBKStatus,
                  UDWORD *pnBytes);
```

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem M7KInitiate–Aufruf.
pPBKStatus	Zeiger auf Struktur des Typs **M7PBKSTATUS** in der der logische und physikalische Status des entfernten Geräts abgelegt werden (vgl. Kapitel 3).
nResultBufsiz	Länge des Ergebnispuffers.
pnBytes	Zeiger auf Anzahl gelesener Bytes.

Beschreibung

Die Funktion `M7PBKStatus` liefert den aktuellen “Status Virtuelles Gerät”

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7PBKResume, M7PBKStop, M7PBKStart

M7PBKStop

Funktion

Stop–Anforderung an Kommunikationspartner senden

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKStop (M7CONNID *ConnID***);**

Parameter

Parametername	Bedeutung
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.

Beschreibung

Die Funktion `M7PBKStop` sendet eine STOP–Anforderung für die Gesamtheit der Anwenderprogramme auf dem Zielrechner.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7PBKResume, M7PBKSatus, M7PBKStart

M7PBKURcv

Funktion

Unkoordiniertes Empfangen über projektierte Verbindungen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKURcv(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UDWORD R_ID,
                  UBYTE nVars,
                  M7VARDATA_PTR pDstVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
flags	Flags (A_ZERO_FLAG)
ConnID	Verbindungs–ID
R_ID	Bausteinkennung für den entfernten U_SEND– bzw. `M7PBKUSend`–Aufruf.
nVars	Anzahl Empfangsparameter
pDstVar	Empfangsparameter
pCommFRB	Zeiger auf den Function Request Block
MPrio	Priorität der verschickten Nachricht (0–255)

Beschreibung

`M7PBKURcv` startet den asynchronen Empfang mehrere Datenbereiche über die Verbindung *ConnID* von einem USEND–Baustein bzw. `M7PBKUSend`–Aufruf mit der Kennung *R_ID*.

Der Parameter *nVars* spezifiziert die Anzahl der zu empfangenen Datenbereiche. *pSrcVar* zeigt auf ein Array von M7VARDATA Objekten. Diese Objekte enthalten jeweils einen Datenbereich für die Empfangsdaten.

Wenn die Daten von der lokalen Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pConnFRB* eine M7MSG_PBK_NDR-Message erzeugt. In der Zeit zwischen dem `M7PBKURcv`-Aufruf und dem Empfang der M7MSG_PBK_NDR Message darf der FRB nicht anderweitig verwendet werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Für den Aufruf `M7PBKURcv` gilt folgende Bedingung für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize \quad 24 \quad 4 * nVars$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit `M7KInitiate` eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_PRIO	Falsche Priorität
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	STEP7–Verbindungsbeschreibung gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7PBKUSend

M7PBKUsend

Funktion

Unkoordiniertes Senden über projektierte Verbindungen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKUSend(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UDWORD R_ID,
                  UBYTE nVars,
                  M7VARDATA_PTR pSrcVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
flags	Flags (A_ZERO_FLAG)
ConnID	Verbindungs–ID
R_ID	Bausteinkennung für den entfernten U_RCV– bzw. `M7PBKURcv`–Aufruf.
nVars	Anzahl Sendeparameter
pSrcVar	Sendeparameter
pCommFRB	Zeiger auf den Function Request Block
MPrio	Priorität der verschickten Nachricht (0–255)

Beschreibung

`M7PBKUSend` startet das asynchrone Senden mehrerer Datenbereiche über die Verbindung *ConnID* an den mit Kennung *R_ID* spezifizierten URCV-Baustein bzw. `M7PBKURcv`-Aufruf der entfernten Station.

Der Parameter *nVars* spezifiziert die Anzahl der zu übertragenden Datenbereiche.

pSrcVar zeigt auf ein Array von M7VARDATA Objekten. Diese Objekte enthalten jeweils einen zu sendenden Datenbereich.

Wenn die Daten von der entfernten Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pConnFRB* eine M7MSG_PBK_DONE-Message erzeugt. In der Zeit zwischen dem `M7PBKUSend`-Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Für den Aufruf `M7PBKUSend` gilt folgende Bedingung für die maximale Nutzdatenlänge:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize \quad 24 \quad 4 * nVars$$

Dabei ist *maxpdusize* die maximale PDU–Größe für die mit `M7KInitiate` eröffnete Verbindung und *nBytes(i)* die geradzahlig aufgerundete Anzahl von Bytes für die i–te Variable.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_PRIO	Falsche Priorität
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	STEP7–Verbindungsbeschreibung gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7PBKURcv

M7PBKXAbort

Funktion

Schließen einer Applikationsbeziehung (für Kommunikation im MPI–Subnetz über nichtprojektierte Verbindungen)

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKXAbort(UWORD *DEST_ID***);**

Parameter

Parametername	Bedeutung
DEST_ID	MPI–Teilnehmeradresse(0-126).

Beschreibung

Die Funktion `M7PBKXAbort` schließt eine Applikationsbeziehung zwischen Client und Server, die mit den Funktionen `M7PBKXSend`, `M7PBKXPut` oder `M7PBKXGet` eingerichtet wurden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zum Teilnehmer *DEST_ID* ist zur Zeit aktiv, sie kann nicht geschlossen werden.
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_NOT_IMPLEMENTED	Funktion wird nicht unterstützt

Siehe auch

M7PBKXGet, M7PBKXPut, M7PBKXSend

M7PBKXCancel

Funktion

Laufenden Empfangsauftrag von `M7PBKXRcv` **abbrechen (für Kommunikation im MPI–Subnetz über nichtprojektierte Verbindungen)**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKXCancel(
                  M7COMMFRB_PTR CommFRB);
```

Parameter

Parametername	Bedeutung
pCommFRB	Zeiger auf Function Request Block

Beschreibung

`M7PBKCancel` bricht einen laufenden `M7PBKXRcv`-Auftrag ab.

Bis zum Empfang der M7MSG_PBK_NDR Nachricht darf der FRB nicht anderweitig verwendet werden. Falls im asynchronen Teil ein Fehler aufgetreten ist, kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_NOT_IMPLEMENTED	Funktion wird nicht unterstützt

Siehe auch

M7GetCommStatus, M7PBKXRcv

M7PBKXGet

Funktion

Asynchrones Lesen einer Variablen (für Kommunikation im MPI–Subnetz über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKXGet(
                UDWORD flags,
                UWORD DEST_ID,
                M7VARADDR_PTR pRemoteVar,
                M7VARADDR_PTR pDstVar,
                M7COMMFRB_PTR pCommFRB,
                unsigned int Mprio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	CONT	Ist CONT gesetzt, bleibt die durch die Datenübertragung eingerichtete Applikationsbeziehung erhalten.
		Ist CONT nicht gesetzt, wird die durch die Datenübertragung eingerichtete Applikationsbeziehung nach der Datenübertragung geschlossen.
DEST_ID	MPI–Adresse (0-126)	
pRemoteVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert die zu lesenden Variable der entfernten Station.	
pDstVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert für den Datenempfang die Variable des S7–Objekt–Servers in der lokalen Station.	
pCommFRB	Zeiger auf den Function Request Block	
MPrio	Priorität der verschickten Nachricht (0-255)	

Beschreibung

`M7PBKXGet` startet den asynchronen Einlesevorgang von einer Variablen aus dem Variablenbereich des S7-Objekt-Servers bzw. S7-CPU-Datenbereich der entfernten Station *DEST_ID* in den Variablenbereich des S7-Objekt-Servers der lokalen Station.

Falls zu der entfernten Station noch keine Applikationsbeziehung besteht, wird sie eingerichtet. Ist das Flag `CONT` gesetzt, bleibt diese Beziehung auch nach Ende der Datenübertragung bestehen. Wird die Applikationsbeziehung nicht mehr benötigt, muß sie mit dem Aufruf `M7PBKXAbort` geschlossen

werden. Ist des Flag CONT nicht gesetzt, wird nach Beendigung der Datenübertragung die Applikationsbeziehung automatisch wieder geschlossen.

pRemoteVar und *pDstVar* sind Zeiger auf Elemente, die einen zusammenhängenden Bereich an Items im S7-Objekt-Server bzw. im S7-CPU-Datenbereich spezifizieren (siehe M7BUBRead).

Wenn die Daten in den über *pDstVa*r spezifizierten Datenbereich abgelegt sind, wird für *pCommFRB* eine M7MSG_PBK_NDR Message erzeugt. Zwischen dem M7PBKXGet-Aufruf und dem Empfang der M7MSG_PBK_NDR Message darf der FRB nicht anderweitig verwendet werden.

Hinweis

Die Nutzdatenlänge beträgt 76 Byte.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zur Station *DEST_ID* ist zur Zeit aktiv. Es können keine Daten übertragen werden.
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung (*DEST_ID* falsch)
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_NOT_IMPLEMENTED	Funktion wird nicht unterstützt
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7BUBRead, M7GetCommStatus, M7PBKXAbort, M7PBKPut

M7PBKXPut

Funktion

Asynchrones Schreiben einer Variablen starten (für Kommunikation im MPI–Subnetz über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKXPut(
                UDWORD flags,
                UWORD DEST_ID,
                M7VARADDR_PTR pRemoteVar,
                M7VARADDR_PTR pSrcVar,
                M7COMMFRB_PTR pCommFRB,
                unsigned int Mprio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	CONT	Ist CONT gesetzt, bleibt die durch die Datenübertragung eingerichtete Applikationsbeziehung erhalten.
		Ist CONT nicht gesetzt, wird die durch die Datenübertragung eingerichtete Applikationsbeziehung nach der Datenübertragung geschlossen.
DEST_ID	MPI–Adresse (0-126)	
pRemoteVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert die zu überschreibende Variable im S7–Objekt–Server bzw. S7–CPU–Datenbereich der entfernten Station.	
pSrcVar	Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert die zu sendende Variable im S7-Objekt-Servers der lokalen Station	
pCommFRB	Zeiger auf den Function Request Block	
MPrio	Priorität der verschickten Nachricht (0-255)	

Beschreibung

`M7PBKXPut` startet das asynchrone Überschreiben von einer Variablen im S7-Objekt-Server bzw. S7-CPU-Datenbereich der entfernten Station *DEST_ID* mit den Werten einer lokalen Variable des S7-Objekt-Servers.

Falls zu der entfernten Station noch keine Applikationsbeziehung besteht, wird sie eingerichtet. Ist das Flag `CONT` gesetzt, bleibt diese Beziehung auch nach Ende der Datenübertragung bestehen. Wird die Applikationsbeziehung nicht mehr benötigt, muß sie mit dem Aufruf `M7PBKXAbort` geschlossen

werden. Ist des Flag CONT nicht gesetzt, wird nach Beendigung der Datenübertragung die Applikationsbeziehung automatisch wieder geschlossen.

pRemoteVar und *pSrcVar* sind Zeiger auf die Adreßspezifikationen der betreffenden entfernten bzw. lokalen Variable im S7-Objekt-Server/S7-CPU-Datenbereich.

Wenn die Daten im entfernten Rechner abgelegt sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_DONE Message erzeugt. In der Zeit zwischen dem M7PBKXPut-Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

Hinweis

Die Nutzdatenlänge beträgt 76 Byte.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zur Station *DEST_ID* ist zur Zeit aktiv. Es können keine Daten übertragen werden.
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung (*DEST_ID* falsch)
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_NOT_IMPLEMENTED	Funktion wird nicht unterstützt
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7BUBWrite, M7GetCommStatus, M7PBKXAbort, M7PBKPXGet

M7PBKXRcv

Funktion

Daten empfangen (für Kommunikation im MPI–Subnetz über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKXRcv(
                  UDWORD flags,
                  UDWORD R_ID,
                  M7VARADDR_PTR pDstVar,
                  UDWORD nLength,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int MPrio);
```

Parameter

Parametername	Bedeutung
flags	Flags A_ZERO_FLAG: Dieses Flag kann mit den gewünschten Optionen "geodert" werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird. A_USER: Das A_USER Flag dient zur Steuerung des Parameters *pDstVar* (siehe unten).
R_ID	Bausteinkennung für den entfernten X_SEND– bzw. M7PBKXSend–Aufruf.
pDstVar	Zeiger auf den Empfangspuffer. A_USER nicht gesetzt Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert einen zusammenhängenden Bereich an Items eines lokalen S7–Objekts, in den die empfangenen Daten kopiert werden. A_USER gesetzt Zeiger auf einen Puffer, in den die empfangenen Daten geschrieben werden.
nLength	Gesamtlänge des Puffers in Bytes
pCommFRB	Zeiger auf den Function Request Block
MPrio	Priorität der verschickten Nachricht (0-255)

Beschreibung

M7PBKXRcv startet den asynchronen Empfangsauftrag für einen Puffer von *nLength* Bytes von einem X_SEND- bzw. M7PBKXSend-Aufruf mit Kennung *R_ID*. Die Daten werden in Abhängigkeit der angegebenen *flags* in einen Puffer im Adreßbereich der Task (flags=A_USER) oder in den Datenbereich des S7-Objekt-Servers (flags=0) geschrieben.

Wenn das Flag A_USER nicht gesetzt ist**,** wird der Parameter *nLength* nicht ausgewertet, sondern die Pufferlänge wird aus der Struktur, auf welche der

Parameter *pDstVar* zeigt, ermittelt. In diesem Fall kann ein beliebiger Wert für *nLength* angegeben werden. Ist jedoch das Flag A_USER gesetzt, muß in *nLength* weiterhin die Pufferlänge angegeben werden.

Wenn die Daten von der lokalen Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_NDR Nachricht erzeugt. In der Zeit zwischen dem `M7PBKXRcv`-Aufruf und dem Empfang der M7MSG_PBK_NDR Message darf der FRB nicht anderweitig verwendet werden.

Nach dem Empfang einer M7MSG_PBK_NDR Message kann über den `M7GetCommRcvLen`-Aufruf die Anzahl der empfangenen Bytes ermittelt werden.

`M7PBKXRcv`-Aufrufe können mit `M7PBKXCancel` abgebrochen werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Hinweis

Die Nutzdatenlänge beträgt 76 Byte.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_NOT_IMPLEMENTED	Funktion wird nicht unterstützt
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch

M7GetCommRcvLen, M7GetCommStatus, M7PBKXCancel, M7PBKXSend

M7PBKXSend

Funktion

Daten senden (für Kommunikation im MPI–Subnetz über nichtprojektierte Verbindungen)

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKXSend(
                UDWORD flags,
                UWORD DEST_ID,
                UDWORD R_ID,
                M7VARADDR_PTR pSrcVar,
                UDWORD nLength,
                M7COMMFRB_PTR pCommFRB,
                unsigned int MPrio);
```

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
	A_USER	Das A_USER Flag dient zur Steuerung des Parameters *pSrcVar* (siehe unten).
	CONT	Ist CONT gesetzt, bleibt die durch die Datenübertragung eingerichtete Applikationsbeziehung erhalten.
		Ist CONT nicht gesetzt, wird die durch die Datenübertragung eingerichtete Applikationsbeziehung nach der Datenübertragung geschlossen.
DEST_ID	MPI–Adresse (0-126)	
R_ID	Bausteinkennung für den entfernten X_RCV– bzw. `M7PBKXRcv`–Aufruf.	
pSrcVar	Zeiger auf die zu sendenden Daten. A_USER nicht gesetzt Zeiger auf **eine** Struktur vom Typ **M7VARADDR**. Sie spezifiziert einen zusammenhängenden Bereich an Items in einem lokalen S7–Objekt. A_USER gesetzt Zeiger auf einen Puffer mit den zu versendenden Daten.	
nLength	Gesamtlänge des Puffers in Bytes	
pCommFRB	Zeiger auf den Function Request Block	
MPrio	Priorität der verschickten Nachricht (0-255)	

Beschreibung

`M7PBKXSend` startet das asynchrone Senden eines Datenbereichs der Länge *nLength* zum Teilnehmer *DEST_ID* an den mit Kennung *R_ID* spezifizierten X_RCV- bzw. `M7PBKXRcv`-Aufruf der entfernten Station.

Falls zu dem Teilnehmer noch keine Applikationsbeziehung besteht, wird sie eingerichtet. Ist das Flag `CONT` gesetzt, bleibt diese Beziehung auch nach Ende der Datenübertragung bestehen. Wird die Applikationsbeziehung nicht mehr benötigt, muß sie mit dem Aufruf `M7PBKXAbort` geschlossen werden. Ist des Flag `CONT` nicht gesetzt, wird nach Beendigung der Datenübertragung die Applikationsbeziehung automatisch wieder geschlossen.

Wenn das Flag A_USER gesetzt ist, beginnen die zu sendenden Daten an der mit *pSrcVar* spezifizierten Adresse.

Ist das Flag A_USER nicht gesetzt, spezifiziert *pSrcVar* die Adresse der zu sendenden Variable im Adreßbereich des S7-Objekt-Servers.

Wenn das Flag A_USER nicht gesetzt ist**,** wird der Parameter *nLength* nicht ausgewertet, sondern die Pufferlänge wird aus der Struktur, auf welche der Parameter *pSrcVar* zeigt, ermittelt. In diesem Fall kann ein beliebiger Wert für *nLength* angegeben werden. Ist jedoch das Flag A_USER gesetzt, muß in *nLength* weiterhin die Pufferlänge angegeben werden.

Wenn die Daten von der entfernten Station übernommen worden sind oder ein Fehler aufgetreten ist, wird mit *pCommFRB* eine M7MSG_PBK_DONE-Message erzeugt. In der Zeit zwischen dem `M7PBKXSend`-Aufruf und dem Empfang der M7MSG_PBK_DONE Message darf der FRB nicht anderweitig verwendet werden.

Falls im asynchronen Teil ein Fehler aufgetreten ist, wird kann dieser aus dem referenzierten M7COMMFRB mit Hilfe des Makros `M7GetCommStatus` ausgelesen werden.

Hinweis

Die Nutzdatenlänge beträgt 76 Byte.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_KSUB_CONN_ACTIVE	Die Verbindung zur Station *DEST_ID* ist zur Zeit aktiv. Es können keine Daten übertragen werden.
M7E_KSUB_NO_SRV	MPI–Treiber nicht aktiv
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung (*DEST_ID* falsch)
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_LENGTH	Länge falsch.
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_NOT_IMPLEMENTED	Funktion wird nicht unterstützt
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset

Fehlercode	Bedeutung
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_PRIO	Falsche Priorität
M7E_TYPE	Datentyp ist ungültig.

Siehe auch **M7GetCommStatus, M7PBKXAbort, M7PBKXRcv**

M7Read

Funktion

S7-Datenbereich lesen

Syntax

```
#include      <m7api.h>
M7ERR_CODE    M7Read(
                VOID_PTR pBuffer,
                UBYTE ObjType,
                UWORD Part,
                UBYTE DataType,
                UWORD Count,
                UDWORD Addr);
```

Parameter

Parametername	Bedeutung
pBuffer	Zeiger auf den Zielpuffer
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
DataType	Datentyp eines Elementes (vgl. Tabelle 2-9) Beim Datentyp M7DT_BOOL ist für den Parameter LENGTH nur 1 zulässig.
Count	Anzahl der zu lesenden Elemente
Addr	Adresse bzw. Offset innerhalb des Objekts bzw. des Teilbereichs in Bit. Ist *DataType* ≠ BOOL, muß *Addr* ein Vielfaches von 8 Bit sein.

Beschreibung

Die Funktion liest eine bestimmte Anzahl Datenelemente aus einem S7-Datenbereich und kopiert sie in einen Anwenderdatenbereich.

Der Inhalt des Datenbereiches wird nicht aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch.
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_TYPE	Datentyp ist ungültig.

Siehe auch **M7ReadBit, M7ReadByte, M7ReadDWord, M7ReadWord**

M7ReadBit

Funktion

Bit aus S7-Objekt lesen

Syntax

```
#include <m7api.h>
BOOL        M7ReadBit(
              UBYTE ObjType,
              UWORD Part,
              UWORD ByteOffset,
              UBYTE BitOffset,
              M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset des Bytes, in dem sich das gewünschte Bit befindet
BitOffset	Offset des gewünschten Bits innerhalb des Bytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion liest ein Bit aus einem S7-Objekt, das durch die o. g. Parameter beschrieben ist.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, so ist der Rückgabewert der Zustand des adressierten Bits übergeben: Ist der Zustand = ’0’, so wird der Wert FALSE, ist der Zustand = ’1’, so wird der Wert TRUE übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_BIT_OFFSET	Bitoffset innerhalb des Bytes falsch.
M7E_LENGTH	Länge falsch.
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_OFFSET	Falscher Offset.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler.
M7E_PART	Teilbereich nicht vorhanden.

Fehlercode	Bedeutung
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.
M7E_TYPE	Datentyp ist ungültig.

Siehe auch **M7Read, M7ReadByte, M7ReadDWord, M7ReadWord**

M7ReadByte

Funktion

Byte aus S7-Objekt lesen

Syntax

```
#include <m7api.h>
UBYTE          M7ReadByte(
                 UBYTE ObjType,
                 UWORD Part,
                 UWORD ByteOffset,
                 M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset des gewünschten Bytes
pError	Zeiger auf eine Variable vom Typ M7ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion liest ein Byte aus einem S7-Objekt, das durch die o. g. Parameter beschrieben ist.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert der Wert des adressierten Bytes übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OFFSET	Falscher Offset.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.

Siehe auch

M7Read, M7ReadBit, M7ReadDWord, M7ReadWord

M7ReadDWord

Funktion

Doppelwort aus S7-Objekt lesen

Syntax

```
#include      <m7api.h>
UDWORD          M7ReadDWord(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8)
ByteOffset	Offset des gewünschten Doppelworts
pError	Zeiger auf eine Variable vom Typ ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion liest ein Doppelwort aus einem S7-Objekt, das durch die o. g. Parameter beschrieben ist.

Der Inhalt des Doppelwortes wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert der Wert des adressierten Doppelwortes in *Intel*-Zahlendarstellung übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OFFSET	Falscher Offset.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.

Siehe auch **M7Read, M7ReadBit, M7ReadByte, M7ReadWord**

M7ReadReal

Funktion

Gleitpunktzahl aus S7-Objekt lesen

Syntax

```
#include <m7api.h>
REAL        M7ReadReal(
              UBYTE ObjType,
              UWORD Part,
              UWORD ByteOffset,
              M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8)
ByteOffset	Offset der gewünschten Gleitpunktzahl
pError	Zeiger auf eine Variable vom Typ ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion liest eine Gleitpunktzahl aus einem S7-Objekt, die durch die o. g. Parameter beschrieben ist.

Der Inhalt der Gleitpunktzahl wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert der Wert der adressierten Gleitpunktzahl in *Intel*-Zahlendarstellung übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OFFSET	Falscher Offset.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.

Siehe auch

M7Read, M7ReadBit, M7ReadByte, M7WriteReal, M7ReadDWord

M7ReadWord

Funktion

Wort aus S7-Objekt lesen

Syntax

```
#include <m7api.h>
UWORD        M7ReadWord(
               UBYTE ObjType,
               UWORD Part,
               UWORD ByteOffset,
               M7ERR_CODE_PTR pError);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8)
ByteOffset	Offset des gewünschten Wortes
pError	Zeiger auf eine Variable vom Typ ERR_CODE, in der ein Fehlercode abgelegt werden soll.

Beschreibung

Die Funktion liest ein Wort aus einem S7-Objekt, das durch die u. g. Parameter beschrieben ist.

Der Inhalt des Wortes wird aus der *SIMATIC*- in die *Intel*-Zahlendarstellung konvertiert.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert der Wert des adressierten Wortes in *Intel*-Zahlendarstellung übergeben.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OFFSET	Falscher Offset.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.

Siehe auch

M7Read, M7ReadBit, M7ReadByte, M7ReadDWord

M7RelocateObject

Funktion

S7–Objekt an Objektserver übergeben

Syntax

```
#include <m7api.h>
M7ERRCODE    M7RelocateObject(
               UBYTE ObjType,
               UWORD Part,
               BOOL Copy);
```

Parameter

Parametername	Bedeutung
ObjType	Kennzeichen des S7-Objektes: Die Kennzeichen der möglichen S7–Objekte sind in der Tabelle 2-7 aufgelistet.
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
Copy	Behandlung des neuen Speicherbereichs TRUE Nutzdaten des Objekts werden in den neuen Speicherbereich kopiert. FALSE Nutzdaten des Objekts werden nicht übertragen

Beschreibung

Mit dieser Funktion kann ein S7–Objekt *ObjType*, welches vorher mit der Funktion `M7LocateObject` in die Verantwortung einer Anwendertask gegeben worden ist, wieder an den Objektserver zurückgegeben werden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NOT_LOCATED	Objekt wurde nicht mit `M7LocateObject` an eine Anwendertask übergeben
M7E_NO_MEM	Kein Speicher vorhanden
M7E_OBJ	Objekttyp nicht unterstützt
M7E_PART	Teilbereich nicht vorhanden

Siehe auch

M7LocateObject

M7RemoveObject

Funktion

S7-Objekt aus BACKDIR oder ROMDIR löschen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7RemoveObject(
                  UBYTE ObjType,
                  UWORD Part,
                  BOOL Rom);
```

Parameter

Parametername	Bedeutung	
ObjType	Kennzeichen des S7-Objektes (vgl. Tabelle 2-7).	
Part	Teilbereich (DB-Nummer usw.). Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objekts abhängig (siehe Tabelle 2-8).	
Rom	*Rom* = FALSE:	S7-Objekt wird aus BACKDIR gelöscht.
	Rom = TRUE:	S7-Objekt wird aus ROMDIR gelöscht.

Beschreibung

Die Funktion löscht abhängig vom Parameter *Rom* ein S7-Objekt aus dem Verzeichnis BACKDIR oder ROMDIR.

Rückgabewert

Wenn die Funktion erfolgreich abläuft, wird von ihr als Rückgabewert der Wert des adressierten Wortes in *Intel*-Zahlendarstellung übergeben.

In **pError* übergibt die Funktion Fehleranzeigen:

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten (siehe Fehlercodes).

Fehlercodes

Fehlercode	Bedeutung
M7E_PART	Teilbereich nicht vorhanden
M7E_NODIR	Verzeichnis nicht lesbar oder nicht vorhanden.
M7E_OBJ	Objektyp nicht unterstützt.
M7E_REM_OBJ	Aktion für remanente Objekte nicht erlaubt.

Siehe auch

M7CreateObject, M7DeleteObject, M7GetObjectInfo

M7RequestState

Funktion **Betriebszustandswechsel anfordern**

Syntax

```
#include <m7api.h>
void M7RequestState(
  M7TSFRB_PTR pTSFRB,
  UWORD State,
  UWORD Reason,
  uint MPrio);
```

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger auf den FRB, der für die Bearbeitung der Anmeldung vorgesehen ist.
State	Angabe des neuen Betriebszustandes, in den gewechselt werden soll. Es können folgende Werte angegeben werden: M7STATE_HALT – Betriebszustand HALT M7STATE_RESET – Betriebszustand URLÖSCHEN M7STATE_RUN – Betriebszustand RUN M7STATE_STOP – Betriebszustand STOP M7STATE_CONTINUE – Fortsetzen aus dem Betriebszustand Halt in den vorherigen Zustand (ANLAUF oder RUN)
Reason	Für Anwenderdiagnoseeinträge; liegt im Bereich von 0xA000 bis 0xBFFF.
MPrio	Priorität der verschickten Nachricht (0–255).

Beschreibung

Die Funktion fordert einen Wechsel zu dem im Parameter *State* angegebenen Betriebszustand an.

Wenn der mit dem Parameter *State* angegebene Betriebszustand eingetreten oder ein Fehler aufgetreten ist, wird die aufrufende Task mit einer Nachricht vom Typ M7MSG_REQ_FINISHED darüber informiert.

Nach dem Eintreffen der Nachricht M7MSG_REQ_FINISHED können Sie aus dem referenzierten FRB mit Hilfe des C–Makros `M7GetFRBErrCode` den Erfolg des Aufrufs ermitteln.

`M7GetFRBErrCode` liefert in diesem Fall die folgenden Fehlerkennungen:

Fehlercodes

Fehlercode	Bedeutung
M7E_OST_CPU_IN_STOP	CPU befindet sich in STOP (im Falle FM)
M7E_OST_ILLEGAL_PARAM_CPU	Parameterfehler
M7E_OST_MODE_SW_IN_STOP	Betriebsartenschalter an der CPU/FM ist in Stellung STOP
M7E_OST_WRONG_STATE	Übergang aus aktuellem Zustand nicht möglich oder angeforderter Zustand bereits erreicht.
M7E_OST_NO_SUCH_STATE	Unbekannter Betriebszustand
M7E_PAR	Parameterfehler
M7E_PRIO	Falsche Priorität

Rückgabewert

= M7SUCCESS Wird immer zurückgeliefert.

Ob die angeforderte Betriebszustand auch tatsächlich eingenommen oder abgelehnt wurde oder ein Fehler aufgetreten ist, müssen Sie **nach dem Eintreffen der Nachricht M7MSG_REQ_FINISHED** über den Aufruf `M7GetFRBErrCode` bzw. `M7GetTSType` ermitteln.

Siehe auch

M7GetState, M7LinkState, M7GetFRBErrCode, M7GetTSType

M7RetriggerCycle

Funktion **Zykluszeit nachtriggern**

Syntax

```
#include <m7api.h>
M7ERR_CODE M7RetriggerCycle(void)
```

Beschreibung Die Funktion zieht die Zykluszeit neu auf, d.h. die Überwachung der maximal zulässigen Zykluszeit beginnt wieder von neuem.

Rückgabewert = M7SUCCESS Wird immer zurückgegeben

Siehe auch **M7LinkCycle, M7UnLinkCycle**

M7SendDiagAlarm

Funktion

Diagnosealarm an S7-CPU senden

Syntax

#include <m7api.h>
M7ERR_CODE M7SendDiagAlarm(VOID_PTR *pAlarmInfo***);**

Parameter

Parametername	Bedeutung
pAlarmInfo	Zeiger auf Speicherbereich, in den die Alarmzusatzinformation eingetragen ist. Die Zusatzinformation ist 16 Bytes lang und wird in den Diagnosedatensatz 1 übernommen

Beschreibung

Die Funktion sendet einen Diagnosealarm zur S7/M7–CPU.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_GL_ALARM_DISABLED	Alle Alarme sind gesperrt (durch S7/M7-CPU ausgelöst).
M7E_ODIS	Output-Disable (durch S7/M7-CPU ausgelöst).
M7E_D_ALARM_BUSY	Diagnose-Alarm ist noch nicht durch S7/M7-CPU quittiert.
M7E_ALARM_GEN_DISABLED	Alarmgenerierung der Baugruppe im Datensatz 0 gesperrt.
M7E_D_ALARM_GEN_DISABLED	Diagnosealarmgenerierung der Baugruppe im Datensatz 0 gesperrt.

Siehe auch

M7GetDiagAlarmBusy

M7SendIOAlarm

Funktion **Prozeßalarm an S7-CPU senden**

Syntax

#include <m7api.h>
M7ERR_CODE M7SendIOAlarm(UDWORD *AlarmInfo***);**

Parameter

Parametername	Bedeutung
AlarmInfo	4 Bytes Alarmzusatzinformation

Beschreibung Die Funktion sendet einen Prozeßalarm zur S7/M7–CPU.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_GL_ALARM_DISABLED	Alle Alarme sind gesperrt (durch S7/M7-CPU ausgelöst).
M7E_ODIS	Output-Disable (durch S7/M7-CPU ausgelöst).
M7E_P_ALARM_BUSY	Prozeß-Alarm ist noch nicht durch S7/M7-CPU quittiert.
M7E_ALARM_GEN_DISABLED	Alarmgenerierung der Baugruppe im Datensatz 0 gesperrt.
M7E_P_ALARM_GEN_DISABLED	Prozeßalarmgenerierung der Baugruppe im Datensatz 0 gesperrt.

Siehe auch **M7GetIOAlarmBusy**

M7SetFRBTag

Funktion

Kennzeichen eines FRBs setzen

Syntax

#include <m7api.h>

void **M7SetFRBTag(**
M7FRBHEADER_PTR *pFRB*,
UWORD *Tag*);

Parameter

Parametername	Bedeutung
pFRB	Zeiger auf den FRB, dessen Kennzeichen gesetzt werden soll.
Tag	Kennzeichen des FRB.

Beschreibung

Der Aufruf setzt das Kennzeichen des FRBs auf den mit dem Parameter *Tag* angegebene Wert.

Der Wert ist anwenderspezifisch und kann innerhalb des für UWORD zulässigen Wertebereichs frei vergeben werden.

Das FRB-Kennzeichen kann mit dem Aufruf `M7GetFRBTag` wieder ausgelesen werden.

Der Aufruf ist als C-Makro definiert.

Siehe auch

M7GetFRBErrCode, M7GetFRBTag

M7SetTime

Funktion

Datum und Uhrzeit einstellen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7SetTime(
                  M7TIME_DATE_PTR pDateTime);
```

Parameter

Parametername	Bedeutung
pDateTime	Zeiger auf den Speicherbereich mit Datum-Uhrzeit-Struktur, in dem die aktuellen Werte für Datum und Uhrzeit hinterlegt sind(vgl Kapitel 3).

Beschreibung

Die Funktion stellt die systeminterne Uhrzeit und das systeminterne Datum ein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler

Siehe auch

M7GetTime

M7SetUserLED

Funktion

Anwender-(USR-)LED steuern

Syntax

#include <m7api.h>
M7ERR_CODE M7SetUserLED(
UWORD *Led***,**
UWORD *Mode***);**

Parameter

Parametername	Bedeutung
Led	Nummer der Anwender-LED: M7USERLED1 M7–300 bzw. M7–400 M7USERLED2 nur M7–400
Mode	Ansteuerungsmodus: M7LED_OFF LED ausschalten M7LED_ON LED einschalten, Dauerlicht M7LED_FLASHSLOW LED einschalten, Blinklicht, 0,5 Hz M7LED_FLASHFAST LED einschalten, Blinklicht, 2 Hz

Beschreibung

Die Funktion schaltet die Anwender-LED entsprechend dem Wert von *Mode* an, aus oder blinkend (0,5 oder 2 Hz).

Mit dem Parameter *Led* geben Sie die Nummer der ”Anwender”–LED an. Bei M7–400 kann für *Led* der Wert M7USERLED1 und M7USERLED2 angegeben werden, bei M7–300 ist nur M7USERLED1 erlaubt.

Im zweiten Parameter *Mode* kann über die Konstanten M7LED_ON und M7LED_OFF die angewählte LED an– bzw. ausgeschaltet werden. Außerdem kann im Parameter *Mode* durch Veroderung mit M7LED_FLASHSLOW bzw. M7LED_FLASHFAST die Blinkfrequenz beeinflußt werden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler

M7StoreBit

Funktion

Bits im Prozeßabbild überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreBit(
                UWORD PIType,
                UWORD ByteOffset,
                UBYTE BitOffset,
                BOOL Value);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalbytes
BitOffset	Bit-Offset innerhalb des Signalbytes
Value	Zustand, auf den das adressierte Bit gesetzt werden soll (TRUE oder FALSE).

Beschreibung

Die Funktion adressiert ein Bit im Prozeßabbild, das mit *PIType* bestimmt ist, und setzt es auf den mit *Value* angegebenen Zustand.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType*, *ByteOffset* oder *BitOffset*

Siehe auch

M7StoreByte, M7StoreDWord, M7StoreWord

M7StoreByte

Funktion

Byte im Prozeßabbild überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreByte(
                  UWORD PIType,
                  UWORD ByteOffset,
                  UBYTE Value);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalbytes
Value	Neuer Wert, mit dem das Byte im Prozeßabbild überschrieben werden soll.

Beschreibung

Die Funktion adressiert ein Byte im Prozeßabbild, das mit *PIType* bestimmt wird, und überschreibt es mit dem durch *Value* angegebenen Wert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Falscher *PIType* oder *ByteOffset*

Siehe auch

M7StoreBit, M7StoreDWord, M7StoreWord

M7StoreDirect

Funktion

Daten direkt in Peripheriebereich schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirect(
                  VOID_PTR pBuffer,
                  UWORD SizeOfItem,
                  UWORD Count,
                  M7IO_LOGADDR Addr);
```

Parameter

Parametername	Bedeutung
pBuffer	Zeiger auf den Quellpuffer
SizeOfItem	Größe eines Elementes in Byte. Die folgenden Konstanten sind vordefiniert: M7PBYTE Element hat Datentyp BYTE M7PWORD Element hat Datentyp WORD M7PDWORD Element hat Datentyp DWORD
Count	Anzahl der Elemente
Addr	logische Adresse des ersten Elementes

Beschreibung

Die Funktion überträgt Daten aus einem mit *pBuffer* referenzierten Datenpuffer direkt zur Prozeßperipherie.

Größe, Anzahl und Ziel der übertragenen Daten werden durch die Aufruf-Parameter bestimmt.

Die Funktion führt keine Konvertierung der Zahlendarstellung (*SIMATIC/Intel*) durch.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit zum Datenaustausch

Siehe auch **M7StoreDirectByte, M7StoreDirectDWord, M7StoreDirectWord**

M7StoreDirectByte

Funktion

Byte direkt in Peripherie schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirectByte(
                  M7IO_LOGADDR Addr,
                  UBYTE Value);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse des Peripheriebytes
Value	Neuer Wert, mit dem das Peripheriebyte überschrieben werden soll.

Beschreibung

Die Funktion adressiert ein Byte der Prozeßperipherie und überschreibt es mit dem durch *Value* angegebenen Wert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit zum Datenaustausch

Siehe auch

M7StoreDirect, M7StoreDirectDWord, M7StoreDirectWord

M7StoreDirectDWord

Funktion

Doppelwort direkt in Peripherie schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirectDWord(
                  M7IO_LOGADDR Addr,
                  UDWORD Value);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse des Peripheriedoppelwortes
Value	Neuer Wert, mit dem das Peripheriedoppelwort überschrieben werden soll, in ***SIMATIC***-Zahlendarstellung.

Beschreibung

Die Funktion adressiert ein Doppelwort der Prozeßperipherie und überschreibt es mit dem durch *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit zum Datenaustausch

Siehe auch

M7StoreDirect, M7StoreDirectByte, M7StoreDirectWord

M7StoreDirectWord

Funktion

Wort direkt in Peripherie schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirectWord(
                  M7IO_LOGADDR Addr,
                  UWORD Value);
```

Parameter

Parametername	Bedeutung
Addr	logische Adresse des Peripheriewortes
Value	Neuer Wert, mit dem das Peripheriewort überschrieben werden soll, in ***SIMATIC***-Zahlendarstellung.

Beschreibung

Die Funktion adressiert ein Wort der Prozeßperipherie und überschreibt es mit dem durch *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug
M7E_DP_SLAVE_STATE	Das Gerät ist nicht bereit zum Datenaustausch

Siehe auch

M7StoreDirect, M7StoreDirectByte, M7StoreDirectDWord

M7StoreDWord

Funktion

Doppelwort im Prozeßabbild überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDWord(
                  UWORD PIType,
                  UWORD ByteOffset,
                  UDWORD Value);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signaldoppelwortes
Value	Neuer Wert, mit dem das Doppelwort im Prozeßabbild überschrieben werden soll, in ***SIMATIC***-Zahlendarstellung.

Beschreibung

Die Funktion adressiert ein Doppelwort im Prozeßabbild, das mit *PIType* bestimmt wird, und überschreibt es mit dem durch *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler

Siehe auch

M7StoreBit, M7StoreByte, M7StoreWord

M7StoreISAByte

Funktion

Byte direkt auf ISA–Bus–Peripherie schreiben

Syntax

#include <m7api.h>

M7ERR_CODE M7StoreISAByte(
M7IO_DESC_PTR *pIODesc*,
UBYTE *Value*);

Parameter

Parametername	Bedeutung
pIODesc	Zeiger auf I/O–Deskriptor, der mit `M7InitISADesc` eingerichtet wurde
Value	zu schreibender Wert

Beschreibung

Die Funktion führt als Makro einen direkten Zugriff auf die ISA–Bus–Prozeßperipherie mit Hilfe eines mit `M7InitISADesc` generierten I/O–Deskriptors durch. Der zu schreibende Wert wird durch *val* bestimmt. Die Adresse des Peripheriebereichs wird durch den I/O–Deskriptor der Ausgangssignale bestimmt. Das Prozeßabbild der Ausgänge wird automatisch nachgeführt.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Datenzugriff auf ISA–Bus ist größer (in Bytes) als in `M7InitISADesc` spezifiziert.

Siehe auch

M7StoreISAWord, M7StoreISADWord, M7InitISADesc

M7StoreISADWord

Funktion

Doppelwort direkt auf ISA–Bus–Peripherie schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreISADWord(
                  M7IO_DESC_PTR pIODesc,
                  UDWORD Value);
```

Parameter

Parametername	Bedeutung
pIoDesc	Zeiger auf I/O–Deskriptor, der mit `M7InitISADesc` eingerichtet wurde
Value	zu schreibender Wert

Beschreibung

Die Funktion führt als Makro einen direkten Zugriff auf die ISA–Bus–Prozeßperipherie mit Hilfe eines mit `M7InitISADesc` generierten I/O–Deskriptors durch. Der zu schreibende Wert wird durch *val* bestimmt. Die Adresse des Peripheriebereichs wird durch den I/O–Deskriptor der Ausgangssignale bestimmt. Das Prozeßabbild der Ausgänge wird automatisch nachgeführt.

Der Aufruf führt vor dem Zugriff eine Konvertierung des Werts von der Intel–Darstellung in die SIMATIC–Darstellung durch.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Datenzugriff auf ISA–Bus ist größer (in Bytes) als in `M7InitISADesc` spezifiziert.

Siehe auch

M7StoreISAByte, M7StoreISAWord, M7InitISADesc

M7StoreISAWord

Funktion

Wort direkt auf ISA–Bus–Peripherie schreiben

Syntax

#include <m7api.h>

M7ERR_CODE M7StoreISAWord(
M7IO_DESC_PTR *pIODesc*,
UWORD *Value*);

Parameter

Parametername	Bedeutung
pIoDesc	Zeiger auf I/O–Deskriptor, der mit `M7InitISADesc` eingerichtet wurde
Value	zu schreibender Wert

Beschreibung

Die Funktion führt als Makro einen direkten Zugriff auf die ISA–Bus–Prozeßperipherie mit Hilfe eines mit `M7InitISADesc` generierten I/O–Deskriptors durch. Der zu schreibende Wert wird durch *val* bestimmt. Die Adresse des Peripheriebereichs wird durch den I/O–Deskriptor der Ausgangssignale bestimmt. Das Prozeßabbild der Ausgänge wird automatisch nachgeführt.

Der Aufruf führt vor dem Zugriff eine Konvertierung des Werts von der Intel–Darstellung in die SIMATIC–Darstellung durch.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Datenzugriff auf ISA–Bus ist größer (in Bytes) als in `M7InitISADesc` spezifiziert.

Siehe auch

M7StoreISAByte, M7StoreISADWord, M7InitISADesc

M7StoreObject

Funktion

S7-Objekt in BACKDIR oder ROMDIR abspeichern

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreObject(
                  UBYTE ObjType,
                  UWORD Part,
                  BOOL Rom);
```

Parameter

Parametername	Bedeutung
ObjType	Kennzeichen des S7-Objektes: M7D_DB Datenbaustein M7D_PAR_READ Parameterdatensatz mit Lesen-Attribut M7D_PAR_WRITE Parameterdatensatz mit Schreiben-Attribut
Part	Teilbereich (DB-Nr., Nr. des Parameterdatensatzes,)
Rom	Rom = TRUE: S7-Objekt wird in ROMDIR gespeichert. Rom = FALSE: S7-Objekt wird in BACKDIR gespeichert

Beschreibung

Die Funktion speichert ein S7-Objekt im Verzeichnis, das über die Umgebungsvariablen BACKDIR oder ROMDIR festgelegt ist. In welchem Speicherbereich das S7-Objekt abgespeichert werden soll, bestimmt der Aufruf-Parameter *Rom*.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PART	Teilbereich nicht vorhanden
M7E_NODIR	Verzeichnis nicht lesbar oder nicht vorhanden.
M7E_OBJ	Objekttyp nicht unterstützt.

Siehe auch

M7CreateObject, M7DeleteObject, M7RemoveObject, M7LocateObject

M7StorePIQ

Funktion

Aktualisieren der Ausgangssignale

Syntax

#include <m7api.h>
M7ERR_CODE M7StorePIQ(UWORD *PIQNo***);**

Parameter

Parametername	Bedeutung
PIINo	No des Teilprozeßabbilds bei M7–400. M7–400: 0 gesamtes Prozeßabbild 1 ... 8 Teilprozeßabbild M7–300: 0 gesamtes Prozeßabbild Teilprozeßabbilder werden nicht unterstützt

Beschreibung

Die Funktion aktualisiert die Ausgangssignale mit dem Inhalt des angegeben Teilprozeßabbilds bzw. mit dem gesamten Prozeßabbilds der Ausgänge.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus Kommandofehler
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus Parityfehler
M7E_QVZ	LB Quittungsverzug

Siehe auch

M7LoadPII, M7ClearPI

M7StoreRecord

Funktion

Datensatz zu einer Signalbaugruppe übertragen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreRecord(
                  UBYTE RecordNum,
                  VOID_PTR pBuffer,
                  UBYTE Size,
                  UBYTE PType,
                  M7IO_BASEADDR Addr);
```

Parameter

Parametername	Bedeutung
RecordNum	Datensatznummer Bereich: 0 bis 255
pBuffer	Zeiger auf den Puffer im Arbeitsspeicher, der den **Inhalt** des mit *RecordNum* referenzierten Datensatzes enthält.
Size	Länge des Datensatzes
PType	Kennzeichen des Peripheriebereichs: M7IO_IN Peripheriebereich der Eingänge M7IO_OUT Peripheriebereich der Ausgänge Handelt es sich um eine Mischbaugruppe, ist die Bereichskennung der niedrigsten Adresse anzugeben. Bei gleichen Adressen ist M7IO_IN anzugeben.
Addr	E/A-Basisadresse der Signalbaugruppe

Beschreibung

Die Funktion überträgt einen Datensatz aus einem Puffer, der mit dem Parameter *pBuffer* referenziert wird, zu einer Peripheriebaugruppe.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BSY	Lokalbus ist busy
M7E_CMD	Lokalbus mit Kommandofehler
M7E_COM_ERROR	Fehler bei Abarbeitung des Transferprotokolls
M7E_HWFAULT	AllgemeinerHardware-Fehler
M7E_PAR	Parameterfehler
M7E_PARITY	Lokalbus mit Parityfehler
M7E_QVZ	Lokalbus mit Quittungsverzug

Fehlercode	Bedeutung
M7E_REC_LENGTH	Baugruppe meldet falsche Datensatzlänge
M7E_REC_NUMBER	Baugruppe meldet falsche Datensatznummer
M7E_DPX2_FAULT	Fehler beim DP–Auftrag für den Datensatz–Transfer
M7E_DP_SLAVE_STATE	DP–Slave ist nicht in DATA Zustand
M7E_INVAL_DEV	Baugruppe eines DP–Slaves ist nicht vorhanden

Siehe auch **M7LoadRecord**

M7StoreWord

Funktion

Wort im Prozeßabbild überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreWord(
                  UWORD PIType,
                  UWORD ByteOffset,
                  UWORD Value);
```

Parameter

Parametername	Bedeutung
PIType	Kennzeichen des Prozeßabbildes: M7IO_PII Prozeßabbild der Eingänge M7IO_PIQ Prozeßabbild der Ausgänge
ByteOffset	Offset des Signalwortes
Value	Neuer Wert, mit dem das Wort im Prozeßabbild überschrieben werden soll.

Beschreibung

Die Funktion adressiert ein Wort im Prozeßabbild, das mit *PIType* bestimmt wird, und überschreibt es mit dem durch *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Fehlerhafter *PIType* oder *ByteOffset*.

Siehe auch

M7StoreBit, M7StoreByte, M7StoreDWord

M7SZLRead

Funktion

Systemzustandsliste lesen

Syntax

#include <m7api.h>
M7ERR_CODE M7SZLRead (
UDWORD *flags*,
M7CONNID *ConnID*,
UBYTE_PTR *pBuffer*,
UDWORD *nBufsiz*,
UWORD *SZLID*,
UWORD *Index*,
UDWORD **pnBytes*);

Parameter

Parametername	Bedeutung	
flags	Flags	
	A_FILE	Wenn A_FILE gesetzt ist, gibt *pBuffer* den Namen der Datei an, in der der SZL–Eintrag abgelegt wird. Sonst erfolgt die Ablage im Speicher.
	A_ZERO_FLAG	Dieses Flag kann mit den gewünschten Optionen “geodert” werden. Es muß gesetzt werden, wenn kein anderes Flag verwendet wird.
ConnID	Verbindungsreferenz aus einem `M7KInitiate`–Aufruf.	
pBuffer	Empfangspuffer. Wenn A_FILE gesetzt ist, gibt *pBuffer* den Namen der Datei an, in der der Eintrag abgelegt wird. Sonst erfolgt die Ablage im Speicher.	
nBufsiz	Länge des Empfangspuffers. Wenn A_FILE gesetzt ist ohne Bedeutung.	
SZLID	ID der zu lesenden SZL–Teilliste.	
Index	Index in die Teilliste.	
pnBytes	Zeiger auf die Anzahl gelesener Bytes.	

Beschreibung

Die Funktion `M7SZLRead` liest den mit *SZLID* und *Inde*x spezifizierten Teil der Systemzustandsliste des Zielrechners aus. Der Anwender muß einen ausreichend großen Puffer zur Aufnahme der SZL–Daten angeben. Bei einem Pufferüberlauf kehrt der Aufruf mit einem entsprechenden Fehlercode zurück.

Der Aufbau der SZL für einen M7 ist im Benutzerhandbuch, Systemsoftware für M7–300/400, Installieren und Bedienen, beschrieben

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_NO_MEM	Kein Speicher mehr vorhanden
M7E_KSUB_PARAM	Parameterfehler
M7E_KSUB_NO_SUCH_CONN	Ungültige Verbindung
M7E_KSUB_CONN_CLOSED	Verbindung abgebaut
M7E_KSUB_FILEIO	Fehler in der Dateibearbeitung
M7E_KSUB_REMOTE	Ausführungsfehler beim Server
M7E_KSUB_SDB_WAS_DELETED	Verbindung in STEP7 gelöscht, die Verbindung ist nicht mehr aktiv.

Siehe auch

M7KInitiate, M7WriteDiagnose

M7UnLinkBatteryFailure

Funktion

FRB für Batteriealarm abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkBatteryFailure(
M7BAFFRB_PTR *pBAFFRB***);**

Parameter

Parametername	Bedeutung
pBAFFRB	Zeiger auf den FRB, der abgemeldet werden soll.

Beschreibung

Die Funktion meldet den FRB beim BZ–Server ab.

Der FRB muß zuvor mit `M7LinkBatteryFailure` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FSCFRB nicht in Bearbeitung

Siehe auch

M7LinkBatteryFailure

M7UnLinkCycle

Funktion

FRB beim FZ–Server abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkCycle(M7FSCFRB_PTR *pFSCFRB***);**

Parameter

Parametername	Bedeutung
pFSCFRB	Zeiger auf den FRB, der abgemeldet werden soll.

Beschreibung

Die Funktion meldet den FRB beim FZ–Server ab.

Der FRB muß zuvor mit `M7LinkCycle` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FSC_NO_SUCH_CYCLE	Unbekannter Zustand
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet.

Siehe auch

M7LinkCycle, M7ConfirmCycle

M7UnLinkDataAccess

Funktion

S7-Objekt für Zugriffsinformation über Nachricht abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDataAccess(M7OBJFRB_PTR *pOBJFRB***);**

Parameter

Parametername	Bedeutung
pOBJFRB	Zeiger auf den FRB, der für die Abmeldung vorgesehen ist.

Beschreibung

Die Funktion meldet eine Zugriffsinformation für ein S7-Objekt beim S7–Objekt-Server ab.

Der FRB muß vorher mit der Funktion `M7LinkDataAccess` angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet.

Siehe auch

M7LinkDataAccess, M7GetFlags, M7GetObjType, M7GetPart

M7UnLinkDataAccessCB

Funktion

Aufruf einerCallback–Funktion bei S7–Objektzugriff abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDataAccessCB(M7CBFRB_PTR *pCBFRB***);**

Parameter

Parametername	Bedeutung
pCBFRB	Zeiger auf den FRB, der für die Abmeldung vorgesehen ist.

Beschreibung

Die Funktion meldet eine Callback–Funktion beim Objekt-Server ab.

Die Callback–Funktion muß vorher mit der Funktion `M7LinkDataAccessCB` angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet.

Siehe auch

M7LinkDataAccessCB, M7GetCBFlags, M7GetCBBuffer, M7GetCBDataType, M7GetCBObjType, M7GetCBPart, M7GetCBCount, M7GetCBByteOffset, M7GetCBBitOffset

M7UnLinkDate

Funktion

Uhrzeitgesteuerte Zeitnachricht abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDate(M7TFRB_PTR *pTFRB***);**

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den FRB, mit dem die uhrzeitgesteuerte Zeitnachricht angemeldet wurde.

Beschreibung

Mit dieser Funktion wird der Auftrag für eine uhrzeitgesteuerte Zeitnachricht beim Time–Server abgemeldet.

Der FRB muß zuvor mit `M7LinkDate` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet.

Siehe auch

M7LinkDate

M7UnLinkDiagAlarm

Funktion

Diagnosealarm von der Bearbeitung abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDiagAlarm(
M7DIAGALARM FRB_PTR *pDAFrb***);**

Parameter

Parametername	Bedeutung
pDAFrb	Zeiger auf den FRB, der abgemeldet werden soll.

Beschreibung

Die Funktion meldet den angegebenen FRB für Alarmbearbeitung beim Alarmserver ab. Es werden der aufrufenden Task anschließend keine Diagnosealarme mehr gemeldet.

Der FRB muß zuvor mit `M7LinkDiagAlarm` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet.
M7E_ALARM_PENDING	Auf der betroffenen Baugruppe steht noch ein Diagnosealarm an, der zuerst quittiert werden muß.

Siehe auch

M7LinkDiagAlarm, M7GetDiagAlarmAddr, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo, M7GetDiagAlarmPType, M7ConfirmDiagAlarm

M7UnLinkIOAlarm

Funktion

Prozeßalarm von der Bearbeitung abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkIOAlarm(
M7IOALARM_FRB_PTR *pPAFrb***);**

Parameter

Parametername	Bedeutung
pPAFrb	Zeiger auf den FRB, der abgemeldet werden soll.

Beschreibung

Die Funktion meldet den angegebenen FRB für Alarmbearbeitung beim Alarmserver ab. Es werden der aufrufenden Task anschließend keine Prozeßalarme mehr gemeldet.

Der FRB muß zuvor mit `M7LinkIOAlarm` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet
M7E_ALARM_PENDING	Auf der betroffenen Baugruppe steht noch ein Diagnosealarm an, der zuerst quittiert werden muß.

Siehe auch

M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPTye, M7ConfirmIOAlarm

M7UnLinkOneShotTimer

Funktion

Singuläre Zeitnachricht abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkOneShotTimer(M7TFRB_PTR *pTFRB***);**

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den FRB, mit dem die singuläre Zeitnachricht angemeldet wurde.

Beschreibung

Mit dieser Funktion wird der Auftrag für eine singuläre Zeitnachricht beim Time–Server abgemeldet.

Der FRB muß zuvor mit `M7LinkOneShotTimer` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet

Siehe auch

M7LinkOneShotTimer

M7UnLinkPeriodicTimer

Funktion

Periodische Zeitnachricht abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkPeriodicTimer(M7TFRB_PTR *pTFRB***);**

Parameter

Parametername	Bedeutung
pTFRB	Zeiger auf den FRB, mit dem die periodische Zeitnachricht angemeldet wurde.

Beschreibung

Mit dieser Funktion wird der Auftrag für eine periodische Zeitnachricht beim Time–Server ab.

Der FRB muß zuvor mit `M7LinkPeriodicTimer` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet

Siehe auch

M7LinkPeriodicTimer, M7ConfirmPeriodicTimer, M7GetLostPeriods

M7UnLinkPIError

Funktion **FRB für Prozeßabbildtransferfehler abmelden**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7UnLinkPIError(
                  M7FRBHEADER_PTR pPIEFRB);
```

Parameter

Parametername	Bedeutung
pPIEFRB	Zeiger auf den FRB, der abgemeldet werden soll

Beschreibung

Die Funktion `M7UnLinkPIError` meldet den FRB zur Bearbeitung von Prozeßabbildtransferfehlern beim Freien Zyklus ab. Dieser FRB muß zuvor mit der Funktion `M7LinkPIError` angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	Der FRB war nicht angemeldet

Siehe auch **M7LinkPIError**

M7UnLinkState

Funktion

Nachricht über bestimmten Betriebszustand abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkState(M7TSFRB_PTR *pTSFRB***);**

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger auf den FRB, der quittiert werden soll.

Beschreibung

Die Funktion meldet eine Benachrichtigung über einen bestimmten Betriebszustand beim BZÜ-Server ab.

Der FRB muß zuvor mit `M7LinkState` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet

Siehe auch

M7LinkState, M7GetState, M7RequestState,

M7UnLinkTransition

Funktion

Nachricht über bestimmten Betriebszustandsübergang abmelden

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkTransition(M7TSFRB_PTR *pTSFRB***);**

Parameter

Parametername	Bedeutung
pTSFRB	Zeiger auf den FRB, der quittiert werden soll.

Beschreibung

Die Funktion meldet eine Benachrichtigung über einen bestimmten Betriebszustandsübergang beim BZÜ-Server ab.

Der FRB muß zuvor mit `M7LinkTransition` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet

Siehe auch

M7LinkTransition, M7GetTSReason, M7GetTSType, M7ConfirmTransition

M7UnLinkZSAlarm

Funktion

Nachricht über Ziehen/Stecken–Alarm abmelden

Syntax

#include <m7api.h>

M7ERR_CODE M7UnLinkZSAlarm(
M7ZSALARM_FRB_PTR *pZSFRB***);**

Parameter

Parametername	Bedeutung
pZSFRB	Zeiger auf den FRB, der abgemeldet werden soll.

Beschreibung

Die Funktion meldet eine Benachrichtigung für ein Ziehen/Stecken–Ereignis ab.

Der FRB muß zuvor mit `M7LinkZSAlarm` zur Bearbeitung angemeldet worden sein.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_FRB_NOT_IN_LIST	FRB ist nicht angemeldet

Siehe auch

M7ConfirmZSAlarm, M7LinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7GetZSAlarmAddr

M7Write

Funktion

Anwenderdaten in S7-Datenbereich schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7Write(VOID_PTR pBuffer,
                  UBYTE ObjType,
                  UWORD Part,
                  UBYTE DataType,
                  UWORD Count,
                  UDWORD Addr);
```

Parameter

Parametername	Bedeutung
pBuffer	Zeiger auf den Puffer mit den Anwenderdaten. Die Anwenderdaten müssen in der *SIMATIC*-Darstellung vorliegen!
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
DataType	Datentyp eines Elements (vgl. Tabelle 2-9) Beim Datentyp M7DT_BOOL ist für LENGTH nur 1 zulässig.
Count	Anzahl der zu kopierenden Elemente
Addr	Adresse bzw. Offset innerhalb des Objekts bzw. des Teilbereichs in Bit. Ist *DataType* ≠ BOOL, muß *Addr* ein Vielfaches von 8 Bit sein.

Beschreibung

Die Funktion kopiert eine bestimmte Anzahl Datenelemente aus einem Anwenderdatenbereich in einen S7-Datenbereich.

Der Inhalt des Anwenderdatenbereiches wird nicht aus der *Intel*- in die *SIMATIC*-Zahlendarstellung konvertiert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objekttyp nicht unterstützt
M7E_OFFSET	Falscher Offset
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt

Fehlercode	Bedeutung
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig
M7E_TYPE	Datentyp ist ungültig
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch **M7WriteBit, M7WriteByte, M7WriteDWord, M7WriteWord**

M7WriteBit

Funktion

Bit im S7-Objekt setzen

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteBit(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  UBYTE BitOffset,
                  BOOL Value);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objektes (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset des Bytes, in dem sich das gewünschte Bit befindet
BitOffset	Offset des gewünschten Bits innerhalb des Bytes
Value	Wert, auf den das adressierte Bit gesetzt werden soll.

Beschreibung

Die Funktion adressiert ein Bit in einem S7-Objekt, das durch die o. g. Parameter beschrieben ist, und setzt es auf den mit *Value* angegebenen Zustand.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_BIT_OFFSET	Bitoffset innerhalb des Bytes falsch.
M7E_LENGTH	Länge falsch.
M7E_OBJ	Objekttyp nicht unterstützt.
M7E_OFFSET	Falscher Offset.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler.
M7E_PART	Teilbereich nicht vorhanden.
M7E_PER_BITS	Bitadressierung im Peripheriebereich unzulässig.

Fehlercode	Bedeutung
M7E_TYPE	Datentyp ist ungültig.
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch **M7Write, M7WriteByte, M7WriteDWord, M7WriteWord**

M7WriteByte

Funktion

Byte im S7-Objekt überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteByte(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  UBYTE Value);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset des gewünschten Bytes
Value	Wert, mit dem das adressierte Byte überschrieben werden soll.

Beschreibung

Die Funktion adressiert ein Byte in einem S7-Objekt, das durch die o. g. Parameter beschrieben ist, und überschreibt es mit dem mit *Value* angegebenen Wert.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch

M7Write, M7WriteBit, M7WriteDWord, M7WriteWord

M7WriteDiagnose

Funktion

Eintrag in Diagnosepuffer schreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteDiagnose
                  UBYTE Type,
                  UBYTE Eventnumber,
                  BOOL Direction,
                  UWORD ZI1,
                  UDWORD ZI23,
                  BOOL Send);
```

Parameter

Parametername	Bedeutung
Type	Ereignisklasse
Eventnumber	Nummer des Ereignisses
Direction	Falls TRUE wird eine 1 übertragen (Ereignis kommend)
ZI1	Zusatzinfo 1
ZI23	Zusatzinfo 2 und 3
Send	Falls TRUE, wird Ereignis über K–BUS gesendet

Beschreibung

Der Aufruf legt ein Diagnoseereignis mit angegebener Klasse/Nummer und Zusatzinformationen ab. Der Eintrag erhält den aktuellen Zeitstempel. Wird der Parameter *Send* angegeben, so wird das Diagnoseereignis an angemeldete Kommunikationspartner weiterverteilt.

Im Betriebszustand STOP können keine Einträge in den Diagnosepuffer geschrieben und damit etwaige gültige Einträge überschrieben werden.

Rückgabewert

= M7SUCCESS Die Funktion wurde erfolgreich ausgeführt.

< M7SUCCESS Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_DIAG_NUMBER	Falsche Ereignisklasse (nur 0x0a oder 0x0b erlaubt)
M7E_DIAG_STATE	Falscher Betriebszustand. Einträge im Zustand STOP nicht möglich.
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch

M7SZLRead

M7WriteDWord

Funktion

Doppelwort im S7-Objekt überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteDWord(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  UDWORD Value);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset des gewünschten Doppelwortes
Value	Wert, mit dem das adressierte Doppelwort überschrieben werden soll, in *Intel*-Darstellung.

Beschreibung

Die Funktion adressiert ein Doppelwort in einem S7-Objekt, das durch die o. g. Parameter beschrieben ist, und überschreibt es mit dem mit *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch

M7Write, M7WriteBit, M7WriteByte, M7WriteWord

M7WriteReal

Funktion

Gleitpunktzahl im S7-Objekt überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteReal(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  REAL Value);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset der gewünschten Gleitpunktzahl
Value	Wert, mit dem die adressierte Gleitpunktzahl überschrieben werden soll, in *Intel*-Darstellung.

Beschreibung

Die Funktion adressiert eine Gleitpunktzahl in einem S7-Objekt, die durch die o. g. Parameter beschrieben ist, und überschreibt sie mit dem mit *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch

M7Write, M7WriteBit, M7WriteByte, M7WriteDWord, M7ReadReal

M7WriteWord

Funktion

Wort im S7-Objekt überschreiben

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteWord(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  UWORD Value);
```

Parameter

Parametername	Bedeutung
ObjType	Typkennzeichen des gewünschten S7-Objekts (vgl. Tabelle 2-7).
Part	Teilbereich (DB-Nummer usw.) Die zulässigen Werte für den Teilbereich sind vom Typ des S7-Objektes abhängig (vgl. Tabelle 2-8).
ByteOffset	Offset des gewünschten Wortes
Value	Wert, mit dem das adressierte Wort überschrieben werden soll, in *Intel*-Darstellung.

Beschreibung

Die Funktion adressiert ein Wort in einem S7-Objekt, das durch die o. g. Parameter beschrieben ist, und überschreibt es mit dem mit *Value* angegebenen Wert.

Vor dem Speichern des durch *Value* angegeben Wertes führt die Funktion eine Konvertierung von der *Intel*- in die *SIMATIC*-Zahlendarstellung durch.

Rückgabewert

= M7SUCCESS	Die Funktion wurde erfolgreich ausgeführt.
< M7SUCCESS	Es ist ein Fehler aufgetreten.

Fehlercodes

Fehlercode	Bedeutung
M7E_LENGTH	Länge falsch
M7E_OBJ	Objektyp nicht unterstützt.
M7E_OVS_WRONG_STATE	Aktion im aktuellen Betriebszustand nicht erlaubt
M7E_PAR	Parameterfehler
M7E_PART	Teilbereich nicht vorhanden.
M7E_TYPE	Datentyp nicht unterstützt.
M7E_WRITE_PROTECT	Objekt schreibgeschützt

Siehe auch

M7Write, M7WriteBit, M7WriteByte, M7WriteDWord

RMOS–API

6

Kapitelübersicht

Aufruf	Kurzbeschreibung	Seite
RmGetIntHandler	Interrupt–Handler auslesen	6-42
RmGetMemPoolInfo	Informationen über Speicherpool abfragen	6-43
RmGetName	Katalog nach einem Eintrag durchsuchen	6-44
RmGetSize	Länge eines vergebenen Speicherbereichs ermitteln	6-46
RmGetTaskID	Task-ID ermitteln	6-47
RmGetTaskPriority	Task-Priorität ermitteln	6-48
RmGetTaskState	Task-Zustand ermitteln	6-49
RmIOClose	Unit schließen	6-52
RmIOControl	Steuerfunktionen für ladbare Treiber	6-53
RmIOOpen	Unit öffnen	6-61
RmIORead	Von Unit lesen	6-63
RmIOWrite	Auf Unit schreiben	6-65
RmKillTask	Task beenden	6-67
RmList	Betriebsmittelkatalog-Einträge auflisten	6-69
RmLoadDevice	Treiber laden	6-71
RmMapMemory	Physikalischen Speicher adressieren	6-73
RmMemPoolAlloc	Speicherbereich aus Speicherpool allokieren	6-74
RmPauseTask	Zeitintervall abwarten	6-76
RmQueueStartTask	Task-Start-Anforderung in Warteschlange einreihen. Die Task wird gestartet, sobald sie sich im Zustand RUHEND befindet.	6-77
RmReadMessage	Nachricht aus Message-Queue abholen	6-79
RmReAlloc	Größe eines Speicherbereichs verändern	6-81
RmReceiveMail	Botschaft aus lokaler Mailbox empfangen	6-83
RmReleaseBinSemaphore	Semaphor rücksetzen	6-85
RmResetFlag	Ereignisflag rücksetzen	6-86
RmRestartTask	Task beenden und nach Zeitintervall erneut starten	6-87
RmResumeTask	Mit `RmPauseTask` oder `RmSuspendTask` angehaltene Task fortsetzen	6-89
RmSendMail	Botschaft an eine Mailbox senden	6-90
RmSendMailCancel	Abbruch eines mit `RmSendMailDelayed` gestarteten Botschaft	6-92
RmSendMailDelayed	Botschaft zeitverzögert an eine Mailbox schicken	6-93
RmSendMessage	Nachricht in Message-Queue einhängen	6-95
RmSetFlag	Ereignisflag setzen	6-97
RmSetFlagDelayed	Ereignisflag nach Intervall setzen	6-98

get2ndparm

Funktion

Startparameter EBX der Task auslesen

Syntax

```
#include <rmapi.h>
unsigned int      get2ndparm (void);
```

Beschreibung

`get2ndparm` liefert als Rückgabewert das EBX der Task. Dabei wird das EAX Register überschrieben. Daher können anschließend die Funktionen `getdword` und `getparm` nicht mehr verwendet werden.

Der Aufruf dieser Funktion muß immer der erste Aufruf innerhalb einer Task sein, da der vom Compiler generierte Code unter Umständen die Register EAX bzw. EBX überschreibt.

Siehe auch

getdword, getparm

getdword

Funktion **Startparameter der Task als long auslesen**

Syntax

```
#include <rmapi.h>
unsigned long    getdword (void);
```

Beschreibung

`getdword` liefert als Rückgabewert ein unsigned long, das dem EAX–Register entspricht.

Der Aufruf dieser Funktion muß immer der erste Aufruf innerhalb einer Task sein, da der vom Compiler generierte Code unter Umständen die Register EAX bzw. EBX überschreibt.

getparm

Funktion

Startparameter der Task als Zeiger auslesen

Syntax

```
#include <rmapi.h>
int *                getparm (void);
```

Beschreibung

`getparm` liefert als Rückgabewert einen Zeiger, der dem EAX–Register entspricht.

Der Aufruf dieser Funktion muß immer der erste Aufruf innerhalb einer Task sein, da der vom Compiler generierte Code unter Umständen die Register EAX bzw. EBX überschreibt.

Siehe auch

get2ndparm, getdword

RmActivateTask

Funktion

Task aktivieren

Syntax

#include <rmapi.h>
int RmActivateTask(uint *TaskID***);**

Parameter

Parametername	Bedeutung
TaskID	Task–ID (RM_OWN_TASK=eigene Task)

Beschreibung

Der Aufruf bringt eine andere Task in den Zustand BEREIT, wenn sie sich vorher im Zustand WARTEND befand.

Bei folgenden Zuständen ist der Aufruf `RmActivateTask` unzulässig und wird mit Fehlermeldung beendet:

- Beenden / Löschen durch RmKillTask wurde bereits angemeldet.
- Page Fault aufgrund des Stack–Überlaufs

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *TaskID* wurde übergeben.
RM_INVALID_TASK_STATE	Aufruf bei jetzigem Zustand der Task unzulässig. (Task ist im Zustand RUHEND, RECHNEND, BEREIT oder WARTEND auf E/A–Beendigung)

Siehe auch

RmDeleteTask, RmEndTask, RmKillTask, RmPauseTask

RmAlloc

Funktion

Speicherbereich aus HEAP allokieren

Syntax

```
#include <rmapi.h>
int          RmAlloc (
               ulong TimeOutValue,
               uint Mode,
               ulong Size,
               void **ppMemory)
```

Parameter

Parametername	Bedeutung
TimeOutValue	Maximale Wartezeit bis zur Ausführung RM_CONTINUE — Task fortsetzen, ohne auf Zuteilung des Speichers zu warten. RM_WAIT — Auf Zuteilung des Speichers warten. 0 ... RM_MAXTIME — Zeitintervall in ms. Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden. RM_HOUR(*hour*) — Wartet (*hour*) Stunden RM_MINUTE(*min*) — Wartet (*min*) Minuten RM_SECOND(*sec*) — Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*) — Wartet (*ms*) Millisekunden
Mode	Allokierungsmethode für den Speicher: RM_AUTOFREE — Der Speicher wird automatisch mit `RmFreeAll` freigegeben. Er ist taskspezifisch zugeordnet. RM_NOAUTOFREE — Der Speicher wird nicht automatisch mit `RmFreeAll` freigegeben.
Size	Größe des Speicherbereichs (–1 = größter verfügbarer Block)
ppMemory	Adresse des Zeigers auf einen Speicherbereich.

Beschreibung

Der Aufruf allokiert einen Speicherbereich der Größe *Size* aus dem HEAP. Danach enthält **ppMemory* einen gültigen Zeiger (32 Bit ”Flach”) auf den allokierten Speicherbereich.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_TASK_WAITING	Aufruf mußte auf Zuteilung des Speichers warten.

Fehlercodes

Fehlercode	Bedeutung
RM_GOT_TIMEOUT	In der angegebenen Zeit konnte kein entsprechender Speicherbereich allokiert werden
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_INVALID_SIZE	*Size*=0 oder *Size* größer als HEAP
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar

Siehe auch

RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmGetSize, RmMemPoolAlloc, RmReAlloc, RmGetMemPoolinfo

RmCatalog

Funktion

Ressource in Betriebsmittelverzeichnis (Katalog) eintragen

Syntax

```
#include <rmapi.h>
int          RmCatalog (
               uint Type,
               uint ID,
               ulong IDEx,
               char * pName)
```

Parameter

Parametername	Bedeutung
Type	Betriebsmittel-Typ (siehe *ID).*
ID	Betriebsmittel-ID Die möglichen IDs sind abhängig von *Type*: 0 RM_CATALOG_TASK 0≤id≤2047 1 RM_CATALOG_DEVICE 0≤id≤255 2 RM_CATALOG_POOL 0≤id≤63 3 RM_CATALOG_SEMAPHORE 0≤id≤4095 4 RM_CATALOG_EVENTFLAG 0≤id≤63 5 RM_CATALOG_CNTRL 0≤id≤255 6 RM_CATALOG_LOCALMAILBOX 0≤id≤255 7 RM_CATALOG_MISC 0≤id≤65535 8 RM_CATALOG_USER 0≤id≤65535 10 RM_CATALOG_UNIT 0≤id≤255 11 RM_CATALOG_MESSAGE 0≤id≤2047
IDEx	Erweiterte ID
pName	Zeiger auf C–String unter dem der Eintrag im Betriebsmittel–Katalog eingetragen wird. Der C–String darf maximal 15 Zeichen + \0 umfassen.

Beschreibung

Der Aufruf trägt die angegebenen Parameter in den Betriebsmittel–Katalog ein.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_CATALOG_EXCEEDED	Der Katalog ist voll.
RM_OUT_OF_MEMORY	Es ist ein interner Versuch fehlgeschlagen, Speicherplatz aus dem HEAP zu allokieren.
RM_INVALID_TYPE	Der angegebene Typ ist nicht zulässig. 0≤ *Type* ≤11

Fehlercode	Bedeutung
RM_INVALID_ID	Die angegebene Id ist unzulässig.
RM_INVALID_POINTER	Der Zeiger auf den String ist falsch.
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert.

Siehe auch **RmUnCatalog, RmGetName, RmGetEntry, RmList**

RmCreateBinSemaphore

Funktion

Semaphor erzeugen

Syntax

```
#include <rmapi.h>
int          RmCreateBinSemaphore(
               char *pSemaphoreName,
               uint *pSemaphoreID);
```

Parameter

Parametername	Bedeutung
pSemaphoreName	Zeiger auf einen C-String, der den Namen enthält, mit dem das Semaphor katalogisiert wird. Ist dieser Zeiger = NULL, wird das Semaphor nicht katalogisiert. Der C–String darf maximal 15 Zeichen + \0 umfassen.
pSemaphoreID	Zeiger auf Semaphor-ID

Beschreibung

`RmCreateBinSemaphore` erzeugt ein Semaphor. Die Semaphor-ID wird im angegebenen Speicherbereich zurückgegeben. Die maximale Anzahl Semaphore beträgt 1024.

Das Semaphor wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pSemaphoreName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Rückgabewert

RM_OK — Funktion erfolgreich ausgeführt, ***pSemaphoreID** enthält gültige Semaphore–ID

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_SEMAPHORES	Die Anforderung überschreitet die maximale Anzahl von Semaphoren.
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich, einen String mehrmals zu katalogisieren.

Siehe auch

RmDeleteBinSemaphore, RmReleaseBinSemaphore, RmGetBinSemaphore

RmCreateChildTask

Funktion

Task beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>
int          RmCreateChildTask (
               char * pTaskName,
               ulong TaskStackSize,
               uint Priority,
               rmfarproc TaskEntry,
               uint * pTaskID)
```

Parameter

Parametername	Bedeutung
pTaskName	Zeiger auf einen C-String, der den Namen enthält, mit dem die Task katalogisiert werden soll. Ist dieser Zeiger = NULL wird die TASK nicht katalogisiert. Der C–String darf maximal 15 Zeichen + \0 umfassen.
TaskStackSize	Größe des benötigten Stack in Worten (32 Bit).
Priority	Task-Priorität (0..255) RM_CURPRI gleiche Priorität, wie erzeugende Task
TaskEntry	Einsprungadresse der Task
pTaskID	Zeiger auf Task-ID

Beschreibung

`RmCreateChildTask` gibt dem Betriebssystem Tasks bekannt. Die Task wird vom Zustand NICHT–EXISTENT in den Zustand RUHEND überführt. Die Task wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pTaskName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung

Beim Erzeugen der Child–Task werden von der Parent–Task die Konsole, das aktuelle Arbeitsverzeichnis und das Environment vererbt.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, **pTaskID* enthält die gültige Task–ID.

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_MEMORY	Nicht genügend Speicher zum Anlegen von Stacksegment oder `RmCatalog` hatte nicht genügend Speicher.
RM_INVALID_SIZE	Die Längenangabe von Stack war 0 oder ≥ 1GB
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).

Fehlercode	Bedeutung
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
RM_INVALID_TASK_ENTRY	Die Einsprungaddresse in die Task ist ungültig.
RM_INVALID_POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch **RmCreateTask, RmDeleteTask, RmQueueStartTask, RmStartTask**

RmCreateFlagGrp

Funktion

Flag-Gruppe erzeugen

Syntax

```
#include <rmapi.h>
int          RmCreateFlagGrp(
               char *pFlagGrpName,
               uint *pFlagGrpID);
```

Parameter

Parametername	Bedeutung
pFlagGrpName	Zeiger auf einen C-String, der den Namen enthält, mit dem die Flag–Gruppe katalogisiert wird. Ist dieser Zeiger = NULL, wird die Flag–Gruppe nicht katalogisiert. Der C–String darf maximal 15 Zeichen + \0 umfassen.
pFlagGrpID	Ausgabeparameter, Zeiger auf ID der Flag–Gruppe

Beschreibung

`RmCreateFlagGrp` erzeugt eine Flag-Gruppe. **pFlagGrpID* enthält die gültige ID der Flag–Gruppe.

Die Flag–Gruppe wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pFlagGrpName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_FLAGGROUPS	Die Anforderung überschreitet die maximale Anzahl von Ereignisflags.
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.

Siehe auch

RmSetFlag, RmResetFlag, RmSetFlagDelayed, RmGetFlag, RmDeleteFlagGrp

RmCreateMailbox

Funktion

Mailbox erzeugen

Syntax

```
#include <rmapi.h>
int          RmCreateMailbox(
               char *pMailboxName,
               uint *pMailboxID);
```

Parameter

Parametername	Bedeutung
pMailboxName	Zeiger auf einen C-String, der den Namen enthält, mit dem die Mailbox katalogisiert wird. Ist dieser Zeiger = NULL, wird die Mailbox nicht katalogisiert. Der C–String darf maximal 15 Zeichen + \0 umfassen.
pMailboxID	Zeiger auf ein Mailbox-ID

Beschreibung

`RmCreateMailbox` erzeugt eine Mailbox. ***pMailboxID** enthält gültige Mailbox–ID.

Die Mailbox wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pMailboxName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich, einen String mehrmals zu katalogisieren.
RM_OUT_OF_MAILBOXES	Die Anforderung überschreitet die maximale Anzahl von Mailboxen.
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar

Siehe auch

RmDeleteMailbox, RmReceiveMail, RmSendMail, RmSetMailboxSize

RmCreateMemPool

Funktion

Speicherpool größer 64 Kbyte erzeugen

Syntax

```
#include <rmapi.h>
int          RmCreateMemPool(
               char *pPoolName,
               void *pPoolAddress,
               ulong Size,
               uint *pPoolID);
```

Parameter

Parametername	Bedeutung
pPoolName	Zeiger auf einen C-String, der den Namen enthält, mit dem der Speicherpool katalogisiert wird. Ist dieser Zeiger = NULL, wird der Speicherpool nicht katalogisiert. Der C–String darf maximal 15 Zeichen + \0 umfassen.
pPoolAddress	Zeiger auf Speicherbereich, in dem der Pool angelegt werden soll.
Size	Länge des Speicherbereichs in Bytes
pPoolID	Zeiger auf Pool-ID

Beschreibung

`RmCreateMemPool` definiert einen Speicherpool, der an einer Paragraphgrenze liegt. ***pPoolID** enthält gültige Speicherpool-ID. Die maximale Anzahl Speicherpools ist 8. Die Mindestgröße eines Speicherbereichs beträgt 16 Bytes.

Der Speicherplatz für einen Speicherpool kann mit `RmAlloc` aus dem Heap allokiert werden. Als Adresse für den Speicherpool wird die zurückgegebene Adresse aus `RmAlloc` verwendet.

Die Speicherpools werden beim Initialisieren auf die nächste durch 16 teilbare Basisadresse gelegt. Die Länge wird auf den nächsten durch 16 teilbaren Wert reduziert.

Der Speicherpool wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pPoolName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_OFFSET	Der Offset (*pPoolAddress*) war außerhalb des gültigen Bereichs.
RM_INVALID_SIZE	Eine Größenangabe war ungültig (*Size* < 16).

Fehlercode	Bedeutung
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich, einen String mehrmals zu katalogisieren.
RM_OUT_OF_MEMORYPOOLS	Die Anforderung überschreitet die maximale Anzahl von Speicherpools.

Siehe auch **RmDeleteMemPool, RmFree, RmFreeAll, RmMemPoolAlloc**

RmCreateMessageQueue

Funktion

Message-Queue einrichten

Syntax

```
#include <rmapi.h>
int          RmCreateMessageQueue (
               char * pMessageQueueName,
               uint TaskID)
```

Parameter

Parametername	Bedeutung
pMessageQueueName	Zeiger auf einen C-String, der den Namen enthält, mit dem die Message–Queue katalogisiert wird. Ist dieser Zeiger = NULL, wird die Message–Queue nicht katalogisiert. Der C–String darf maximal 15 Zeichen + \0 umfassen.
TaskID	Zieltask-ID

Beschreibung

Der Aufruf richtet eine Message–Queue für die Task *TaskID* ein.

Die Message–Queue wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pMessageQueueName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Task-ID ungültig
RM_QUEUE_EXIST	Die Message-Queue existiert bereits.
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.

Siehe auch

RmDeleteMessageQueue, RmReadMessage, RmSendMessage, RmSetMessageQueueSize

RmCreateTask

Funktion

Task beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>
int          RmCreateTask (
               char * pTaskName,
               ulong TaskStackSize,
               uint Priority,
               rmfarproc TaskEntry,
               uint * pTaskID)
```

Parameter

Parametername	Bedeutung
pTaskName	Zeiger auf einen C-String, der den Namen enthält, mit dem die Task katalogisiert werden soll. Ist dieser Zeiger = NULL wird die TASK nicht katalogisiert. Der C–String darf max. 15 Zeichen + \0 umfassen.
TaskStackSize	Größe des benötigten Stack in Worten (32 Bit).
Priority	Task-Priorität (0..255)
TaskEntry	Einsprungadresse der Task
pTaskID	Zeiger der Task-ID

Beschreibung

Mit dem Aufruf wird eine Task dem Betriebssystem bekannt gegeben. Die Task wird vom Zustand NICHT-EXISTENT in den Zustand RUHEND überführt. **pTaskID* enthält die gültige Task–ID.

Die Task wird automatisch unter dem angegebenen Namen katalogisiert. Wird für *pTaskName* ein Nullzeiger übergeben, so erfolgt keine Katalogisierung.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_MEMORY	Nicht genügend Speicher zum Anlegen von Stacksegment oder `RmCatalog` hatte nicht genügend Speicher.
RM_INVALID_SIZE	Die Längenangabe von Stack war 0 oder ≥ 1GB
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.

Fehlercode	Bedeutung
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
RM_INVALID_TASK_ENTRY	Die Einsprungadresse in die Task ist ungültig.
RM_INVALID_POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Hinweis

Im Gegensatz zum Aufruf `RmCreateChildTask` findet keine Vererbung der Konsole, des aktuellen Arbeitsverzeichnisses und des Environment statt.

Siehe auch

RmCreateChildTask, RmDeleteTask, RmQueueStartTask, RmStartTask

RmCreateTaskEx

Funktion

Task beim Betriebssystem anmelden

Syntax

```
#include <rmapi.h>
int          RmCreateTaskEx(
               char  *pTaskName,
               RmTCDStruct *pTCD,
               uint *pTaskID);
```

Parametername	Bedeutung
pTaskName	Zeiger auf einen C-String, der den Namen enthält, mit dem die Task katalogisiert werden soll. Ist dieser Zeiger = NULL wird die TASK nicht katalogisiert.
pTCD	Zeiger auf eine Struktur vom Typ **RmTCDStruct**
pTaskID	Zeiger auf die zurückgegebene Task-ID

Beschreibung

`RmCreateTaskEx` verändert den Zustand einer dynamischen Task von NICHT EXISTENT in DORMANT. Die Struktur vom Typ **RmTCDStruct** muß vorab initialisiert sein.Alle nicht benutzten Werte müssen 0 sein. Die Struktur wird nach dem Aufruf nicht mehr benötigt.

Die Task wird danach immer über die zurückgegebene Task-ID angesprochen. Die Task wird automatisch unter dem angegebenen Namen katalogisiert.

In den Task-Flags (TCD.flags) wird festgelegt, ob die Taskmerkmale mit `RM_TFL_CHILD` (siehe `RmCreateChildTask`) an die erzeugte Task vererbt werden sollen.

Das Flag RM_TFL_STK muß immer gesetzt werden. Die Größe des Stacks wird in Worten (32 Bit) in TCD .stck angegeben (siehe Beispiel).

Die Priorität der Task wird in TCD.inpri angegeben (von 0 bis 255).

Die Einsprungadresse der Task wird in TCD.task angegeben.

Hinweis

Das Flag für den Coprozessor (RM_TFL_NPX) wird automatisch gesetzt, sobald die Task auf den Coprozessor zugreift. Daher wird der Aufruf `RmCreateTaskEx` nicht mehr benötigt und exisitert nur noch aus Gründen der Kompatibilität zu früheren Versionen.

Rückgabewert

RM_OK **pTaskID* enthält gültige Task-ID

Fehlercode	Bedeutung
RM_OUT_OF_MEMORY	Nicht genügend Speicher zum Anlegen von Stacksegment oder `RmCatalog` hatte nicht genügend Speicher.
RM_INVALID_SIZE	Die Längenangabe von Stack war 0.
RM_CATALOG_EXCEEDED	Der Katalog ist voll (siehe `RmCatalog`).
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder ist sie Null oder größer als 15.
RM_IS_ALREADY_CATALOGED	Der angegebene String ist bereits katalogisiert. Der String muß eindeutig sein. Daher ist es nicht möglich einen String mehrmals zu katalogisieren.
RM_INVALID_TASK_ENTRY	Die Einsprungaddresse in die Task ist ungültig.
RM_INVALID_PARAMETER	Flag RM_TFL_DS darf für Flat–Aufrufe nicht gesetzt werden.
RM_INVALID_POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Beispiel

Im folgenden Beispiel wird eine Task erzeugt. Mit dem Aufruf `memset` wird die Struktur RmTCDStruct auf 0 initialisiert.

```
main()
 {
 uint TaskID
 RmTCDStruct Tcd;

 memset(&Tcd,0,sizeof(RmTCDStruct));

 Tcd.stck = (void *) 0x400;        /* stacksize */
 Tcd.task = (rmfarproc) entry;   /* taskentry */
 Tcd.inpri=90;                     /* priority  */
 Tcd.flags = RM_TFL_STK | RM_TFL_CHILD;
 Error = RmCreateTaskEx("TaskName",&Tcd,&TaskID);
 ...
}
```

Siehe auch

RmCreateTask, RmCreateChildTask, RmDeleteTask, RmStartTask

RmDeleteBinSemaphore

Funktion

Semaphor löschen

Syntax

#include <rmapi.h>
int **RmDeleteBinSemaphore(uint** *SemaphoreID***);**

Parameter

Parametername	Bedeutung
SemaphoreID	Semaphor-ID

Beschreibung

`RmDeleteBinSemaphore` löscht ein mit `RmCreateBinSemaphore` erzeugtes Semaphor. Der Parameter *SemaphoreID* gibt die ID des Semaphors an, das gelöscht werden soll.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige ID wurde übergeben.
RM_RESOURCE_BUSY	Das Semaphor befindet sich noch im Besitz einer Task.

Siehe auch

RmCreateBinSemaphore, RmReleaseBinSemaphore, RmGetBinSemaphore

RmDeleteFlagGrp

Funktion **Flaggruppe löschen**

Syntax

```
#include <rmapi.h>
int RmDeleteFlagGrp(uint s);
```

Parameter

Parametername	Bedeutung
FlagGrpID	ID der Flag–Gruppe

Beschreibung

`RmDeleteFlagGrp` löscht eine mit `RmCreateFlagGrp` erzeugte globale Flag-Gruppe. Der Parameter *FlagGrpID* gibt die ID der Flag–Gruppe an, die gelöscht werden soll. Das Löschen der lokalen Flaggruppe *FlagGrpID=0* ist nicht erlaubt.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Flaggruppe = 0 oder ID ungültig
RM_RESOURCE_BUSY	Es warten noch Tasks auf das Setzen von Flags dieser Flag–Gruppe (`RmGetFlag`) oder ein `RmSetFlagDelayed` ist noch aktiv.

Siehe auch **RmCreateFlagGrp, RmGetFlag**

RmDeleteMailbox

Funktion

Mailbox löschen

Syntax

#include <rmapi.h>
int **RmDeleteMailbox(uint** *MailboxID***);**

Parameter

Parametername	Bedeutung
MailboxID	Mailbox-ID

Beschreibung

`RmDeleteMailbox` löscht eine mit `RmCreateMailbox` definierte Mailbox. Der Parameter *MailboxID* gibt die ID der Mailbox an, die gelöscht werden soll.

Wird eine Mailbox gelöscht, die von einem Interrupt–Mailboxhandler verwendet wurde, ist der entsprechende Handler ebenfalls zu löschen.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige ID wurde übergeben.
RM_RESOURCE_BUSY	Es warten noch Tasks auf Nachrichten in dieser Mailbox, oder in der Mailbox sind noch Nachrichten enthalten oder ein `RmSendMailDelayed`ist noch aktiv.

Siehe auch

RmCreateMailbox

RmDeleteMemPool

Funktion **Speicherpool löschen**

Syntax

```
#include <rmapi.h>
int RmDeleteMemPool(uint PoolID);
```

Parameter

Parametername	Bedeutung
PoolID	Pool-ID

Beschreibung `RmDeleteMempool` löscht einen mit `RmCreateMemPool` erzeugten Speicherpool. Der Parameter *PoolID* gibt die ID des Speicherpools an, der gelöscht werden soll.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Pool-ID = 0 (entspricht Heap–ID) oder ungültige ID
RM_RESOURCE_BUSY	Speicherbereiche aus diesem Pool sind noch zugeteilt.

Siehe auch **RmCreateMemPool, RmMemPoolAlloc**

RmDeleteMessageQueue

Funktion

Message-Queue entfernen

Syntax

#include <rmapi.h>
int RmDeleteMessageQueue (uint *TaskID)*

Parameter

Parametername	Bedeutung
TaskID	Task-ID

Beschreibung

Der Aufruf `RmDeleteMessageQueue` entfernt die Message–Queue für die Task *TaskID*.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Task-ID ungültig
RM_QUEUE_NOT_EXIST	Die Message-Queue existiert nicht.
RM_RESOURCE_BUSY	In der Message-Queue stehen noch Nachrichten, oder die Task mit der ID *TaskID* wartet noch auf Nachrichten.

Siehe auch

RmCreateMessageQueue, RmSendMessage, RmReadMessage

RmDeleteTask

Funktion

Task löschen

Syntax

#include <rmapi.h>
int RmDeleteTask(uint *TaskID***);**

Parameter

Parametername	Bedeutung
TaskID	Task-ID (RM_OWN_TASK = eigene Task)

Beschreibung

`RmDeleteTask` löscht die Task *TaskID*, sofern sich diese im Zustand RUHEND oder RECHNEND befindet.

War die Task für die CRUN initialisiert, so wird diese Initialisierung gelöscht und offene Dateien geschlossen.

Wird mit `RmDelete` eine Task gelöscht, die von einem Interrupt–Taskhandler aufgerufen wurde, ist der entsprechende Handler ebenfalls zu löschen.

Wurde ein Katalogeintrag erstellt, wird dieser nun wieder gelöscht.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_TASK_NOT_DORMANT	Es wurde versucht, eine Task zu löschen, die sich nicht im Zustand RUHEND befindet.
RM_INVALID_ID	Eine ungültige Task-ID wurde übergeben.

Hinweis

Für Task in anderen Zuständen kann der Aufruf `RmKillTask` verwendet werden.

Siehe auch

RmCreateTask, RmKillTask, x_cr_killtsk

RmDisableScheduler

Funktion

Scheduler sperren

Syntax

```
#include <rmapi.h>
int          RmDisableScheduler(void);
```

Beschreibung

`RmDisableScheduler` deaktiviert den Scheduling–Mechanismus. Bei deaktiviertem Scheduling–Mechanismus ist nur die Task aktiv, die den Aufruf absetzte (auch höher priorisierte Tasks erhalten keine CPU–Zuteilung).

`RmDisableScheduler` ist nicht schachtelbar, d.h. jeder Aufruf führt zum Aussetzen des Scheduling.

Bei ausgesetztem Scheduling sind die Aufrufe `RmDeleteTask` und `RmRestartTask` nicht erlaubt. Ferner sind RMOS–API–Aufrufe zu vermeiden, bei denen eine Task u.U. auf die Abarbeitung einer anderen Task warten muß.

Darunter fallen:
`RmAlloc`, `RmGetEntry`, `RmQueueStartTask`, `RmReceiveMail`, `RmSendMail`, `RmStartTask`, `RmGetFlag` und `RmGetBinSemaphore`.

Ebenso kann bei ausgesetztem Scheduling ein CLI–Job nicht mit <Ctrl>+<C> abgebrochen werden.

Durch ein zu langes Aussetzen des Scheduling kann die Echtzeitfähigkeit des Systems leiden. Dies gilt insbesondere bei Verwendung von `RmRestartTask` und `RmPauseTask`.

Hinweis

Die Schedulingsperre wird automatisch aufgehoben, sobald sich die Task blockiert (z.B. Aufrufe mit Warten–Option, Laufzeitfehler, `printf`).

Rückgabewert

RM_OK Es wird immer RM_OK zurückgeliefert.

Siehe auch

RmEnableScheduler, Scheduler–Beschreibung im Programmierhandbuch

RmEnableScheduler

Funktion **Scheduler freigeben**

Syntax

```
#include <rmapi.h>
int RmEnableScheduler(void);
```

Beschreibung `RmEnableScheduler` aktiviert den Scheduling–Mechanismus, der durch `RmDisableScheduler` deaktiviert wurde.

`RmEnableScheduler` ist nicht schachtelbar, d.h. jeder Aufruf führt zur Wiederaufnahme des Scheduling.

Rückgabewert

RM_OK	Es wird immer RM_OK zurückgeliefert.

Siehe auch **RmDisableScheduler**, Scheduler–Beschreibung im Programmierhandbuch

RmEndTask

Funktion

Task beenden

Syntax

```
#include <rmapi.h>
void        RmEndTask(void);
```

Beschreibung

`RmEndTask` beendet die Task–Ausführung. Die Task wird in den Zustand RUHEND überführt, wenn keine weiteren Task–Start–Anforderungen anstehen.

Hinweis

Dieser Aufruf darf auch auf Tasks angewendet werden, die Funktionen der ANSI–C–Bibliothek benutzen. Anstelle von `RmEndTask` kann auch die C–Bibliotheks–Funktion `exit(x)` verwendet werden.

Rückgabewert

Der Aufruf hat keinen Rückgabewert

Siehe auch

RmQueueStartTask, RmStartTask, RmDeleteTask, Starten, Unterbrechen und Beenden von Tasks

RmFree

Funktion

Einen Speicherbereich freigeben

Syntax

#include <rmapi.h>
int RmFree(void *_pMemory_);

Parameter

Parametername	Bedeutung
pMemory	Zeiger auf den freizugebenden Speicherbereich

Beschreibung

Mit `RmFree` wird ein mit `RmAlloc` oder `RmMemPoolAlloc` von einer Task angeforderter Speicherbereich freigegeben.

Die teilweise Rückgabe eines Speicherbereichs ist nicht möglich.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_MEMORYBLOCK	Speicherbereich war nicht zugeteilt
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Siehe auch

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFreeAll, RmMemPoolAlloc, RmReAlloc

RmFreeAll

Funktion

Alle Speicherbereiche Task-spezifisch freigeben

Syntax

#include <rmapi.h>
int RmFreeAll(uint *TaskID***);**

Parameter

Parametername	Bedeutung
TaskID	ID der Task, deren gesamter angeforderter Speicher freigegeben werden soll (RM_OWN_TASK = eigene Task).

Beschreibung

Mit `RmFreeAll` werden alle mit `RmAlloc` oder `RmMemPoolAlloc` von einer Task angeforderten Speicherbereiche freigegeben. Speicherbereiche, die mit den CRUN Funktionen `malloc`, `calloc` und `realloc` allokiert wurden, werden durch `RmFreeAll` auch freigegeben

Die teilweise Rückgabe von Speicherbereichen ist nicht möglich.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	*TaskID* ungültig
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Hinweis

Es erfolgt keine Fehlermeldung, wenn die Task keinen Speicher angefordert hatte. Speicher, den die Task mit RM_NOAUTOFREE angefordert hat, wird nicht freigegeben.

Siehe auch

calloc, malloc, realloc, RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmMemPoolAlloc, RmReAlloc

RmGetAbsTime

Funktion **Absolute Systemzeit abfragen**

Syntax

```
#include <rmapi.h>
int RmGetAbsTime(RmAbsTimeStruct *pAbsTime);
```

Parameter

Parametername	Bedeutung
pAbsTime	Zeiger auf eine Struktur vom Typ **RmAbsTimeStruct**, in welche die Systemzeit eingetragen wird.

Beschreibung Mit `RmGetAbsTime` wird die absolute Systemzeit in Millisekunden seit dem letzten Neustart in eine Struktur vom Typ **RmAbsTimeStruct** kopiert.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_POINTER	*pAbsTime* unzulässig

Siehe auch **RmAbsTimeStruct**

RmGetBinSemaphore

Funktion

Semaphor testen und setzen

Syntax

```
#include <rmapi.h>
int          RmGetBinSemaphore(
               ulong TimeOutValue,
               uint SemaphoreID);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE	Task fortsetzen und nicht auf Semaphor warten
	RM_WAIT	Auf Semaphor warten
	0 ... RM_MAXTIME	Zeitintervall in ms. Die Task wartet bis sie die Semaphor erhält oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.	
	RM_HOUR(*hour*)	Wartet (*hour*) Stunden
	RM_MINUTE(*min*)	Wartet (*min*) Minuten
	RM_SECOND(*sec*)	Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*)	Wartet (*ms*) Millisekunden
SemaphoreID	Semaphor-ID	

Beschreibung

`RmGetBinSemaphore` testet und setzt ein Semaphor.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_TASK_WAITING	Die Task mußte auf das Semaphor warten.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *SemaphoreID* wurde übergeben.
RM_GOT_TIMEOUT	Der Aufruf wurde nach der eingestellten Timeout-Zeit abgebrochen.
RM_RESOURCE_NOT_AVAILABLE	Das gewünschte Betriebsmittel ist nicht verfügbar.

Hinweis

Die Belegung und Freigabe von Semaphoren ist nicht taskspezifisch.

Siehe auch **RmCreateBinSemaphore, RmDeleteBinSemaphore, RmReleaseBinSemaphore**

RmGetEntry

Funktion

Eintrag in Katalog suchen

Syntax

```
#include <rmapi.h>
int          RmGetEntry (
               ulong TimeOutValue,
               char *pName
               RmEntryStruct *pEntry)
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE	Task fortsetzen, ohne Katalogisierung des Eintrags abzuwarten.
	RM_WAIT	Katalogisierung des Eintrags abwarten.
	0 ... RM_MAXTIME	Zeitintervall in ms. Die Task wartet entweder bis der Eintrag katalogisiert wurde oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.	
	RM_HOUR(*hour*)	Wartet (*hour*) Stunden
	RM_MINUTE(*min*)	Wartet (*min*) Minuten
	RM_SECOND(*sec*)	Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*)	Wartet (*ms*) Millisekunden
pName	Adresse des Namens, der im Katalog gesucht werden soll. Der String kann nach C- oder PLM-Notation definiert sein.	
pEntry	Adresse einer Struktur vom Typ **RmEntryStruct** siehe Kapitel 3.	

Beschreibung

`RmGetEntry` sucht im Betriebsmittelkatalog einen Eintrag.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt.
RM_TASK_WAITING	Aufruf mußte auf Katalogisierung des Eintrags warten.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_STRING	Die Länge des Strings ist unzulässig. Entweder war sie Null oder größer als 15.
RM_IS_NOT_CATALOGED	Der angegebene String ist nicht katalogisiert (nur bei TimeOutValue == RM_CONTINUE)

Fehlercode	Bedeutung
RM_GOT_TIMEOUT	Der angegebene Timeout ist abgelaufen, ohne daß der String katalogisiert wurde.
RM_INVALID_POINTER	Der Zeiger auf den String bzw. auf die Struktur ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch **RmCatalog, RmUncatalog, RmGetName**

RmGetFlag

Funktion

Ereignisflag testen

Syntax

```
#include <rmapi.h>
int          RmGetFlag(
               ulong TimeOutValue,
               uint Type,
               uint FlagGrpID,
               uint TestMask,
               uint *pFlagMask);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE	Task fortsetzen, ohne Setzen des Ereignisflags abzuwarten
	RM_WAIT	Setzen des Ereignisflags abwarten
	0 ... RM_MAXTIME	Zeitintervall in ms. Die Task wartet entweder bis das Ereignisflag gesetzt wird oder das Timeout abgelaufen ist
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.	
	RM_HOUR(*hour*)	Wartet (*hour*) Stunden
	RM_MINUTE(*min*)	Wartet (*min*) Minuten
	RM_SECOND(*sec*)	Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*)	Wartet (*ms*) Millisekunden
Type	RM_TEST_ALL	Testen, ob alle angegebenen Bits gesetzt sind
	RM_TEST_ONE	Testen, ob mindestens ein Bit gesetzt ist
FlagGrpID	ID der Flag-Gruppe. Der Wert 0 kennzeichnet die lokale Flaggruppe.	
TestMask	Maske legt fest, welche Bits getestet werden	
pFlagMask	Pointer auf ein uint, in dem die Werte aller Bits der Flaggruppe zurückgegeben werden.	

Beschreibung

`RmGetFlag` testet bei einer Ereignisflaggruppe, ob entweder alle (RM_TEST_ALL) oder mindestens ein angegebenes Bit (RM_TEST_ONE) gesetzt ist. Bei Angabe einer Wartezeit wird entsprechend auf das Setzen der Bits gewartet. Die Bits einer Flaggruppe werden mit TestMask geUNDet und in *pFlagMask* zurückgegeben.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt

RM_TASK_WAITING	Aufruf mußte auf Setzen des Flag warten.
RM_FLAG_ALREADY_SET	Das Flag war bereits gesetzt.

Fehlercodes

Fehlercode	Bedeutung
RM_TEST_NOT_OK	Ein oder mehrere Flags aus TestMask nicht gesetzt (nur bei RM_CONTINUE)
RM_INVALID_ID	Eine ungültige *FlagGrpID* wurde übergeben.
RM_GOT_TIMEOUT	Der Aufruf wurde nach dem eingestellten Timeout abgebrochen.
RM_INVALID_POINTER	Der Zeiger auf *pFlagMask* ist ungültig bzw. würde eine Schutzverletzung auslösen.

Siehe auch

RmSetFlag, RmSetFlagDelayed, RmResetFlag

RmGetIntHandler

Funktion

Interrupt–Handler auslesen

Syntax

```
#include <rmapi.h>
int          RmGetIntHandler (
               uint IntNum
               rmfarproc *pHandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	Interrupt-Nummer (0–255) IRQx (x=0 bis 63) HW–Interrupt IRQ(n) (n=0bis63) HW–Interrupt Auf PC–Hardware sind die HW–Interrupts auf 0 bis 15.
pHandlerEntry	Einsprungadresse des Interrupt–Handler.

Beschreibung

`RmGetIntHandler` dient dazu, den aktuellen Interrupt–Handler aus der IDT auszulesen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, **pHandlerEntry* enthält die Einsprungadresse des zugehörigen Interrupt–Handlers.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_INTERRUPT_NUMBER	Ungültige Interrupt-Nummer
RM_INVALID_IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
RM_INVALID_POINTER	Ungültiger Zeiger

Siehe auch

RmSetIntDefHandler, RmSetIntISHandler, RmSetIntMailboxHandler, RmSetIntTaskHandler,

RmGetMemPoolInfo

Funktion

Informationen über Speicherpool abfragen

Syntax

```
#include <rmapi.h>
int          RmGetMemPoolInfo(
               uint PoolID,
               RmMemPoolInfoStruct *pInfo)
```

Parameter

Parametername	Bedeutung
PoolID	ID des Memory Pools (RM_HEAP für Heap).
pInfo	Zeiger auf Struktur vom Typ **RmMemPoolInfoStruct**.

Beschreibung

Mit dem Aufruf `RmGetMemPoolInfo` wird die Größe des Pools, des freien Speichers und des größten verfügbaren Blocks (`RmAlloc`(Size=–1))ermittelt. Die Information über den mit *PoolID* angegebenen Pool wird in der Struktur **RmMemPoolInfoStruct** abgelegt, auf die der Zeiger *pInfo* zeigt.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Pool ID ungültig
RM_INVALID_POINTER	*pInfo* ist ein ungültiger Zeiger.

Siehe auch

RmCreateMemPool, RmDeleteMemPool, RmMemPoolAlloc, RmMemPoolInfoStruct

RmGetName

Funktion

Katalog nach einem Eintrag durchsuchen

Syntax

```
#include <rmapi.h>
int          RmGetName (
               uint Type,
               uint ID,
               ulong IDEx,
               char * pName)
```

Parameter

Parametername	Bedeutung
Type	Betriebsmittel-Typ (siehe *ID*)
ID	Betriebsmittel–ID 0 RM_CATALOG_TASK 0≤id≤2047 1 RM_CATALOG_DEVICE 0≤id≤255 2 RM_CATALOG_POOL 0≤id≤63 3 RM_CATALOG_SEMAPHORE 0≤id≤4095 4 RM_CATALOG_EVENTFLAG 0≤id≤63 5 RM_CATALOG_CNTRL 0≤id≤255 6 RM_CATALOG_LOCALMAILBOX 0≤id≤255 7 RM_CATALOG_MISC 0≤id≤65535 8 RM_CATALOG_USER 0≤id≤65535 10 RM_CATALOG_UNIT 0≤id≤255 11 RM_CATALOG_MESSAGE 0≤id≤2047 255 RM_CATALOG_ALL 0≤id≤65535
IDEx	Erweiterte Betriebsmittel-ID (-1 = nicht spezifiziert)
pName	Adresse eines Puffers, in den der String abgelegt werden soll. Die Länge des Puffers muß mindestens 15 Zeichen + \0 umfassen.

Beschreibung

Der Aufruf `RmGetName` durchsucht den Katalog und gibt den zu *Type*, *ID* und *IDEx* gehörenden Namen zurück.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, der Puffer enthält den gültigen Namen des spezifizierten Betriebsmittels.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_TYPE	Der angegebene Type ist nicht zulässig (0≤ *Type* ≤11).
RM_INVALID_ID	Die angegebene ID ist unzulässig.

Fehlercode	Bedeutung
RM_IS_NOT_CATALOGED	Es wurde kein passender Eintrag gefunden.
RM_INVALID_POINTER	Der Zeiger auf den String ist ungültig.

Siehe auch **RmCatalog, RmUncatalog, RmGetEntry**

RmGetSize

Funktion

Länge eines vergebenen Speicherbereichs ermitteln

Syntax

```
#include <rmapi.h>
int          RmGetSize(
               void *pMemory,
               ulong *pSize);
```

Parameter

Parametername	Bedeutung
pMemory	Zeiger auf den Speicherbereich
pSize	Zeiger auf Speicherstelle, in der die Länge des Speicherbereichs zurückgeschrieben wird

Beschreibung

Mit diesem Aufruf kann die Länge eines Speicherbereichs ermittelt werden, der vorher mit `RmAlloc` bzw. `RmMemPoolAlloc` angefordert wurde. **pSize* enthält die Länge des spezifizierten Speicherbereichs.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_MEMORY_BLOCK	Angegebener Speicherbereich war ungültig
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_INVALID_SIZE	Eine Größenangabe war ungültig.

Siehe auch

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmMemPoolAlloc, RmReAlloc

RmGetTaskID

Funktion

Task-ID ermitteln

Syntax

```
#include <rmapi.h>
int          RmGetTaskID(
               uint Tcb,
               uint *pTaskID);
```

Parameter

Parametername	Bedeutung
Tcb	nur RM_OWN_TASK (= eigene Task) erlaubt
pTaskID	Zeiger auf Task-ID

Beschreibung

Mit Mit `RmGetTaskID` kann die Task–ID der eigenen Task ermittelt werden. **pTaskID* enthält die gültige Task–ID der eigenen Task.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_PARAMETER_ERROR	Es wurde ein anderer Parameter als RM_OWN_TASK übergeben

Siehe auch

RmCreateTask, RmCreateTaskEx, RmCreateChildTask

RmGetTaskPriority

Funktion **Task-Priorität ermitteln**

Syntax

```
#include <rmapi.h>
int          RmGetTaskPriority(
               uint TaskID,
               uint *pPriority);
```

Parameter

Parametername	Bedeutung
TaskID	Task-ID (RM_OWN_TASK = eigene Task)
pPriority	Zeiger auf eine Speicherstelle, die die Priorität der Task enthält

Beschreibung `RmGetTaskPriority` liefert die Task-Priorität zurück. **pPriority* enthält die Priorität der spezifizierten Task.

Rückgabewert RM_OK Funktion erfolgreich ausgeführt.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	TaskID ungültig
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Siehe auch **RmGetTaskState**

RmGetTaskState

Funktion **Task-Zustand ermitteln**

Syntax

```
#include <rmapi.h>
int          RmGetTaskState(
               uint TaskID,
               uint *pTaskState);
```

Parameter

Parametername	Bedeutung
TaskID	Task-ID (RM_OWN_TASK = eigene Task)
pTaskState	Zeiger auf eine Speicherstelle, die den Zustand der Task enthält. Mögliche Task–Zustände sind: RM_READY Task im Zustand BEREIT RM_DORMANT Task im Zustand RUHEND RM_ACTIVE Task im Zustand RECHNEND RM_BLOCKED Task im Zustand WARTEND Der Grund ist in den höherwertigen 6 Bits von **pTaskState* aufgeschlüsselt. **pTaskState* nimmt einen der folgende Werte an: RM_STA_EF Auf Ereignisflag wartend RM_STA_SEMA Auf Semaphor wartend RM_STA_LOAD Warten bis Zieltask geladen ist RM_STA_STRT Auf das Starten der Zieltask wartend RM_STA_ENDT Auf das Beenden der Zieltask wartend RM_STA_MSG Auf Empfangen einer Botschaft wartend RM_STA_MSGRCVD Auf das Empfangen einer gesendeten Botschaft wartend RM_STA_POOL Auf Zuweisung von Speicherplatz aus einem Speicherpool wartend RM_STA_HLT Vom DEBUGGER oder durch `RmSuspendTask`angehalten. RM_STA_BREAK Vom DEBUGGER Breakpoint unterbrochen RM_STA_PAUSE Auf den Ablauf eines Zeitintervalls wartend (`RmPauseTask`) RM_STA_WAIT Warten bis Zeitintervall abgelaufen RM_STA_ERR0 Laufzeit-Fehler, Typ 0 (Division by 0 Interrupt) RM_STA_ERR1 Laufzeit-Fehler, Typ 1 (Single Step Interrupt) RM_STA_ERR2 Laufzeit-Fehler, Typ 3 (Breakpoint Interrupt) RM_STA_ERR3 Laufzeit-Fehler, Typ 4 (Overflow Interrupt) RM_STA_ERR4 Laufzeit-Fehler, Typ 5 (Array Bound Interrupt)

Parametername	Bedeutung	
	RM_STA_ERR5	Laufzeit-Fehler, Typ 6 (Unused Opcode)
	RM_STA_ERR6	Laufzeit-Fehler, Typ 7 (Escape Opcode)
	RM_STA_ERR7	Laufzeit-Fehler, Typ 8 (Double Fault)
	RM_STA_ERR8	Laufzeit-Fehler, Typ 9 (NDP Segment Overrun)
	RM_STA_ERR9	Laufzeit-Fehler, Typ 10 (Invalid TSS)
	RM_STA_ERR10	Laufzeit-Fehler, Typ 11 (Segment Not Present)
	RM_STA_ERR11	Laufzeit-Fehler, Typ 12 (Stack Fault)
	RM_STA_ERR12	Laufzeit-Fehler, Typ 13 (General Protection)
	RM_STA_ERR13	Laufzeit-Fehler, Typ 14 (Page Fault)
	RM_STA_ERR14	Laufzeit-Fehler, Typ 16 (Floating Point Error)
	RM_STA_ERR15	Laufzeit-Fehler, Typ 17 (Alignment Check)
	RM_STA_LOOK	Warten auf Catalog-Eintrag
	RM_STA_KEND	Task wird mit `RmKillTask` beendet (ggfs. nach Abschluß der laufenden E/A-Operation)
	RM_STA_KDEL	Task wird mit `RmKillTask` gelöscht (ggfs. nach Abschluß der laufenden E/A-Operation)
	RM_ACTIVE	Die Task befindet sich im Zustand RECHNEND.

Beschreibung

`RmGetTaskState` liefert den Task-Status zurück.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, **pTaskState* enthält den Zustand der spezifizierten Task.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	TaskID ungültig
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Hinweis

Der Aufruf `RmGetTaskState` liefert für eine nicht vorhandene Task den Fehlercode RM_INVALID_ID.

Siehe auch **RmGetTaskPriority**

RmIOClose

Funktion **Unit schließen**

Syntax

#include <rmapi.h>
int RmIOClose(RmIOHandle *Handle***);**

Parameter

Parametername	Bedeutung
Handle	Deskriptor

Beschreibung

`RmIOClose` schließt die mit *Handle* angegebene Unit. *Handle* ist ein Deskriptor, der mit `RmIOOpen` erzeugt wurde. War die Unit für die aufrufende Task reserviert, wird sie (vom Treiber) wieder freigegeben und anstehende Aufträge anderer Tasks werden abgearbeitet.

Der Aufruf `RmIOClose` blockiert nicht, wenn die Unit für eine andere Task reserviert ist.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_BOUND_REACHED	Message–Queue der Unit voll.
RM_EIO_UNIT_RESET	Auftrag durch Steuerfunktion RM_IOCTL_RESET abgebrochen.
RM_INVALID_HANDLE	Deskriptor ist ungültig
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar
RM_QUEUE_NOT_EXIST	Message–Queue der Unit noch nicht eingerichtet.

Siehe auch **RmIOControl, RmIOOpen, RmIORead, RmIOWrite, RmLoadDevice**

RmIOControl

Funktion

Steuerfunktion für ladbare Treiber

Syntax

```
#include <rmapi.h>
int          RmIOControl(
               uint Wait,
               uint FlagMask,
               RmIOHandle Handle,
               uint Control,
               void *pBuffer,
               int *pIOStatus);
```

Parameter

Parametername	Bedeutung
Wait	Angabe, ob die Steuerfunktion mit oder ohne Warten ausgeführt werden soll. RM_CONTINUE: Task fortsetzen, ohne auf Beendigung der Steuerfunktion zu warten RM_WAIT: Beendigung der Steuerfunktion abwarten
FlagMask	Bitmaske, die bei Beendigung der Steuerfunktion in der lokalen Flaggruppe der aufrufenden Task gesetzt werden soll (bei RM_CONTINUE)
Handle	Deskriptor
Control	Funktionscode der Steuerfunktion, siehe unten
pBuffer	Zeiger auf Parameterblock für die Steuerfunktion.
pIOStatus	Zeiger auf `int` mit Fehlerstatus der Operation oder NULL–Pointer

Beschreibung

`RmIOControl` führt eine Steuerfunktion auf der mit *Handle* spezifizierten Unit aus. *Handle* ist ein Deskriptor der mit `RmIOOpen` erzeugt wurde.

Der Parameter *Wait* gibt an, ob die Task auf die Beendigung der Steuerfunktion warten (RM_WAIT) oder fortgesetzt werden soll (RM_CONTINUE).

Mit dem Parameter *FlagMask* kann eine Bitmaske angegeben werden, die bei Aufruf ohne Warten nach Beendigung der Steuerfunktion in der lokalen Flaggruppe (FlagGroupId=0) der aufrufenden Task gesetzt werden soll. Bei Angabe von 0 wird keine Bitmaske gesetzt.

Der Parameter *Control* gibt die auszuführende Steuerfunktion an. Unterstützt die Unit die angegebene Steuerfunktion nicht, so wird die Steuerfunktion mit RM_EIO_INVALID_CONTROL beendet.

Durch *pBuffer* wird ein Parameterblock übergeben, dessen Aufbau von der angegebenen Steuerfunktion abhängt.

In dem `int`, auf das *pIOStatus* zeigt, wird nach Beendigung der Steuerfunktion der Status eingetragen. Bei Aufträgen mit Warten ist dieser Status identisch mit dem Rückgabewert des Aufrufs. Wird der Auftrag ohne Warten ausgeführt, so steht dort, während sich der Auftrag in der Warteschlange befindet, der Wert RM_IO_QUEUED; während der Bearbeitung durch den Treiber steht dort der Wert RM_IO_IN_PROGRESS; nach der Bearbeitung steht dort der Fehlerstatus der Operation. Wird die Rückgabe des Status in *pIOStatus* nicht benötigt (z.B. wegen Aufruf mit RM_WAIT), kann auch ein NULL–Pointer übergeben werden. In diesem Fall wird der Status nur als Rückgabewert der Funktion zurückgemeldet.

Steuerfunktionen

Im folgenden finden Sie die möglichen Steuerfunktionen für den Treiber für serielle Schnittstellen SER8250.DRV und den 3964(R) Treiber 3964.DRV.

Steuerfunktionen für den SER8250.DRV

RM_IOCTL_BUFFER_FLUSH
Hintergrundpuffer leeren. *pBuffer* wird ignoriert.

RM_IOCTL_BUFFER_GETSIZE
Größe des Hintergrundpuffers ermitteln. In das `ulong`, auf das *pBuffer* zeigt, wird die Puffergröße in Anzahl Zeichen geschrieben.

RM_IOCTL_BUFFER_SETSIZE
Größe des Hintergrundpuffers setzen. Dabei werden Daten, die sich bereits im Hintergrundpuffer befinden gelöscht. Im Fehlerfall (z.B. nicht ausreichend freier Speicher) bleibt der Hintergrundpuffer unverändert. *pBuffer* zeigt auf ein `ulong`, in dem die neue Puffergröße in Anzahl Zeichen angegeben wird.

RM_IOCTL_BUFFER_USED
Anzahl der Zeichen ermitteln, die sich im Hintergrundpuffer befinden. Die Anzahl wird in einem `ulong`, auf das *pBuffer* zeigt, hinterlegt

RM_IOCTL_CANCEL
Aktuellen I/O–Auftrag abbrechen. *pBuffer* wird ignoriert

RM_IOCTL_GET_PROPERTIES
Funktionsumfang des Treibers ermitteln. *pBuffer* zeigt auf eine Struktur vom Typ `RmIOCTLPropertiesStruc`t.

RM_IOCTL_GET_VERSION
Version des Treibers ermitteln. *pBuffer* zeigt auf eine Struktur vom Typ `RmIOCTLVersionStruct`.

RM_IOCTL_INIT
Unit mit neuen Werten konfigurieren. *pBuffer* zeigt auf eine Struktur vom Typ `Ser8250InitStruct`, in der die Konfigurationsdaten zu übergeben sind.

RM_IOCTL_INIT_ASCII
Unit mit neuen Werten konfigurieren. Die neuen Konfigurationswerte werden in Form von ASCII–Strings übergeben. *pBuffer* zeigt auf ein Array von Pointern, die auf die Konfigurationsparameter zeigen. Das letzte Element des Arrays muß ein NULL–Pointer sein.
Folgende Konfigurationsparameter sind zulässig:

“IRQ:<irq–number>”
<irq–number> IRQ–Nummer der Schnittstelle (z.B. 4 für COM1).

Diese Angabe ist nur beim ersten RM_IOCTL_INIT_ASCII oder RM_IOCTL_INIT Aufruf für eine Unit zulässig (z.B. DEVICE–Kommando).

“BASE:<i/o–address>”
<i/o–address> I/O Basis–Adresse des 8250 (z.B. 0x3F8 für COM1)
Diese Angabe ist nur beim ersten RM_IOCTL_INIT_ASCII oder RM_IOCTL_INIT Aufruf für eine Unit zulässig (z.B. DEVICE–Kommando).

“MODE:<baud–rate>–<parity>–<data–bit>–<stop–bit>”
Konfiguration der Übertragungsparameter. Dabei bedeuten:

<baud–rate> Baudrate.
Zulässig sind alle Werte, durch die der Wert 115200 ohne Rest teilbar ist.

<parity> Parität. Zulässig sind folgende Angaben:

N	keine Paritäts–Prüfung (none)
E	gerade Parität (even)
O	ungerade Parität (odd)
S	Paritäts–Bit fest auf 0 (space)
M	Paritäts–Bit fest auf 1 (mark)

<data–bit> Anzahl der Daten–Bits. Zulässig sind folgende Angaben: 5, 6, 7, 8

<stop–bit> Anzahl der Stop–Bits. Zulässig sind folgende Angaben:

1	1 Stop–Bit
2	2 Stop–Bit (nicht bei 5 Daten–Bits)
15	1,5 Stop–Bit (nur bei 5 Daten–Bits)

“BUFFER:<size>”
<size> Größe des Hintergrundpuffers

Beispiel**:**

```
char            *parameter[5];
int             status
int             iostatus;
parameter[0] = “IRQ:4”;
parameter[1] = “BASE:0x3F8”;
parameter[2] = “MODE:19200–n–8–1”;
parameter[3] = “BUFFER:512”;
parameter[4] = NULL;
status = RmIOControl(RM_WAIT, 0, handle, RM_IOCTL_INIT_ASCII,
                     parameter, &iostatus);
```

RM_IOCTL_INIT_GET
Aktuelle Konfiguration der Unit einlesen. *pBuffer* zeigt auf einen Puffer mit der Struktur vom Typ `Ser8250InitStruct`.

RM_IOCTL_MODE
Unit mit neuen Werten für Kommunikation (z.B. Baudrate) konfigurieren. *pBuffer* zeigt auf eine Struktur vom Typ `RmIOCTLModeSerialStruct`.

RM_IOCTL_READLEN
Anzahl der Zeichen definieren, nach deren Empfang Leseaufträge automatisch beendet werden. (Nur gültig, wenn durch RM_IOCTL_READSTOP aktiviert.) *pBuffer* muß auf ein `ulong` zeigen, in dem die Anzahl der Zeichen enthalten ist.

RM_IOCTL_READLEN_GET
Einlesen der durch RM_IOCTL_READLEN festgelegten Anzahl von Zeichen. Die Anzahl der Zeichen wird in das `ulong` geschrieben, auf das *pBuffer* zeigt.

RM_IOCTL_READ_MODE
Auswahl der Betriebsart von `RmIORead`. *pBuffer* zeigt auf ein ulong, in dem entweder RM_WAIT oder RM_CONTINUE angegeben wird.

Die Angabe von RM_WAIT bewirkt, daß ein Leseauftrag erst abgeschlossen wird, wenn die Endebedingung (Anzahl der Zeichen, Stopzeichen, Timeout, ...) erreicht wird oder ein Fehler auftritt. Bei Angabe von RM_CONTINUE wird der Leseauftrag mit RM_IO_NO_DATA beendet, wenn sich keine Daten (einschließlich der Endebedingung) im Hintergrundpuffer befinden.

Die Default–Einstellung ist RM_WAIT.

RM_IOCTL_READSTOP
Definieren, welche Endebedingung für Leseaufträgen benutzt werden soll. Das/die Stopzeichen wird/werden nicht in den Puffer des Anwenders geschrieben. Die Endebedingung wird durch das `char`, auf das *pBuffer* zeigt, festgelegt. Folgende Werte sind zulässig:

SER8250_READSTOP_OFF
Keine Endebedingung verwenden

SER8250_READSTOP_CHAR_1
Stopzeichen 1 verwenden

SER8250_READSTOP_CHAR_1_2
Stopzeichen 1 und 2 verwenden, d.h. Abbruch wenn das 1. Stoppzeichen gefolgt vom 2. Stoppzeichen auftritt.

SER8250_READSTOP_LEN
Leseauftrag beenden, wenn die durch RM_IOCTL_READLEN definierte Anzahl von Zeichen eingelesen wurde.

SER8250_READSTOP_CHAR_1 bzw. SER8250_READSTOP_CHAR_1_2 und SER8250_READSTOP_LEN können durch Oder–Verknüpfung miteinander kombiniert werden.

Die Default–Einstellung ist SER8250_READSTOP_OFF.

RM_IOCTL_READSTOP1
Stopzeichen 1 definieren, das Leseaufträge beendet. Nur gültig, wenn durch RM_IOCTL_READSTOP aktiviert. *pBuffer* muß auf ein `char` zeigen, in dem das Stopzeichen enthalten ist.

RM_IOCTL_READSTOP2
Stopzeichen 2 definieren, das Leseaufträge beendet. Nur gültig, wenn durch RM_IOCTL_READSTOP aktiviert. *pBuffer* muß auf ein `char` zeigen, in dem das Stopzeichen enthalten ist.

RM_IOCTL_READSTOP_GET
Einlesen der durch RM_IOCTL_READSTOP aktivierten Endebedingung und der eingetragenen Stopzeichen. *pBufffer* muß auf ein Feld mit 3 char zeigen, in das die aktuellen Werte von RM_IOCTL_READSTOP, RM_IOCTL_READSTOP1 und RM_IOCTL_READSTOP2 eingetragen werden.

RM_IOCTL_READTIMEOUT
Definiert eine Zeitspanne (in ms), die bei Leseaufträgen als maximale Pause zwischen zwei Zeichen betrachtet wird. Ist die Pause länger wird der Leseauftrag beendet. Die Angabe von RM_CONTINUE deaktiviert den Timeout. *pBuffer* muß auf ein `ulong` zeigen, in dem die Zeitspanne angegeben ist.

Die Default–Einstellung ist RM_CONTINUE.

RM_IOCTL_READTIMEOUT_GET
Einlesen der durch RM_IOCTL_READTIMEOUT angegebenen Zeitspanne. Die Zeitspanne wird in das `ulong` geschrieben, auf das *pBuffer* zeigt.

RM_IOCTL_RELEASE
Reservierung der Unit aufheben. I/O–Aufträge, die wegen der Reservierung blockiert waren, werden nun ausgeführt. *pBuffer* wird ignoriert.

RM_IOCTL_RESERVE
Unit für aufrufende Task reservieren. I/O–Aufträge anderer Tasks werden angenommen, aber erst nach Freigabe der Unit ausgeführt. *pBuffer* wird ignoriert.

RM_IOCTL_RESET
Unit zurücksetzen und neu starten. Alle noch nicht abgearbeiteten I/O–Aufträge der Unit werden mit RM_EIO_UNIT_RESET abgewiesen. Die Unit muß anschließend neu initialisiert werden (durch die Steuerfunktionen RM_IOCTL_INIT oder RM_IOCTL_INIT_ASCII). *pBuffer* wird ignoriert.

RM_IOCTL_WRITEDELAY
Definiert eine Zeitspanne (in ms), die bei Schreibaufträgen vom Treiber nach dem Senden des letzten Zeichens als minimale Pause eingehalten wird, bevor der Auftrag beendet wird und somit ein neuer Auftrag bearbeitet werden kann. Die Angabe von RM_CONTINUE deaktiviert den Timeout.

pBuffer muß auf ein `ulong` zeigen, in dem die Zeitspanne angegeben ist.

Die Default–Einstellung ist RM_CONTINUE.

RM_IOCTL_WRITEDELAY_GET
Einlesen der durch durch RM_IOCTL_WRITEDELAY angegebenen Zeitspanne. Die Zeitspanne wird in das `ulong` geschrieben, auf das *pBuffer* zeigt.

RM_IOCTL_WRITESTOP
Definieren welche Endebedingung für Schreibaufträgen benutzt werden sollen. Das/die Stopzeichen wird/werden bei Schreibaufträgen zusätzlich zu den vom Anwender übergebenen Daten gesendet. Die Endebedingung wird durch das `char`, auf das *pBuffer* zeigt, festgelegt. Folgende Werte sind zulässig:

SER8250_WRITESTOP_OFF
Keine Stopzeichen verwenden

SER8250_WRITESTOP_CHAR_1
Stopzeichen 1 verwenden

SER8250_WRITESTOP_CHAR_1_2
Stopzeichen 1 gefolgt von Stopzeichen 2 verwenden.

Die Default–Einstellung ist SER8250_WRITESTOP_OFF.

RM_IOCTL_WRITESTOP1
Stopzeichen 1 für Schreibaufträge definieren. Nur gültig, wenn durch RM_IOCTL_WRITESTOP aktiviert. *pBuffer* muß auf ein `char` zeigen, in dem das Stopzeichen enthalten ist.

RM_IOCTL_WRITESTOP2
Stopzeichen 2 für Schreibaufträge definieren. Nur gültig, wenn durch RM_IOCTL_WRITESTOP aktiviert. *pBuffer* muß auf ein `char` zeigen, in dem das Stopzeichen enthalten ist.

RM_IOCTL_WRITESTOP_GET
Einlesen der durch RM_IOCTL_WRITESTOP aktivierten Endebedingung und der eingetragenen Stopzeichen. *pBufffer* muß auf ein Feld mit 3 char zeigen, in das die aktuellen Werte von RM_IOCTL_WRITESTOP, RM_IOCTL_WRITESTOP1 und RM_IOCTL_WRITESTOP2 eingetragen werden.

Steuerfunktionen für den 3964.DRV

RM_IOCTL_CANCEL
Aktuellen I/O–Auftrag abbrechen. *pBuffer* wird ignoriert

RM_IOCTL_GET_PROPERTIES
Funktionsumfang des Treibers ermitteln. *pBuffer* zeigt auf eine Struktur vom Typ `RmIOCTLPropertiesStruc`t.

RM_IOCTL_GET_VERSION
Version des Treibers ermitteln. *pBuffer* zeigt auf eine Struktur vom Typ `RmIOCTLVersionStruct`.

RM_IOCTL_INIT
Unit mit neuen Werten konfigurieren. *pBuffer* zeigt auf eine Struktur vom Typ `Rm3964InitStruct`, in der die Konfigurationsdaten zu übergeben sind.

RM_IOCTL_INIT_ASCII
Unit mit neuen Werten konfigurieren. Die neuen Konfigurationswerte werden in Form von ASCII–Strings übergeben. *pBuffer* zeigt auf ein String–Array, das die Konfigurationsparameter enthält. Das letzte Element des Arrays muß ein NULL–Pointer sein.

Gültige Angaben sind:
“IRQ:<irq–number>”
<irq–number> IRQ–Nummer der Schnittstelle, über die der Treiber kommunizieren soll (z.B. 4 für COM1). Diese Angabe ist nur beim ersten RM_IOCTL_INIT_ASCII oder RM_IOCTL_INIT Aufruf für eine Unit zulässig (z.B. beim DEVICE–Kommando).

“BASE:<i/o–address>”
<i/o–address> I/O Basis–Adresse der Schnittstelle, über die der Treiber kommunizieren soll (z.B. 0x3F8 für COM1). Diese Angabe ist nur beim ersten RM_IOCTL_INIT_ASCII oder RM_IOCTL_INIT Aufruf für eine Unit zulässig (z.B. beim DEVICE–Kommando).

“MODE:<baud>–<parity>–<data>–<stop>”
Kommunikationsparameter:

<baud–rate> Baudrate.
Zulässig sind alle Werte, durch die der Wert 115200 ohne Rest teilbar ist.

<parity> Parität.
Zulässig sind folgende Angaben:

N	keine Paritäts–Prüfung (none)
E	gerade Parität (even)
O	ungerade Parität (odd)
S	Paritäts–Bit fest auf 0 (space)
M	Paritäts–Bit fest auf 1 (mark)

<data–bit> Anzahl der Daten–Bits. Zulässig sind folgende Angaben: 5, 6, 7, 8

<stop–bit> Anzahl der Stop–Bits.
Zulässig sind folgende Angaben:

1	1 Stop–Bit
2	2 Stop–Bit (nicht bei 5 Daten–Bits)
15	1,5 Stop–Bit (nur bei 5 Daten–Bits)

“PROT:<protokoll>–<master>”
Protokollparameter:

<protokoll>	Protokollauswahl 3964 oder 3964R: 1 für 3964R, 0 für 3964
<master>	Auswahl Master oder Slave: 1 für Master, 0 für Slave

Beispiel:

```
char        *parameter[5];
int         status
int         iostatus;
parameter[0]  = “IRQ:4”
parameter[1]  = “BASE:0x3F8”;
parameter[2]  = ”MODE:19200–n–8–1”;
parameter[3]  = “PROT:1–1”;
parameter[4]  = NULL;
status = RmIOControl( RM_WAIT, 0, handle, RM_IOCTL_INIT_ASCII,
                      parameter, &iostatus);
```

RM_IOCTL_INIT_GET
Aktuelle Konfiguration der Unit einlesen. *pBuffer* zeigt auf einen Puffer mit der Struktur `Rm3964InitStruct`.

RM_IOCTL_MODE
Unit mit neuen Werten für Kommunikation (z.B. Baudrate) konfigurieren. pBuffer zeigt auf die Konfigurationsdaten, die einer Struktur `RmIOCTLModeSerialStruct` zu übergeben sind.

RM_IOCTL_RELEASE
Reservierung der Unit aufheben. I/O–Aufträge, die wegen der Reservierung blockiert waren, werden nun ausgeführt. *pBuffer* wird ignoriert.

RM_IOCTL_RESERVE
Unit für aufrufende Task reservieren. I/O–Aufträge anderer Tasks werden angenommen, aber erst nach Freigabe der Unit ausgeführt. *pBuffer* wird ignoriert.

RM_IOCTL_RESET
Unit zurücksetzen und neu starten. Alle noch nicht abgearbeiteten I/O–Aufträge der Unit werden mit RM_EIO_UNIT_RESET abgewiesen. Die Unit

muß anschließend neu initialisiert werden (durch die Steuerfunktionen RM_IOCTL_INIT oder RM_IOCTL_INIT_ASCII). *pBuffer* wird ignoriert.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_BOUND_REACHED	Message–Queue der Unit voll.
RM_EIO_INVALID_CONTROL	Die angegebene Steuerfunktion wird nicht unterstützt
RM_EIO_UNIT_RESET	Auftrag durch RM_IOCTL_RESET Steuerfunktion abgebrochen.
RM_EIO_×××	Sonstige Fehlercodes der Operation
RM_INVALID_POINTER	Zeiger ungültig
RM_INVALID_TYPE	Ungültiger Wert für *Wait*
RM_INVALID_HANDLE	*Handle* ungültig
RM_IO_QUEUED	Auftrag in Message–Queue eingereiht.
RM_IO_IN_PROGRESS	Auftrag ist in Bearbeitung
RM_OUT_OF_MEMORY	Nicht ausreichend freier Speicher im Heap verfügbar
RM_QUEUE_NOT_EXIST	Message–Queue der Unit noch nicht eingerichtet.

Siehe auch

RmIOClose, RmIOOpen, RmIORead, RmIOWrite, RmLoadDevice

RmIOOpen

Funktion

Unit öffnen

Syntax

```
#include <rmapi.h>
int          RmIOOpen(
               const char * pUnitName,
               uint Mode,
               RmIOHandle * pHandle);
```

Parameter

Parametername	Bedeutung
pUnitName	Name der Unit im RMOS–Betriebsmittelkatalog
Mode	Modus zum Öffnen der Unit RM_IO_READ Unit für Lesezugriffe öffnen RM_IO_WRITE Unit für Schreibzugriffe öffnen RM_IO_RESERVE Unit für Task reservieren
pHandle	Zeiger auf Variable, in der ein Deskriptor zum Ansprechen der Unit abgelegt wird.

Beschreibung

`RmIOOpen` öffnet die mit *pUnitName* angegebene Unit für eine Bearbeitung mit den Aufrufen `RmIORead`, `RmIOWrite` und `RmIOControl`. `RmIOOpen` legt den Deskriptor der geöffneten Unit im durch *pHandle* bezeichneten Speicher ab.

Der Parameter *Mode* gibt an, welche Zugriffe auf die Unit durchgeführt werden sollen. Dabei bezeichnet RM_IO_READ Lesezugriffe und RM_IO_WRITE Schreibzugriffe.

Die zusätzliche Angabe von RM_IO_RESERVE bewirkt, daß nur Aufträge der aufrufenden Task bearbeitet werden. Aufträge anderer Tasks werden angenommen, jedoch erst nach Freigabe durch die Task (`RmIOControl` mit RM_IOCTL_RELEASE) bzw. Schließen der Unit mit `RmIOClose` ausgeführt.

Die Werte können bei Bedarf durch OR–Verknüpfung miteinander kombiniert werden (z.B. RM_IO_READ | RM_IO_WRITE | RM_IO_RESERVE, die Unit wird für Lese– und Schreibzugriffe nur der aufrufenden Task geöffnet.).

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_BOUND_REACHED	Message–Queue der Unit voll.
RM_EIO_UNIT_RESERVED	Unit ist bereits reserviert (`RmIOOpen` mit RM_IO_RESERVE bzw. `RmIOControl` mit RM_IOCTL_RESERVE).

Fehlercode	Bedeutung
RM_EIO_UNIT_RESET	Auftrag durch Steuerfunktion RM_IOCTL_RESET abgebrochen.
RM_INVALID_POINTER	Zeiger ungültig
RM_INVALID_TYPE	Ungültiger Wert für *Mode*
RM_INVALID_UNIT	*UnitName* ist keine Unit eines ladbaren Treibers.
RM_IS_NOT_CATALOGED	Unit ist nicht mit angegebenem Namen katalogisiert
RM_OUT_OF_MEMORY	Nicht ausreichend freier Speicher im Heap verfügbar
RM_QUEUE_NOT_EXIST	Message–Queue der Unit noch nicht eingerichtet.

Siehe auch **RmIOClose, RmIOControl, RmIORead, RmIOWrite, RmLoadDevice**

RmIORead

Funktion **Von Unit lesen**

Syntax

```
#include <rmapi.h>
int          RmIORead(
               uint Wait,
               uint FlagMask,
               RmIOHandle Handle,
               ulong Length,
               void *pBuffer,
               ulong BlockAddress,
               ulong *pIOCount,
               int *pIOStatus);
```

Parameter

Parametername	Bedeutung
Wait	Angabe, ob der Auftrag mit oder ohne Warten ausgeführt werden soll. RM_CONTINUE: Task fortsetzen, ohne auf Beendigung des Leseauftrags zu warten RM_WAIT: Beendigung des Leseauftrags abwarten
FlagMask	Bitmaske, die bei Beendigung des Auftrags in der lokalen Flaggruppe der aufrufenden Task gesetzt werden soll (bei RM_CONTINUE)
Handle	Deskriptor
Length	Länge des Speicherbereichs in Byte/Blöcken (numerisch)
pBuffer	Zeiger auf den Speicherbereich
BlockAddress	Adresse des ersten Blocks bei blockorientierten Treibern
pIOCount	Zeiger auf ein `ulong`, für die Anzahl gelesener Byte/Blöcke (gültig erst nach Abschluß des Leseauftrags)
pIOStatus	Zeiger auf `int` für Fehlerstatus der Operation oder NULL–Pointer

Beschreibung

Der Aufruf `RmIORead` liest *Length* Bytes (bei zeichenorientierten Treibern) bzw. Blöcke (bei blockorientierten Treibern) von der mit *Handle* spezifizierten Unit in den mit *pBuffer* angegeben Speicherbereich ein. *Handle* ist ein Deskriptor, der mit `RmIOOpen` erzeugt wurde.

Bei blockorientierten Treibern wird zusätzlich in *BlockAddress* die Adresse des ersten zu lesenden Blocks übergeben. Bei zeichenorientierten Treibern (SER8250.DRV, 3964.DRV) wird *BlockAddress* ignoriert.

Wait gibt an, ob die Task auf Beendigung des Leseauftrags warten (RM_WAIT) oder ohne Warten fortgesetzt werden soll (RM_CONTINUE).

Mit dem Parameter *FlagMask* kann eine Bitmaske angegeben werden, die bei Aufruf ohne Warten nach Beendigung des Auftrags in der lokalen Flaggruppe (FlagGroupId=0) der aufrufenden Task gesetzt werden soll. Bei Angabe von 0 wird keine Bitmaske gesetzt.

In dem `ulong`, auf das *pIOCount* zeigt, wird nach Beendigung des Leseauftrags die Anzahl der übertragenen Bytes/Blöcke hinterlegt.

In dem `int`, auf das *pIOStatus* zeigt, wird nach Beendigung des Leseauftrags der Status eingetragen. Bei Aufträgen mit Warten ist dieser Status identisch mit dem Rückgabewert des Aufrufs. Wird der Auftrag ohne Warten ausgeführt, so steht dort, während sich der Auftrag in der Warteschlange befindet, der Wert RM_IO_QUEUED; während der Bearbeitung durch den Treiber steht dort der Wert RM_IO_IN_PROGRESS; nach der Bearbeitung steht dort der Fehlerstatus der Operation. Wird die Rückgabe des Status in *pIOStatus* nicht benötigt (z.B. wegen Aufruf mit RM_WAIT), kann auch ein NULL–Pointer übergeben werden. In diesem Fall wird der Status nur als Rückgabewert der Funktion zurückgemeldet.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	**Bedeutung**
RM_BOUND_REACHED	Message–Queue der Unit voll.
RM_EIO_INVALID_ACCESS	Deskriptor ist nicht für Read geöffnet
RM_EIO_UNIT_RESET	Auftrag durch Steuerfunktion RM_IOCTL_RESET abgebrochen.
RM_INVALID_HANDLE	Deskriptor ist ungültig
RM_INVALID_POINTER	Ungültiger Zeiger
RM_INVALID_TYPE	Der Wert für *Wait* ist ungültig
RM_IO_IN_PROGRESS	Auftrag ist in Bearbeitung
RM_IO_QUEUED	Auftrag in Warteschlangeeingereiht
RM_OUT_OF_MEMORY	Nicht ausreichend freier Speicher im Heap verfügbar
RM_QUEUE_NOT_EXIST	Message–Queue der Unit noch nicht eingerichtet.

Siehe auch

RmIOClose, RmIOControl, RmIOOpen, RmIOWrite, RmLoadDevice

RmIOWrite

Funktion **Auf Unit schreiben**

Syntax

```
#include <rmapi.h>
int RmIOWrite(
  uint Wait,
  uint FlagMask,
  RmIOHandle Handle,
  ulong Length,
  void *pBuffer,
  ulong BlockAddress,
  ulong *pIOCount,
  int *pIOStatus);
```

Parameter

Parametername	Bedeutung
Wait	Angabe, ob der Auftrag mit oder ohne Warten ausgeführt werden soll. RM_CONTINUE: Task fortsetzen, ohne auf Beendigung des Schreibauftrags zu warten RM_WAIT: Beendigung des Schreibauftrags abwarten
FlagMask	Bitmaske, die bei Beendigung des Auftrags in der lokalen Flaggruppe der aufrufenden Task gesetzt werden soll (bei RM_CONTINUE)
Handle	Deskriptor
Length	Länge des Speicherbereichs in Byte/Blöcken (numerisch)
pBuffer	Zeiger auf den Speicherbereich
BlockAddress	Adresse des ersten Blocks bei blockorientierten Treibern
pIOCount	Zeiger auf ein `ulong`, für die Anzahl gelesenen Byte/ Blöcke (gültig erst nach Abschluß des Schreibauftrags)
pIOStatus	Zeiger auf `int` für Fehlerstatus der Operation oder NULL–Pointer

Beschreibung

Der Aufruf `RmIOWrite` schreibt *Length* Bytes (bei zeichenorientierten Treibern) bzw. Blöcke (bei blockorientierten Treibern) von dem durch *pBuffer* angegeben Speicherbereich auf die mit *Handle* spezifizierten Unit. *Handle* ist ein Deskriptor der mit `RmIOOpen` erzeugt wurde.

Bei blockorientierten Treibern wird zusätzlich in *BlockAddress* die Adresse des ersten zu schreibenden Blocks übergeben. Bei zeichenorientierten Treibern (SER8250.DRV, 3964.DRV) wird *BlockAddress* ignoriert.

Wait gibt an, ob die Task auf Beendigung des Schreibauftrags warten (RM_WAIT) oder ohne Warten fortgesetzt werden soll (RM_CONTINUE).

Mit dem Parameter *FlagMask* kann eine Bitmaske angegeben werden, die bei Aufruf ohne Warten nach Beendigung des Auftrags in der lokalen Flaggruppe (FlagGroupId=0) der aufrufenden Task gesetzt werden soll. Bei Angabe von 0 wird keine Bitmaske gesetzt.

In dem `ulong`, auf das *pIOCount* zeigt, wird nach Beendigung des Schreibauftrags die Anzahl der übertragenen Bytes/Blöcke hinterlegt.

In dem `int`, auf das *pIOStatus* zeigt, wird nach Beendigung des Schreibauftrags der Status eingetragen. Bei Aufträgen mit Warten ist dieser Status identisch mit dem Rückgabewert des Aufrufs. Wird der Auftrag ohne Warten ausgeführt, so steht dort, während sich der Auftrag in der Warteschlange befindet, der Wert RM_IO_QUEUED; während der Bearbeitung durch den Treiber steht dort der Wert RM_IO_IN_PROGRESS; nach der Bearbeitung steht dort der Fehlerstatus der Operation. Wird die Rückgabe des Status in *pIOStatus* nicht benötigt (z.B. wegen Aufruf mit RM_WAIT), kann auch ein NULL–Pointer übergeben werden. In diesem Fall wird der Status nur als Return–Wert der Funktion zurückgemeldet.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_BOUND_REACHED	Message–Queue der Unit voll.
RM_EIO_INVALID_ACCESS	Deskriptor ist nicht für Write geöffnet
RM_EIO_UNIT_RESET	Auftrag durch Steuerfunktion RM_IOCTL_RESET abgebrochen.
RM_INVALID_HANDLE	Deskriptor ist ungültig
RM_INVALID_POINTER	Ungültiger Zeiger
RM_INVALID_TYPE	Der Wert für *Wait* ist ungültig
RM_IO_IN_PROGRESS	Auftrag ist in Bearbeitung
RM_IO_QUEUED	Auftrag in Warteschlange eingereiht
RM_OUT_OF_MEMORY	Nicht ausreichend freier Speicher im Heap verfügbar
RM_QUEUE_NOT_EXIST	Message–Queue der Unit noch nicht eingerichtet.

Siehe auch

RmIOClose, RmIOControl, RmIOOpen, RmIORead, RmLoadDevice

RmKillTask

Funktion **Task beenden**

Syntax

```
#include <rmapi.h>
int          RmKillTask(
               uint Mode,
               uint TaskID);
```

Parameter

Parametername	Bedeutung	
Mode	Gewünschter Task-Zustand:	
	RM_TASK_END	Task in den Zustand RUHEND versetzen (entspricht `RmEndTask`)
	RM_TASK_DELETE	Task löschen (entspricht `RmDeleteTask`)
TaskID	ID der zu löschenden Task (RM_OWN_TASK = eigene Task)	

Beschreibung

Der Aufruf bringt eine beliebige Task (auch die aufrufende Task) in den Zustand RUHEND oder NICHT-EXISTENT und zwar unabhängig in welchem Zustand sie sich vorher befand.

Besonderheiten treten auf, wenn sich die Zieltask im Zustand WARTEND befindet.

Bei folgenden Zuständen ist `RmKillTask` unzulässig und wird mit Fehlermeldung beendet:

- Beenden/Löschen durch `RmKillTask` bereits angemeldet (zweimaliges Anwenden des Aufrufs `RmKillTask` auf dieselbe Task)
- Page Fault aufgrund des Stack–Überlaufs

Im folgenden Fall wird die Task nicht sofort in den Zustand RUHEND oder NICHT-EXISTENT überführt, sondern lediglich angemeldet:

Warten auf Beendigung eines E/A-Auftrags:
Die betreffende Task bleibt weiterhin im Zustand WARTEND. Erst wenn der E/A-Auftrag beendet wurde, tritt die Zustandsänderung ein. Es kann also sein, daß die Task nach dem Aufruf weiterhin in einem passiven (geblockten) Zustand sichtbar bleibt. Die Task befindet sich dann in den Blockzuständen RM_STA_KEND bzw. RM_STA_KDEL.

Option RM_TASK_DELETE

Alle Startanforderungen in der Warteschlange werden entfernt. Falls die Zieltask mit der Koordination ”Warten bis bereit” oder ”Warten bis Beendigung”

gestartet wurde, werden alle betreffenden Tasks, die `RmStartTask bzw. RmQueueStartTask` abgesetzt haben, auf das vorzeitige Beenden/Löschen der Zieltask hingewiesen.

Option RM_TASK_END

Alle Startanforderungen in der Warteschlange bleiben erhalten. Der weitere Ablauf der aufrufenden Tasks ist der gleiche, als hätte die Zieltask `RmEndTask` abgesetzt. Falls die Zieltask mit der Koordination "Warten bis bereit" oder "Warten bis Beendigung" gestartet wurde, wird die Task, die `RmStartTask bzw. RmQueueStartTask` abgesetzt hat, auf das vorzeitige Beenden/Löschen der Zieltask hingewiesen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_TYPE	Ein ungültiger Parameter (*Mode*) wurde übergeben.
RM_INVALID_ID	Eine ungültige *TaskID* wurde übergeben.
RM_INVALID_TASK_STATE	Aufruf bei jetzigem Zustand der Task unzulässig

Hinweis

Betriebsmittel, wie z.B. Speicherpools, Mailboxen oder Semaphore, die noch im Besitz der Task sind, werden nach dem Versetzen in den Zustand RUHEND oder dem Löschen der Task nicht automatisch freigegeben. Diese Betriebsmittel müssen – falls möglich – von einer anderen Task freigegeben werden, ansonsten sind sie für den weiteren Betrieb verloren.

Siehe auch

RmDeleteTask, x_cr_killtsk

RmList

Funktion **Betriebsmittelkatalog-Einträge auflisten**

Syntax

```
#include <rmapi.h>
int          RmList (
               uint Type,
               uint Count,
               uint *pIndex,
               uint *pNumEntries
               RmEntryStruct *pEntry)
```

Parameter

Parametername	Bedeutung
Type	Betriebsmittel-Art(siehe `RmGetName`)
Count	Anzahl der Betriebsmittel–Einträge, die in einem Aufruf ausgelesen werden sollen. In *NumEntries* wird zurückgemeldet, wie viele Einträge gefunden und in *pEntry* abgelegt wurden. Ist *Count* >1, muß *pEntry* auf ein Feld aus *NumEntries* Elementen der Struktur **RmEntryStruct** zeigen.
pIndex	Dieser Parameter wird sowohl als Ein– und als Ausgangsparameter verwendet. **Eingangsparameter:** **pIndex* spezifiziert den Wert, ab dem die Betriebsmittel–Einträge ausgelesen werden sollen. Beim ersten Aufruf muß **pIndex* 0 sein. Sind weitere Aufrufe erforderlich, darf **pIndex* nicht verändert werden. **Ausgangsparameter:** Der Aufruf liefert in **pIndex* den Index des nächsten noch nicht ausgelesenen Eintrags. Dieser Index dient nur für systeminterne Zwecke und kann vom Anwender nicht ausgewertet werden.
pNumEntries	Anzahl der gefundenen Einträge.
pEntry	Zeiger auf eine Struktur bzw. auf ein Feld (abhängig von *Count)* von Strukturen des Typs **RmEntryStruct** siehe Kapitel 3.

Beschreibung

Der Aufruf `RmList` liest eine Anzahl von Einträgen des Katalogs aus und legt sie im angegebenen Puffer, dessen Anfangsadresse durch *pEntry* spezifiziert wird ab.

Über den Parameter **pIndex* kann beim Aufruf der erste auszulesende Eintrag angegeben werden (Start der Liste =0). Bei der Rückkehr des Aufrufs enthält **pIndex* einen Verweis für den nächsten noch nicht ausgelesenen Eintrag.
**pIndex* darf nicht verändert werden.

Das Ende des Katalogverzeichnisses ist dann erreicht, wenn die Anzahl der tatsächlich ausgelesenen Einträge **pNumEntries* kleiner ist als die angeforderte Anzahl *Count*.

Das Auslesen kann über *Type* auf eine bestimmte Betriebsmitteltyp eingeschränkt werden.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, der Puffer enthält gültige Einträge.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_TYPE	Der angegebene Type ist nicht zulässig (0≤ *Type* ≤11)
RM_INVALID_POINTER	Der Zeiger auf den String ist falsch, bzw. würde eine Schutzverletzung auslösen.

Siehe auch

RmCatalog, RmGetEntry, RmGetName, RmUncatalog

RmLoadDevice

Funktion **Treiber laden**

Syntax

```
#include <rmapi.h>
int          RmLoadDevice(
               const char *pDeviceName,
               const char *pArguments);
```

Parameter

Parametername	Bedeutung
pDeviceName	Zeiger auf Namen des Treibers
pArguments	Zeiger auf Argumente (durch Leerzeichen getrennt)

Beschreibung

`RmLoadDevice` lädt und startet den durch *pDeviceName* angegebenen Treiber bzw. erzeugt für den durch *pDeviceName* angegebenen Treiber eine neue Unit, falls *pDeviceName* im RMOS Betriebsmittelkatalog als ladbarer Treiber eingetragen ist (SER8250, 3964).

Beim erstmaligen Laden muß der Treiber mit absoluten Pfad angegeben werden. Dabei ist der Name des Treibers anzugeben (SER8250.DRV, 3964.DRV). Nach dem Laden erfolgt die Katalogisierung des Treibers.

Bei weiteren Aufrufen muß der im Betriebsmittelkatalog eingetragene Name verwendet werden (SER8250, 3964).

pArguments gibt die Argumente zur Initialisierung des Treibers bzw. der Unit an. Die einzelnen Argumente werden durch Leerzeichen getrennt. Siehe `RmIOControl` mit der Steuerfunktion RM_IOCTL_INIT_ASCII für detailliertere Angaben.

Beim Linken der Applikation wird die Bibliothek RMFCRIFB.LIB benötigt.

Rückgabewert RM_OK Funktion erfolgreich

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_DEVICE	Ungültige Angabe für *pDeviceName* (z.B. Katalogeintrag ist kein ladbarer Treiber oder Treiber nicht gefunden).
RM_OUT_OF_MEMORY	Kein freier Speicherplatz verfügbar.
RM_EIO_INIT_FAILED	Der Treiber hat sich aufgrund eines Fehlers beendet und wurde aus dem System entfernt.

Beispiel

Laden des Treibers SER8250 ohne Argumente:
RmLoadDevice(”\\M7RMOS32\\ser8250.drv”, NULL);

Laden des Treibers 3964 mit Unit 3964_COM1 und Initialisierungswerten:
RmLoadDevice(”\\M7RMOS32\\3964.drv”,
“3964_COM1 IRQ:4 BASE:0x3F8 MODE:19200–N–8–1 PROT:1–1”);

Unit COM2 für den bereits geladenen Treiber SER8250 mit Initialisierungswerten erzeugen:
RmLoadDevice(”SER8250”,
“COM2 IRQ:3 BASE:0x2F8 MODE:19200–N–8–1”);

Siehe auch

RmIOClose, RmIOControl, RmIOOpen, RmIORead, RmIOWrite

RmMapMemory

Funktion

Physikalischen Speicher adressieren

Syntax

```
#include <rmapi.h>
int          RmMapMemory (
               ulong PhysAddress,
               ulong Length,
               void **pPointer );
```

Parameter

Parametername	Bedeutung
PhysAddress	Physikalische Anfangsadresse
Length	Länge des zu mappenden Speicherbereichs
pPointer	Adresse einer Variablen vom Typ Zeiger, in die die lineare Adresse des neu eingerichteten Speicherbereichs eingetragen wird. Über **pPointer* können Programme direkt auf den gemappten Adreßbereich zugreifen. Ist die lineare Adresse, d.h. **pPointer* gleich NULL, konnte der Speicherbereich nicht abgebildet werden.

Beschreibung

Die Funktion `RmMapMemory` bildet einen physikalischen Speicherbereich (z.B. Dual–Port–RAM oder Memory Mapped I/O) auf einen linaren Adreßraum (Anfangsadresse: **pPointer*, Länge: *Length*) ab. Anwenderprogramme können über den zurückgelieferten Zeiger **pPointer* auf den Speicher zugreifen (Zugriff ist READ/WRITE).

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_POINTER	Ein Zeiger war ungültig.

RmMemPoolAlloc

Funktion

Speicherbereich aus Speicherpool allokieren

Syntax

```
#include <rmapi.h>
int          RmMemPoolAlloc (
               ulong TimeOutValue,
               uint Mode,
               uint PoolID,
               ulong Size,
               void ** ppMemory)
```

Parameter

Parametername	Bedeutung
TimeOutValue	Maximale Wartezeit bis zur Ausführung RM_CONTINUE: Task fortsetzen, ohne auf Zuteilung des Speichers zu warten. RM_WAIT: Auf Zuteilung des Speichers warten. 0 ... RM_MAXTIME: Zeitintervall in ms. Die Task wartet entweder bis der Speicher zugeteilt wird oder das Timeout abgelaufen ist. Zur Zeitangabe können die Werte für Stunden, Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden. RM_HOUR(*hour*): Wartet (*hour*) Stunden RM_MINUTE(*min*): Wartet (*min*) Minuten RM_SECOND(*sec*): Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*): Wartet (*ms*) Millisekunden
Mode	Allokierungsmethode für den Speicher: RM_AUTOFREE: Der Speicher wird automatisch mit `RmFreeAll` freigegeben. Er ist taskspezifischzugeordnet. RM_NOAUTOFREE: Der Speicher wird nicht automatisch mit `RmFreeAll` freigegeben.
PoolID	ID des Speicherpools, aus dem der Speicherbereich angefordert wird
Size	Größe des Speicherbereichs
ppMemory	Adresse des Zeigers auf einen Speicherbereich.

Beschreibung

Der Aufruf allokiert einen Speicherbereich der Größe *Size* aus dem angegebenen Speicherbereich. **ppMemory* enthält einen gültigen Zeiger auf den allokierten Speicherbereich.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_TASK_WAITING	Aufruf mußte auf Zuteilung des Speichers warten.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_SIZE	*Size*=0 oder *Size* größer als Speicherpool
RM_INVALID_ID	Zur angegebenen ID existiert kein Speicherpool.
RM_OUT_OF_MEMORY	Kein Speicherbereich in der angegebenen Größe verfügbar
RM_GOT_TIMEOUT	In der angegebenen Zeit konnte kein entsprechender Speicherbereich allokiert werden

Siehe auch

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmGetMemPoolInfo, RmGetSize, RmReAlloc

RmPauseTask

Funktion **Zeitintervall abwarten**

Syntax

#include <rmapi.h>
int RmPauseTask(ulong *TimeValue***);**

Parameter

Parametername	Bedeutung
TimeValue	Zeitdauer der Pause 0... RM_MAXTIME Zeitintervall in ms. Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden. RM_HOUR(*hour*) Wartet (*hour*) Stunden RM_MINUTE(*min*) Wartet (*min*) Minuten RM_SECOND(*sec*) Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*) Wartet (*ms*) Millisekunden

Beschreibung

Mit `RmPauseTask` wird ein vorgegebenes Zeitintervall abgewartet. Bei *TimeValue*=0 wird bis zum nächsten Systemtakt gewartet.

Eine mit `RmPauseTask` unterbrochene Task wird durch `RmResumeTask` vorzeitig vom Zustand WARTEND in BEREIT überführt.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_TASK_RESUMED	Die Task wurde durch `RmResumeTask` fortgesetzt.

Siehe auch **RmRestartTask, RmResumeTask**

RmQueueStartTask

Funktion

Task-Start-Anforderung in Warteschlange einreihen. Die Task wird gestartet, sobald sie sich im Zustand RUHEND befindet.

Syntax

```
#include <rmapi.h>
int          RmQueueStartTask(
               uint Wait,
               uint TaskID,
               uint Priority,
               uint RegVal1,
               uint RegVal2);
```

Parameter

Parametername	Bedeutung	
Wait	RM_NO_WAIT	Zieltask starten und Task fortsetzen
	RM_WAIT_READY	Warten bis Zieltask im Zustand BEREIT ist.
	RM_WAIT_END	Warten bis Zieltask beendet ist
TaskID	Zieltask-ID	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI	Priorität aus TCD entnehmen
	RM_CURPRI	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI	Maximum einstellen (RM_TCDPRI, RM_CURPRI)
RegVal1	Parameter 1 (Übergabe in EAX der Zieltask)	
RegVal2	Parameter 2 (Übergabe in EBX der Zieltask)	

Beschreibung

Mit `RmQueueStartTask` wird eine Task gestartet. Der Aufruf benötigt dieselben Parameter wie `RmStartTask`.

Dieser Aufruf unterscheidet sich vom Aufruf `RmStartTask` in der Hinsicht, daß der Startaufruf in eine systeminterne Warteschlange eingetragen wird und ausgeführt wird, sobald die Task in den Zustand RUHEND überführt wird.

Befindet sich die zu startende Task bereits im Zusand RUHEND, so wirkt `RmQueueStartTask` identisch zu `RmStartTask`.

Rückgabewert

RM_OK	Aufruf erfolgreich ausgeführt, Zieltask wurde vom Zustand RUHEND in den Zustand BEREIT überführt bzw. die Startanforderung wurde in die systeminterne Warteschlange eingetragen.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *TaskID* wurde übergeben.
RM_TASK_KILLED	Die Zieltask wurde vor Erreichen des Zustands BEREIT oder vor ihrer Beendigung durch `RmKillTask` in den Zustand RUHEND versetzt bzw. gelöscht
RM_INVALID_TYPE	Ein ungültiger Parameter (*Priority*) wurde übergeben.

Siehe auch

RmEndTask, RmStartTask

RmReadMessage

Funktion

Nachricht aus Message-Queue abholen

Syntax

```
#include <rmapi.h>
int          RmReadMessage (
               ulong TimeOutValue,
               uint * pMessage,
               void **pMessageParam)
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Gibt an, wie lange bei leerer Message-Queue auf das Eintreffen einer Nachricht gewartet werden soll.	
	RM_CONTINUE	Task fortsetzen, ohne Eintreffen der Message abzuwarten
	RM_WAIT	Eintreffen der Message abwarten
	0 ... RM_MAXTIME	Zeitintervall in ms. Die Task wartet entweder bis die Message eingetroffen ist oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.	
	RM_HOUR(*hour*)	Wartet (*hour*) Stunden
	RM_MINUTE(*min*)	Wartet (*min*) Minuten
	RM_SECOND(*sec*)	Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*)	Wartet (*ms*) Millisekunden
pMessage	Adresse einer Variablen in der die Message–ID abgelegt wird.	
pMessageParam	Adresse eines Zeigers auf die Message–Parameter	

Beschreibung

Holt die Nachricht mit der höchsten Priorität aus der Message–Queue der aufrufenden Task.

Die Speicherplätze für die Message–ID und einen Zeiger auf die Message–Parameter müssen von der aufrufenden Task bereitgestellt werden.

`RmReadMessage` trägt in **pMessage* die Message–ID und in **pMessageParam* den Zeiger auf die eigentlichen Message–Parameter ein.

Ist keine Nachricht vorhanden, wird entsprechend *TimeOutValue* gewartet. Wird in dieser Zeit keine Nachricht empfangen erfolgt ein Abbruch mit Timeout.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt, eine Message wurde aus der Message–Queue ausgelesen. Der Parameter **pMessage* enthält die Nachrichten–ID und **pMessageParam* einen gültigen Zeiger auf die übermittelte Nachricht.

Fehlercodes

Fehlercode	Bedeutung
RM_GOT_TIMEOUT	In der angegebenen Zeit wurde keine Nachricht empfangen
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_NO_MESSAGE	Die Message–Queue enthält keine Nachricht (nur bei TimeOutValue = RM_CONTINUE)
RM_QUEUE_NOT_EXIST	Die Message-Queue existiert nicht.

Siehe auch

RmCreateMessageQueue, RmDeleteMessageQueue, RmSendMessage

RmReAlloc

Funktion

Größe eines Speicherbereichs verändern

Syntax

```
#include <rmapi.h>
int          RmReAlloc (
               ulong TimeOutValue,
               uint Mode,
               ulong NewSize,
               void **ppMemory)
```

Parameter

Parametername	Bedeutung
TimeOutValue	Maximale Wartezeit bis zur Ausführung RM_CONTINUE — Task fortsetzen, ohne auf Zuteilung des Speichers zu warten. RM_WAIT — Auf Zuteilung des Speichers warten. 0 ... RM_MAXTIME — Zeitintervall in ms. Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden. RM_HOUR(*hour*) — Wartet (*hour*) Stunden RM_MINUTE(*min*) — Wartet (*min*) Minuten RM_SECOND(*sec*) — Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*) — Wartet (*ms*) Millisekunden
Mode	Allokierungsmethode für den Speicher: RM_AUTOFREE — Der Speicher wird automatisch mit `RmFreeAll` freigegeben. Er ist taskspezifischzugeordnet. RM_NOAUTOFREE — Der Speicher wird nicht automatisch mit `RmFreeAll` freigegeben.
NewSize	Neue Größe des Speicherbereichs
ppMemory	Adresse des Zeigers auf einen Speicherbereich.

Beschreibung

Der Aufruf vergrößert oder verkleinert den mit **ppMemory* angegebenen Speicherbereich, ohne dessen Inhalt zu ändern. **ppMemory* enthält einen gültigen Zeiger auf den angepaßten Speicherbereich. Dieser Zeiger muß nicht mit dem übergebenen Zeiger übereinstimmen, da der Speicherbereich unter Umständen verschoben wurde.

Ist der ursprüngliche Speicherbereich **ppMemory* aus einem Pool angefordert worden, wird bei `RmReAlloc` ebenfalls dieser Pool verwendet.

Rückgabewert

RM_OK — Funktion erfolgreich ausgeführt

RM_TASK_WAITING	Aufruf mußte auf Zuteilung des Speichers warten.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_INVALID_SIZE	*Size*=0 oder *Size* größer als HEAP/Speicherpool
RM_OUT_OF_MEMORY	Kein Speicherbereich in der angegebenen Größe verfügbar
RM_GOT_TIMEOUT	In der angegebenen Zeit konnte kein entsprechender Speicherbereich allokiert werden

Siehe auch

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmGetSize, RmMemPoolAlloc

RmReceiveMail

Funktion

Botschaft aus lokaler Mailbox empfangen

Syntax

```
#include <rmapi.h>
int          RmReceiveMail(
               ulong TimeOutValue,
               uint MailboxID,
               void *pMail);
```

Parameter

Parametername	Bedeutung
TimeOutValue	Maximale Wartezeit bis zur Ausführung RM_CONTINUE — Task fortsetzen, ohne Eintreffen der Botschaft abzuwarten RM_WAIT — Eintreffen der Botschaft abwarten 0 ... RM_MAXTIME — Zeitintervall in ms. Die Task wartet entweder bis die Botschaft eingetroffen ist oder das Timeout abgelaufen ist. Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. RM_HOUR(*hour*) — Wartet (*hour*) Stunden RM_MINUTE(*min*) — Wartet (*min*) Minuten RM_SECOND(*sec*) — Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*) — Wartet (*ms*) Millisekunden
MailboxID	Mailbox-ID
pMail	Zeiger auf 12 Byte-Puffer (12 Byte Puffer)

Beschreibung

`RmReceiveMail` kopiert die 3-Wort-Botschaft mit der höchsten Priorität aus einer Mailbox in einen Anwender-Puffer und löscht die Botschaft in der Mailbox.

Ein 3–Wort großer Anwenderpuffer muß von der aufrufenden Task bereitgestellt werden.

Rückgabewert

RM_OK Inhalt von ***pMail** enthält Botschaft

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Mailbox-ID ungültig
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Fehlercode	Bedeutung
RM_NO_MESSAGE	Die Mailbox enthält keine Nachricht (nur bei TimeOutValue = RM_CONTINUE).
RM_GOT_TIMEOUT	Der Aufruf wurde nach dem eingestelltem Timeout abgebrochen.

Hinweis

Normalerweise enthält eine 3–Wort lange Botschaft entweder die eigentliche Nachricht oder einen Zeiger auf den eigentlichen Nachrichtenblock.
Im zweiten Fall wird von der Sender–Task der Nachrichtenblock für die eigentliche Information aus einem Speicherpool geholt und von der Task, die die Botschaft aus der Mailbox liest, wieder an den Speicherpool zurückgegeben. Die Wortlänge beträgt 32 Bit.

Siehe auch

RmCreateMailbox, RmDeleteMailbox, RmSendMail, RmSendMailCancel, RmSendMailDelayed

RmReleaseBinSemaphore

Funktion

Semaphor rücksetzen

Syntax

#include <rmapi.h>
int RmReleaseBinSemaphore(uint *SemaphoreID***);**

Parameter

Parametername	Bedeutung
SemaphoreID	Semaphor-ID

Beschreibung

`RmReleaseBinSemaphore` setzt das Semaphor *SemaphoreID* zurück.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *SemphoreID* wurde übergeben.

Hinweis

Die Belegung und Freigabe von Semaphoren ist nicht taskspezifisch.

Siehe auch

RmCreateBinSemaphore, RmDeleteBinSemaphore, RmGetBinSemaphore, automatische Prioritätsänderung durch Semaphor-Besitz im Programmierhandbuch

RmResetFlag

Funktion

Ereignisflag rücksetzen

Syntax

```
#include <rmapi.h>
int          RmResetFlag(
               uint FlagGrpID,
               uint FlagMask);
```

Parameter

Parametername	Bedeutung
FlagGrpID	Ereignisflag-Gruppe-ID. Der Wert 0 kennzeichnet die lokale Flaggruppe.
FlagMask	Maske legt fest, welche Bits rückgesetzt werden

Beschreibung

`RmResetFlag` setzt die in der Flagmaske spezifizierten Ereignisflags zurück und meldet, ob es zuvor gesetzt war.

Rückgabewert

RM_OK Funktion erfolgreich, kein Bit zurückgesetzt.

RM_FLAG_RESET Es wurde mindestens ein Bit zurückgesetzt.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *FlagGrpID* wurde übergeben.

Siehe auch

RmCreateFlagGrp, RmDeleteFlagGrp, RmGetFlag, RmSetFlag, RmSetFlagDelayed

RmRestartTask

Funktion

Task beenden und nach Zeitintervall erneut starten

Syntax

```
#include <rmapi.h>
int          RmRestartTask(
               uint Mode,
               ulong TimeValue);
```

Parameter

Parametername	Bedeutung	
Mode	RM_LAST_READY_TIME	Zeitrechnung auf letzte Überführung in den Zustand BEREIT beziehen
	RM_CURRENT_TIME	Zeitrechnung auf aktuelle Zeit beziehen
TimeValue	Wartezeit bis zum erneuten Start	
	0... RM_MAXTIME	Zeitintervall in ms.
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.	
	RM_HOUR(*hour*)	Wartet (*hour*) Stunden
	RM_MINUTE(*min*)	Wartet (*min*) Minuten
	RM_SECOND(*sec*)	Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*)	Wartet (*ms*) Millisekunden

Beschreibung

`RmRestartTask` beendet die Task-Ausführung und startet sie erneut nach Ablauf eines Zeitintervalls.

Wenn *TimeValue*=0 , wird die Task mit dem nächsten Timer-Interrupt in den Zustand BEREIT überführt.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Hinweis

`RmRestartTask` überführt eine Task in den Zustand WARTEND und nicht in den Zustand RUHEND. Im Unterschied zu `RmPauseTask` wird die Task nach Ablauf der in `RmRestartTask` festgelegten Zeit gestartet, d.h. die Programmabarbeitung beginnt an der Einsprungadresse der Task.

Eine durch `RmRestartTask` unterbrochene Task kann nur durch Ablauf des Zeitintervalls in den Zustand BEREIT überführt werden.

Bei einer mit `RmRestartTask` gestarteten Task ist keine Parameterübergabe in EAX, EBX möglich. Die Parameter können beim ersten Start der Task (mit einem anderen Startbefehl) übergeben und zwischengespeichert werden. Dadurch können diese Parameter nach jedem durch `RmRestartTask` erfolgten Start der Task wieder verwendet werden.

Wurde eine Task (`main()`) vom CLI gestartet, darf sie mit `RmRestart-Task` nicht erneut gestartet werden.

Siehe auch **RmActivateTask, RmPauseTask, RmResumeTask**

RmResumeTask

Funktion

Mit `RmPauseTask` **oder** `RmSuspendTask` **angehaltene Task fortsetzen**

Syntax

#include <rmapi.h>
int RmResumeTask(uint *TaskID***);**

Parameter

Parametername	Bedeutung
TaskID	Task-ID

Beschreibung

`RmResumeTask` überführt eine Task, die durch `RmSuspendTask` bzw. `RmPauseTask` in den Zustand WARTEND versetzt wurde, in den Zustand BEREIT.

Die Programmabarbeitung wird, im Gegensatz zu `RmRestartTask`, direkt nach dem Aufruf von `RmSuspendTask` bzw. `RmPauseTask` fortgesetzt.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *TaskID* wurde übergeben.
RM_TASK_NOT_PAUSED	Die mittels `RmResumeTask` fortzusetzende Task wurde nicht durch `RmPauseTask` angehalten oder ist nicht mehr im Zustand WARTEND.

Siehe auch

RmActivateTask, RmPauseTask, RmRestartTask, RmSuspendTask

RmSendMail

Funktion

Botschaft an eine Mailbox senden

Syntax

```
#include <rmapi.h>
int          RmSendMail(
               ulong TimeOutValue,
               uint Priority,
               uint MailboxID,
               void *pMail);
```

Parameter

Parametername	Bedeutung	
TimeOutValue	Maximale Wartezeit bis zur Ausführung	
	RM_CONTINUE	Task fortsetzen, ohne Abholen der Botschaft abzuwarten
	RM_WAIT	Abholen der Botschaft abwarten
	0 ... RM_MAXTIME	Zeitintervall in ms. Die Task wartet entweder bis die Botschaft abgeholt wurde oder das Timeout abgelaufen ist.
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.	
	RM_HOUR(*hour*)	Wartet (*hour*) Stunden
	RM_MINUTE(*min*)	Wartet (*min*) Minuten
	RM_SECOND(*sec*)	Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*)	Wartet (*ms*) Millisekunden
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI	Priorität aus TCD entnehmen
	RM_CURPRI	Aktuelle Priorität der aufrufenden Task verwenden
MailboxID	Mailbox-ID	
pMail	Zeiger auf 3-Wort-Puffer	

Beschreibung

`RmSendMail` kopiert eine 3 Wort lange, prioritätsbehaftete Botschaft in eine Mailbox. Die Task kann in den Zustand WARTEND versetzt werden, bis die Botschaft abgeholt oder der Aufruf nach Ablauf eines Timeout abgebrochen wird.

Das Botschaftsformat ist frei wählbar. Es kann beispielsweise eine 3 Wort (32 Bit) lange Botschaft oder die Adresse und Länge einer Botschaft mit folgendem Schema angegeben werden:

Botschaftswort 1 : Adresse des Nachrichtenblocks
Botschaftswort 2 : beliebig
Botschaftswort 3 : Länge des Nachrichtenblocks in Bytes

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt, die Botschaft wurde in die Mailbox kopiert.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *MailboxID* wurde übergeben.
RM_INVALID_TYPE	Ein ungültiger Parameter (*Priority*) wurde übergeben.
RM_INVALID_POINTER	Ein Zeiger war ungültig.
RM_GOT_TIMEOUT	Der Aufruf wurde nach dem eingestellten Timeout abgebrochen.
RM_BOUND_REACHED	Die Anforderung überschreitet die eingetragene Grenze der Mailbox (siehe `RmSetMailboxSize`).

Siehe auch

RmCreateMailbox, RmDeleteMailbox, RmReceiveMail, RmSetMailboxSize

RmSendMailCancel

Funktion

Abbruch einer mit `RmSendMailDelayed` **gestarteten Botschaft**

Syntax

```
#include <rmapi.h>
int          RmSendMailCancel (
               RmMailIDStruct *pMailID,
               void *pMail);
```

Parameter

Parametername	Bedeutung
pMailID	Zeiger auf eine Struktur vom Typ **RmMailIDStruct** (siehe Kapitel 3). Der Aufruf `RmSendMailDelayed` liefert den Zeiger auf die zugehörige **RmMailIDStruct** zurück.
pMail	Zeiger auf Puffer, in den die vorher verschickte Botschaft zurückgeschrieben wird. Die Länge der Botschaft beträgt 12 Byte.

Beschreibung

Der Aufruf bricht eine durch `RmSendMailDelayed` gestartete Botschaft ab. Der Abbruch ist nur möglich, solange das Zeitintervall noch nicht abgelaufen ist oder die betreffende Botschaft noch nicht abgeholt wurde. Im letzteren Fall wird die Botschaft außerdem aus der Mailbox entfernt.

Der vorausgehende Aufruf `RmSendMailDelayed` liefert seinerseits Information in einer Struktur **RmMailIDStruct** zurück. Die Adresse dieser Struktur muß dem Aufruf `RmSendMailCancel` übergeben werden.

Die aufrufende Task erhält den Inhalt der Botschaft zurück, um eventuell darin enthaltene Informationen auswerten zu können.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige Botschaft wurde in *pMailID* übergeben. Trifft auch zu, wenn eine verschickte Botschaft bereits abgeholt wurde. Der durch *pMail* definierte Speicher ist undefiniert.
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Siehe auch

RmCreateMailbox, RmDeleteMailbox, RmReceiveMail, RmSendMail, RmSendMailDelayed, RmSetMailboxSize

RmSendMailDelayed

Funktion

Botschaft zeitverzögert an eine Mailbox schicken

Syntax

```
#include <rmapi.h>
int          RmSendMailDelayed (
               ulong TimeValue,
               uint Priority,
               uint MailboxID,
               void *pMail,
               RmMailIDStruct *pMailID);
```

Parameter

Parametername	Bedeutung
TimeValue	Zeit bis Botschaft abgeschickt wird 0... RM_MAXTIME Zeitintervall in ms. Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden. RM_HOUR(*hour*) Wartet (hour) Stunden RM_MINUTE(*min*) Wartet (*min*) Minuten RM_SECOND(*sec*) Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*) Wartet (*ms*) Millisekunden
Priority	0..255 Vorgegebenen Wert einstellen RM_TCDPRI Priorität aus TCD entnehmen RM_CURPRI Aktuelle Priorität der aufrufenden Task verwenden
MailboxID	Mailbox-ID
pMail	Zeiger auf Botschaft. Die Länge der Botschaft beträgt 12 Byte.
pMailID	Zeiger auf eine Struktur vom Typ **RmMailIDStruct** (siehe Kapitel 3).

Beschreibung

Mit `RmSendMailDelayed` wird eine Botschaft zeitverzögert an eine Mailbox gesendet. Die aufrufende Task muß die Adresse einers Speicherbereichs des Typs **RmMailIDStruct** übergeben.
In diesem Speicherbereich trägt der Aufruf eine Identifikation ein, mit der ggfs. die Aktion mittels `RmSendMailCancel` abgebrochen werden kann.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt, die **RmMailIDStruct**–Variable enthält die Identifikation des zugehörigen Auftrags.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_TYPE	Ein ungültiger Parameter (*Priority*) wurde übergeben.
RM_INVALID_ID	Ungültige Flag–Gruppe
RM_INVALID_POINTER	Ein Zeiger war ungültig.

Hinweis

Ein durch `RmSetMailboxSize` festgelegter Grenzwert von Botschaften, die in einer Mailbox auf Abholung warten, wird beim Verschicken der Botschaft mit `RmSendMailDelayed` nicht berücksichtigt.

Es ist prinzipiell möglich, daß die Mailbox, an welche die Botschaft geschickt werden soll, vor Ablauf des Zeitintervalls bereits durch den Systemaufruf `RmDeleteMailbox` gelöscht wurde. In diesem Fall wird die Botschaft stillschweigend verworfen.

Siehe auch

RmCreateMailbox, RmDeleteMailbox, RmReceiveMail, RmSendMail, RmSendMailCancel, RmSetMailboxSize

RmSendMessage

Funktion

Nachricht in Message-Queue einhängen

Syntax

```
#include <rmapi.h>
int          RmSendMessage (
               ulong TimeOutValue,
               uint Priority,
               uint TaskID,
               uint Message,
               void *pMessageParam)
```

Parameter

Parametername	Bedeutung
TimeOutValue	Gibt an, wie lange auf das Abholen der Nachricht gewartet werden soll. RM_CONTINUE — Task fortsetzen, ohne Abholen der Nachricht abzuwarten RM_WAIT — Abholen der Nachricht abwarten 0 ... RM_MAXTIME — Zeitinterval in ms. Die Task wartet entweder bis die Nachricht abgeholt wird oder das Timeout abgelaufen ist. Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden. RM_HOUR(*hour*) — Wartet (hour) Stunden RM_MINUTE(*min*) — Wartet (*min*) Minuten RM_SECOND(*sec*) — Wartet (*sec*) Sekunden RM_MILLISECOND(*ms*) — Wartet (*ms*) Millisekunden
Priority	Priorität der Nachricht 0..255 — Vorgegebenen Wert einstellen RM_TCPPRI — Priorität aus TCD entnehmen RM_CURPRI — Aktuelle Priorität der aufrufenden Task verwenden
TaskID	Zieltask-ID
Message	Identifikationskennung der Nachricht. Die Identifikationskennung der Nachrichten ist wie folgt festgelegt: RM_MSG_USER..RM_MSG_MAX reserviert für den Anwender
pMessageParam	Zeiger auf den Inhalt der Nachricht.

Beschreibung

Der Aufruf hängt die Nachricht *Message* zusammen mit dem Zeiger auf die Message–Parameter und der vorgegebenen Priorität in die entsprechende Stelle der Message–Queue der Task *TaskID*. Der Parameter *TimeOutValue*

gibt an, ob und wie lange auf das Abholen der Nachricht gewartet werden soll.

Hinweis

Wird ***RmSendMessage*** mit ***TimeOutValue=RM_WAIT*** aufgerufen, so kann es zu folgendem Verhalten kommen:

Wird die Task geweckt (z.B. mit ***RmActivateTask***) während ***RmSendMessage*** auf das Abholen der Nachricht wartet, so meldet ***RmSendMessage*** Erfolg, obwohl nicht sichergestellt ist, ob die Nachricht tatsächlich abgeholt wurde oder nicht.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, Nachricht in die Task–eigene Message Queue kopiert.

Fehlercodes

Fehlercode	Bedeutung
RM_GOT_TIMEOUT	In der angegebenen Zeit wurde die Nachricht nicht abgeholt.
RM_INVALID_ID	Task-ID ungültig
RM_INVALID_POINTER	Ungültiger Zeiger
RM_INVALID_TYPE	Ein ungültiger Parameter (*Priority*) wurde übergeben.
RM_QUEUE_NOT_EXIST	Die Message Queue existiert nicht.
RM_BOUND_REACHED	Die Message Queue ist voll.

Siehe auch

RmCreateMessageQueue, RmDeleteMessageQueue, RmReadMessage

RmSetFlag

Funktion

Ereignisflag setzen

Syntax

```
#include <rmapi.h>
int          RmSetFlag(
               uint FlagGrpID,
               uint FlagMask);
```

Parameter

Parametername	Bedeutung
FlagGrpID	Flaggruppen-ID. DerWert 0 kennzeichnet die lokale Flaggruppe.
FlagMask	Maske legt fest, welche Bits gesetzt werden

Beschreibung

`RmSetFlag` setzt ein Ereignisflag und meldet, ob es zuvor gesetzt war.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt, kein Bit gesetzt
RM_FLAG_SET	Es wurde mindestens ein Flag gesetzt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *FlagGrpID* wurde übergeben.

Siehe auch

RmCreateFlagGrp, RmDeleteFlagGrp, RmGetFlag, RmResetFlag

RmSetFlagDelayed

Funktion

Ereignisflag nach Intervall setzen

Syntax

```
#include <rmapi.h>
int          RmSetFlagDelayed(
               ulong TimeValue,
               uint FlagGrpID,
               uint FlagMask);
```

Parameter

Parametername	Bedeutung
TimeValue	Verzögerungszeit bis zum Setzen des Flags
	0... RM_MAXTIME Zeitinterval in ms.
	Zur Zeitangabe können die Werte für Stunden. Minuten und Sekunden durch Addition verknüpft werden. Die maximale Wartezeit beträgt 2^31 Millisekunden.
	RM_HOUR(*hour*) Wartet (hour) Stunden
	RM_MINUTE(*min*) Wartet (*min*) Minuten
	RM_SECOND(*sec*) Wartet (*sec*) Sekunden
	RM_MILLISECOND(*ms*) Wartet (*ms*) Sekunden
FlagGrpID	Flaggruppen-ID. DerWert 0 kennzeichnet die lokale Flaggruppe.
FlagMask	Maske legt fest, welche Bits beeinflußt werden

Beschreibung

Mit `RmSetFlagDelayed` werden die mit *FlagMask* angegebenen Bits gelöscht und nach Ablauf des Zeitintervalls gesetzt. Nicht gesetzte und durch *FlagMask* angegebene Bits derselben *FlagGrpID* werden überprüft. Die Timerwerte dieser Bits werden ggf. auf den neuen Wert gesetzt.
Ein `RmSetFlagDelayed` wird durch ein zweites `RmSetFlagDelayed` mit identischer *FlagGrpID* und *FlagMask* bei positiver Zeitangabe überschrieben bzw. mit Zeitangabe = 0 gelöscht. Ein `RmResetFlag` wirkt sich nicht auf `RmSetFlagDelayed` aus.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *FlagGrpID* wurde übergeben.
RM_PARAMETER_ERROR	Der Aufruf wurde mit falschen Parametern versorgt (*FlagMask*=0).

Siehe auch **RmCreateFlagGrp, RmDeleteFlagGrp, RmGetFlag, RmResetFlag, RmSetFlag**

RmSetIntDefHandler

Funktion

Default Interrupt–Handler installieren

Syntax

#include <rmapi.h>
int RmSetIntDefHandler (uint *IntNum***);**

Parameter

Parametername	Bedeutung
IntNum	SW–Interrupt-Nummer (0–255) oder IRQx (x=0 bis 63) HW–Interrupt IRQ(n) (n=0bis63) HW–Interrupt Auf M7–300/400 sind die HW–Interrupts auf 0 bis 15.

Beschreibung

Mit diesem Aufruf kann ein eigener Interrupt–Handler für den angegebenen Interrupt *IntNum* deinstalliert und wieder mit dem defaultmäßigen Interrupt–Handler vorbesetzt werden.

Die Interrupt–Nummer indiziert die Einträge in der Interrupt–Deskriptor–Tabelle, d.h. die Interrupt–Nummer entspricht dem Selektor des zugehörigen Deskriptors. Im Deskriptor ist die Einsprungadresse des zugehörigen Interrupt–Handlers eingetragen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt, eigener Interrupt–Handler wurde deinstalliert.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_INTERRUPT_NUMBER	Ungültige Interrupt-Nummer
RM_INVALID_IRQ_NUMBER	IRQx ungültig, PIC nicht definiert

Siehe auch

RmGetIntHandler, RmSetIntISHandler, RmSetIntMailboxHandler, RmSetIntTaskHandler

RmSetIntISHandler

Funktion

S- bzw I-Interrupt–Handler vorbesetzen

Syntax

```
#include <rmapi.h>
int          RmSetIntISHandler (
               uint IntNum,
               rmfarproc IHandlerEntry,
               rmfarproc SHandlerEntry);
```

Parameter

Parametername	Bedeutung
IntNum	SW–Interrupt-Nummer (0–255) oder IRQx (x=0 bis 63) HW–Interrupt IRQ(n) (n=0bis63) HW–Interrupt Auf M7–300/400 sind die HW–Interrupts auf 0 bis 15.
IHandlerEntry	Einsprungadresse des I-Interrupt–Handler.
SHandlerEntry	Einsprungadresse des S-Interrupt–Handler.

Beschreibung

Der Aufruf definiert einen I- und/oder S-Interrupt–Handler.

Handelt es sich hierbei um einen HW-Interrupt z.B. IRQ1, wird dieser automatisch maskiert.

Während der neue Interrupt–Handler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Der in *IHandlerEntry* bzw. *SHandlerEntry* angegebene Interrupt–Handler wird direkt nach einem Interrupt im I- bzw. S-Zustand angesprungen. Soll ein Handler nicht installiert werden, so muß NULL angegeben werden.

Der *SHandlerEntry* wird nur aufgerufen, wenn der Rückgabewert vom I–Zustand ≠ 0 ist. Ist der Rückgabewert gleich 0, erfolgt kein Übergang in den S–Zustand.

Die Interrupt–Nummer indiziert die Einträge in der Interrupt–Deskriptor–Tabelle, d.h. die Interrupt–Nummer entspricht dem Selektor des zugehörigen Deskriptors. Im Deskriptor ist die Einsprungadresse des zugehörigen Interrupt–Handlers eingetragen.

Mit `RmSetIntISHandler` wird ein Interrupt-Gate in die IDT eingetragen.

Der Vor– und Nachspann für den Interrupt–Handler werden vom Betriebssystemkern generiert. Die Handler können dann einfache Prozeduren sein. Der Speicherbedarf für einen Interrupt–Handler beträgt ca. 130 Byte und wird dem HEAP entnommen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt.

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar.
RM_INVALID_INTERRUPT_NUMBER	Ungültige Interrupt-Nummer
RM_INVALID_IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
RM_INVALID_POINTER	Ungültiger Zeiger

Hinweis

Wird der Funktionsaufruf nicht erfolgreich ausgeführt, bleibt der bisherige Interrupt–Handler aktiv.

Ein Anwenderprogramm läuft auf dem M7–System im “User Level”. Dabei ist nur der Bereich für Anwenderdaten schreibbar, Code– und Systembereiche sind vor dem Überschreiben durch eine Anwendertask geschützt.

Ein I– bzw. S–Handler wird im “System Level” ausgeführt., d.h. innerhalb eines Interupthandlers ist der Speicherschutz aufgehoben.

Siehe auch

RmGetIntHandler, RmSetIntDefHandler, RmSetIntMailboxHandler, RmSetIntTaskHandler

RmSetIntMailboxHandler

Funktion

Mailbox-Interrupt-Handler vorbesetzen

Syntax

```
#include <rmapi.h>
int          RmSetIntMailboxHandler (
               uint IntNum,
               uint MailboxID,
               uint MailPriority);
```

Parameter

Parametername	Bedeutung
IntNum	SW–Interrupt-Nummer (0–255) oder IRQx (x=0 bis 63) HW–Interrupt IRQ(n) (n=0bis63) HW–Interrupt Auf M7–300/400 sind die HW–Interrupts auf 0 bis 15.
MailboxID	Mailbox-ID Es wird eine Botschaft an die mit *MailBoxID* angegebene Mailbox geschickt. Unterliegt diese Mailbox einem `RmSetMailboxSize`, d.h. es darf nur eine bestimmte Anzahl von Botschaften in der Mailbox auf Abholung warten und ist diese maximale Anzahl bereits erreicht, so wird keine Botschaft abgeschickt. Der Interrupt gilt als verloren. Die Botschaft ist eine Struktur vom Typ **RmIntrhandMailStruct** und ist in Kapitel 3 beschrieben.
MailPriority	Priorität der Botschaft

Beschreibung

Der Aufruf definiert einen Handler zum Senden einer Botschaft.

Handelt es sich hierbei um einen HW-Interrupt z.B. IRQ1 wird dieser automatisch maskiert.

Während der neue Interrupt–Handler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Ist die Anzahl von Botschaften einer Mailbox (siehe `RmSetMailboxSize`) begrenzt, wird bei Erreichen dieser Grenze keine Botschaft abgeschickt. Der Interrupt ist verloren.

Mit `RmSetIntMailboxHandler` wird ein Interrupt-Gate in die IDT eingetragen. Bereits vorhandene Einträge in der IDT bleiben erhalten, können aber durch den Aufruf überschrieben werden.

Der Code für den Interrupt–Handler zum Verschicken der Botschaft wird vom Betriebssystemkern generiert. Der Speicherbedarf für einen Interrupt–Handler beträgt ca. 130 Byte und wird dem HEAP entnommen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar.
RM_INVALID_INTERRUPT_NUMBER	Ungültige Interrupt-Nummer
RM_INVALID_IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
RM_INVALID_ID	Ungültige Mailbox-ID

Hinweis

Wird der Aufruf nicht erfolgreich ausgeführt, bleibt der bisherige Interrupt–Handler aktiv.

Siehe auch

RmGetIntHandler, RmSetIntDefHandler, RmSetIntISHandler, RmSetIntTaskHandler

RmSetIntTaskHandler

Funktion

Interrupt-Handler für Taskstart vorbesetzen

Syntax

```
#include <rmapi.h>
int          RmSetIntTaskHandler (
               uint IntNum,
               uint TaskID);
```

Parameter

Parametername	Bedeutung
IntNum	SW–Interrupt-Nummer (0–255) oder IRQx (x=0 bis 63) HW–Interrupt IRQ(n) (n=0bis63) HW–Interrupt Auf M7–300/400 sind die HW–Interrupts auf 0 bis 15.
TaskID	Task-ID (RM_OWN_TASK = ID der aufrufenden Task)

Beschreibung

Der Aufruf definiert einen Handler für einen Taskstart per Interrupt.

Handelt es sich hierbei um einen HW-Interrupt z.B. IRQ1, wird dieser automatisch maskiert.

Während der neue Interrupt–Handler gesetzt wird, darf kein Interrupt für diesen Handler auftreten.

Die in *TaskID* angegebene Task soll direkt nach einem Interrupt angesprungen werden.

Die Interrupt-Nummer entspricht dem Selektor des betreffenden Deskriptors in der IDT.

Mit `RmSetIntTaskHandler` wird ein Interrupt-Gate in die IDT eingetragen.

Der Code für den Interrupt–Handler zum Starten der Task wird vom Betriebssystemkern generiert. Der Speicherbedarf für einen Interrupt–Handler beträgt ca. 130 Byte und wird dem HEAP entnommen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar.
RM_INVALID_INTERRUPT_NUMBER	Ungültige Interrupt-Nummer
RM_INVALID_IRQ_NUMBER	IRQx ungültig, PIC nicht definiert
RM_INVALID_ID	Ungültige Task-ID

Hinweis

Wird der Aufruf nicht erfolgreich ausgeführt, bleibt der bisherige Interrupt–Handler aktiv.

Siehe auch

RmGetIntHandler, RmSetIntDefHandler, RmSetIntISHandler, RmSetIntMailboxHandler

RmSetMailboxSize

Funktion

Grenzwerte für Mailboxes festlegen

Syntax

```
#include <rmapi.h>
int          RmSetMailboxSize (
               uint MailboxID
               uint Limit);
```

Parameter

Parametername	Bedeutung	
MailboxID	Mailbox-ID	
Limit	1-0FFFFH	Maximale Anzahl von Botschaften in Warteschlange
	0	Bedeutet, der Grenzwert soll aufgehoben werden.

Beschreibung

Der Aufruf setzt einen Grenzwert für die Anzahl von Botschaften, die an einer Mailbox warten können. Der Grenzwert kann beliebig modifiziert und auch wieder aufgehoben werden.

Beim Überschreiten des Grenzwerts, werden alle nachfolgenden Versuche, mit dem Aufruf `RmSendMail` eine Botschaft an diese Mailbox zu schicken, zurückgewiesen. Aufrufe von `RmSendMail` werden erst wieder akzeptiert, wenn der Grenzwert durch Abholen von Botschaften wieder unterschritten wird.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Mailbox-ID ungültig

Hinweis

Der bei Mailboxen festgelegte Grenzwert hat bei dem Systemaufruf `RmSendMailDelayed` keine Wirkung.

Siehe auch

RmReceiveMail, RmSendMail, RmSendMailCancel, RmSendMailDelayed

RmSetMessageQueueSize

Funktion

Länge der Message-Queue festlegen

Syntax

```
#include <rmapi.h>
int          RmSetMessageQueueSize (
               uint TaskID,
               uint Limit)
```

Parameter

Parametername	Bedeutung
TaskID	Zieltask-ID
Limit	Anzahl von freien Plätzen in der Message-Queue

Beschreibung

Der Aufruf legt die Größe der Message-Queue der Task *TaskID* fest.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt.

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Task-ID ungültig
RM_INVALID_TYPE	Ein ungültiger Parameter (*Limit*) wurde übergeben.
RM_QUEUE_NOT_EXIST	Die Message-Queue existiert nicht.

Siehe auch

RmCreateMessageQueue, RmDeleteMessageQueue, RmReadMessage, RmSendMessage

RmSetTaskPriority

Funktion

Task-Priorität ändern

Syntax

```
#include <rmapi.h>
int          RmSetTaskPriority(
               uint TaskID,
               uint Priority);
```

Parameter

Parametername	Bedeutung	
TaskID	Zieltask-ID (RM_OWN_TASK=eigene)	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI	Priorität aus TCD entnehmen
	RM_CURPRI	Aktuelle Priorität der aufrufenden Task verwenden
	RM_INCPRI	Priorität der Task um 1 erhöhen
	RM_DECPRI	Priorität der Task um 1 erniedrigen

Beschreibung

Mit `RmSetTaskPriority` wird die Priorität einer beliebigen Task verändert.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	*TaskID* ungültig
RM_INVALID_TYPE	Ein ungültiger Parameter (*Priority*) wurde übergeben.
RM_PRI_NOT_CHANGED	Die Priorität hatte bereits den angegebenen Wert.
RM_TASK_DORMANT	Task momentan RUHEND

Siehe auch

RmStartTask, RmQueueStartTask, Beschreibung der Task-Prioritäten im Programmierhandbuch

RmStartTask

Funktion

Start-Anforderung für Tasks, die sich im Zustand RUHEND befinden

Syntax

```
#include <rmapi.h>
int          RmStartTask(
               uint Wait,
               uint TaskID,
               uint Priority,
               uint RegVal1,
               uint RegVal2);
```

Parameter

Parametername	Bedeutung	
Wait	RM_NO_WAIT	Zieltask starten und fortsetzen
	RM_WAIT_READY	Warten bis Zieltask im Zustand BEREIT ist
	RM_WAIT_END	Warten bis Zieltask beendet ist
TaskID	Zieltask-ID (RM_OWN_TASK = eigene Task)	
Priority	0..255	Vorgegebenen Wert einstellen
	RM_TCDPRI	Priorität aus TCD entnehmen
	RM_CURPRI	Aktuelle Priorität der aufrufenden Task verwenden
	RM_MAXPRI	Maximum (RM_TCDPRI, RM_CURPRI) einstellen
RegVal1	Parameter 1 (Übergabe in eax der Zieltask)	
RegVal2	Parameter 2 (Übergabe in ebx der Zieltask)	

Beschreibung

Mit `RmStartTask` wird eine Task gestartet. Der Aufruf benötigt dieselben Parameter wie `RmQueueStartTask`. Der Unterschied zu `RmQueue-StartTask` besteht darin, daß `RmQueueStartTask` die Start-Anforderung ggf. in eine Warteschlange einreiht, falls sich die Task nicht im Zustand RUHEND befindet. Der Aufruf `RmStartTask` hingegen ist in diesem Fall wirkungslos.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt, Zieltask wurde in den Zustand BEREIT überführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	Eine ungültige *TaskID* wurde übergeben.
RM_INVALID_TYPE	Ein ungültiger Parameter (*Wait*) wurde übergeben.

Fehlercode	Bedeutung
RM_TASK_NOT_DORMANT	Es wurde versucht, eine Task, die nicht im Zustand RUHEND ist, zu starten.
RM_TASK_KILLED	Zieltask wurde vor Erreichen des Zustands BEREIT oder vor ihrer Beendigung durch RmKillTask in den Zustand RUHEND versetzt bzw. gelöscht.

Siehe auch **RmEndTask, RmQueueStartTask**

RmSuspendTask

Funktion

Task vom Zustand BEREIT in Zustand WARTEND setzen

Syntax

#include <rmapi.h>
int **RmSuspendTask(uint** *TaskID***);**

Parameter

Parametername	Bedeutung
TaskID	Task-ID

Beschreibung

`RmSuspendTask` hält die mit *TaskID* angegebene Task an. Die angehaltene Task muß im Zustand BEREIT sein und befindet sich danach im Zustand WARTEND. Eine Task kann sich auch selbst anhalten.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_INVALID_ID	TaskID ungültig
RM_TASK_NOT_READY	Task war nicht im Zustand BEREIT

Siehe auch

RmResumeTask

RmUncatalog

Funktion

Betriebsmittel aus Katalog löschen

Syntax

#include <rmapi.h>
int **RmUncatalog (char *_pName_)**

Parameter

Parametername	Bedeutung
pName	Zeiger auf ein Zeichenstring (Der String kann nach C– oder PLM–Notation definiert sein).

Beschreibung

`RmUncatalog` entfernt das durch einen Zeichenstring identifizierte Betriebsmittel aus dem Katalog.

Rückgabewert

RM_OK Funktion ausgeführt.

Fehlercodes

Fehlercode	Bedeutung
RM_IS_NOT_CATALOGED	Eintrag nicht gefunden
RM_INVALID_POINTER	Zeiger *pName* war ungültig.
RM_INVALID_STRING	Stringlänge = 0 oder > 15

Hinweis

Ist ein Betriebsmittel mit unterschiedlichen Strings mehrmals katalogisiert, so werden alle Einträge dieses Betriebsmittels aus dem Katalog gelöscht.

Siehe auch

RmCatalog, RmGetEntry, RmGetName, RmList

SerialCheckChar

Funktion

Einzelnes Zeichen von Unit einlesen

Syntax

```
#include <serial.h>
int          SerialCheckChar(
               RmIOHandle Handle,
               char *Char );
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
Char	Adresse eines `char`, in dem das eingelesene Zeichen abgespeichert wird.

Beschreibung

`SerialCheckChar` liest ein einzelnes Zeichen von der mit *Handle* bezeichneten Unit und speichert es an der durch *Char* angegebenen Adresse. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Im Gegensatz zum Aufruf `SerialGetChar` wartet `SerialCheckChar` nicht auf das Eintreffen des Zeichens. Befindet sich im Hintergrundpuffer der Unit kein Zeichen, beendet sich `SerialCheckChar`.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_IO_NO_DATA	Keine Daten vorhanden

Weitere Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialCheckString, SerialGetChar, SerialGetString, SerialOpen

SerialCheckString

Funktion

String von Unit einlesen

Syntax

```
#include <serial.h>
int          SerialCheckString(
               RmIOHandle Handle,
               ulong MaxLen,
               char *String,
               ulong *Count );
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
MaxLen	Maximale Anzahl zu lesender Zeichen.
String	Adresse eines Speicherbereichs, in dem die gelesenen Zeichen abgespeichert werden.
Count	Adresse eines `ulong`, in dem die Anzahl der gelesenen Zeichen abgelegt wird. Wert > 0 Anzahl gelesener Zeichen Wert = 0 Fehler bzw. keine Zeichen vorhanden

Beschreibung

`SerialCheckString` liest *MaxLen* Zeichen von der mit *Handle* bezeichneten Unit und speichert es an der durch *String* angegebenen Adresse. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Der Parameter *Count* enthält nach erfolgreichem Leseauftrag die Anzahl gelesener Zeichen. Schlug der Leseauftrag fehl oder waren keine Zeichen vorhanden, enthält der Parameter den Wert 0.

Im Gegensatz zu `SerialGetString` wartet `SerialCheckString` nicht auf das Eintreffen des Zeichens. Befindet sich im Hintergrundpuffer der Unit kein Zeichen, beendet sich `SerialCheckString`.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlercode	Bedeutung
RM_IO_NO_DATA	Keine Daten vorhanden

Weitere Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch **SerialCheckChar, SerialGetChar, SerialGetString, SerialOpen**

SerialClose

Funktion **Verbindung zu einer Unit eines Treibers schließen**

Syntax

#include <serial.h>
int **SerialClose(RmIOHandle** *Handle***);**

Parameter

Parametername	Bedeutung
Handle	Deskriptor

Beschreibung `SerialClose` schließt die durch *Handle* bezeichnete Verbindung. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Rückgabewert RM_OK Funktion erfolgreich ausgeführt

Fehlercodes Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch **SerialOpen**

SerialGetChar

Funktion

Einzelnes Zeichen von Unit einlesen

Syntax

```
#include <serial.h>
int          SerialGetChar(
               RmIOHandle Handle,
               char *Char);
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
Char	Adresse eines `char`, in dem das eingelesene Zeichen abgespeichert wird.

Beschreibung

`SerialGetChar` liest ein einzelnes Zeichen von der mit *Handle* bezeichneten Unit und speichert es an der durch *Char* angegebenen Adresse. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde. Der Aufruf wartet auf das Eintreffen des Zeichens.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialCheckChar, SerialCheckString, SerialGetString, SerialOpen

SerialGetString

Funktion

String von Unit einlesen

Syntax

```
#include <serial.h>
int          SerialGetString(
               RmIOHandle Handle,
               ulong MaxLen,
               char *String,
               ulong *Count );
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
MaxLen	Maximale Anzahl zu lesender Zeichen.
String	Adresse eines Speicherbereichs, in dem die gelesenen Zeichen abgespeichert werden.
Count	Adresse eines `ulong`, in dem die Anzahl der gelesenen Zeichen abgelegt wird. Wert > 0 Anzahl gelesener Zeichen Wert = 0 Fehler bzw. keine Zeichen vorhanden

Beschreibung

`SerialGetString` liest maximal *MaxLen* Zeichen von der mit *Handle* bezeichneten Unit und legt diese im Speicherbereich mit der Adresse *String* ab. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Der Parameter *Count* enthält nach erfolgreichem Leseauftrag die Anzahl gelesener Zeichen. Schlug der Leseauftrag fehl oder waren keine Zeichen vorhanden, enthält der Parameter den Wert 0.

Der Aufruf wartet auf das Eintreffen der Zeichen.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialCheckChar, SerialCheckString, SerialGetChar, SerialOpen

SerialInit

Funktion

Unit initialisieren

Syntax

```
#include <serial.h>
int          SerialInit(
               RmIOHandle Handle,
               ulong Baud,
               uint Data,
               uint Parity,
               uint Stop);
```

Parameter

Parametername	Bedeutung	
Handle	Deskriptor	
Baud	Baudrate als Zahlenwert (z.B. 19200)	
Data	Anzahl der Daten–Bits als Zahlenwert (z.B. 8)	
Parity	Parität	
	SERIAL_PARITYNONE	keine Prüfung
	SERIAL_PARITYEVEN	gerade Parität
	SERIAL_PARITYODD	ungerade Parität
	SERIAL_PARITY0	Paritätsbit fest auf 0
	SERIAL_PARITY1	Paritätsbit fest auf 1
Stop	Anzahl der Stop–Bits. Folgende Angaben sind zulässig:	
	SERIAL_STOP1	1 Stop–Bit
	SERIAL_STOP2	2 Stop–Bit
	SERIAL_STOP15	1,5 Stop–Bit

Beschreibung

`SerialInit` dient zur Initialisierung der durch *Handle* bezeichneten Unit eines Treibers für eine serielle Schnittstelle. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde. Der Parameter *Baud* gibt die Baudrate an. Die Parameter *Data* und *Stop* geben die Anzahl der Daten– bzw. Stop–Bits an. Der Parameter *Parity* dient zur Steuerung der Parität.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch **SerialClose, SerialInitEx, SerialOpen**

SerialInitEx

Funktion **Erweiterte Initialisierung der Unit**

Syntax

```
#include <serial.h>
int          SerialInitEx(
               RmIOHandle Handle,
               ulong Baud,
               uint Data,
               uint Parity,
               uint Stop,
               ulong BufferSize,
               uchar SendStopMode,
               uchar SendStop1,
               uchar SendStop2,
               ulong SendDelay,
               uchar RecStopMode,
               uchar RecStop1,
               uchar RecStop2,
               ulong RecTimeout,
               ulong RecLen);
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
Baud	Baudrate als Zahlenwert (z.B. 19200)
Data	Anzahl der Daten–Bits als Zahlenwert (z.B. 8)
Parity	Parität SERIAL_PARITYNONE keine Prüfung SERIAL_PARITYEVEN gerade Parität SERIAL_PARITYODD ungerade Parität SERIAL_PARITY0 Paritätsbit fest auf 0 SERIAL_PARITY1 Paritätsbit fest auf 1
Stop	Anzahl der Stop–Bits. Folgende Angaben sind zulässig: SERIAL_STOP1 1 Stop–Bit SERIAL_STOP2 2 Stop–Bit SERIAL_STOP15 1,5 Stop–Bit
BufferSize	Größe des Hintergrundpuffers (Anzahl Zeichen)

Parametername	Bedeutung
SendStopMode	Angabe, welches Stopzeichen Schreibaufträge beenden soll. Das/die Stopzeichen werden nach den Anwenderdaten übertragen. SERIAL_SENDSTOP_OFF Keine Stopzeichen verwenden. SERIAL_SENDSTOP_CHAR_1 Stopzeichen 1 verwenden SERIAL_SENDSTOP_CHAR_1_2 Stopzeichen 1 und 2 verwenden, d.h. Abbruch wenn das 1. Stopzeichen gefolgt vom 2. Stopzeichen auftritt.
SendStop1	1. Stopzeichen für Schreibaufträge
SendStop2	2. Stopzeichen für Schreibaufträge
SendDelay	Minimale Pause zwischen zwei Schreibaufträgen (in ms). Angabe von 0 deaktiviert die Funktion
RecStopMode	Angabe, welches Stopzeichen Leseaufträge beenden soll. Das/die Stopzeichen werden nicht in den Anwenderpuffer übertragen. SERIAL_RECSTOP_OFF Keine Stopzeichen verwenden. SERIAL_RECSTOP_CHAR_1 Stopzeichen 1 verwenden SERIAL_RECSTOP_CHAR_1_2 Stopzeichen 1 und 2 verwenden, d.h. Abbruch wenn das 1. Stopzeichen gefolgt vom 2. Stopzeichen auftritt. SERIAL_RECSTOP_LEN Lese–Auftrag beenden, wenn die durch *RecLen* definierte Anzahl von Zeichen eingelesen wurde.
RecStop1	1. Stopzeichen für Leseaufträge
RecStop2	2. Stopzeichen für Leseaufträge
RecTimeout	Maximale Zeitspanne, die zwischen dem Lesen zweier Zeichen verstreichen darf (ms). Wird dieser Zeitraum überschritten, wird der Leseauftrag abgebrochen. Angabe von 0 deaktiviert die Funktion
RecLen	Anzahl der Zeichen, nach denen Leseaufträge beendet werden

Beschreibung

`SerialInitEx` dient zur erweiterten Initialisierung der durch *Handle* bezeichneten Unit eines Treibers für eine serielle Schnittstelle. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Der Parameter *Baud* gibt die Baudrate an. Die Parameter *Data* und *Stop* geben die Anzahl der Daten– bzw. Stop–Bits an. Der Parameter *Parity* dient zur Steuerung der Parität.

Der Parameter *BufferSize* gibt die Größe des Hintergrundpuffers an.

Die Parameter *SendStopMode*, *SendStop1* und *SendStop2* legen die Verwendung und Art von Stopzeichen für Schreibaufträge fest. Der Parameter *SendDelay* gibt die minimale Pause zwischen zwei Schreibaufträgen an.

Die Parameter *RecStopMode*, *RecStop1* und *RecStop2* legen die Verwendung und Art von Stopzeichen für Leseaufträge fest. Der Parameter *RecTimeout* gibt die Zeit an, nach der ein Leseauftrag abgebrochen wird.

Der Parameter *RecLen* gibt die Anzahl der Zeichen an, nach denen Leseaufträge beendet werden.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialClose, SerialInit, SerialOpen

SerialOpen

Funktion

Verbindung zu einer Unit eines Treibers herstellen

Syntax

```
#include <serial.h>
int          SerialOpen(
               const char *UnitName,
               RmIOHandle *Handle );
```

Parameter

Parametername	Bedeutung
UnitName	Name der Unit im RMOS Betriebsmittelkatalog. Dieser Name wird beim Erzeugen der Unit vergeben.
Handle	Zeiger auf Variable vom Typ RmIOHandle, in der ein Deskriptor zum Ansprechen der Unit abgelegt wird.

Beschreibung

`SerialOpen` stellt die Verbindung zu der mit *UnitName* bezeichneten Unit her.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialClose

SerialPutChar

Funktion

Einzelnes Zeichen zur Unit schreiben

Syntax

```
#include <serial.h>
int          SerialPutChar(
               RmIOHandle Handle,
               char Char );
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
Char	Zeichen, das geschrieben werden soll

Beschreibung

`SerialPutChar` schreibt das Zeichen *Char* auf die durch *Handle* gekennzeichnete Unit. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialGetChar, SerialGetString, SerialPutString

SerialPutString

Funktion

Zeichen auf die Unit schreiben

Syntax

```
#include <serial.h>
int          SerialPutString(
               RmIOHandle Handle,
               char *String,
               ulong MaxLen );
```

Parameter

Parametername	Bedeutung
Handle	Deskriptor
String	Adresse eines Speicherbereichs mit den zu schreibenden Zeichen.
MaxLen	Anzahl von Zeichen, die geschrieben werden sollen.

Beschreibung

`SerialPutString` schreibt *MaxLen* Zeichen von der Adresse *String* auf die durch *Handle* gekennzeichnete Unit. *Handle* ist ein Deskriptor, der mit `SerialOpen` erzeugt wurde.

Rückgabewert

RM_OK Funktion erfolgreich ausgeführt

Fehlercodes

Fehlermeldungen siehe “Fehlermeldungen ladbarer Treiber”

Hinweis

Dieser Aufruf ist nur für den Treiber für serielle Schnittstellen SER8250.DRV einsetzbar. Zusätzlich muß beim Linken die Bibliothek RMFSERB.LIB eingebunden werden.

Siehe auch

SerialGetChar, SerialGetString, SerialPutChar

x_dos_cpyin

Funktion

Einen Speicherbereich aus dem Transferpuffer allokieren und Daten hineinkopieren
NICHT für weitere Entwicklungen vorgesehen!

Syntax

```
#include <rm3dos.h>
char *          x_dos_cpyin  (
                  char *buffer,
                   int  len ) ;
```

Parameter

Parametername	Bedeutung
buffer	Zeiger auf die Daten, die in den Transferpuffer kopiert werden sollen. Soll der Speicherbereich nur allokiert werden, muß hier NULL eingetragen werden.
len	Länge des zu allokierenden Speicherbereichs in Byte.

Beschreibung

Diese Funktion allokiert als erstes einen Speicherbereich aus dem Transferpuffer. Danach werden die Daten in den allokierten Speicher kopiert.

Der Transferpuffer liegt unterhalb von 1 Mbyte und wird für den Datenaustausch mit der DOS-Task und mit DOS-/BIOS-Systemaufrufen benötigt.

Der allokierte Speicherbereich kann mit der Funktion `x_dos_cpyout` wieder freigegeben werden.

Die Größe des Transferpuffers kann beim Laden des speicherresidenten DOS-Programms RM3_TSR angegeben werden. Sie beträgt maximal 30 Kbyte. Damit immer Speicher zur Verfügung steht, sollen nicht benötigte Bereiche des Transferpuffers immer freigegeben werden.

Der Transferpuffer wird nach einem Warmstart neu initialisiert. Der allokierte Speicherbereich ist danach wieder frei. Unter Umständen liegt der Transferpuffer nun an einer ganz anderen Stelle und die Daten sind verloren.

Rückgabewert

Der Rückgabewert ist ein Pointer.

Ist bei diesem das Bit 31 gesetzt, also der Wert negativ, so konnte der geforderte Speicher nicht allokiert werden. Die unteren 16 Bit geben in diesem Fall den größten momentan verfügbaren Speicherbereich an.

Ist der Pointer positiv (Bit 31=0) enthält er die physikalische Anfangsadresse des allokierten Speicherbereichs.

Hinweis

Der Rückgabewert der Funktion kann in dieser Form nicht an MS–DOS oder an das BIOS weitergegeben werde. Der Pointer muß zuerst in einen Real-Mode–Pointer, bestehend aus Segment plus Offset, konvertiert werden.

Beispiel

```
const char filename=”c:\clistart.bat”;
char *pptr;
```

Beispiel

```
              unsigned short             dos_seg;
unsigned short             dos_off;

pptr=x_dos_cpyin (filename,strlen(filename));

dos_seg=(unsigned short) (pptr>>4)
dos_off=(unsigned short) (pptr&0xF)
```

Siehe auch

x_dos_cpyout

x_dos_cpyout

Funktion

Daten aus einem zuvor allokierten Bereich im Transferpuffer kopieren und anschließend den Bereich freigeben
NICHT für weitere Entwicklungen vorgesehen!

Syntax

```
#include <rm3dos.h>
int          x_dos_cpyout (
               char *addr,
               char *buffer,
               int len) ;
```

Parameter

Parametername	Bedeutung
addr	Zeiger auf den Speicherbereich im Transferpuffer. Dieser Wert entspricht dem Rückgabewert der Funktion `x_dos_cpyin`.
buffer	Zeiger auf den Bereich, in den die Daten aus dem Transferpuffer kopiert werden sollen. Ist dieser Wert NULL, wird der Speicherbereich, auf den addr zeigt freigegeben, ohne die Daten zu kopieren.
len	Länge des zu kopierenden Speicherbereichs in Byte. Ist dieser Wert kleiner als die tatsächliche Länge des allokierten Bereichs, wird trotzdem der gesamte Bereich freigegeben.

Beschreibung

Diese Funktion kopiert zuerst Daten aus einem Speicherbereich im Transferpuffer. Anschließend wird der Bereich freigegeben.

Rückgabewert

Länge des freigegeben Bereichs.

Ist dieser Wert 0, so wurde ein ungültiger Wert im Parameter *addr* übergeben.

Siehe auch

x_dos_cpyin

Index

E

F

I

K

L

M

N

O

P

R

S

T

U

V

W

Z