SIEMENS

SIMATIC

System Software for M7-300 and M7-400 System and Standard Functions, Volume 1

Reference Manual

This manual is part of the documentation package with the order number:

6ES7802–0FA14–8BA0

C79000–G7076–C852–02

Safety Guidelines

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

Danger

indicates that death, severe personal injury or substantial property damage **will** result if proper precautions are not taken.

Warning

indicates that death, severe personal injury or substantial property damage **can** result if proper precautions are not taken.

Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken.

Note

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Qualified Personnel

Only **qualified personnel** should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and sys–tems in accordance with established safety practices and standards.

Correct Usage

Note the following:

Warning

This device and its components may only be used for the applications described in the catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safetly if it is transported, stored, set up and installed cor-rectly, and operated and maintained as recommended.

Trademarks

SIMATIC®, SIMATIC HMI® and SIMATIC NET® are registered trademarks of SIEMENS AG.

Some of the other designations used in these documents are also registered trademarks; the owner's rights may be violated if they are used by third parties for their own purposes.

Siemens AG
Automation and Drives Group
Industrial Automation Systems
P.O.Box 4848, D- 90327 Nuremberg

Preface

Purpose

This manual supports you when programming M7 300 and M7 400 automation computers in C under the M7 RMOS32 operating system. It provides you with detailed information on the range of functions for the call interface of M7 RMOS32. The information contained in the manual includes:

- Notations and data types
- Functional classification of the various calls
- Data structures used
- Error codes and messages
- Detailed information on the function calls

Audience

This manual is intended primarily for C programmers of M7 300 and M7 400 automation computers.

Scope of this Manual

This manual is valid for M7 300 and M7 400 automation computers with the system software M7–SYS RT from V 4.0.

Scope of the Documentation Package

The system software for automation computers M7 300 and M7 400 with M7 RMOS32 is documented in several manuals, which can be ordered separately from each product. The manuals are listed in the following table.

Manual	Contents
System Software for M7-300/400 Installation and Operation, User Manual	Installation and operation of M7-300/400 automation computers.
System Software for M7-300/400 Program Design, Programming Manual	Design and creation of C/C++ programs
System Software for M7-300/400 System and Standard Functions, Reference Manual	Detailed information for programming with M7 RMOS32.
System Software for M7-300/400 Writing Loadable Drivers Electronic Manual	Designing and writing loadable device drivers for M7 RMOS32, programming and reference information.

How to Use this Manual

This Reference Manual supports you primarily when programming applications for M7 RMOS32. It is your main reference document for programming, testing and checking the source code. The manual is divided into two volumes containing the following:

Volume 1

Function groups

Chapter 1 provides an introduction and presents the programming functions in logical order. If you are looking for a function to perform a specific task, you can find it here.

This chapter also describes the conditions required for the use of the individual groups of calls. You will find a detailed description of the individual functions in Chapters 5 and 6 of Volume 1 and Chapters 1 to 3 of Volume 2 .

Type identifiers

The second chapter contains the main type identifiers used when programming. It lists the identifiers for the system messages, S7 objects and data types used.

Data structures

The third chapter describes the data structures used in the RMOS API, M7 API and socket calls.

Error codes and messages

The fourth chapter explains the error codes and messages returned by the M7 RMOS32 kernel and the individual function calls.

Description of the function calls

Chapters 5 and 6 provide a detailed description, in alphabetical order, of the M7 API and RMOS API calls.

Volume 2

Libraries

Chapters 1, 2 and 3 provide a detailed description, in alphabetical order, of the C runtime librarycalls, the socket library calls and miscellaneous function calls.

Index

Each volume contains an index which helps you to find text relating to important topics quickly.

Manual and Online Help

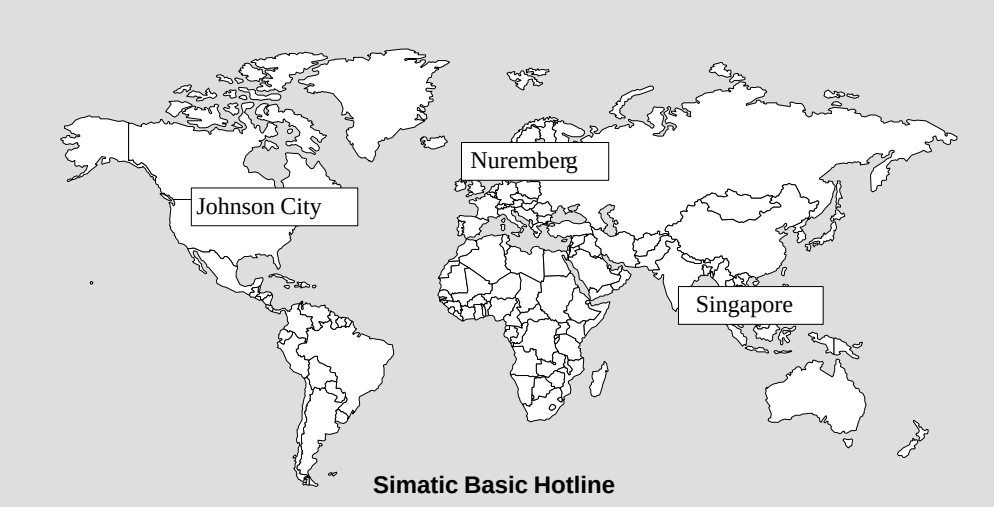
This manual is available both in printed form and in electronic format as part of the SIMATC Manual Collection. Its contents is also available in the online help file M7SYS40B.HLP in the S7BIN directory of STEP 7. You can include this file in the search range of the OpenHelp function of the Borland IDE for context–sensitive support during programming.

Feedback

We need your help to enable us to provide you and future M7-SYS users with optimum documentation. If you have any questions or comments on this *manual* or the *online help*, please fill in the remarks form at the end of the manual and return it to the address shown on the form. We would be grateful if you could also take the time to answer the questions giving your personal opinion of the manual.

SIMATIC Customer Support Hotline

Contactable worldwide round the clock:



Nuremberg

SIMATIC BASIC Hotline

Local time: Mo.-Fr. 8:00 to 18:00

Phone: +49 (911) 895-7000

Fax: +49 (911) 895-7002

E-Mail: simatic.support@nbgm.siemens.de

SIMATIC Premium Hotline

(Calls billed, only with SIMATIC Card)

Time: Mo.-Fr. 0:00 to 24:00

Phone: +49 (911) 895-7777

Fax: +49 (911) 895-7001

Johnson City

SIMATIC BASIC Hotline

Local time: Mo.-Fr. 8:00 to 17:00

Phone: +1 423 461-2522

Fax: +1 423 461-2231

E-Mail: simatic.hotline@sea.siemens.com

Singapore

SIMATIC BASIC Hotline

Local time: Mo.-Fr. 8:30 to 17:30

Phone: +65 740-7000

Fax: +65 740-7001

E-Mail: simatic@singnet.com.sg

SIMATIC Customer Support Online Services

The SIMATIC Customer Support team provides you with comprehensive additional information on SIMATIC products via its online services:

- You can obtain general current information:
  - On the **Internet** at `http://www.ad.siemens.de/simatic`
  - Using **fax polling** no. 08765-93 02 77 95 00
- Current Product Information leaflets and downloads which you may find useful for your product are available:
  - On the **Internet** at `http://www.ad.siemens.de/support/html_00/`
  - Via the **Bulletin Board System** (BBS) in Nuremberg (*SIMATIC Customer Support Mailbox)* at the number +49 (911) 895-7100.

    To access the mailbox, use a modem with up to V.34 (28.8 kbps), whose parameters you should set as follows: 8, N, 1, ANSI, or dial in using ISDN (x.75, 64 kbps).

SIMATIC Training Center

Siemens also offers a number of training courses to introduce you to the SIMATIC S7 and M7 automation systems. Please contact your regional training center or the central training center in Nuremberg, Germany for details:

D-90327 Nuremberg, Tel. (+49) (911) 895 3154.

Further Support

If you have any further questions about SIMATIC products, please contact your Siemens partner at your local Siemens representative's or regional office. You will find the addresses in our catalogs and in Compuserve (`go autforum`).

Table of Contents

Tables

Function Groups

1

In this Chapter

1.1 Overview

What is Described in this Chapter?

The following sections describe the functions used when programming with M7-SYS RT. The individual calls are subdivided into logical function groups.

Libraries and Header Files

If functions from a group are to be used in M7 RMOS32 tasks, the header file belonging to the group must be included and the corresponding library linked, as specified in the following table:

Table 1-1 Overview of Function Groups

Function Group	Header File	Library
RMOS API functions	RMAPI.H	RMFHLI.LIB
M7 API functions	M7API.H	M7APIBL.LIB
MS-DOS Interface functions	RM3DOS.H	RMFDOSIB.LIB
C Library functions	ANSI-compliant	RMFCRIFB.LIB
Socket Interface functions	SOCKET.H	RMFSK2IB.LIB
Serial Interface functions	SERIAL.H	RMFSER.LIB
Other functions	MISC86.H	RM3BCC.LIB

1.2 RMOS API Functions

1.2.1 Information on RMOS API Functions

General Information

M7 RMOS32 presents a pure function interface for accessing the services of the M7 RMOS32 kernel. The functions return values which indicate whether or not the functions have been successfully executed. Special calls also return additional information.

RMAPI.H is included as the header file with the prototypes for the API. The file is automatically included when creating M7 RMOS32 applications in the integrated development environment. RMAPI.H in turn includes the files **RMTYPES.H** (RMOS-API-specific type definitions) and **RMDEF.H** (general definitions such as error codes, etc.)

Note

M7 RMOS32 applications are created in the FLAT memory model, that is all pointers consist only of a 32-bit offset.

There is no protection in the FLAT memory model for address areas of external tasks or tasks of the M7 RMOS32 kernel. Special care should therefore be exercised when using pointers, if problems are to be avoided.

Information for Programming in C

Examples of code in C are used to illustrate the RMOS API calls.

The C interface is described by RMAPI.H in the INC directory. All the function prototypes of the RMOS API are contained there. The files RMDEF.H and RMTYPES.H are also included. RMDEF.H contains the define constants and RMTYPES.H contains the data types and structures for programming the system calls.

In order to prevent problems arising from parameter errors, the defined constants from RMDEF.H should be used.

The parameters are always passed on the stack; the return value contains the error code of the RMOS API call. If no error occurs, RM_OK (=0) is returned. In the event of an error, a value greater than 0 is returned. Certain RMOS API calls also have negative return values; these are used for additional information. For example, RmSetFlag returns RM_FLAG_ALREADY_SET if the flag was already set.

Example of an RMOS-API-Call

This call allocates a memory area of 1000 bytes which is not freed automatically and is thus not allocated to a specific task. If insufficient memory is available, the system does not wait for memory to be released.

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

General Data Types

The following data types can be used for programming RMOS API calls.

Table 1-2 General Data Types of C

Data Type	Description
char	Character : 8 bits
short	Integer: 16 bits
int	Integer: 32 bits
long	Integer: 32 bits
void *	Pointer (FLAT): 32 bits
enum	Enumerator type: 32 bits
float	Floating-point number: 32 bits
double	Floating-point number: 64 bits

The RMOS API-specific data types (which are shown in Table 3-1 and defined in header file RMTYPES.H) should be used, in addition to the general C data types, for RMOS API calls.

Interrupt Numbers

In all RMOS API calls for checking, installing and deinstalling interrupt handlers, the interrupt number can be specified in two different ways:

1. Number between 0 and 255
   The interrupt is treated as a software interrupt.
2. IRQ<n>
   The number <n> is entered directly, e.g. IRQ1, IRQ2. The interrupt is interpreted as a hardware interrupt.
   The values IRQ1, IRQ2, etc. are defined in an include file. The IRQ(x) macro can be used to pass the IRQ number to a variable. The value of (x) can be between 0 and the highest available interrupt. The value range of 0..15 is valid on the PC.

Information for Timer Programming

If you use timeout values in your program, these are entered in the timer queue according to their execution time (in the order of the timer ticks). If several timeout requests are registered for the same timer tick, these are executed according to the *Last In First Out* principle.

If a timer tick lies between two timeout requests of the same length, these requests are distributed across different timer ticks.

Example:
Timeout requests 1, 2, 3 within one timer tick; timeout requests 4, 5, 6 within the next timer tick. The order in the timer queue is 3, 2, 1, 6, 5, 4.

In order to ensure that all timer calls occur within one timer tick, you should proceed as follows:

1. Set a very high priority for the task (the highest system priority), to prevent it from being interrupted by other tasks.
2. Initiate a pause call with 0 for synchronization with the next timer tick.
3. Inititate timeout requests.
4. Reset the task priority to the initial value.

Please note that the entire process must be executed completely within a timer tick.

1.2.2 Brief Description of the RMOS API Functions

Overview

In the form of a C interface, the RMOS API provides M7 RMOS32 applications with all the functions necessary to implement a multitasking system. RMOS API functions present the interface to the M7 RMOS32 kernel.

You will find a detailed description of these functions in Chapter 6.

Memory Managament

The following table lists all the functions for memory management, together with a brief description.

Table 1-3 Functions for Memory Management

Function	Brief Description
RmAlloc	Allocate memory from heap
RmCreateMemPool	Create memory pool from heap
RmDeleteMemPool	Delete memory pool
RmFree	Free memory area
RmFreeAll	Free all memory areas of a task
RmGetMemPoolInfo	Get memory pool information
RmGetSize	Get the size of a memory area
RmMapMemory	Map physical memory

Table 1-3 Functions for Memory Management

Function	Brief Description
RmMemPoolAlloc	Allocate memory area from memory pool
RmReAlloc	Increase size of memory area

Task Control

The following table 1-4lists all the functions you can use for task control, together with a brief description.

Table 1-4 Functions for Task Control

Function	Brief Description
RmActivateTask	Set task to READY state
RmCreateTask	Create task
RmCreateTaskEx	Create task
RmCreateChildTask	Create child task
RmDeleteTask	Terminate calling task (and delete)
RmDisableScheduler	Disable scheduler
RmEnableScheduler	Enable scheduler
RmEndTask	End calling task (without deletion)
RmGetTaskID	Get the ID of a task
RmGetTaskPriority	Get task priority
RmGetTaskState	Get task state
RmKillTask	Set task to DORMANT or NOTEXISTENT state
RmPauseTask	Pause calling task
RmQueueStartTask	Add task to queue. The task is started immediately it switches to the DORMANT state
RmRestartTask	Terminate the calling task and automatically start it again after a given interval
RmResumeTask	Resume task execution after an interval commencing with `RmPauseTask`
RmSetTaskPriority	Change the priority of a task
RmStartTask	Request the start of a task currently in the DORMANT state
RmSuspendTask	Set task to BLOCKED state

Resource Management

The following table lists all the functions you can use for the management of resources, together with a brief description.

Table 1-5 Functions for Cataloging Resources

Function	Brief Description
RmCatalog	Enter resources in resource catalog
RmGetEntry	Get entry (ID) in resource catalog
RmGetName	Get name in resource catalog
RmList	List entries in resource catalog
RmUncatalog	Delete entries from resource catalog
RmGetAbsTime	Get absolute system time

Message Exchange

The following table lists all the functions you can use for message exchange, together with a brief description.

Table 1-6 Functions for Message Exchange

Function	Brief Description
RmCreateMessageQueue	Create message queue
RmDeleteMessageQueue	Delete message queue
RmReadMessage	Read message from message queue
RmSendMessage	Store message in message queue
RmSetMessageQueueSize	Limit the length of the message queue

Mailboxes

The following table lists all the functions you can use for message exchange via mailboxes, together with a brief description.

Table 1-7 Functions for Message Exchange via Mailboxes

Funktion	Brief Description
RmCreateMailbox	Create mailbox
RmDeleteMailbox	Delete mailbox
RmReceiveMail	Read message from mailbox
RmSendMail	Store message in mailbox
RmSendMailCancel	Cancel delayed message storage
RmSendMailDelayed	Delayed message storage in mailbox
RmSetMailboxSize	Limit length of mailbox

Event Flags

The following table lists all the functions you can use for coordination with event flags, together with a brief description.

Table 1-8 Functions for Coordination with Event Flags

Function	Brief Description
RmCreateFlagGrp	Create flag group
RmDeleteFlagGrp	Delete flag group
RmGetFlag	Test bit in flag group
RmResetFlag	Reset bit in flag group
RmSetFlag	Set bit in flag group
RmSetFlagDelayed	Set bits in flag group after interval

Semaphore Handling

The following table lists all the functions you can use for semaphore handling, together with a brief description.

Table 1-9 Functions for Semaphore Handling

Function	Brief Description
RmCreateBinSemaphore	Create semaphore
RmDeleteBinSemaphore	Delete semaphore
RmGetBinSemaphore	Assign semaphore
RmReleaseBinSemaphore	Release semaphore

Interrupt Handling

The following table lists all the functions you can use for interrupt handling, together with a brief description.

Table 1-10 Functions for Interrupt Handling

Function	Brief Description
RmGetIntHandler	Get current interrupt handler
RmSetIntDefHandler	Deinstall interrupt handler
RmSetIntISHandler	Install interrupt handler for I and S states
RmSetIntMailboxHandler	Install mailbox interrupt handler
RmSetIntTaskHandler	Install interrupt handler for task start

Loadable Drivers

The following table lists all functions for loadable drivers with a brief description.

Table 1-11 Functions for loadable drivers

Function	Brief Description
RmIOClose	Close Unit
RmIOControl	Control functions for loadable drivers
RmIOOpen	Open Unit
RmIORead	Read from Unit
RmIOWrite	Write on Unit
RmLoadDevice	Load driver

Other Calls

The following table lists all other RMOS API calls, together with a brief description.

Table 1-12 Other Functions

Function	Brief Description
get2ndparm	Read EBX start parameter of task
getdword	Read start parameter of task in long format
getparm	Read start parameter of task as pointer

1.2.3 RMOS API Calls in MS-DOS Applications

General Information

RMOS also provides an API which can be used by MS-DOS applications. This enables DOS applications to issue system calls to the RMOS kernel (not to M7 servers!), to start an RMOS task, for example, or send messages to a mailbox or the message queue of a task.

The RMOS API for MS-DOS applications is not for further development!

Header Files and Conventions

MS-DOS programs which use the interface must include the prototypes of header file **RMAPI.H**.

MS-DOS can only use the 16-bit real mode call interface under M7 RMOS32. The definition of data formats, types and structures conforms to the real mode programming of MS-DOS.

C prototypes and macros of the RMOS API interface are defined in the file RMAPI.H or the files **RMDEF.H** and **RMTYPES.H.** The **RM3** switch is used to select whether the file for M7 RMOS32 or for MS-DOS applications is used.

Including in DOS-Programs

Consequently, the switches must be set, as shown below, before the RMAPI.H include statement in an MS-DOS program:

```
#define RM3 0
#include “RMAPI.H”
```

Libraries

An appropriate interface library must be included in the link statement for the program. This is the library **DOSHLIB.LIB** for MS-DOS programs.

Data Conversion

M7 RMOS32 converts the parameters internally to the M7 RMOS32 format on an RMOS API call from a DOS program.

Data types which are 16 bits wide, and 32 bits wide with M7 RMOS32, are “zero extended”, that is bits 31 to 16 are set to 0 and transmitted to the RMOS kernel.

Interrupt Number

An RMOS API call is invoked from an MS–DOS program using a software interrupt. The interrupt vector used is configured permanently as **79H**. This interrupt may therefore not be reassigned by MS–DOS applications.

RMOS API Calls Which are Not Supported

The following table lists the RMOS API calls which may not be used in MS-DOS programs. If they are used, the call returns an error message.

Table 1-13 RMOS API Calls Which are Not Supported

RMOS API Calls	Cause
RmAlloc, RmMemPoolAlloc, RmFree, RmFreeAll, RmReAllocMem, RmMapMemory	It is not permitted for RMOS to manage a memory pool within the memory area addressed by MS-DOS. Therefore each RMOS memory pool must be situated above this area.
RmSetISHandler, RmSetIntTaskHandler, RmSetIntMailboxHandler, RmSetIntDefHandler	The RMOS-API calls for interrupt management are used to set interrupt vectors in the RMOS environment The functions available under MS-DOS must be used in order to change or enter an interrupt vector in the MS-DOS environment.
RmEndTask, RmRestartTask	An MS-DOS program cannot terminate itself with these calls.
RmCreateTask	An MS-DOS program cannot create another task, since no task can be created within the memory area managed by MS-DOS.
RmReadMessage, RmSendMessage, RmCreateMessageQueue, RmDeleteMessageQueue,	Using these calls, specifical operating system pointers are transmited , which may be not converted automatically, Instead of this the communication may be effected by mailboxes (see table 1–7)

Special Properties of RMOS API Calls

The following table shows the special properties of RMOS API calls in an MS-DOS environment. Failure to handle these calls correctly will cause system errors.

Table 1-14 Special Properties of RMOS API Calls

RMOS API Call	Cause
RmDeleteTask	Although an MS-DOS program can delete another RMOS task with this call, it cannot delete itself. Calls with Task_ID equal to RM_OWN_TASK are illegal.
RmSetTaskPriority	Although an MS-DOS program can change the priority of another RMOS task with this call, it cannot change its own priority. Calls with Task_ID equal to RM_OWN_TASK are illegal.

Communication using Mailbox Services

Please also note the following points for communication between RMOS and MS-DOS programs via mailboxes (see Table 1-7):

The mailbox call RmSendMail transfers the contents of a “3-word buffer” (message). This buffer is 12 bytes long in M7 RMOS32.

If the `RmSendMail` call is issued by an RMOS task under M7 RMOS32, a 12-byte data area is also transferred internally to the mailbox addressed.

If an MS-DOS program now reads the message from the mailbox, 12 bytes are also transferred to the memory area of the MS-DOS program. You should therefore make sure that the “3-word buffer” in an MS-DOS program is also 12 bytes in length.

A pointer in a message is not converted by the RMOS kernel, that is a flat pointer (linear address under M7 RMOS32) is not converted to a real mode pointer (physical address under MS-DOS).

Note

If a blocking call is issued within the MS-DOS program, the DOS task, that is the entire DOS machine, is blocked (task state: BLOCKED).

1.3 M7 API Functions

1.3.1 Information on M7 API Functions

Conventions and Header Files for M7 RMOS32 Applications

M7 RMOS32 programs must include header filei **M7API.H** as the header file for the prototypes of the functions.

You will also find all the data type and structure definitions and the error codes in M7API.H.

General Data Types of the M7 API

In order to facilitate future porting of programs to other systems, the M7 API environment also uses its own type definitions instead of machine-specific data type identifiers such as int or long. The data types are defined in header file M7API.H (see Table 3-2)

1.3.2 Brief Description of the M7 API Functions

Overview

The M7 API provides all the functions necessary for solving an automation task to the M7 RMOS32 applications in the shape of a C interface.

As well as access to the process I/Os, the M7 API presents functions for the management of internal S7 objects, calls for communication with other automation components, and further functions for the transparent integration of your M7 automation computer in an S7 programmable controller system.

You will find a detailed description of these functions in Section 5.

Initialization

The following table shows the function for task-specific initialization of the M7 API.

Table 1-15 Function for Initialization

Function	Brief Description
M7InitAPI	Initialize M7 API

Access to Process I/Os

The following table lists all the functions you can use to access process I/Os, together with a brief description.

Table 1-16 Functions for Access to Process I/Os

Function	Brief Description
M7ClearPI	Clear process image
M7LoadBit	Load bit from process image
M7LoadByte	Load byte from process image
M7LoadDWord	Load doubleword from process image
M7LoadDirect	Read data direct from I/O area
M7LoadDirectByte	Read byte direct from I/O
M7LoadDirectDWord	Read doubleword direct from I/O
M7LoadDirectWord	Read word direct from I/O
M7LoadISAByte	Read byte from ISA bus I/O
M7LoadISADWord	Read doubleword from ISA bus I/O
M7LoadISAWord	Read word from ISA bus I/O
M7LoadPII	Update process image of inputs
M7LoadRecord	Read data record from signal module
M7LoadRecordEx	Read data record from signal module
M7LoadWord	Load word from process image
M7StoreBit	Overwrite bit in process image
M7StoreByte	Overwrite byte in process image
M7StoreDWord	Overwrite doubleword in process image
M7StoreDirect	Transfer data direct to I/O area
M7StoreDirectByte	Write byte direct to I/O
M7StoreDirectDWord	Write doubleword direct to I/O
M7StoreDirectWord	Write word direct to I/O
M7StoreISAByte	Write byte to ISA bus I/O
M7StoreISAWord	Write word to ISA bus I/O
M7StoreISADWord	Write doubleword to ISA bus I/O
M7StorePIQ	Update I/O from process image of outputs
M7StoreRecord	Transfer data record to signal module
M7StoreWord	Overwrite word in process image

FRB Handling

The following table lists the calls for the general handling of FRBs (Function Request Blocks).

Table 1-17 Functions for FRB Handling

Function	Brief Description
M7GetFRBErrCode	Get error code from FRB header
M7GetFRBTag	Get tag from FRB header
M7SetFRBTag	Set tag in FRB header

Alarm Handling (Slave Functions)

The following table lists all the functions for sending alarms and checking the alarm handling status, together with a brief description.

Table 1-18 Functions for Alarm Processing (Slave Functions)

Function	Brief Description
M7GetDiagAlarmBusy	Check status of a diagnostics alarm
M7GetIOAlarmBusy	Check status of a process alarm
M7SendDiagAlarm	Send diagnostics alarm to CPU
M7SendIOAlarm	Send process alarm to CPU

Management of S7 Objects

The following table lists all the functions you can use for the management of S7 objects, together with a brief description.

Table 1-19 Functions for the Management of S7 Objects

Function	Brief Description
M7CreateObject	S7-Objekt erzeugen
M7DeleteObject	Delete S7 object from working memory and “permanent load memory”
M7GetFlags	Get access type for S7 object from OBJFRB
M7GetObjectInfo	Read information on data structure of S7 object
M7GetObjType	Get type identifier of S7 object from OBJFRB
M7GetPart	Get subarea number of S7 object from OBJFRB
M7LinkDataAccess	Link OBJFRB for access to S7 object
M7LocateObject	Move S7 object in working memory
M7Read	Read S7 data area
M7ReadBit	Read byte from S7 object

Table 1-19 Functions for the Management of S7 Objects

Function	Brief Description
M7ReadByte	Read word from S7 object
M7ReadWord	Read doubleword from S7 object
M7ReadDWord	Read doubleword from S7 object
M7ReadReal	Read floating point number from S7 object
M7RelocateObject	Transmit S7 object to object server
M7RemoveObject	Delete S7 object from “read-only” **or** “permanent load memory”
M7StoreObject	Store S7 object in “read-only” or “permanent load memory”
M7UnLinkDataAccess	Unlink OBJFRB for access to S7 object
M7Write	Copy user data to S7 data area
M7WriteBit	Overwrite bit in S7 object
M7WriteByte	Overwrite byte in S7 object
M7WriteWord	Overwrite word in S7 object
M7WriteDWord	Overwrite doubleword in S7 object
M7WriteReal	Overwrite floating point number in S7 object

Callback Function Calls for S7 Object Access

The following table lists all the functions you can use for linking callback functions and evaluating the access information within the callback function, together with a brief description.

Table 1-20 Calls for the Management of Callback Functions

Function	Brief Description
M7GetCBBitOffset	Get bit offset from CBFRB
M7GetCBBuffer	Get read or write buffer from CBFRB
M7GetCBByteOffset	Byte Offset aus CBFRB ermitteln
M7GetCBCount	Get number of elements from CBFRB
M7GetCBDataType	Get data type from CBFRB
M7GetCBFlags	Get access type from CBFRB
M7GetCBObjType	Get type identifier of S7 object from CBFRB
M7GetCBPart	Get subarea number of S7 object from CBFRB
M7LinkDataAccessCB	Link callback function for S7 object access
M7UnLinkDataAccessCB	Unlink callback function for S7 object access

Alarm Handling (Master Functions)

The following table lists all the functions you can use for alarm handling as master, together with a brief description.

Table 1-21 Functions for Alarm Handling

Function	Brief Description
M7ConfirmDiagAlarm	Confirm diagnostics alarm
M7ConfirmIOAlarm	Confirm process alarm
M7ConfirmSAlarm	Confirm of drawing/ streching
M7DPNormDiagnose	Get DP standard diagnostics for a DP station
M7GetDiagAlarmAddr	Get base address of module from DIAGFRB
M7GetDiagAlarmInfo	Get alarm information from DIAGFRB
M7GetDiagAlarmPType	Get I/O type of module from DIAGFRB
M7GetIOAlarmAddr	Get base address of module from IOFRB
M7GetIOAlarmMask	Get alarm mask from IOFRB
M7GetIOAlarmState	Get alarm information from IOFRB
M7GetIOAlarmPType	Get I/O type of module from IOFRB
M7GetPIErrorAddr	Get address of I/O type with transfer error
M7GetPIErrorPIType	Get I/O type with transfer error
M7GetZSAlarmAddr	Get base address of module from ZSFRB
M7GetZSAlarmIdent	Get identifier of a module
M7GetZSAlarmIMRBaddr	Get base address of IMR module, which was signed on for the alarm of drawing/ streching
M7GetZSAlarmMode	Get mode of module from ZSFRB
M7GetZSAlarmPType	Get I/O type of module from ZSFRB
M7GetZSAlarmRackNo	Get rack number from ZSFRB
M7LinkDiagAlarm	Link diagnostics alarm for handling
M7LinkIOAlarm	Sign on process alarm for working
M7LinkPIError	Initializise FRB for transfer of I/O type
M7LinkZSAlarm	Link ZS alarm for handling
M7UnLinkDiagAlarm	Unlink diagnostics alarm
M7UnLinkIOAlarm	Unlink process alarm
M7UnlinkPIError	Unlink FRB for transfer of I/O type
M7UnlinkZSAlarm	Unlink ZS alarm

Time Handling

The following table lists all the functions you can use for time handling, together with a brief description.

Table 1-22 Functions for Time Handling

Function	Brief Description
M7ConfirmPeriodicTimer	Confirm periodic time signal
M7GetLostPeriods	Check lost periodic time messages
M7GetPeriod	Get multiple of time base from TFRB
M7GetTime	Read out date/time
M7GetTimeBase	Get time base from TFRB
M7LinkDate	Link time-controlled time message
M7LinkOneShotTimer	Link one-shot time message
M7LinkPeriodicTimer	Link periodic time message
M7SetTime	Set date/time
M7UnLinkDate	Unlink time-controlled time message
M7UnLinkOneShotTimer	Unlink one-shot time message
M7UnLinkPeriodTimer	Unlink periodic time message

Operating State Handling

The following table lists all the functions you can use for monitoring the operating state, together with a brief description.

Table 1-23 Functions for Operating State Handling

Function	Brief Description
M7ConfirmTransition	Confirm operating state transition message
M7GetState	Check operating state
M7GetTSReason	Get reason for transition from TSFRB
M7GetTSType	Get operating state from TSFRB
M7LinkBatteryFailure	Link a BAFFRB for battery alarm
M7LinkState	Request a message on a specific operating state
M7LinkTransition	Request a message on a specific operating state transition
M7RequestState	Request operating state change
M7UnLinkBatteryFailure	Unlink BAFFRB for battery alarm
M7UnLinkState	Unlink a TSFRB linked with M7LinkState
M7UnLinkTransition	Unlink a TSFRB linked with `M7LinkTransition`

Free Cycle

The following table lists all the functions you can use for linking and unlinking the start-up, cycle control point, “free cycle” and cycle timeout, together with a brief description.

Table 1-24 Functions for Cycle Control Point and “Free Cycle”

Function	Brief Description
M7ConfirmCycle	Confirm a message
M7GetFSCTyp	Get type of message from FSCFRB
M7LinkCycle	Request message for start-up, cycle control point, “free cycle” and cycle timeout
M7RetriggerCycle	Retrigger cycle monitoring
M7UnLinkCycle	Unlink message for start-up, cycle control point, “free cycle” and cycle timeout

User LED Control

The following table shows the function for controlling the user LEDs on the M7:

Table 1-25 Functions for Controlling the User LED

Function	Brief Description
M7SetUserLED	Set user LED

Application Link Management

The following table lists the functions for initiating, aborting and legitimizing a communication bus application link, together with a brief description.

Table 1-26 Functions for Application Link Management

Function	Brief Description
M7GetConnStatus	Interrogate state of application link
M7KAbort	Close an application link
M7KInitiate	Set up application link
M7KPassWord	Password for functions with special protection level
M7GetPduSize	Get PDU size

Communications Functions

The following table lists the communications functions, together with a brief description.

Table 1-27 Communikations Functions

Function	Brief Description
M7PBKBrcv	Receive data from partner (double-ended communication function)
M7PBKBsend	Send data to partner (double-ended communication function)
M7PBKCancel	Cancel `M7PBKBsend` or `M7PBKBrcv` job
M7PBKGet	Request data from partner (single-ended communication function)
M7PBKIAbort	Close an application link
M7PBKIGet	Start asynchronous reading with a variable
M7PBKIPut	Sart asynchronous writing with a variable
M7PBKPrint	Send dates with a description of format
M7PBKPut	Send data to partner (single-ended communication function)
M7PBKResume	Request resume all user programs
M7PBKStart	Request start all user programs
M7PBKStatus	Check “virtual device status”
M7PBKStop	Request stop all user programs
M7PBKUrev	Uncoordinated receiving by planning connections
M7PBKUsend	Uncoordinated sending by planning connections
M7PBKXAbort	Close an application link
M7PBKXCancel	Stop actual job of receiving from M7PBKXrv
M7PBKXGet	Start asynchronous reading of a variable
M7PBKXPut	Start asynchronous writing of a variable

MMI Functions

The following table lists the MMI functions, together with a brief description.

Table 1-28 MMI Functions

Function	Brief Description
M7BUBCycRead	Set up MMI job for cyclical read
M7BUBCycReadDelete	Delete MMI job for cyclical read
M7BUBCycReadStart	Start MMI job for cyclical read
M7BUBCycReadStop	Stop cyclical read

Table 1-28 MMI Functions

Function	Brief Description
M7BUBRead	One-shot MMI variable read
M7BUBWrite	One-shot MMI variable write

Object Management Functions

The following table lists the functions of the object management system (OVS), together with a brief description.

Table 1-29 Object Management Functions

Function	Brief Description
M7OVSCompress	Compress load memory
M7OVSDelete	Delete a block
M7OVSFindFirst	Read out first entry from block directory
M7OVSFindNext	Read out next entry from block directory
M7OVSLinkln	Link a block
M7OVSMemMode	Set memory mode
M7OVSRead	Load a block
M7OVSSetObjectHeader	Set an S7 object header
M7OVSWrite	Copy a block

Time Functions

The following table lists the functions for reading and setting the time, together with a brief description.

Table 1-30 Functions for Reading/Setting the Time

Function	Brief Description
M7KReadTime	Read time via K bus
M7KWriteTime	Set time via K bus

Diagnostics Server

The following table lists the functions for the diagnostics server, together with a brief description.

Table 1-31 Functions for the Diagnostics Server

Function	Brief Description
M7DiagMode	Link for sending diagnostics events via K bus
M7SZLRead	Read out system state list via K bus
M7WriteDiagnose	Write user entry to local diagnostics server

Other Functions

The following table lists the other functions, together with a brief description.

Table 1-32 Other Functions

Function	Brief Description
M7GetCommRequest	Get job number from COMMFRB
M7GetCommStatus	Get data communication status from COMMFRB
M7KEvent	Fetch data after a message

1.4 DOS Interface Functions

Introduction

A memory area shared by M7 RMOS32 and MS-DOS is provided for fast exchange of large volumes of data. Attention should be paid, however, to the memory allocation between the MS-DOS and M7 RMOS32 operating systems and the different interpretation of address pointers (real-mode versus flat).

The DOS interface functions are not for further development!

Memory Management

Because MS-DOS applications can generally only access the address area below 1 Mbyte, but the private memory area of M7 RMOS32 tasks always lies over the 1 Mbyte threshold, M7 RMOS32 provides a special memory management system.

The TSR program RM3_TSR is used to create a transfer buffer below 1 Mbyte, from which RMOS tasks can allocate or release memory areas.

Header Files and Libraries

In order to use the memory management functions of the transfer buffer in M7 RMOS32 applications, you have to include the **RM3DOS.H** header file in your C programs.

You should also include the corresponding library **RMFDOSIB.LIB** in the link statement.

Brief Description of Functions

The following table lists all the functions that can be used by M7 RMOS32 tasks for communication with MS-DOS applications, together with a brief description.

You will find a detailed description of these functions in Chapter 6.

Table 1-33 Functions for DOS Communication

Function	Brief Description
x_dos_cpyin	Allocate a memory area from the transfer buffer and copy data to it.
x_dos_cpyout	Copy data from a previously allocated area in the transfer buffer and then release the area.

1.5 Functions of the C Runtime Library

1.5.1 Overview

Introduction

The preconfigured C runtime support presents all functions in compliance with the ANSI Draft International Standard ISO/IEC DIS 9899 (published in 1990).

Memory Management Requirements

The following memory capacity is required for any task which requests C runtime support:

- Approximately 1.3 Kbytes when calling the initialization function `xinitt`. This request is also made implicitly if a task uses C functions, but does not call `xinitt`.
- Approximately 1 Kbyte for each stream opened, if the size of the buffer for this stream has not been redimensioned with the functions `setvbuf` or

  The memory required for initialization and the stream buffers is taken from the **heap**.
- Each task which uses C functions from the runtime library also needs an additional stack area of approximately 1 Kbyte.

Initialization of the C Runtime Support

The function `xinitt` must also be called at the beginning of each task, in order to initialize task-specific data. Only then are the functions of the C library actually available.

Note

If the `xinitc` is missing, the initialization is performed automatically.

Functions of the C Library

The C library includes functions and macros organized according to the following criteria or function classes (these function classes are mainly identical to those used in technical documentation currently available):

- I/O operations, e.g. hard disk, terminal, printer, etc.
- Character management
- String operations
- Memory operations
- Memory allocation
- Mathematical functions
- Time and date functions
- Control functions
- Error handling
- Other functions

1.5.2 I/O Operations

Introduction

The largest function class of the C library is devoted to I/O operations. It contains functions used to perform input and output from C programs.

It also contains functions for checking and formating input/output and for file management. The functions are declared in the header files **IO.H** and **STDIO.H**.

Current Working Directory

The functions for opening, renaming and deleting files require the specification of a file or directory name.

This name always refers to a current working directory (CWD), whose allocation is task-specific. At first, however, the CWD is not initialized for a task. The initialization of the CWD is performed with the function `chdir`.

Rules for File and Directory Names

The following rules apply to the specification of file or directory names:

- The colon ':' is used to separate the drive name and the file or directory name. It may only be entered as the second or third character in a path name, and may not be entered at any other point. This means that drive names may only be one or two characters in length.
  Example: `R:TEST`
- The characters '\' and '/' are inserted between different directory names or between a directory and a file name.
  Example: `R:TEST\DIR1/DIR2\FILE`

- Path names that begin with a drive name (that is the second or third character is a colon ':' preceded by the name of a drive) are absolute path names.
  Example: `R:TEST\DIR1\DIR2\FILE`
- Path names that begin with a '\' or '/' are a special form of absolute path name. In this case, the drive letter only is taken from the CWD and placed in front of the specified path name.
  The CWD must always be initialized when using this type of path name.
  Example:
  ```
  R:TEST                   (CWD)
  \TEST2\DIR1\DIR2\FILE  (Specified path name)
  R:TEST2\DIR1\DIR2\FILE (Resultant path name)
  ```
  One variant is to specify the path "\" or "/". This addresses the core directory of the drive specified in the CWD, and can be used with the function `chdir("\\")` or `chdir("/")`.
- Path names that begin neither with '\' nor '/' are relative path names referring to the CWD.
  Example:
  ```
  R:TEST                   (CWD)
  DIR2\FILE              (Specified path name)
  R:TEST\DIR2\FILE       (Resultant path name)
  ```
- Path names that begin with . .<delimiter> are a special form of relative path name. In this case, the path refers to the parent directory of the CWD.
  Example:
  ```
  R:TEST\DIR1             (CWD)
  ..\DIR2\FILE           (Specified path name)
  R:TEST\DIR2\FILE       (Resultant path name)
  ```
  One variant is to specify the path "`..`". This addresses the directory which is one level closer to the drive name than the CWD, and can be used with the function `chdir("..")`.

Note

If the CWD has not been initialized for a task, absolute path names must be used.

As in the MS-DOS file system, it is not necessary to distinguish between upper and lower case letters.

Text Mode/Binary Mode

With the function `fopen`, `fduopen`, `freopen`, `fdureopen` and `open`, you specify whether a stream or a handle is to be opened in text mode or binary mode.

If a stream or handle is opened in text mode, all '\n' references (New Line) are converted to '\r\n' (Carriage Return - New Line) for write operations, and the opposite is performed for read operations (that is all '\r\n' references are converted to '\n').

No conversion takes place for streams or handles that are opened in binary mode.

NUL File

A NUL file can be opened which does not actually exist physically. All operations permitted with normal files can be performed when the NUL file is opened.

The difference is that read and write calls are terminated immediately without performing input/output operations.

All write operations on the NUL file are terminated without signaling an error (**errno**, **errno2**, etc.). Read operations always return **EOF** (End of File).

The **NUL** file is addressed if NUL (in any combination of upper and lower case letters) is specified for file or path names, (e.g. `fopen("NUL","w")`).

Table 1-34 Input/Output Operations

Call	Meaning	Header File
access	Check file access rights of user	IO.H
changevib	Change description block on a data storage device	IO.H
chdir	Change the CWD	DIRECT.H
checkpoint	Write back the (HSFS) buffer of a file	IO.H
chmod	Change the attributes of a file	IO.H
clearerr	Clear the error status of a stream	STDIO.H
close	Close an open file, a unit of a loadable driver or a socket	IO.H
createvib	Create new description block on a data storage device	IO.H
dismount	Dismount an HSFS device	IO.H
duread	Read character via RMOS driver	IO.H
duwrite	Write character via RMOS driver	IO.H
efsstop	Cancel connection between network unit and server unit	IO.H
efsuse	Set up connection between network unit and server unit	IO.H
fclose	Close a stream	STDIO.H
fduopen	Open a stream via RMOS driver	STDIO.H
fdureopen	Redirect stream to RMOS driver	STDIO.H

Table 1-34 Input/Output Operations

Call	Meaning	Header File
feof	Check whether end of file has been reached	STDIO.H
ferror	Check stream status	STDIO.H
fflush	Empty the buffer of a stream	STDIO.H
fgetc	Read character from a stream	STDIO.H
fgetpos	Get position in file	STDIO.H
fgets	Read string from a stream	STDIO.H
fileno	Return the file descriptor assigned to the specified stream	STDIO.H
fopen	Open stream	STDIO.H
fprintf	Write formatted output to a stream	STDIO.H
fputc	Write a character to a stream	STDIO.H
fputs	Write string to a stream	STDIO.H
fread	Read from a stream	STDIO.H
freopen	Change the file assigned to a stream	STDIO.H
fscanf	Read formatted input from a stream	STDIO.H
fseek	Position file pointer in a stream	STDIO.H
fsetpos	Set position in a file	STDIO.H
ftell	Return the distance from the file pointer to the start of file	STDIO.H
fwrite	Write to a stream	STDIO.H
getc	Read a character from a stream	STDIO.H
getchar	Read a character from stdin	STDIO.H
getcwd	Get CWD	DIRECT.H
gets	Read a string from a stream	STDIO.H
getvolumestatus	Get status information for a data storage device	IO.H
getw	Read a word from a stream	STDIO.H
ioctl	Execute control function for a socket or a unit of a loadable driver	IO.H
lseek	Position file pointer	IO.H
mkdir	Make directory	DIRECT.H
mount	Mount HSFS device	IO.H
open	Open file for reading and/or writing	IO.H
printf	Write formatted output to stdout	STDIO.H
putc	Write character to a stream	STDIO.H
putchar	Write a character to stdout	STDIO.H
puts	Write a string to a stream	STDIO.H
putw	Write a word to a stream	STDIO.H

Table 1-34 Input/Output Operations

Call	Meaning	Header File
read	Read from a file	IO.H
remap	Format a data storage device	IO.H
remove	Delete a file	STDIO.H
rename	Change the name of a file	STDIO.H
rewind	Position file pointer at start	STDIO.H
rmdir	Remove directory	DIRECT.H
scanf	Read formatted input from stdin	STDIO.H
search	Find files	IO.H
setbuf	Allocate buffer to stream	STDIO.H
setvbuf	Allocate buffer to stream	STDIO.H
sprintf	Write formatted output to a string	STDIO.H
sscanf	Read formatted input from a string	STDIO.H
tmpfile	Create temporary file	STDIO.H
tmpnam	Create name for temporary file	STDIO.H
ungetc	Write character back to stream	STDIO.H
unlink	Delete file	IO.H
vfprintf	Output formatted varargs argument list	STDIO.H
vprintf	Output formatted varargs argument list	STDIO.H
vsprintf	Output formatted varargs argument list	STDIO.H
write	Write to file	IO.H

1.5.3 Character Management Functions

The character management system provides functions for the conversion and classification of character types. It is declared in header file **CTYPE.H**.

Table 1-35 Character Management Functions

Call	Meaning	Header File
_tolower	Convert upper case to lower case	CTYPE.H
_toupper	Convert lower case to upper case	CTYPE.H
isalnum	Specify character type (alphanumeric)	CTYPE.H
isalpha	Specify character type (alpha character)	CTYPE.H
isascii	Specify character type (ASCII code 0-127)	CTYPE.H
iscntrl	Specify character type (ASCII-Code > 127 or < 32)	CTYPE.H
isdigit	Specify character type (decimal number (0 - 9)	CTYPE.H
isgraph	Specify character type (decimal number (0 - 9)	CTYPE.H
islower	Specify character type (printable character, no Space characters)	CTYPE.H
isprint	Specify character type (ASCII-Code 32 - 126)	CTYPE.H
ispunct	Specify character type (punctuation)	CTYPE.H
isspace	Specify character type (Space character, Tab character,...)	CTYPE.H
isupper	Specify character type (upper case letter)	CTYPE.H
isxdigit	Specify character type (hexadecimal number 0 – 9, A – F, a – f)	CTYPE.H
toascii	Mask all non-ASCII bits	CTYPE.H
tolower	Convert upper case to lower case	CTYPE.H
toupper	Convert lower case to upper case	CTYPE.H

1.5.4 String Operations

The string operations can be used to check, handle and process character or byte strings. They are declared in header files **STRING.H** and **STDLIB.H**.

Table 1-36 String Operations

Call	Meaning	Header File
atof	Convert string to double number	STDLIB.H
atoi	Convert string to integer number	STDLIB.H
atol	Convert string to long number	STDLIB.H
strcat	Concatenate two strings	STRING.H
strchr	Get a character in a string	STRING.H
strcmp	Compare two strings	STRING.H
strcpy	Copy one string into another	STRING.H
strcspn	Indicate to what extent one string matches another	STRING.H
strlen	Indicate to what extent one string matches another	STRING.H
strncat	Return the number of characters in a string	STRING.H
strncmp	Append up to n characters from one string to another	STRING.H
strncpy	Copy one string into another, up to n characters	STRING.H
strpbrk	Search a string for the first appearance of a character	STRING.H
strrchr	Search a string for the last appearance of a character	STRING.H
strspn	Return the length of the substring in String 1 consisting exclusively of the characters specified in String 2	STRING.H
strstr	Find the first match between String 1 and String 2	STRING.H
strtod	Convert string to a double number	STDLIB.H
strtok	Search a string for the first of several character sequences	STRING.H
strtol	Convert a string to a long number	STDLIB.H
strtoul	Convert string to an unsigned long number	STDLIB.H

1.5.5 Memory Operations

Memory operations are used to copy characters, and to compare or write memory areas. The memory operations are declared in header files **STRING.H** and **MEMORY.H.**

Table 1-37 Memory Operations

Call	Meaning	Header File
memccpy	Copy character from source area to destination area	STRING.H, MEMORY.H
memchr	Find a character in a memory area	STRING.H, MEMORY.H
memcmp	Compare two memory areas	STRING.H, MEMORY.H
memcpy	Copy character from source area to destination area	STRING.H, MEMORY.H
memmove	Move character from source area to destination area	STRING.H, MEMORY.H
memset	Write a character to a memory area n times	STRING.H, MEMORY.H

1.5.6 Memory Allocation

These functions can be used to allocate memory from the heap. You will find the function declarations in header files **MALLOC.H** and **STDLIB.H**.

Table 1-38 Memory Allocation Operations

Call	Meaning	Header File
calloc	Allocate memory for a number n elements of a specified size	MALLOC.C, STDLIB.H
free	Free memory	MALLOC.C, STDLIB.H
malloc	Allocate memory	MALLOC.C, STDLIB.H
realloc	Change the size of a previously allocated memory area	MALLOC.C, STDLIB.H

1.5.7 Mathematical Functions

The functions declared in header file **STDLIB.H** can only be used on integers. Floating-point functions are declared in header file **MATH.H.**

Table 1-39 Mathematical Functions

Call	Meaning	Header File
abs	Get absolute value of an integer	STDLIB.H
acos	Calculate arc cosine of a double number	MATH.H
asin	Calculate arc sine of a double number	MATH.H
atan	Calculate arc tangent of a double number	MATH.H
atan2	Calculate arc tangent of two double numbers allowing for all four quadrants	MATH.H
ceil	Round up to the nearest whole double number	MATH.H
cos	Calculate the cosine of a double number	MATH.H
cosh	Calculate the hyperbolic cosine of a double number	MATH.H
div	Divide two integers	STDLIB.H
exp	Calculate e^x of a double number	MATH.H
fabs	Calculate the absolute value of a double number	MATH.H
floor	Round down to the nearest whole double number	MATH.H
fmod	Calculate the remainder from the division of two double numbers	MATH.H
frexp	Return the mantissa and binary exponent	MATH.H
labs	Get the absolute value of a long number	STDLIB.H
ldexp	Calculate double number*2 ^{integer}	MATH.H
ldiv	Divide two integers	STDLIB.H
log	Calculate the natural logarithm of a double number	MATH.H
log10	Calculate the logarithm to base 10 of a double number	MATH.H
matherr	User-specific function for error handling in numeric functions	MATH.H
modf	Subdivides a double number into mantissa and exponent	MATH.H

Table 1-39 Mathematical Functions

Call	Meaning	Header File
pow	Calculate the power of two double numbers	MATH.H
rand	Generate a random integer	STDLIB.H
sin	Calculate the sine of a double number	MATH.H
sinh	Calculate the hyperbolic sine of a double number	MATH.H
sqrt	Calculate the square root of a double number	MATH.H
srand	Initialization value for pseudorandom numbers	STDLIB.H
tan	Calculate the tangent of a double number	MATH.H
tanh	Calculate the hyperbolic tangent of a double number	MATH.H

1.5.8 Time and Date Functions

These functions can be used to convert time and date parameters, for example to adapt them to different time zones. The functions are declared in header file **TIME.H.**

Table 1-40 Time and Date Functions

Call	Meaning	Header File
asctime	Convert a time parameter to a string	TIME.H
ctime	Convert date and time to a string	TIME.H
difftime	Find the difference between two times	TIME.H
gmtime	Convert time to Greenwich Mean Time (GMT)	TIME.H
localtime	Correct local time according to time zone differences	TIME.H
mktime	Convert time	TIME.H
strftime	Formatted output of date and time	TIME.H
time	Get system time	TIME.H
tzset	Calculate time zone conversion	TIME.H

1.5.9 Control Functions

The control functions are needed in order to terminate tasks. They are declared in header file **STDLIB.H**.

Table 1-41 Control Functions

Call	Meaning	Header File
abort	Send SIGABRT signal to calling task	STDLIB.H
assert	Check a condition and abort task if not fulfilled	ASSERT.H
atexit	Define routines to be called at the end of a task	STDLIB.H
exit	Resolve task and terminate with defined status	STDLIB.H
x_cr_killtsk	Delete task	TASK.H

1.5.10 Error Handling

errno and **errno2** (RMOS extension) are both available.

Table 1-42 Error Handling Functions

Call	Meaning	Header File
errno, errno2	Error number	ERRNO.H
perror	Output operating system error messages	STDIO.H
strerror	Return a pointer to an error text	STRING.H
sys_nerr	Return the number of error messages in sys_errlist	ERRNO.H
sys_errlist	Return a string array with error messages	ERRNO.H

1.5.11 Other Functions

The following functions are not allocated to any specific class.

Table 1-43 Other Functions

Call	Meaning	Header File
bsearch	Binary search in a sorted table	STDLIB.H
getenv	Get contents of an environment variable	STDLIB.H
longjmp	Perform a non-local jump	SETJMP.H
putenv	Change an environment variable or add a new one	STDLIB.H
qsort	Sort data elements in the specified order	STDLIB.H
raise	Pass control to a signal handler	SIGNAL.H
setjmp	Set the label for a subsequent non-local jump	SETJMP.H
signal	Install a signal handler for exception handling	SIGNAL.H
sleep	Stop task for a specified time	TIME.H
x_cr_gettaskid	Get the ID of the calling task	TASK.H
x_cr_gettaskparam	Get `stdin`, `stdout`, `stderr` and task environment	TASK.H
x_cr_initenv	Initialize task environment	TASK.H
x_cr_setexit	Set task-specific exit handler	TASK.H
xinitc	Initialize C library	TASK.H
xinitt	Perform task-specific initialization of C library	TASK.H

1.6 Functions of the Socket Interface

General Information

M7-SYS RT presents the Socket Interface functions for TCP/IP communications. In order to use these functions you have to include the **SOCKET.H** header file in your M7 RMOS32 applications. You should also include the library **RMFSK2IB.LIB** in the link statement.

Table 1-44 Functions of the Socket Interface

Call	Meaning
accept	Accept a connection on a socket
bind	Bind a name to a socket
connect	Request a connection on a socket
endhostent	Close the HOSTS file
endnet	Release the task–related resources of sockets
endservent	Close the SERVICES file
gethostbyaddr	Read a communication host entry from the HOSTS file
gethostbyname	Read a communication host entry from the HOSTS file
gethostent	Read an entry from the HOSTS file
getpeername	Read the name of the peer associated with the socket
getservbyname	Read communication service entry from SERVICES file
getservbyport	Read a communication service entry from the SERVICES file
getservent	Read an entry from SERVICES file
getsockname	Read socket name
getsockopt	Read socket options
htons	Convert a value from host byte order to network byte order
listen	Prepare a socket to establish a passive connection
nselect	Wait for events simultaneously on several sockets
ntohs	Convert a value from network byte order to host byte order
recv	Receive a message from a socket
recvfrom	Receive a datagram
send	Send a message to a connected socket
sendto	Send a message to a socket with a specific address
sethostent	Open the HOSTS file
setservent	Open the SERVICES file
setsockopt	Set socket options
shutdown	Close a socket for sending messages
socket	Create an end point for communication

1.7 Serial Interface Functions

General Information

RMOS presents an API for serial functions. In order to use these functions you have to include the **SERIAL.H** header file in your M7 RMOS32 applications. You should also include the library **RMFSER.LIB** in the link statement.

Table 1-45 Serial Interface Functions

Call	Meaning
SerialCheckChar	Read in single character from unit
SerialCheckString	Read string from unit
SerialClose	Close a connection to a unit of a order
SerialGetChar	Read in single character from unit
SerialGetString	Read string from unit
SerialInit	Initialize unit
SerialInitEx	Extended initialization of unit
SerialOpen	Establish a connection to a unit of a driver
SerialPutChar	Write a single character to a unit
SerialPutString	Write characters to the unit

1.8 Other Functions

General Information

RMOS presents other functions for hardware–orientated I/O–operations and interrupt working. The header file **MISC86.H** must be included from M7 RMOS32–programmes as an header file for prototyps of the functions.

1.8.1 Functions for interrupt working

The following functions are available for interrupt working

Table 1-46 Functions for interrupt working

Call	Meaning
causeinterrupt	Generate Software–Interrupt
geniinterrupt	Generate Software–Interrupt

1.8.2 Functions for hardware–orientated I/O–operations

The following functions are available for hardware–orientated I/O–operations.

Table 1-47 Functions for hardware–orientated I/O–operations

Call	Meaning
disable	Disable hardware–interrupts
enable	Enable hardware–interrupts
inbyte	Read byte from a hardware port
inp	Read byte from a hardware port
inport	Read word from a hardware port
inport b	Read byte from a hardware port
inpw	Read a word from a hardware port
inword	Read a word from a hardware port
outbyte	Output a byte to a hardware port
outp	Output a byte to a hardware port
outport	Output a word to a hardware port
outportb	Output a byte to a hardware port
outpw	Output a word to a hardware port
outword	Output a word to a hardware port

Type Identifiers

2

In this chapter

2.1 System Messages of the M7 Server

Notes

The identifiers for the system messages of the M7 servers are listed below in ascending numerical order. The M7 RMOS32 tasks can register themselves on M7 servers, so that they can receive a message when an event occurs.

The M7 servers send the messages with the accompanying identifier to the task message queue. The tasks read the message using the function `RmReadMessage` , and evaluate the message identifier passed in the *Message* variable, for example using a “switch” statement.

In the parameter *pMessageParam*, all messages also contain the address of the FRB referenced on registration with the `M7Link...`(..) call.

The constants listed below are defined in the **M7API.H** file. All numeric constants in the header file are “cast” explicitly in the C type *unsigned int*. The following list shows the numeric constants without this cast.

OST Server

The following table shows the message identifiers passed in the *Message* parameter for messages sent from the OST (Operating State Transition) server to M7 RMOS32 tasks.

Table 2-1 Messages of the OST Server

Identifier	Description
M7MSG_TRANSITION	The message is sent from the OST server before the transition to a new operating state. The *pMessageParam* variable references the **M7TSFRB** passed on registration with `M7LinkTransition`
M7MSG_STATE	The message is sent from the OST server immediately after the transition to a new operating state. The *pMessageParam* variable references the **M7TSFRB** passed on registration with `M7LinkState`.
M7MSG_REQ_FINISHED	The message is sent from the OST server immediately **after** the transition to or denial of the new operating state requested. The *pMessageParam* variable references the **M7TSFRB** passed on registration with `M7RequestState.`
M7MSG_BATTERY_FAILURE	The message is sent from the OST server immediately **after** the battery voltage drops below the threshold limit. The *pMessageParam* variable references the **M7TSFRB** passed on registration with `M7LinkBatteryFailure`

S7 Object Server

The following table shows the message identifiers sent from the S7 object server to M7 RMOS32 tasks.

Table 2-2 Messages of the S7 Object Server

Identifier	Description
M7MSG_DATA_ACCESS_R	The message is sent from the S7 object server immediately **after** the read access to an S7 object. The *pMessageParam* variable references the **M7OBJFRB** passed on registration with `M7LinkDataAccess`
M7MSG_DATA_ACCESS_W	The message is sent from the S7 object server immediately **after** the write access to an S7 object. The *pMessageParam* variable references the M7OBJFRB passed on registration with `M7LinkDataAccess.`
M7MSG_DATA_ACCESS_CREATE	The message is sent from the S7 object server immediately **after** the creation of a new S7 object. The *pMessagePara*m variable references the M7OBJFRB passed on registration with `M7LinkDataAccess.`
M7MSG_DATA_ACCESS_DEL	The message is sent from the S7 object server immediately **after** the deletion of an S7 object. The *pMessageParam* variable references the **M7OBJFRB** passed on registration with M7LinkDataAccess.
M7MSG_DATA_ACCESS_LINK	The message is sent from the S7 object server immediately **after** the linking of an S7 object. The *pMessageParam* variable references the **M7OBJFRB** passed on registration with `M7LinkDataAccess`

Time Server

The following table shows the message identifiers sent from the time server to M7 RMOS32 tasks.

Table 2-3 Message of the Time-Servers

Identifier	Description
M7MSG_TIMESERVER	The message is sent from the S7 object server immediately **after** the time event. The *pMessageParam* variable references the **M7TFRB** passed on registration with `M7Link..`

FC Server

The following table shows the message identifiers sent from the FC (Free Cycle) server to M7 RMOS32 tasks.

Table 2-4 Message of the FC Server

Identifier	Description
M7MSG_CYCLE	The message is sent from the FC server at the beginning of a state (STARTUP, FREECYCLE, ZKP).
M7MSG_PI_ERROR	The message is sent from the FC server after the appearing of an I/O type transfer error.

Alarm Server

The following table shows the message identifiers sent from the alarm server to M7 RMOS32 tasks.

Table 2-5 Messages of the Alarm Server

Identifier	Description
M7MSG_IO_ALARM	The message is sent from the alarm server immediately **after** an I/O alarm is signaled by the corresponding module. The *pMessageParam* variable references the **M7IOFR**B passed on registration with `M7LinkIOAlarm`
M7MSG_DIAG_ALARM	The message is sent from the alarm server immediately **after** a diagnostics alarm is signaled by the corresponding module. The *pMessageParam* variable references the **M7DIAGFRB** passed on registration with `M7LinkDiagAlarm`
M7MSG_ZS_ALARM	The message is sent from the alarm server immediately **after** an insert/remove module alarm is signaled by the corresponding module. The *pMessageParam* variable references the **M7ZSFRB** passed on registration with `M7LinkZSAlarm`

K Bus Subsystem

The following list shows the message identifiers sent from the communication bus subsystem to M7 RMOS32 tasks.

Table 2-6 Messages of the K Bus Subsystem

Identifier	Description
M7MSG_DIAG_MSG	The message from the K BUS subsystem indicates the receipt of a diagnostics message, which can be read out by the M7 RMOS32 task with the `M7KEvent` call.
M7MSG_BUB_NDR	The message from the K BUS subsystem indicates the receipt of new MMI data, which can be read out by the M7 RMOS32 task with the `M7KEvent` call.
M7MSG_PBK_NDR	The message from the K BUS subsystem indicates the receipt of new data after an `M7PBKBrcv` call.
M7MSG_PBK_DONE	The message from the K BUS subsystem indicates the completion of a `M7PBKBsend` call.

2.2 Identifiers for S7 Objects and Data Types

Type Identifiers

The S7 objects listed in the following table are supported by the S7 object server on an M7 automation computer. The type identifiers listed below are defined in header file **M7API.H**, and are required in the corresponding M7 API function calls, in order to address S7 objects.

The accompanying numerical values are cast in **M7API.H** in the M7 data type UBYTE.

Table 2-7 Objects Supported on the M7

S7 Object	Type Identifier	Initialization
I/O area	M7D_IO	Automatic
Process image of inputs	M7D_PII	Automatic
Process image of outputs	M7D_PIQ	Automatic
Flag area	M7D_M	C user program
Data block	M7D_DB	C user program
Data records, read * (for communication only for MMI functions)	M7D_PAR_READ	C user program
Data records, write * (for communication only for MMI functions)	M7D_PAR_WRITE	C user program

* The attributes “Read” and “Write” for data records are considered on a FM from the view of the CPU. The FM read the data records – for example data records of parameter – which were written by the CPU (Type Identifier M7D_PAR_WRITE). On the other side the FM write data records – for example data records of diagnosis – which shall be read by the CPU (Type Identifier M7D_PAR_READ).

Subarea Number

The following table lists the subarea numbers for the individual S7 objects. The listed subarea numbers are required in the corresponding M7 API function calls, in order to address S7 objects of an S7 CPU or an M7.

Table 2-8 Subarea Numbers for S7 Objects

S7 Object	Type Identifier	Subarea Number	Value Range
I/O area	M7D_IO	0	0...0xFFFF
Process image of inputs	M7D_PII	0	0 ... 255 or 511
Process image of outputs	M7D_PIQ	0	0 ... 255 or 511
Flag area	M7D_M	0	0 ... 65 535
Data block	M7D_DB	DB number	0 ... 65 535 for M7
Data records, read	M7D_PAR_READ	No. of data record	0 ... 255 for M7
Data records, write	M7D_PAR_WRITE	No. of data record	0.... 255 for M7

Data Type Identifiers

The identifiers in the following table specify the possible data types of variables within S7 objects. The identifiers are used in all M7 calls which access a variable area within an S7 object.

The corresponding M7 data types are listed in the following table.

Table 2-9 Data Type Identifiers for Accessing S7 Objects

M7 Data Type	Type Identifier
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

Block Type Identifiers

The identifiers in the table specify the possible block types which can be stored in the working memory of an S7 CPU or M7. The identifiers are used in M7 calls to the object management system.

Table 2-10 Block Type Identifiers

Block Type	Type Identifier	Remarks
OB organization block	M7BLKTYP_OB	S7-CPU only
Data block	M7BLKTYP_DB	M7 and S7-CPU
Function call	M7BLKTYP_FC	S7-CPU only
System function call	M7BLKTYP_SFC	S7-CPU only
Function block	M7BLKTYP_FB	S7-CPU only
System function block	M7BLKTYP_SFB	S7-CPU only

Data Structures

3

In this chapter

3.1 Data Types of the RMOS API

Notes

The following general data types are defined in header file **RMTYPES.H** of the RMOS API. These data types should be used instead of the general C data types for the appropriate RMOS API calls.

Table 3-1 GeneralData Type Definitions of the RMOS API

Name	Type Definition	Meaning
uchar	unsigned char	Unsigned char (value range: 0 ... 255)
ushort	unsigned short	Unsigned 16-bit integer (value range: 0 ... 65 535)
uint	unsigned int	Unsigned 32-bit integer (value range: 0 ... 2^32 - 1)
ulong	unsigned long	Unsigned 32-bit integer (value range: 0 ... 2^32 - 1)
rmproc	void(*rmproc)(void)	Pointer to function with no input or return parameters

3.2 Data Structures of the RMOS API

Notes

The following general data structures are defined in header file **RMTYPES.H** of the RMOS API. These data structures are used in the corresponding RMOS API calls.

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;
```

Description

The `Rm3964InitStruct` structure contains the configuration data for the initialization of a unit for 3964(R) communication. The configuration is performed with the `RmIOControl` control function RM_IOCTL_INIT.

Field	Type	Meaning
irq	ushort	IRQ number of the interface (e.g. 4 for COM1) The IRQ parameter is only evaluated the first time the unit is initialized. It is ignored on further calls of control function RM_IOCTL_INIT.
base	ushort	I/O base address of the 8250 chip (e.g. 0x3F8 for COM1) The base address is only evaluated the first time the unit is initialized. It is ignored on further calls of control function RM_IOCTL_INIT.
mode_baud	ulong	Baud rate (numeric value, e.g. 19200)
mode_parity	uchar	Control of the parity bit. The following are permitted: RM_IOCTL_MODE_PARITYNONE No parity check RM_IOCTL_MODE_PARITYEVEN Even parity RM_IOCTL_MODE_PARITYODD Odd parity RM_IOCTL_MODE_PARITY0 Parity bit always 0 RM_IOCTL_MODE_PARITY1 Parity bit always 1
mode_data	uchar	Number of data bits (possible values: 5,6,7,8)
mode_stop	uchar	Number of stop bits. RM_IOCTL_MODE_STOP1 — 1 stop bit RM_IOCTL_MODE_STOP2 — 2 stop bits RM_IOCTL_MODE_STOP15 — 1.5 stop bits

Field	Type	Meaning
mode_fill	uchar	Reserved
prot3964r	int	Protocol selection 0 3964–Protokoll 1 3964R–Protokoll
master	int	Master/slave definition 0 Slave 1 Master

Example

```
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);
```

See Also

RmIOControl

RmAbsTimeStruct

Syntax

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

Description

This structure contains the absolute system time in milliseconds since the last complete restart and it is used by the `RmGetAbsTime` function call.

Field	Type	Meaning
`lotime`	ulong	Low-order part of the absolute time
`hitime`	ulong	High-order part of the absolute time

See Also

RmGetAbsTime

RmEntryStruct

Syntax

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

Description

The **`RmEntryStruct`** structure is used in RMOS API calls `RmList` and `RmGetEntry`, in order to read items from the resource catalog.

Field	Type	Meaning
`slen`	uchar	Length of following character string.
`string`	char[16]	Character string containing the name of a resource.
`type`	uchar	Specifies the type of source. The following values are possible: Value Define — Meaning 0 RM_CATALOG_TASK — Task 1 RM_CATALOG_DEVICE — Device driver 2 RM_CATALOG_POOL — Memory pool 3 RM_CATALOG_SEMAPHORE — Semaphore 4 RM_CATALOG_EVENTFLAG — Global event flag 5 RM_CATALOG_CNTRL — Monitored program access 6 RM_CATALOG_LOCALMAILBOX — Local mailbox 7 RM_CATALOG_MISC — Reserved 8 RM_CATALOG_USER — User-defined type 10 RM_CATALOG_UNIT — Unit 11 RM_CATALOG_MESSAGE — Messages 255 RM_CATALOG_ALL
`ide`	ulong	Specifies the extended ID of the resource. The value range depends on the type and the maximum values configured.
`id`	ushort	Specifies the ID of the resource. The value range depends on the type and the maximum values configured.

Note

Resource type RM_CATALOG_USER is not reserved for specific RMOS resources, and can be used by the programmer for any purposes of his own. It

could be used, for example, to display the availability of specific library modules by cataloging them under the library name and the RM_CATALOG_USER type.

See Also **RmCatalog, RmList, RmGetEntry, RmGetName**

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;
```

Description

The `RmSetIntMailboxHandler` call of the RMOS API can be used to define interrupt handlers for sending a message to a mailbox. The **`RmIntrhandMailStruct`** structure defines the format of this message, which is stored in the mailbox when the interrupt is triggered. The structure incorporates a total of three 32-bit words.

Field	Typ	Meaning
`int_no`	uint	Identifies the number of current interrupt received.
`int_vec`	8 bits	Specifies the interrupt vector.
`int_kind`	1 bit	Identifiers the type of interrupt: Value Meaning 0 Hardware interrupt 1 Software interrupt
`lost_int_ overflow`	1 bit	This bit is enabled (= 1) if interrupts are lost.
`dummy_2`	22 bits	Reserved
`lost_int`	ushort	Specifies the number of lost interrupts.
`dummy_3`	ushort	Reserved

See Also

RmSetIntMailboxHandler

RmIOCTLModeSerialStruct

Syntax

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

Description

The `RmIOCTLModeSerialStruct` structure contains the configuration data for drivers for serial interfaces (e.g. 8250). It is required with the `RmIOControl` control function RM_IOCTL_MODE in order to reconfigure the unit.

Field	Type	Meaning
baud	ulong	Transmission rate (numeric value, e.g. 19200)
parity	uchar	Control of the parity bit. The following are permitted: RM_IOCTL_MODE_PARITYNONE — No parity check RM_IOCTL_MODE_PARITYEVEN — Even parity RM_IOCTL_MODE_PARITYODD — Odd parity RM_IOCTL_MODE_PARITY0 — Parity bit always 0 RM_IOCTL_MODE_PARITY1 — Parity bit always 1
data	uchar	Number of data bits (numeric value, e.g. 8)
stop	uchar	Number of stop bits. The following are permitted: RM_IOCTL_MODE_STOP1 — 1 stop bit RM_IOCTL_MODE_STOP2 — 2 stop bits RM_IOCTL_MODE_STOP15 — 1.5 stop bits

Example

```
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);
```

See Also

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 ;
```

Description

The `RmIOCTLPropertiesStruct` structure contains information about the function scope of the loadable driver.

Field	Type	Meaning
block_device	1 bit	Type of driver 0: Character–oriented driver 1: Block–oriented driver
convert	1 bit	Reserved

Field	Type	Meaning
protocol	1 bit	Protocol driver (e.g. 3964R) 1 = yes, 0 = no
terminal	1 bit	Terminal driver 1 = yes, 0 = no
hsfs	1 bit	Mass storage driver (e.g. for hard disk) 1 = yes, 0 = no
serial	1 bit	Driver for serial interface 1 = yes, 0 = no
buffer	1 bit	Background buffer exists? 1 = yes, 0 = no
reserved1	9 bits	Reserved
reserved2	16 bits	Reserved
ioctl_lock	1 bit	Lock function (RM_IOCTL_LOCK) exists 1 = yes, 0 = no
ioctl_get_status	1 bit	RM_IOCTL_GET_STATUS exists 1 = yes, 0 = no
ioctl_verify	1 bit	Verify function (RM_IOCTL_VERIFY_ON / OFF) 1 = yes, 0 = no
ioctl_linemode	1 bit	Line–oriented reading (RM_IOCTL_LINEMODE_ON / OFF) 1 = yes, 0 = no
ioctl_readterm	1 bit	Terminator character for reading (RM_IOCTL_READTERM_ON / OFF) 1 = yes, 0 = no
ioctl_writeterm	1 bit	Terminator character for writing (RM_IOCTL_WRITETERM_ON / OFF) 1 = yes, 0 = no
ioctl_readstop	1 bit	Stop character for reading (RM_IOCTL_READSTOP) and maximum number of characters (RM_IOCTL_READLEN) 1 = yes, 0 = no
ioctl_writestop	1 bit	Stop character for writing (RM_IOCTL_WRITESTOP) 1 = yes, 0 = no
ioctl_readtout	1 bit	Timeout for reading (RM_IOCTL_READTIMEOUT) 1 = yes, 0 = no
ioctl_writetout	1 bit	Delay for writing (RM_IOCTL_WRITEDELAY) 1 = yes, 0 = no

Field	Type	Meaning
ioctl_echo	1 bit	Activate/deactivate echo function (RM_IOCTL_ECHO_ON / OFF) 1 = yes, 0 = no
ioctl_line_feed	1 bit	Line feed (RM_IOCTL_LINE_FEED) 1 = yes, 0 = no
ioctl_form_feed	1 bit	Form feed (RM_IOCTL_FORM_FEED) 1 = yes, 0 = no
ioctl_abort_char	1 bit	Abort character (RM_IOCTL_ABORTCHAR_ON / OFF) 1 = yes, 0 = no
ioctl_terminal	1 bit	Select terminal/transparent mode (RM_IOCTL_TERMINAL_ON / OFF) 1 = yes, 0 = no
reserved3	1 bit	Reserved
reserved4	16 bits	Reserved
block_size	ulong	Block size for block–oriented drivers (Bytes)
number_of_blocks	ulong	Number of blocks for block–oriented drivers
reserved5	ulong	Reserved
reserved6	ulong	Reserved
reserved7	ulong	Reserved

See Also **RmIOControl**

RmIOCTLVersionStruct

Syntax

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

Description

The structure `RmIOCTLVersionStruct` is used to find out the version of a loadable driver.

Feld	Typ	Bedeutung
MajorVersion	int	Version of the driver (value before the point). For example for Version 1.0 is the MajorVersion 1
MinorVersion	int	Version of the driver (value after the point). For example for Version 1.0 is the MinorVersion 0
DriverInfo1	int	Dependent information of the driver (For SER8250.DRV and 3964.DRV always 0)
DriverInfo2	int	Dependent Information of the driver (For SER8250.DRV and 3964.DRV always 0)
Name	char Array	Name of the driver, which is registered in the catalog (SER8250 or. 3964).

See Also

RmIOControl

RmMailboxStruct

Syntax

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

Description

`RmMailboxStruct` is used to send a message indirectly via the mailbox by passing the memory address and length of the message to the mailbox, instead of the message itself.

Field	Type	Meaning
`adr`	void *	Contains a pointer to the memory address of the message
`adr_res`	ushort	Padding word for FLAT model
`pad`	ushort	Is padded up to 64 bits
`len`	uint	Specifies the length of the message

See Also

RmSendMail, RmReceiveMail

RmMailIDStruct

Syntax

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

Description

Return value of the `RmSendMailDelayed` function. This return value is required, for example, to delete send-delayed mail.

Field	Type	Meaning
`low`	ulong	Least-significantpart of mail ID
`high`	ulong	Most significant part of mail ID

See Also

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;
```

Description

Return value of `RmGetMemPoolInfo` function. The return value contains information on the specified memory pool.

Field	Type	Meaning
`pool_size`	ulong	Total size of memory pool
`avail_mem_size`	ulong	Total size of memory available
`max_block_size`	ulong	Size of the largest block of memory available (always –1)

See Also

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;
```

Description

The `Ser8250InitStruct` structure contains the configuration data for initializing a unit for the driver of a serial interface. The configuration is performed with the `RmIOControl` control function RM_IOCTL_INIT.

Field	Type	Meaning
irq	ushort	IRQ number of the interface (e.g. 4 for COM1) The IRQ parameter is only evaluated the first time the unit is initialized. It is ignored on further calls of control function RM_IOCTL_INIT.
base	ushort	I/O base address of the 8250 chip (e.g. 0x3F8 for COM1) The base address is only evaluated the first time the unit is initialized. It is ignored on further calls of control function RM_IOCTL_INIT.
mode_baud	ulong	Baud rate (numeric value, e.g. 19200)
mode_parity	uchar	Control of the parity bit. The following are permitted: RM_IOCTL_MODE_PARITYNONE No parity check RM_IOCTL_MODE_PARITYEVEN Even parity RM_IOCTL_MODE_PARITYODD Odd parity RM_IOCTL_MODE_PARITY0 Parity bit always 0 RM_IOCTL_MODE_PARITY1 Parity bit always 1
mode_data	uchar	Number of data bits (possible values 5,6,7,8)
mode_stop	uchar	Number of stop bits. RM_IOCTL_MODE_STOP1 — 1 stop bit RM_IOCTL_MODE_STOP2 — 2 stop bits RM_IOCTL_MODE_STOP15 — 1.5 stop bits

Field	Type	Meaning
mode_fill	uchar	Ignored
buffer_size	ulong	Size of the background buffer of the driver (number of characters)

Example

```
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);
```

See Also

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;
```

Description

The **STDSTRUCT** structure defines the input and output channels **stdin**, **stdout**, and the error output channel **stderr** of a program. A channel can be defined by specifying either a device/unit number combination or a file name.

Field	Type	Meaning
stdxx_dev	int	A value >= 0 defines the number of an I/O driver (device number). Values < 0 have the following meaning: Value — Meaning -1 — The file name specified in stdxx_fname is used. In the case of stdout and stderr, a new file is created stdxx_unit is not used. -2 — The file name specified in stdxx_fname is used. In the case of stdout and stderr, the outputs are appended to the end of the file if it already exists. stdxx_unit is not used. -3 — Users should treat this value in exactly the same way as the value stdxx_dev = -2 , because it only has the following meaning for the interactive CLI command START: The output file was inherited by the calling job and may not be passed down further.

Field	Type	Meaning
stdxx_unit	int	If stdxx_dev has a value >= 0, stdxx_unit defines the number of an I/O device (unit number). If stdxx_dev has a value < 0, stdxx_unit is ignored.
stdxx_fname	char *	Pointer to a file name character string. The file identified by the file name is used if stdxx_dev has a value < 0, as described above.
stdxx_fill	unsigned short	Reserved, padding word for FLAT model
tmp_path	char *	Pointer to a file name character string which specifies a file for temporary data.
tmp_fill	unsigned short	Reserved, padding word for FLAT model

Note

The values –2 and –3 described above for **`stdxx_dev`** , are only relevant to CLI. The `x_cr_gettaskparam` function always returns values >= –1 for **`stdxx_dev`**.

The file name defined by the **`tmp_path`** field is identical to the name specified for the temporary file in the `xinitt` function.

3.3 Data Types of the M7 API

3.3.1 General Data Types of the M7 API

Notes

The following general data types are defined in header file **M7API.H** of the M7 API. These data types should be used instead of the general C data types for the appropriate RMOS API calls.

The following table lists the names of the basic M7 data types used in the M7 API environment. Their definitions can be found in the **M7API.H** header file.

Table 3-2 General Data Types of the M7 API

Name	Type Definition	Meaning
UBYTE	unsigned char	Unsigned character (value range: 0 ... 255)
UWORD	unsigned short	Unsigned 16-bit integer (value range: 0 ... 65535)
UDWORD	unsigned long	Unsigned 32-bit integer (value range: 0...2^32 – 1)
SBYTE	signed char	Signed character (value range: –128...127)
SWORD	signed short	Signed 16-bit integer (value range: –32 768...32 767)
SDWORD	signed long	Signed 32-bit integer (value range: –2^31...2^31 – 1)
BOOL	unsigned int	Boolean value
REAL	float	32-bit floating point number
BYTE	UBYTE	Unsigned character (value range: 0...255)
UBYTE_PTR	UBYTE *	Pointer to UBYTE
WORD	UWORD	Unsigned 16-bit integer (value range: 0...32 767)
DWORD	UDWORD	Unsigned 32-bit integer (value range: 0... 2^32 –1)
M7ERR_CODE	int	Error return value
M7ERR_CODE_PTR	M7ERR_CODE *	Pointer to M7ERR_CODE variable
M7IO_LOGADDR	UWORD	Logical address of a signal
M7IO_BASEADDR	UWORD	Base address of an I/O module
M7CONNID	UWORD	ID of an application connection

3.3.2 FRB – Data Types of the M7 Server

Notes

The following FRB (Function Request Block) structures are defined in header file **M7API.H** of the M7 API. The FRBs are required when registering on the corresponding M7 servers. The following table lists the FRB structures and the accompanying pointer definitions.

Information in the FRBs is accessed exclusively by means of macros. These are also defined in header file **M7API.H.**

Table 3-3 FRB Definitions for M7 API

Type Definition	Meaning
M7FRBHEADER	Header of any FRB. Contains general management information
M7FRBHEADER_PTR	Pointer to an FRB header
M7CBFRB	FRB for registering a callback function on the S7 object server
M7CBFRB_PTR	Pointer to an FRB of type M7CFRB
M7OBJFRB	FRB for registering the access message from the S7 object server
M7OBJFRB_PTR	Pointer to an FRB of type M7OBJFRB
M7IOALARM_FRB	FRB for registering the message for an I/O alarm from the alarm server
M7IOALARM_FRB_PTR	Pointer to an FRB of type M7IOALARM_FRB
M7DIAGALARM_FRB	FRB for registering the message for a diagnostics alarm from the alarm server
M7DIAGALARM_FRB_PTR	Pointer to an FRB of type M7SDIAGALARM_FRB
M7ZSALARM_FRB	FRB for registering the message for an insert/remove alarm from the alarm server
M7ZSALARM_FRB_PTR	Pointer to an FRB of type M7ZSALARM_FRB
M7TFRB	FRB for registering the message for time events from the time server
M7TFRB_PTR	Pointer to an FRB of type M7TFRB
M7TSFRB	FRB for registering the message for new operating states or operating state transitions from the OST server
M7TSFRB_PTR	Pointer to an FRB of type M7TSFRB
M7FSCFRB	FRB for registering the message for free cycle, cycle control point, STARTUP and cycle time monitoring from the FC (Free Cycle) server
M7FSCFRB_PTR	Pointer to an FRB of type M7FSCFRB

Table 3-3 FRB Definitions for M7 API

Type Definition	Meaning
M7COMMFRB	Required when calling single-ended PBK functions
M7COMMFRB_PTR	Pointer to an FRB of type M7COMMFRB

3.3.3 Other Data Types of the M7 Server

Notes

The table lists other data types of the M7 API. The structures for the data types are not described in detail here, because the individual items are accessed exclusively by means of macros.

Table 3-4 Other Data Types of the M7 API

Type Definition	Meaning
M7IO_DESC	Data structure for recording the descriptor information for access to ISA modules
M7IO_DESC_PTR	Pointer to an ISA module descriptor

3.4 Data Structures of the M7 API

Notes

The following general data structures are defined in header file M7API.H of the M7 API. These data structures are used in the corresponding M7 API calls.

M7BLKINFO

Syntax

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

typedef M7BLKINFO * M7BLKINFO_PTR
```

Description

The **M7BLKINFO** structure is used by object management functions when reading the block directory from an S7 CPU or M7. The call uses the structure to return information about a block.

Field	Type	Meaning
`Language`	UWORD	The field returns the identifier of the language in which a block has been created from the block header.
`Blktyp`	UWORD	Block type: The identifiers of the possible block types are listed in Table .
`Blknum`	UWORD	Number of the block
`Bitmap`	UBYTE	The individual bits can be “rounded” using predefined constants, and checked if not equal to zero. M7BLKINFO_PASSIV Block is copied (passive), that is in the temporary load memory M7BLKINFO_ACTIVE Block is linked (active), that is in the working memory M7BLKINFO_RAM Block is in RAM memory or RAM mode M7BLKINFO_EPROM Block is in EPROM memory or EPROM mode M7BLKINFO_BESY Block is in operating system
`filler`	UBYTE	Reserved

See Also

M7OVSFindFirst, M7OVSFindNext

M7BLKLIST

Syntax

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

typedef M7BLKLIST * M7BLKLIST_PTR
```

Description

The **M7BLKLIST** structure is used by object management functions for the simultaneous linking or deletion of multiple blocks.

Field	Type	Meaning
Blktyp	UWORD	Type of block. The identifiers of the possible block types are listed in Table .
Blknum	UWORD	Number of the block

See Also

M7OVSLinkIn, M7OVSDelete

M7CBRet

Syntax

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

Description

A callback function which is registered by a task through an `M7LinkDataAccessCB` call must pass the **M7CBRet** structure back to the M7 API in the return parameter.

The callback function uses the return value to determine whether or not further processing is desired on the S7 object server.

Field	Type	Meaning
`process`	UBYTE	TRUE: Object server performs further processing FALSE: Processing by callback function completed
`result`	UBYTE	Error number if `process`=FALSE
`errcls`	UBYTE	Not relevant
`errcode`	UBYTE	Not relevant

Note

Processing by the object server takes place both if `process` = FALSE and if result ≠ 0.

See Also

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
```

Description

The **M7KTIME** structure is used by the M7 API functions to read and write the time on the K bus.

Field	Type	Meaning
`TimeState`	UWORD	Time state. The use of `TimeState` with the following predefined constants and evaluation for not equal to zero produces the following state values: M7KTIME_SYA Time synchronization performed M7KTIME_ESY Substitute time synchronization performed on LAN M7KTIME_UZS Time jump performed M7KTIME_ZNA Time value is not up-to-date M7KTIME_KMASK Mask for correction value for summer, winter and world time in 1/2 hours If `TimeState` is used with the mask M7KTIME_UA_MASK and subsequently compared if equal to the following constants, the time resolution is as follows:

Field	Type	Meaning
		M7KTIME_UA_M_SEC_1 Resolution 1 msec M7KTIME_UA_M_SEC_10 Resolution 10 msec M7KTIME_UA_M_SEC_100 Resolution 100 msec M7KTIME_UA_SECOND Resolution 1 sec
`Year`	UBYTE	Specifies year: 00 ... 99 (BCD number)
`Month`	UBYTE	Specifies month: 01 ... 12 (BCD number)
`Day`	UBYTE	Specifies day: 01 ... 31 (BCD number)
`Hour`	UBYTE	Specifies hours: 00 ... 23 (BCD number)
`Minute`	UBYTE	Specifies minutes: 00 ... 59 (BCD number)
`Second`	UBYTE	Specifies seconds: 00 ... 59 (BCD number)
`m_sec_10`	unsigned int	Specifies 1/100 seconds: 0 ... 9 When reading time only, during writing = 0
`m_sec_100`	unsigned int	Specifies 1/10 seconds: 0 ... 9 When reading time only, during writing = 0
`Weekday`	unsigned int	Specifies weekday: 1: Sunday 2: Monday 3: Tuesday 4: Wednesday 5: Thursday 6: Friday 7: Saturday
`m_sec_1`	unsigned int	Specifies 1/1000 seconds: 0...9 When reading time only, during writing = 0

See Also **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
```

Description

The **M7OBJ_INFO** structure is used in the `M7GetObjectInfo` call to get information on an S7 object.

Field	Type	Meaning
`Size`	UWORD	Length of S7 object in bytes
`Attrib`	UWORD	Object attributes 0x00 Object allocated by the user 0x01 Object allocated by the Object Server 0x02 Object in SRAM 0x10 Object in RAM–Mode 0x20 Object in ROM–Mode 0x40 Object in BESY–Mode The value of Attrib can also contains a combination of the values above. For example the value 0x11 means, that the S7 Objekt is in RAM–Mode and is allocated by the Objekt Server.
`Data`	unsigned long	Pointer to the data of an S7 object. The structure element has to be casted to the required pointer type by the user.
`External`	UBYTE	TRUE: Memory for the S7 object was allocated by M7 RMOS32 task. FALSE: Memory for the S7 object was allocated by S7 object server.

See Also

M7GetObjectInfo

M7PBKSTATUS

Syntax

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

typedef M7PBKSTATUS * M7PBKSTATUS_PTR
```

Description

The structure is used by the M7 API `M7PBKStatus` function to specify the virtual device.

Field	Type	Meaning
`Logical_state`	UBYTE	Specifies the logical state of the virtual device. The following logical states are possible: M7LSTATE_OK Operating state changes are permitted
`Physical_state`	UBYTE	Specifies the physical state of the virtual device, The following physical states are possible: M7PSTATE_OPERATIONAL Device operational M7PSTATE_NEED_SERVICE Device needs service
`LocalSupplement`	UBYTE	Supplementary information. Within byte 0 of the supplementary information the following state data is transmitted : M7LSUPPL_STOP: Device is in STOP operating state M7LSUPPL_START Device is in START operating state M7LSUPPL_RUN Device is in RUN operating state M7LSUPPL_RESTART Device is in RESTART operating state M7LSUPPL_HALT Device is in HALT operating state M7LSUPPL_DEFECT Device is non-operational

See Also

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
```

Description

The **M7TIME_DATE** structure is used by the M7 API functions to read and set the internal system time.

Field	Type	Meaning
`Hour`	UBYTE	Specifies hours: 0 ... 23
`Minute`	UBYTE	Specifies minutes: 0 ... 59
`Second`	UBYTE	Specifies seconds: 0 ... 59
`HSecond`	UBYTE	Specifies seconds: 0 ... 99 When reading time only
`Day`	UBYTE	Specifies day: 1 ... 31
`Month`	UBYTE	Specifies month: 1 ... 12
`Year`	UWORD	Specifies year e.g.: 1997
`DayOfWeek`	UBYTE	Specifies weekday: 0: Sunday 1: Monday 2: Tuesday 3: Wednesday 4: Thursday 5: Friday 6: Saturday

See Also

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
```

Description

The **M7VARADDR** structure is used by PBK and MMI functions to address a contiguous number of items within an S7 object.

Field	Type	Meaning
`Syntax`	UBYTE	Must always be set to value: 0x10 for this data structure
`DataType`	UBYTE	Specifies the data type of an item within the addressed S7 object. The identifiers for the possible M7 data types are listed in Table .
`Length`	UWORD	Number of items. For data type M7DT_BOOL is only available the value 1 for the parameter LENGTH.
`Part`	UWORD	Specifies the subarea number (DB number, etc.) of an S7 object. The possible subarea numbers for the individual S7 objects are listed in Table .
`Area`	UBYTE	Specifies the type identifier of the S7 object. The possible type identifiers are listed in Table .
`filler`	UBYTE	Reserved; must be set to 0x00.
`Offset`	UDWORD	Specifies the address offset of the first item within the S7 object. The address offset must always be a multiple of the bit length of the specified data type (see `DataType`). For data records byte 0 and 1 (Intel format) specify the logical module address, byte 2 specifies whether Input– or Output address (0 for input, 1 for Output).

See Also

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

M7VARDATA

Syntax

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

typedef M7VARDATA * M7VARDATA_PTR
```

Description

The **M7VARDATA** structure is used by MMI functions to specify a buffer.

The specified buffer is used to hold either the values of the addressed variables (read access) or the data which overwrite the addressed variables (write access).

Field	Typ	Meaning
`Buffer`	UBYTE_PTR	Pointer to the actual buffer. The user program must allocate the buffer either in the global data area or from the heap (remaining memory pool)
`Length`	UDWORD	Length of the data buffer expressed in number of items
`AccessResult`	UBYTE	Specifies the result of the access (read or write). Possible error identifiers are: M7RES_SUCCESS: Transfer successfully completed M7RES_HWERROR: Hardware error M7RES_NOACCESS: No access authorization for object M7RES_INVADDR: Invalid item addressed in S7 object M7RES_INVDTYP: Invalid data type M7RES_NOOBJECT: No such object or invalid length
`DataType`	UBYTE	Specifies the data type of an item. The possible data types can be found in Table .

See Also

M7BUBRead, M7BUBCycRead, M7BUBWrite

3.5 Data Structures of the Socket Interface

The following data structures are defined in header file SOCKET.H of the socket interface. These data structures are used in the corresponding socket calls.

HOSTENT

Syntax

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

Description

The HOSTENT structure is used in the `gethostent`, `gethostbyname` and `getservbyaddr` calls to querry entries in the \ETC\HOSTS file. It contains the individual fields of the HOSTS file. The meaning of the fields is as follows:

Field	Type	Meaning
h_name	char *	Official name of the host
h_aliases	char **	Field with alternative (alias) names for the host (terminated with NULL)
h_addrtype	short	Address type of the host; always AF_INET
h_length	short	Address length in bytes
h_addr	char *	Internet (IP) address of the host; (specified in network byte order)

See Also

gethostent, gethostbyname, gethostbyaddr

SERVENT

Syntax

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

Description

The SERVENT structure is used in the `getservent`, `getservbyname` and `getservbyport` calls to querry entries in the SERVICES file. It contains the individual fields of the SERVICES file. The meaning of the fields is as follows:

Field	Type	Meaning
s_name	char *	Official name of the service
s_aliases	char **	Field with alternative (alias) names for the service (terminated with NULL)
s_port	int	Number of the port over which the service can be accessed
s_proto	char *	Protocol which must be used to address the service

The port number s_port is represented in host byte order; it must be converted, if necessary, to network byte order with `htons`.

See Also

getservent, getservbyname, getservbyport

SOCKADDR

Syntax

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

Description

The SOCKADDR structure is used in socket interface calls to specify or check the addresses of the communication hosts. The meaning of the fields is as follows:

Field	Type	Meaning
sa_family	short	Address family
sin_port	short	Internet port number
sin_addr	char [4]	Internet (IP) address

See Also

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;
```

Description

The SOCKSEL structure is used in the `nselect` call to check events on a specific socket. The meaning of the fields is as follows:

Field	Type	Meaning
se_inflags	unsigned short	Input/request flags
se_outflags	unsigned short	Output/reply flags
se_fd	int	Socket descriptor
se_1reserved	int	Reserved
se_user	unsigned long	Free for the user
se_2reserved	unsigned long	Reserved

See Also

nselect

3.6 Parameter Data Records for the IF 961-AIO/DIO Interface Modules

Options

There are two ways to initialize the interface modules:

1. Using STEP 7
2. By calling the *M7StoreRecord* function in the user program

Analog Input/Output Module IF 961-AIO

The table below contains the parameters which you may assign the IF 961-AIO interface module. The interface module has:

- 4 input channels and
- 2 output channels.

Table 3-5 Parameters for the IF 961-AIO Interface Module

Parameter	Data Type	Value Range	Coding	Default Value	Byte ADD	Bit ADD
Data record DS0, 2 bytes long						
Conversion time (scan cycle time)	FIELD3	{5.7 ms | 2.8 ms | 1.3 ms | 0.6 ms | 0.185 ms }	{0|1|2|3|4}	0	0	0
Interrupt generation	FIELD1	{No | Yes }	{0|1}	0	0	6
Analog conversion (method of sampling the analog channels)	FIELD1	{ Selective | Cyclic }	{0|1}	0	0	7
...	BIT[3]	...	...	0	0	3
Interrupt selection	FIELD2	{None | Process | Process + Diagnostics }	{0|1|2}	0	1	0
...	BIT[6]	...	...	0	1	2

Process Interrupts and Diagnostic Interrupts

If the IF 961-AIO interface module has been configured for cyclic conversion (analog conversion = 1), it is possible to initiate process interrupts at the end of the cycle. It is also possible to initiate a diagnostic interrupt in the event of a lost process interrupt.

Digital Input/Output Module IF 961-DIO

The following table 3-6contains the parameters which you may assign the IF 961-DIO interface module.

Figure 3-1 shows the structure of data record 1 of the parameters for the IF 961-DIO interface module.

Table 3-6 Parameters for the IF 961-DIO Interface Module

Parameter	Data Type	Value Range	Coding	Default Value	Byte ADD	Bit ADD
Data record DS0, 2 bytes long						
Input delay	FIELD1	{ 3 ms | 0,5 ms }	{0|1}	0	0	0
Data record DS1, 4 bytes long						
Interrupt enable (for process interrupts)	FIELD1	{ NO | YES }	{ 0|1 }	0	0	7
Interrupt enable on positive signal edge	FIELD1	{ NO | YES }	{0|1}	0	1	0+IC
Interrupt enable on negative signal edge	FIELD1	{ NO | YES }	{0|1}	0	2	0+IC

IC = Input channel: [0 .. 7]

Structure of Data Record 1

A parameter is activated by setting the respective bit to “1”. A “1” in bytes 1 and 2 means that the process interrupt is enabled.

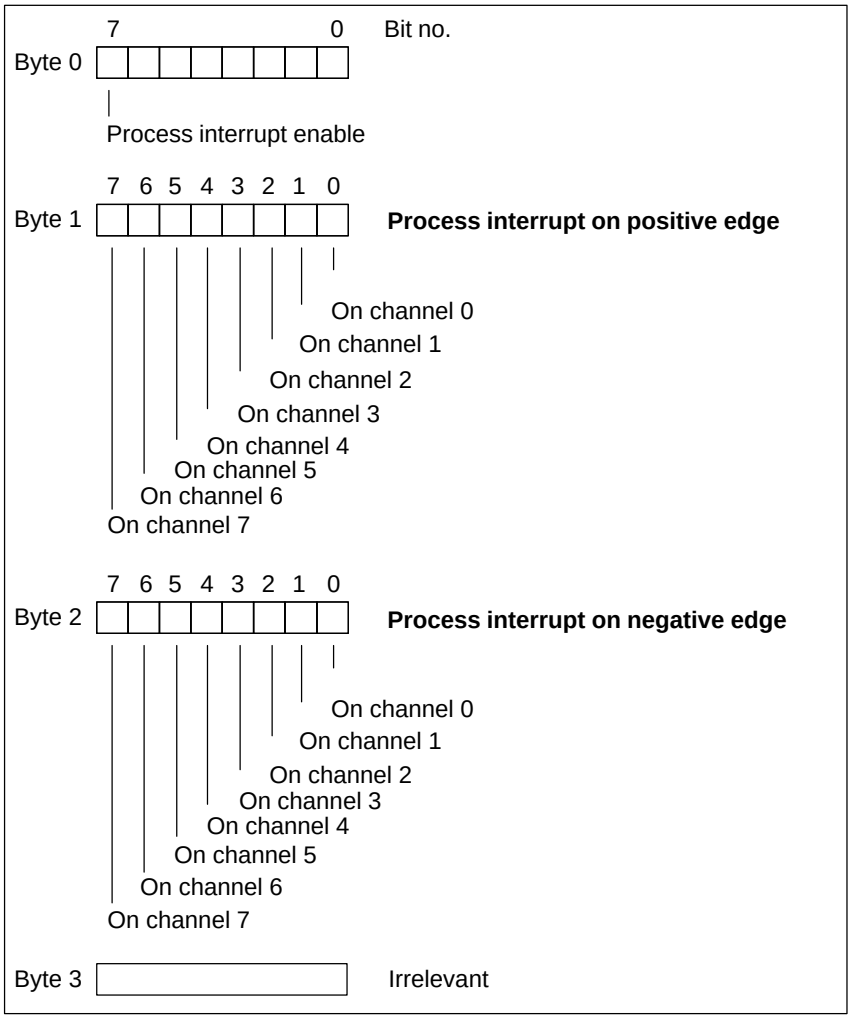


Figure 3-1 Parameter Data Record 1 for the IF 961-DIO Interface Module

Error Codes and Messages

4

In this chapter

4.1 Error Messages of the M7 RMOS32 Kernel

The M7 RMOS32 kernel (nucleus) outputs error messages on the system console. The default setting for the system console is the serial COM2 interface, but this can be reconfigured (see User Manual).

Missing System Resources

The M7 RMOS32 kernel requires system memory blocks for the management of resources. These are allocated from the heap and are released again dynamically.

The following error messages can be output when there are insufficient system resources:

***** nuc: <date> <time> no SRBS, SYSTEM HALTED**
There are no more system request blocks (SRB) available for the operating system.

***** nuc: <date> <time> no SMRS, SYSTEM HALTED**
There are no more system memory blocks (SMR) available for the operating system (e.g. driver requests SMR).

***** nuc: <date> <time> SMRS increased**
The kernel has increased the number of system memory blocks (SMRs) by 50.

***** nuc: <date> <time> SMRS reached 0**
The number of system memory blocks (SMRs) could not be increased again; the RMOS API call has been delayed. This state only occurs if no memory is available in the heap, or if the data segment of the kernel could not be increased because of the fragmentation of the heap.

Only tasks which request SMRs indirectly (e.g. through RMOS API calls) are disabled. Other tasks – even those with lower priorities – continue to run. Disabled tasks are continued immediately SMRs become available again.

Exception Interrupt Handler

The exception interrupt handler logs the processor exceptions of the 80x86 processor, and the unexpected interrupts.

The log output of the processor exception interrupts specifies the time and type of interrupt in the first line. The second line outputs the error code passed by the processor to the stack for exception interrupts 8, 10, 11, 12, 13, 14 and 17. The fourth line provides more detailed information on the cause of the interrupt. Finally, the current register values are shown. The decoded flag register appears in the last line.

If, for example, an exception interrupt is initiated by a task in A state, the output appears as follows.

```
*** 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 )
```

If the exception interrupt was initiated by an interrupt routine in the I state, the fourth line appears as follows:

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

If the exception interrupt was initiated by an interrupt routine in the S state, the fourth line appears as follows:

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

In both of the last two cases, the exception interrupt handler halts the system.

<Exception-Text> depends on the exception interrupt and represents the following character strings:

INT-NUM	CHARACTER STRING
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:

Either AT ADDRESS or NEAR ADDRESS is output, depending on whether the EIP register contains the address of the initiating command or the address of the next command.

NMI Interrupt

The following character string is output with the NMI interrupt (INT 2):

```
*** nuc: <date> <time> NMI INTERRUPT
```

Unexpected Interrupts

The following message is output for unexpected interrupts:

```
*** nuc: <date> <time> UNEXPECTED INTERRUPT
```

4.2 M7 RMOS32 Exception Handler

An exception handler logs all RMOS API calls which are terminated with an error on the system console. The exception handler is not activated in the default setting (see User Manual, System Software for M7-300/400, Installation and Operation):

```
*** nuc: <date> <time>, svc <name> <state text>
    failed: <error number> (<error text>)
```

The meanings of the above are as follows:

<name>	Name of the decoded RMOS API call, e.g. RmGetFlag
<statetext>	Depending on the system state, one of the following texts is inserted when the RMOS exception handler is called. 1. from task: <name> id: 0xXX 2. during system startup 3. in monitor mode 4. in s–state 5. in i–state
<error nummer>	Error number
<error text>	Decoded error text

Example

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

4.3 Error Codes of RMOS API Calls

Return Values

In certain circumstances, an RMOS API call can generate an error. ***Error codes*** are therefore returned by all functions of the RMOS API. By checking the return value, you can determine whether or not the function was performed successfully. The data type of the return value is ***int.***

The error-free execution of an RMOS API call is indicated by the return value ***RM_OK*** (=0).

RM_OK:
No error has occurred.

Certain RMOS API calls return values which, instead of indicating an error, serve as memos for the caller. These memos always have a ***negative*** integer value (< 0).

Unsuccessful RMOS API calls contain error codes whose integer value is ***positive*** (> 0).

Overview: Memos

The following return values are memos, not error numbers. They have negative values.

RM_ENTRY_REMOVED:(-263)
The entry was removed from the catalog.

RM_ERROR_OUT_OF_RANGE:(-265)
Invalid error number.

RM_FLAG_ALREADY_SET:(-258)
A flag was already set.

RM_FLAG_RESET:(-260)
A flag was reset.

RM_FLAG_SET:(-259)
A flag was set.

RM_PRI_NOT_CHANGED:(-261)
The priority was not changed.

RM_TASK_RESUMED:(-256)
The task was resumed.

RM_TASK_WAITING:(-262)
The task had to wait for exception (for BLOCKED mode).

Overview: Error Codes

The following list shows the error codes which can be returned by RMOS API calls.

RM_ALL_DEBUGREGISTERS_USED:(45)
All debug registers are already being used.

RM_BOUND_REACHED:(27)
The boundary entered with `RmSetMailboxSize` has been exceeded.

RM_BREAKPOINT_ALREADY_SET:(29)
A breakpoint has already been set for the specified address.

RM_BREAKPOINT_ID_ALREADY_USED:(28)
The specified breakpoint ID has already been used.

RM_CATALOG_EXCEEDED:(100)
The configured number of possible catalog entries has been exceeded.

RM_GOT_TIMEOUT:(4)
An RMOS API call was aborted after the configured timeout.

RM_HEAP_NOT_REDEFINEABLE:(14)
The heap is already defined.

RM_INVALID_DESCRIPTOR:(5)
An invalid descriptor was used.

RM_INVALID_FUNCTION:(44)
An invalid or non-supported function number was passed.

RM_INVALID_ID:(36)
An invalid ID was passed.

RM_INVALID_INTERRUPT_NUMBER:(56)
The interrupt number was outside the valid range (0–255).

RM_INVALID_IRQ_NUMBER:(41)
An IRQ number was used for a PIC which has not been defined.

RM_INVALID_MEMORYBLOCK:(17)
An attempt was made to free an invalid memory area.

RM_INVALID_NULLPOINTER:(10)
A null pointer is not permitted at this point.

RM_INVALID_OFFSET:(39)
The offset was outside the valid range.

RM_INVALID_POINTER:(42)
A pointer was invalid.

RM_INVALID_SEGMENTLENGTH:(6)
An invalid segment length was specified.

RM_INVALID_SELECTOR:(21)
An invalid selector was used.

RM_INVALID_SIZE:(38)
A size parameter was invalid.

RM_INVALID_STRING:(37)
A string is not within the defined size.

RM_INVALID_TASK_ENTRY:(60)
Invalid task entry.

RM_INVALID_TASK_STATE:(22)
An illegal `RmKillTask` call was activated.

RM_INVALID_TYPE:(35)
An invalid parameter (*mode*, *type*, *pri_type*, etc.) was passed.

RM_IS_ALREADY_CATALOGED:(47)
The string to be cataloged has already been entered.

RM_IS_NOT_CATALOGED:(48)
The string is not cataloged.

RM_MEMORY_ALREADY_USED:(25)
The memory block to be reserved is already allocated.

RM_NO_MESSAGE:(43)
The mailbox (message queue) does not contain a message.

RM_NOT_HALTABLE:(46)
The task could not be halted.

RM_OUT_OF_FLAGGROUPS:(12)
The configured number of event flags has been exceeded.

RM_OUT_OF_MAILBOXES:(15)
The configured number of mailboxes has been exceeded.

RM_OUT_OF_MEMORY:(3)
No memory area of sufficient size is available.

RM_OUT_OF_MEMORYPOOLS:(13)
The configured number of memory pools has been exceeded.

RM_OUT_OF_SEMAPHORES:(16)
The configured number of semaphores has been exceeded.

RM_PARAMETER_ERROR:(2)
Incorrect parameters have been passed.

RM_QUEUE_EXIST:(59)
The message queue already exists.

RM_QUEUE_NOT_EXIST:(58)
No message queue exists.

RM_RESOURCE_BUSY:(18)
The resource to be deleted is busy.

RM_RESOURCE_NOT_AVAILABLE:(23)
The desired resource is not available.

RM_SVC_NOT_CONFIGURED:(33)
An attempt was made to execute a non-configured RMOS API call. Check the output of the RMOS exception handler to determine which RMOS API call is meant.

RM_TASK_DORMANT:(7)
The task is in the DORMANT state.

RM_TASK_KILLED:(49)
The task was deleted with the `RmKillTask` RMOS API call.

RM_TASK_NOT_DORMANT:(20)
An attempt was made to delete or start a task in the DORMANT state.

RM_TASK_NOT_IN_BP_CONTEXT:(31)
The task was not interrupted by a breakpoint.

RM_TASK_NOT_IN_RTE_HALT:(32)
The task was not interrupted by a runtime error.

RM_TASK_NOT_PAUSED:(26)
The task to be resumed with `RmResumeTask` was not halted with `RmPauseTask`.

RM_TEST_NOT_OK:(57)
A test was not successfully completed.

RM_TASK_NOT_READY:(30)
An attempt was made to halt a task which is not in the READY state.

4.4 Error Codes of M7 API Calls

Notes

Error codes are returned from the functions of the M7 API either in the return value of the function or – unlike the RMOS API – via a pointer variable.

The data type of the returned error code is M7ERR_CODE and is defined in the **M7API.H** file.

Since the functionality of the M7 API is presented by individual M7 servers, the error codes are classified accordingly.

General Errors

The following list shows the general error codes returned by M7 API calls. All constants are defined in the **M7API.H** header file.

M7SUCCESS:(0)
Function was successfully performed, no error occurred.

M7E_NO_MEM:(–1)
Function must allocate memory dynamically for execution, no memory available.

M7E_PAR:(–100)
An incorrect parameter was passed in the function call.

M7E_PRIO:(–3)
The priority passed in the function call is outside the valid range.

M7E_RESSOURCE_LIMIT:(–2)
No resources available

PSUB Interface

The following list shows the error codes returned by P BUS I/O drivers.

M7E_ALARM_GEN_DISABLED:(–121)
Alarm generation was disabled in data record 0.

M7E_Alarm_Pending:(–128)
There is still an Alarm which must be confirmed

M7E_BSY:(–104)
Local bus is busy.

M7E_CMD:(–105)
Local bus command error

M7E_COM_ERROR:(–110)
Module has aborted communication.

M7E_D_ALARM_BUSY:(–117)
Diagnostics alarm was not confirmed by CPU.

M7E_D_ALARM_GEN_DISABLED:(–119)
Diagnostics alarm disabled in data record 0

M7E_DP_SLAVE_STATE:(–123)
Action is not possible in the actual Slave–Status

M7E_DPX2_FAULT:(–124)
DPX2 call is stopped

M7E_GL_ALARM_DISABLED:(–122)
All alarms are disabled.

M7E_HWFAULT:(–101)
General hardware error

M7E_INVAL_DEV:(–126)
Error of Parameter

M7E_IO_DESC:(–109)
Incorrect I/O descriptor

M7E_NORM_DIAG:(–127)
Dates of diagnosis are not available

M7E_ODIS:(–120)
CPU has initiated ODIS (Output Disabled) signal.

M7E_P_ALARM_BUSY:(–116)
Process alarm has not yet been acknowledged by CPU.

M7E_P_ALARM_GEN_DISABLED:(–118)
Process alarm disabled in data record 0.

M7E_PARITY:(–106)
Local bus parity error

M7E_PEU:(–102)
Error in I/O expansion unit

M7E_QVZ:(–103)
Local bus timeout

M7E_REC_LENGTH:(–111)
Incorrect data record length

M7E_REC_NUMBER:(–112)
Incorrect data record number

S7 Object Server

The following list shows the error codes returned by the S7 object server.

M7E_BIT_OFFSET:(–203)
The bit offset within a byte is incorrect.

M7E_BLOCK_ROMDIR:(–211)
Cannot read block in ROMDIR directory

M7E_LENGTH:(–208)
The length specified in the read, write or create operation is 0.

M7E_LINK_PAR:(–214)
Parameters passed in `M7LinkDataAccess` or `M7LinkDataAccessCB` calls are incorrect.

M7E_NODIR:(–203)
The directory of S7 objects does not exist or cannot be read.

M7E_OBJ:(–200)
Object type is not supported by S7 object server.

M7E_OBJ_EXISTS:(–205)
The S7 object already exists.

M7E_OFFSET:(–202)
The offset specified in S7 object is incorrect.

M7E_OVS_WRONG_STATE:(–216)
Action is not allowed in the actual working state

M7E_PART:(–201)
The subarea specified for the object type is not available.

M7E_PART_INVALID:(–206)
Specified subarea number is invalid.

M7E_PER_BITS:(–213)
Bit addressing illegal in I/O area

M7E_SIZE:(–212)
The length information in the block header and the file length are different.

M7E_TYPE:(–207)
The specified data type is not supported.

M7E_WRITE_PROTECT:(–204)
The S7 object is write-protected.

OST Server

The following list shows the error codes returned by the OST (Operating State Transition) server.

M7E_OST_CPU_IN_STOP:(–306)
CPU is in STOP state.

M7E_OST_DENIED:(–308)
The requested operating state transition was denied by at least one task.

M7E_OST_ILLEGAL_PARAM_CPU:(–305)
Invalid CPU parameter

M7E_OST_MODE_SW_IN_STOP:(–304)
Operating mode selector of the module is set to STOP.

M7E_OST_NO_SUCH_FRB:(–301)
Specified TSFRB is not being processed.

M7E_OST_NO_SUCH_STATE:(–302)
Unknown operating state

M7E_OST_NO_SUCH_TRANSITION:(–300)
Unknown operating state transition

M7E_OST_TIMEOUT:(–307)
Requested operating state transition was cancelled with timeout.

M7E_OST_WRONG_STATE:(–303)
Operating state transition is not possible from present operating state.

FC Server

The following list shows the error codes returned by the FC (Free Cycle) server.

M7E_FSC_NO_SUCHCYCLE:(–400)
Unknown state

M7E_FSC_NO_SUCHFRB:(–401)
Specified FSCFRB is not being processed

Diagnosis Server

The following list shows the returned Error Codes from the Diagnosis–Server.

M7E_DIAG_NUMBER:(–500)
Wrong class (only allowed 0x0a or 0x0b)

M7E_DIAG_STATE:(–501)
Wrong working state

K BUS Interface

The following list shows the error codes returned by the communication functions.

M7E_KSUB_BLOCK_TOO_LARGE:(–604)
Specified buffer has insufficient capacity.

M7E_KSUB_CONN_ACTIVE:(–609)
The connection is active at the moment and may be not closed

M7E_KSUB_CONN_CLOSED:(–602)
Specified connection has already been closed.

M7E_KSUB_EOF:(–607)
End of file or end of directory.

M7E_KSUB_FILEIO:(–606)
Error during file handling.

M7E_KSUB_NO_SRV:(–603)
K BUS is not available.

M7E_KSUB_NO_SUCH_CONN:(–601)
Specified connection ID is invalid.

M7E_KSUB_NO_SUCH_FRB:(–605)
Specified COMMFRB is not being processed.

M7E_KSUB_PARAM:(–600)
Specified parameters are incorrect.

M7E_KSUB_REMOTE:(–608)
Execution error on remote server

M7E_KSUB_SDB_WAS_DELETED:(–611)
Connection deleted by STEP7, connection is no longer active

FRB Handling

The following list shows the error codes which may occur during the general processing of FRBs. The error code can be read out from the header of the FRB using macro `M7GetFRBErr` .

M7E_FRB_NOT_BUSY:(–700)
Specified FRB is not being processed.

M7E_FRB_NOT_IN_LIST:(–701)
Specified FRB is not in the linked internal FRB list.

M7E_FRB_ALREADY IN_LIST:(–702)
FRB is already included

Internal Errors

The following list shows the error codes which may occur during internal processing.

M7E_INTERNAL_ERROR:(-9901)
Internal error has occurred.

M7E_NOT_IMPLEMENTED:(-9900)
Server does not exist.

4.5 Error Codes for Loadable Drivers

This section describes the error codes which can be returned by the calls for loadable drivers. The corresponding numeric value and a brief explanation is provided in addition to definition.

Error Codes

The following error codes can occur with all loadable drivers (SER8250.DRV, 3964.DRV).

RM_EIO_PARAMETER 0×0401
Parameter error

RM_EIO_INVALID_CONTROL 0×0402
The specified control function is not supported

RM_EIO_INVALID_ACCESS 0×0403
Descriptor is not open for type of access used (Read/Write)

RM_EIO_UNIT_RESERVED 0×0404
Unit is already reserved or unit was not reserved by the calling task

RM_EIO_CANCEL 0×0405
Request was canceled by RM_IOCTL_CANCEL

RM_EIO_LOCKED 0×0406
The unit has been locked by RM_IOCTL_LOCK

RM_EIO_IO_ERROR 0×0407
Request canceled due to I/O error

RM_EIO_PARITY_ERROR 0×0408
Request canceled due to parity error

RM_EIO_OVERRUN_ERROR 0×0409
Request canceled due to overrun error

RM_EIO_TIMEOUT 0×040A
Request canceled with timeout

RM_EIO_INVALID_STATE 0×040B
An error has occurred during status check of the controller (e.g. parity)

RM_EIO_NO_HARDWARE 0×040C
Hardware does not exist or is defective

RM_EIO_INIT_FAILED 0×040D
Initialization of the unit was not possible

RM_EIO_UNIT_RESET 0×040E
Request canceled by RM_IOCTL_RESET

Notes

The following messages can occur as return values

RM_IO_QUEUED× −1024
Request appended to queue

RM_IO_IN_PROGRESS −1025
Request currently being processed

RM_IO_NO_DATA× −1026
No data exist

Error Codes for 3964(R) Driver

The following errors can also occur with the 3964(R) driver (3964.DRV):

RM_EIO_3964_NO_TIMER 0x480
No timer could be started

RM_EIO_3964_BUFFER_OVERFLOW 0x481
More data were received than specified in the read request

RM_EIO_3964_UNEXPECTED_CHARACTER 0x482
Unexpected character received

RM_EIO_3964_CHECKSUM_ERROR 0x483
Error in checksum (with 3964R protocol)

RM_EIO_3964_REQUEST_SUSPENDED 0x484
The request was terminated because of an initiation conflict (master and slave transmitting simultaneously)

RM_EIO_3964_CONNECTION_REFUSED 0x485
Reserved

RM_EIO_3964_TRANSFER_ABORT 0x486
The communication partner has canceled the transfer (send or receive) with NACK

RM_EIO_3964_READ_CANCELED 0x487
Read request canceled with RM_IOCTL_CANCEL

RM_EIO_3964_WRITE_CANCELED 0x488
Write request canceled with RM_IOCTL_CANCEL

4.6 Error codes of C Runtime Library

Structure of Error Messages

Error messages of the C runtime library (CRUN) are output as follows:

```
*** crun: <date> <time>, <error message>
    caused by task id: <taskid>: '<taskname>'
```

<date>	Date on which error occurred
<time>	Time at which error occurred
<error message>	Actual error message
<taskid>	ID of task which caused error
<taskname>	String used to enter the task which caused the error in the resource catalog

Example:

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

The error messages are also output on the system console.

Error Messages

Error messages of the C runtime library (CRUN)

<function>: cannot allocate memory
No more memory could be allocated for internal operations in CRUN function <function>.

<function> not configured – task aborted
Function <function> was called by a downloadable task, but is not configured for the interface for downloadable tasks. The calling task was terminated with exit.

<function>: unknown hsfs return value xxxx
An HSFS call was terminated with the (unexpected) error code xxxx in CRUN function <function>.

automatic xinitc failed – task aborted
The automatic initialization of CRUN (see also xinitc) failed. The task which caused the automatic CRUN initialization was aborted with exit.

automatic xinitt failed – task aborted
The automatic initialization of a task within CRUN (see also xinitt) failed. The task which caused the automatic CRUN initialization was aborted with exit.

catalog entry "ERRLOG" not found
The "ERRLOG" entry was not found in the resource catalog. CRUN can therefore not use the error logger task for error output. Instead, it outputs error messages on the system console via the BYT driver.

`fclose: cannot delete temporary file`
A temporary file created with tmpfile could not be deleted when closing with `fclose.`

`illegal function code xxxx – task aborted`
The invalid function code xxxx was passed to the interface for downloadable tasks. The calling task was terminated with `exit.`

`reserved function code xxxx – task aborted`
The reserved function code xxxx was passed to the interface for downloadable tasks. The calling task was terminated with `exit.`

4.7 Error Codes of the Socket Interface

This section describes the error codes which can be returned by the calls of the Socket Interface. The corresponding numeric value and a brief explanation is provided in addition to definition. In addition standard error codes of the C Runtime Library may be assigned to *errno*(see description of ***errno***).

EWOULDBLOCK 61
The sockt is in nonblocking mode and the function cannot be executed

EINPROGRESS 62
The call is now in progress

EALREADY 63
Operation already in progress

EDESTADDRREQ 64
A destination address is required

EMSGSIZE 65
Message too long

EPROTOTYPE 66
Wrong protocol type for socket

ENOPROTOOPT 67
Protocol not available

EPROTONOSUPPORT 68
Protocol not ksupported

ESOCKNOSUPPORT 69
Socket type not supported

EOPNOTSUPP 70
Operation not supported on socket

EPFNOSUPPORT 71
Protocol family not supported

EAFNOSUPPORT 72
Address family not supported

EADDRINUSE 73
Port number or address already in use

EADDRNOTAVAIL 74
Wrong IP address

ENETDOWN 75
Driver not correctly initialised

ENETUNREACH 76
Network is unreachable

ENETRESET 77
Network has been reset and connection has been released

ECONNABORTED 78
Die Verbindung ist abgebaut.

ECONNRESET 79
Connection reset by peer

ENOBUFS 80
No more memory available for another socket or another connection

EISCONN 81
Socket is already connected.

ENOTCONN 82
Socket is not connected.

ESHUTDOWN 83
Can’t send after socket shutdown

ETOOMANYREFS 84
Too many references

ETIMEDOUT 85
Connection timed out

ECONNREFUSED 86
Connection refused

EBUFTOOSMALL 87
Buffer too small for this operation

ESMODEXISTS 88
Socket module already exists

ENOTSOCK 89
The socket operation on non-socket.

EDEADLOCK 90
Deadlock

EHOSTDOWN 91
Communication host not active

EHOSTUNREACH 92
Communication host unrachable

ENOURGENTDATA 93
No urgent data available

EMAYBEISO 95
Invalid protocol on peer

M7 API

5

In this chapter

M7_SWAP_DWORD

Function

Convert doubleword from Intel to SIMATIC representation and vice-versa

Syntax

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

Parameters

Parameter Name	Meaning
x	Doubleword (M7 data type DWORD, 32 bits) in Intel or SIMATIC representation

Description

The function converts a doubleword (M7 data type DWORD) from the Intel representation to a doubleword in SIMATIC representation (Motorola format) and vice-versa.

The call is implemented as a macro. No type checking is performed on the input parameter.

Return Value

Doubleword in Intel representation if input parameter in SIMATIC representation

Doubleword in SIMATIC representation if input parameter in Intel representation

See Also

M7_SWAP_WORD

M7_SWAP_WORD

Function **Convert word from Intel to SIMATIC representation and vice-versa**

Syntax

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

Parameters

Parameter Name	Meaning
x	Doubleword (M7 data type DWORD, 32 bits) in Intel or SIMATIC representation

Description

The function converts a doubleword (M7 data type DWORD) from the Intel representation to a doubleword in SIMATIC representation (Motorola format) and vice-versa.

The call is implemented as a macro. No type checking is performed on the input parameter.

Return Value

Doubleword in Intel representation if input parameter in SIMATIC representation

Doubleword in SIMATIC representation if input parameter in Intel representation

See Also **M7_SWAP_DWORD**

M7BUBCycRead

Function **Set up job for cyclical read**

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);
```

Parameters

Parameter Name	Meaning
flags	Flags A_IMMEDIATE: If this flag is set, the job is started immediately, otherwise the registered job must be started explicitly with `M7BUBCycReadStart` A_ZERO_FLAG: This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
ConnID	Connection reference from an `M7KInitiate`call.
pCommFRB	Pointer to a function request block for asynchronous communication.
nVars	Number of variables to be read, that is items in the address buffer.
pAddrBuffer	Pointer to an array with *nVars* elements. Each element is type **M7VARADDR** and specifies a contiguous area of items within an S7 object (see Chapter 3).
pDataBuffer	Pointer to an array with *nVars* elements. Each element is type **M7VARDATA** and specifies a buffer (address, size, etc.) for storing a variable (see Chapter 3). The individual buffers must be initialized in the global data or the heap before the above call is activated.
CycTime	Cycle time in ms. The following cycle times are possible: 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.
pnRequest	Pointer to the job number returned.
MPrio	Priority with which the message is dispatched (0–255).

Description

The `M7BUBCycRead` function sets up an MMI job for cyclical reading. The variable specification is stored in the address buffer and matches the specification in `M7BUBRead`. The data are transmitted asynchronously to the application.

The following conditions for the maximum user data length apply to the `M7BUBCycRead` call:

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

and

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

maxpdusize is the maximum PDU size for the connection opened with M7KInitiate and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

The application is informed about new data by the M7MSG_BUB_NRD message, and can fetch the data with `M7KEvent` .

Return Value

= M7SUCCESS The function was successfully executed (see Note).

< M7SUCCESS An error occurred.

Note

The return value M7SUCCESS does not guarantee that the whole read procedure was executed successfully. Additional information on the reset of the individual data transfer can be found in the component `AccessResult` in the structure **M7VARDATA**.

Error Codes

Error Codes	Meaning
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_BLOCK_TOO_LARGE	Insufficient buffer capacity
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_NO_MEM	No more memory available
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available

Error Codes	Meaning
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_PRIO	Incorrect priority
M7E_TYPE	Data type is invalid

See Also **M7BUBCycReadDelete, M7BUBCycReadStart, M7BUBCycReadStop**

M7BUBCycReadDelete

Function **Delete job for cyclical read**

Syntax

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

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call
nRequest	Job number from `M7BUBCycRead`

Description

The `M7BUBCycReadDelete` function deletes an MMI job for cyclical reading set up with `M7BUBCycRead`.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7BUBCycRead, M7BUBCycReadStart, M7BUBCycReadStop**

M7BUBCycReadStart

Function **Start job for cyclical read**

Syntax

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

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call
nRequest	Job number from `M7BUBCycRead`

Description The `M7BUBCycReadStart` function starts an MMI job for cyclical reading set up with `M7BUBCycRead`.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7BUBCycRead, M7BUBCycReadDelete, M7BUBCycReadStop**

M7BUBCycReadStop

Function **Stop job for cyclical read**

Syntax

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

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate`call
nRequest	Job number from `M7BUBCycRead`

Description

The `M7BUBCycReadStop` function stops an MMI job for cyclical reading started with `M7BUBCycRead` or `M7BUBCycReadStart` .

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7BUBCycRead, M7BUBCycReadDelete, M7BUBCycReadStart**

M7BUBRead

Function **Read MMI variable**

Syntax

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

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
nVars	Number of variables to be read, that is items in the address buffer.
pAddrBuffer	Pointer to an array with *nVars* elements. Each element is type **M7VARADDR** and specifies a contiguous area of items within an S7 object (see Chapter 3).
pDataBuffer	Pointer to an array with *nVars* elements. Each element is type **M7VARDATA** and specifies a buffer (address, size, etc.) for storing a variable. The individual buffers must be initialized in the global data or the heap before the above call is initiated.
pnBytes	Pointer to variable. This variable returns the number of bytes actually read.

Description

The `M7BUBRead` function starts a synchronous call for reading the variables specified in the *pAddrBuffer* address array into the data buffer specified in the *pDataBuffer* array.

The following conditions for the maximum user data length apply to the `M7BUBRead` call:

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

and

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

maxpdusize is the maximum PDU size for the connection opened with M7KInitiate and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

Return Value

= M7SUCCESS	The function was successfully executed (see Note).
< M7SUCCESS	An error occurred.

Note

The return value M7SUCCESS does not guarantee that the whole read procedure was executed successfully. Additional information on the reset of the individual data transfer can be found in the component `AccessResult` in the structure **M7VARDATA**.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_BLOCK_TOO_LARGE	Insufficient buffer capacity
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_NO_MEM	No more memory available
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7BUBWrite

M7BUBWrite

Function **Write MMI variable**

Syntax

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

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
nVars	Number of variables to be written.
pAddrBuffer	Pointer to an array with *nVars* elements. Each element is type **M7VARADDR** and specifies the data type, the block type, the block number and the start offset of the variables to be overwritten in the data area of the S7 object server (M7) or in the S7 CPU data area.
pDataBuffer	Pointer to an array with *nVars* elements. Each element is type **M7VARDATA** and specifies a buffer (address, size, etc.) for storing a value with which the variable in the data area of the S7 object server (M7) or in the S7 CPU data area is to be overwritten.

Description

The `M7BUBWrite` function starts a synchronous call for overwriting the variables specified in the *pAddrBuffer* address array with the values specified indirectly in the *pDataBuffer* data array.

The address and data specifications match those of `M7BUBRead`.

The following conditions for the maximum user data length apply to the `M7BUBWrite` call:

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

maxpdusize is the maximum PDU size for the connection opened with M7KInitiate and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Note

The return value M7SUCCESS does not guarantee that the whole write procedure was executed successfully. Additional information on the reset of the individual data transfer can be found in the component `AccessResult` in the structure **M7VARDATA**.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_NO_MEM	No more memory available
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7BUBRead

M7CheckResource

Function **Check battery and SRAM**

Syntax

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

Parameters

Parameter Name	Bedeutung	
pFlags	Pointer to flags.	
	M7SRAM_OK	SRAM is free of error
	M7BATTERY_OK	There is at least one battery free of error
	M7BATTERY_CHARGE_OK	All batteries are free of error
	If one of the bits is not set, the corresponding resource has an error.	

Description

The `M7CheckResource` function is used to check the SRAM and battery.

The battery back–up for a M7 300 CPU/FM is on the module (one battery), for a M7 400 CPU it is on the power supply of the central rack (two batteries).

Note

M7VARDATA The `M7CheckResource` function is not supplied on a FM 456–4. `M7CheckResource` returns on a FM 456–4 always BATTERY_OK.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

M7ClearPI

Function **Clear process image**

Syntax

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

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs

Description

The function resets the entire process image specified by the *PIType* parameter to ’0’.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType*

See Also **M7LoadPII, M7StorePIQ**

M7ConfirmCycle

Function **Confirm FC server message**

Syntax

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

Parameters

Parameter Name	Meaning
pFSCFRB	Pointer to the FRB which is to be confirmed.

Description The function confirms a message of the type M7MSG_CYCLE. The FC server waits for all registered FRBs to be confirmed.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_FSC_NO_SUCH_CYCLE	Unknown state
M7E_FSC_NO_SUCH_FRB	FSCFRB is not registered
M7E_FRB_NOT_BUSY	Specified FRB is not being processed

See Also **M7LinkCycle, M7UnLinkCycle**

M7ConfirmDiagAlarm

Function

Confirm diagnostics alarm

Syntax

#include <m7api.h>

M7ERR_CODE M7ConfirmDiagAlarm(
M7DIAGALARM_FRB_PTR *pDAFRB***);**

Parameters

Parameter Name	Meaning
pDAFRB	Pointer to the FRB of the diagnostics alarm to be confirmed.

Description

The function confirms a diagnostics alarm.

When a diagnostics alarm has occurred, a new diagnostics alarm cannot be received by the initiating module until the currently registered diagnostics alarm has been confirmed. Diagnostics events which occur in the mean time are stored on the module.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist or has not initiated alarm
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DPX2_FAULT	Error in DP job for alarm confirmation
M7E_SLAVE_TYPE	Alarms from DP standard slaves do not have to be confirmed
M7E_DP_SLAVE_STATE	DP–SLAVE is not in DATA state
M7E_INVAL_DEV	Module of a DP–Slaves is not available

Additional Error Messages in FRB

Further error messages can be stored in the FRB of the registered diagnostics alarm. These can be read out with the following C macro:

error = M7GetFRBErrCode(pDiagFrb);

The *error* variable must be of the type *M7ERR_CODE*.

The meaning of the FRB error messages is listed in the following table.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout

See Also

M7LinkDiagAlarm, M7GetDiagAlarmAddr, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo,M7GetDiagAlarmPT ype, M7UnlinkDiagAlarm

M7ConfirmIOAlarm

Function

Confirm process alarm

Syntax

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

Parameters

Parameter Name	Meaning
pPAFRB	Pointer to the FRB of the alarm to be confirmed.

Description

The function confirms a process alarm.

When a process alarm has occurred, a new process alarm cannot be received from the same module until the currently registered process alarm has been confirmed. Process alarms which occur in the mean time are stored on the module.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist or has not initiated alarm
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DPX2_FAULT	Error in DP job for alarm confirmation
M7E_DP_SLAVE_STATE	DP–SLAVE is not in DATA state
M7E_INVAL_DEV	Module of a DP–Slaves is not available

Additional Error Messages in FRB

Further error messages can be stored in the FRB of the registered process alarm. These can be read out with the following C macro:

error = M7GetFRBErrCode(pIOFrb);

The *error* variable must be of the type *M7ERR_CODE*.

The meaning of the FRB error messages is listed in the following table.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout

See Also

M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPType, M7UnLinkIOAlarm

M7ConfirmPeriodicTimer

Function **Confirm periodic time message**

Syntax

#include <m7api.h>

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

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the FRB used to register the periodic time message.

Description

The call confirms a periodic time message. If confirmation is configured when registering an FRB for periodic time messages, the time server does not send a new time message until the previous one has been confirmed.

The call is implemented as a C macro. The system does not check whether the pointer *pTFRB* references a valid FRB.

The number of lost time messages can be checked with the `M7GetLost-Periods` function.

See Also **M7LinkPeriodicTimer, M7UnLinkPeriodicTimer, M7GetLostPeriods**

M7ConfirmTransition

Function

Confirm message for operating state transition

Syntax

#include <m7api.h>

M7ERR_CODE M7ConfirmTransition(
M7TSFRB_PTR *pTSFRB***,**
BOOL *AllowTransition***);**

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB to be confirmed.
AllowTransition	This flag can be used to inhibit the transition to STARTUP or RUN. To suppress the transition after STARTUP or RUN, pass *FALSE*, otherwise pass *TRUE*.

Description

The function confirms a message of the type M7MSG_TRANSITION.

The OST server does not change to the new operating state until all tasks registered by the FRB for the new operating state transition have been confirmed.

On request of all operating states except for STARTUP and RUN, the operating state transition is performed regardless of whether *TRUE* or *FALSE* was specified in the *AllowTransition* parameter. Confirmation must always take place, however.

Note

When the STOP-to-STARTUP transition is rejected (***M7ConfirmTransition(.. AllowTransition=FALSE***), then no M7MSG_STATE message is issued upon reaching the STARTUP state.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_OST_NO_SUCH_TRANSITION	Unknown operating state transition in FRB
M7E_OST_NO_SUCH_FRB	FRB is not being processed

See Also

M7GetTSReason, M7GetTSType, M7LinkTransition, M7UnLinkTransition

M7ConfirmZSAlarm

Function

Confirm insert/remove–module alarm

Syntax

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

Parameters

Parameter Name	Meaning
pZSFRB	Pointer ro insert/remove FRB

Description

`M7ConfirmZSAlarm` confirms an insert/remove–module alarm.

The `M7ConfirmZSAlarm` function must be called up by the user after evaluation of the insert/remove–module information, so that the FRB allocated by the system with the insert/remove–module alarm can be released again.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	Specified FRB is not in the linked internal FRB list.

See Also

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

M7CreateObject

Function **Create an S7 object**

Syntax

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

Parameters

Parameter Name	Meaning
ObjType	Identifier for S7 object. which can be set up by the user program on an M7 are listed in Table 2-7.
Part	Subarea number. The permissible values are listed in Table 2-8.
Count	Number of elements of which the S7 object is to consist; indirectly defines the length of the S7 object., this value has always to be even.
Ptr	Pointer to the memory area for the execution-related part of the object. If the value NUL is specified for *Ptr*, the object server allocates the memory for the object independently.

Description

The function creates an S7 object described by the above parameters. The object is subsequently linked automatically.

You can define the memory for the object yourself, or leave the memory allocation to the object server. If you define the memory yourself, you should make sure that there is sufficient capacity for the desired object.

Note

When you create a data block, you can use the numbers (*part* parameter) 0 to 65535. The area for the numbers is **not** limited by the numeric range permitted on the S7 CPU.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length or even number of bytes.
M7E_NO_MEM	Working memory allocated or error on memory request.

Error Code	Meaning
M7E_OBJ	Object type not supported.
M7E_OBJ_EXISTS	Block already exists.
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PART	Subarea does not exist.
M7E_RESOURCE_LIMIT	Resources exceeded.
M7E_REM_OBJ	Illegal action because the object is retentive

See Also **M7StoreObject, M7DeleteObject, M7RemoveObject, M7LocateObject**

M7DeleteObject

Function **Delete S7 object from working memory and delete BACKDIR**

Syntax

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

Parameters

Parameter Name	Meaning
ObjType	Identifier for S7 object. The identifiers of possible S7 objects are listed in Table 2-7.
Part	Subarea number. The subarea numbers of the S7 objects are listed in Table 2-8.

Description The function deletes an S7 object described by *ObjType* and *Part* from the working memory **and** from the BACKDIR catalog.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_OBJ	Object type not supported.
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PART	Subarea does not exist.
M7E_REM_OBJ	Illegal action because the object is retentive
M7E_WRITE_PROTECT	Objectwrite-protected.

See Also **M7CreateObject, M7LocateObject, M7RemoveObject, M7StoreObject**

M7DiagMode

Function

Link or unlink diagnostics

Syntax

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

Parameters

Parameter Name	Meaning	
flags	Flags	
	A_BESYMSG	Operating system diagnostics message
	A_SYSMSG	System diagnostics message
	A_USERMSG	User-defined diagnostics message
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
ConnID	Connection reference from an `M7KInitiate` call	
pCommFRB	Function request block for asynchronous communication	
pszUserName	The application uses this string (max. 8 bytes) to identify itself to the server	
Mprio	Priority with which the message was dispatched (0–255).	

Description

The M7DiagMode function is used to reset the diagnostics filter of the user. An application can register itself for the appropiate diagnostics messages using the flags A_BESYMSG, A_SYSMSG and A_USERMSG, which are sum-totalled. Disabled flags indicate deregistration.

Incoming messages are indicated by M7MSG_DIAG_MSG.

When an M7MSG_DIAG_MSG is received, the job number for the current message can be checked with `M7GetCommRequest`.

The following job numbers are possible:

Operating system messages have job number DIAG_BESYMSG.

System diagnostics messages have job number DIAG_SYSMSG.

User diagnostics messages have job number DIAG_USERMSG.

If both system and user messages are received, the job number is DIAG_SYS_USER_MSG.

The message itself must be initiated with the `M7KEvent` call.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_NO_SUCH_FRB	*M7COMMFRB not being processed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KEvent**

M7DPNormDiagnose

Function **Get standard diagnostics for a DP slave**

Syntax

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

Parameters

Parameter Name	Meaning
Baddr	Base address of ET ER
pBuffer	Pointer to data buffer for standard diagnostics frame

Description The function returns the diagnostics for a DP slave coded according to the DP standard.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect base address.
M7E_NORM_DIAG	Diagnostics data is not available for the module.
M7E_NOT_IMPLEMENTED	L2–DP server not available

See Also **M7GetDiagAlarmInfo**

M7GetCBBitOffset

Function **Get bit offset within a callback function**

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description The function determines the bit offset of a variable, which another application is attempting to access via the S7 object server, from a CBFRB passed to a callback function.

The call is implemented as a C macro.

Return Value The bit offset is returned.

See Also **M7GetCBBuffer, M7GetCBByteOffset, M7GetCBCount, M7GetCBDataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart**

M7GetCBBuffer

Function **Get buffer address within a callback function**

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description

The function determines the address of the data buffer from a CBFRB passed to a callback function.

If the task has been registered for a write access with a callback function, the buffer contains the data with which variables of the S7 object server are to be overwritten.

In read accesses, it is used to store the variables to be read from the S7 object server.

The call is implemented as a C macro.

Return Value

The return value is a pointer to the buffer.

See Also

M7GetCBBitOffset, M7GetCBByteOffset, M7GetCBCount, M7GetCB-DataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBByteOffset

Function **Get byte offset within a callback function**

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description

The function determines the byte offset of a variable, which another application is attempting to access via the S7 object server, from a CBFRB passed to a callback function.

The call is implemented as a C macro.

Return Value

The byte offset is returned.

See Also

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

M7GetCBCount

Function **Get number of elements within a callback function**

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description

The function determines the number of elements, which another application is attempting to access via the S7 object server, from a CBFRB passed to a callback function.

The call is implemented as a C macro.

Return Value

The number of elements is returned.

See Also

M7GetCBBitOffset,M7GetCBBuffer, M7GetCBByteOffset, M7GetCB-DataType, M7GetCBFlags, M7GetCBObjType, M7GetCBPart

M7GetCBDataType

Function **Get data type within a callback function**

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description

The function determines the data type of the variables, which another application is attempting to access via the S7 object server, from a CBFRB passed to a callback function.

The call is implemented as a C macro.

Return Value

The data type is returned by the call.

The possible data types are listed in Table 2-9.

See Also **M7GetCBBitOffset, M7GetCBBuffer,M7GetCBByteOffset, M7GetCBCount, M7GetCBFlags, M7GetCBObjType, M7GetCBPart**

M7GetCBFlags

Function **Get access type within a callback function**

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description

The function determines, from a CBFRB passed to a callback function, the actual access type (read, write, delete, etc.) with which another application is attempting to access variables on the S7 object server.

The call is implemented as a C macro.

Return Value

The actual access type is returned.

The possible data types are listed in the following table:

Access Type	Type Identifier
Read S7 object variable	M7READ_ACCESS
Write S7 object variable	M7WRITE_ACCESS
Create S7 object variable	M7CREATE_ACCESS
Delete S7 object variable	M7DELETE_ACCESS
Link S7 object	M7LINK_ACCESS

See Also **M7GetCBBitOffset, M7GetCBBuffer,M7GetCBByteOffset, M7GetCB-Count, M7GetCBDataType, M7GetCBObjType, M7GetCBPart**

M7GetCBObjType

Function **Get type identifier of S7 object within a callback function**

Syntax

```
#include <m7api.h>
UBYTE          M7GetCBObjType(
                 M7CBFRB_PTR pCBFRB);
```

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the CBFRB passed by the M7 API when the callback function is called.

Description

The function determines the type identifier of the S7 object, which another application is attempting to access, from a CBFRB passed to a callback function.

The call is implemented as a C macro.

Return Value

The type identifier of the S7 object type is returned (see Table 2-7).

See Also

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

M7GetCBPart

Function

Get the subarea number of the S7 object within a callback function

Syntax

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

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the *CBFRB* passed by the M7 API when the callback function is called.

Description

The function determines the subarea number of the S7 object, which another application is attempting to access, from a CBFRB passed to a callback function.

The call is implemented as a C macro.

Return Value

The type identifier of the S7 object type is returned (see Table 2-7).

See Also

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

M7GetCommRcvLen

Function **Get length of received data after** `M7PBKBrcv` **call**

Syntax

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

Parameters

Parameter Name	Meaning
pFRB	Pointer to the FRB from which the length is to be read.

Description

The `M7GetCommRcvLen` call determines the length of received data from the FRB referenced by *pFRB* after receiving an M7MSG_PBK_NDR message.

The call is implemented as a C macro.

Return Value

The number of received bytes by a `M7PBKBrcv` call is returned.

M7GetCommRequest

Function

Get job number

Syntax

#include <m7api.h>

UDWORD M7GetCommRequest(
M7COMMFRB_PTR *pFRB*);

Parameters

Parameter Name	Meaning
pFRB	Pointer to the FRB from which the job number is to be read.

Description

The `M7GetCommRequest` call determines the job number from the FRB referenced by *pFRB* after receiving an M7MSG_PBK_DONE, M7MSG_PBK_NDR, M7MSG_BUB_NDR or M7MSG_DIAG_MSG message.

The messages are sent by the PBK, MMI and diagnostics calls.

The call is implemented as a C macro.

Return Value

The job number is returned.

See Also

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

M7GetCommStatus

Function **Check return status of application link**

Syntax

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

Parameters

Parameter Name	Meaning
pFRB	Pointer to the FRB from which the PBK status is to be read.

Description

The `M7GetCommStatus` call evaluates the *pFRB* after receiving an M7MSG_PBK_DONE or M7MSG_PBK_NDR message. These messages are sent by the calls `M7PBKPut`, `M7PBKGet`, `M7PBKBsend` or `M7PBKBrcv` .

The call is implemented as a C macro.

Return Value

Possible results are listed in the following table:

Status	Meaning
M7COMMSTATE_OK	Job terminated without error
M7COMMSTATE_NO_CONN	Communication problems
M7COMMSTATE_NACK	Negative acknowledgement, function not executable
M7COMMSTATE_RID_UNKNOWN	Unknown R_ID or Receive has not been called.
M7COMMSTATE_WRONG_DATA	Number of data areas or individual data types do not match
M7COMMSTATE_RES_REQ	Reset request detected
M7COMMSTATE_REM_BLCK_ DISABLED	Remote block DISABLED
M7COMMSTATE_REM_ WRONG_STATE	Remote partner in incorrect state
M7COMMSTATE_REM_ ACCESS_DENIED	Access error on remote partner
M7COMMSTATE_OVERRUN	Receive data were overwritten by new data
M7COMMSTATE_MEM_ ACCESS_DENIED	Access to local user memory denied

Status	Meaning
M7COMMSTATE_NOT_ FINISHED	Previous job not yet finished
M7COMMSTATE_TERM_ BY_USER	Job was canceled by user

See Also **M7PBKBrcv, M7PBKBsend, M7PBKGet, M7PBKPut, M7GetCommRequest**

M7GetConnStatus

Function

Scan status of an application link

Syntax

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

Description

The `M7GetConnStatus` function permits determination of the status of an application link specified with ConnID.

The following states have been defined (M7_CONN_STATE):

M7_CNST_CLOSED	The application link is closed
M7_CNST_CONNECTING	The application link is just being established
M7_CNST_CONNECTED	The application link is established
M7_CNST_DISCONNECTING	The application link is just being closed

The K bus functions `M7KAbort` and `M7GetConnStatus` can be called up via a valid *ConnID* irrespective of the status of an application link.

All other K bus functions specific to application link are processed in the M7_CNST_CONNECTED state only. In other states, these calls are rejected with M7E_KSUB_CONN_CLOSED.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SUCH_CONN	Specified connection ID is invalid.

See Also

M7KAbort, M7KInitiate

M7GetDiagAlarmAddr

Function

Read logical base address for diagnostics alarm from FRB

Syntax

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

Parameters

Parameter Name	Meaning
pDiagFrb	Pointer to FRB from which address is to be read.

Description

The call returns the logical base address of the module that initiated the alarm from the FRB referenced by *pDiagFrb*.

The call is implemented as a C macro.

Return Value

The return value is the logical base address of the module that initiated the alarm.

See Also

M7LinkDiagAlarm, M7UnLinkDiagAlarm, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo,M7GetDiagAlarmPType, M7ConfirmDiagAlarm

M7GetDiagAlarmBusy

Function **Check status of a diagnostics alarm from M7/S7 CPU**

Syntax

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

Parameters

Parameter Name	Meaning
pError	Pointer to a variable of the type M7ERR_CODE.

Description The function determines whether a diagnostics alarm sent to the M7/S7 CPU has been acknowledged by the M7/S7 CPU.

Return Value If the function is executed successfully, it returns the identifier of the current alarm state. The meaning of the state identifiers is listed in the following table.

State Identifier	Meaning
TRUE	The alarm is still waiting.
FALSE	The alarm was detected by the S7/M7 CPU and has been processed.

Error Codes **pError* is always 'M7SUCCESS'

See Also **M7SendDiagAlarm**

M7GetDiagAlarmInfo

Function

Read diagnostics information from FRB

Syntax

```
#include <m7api.h>
void        M7GetDiagAlarmInfo(
              M7DIAGALARM_FRB_PTR pDiagFrb,
              UBYTE_PTR *Info);
```

Parameters

Parameter Name	Meaning
pDiagFrb	Pointer to the FRB from which the diagnostics information is to be read.
Info	Pointer to a buffer in which the 4 bytes containing the diagnostics information are to be stored.

Description

The call returns the 4 bytes containing the diagnostics information for a diagnostics alarm from the FRB referenced by *pDiagFrb*. The diagnostics information is module-specific.

The call is implemented as a C macro.

Return Value

The function stores the diagnostics information in the buffer referenced by *Info*.

See Also

M7LinkDiagAlarm, M7UnLinkDiagAlarm, M7GetDiagAlarmBusy, M7GetDiagAlarmAddr, M7GetDiagAlarmPType, M7ConfirmDiagAlarm

M7GetDiagAlarmPType

Function

Read identifier for the signal module of a diagnostics alarm from FRB

Syntax

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

Parameters

Parameter Name	Meaning
pDiagFrb	Pointer to the FRB from which the identifier is to be read.

Description

The call returns the identifier of the signal module for a diagnostics alarm from the FRB referenced by *pDiagFrb* when the `M7LinkDiagAlarm` function is called with the parameter *pType*.

The call is implemented as a C macro.

Return Value

The identifier for the module type is returned.

I/O Type	Meaning
M7IO_IN	Module is input module
M7IO_OUT	Module is output module

See Also

M7LinkDiagAlarm, M7UnLinkDiagAlarm, M7GetDiagAlarmBusy, M7GetDiagAlarmAddr, M7GetDiagAlarmInfo, M7ConfirmDiagAlarm

M7GetFlags

Function **Read registered access type from FRB**

Syntax

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

Parameters

Parameter Name	Meaning
pOBJFRB	Pointer to the OBJFRB passed on linking of communication for S7 object access.

Description

The call returns the *flags* parameter from the OBJFRB referenced when linking with `M7LinkDataAccess`.

The call is implemented as a C macro.

Return Value

The *flags* parameter is returned by the function. The *flags* parameter represents the access type specified on linking.

The possible access types are listed in the following table:

Type of Access	Identifier
Read S7 objects	M7READ_ACCESS
Write S7 objects	M7WRITE_ACCESS
Create S7 objects	M7CREATE_ACCESS
Delete S7 objects	M7DELETE_ACCESS
Link S7 object	M7LINK_ACCESS

See Also **M7LinkDataAccess, M7UnLinkDataAcess, M7GetObjType, M7GetPart**

M7GetFRBErrCode

Function **Read FRBs**

Syntax

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

Parameters

Parameter Name	Meaning
pFrbHeader	Pointer to FRB header whose error identifier is to be read.

Description

The call returns the error identifier of the FRB referenced by *pFrbHeader*. The error identifier indicates the general error code that can occur during handling of the FRB.

The call is implemented as a C macro.

Return Value

The function returns the error identifier of the referenced FRB.

The possible error identifiers depend on the type of FRB.

See Also **GetFRBTag, SetFRBTag**

M7GetFRBTag

Function **Read identifier of an FRB**

Syntax

```
#include <m7api.h>
UWORD          M7GetFRBTag(
                 M7FRBHEADER_PTR pFRBHeader);
```

Parameters

Parameter Name	Meaning
pFRBHeader	Pointer to FRB whose identifier is to be read.

Description The call returns the identifier of the FRB referenced by the *pFrbHeader* parameter.

The call is implemented as a C macro.

Return Value The function returns the identifier of the referenced FRB.

See Also **M7SetFRBTag, GetFRBErrCode**

M7GetFSCType

Function **Read type of FC server message from FRB**

Syntax

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

Parameters

Parameter Name	Meaning
pFSCFRB	Pointer to FRB from which the address is to be read.

Description

This call can be used to determine, from an FC server message, the service (scan cycle checkpoint, free cycle, etc.) for which the application has registered on the FC server. All messages sent by the FC server have the message identifier M7MSG_CYCLE.

The call is implemented as a C macro.

Return Value

The type of service is returned.

The possible services of the FC server are listed in the following table:

Services of FC Server	Identifier
Scan cycle checkpoint	M7S_CYCLECONTROLPOINT
Free cycle	M7S_FREECYCLE
STARTUP	M7S_STARTUPCYCLE
Cycle overflow	M7S_CYCLEOVERFLOW

See Also **M7LinkCycle, M7ConfirmCycle, M7UnLinkCycle**

M7GetIOAlarmAddr

Function **Read logical base address for process alarm from FRB**

Syntax

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

Parameters

Parameter Name	Meaning
pIOFrb	Pointer to FRB from which the address is to be read.

Description

The call returns the logical base address of the module which initiated a process alarm from the FRB referenced by *pIOFrb*.

The call is implemented as a C macro.

Return Value

The function returns the logical base address of the module which initiated the process alarm.

See Also

M7LinkIOAlarm, M7UnLinkIoAlarm, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPType, M7ConfirmIOAlarm

M7GetIOAlarmBusy

Function **Check status of a process alarm from M7/S7 CPU**

Syntax

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

Parameters

Parameter Name	Meaning
pError	Pointer to a variable of the type M7ERR_CODE.

Description The function detects whether a process alarm sent to the M7/S7 CPU has been acknowledged by the M7/S7 CPU.

Return Value When the function is successful, it returns an identifier for the current alarm state. The meaning of the state identifiers is shown in the following table.

State Identifier	Meaning
TRUE	The alarm is still waiting to be processed.
FALSE	The alarm has been detected by the S7 CPU and processed.

Error Codes **pError* is always 'M7SUCCESS'

See Also **M7SendIOAlarm**

M7GetIOAlarmMask

Function **Read alarm mask for a process alarm from FRB**

Syntax

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

Parameters

Parameter Name	Meaning
pIOFrb	Pointer to FRB from which the alarm mask is to be read.

Description The call returns the alarm mask for a process alarm from the FRB referenced via *pIOFrb*.

The call is implemented as a C macro.

Return Value The return value is the alarm mask from the FRB.

See Also **M7LinkIOAlarm, M7GetIOAlarmAddr, M7UnLinkIOAlarm, M7GetIOAlarmState, M7GetIOAlarmPType, M7ConfirmIOAlarm**

M7GetIOAlarmState

Function **Read supplementary information for a process alarm from FRB**

Syntax

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

Parameters

Parameter Name	Meaning
pIOFrb	Pointer to the FRB from which the state information is to be read.

Description The call returns the supplementary information for a process alarm from the FRB referenced by *pIOFrb*. The supplementary information is module-specific and is given in Intel representation.

The call is implemented as a C macro.

Return Value The return value is the supplementary information from the FRB.

See Also **M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7UnLinkIOAlarm, M7GetIOAlarmPType, M7ConfirmIOAlarm**

M7GetIOAlarmPType

Function

Read identifier for the signal module of a process alarm from FRB

Syntax

#include <m7api.h>

UWORD M7GetIOAlarmPType(
M7IOALARM_FRB_PTR *pIOFrb***);**

Parameters

Parameter Name	Meaning
pIOFrb	Pointer to the FRB from which the identifier is to be read.

Description

The call returns the identifier of the signal module from the FRB referenced by *pIOFrb* and specified when calling the `M7LinkIOAlarm` function with the *pType* parameter.

The call is implemented as a C macro.

Return Value

The return value is the identifier for the I/O type.

I/O Type	Meaning
M7IO_IN	Module is input module
M7IO_OUT	Module is output module

See Also

M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7GetIOAlarmState, M7UnLinkIOAlarm, M7ConfirmIOAlarm

M7GetLostPeriods

Function **Check number of periodic time messages lost**

Syntax

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

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the FRB with which the periodic time messages were linked.

Description This function detects the number of periodic time messages which were not sent due to a missing acknowledement. The internal system counters for the lost periods are subsequently cleared.

Return Value The function returns the number of periodic time messages lost.

See Also **M7LinkPeriodicTimer, M7ConfirmPeriodicTimer, M7UnLinkPeriodic-Timer**

M7GetObjectInfo

Function **Read information about data structure of an S7 object**

Syntax

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

Parameters

Parameter Name	Meaning
ObjType	Type identifier of the desired S7 object (see Table 2-7.)
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object.
pObjInfo	Pointer to a memory area with the **M7OBJ_INFO** data structure where the information about the S7 object is stored.

Description

The function returns all information about the data structure of an S7 object described by the parameters *ObjType* and *Part.* The memory for the information must be provided by the calling program.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PART	Subarea not available.
M7E_OBJ	Object type not supported.

See Also **M7CreateObject, M7DeleteObject, M7RemoveObject, M7LocateObject, M7StoreObject**

M7GetObjType

Function **Get type identifier for S7 object access**

Syntax

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

Parameters

Parameter Name	Meaning
pOBJFRB	Pointer to the OBJFRB referenced on linking for S7 object access.

Description

The call returns the type identifier of the object accessed from the OBJFRB referenced on communication by the S7 object server.

The call is implemented as a C macro.

Return Value

The return value is the type identifier of the S7 object type.

The possible type identifiers of the addressable S7 objects can be found in Table 2-7.

See Also **M7LinkDataAccess, M7UnLinkDataAccess, M7GetPart, M7GetFlags**

M7GetPart

Function

Get subarea number for S7 object access

Syntax

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

Parameters

Parameter Name	Meaning
pOBJFRB	Pointer to the OBJFRB referenced on linking for S7 object access.

Description

The call returns the subarea number of the object accessed from the OBJFRB referenced on communication by the S7 object server.

The call is implemented as a C macro.

Return Value

The return value is the subarea number of the S7 object type.

The possible subarea numbers for the addressable S7 objects are listed in the following table:

S7 Object	Type Identifier	Subarea Number
Data block	M7D_DB	DB number
Parameter data record, read	M7D_PAR_READ	DS number
Parameter data record, write	M7D_PAR_WRITE	DS number

See Also

M7LinkDataAccess, M7UnLinkDataAccess, M7GetOblType, M7GetFlags

M7GetPduSize

Function **Check maximum PDU size**

Syntax

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

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an M7KInitiate() call.
pnPduSize	Buffer for PDU size.

Description The function returns the maximum PDU size for a connection.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7PBKGet, M7PBKPut, M7BUBRead, M7BUBWrite**

M7GetPeriod

Function **Get multiple of time base from TFRB**

Syntax

```
#include <m7api.h>
UDWORD          M7GetPeriod
                  M7TFRB_PTR pTFRB);
```

Parameters

Parameter Name	Meaning
pTFRB	Pointer to FRB from which the multiple (parameter: *TimeBase*) of the time base is to be read.

Description

The call returns the *Period* parameter from the TFRB referenced by a periodic or one-shot time message. The *Period* parameter is specified when linking the FRB.

The call is implemented as a C macro.

Return Value

The call returns the *Period* parameter from the referenced TFRB.

See Also

M7LinkPeriodicTimer, M7LinkOneShotTimer, M7GetTimeBase

M7GetPIErrorAddr

Function **Get address of process image with transfer error**

Syntax

```
#include <m7api.h>
                    M7GetPIErrorAddr(
                      void *PIErrMsgBuf,
                      M7IO_LOGADDR Addr);
```

Parameters

Parameter Name	Meaning
PIErrMsgBuf	Message buffer for the process image transfer error
Addr	Address of the process image in which a transfer error occurred.

Description

The call accesses the process image transfer error message and returns the address at which a transfer error occurred in the variable *Addr*.

The call is implemented as a C macro.

See Also **M7GetPIErrorPIType, M7LinkPIError, M7UnLinkPIError**

M7GetPIErrorPIType

Function **Get type of process image with transfer error**

Syntax

```
#include <m7api.h>
                    M7GetPIErrorPIType(
                      void *PIErrMsgBuf
                      UBYTE PIType);
```

Parameters

Parameter Name	Meaning
PIType	Type of process image in which an error occurred.
	M7IO_PII Process image of inputs
	M7IO_PIQ Process image of outputs

Description

The call accesses the process image transfer error message and returns the type of process image in which an error occurred in the variable *PIType*.

The call is implemented as a C macro.

See Also **M7GetPIErrorAddr, M7LinkPIError, M7UnLinkPIError**

M7GetResetCause

Function **Query cause of reset**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7GetResetCause(
                  UDWORD *pState);
```

Parameters

Parameter Name	Bedeutung
pState	Shows the state. If one of the following bits is set the corresponding state applies. Several bits can also be set at the same time: M7WD_RESET — The system was previously reset by the watchdog. M7KEY_RESET — The system was previously reset by the key switch. If neither of the above bits is set, then the system was reset by a failure.

Description The function supplies the application with information on why the system was last stored.

Return Value = M7SUCCESS: The function was succesfully executed

M7GetState

Function **Check operating state**

Syntax

```
#include <m7api.h>
UWORD          M7GetState(void);
```

Description The function returns the current operating state.

Return Value The return value is an identifier for the current operating state. The meaning of the state identifiers is shown in the following table.

Parameters

State Identifier	Meaning
M7STATE_STOP	STOP operating state
M7STATE_STARTUP	STARTUP operating state
M7STATE_RUN	RUN operating state
M7STATE_HALT	HALT operating state
M7STATE_RESET	RESET operating state

See Also **M7LinkState, M7UnLinkState, M7RequestState,**

M7GetTime

Function **Read out date/time**

Syntax

#include <m7api.h>
M7ERR_CODE M7GetTime(M7TIME_DATE_PTR *pDateTime***);**

Parameters

Parameter Name	Meaning
pDateTime	Pointer to memory area with date/time structure

Description

The function reads the internal system time and date, and stores them in the memory area specified by *pDateTime*.

Please see Chapter 3 for details of the M7TIME_DATE structure.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

See Also **M7SetTime**

M7GetTimeBase

Function **Get time base from TFRB**

Syntax

```
#include <m7api.h>
UWORD          M7GetITimeBase(
                 M7TFRB_PTR pTFRB);
```

Parameters

Parameter Name	Meaning
pTFRB	Pointer to FRB from which the time base (parameter: *TimeBase*) is to be read.

Description

The call returns the *TimeBase* parameter from the TFRB referenced by a periodic or one-shot time message. The *TimeBase* parameter is specified when linking the FRB.

The call is implemented as a C macro.

Return Value

The call returns the *TimeBase* parameter from the referenced TFRB. Possible values of *TimeBase* are:

Return Value	Meaning	
TimeBase	Value for the time base:	
	M7TB_1MS:	1 ms
	M7TB_10MS:	10 ms
	M7TB_100MS:	100 ms
	M7TB_1S:	1s s

See Also **M7LinkPeriodicTimer, M7LinkOneShotTimer, M7GetPeriod**

M7GetTSReason

Function **Read reason for operating state/transition from FRB**

Syntax

```
#include <m7api.h>
UWORD          M7GetTSReason(M7TSFRB_PTR pTSFRB);
```

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB from which the reason for the operating state or operating state transition is to be read.

Description

When a state is attained, the `M7GetTSReason` macro can be used to check why a change to this state was output by `M7RequestState` . The value specified in the *Reason* parameter of an `M7RequestState` call is evaluated.

The call is implemented as a C macro.

Return Value The reason is returned from the FRB.

See Also **M7LinkTransition, M7UnLinkTransition, M7GetTSType, M7ConfirmTransition**

M7GetTSType

Function **Read operating state from an FRB**

Syntax

```
#include <m7api.h>
UWORD          M7GetTSType(M7TSFRB_PTR pTSFRB);
```

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB from which the operating state is to be read.

Description

The call returns an identifier for the operating state or operating state transition from a TSFRB of the OST server.

The call is implemented as a C macro.

Return Value

When a message of the type M7MSG_STATE (linked with `M7LinkState`) or M7MSG_REQ_FINISHED (requested with `M7RequestState`) is received, the following identifiers are possible in the referenced TSFRB:

Identifier	Meaning
M7STATE_STOP	M7 is in STOP state
M7STATE_STARTUP	M7 is in STARTUP state
M7STATE_RUN	M7 is in RUN state
M7STATE_HALT	M7 is in HALT state
M7STATE_RESET	M7 is in RESET state

When a message of the type M7MSG_TRANSITION (linked with `M7LinkTransition`) is received, the following identifiers are possible in the referenced TSFRB:

Identifier	Meaning
M7TRANS_STOPSTARTUP	Operating state transition from STOP to STARTUP requested
M7TRANS_STOPRESET	Operating state transition from STOP to RESET requested
M7TRANS_STARTUPSTOP	Operating state transition from STARTUP to STOP requested

Identifier	Meaning
M7TRANS_STARTUPRUN	Operating state transition from STARTUP to RUN requested
M7TRANS_STARTUPHALT	Operating state transition from STARTUP to HALT requested
M7TRANS_RUNSTOP	Operating state transition from RUN to STOP requested
M7TRANS_RUNHALT	Operating state transition from RUN to HALT requested
M7TRANS_HALTSTOP	Operating state transition from HALT to STOP requested
M7TRANS_HALTSTARTUP	Operating state transition from HALT to STARTUP requested
M7TRANS_HALTRUN	Operating state transition from HALT to RUN requested
M7TRANS_RESETSTOP	Operating state transition from RESET to STOP requested

See Also **M7LinkState, M7UnLinkState, M7RequestState, M7GetTSReason, M7LinkTransition, M7UnLinkTransition, M7ConfirmTransition**

M7GetZSAlarmAddr

Function

Get base address of an I/O module

Syntax

```
#include <m7api.h>
M7IO_BASEADDR  M7GetZSAlarmAddr(
                  M7ZSALARM_FRB_PTR pZSFRB,
                  UWORD SlotNum);
```

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the ZSFRB from which the base address of the I/O module is determined.
SlotNum	Number of the slot in which the module is installed. The slot number must be within the range 1 ... MAX_SLOT_400. The MAX_SLOT_400 constant identifies the maximum number of slots in the S7-400 system.

Description

The call returns the base address of the module at slot number *SlotNum* on an insert/remove module alarm.

The call is implemented as a C macro.

The function is only supported on the SIMATIC S7-400 system.

Return Value

The base address is returned by the call.

See Also

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7GetZSAlarmIdent

M7GetZSAlarmIdent

Function

Get identifier of an I/O module

Syntax

```
#include <m7api.h>
UBYTE          M7GetZSAlarmIdent(
                 M7ZSALARM_FRB_PTR pZSFRB,
                 UWORD SlotNum);
```

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the ZSFRB from which the identification number of the I/O module is determined.
SlotNum	Number of the slot in which the module is installed. The slot number must be within the range 1...MAX_SLOT_400. The MAX_SLOT_400 constant identifies the maximum number of slots in the S7-400 system.

Description

The call returns the identification number of the module at slot number *SlotNum* on an insert/remove module alarm.

The call is implemented as a C macro.

The function is only supported on the SIMATIC S7-400 system.

Return Value

The identification number is returned by the call. The identification number of a module is explained in the appropriate hardware description.

See Also

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmAddr, M7GetZSAlarmPType, M7GetZSAlarmMode

M7GetZSAlarmIMRBaddr

Function

Define base address of the IM module for which an insert/remove–module alarm was linked

Syntax

#include <m7api.h>

UBYTE M7GetZSAlarmIMRBaddr(
M7ZSALARM_FRB_PTR *pZSFRB***);**

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the ZSFRB

Description

The call returns information about the base address of the IM module which is installed in the rack or S7 slave on which the error occurred (CR_BADDR for the central rack).

The call is implemented as a C macro.

The function is only supported on the SIMATIC S7-400 system.

Return Value

The base address of the IM module is returned by the call.

See Also

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmPType, M7GetZSAlarmAddr, M7GetZSAlarmMode, M7GetZSAlarmIdent

M7GetZSAlarmMode

Function

Get mode of an I/O module

Syntax

```
#include <m7api.h>
UBYTE          M7GetZSAlarmMode(
                 M7ZSALARM_FRB_PTR pZSFRB,
                 UWORD SlotNum);
```

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the ZSFRB from which the mode of the I/O module is determined.
SlotNum	Number of the slot in which the module is installed. The slot number must be within the range 1 ... MAX_SLOT_400. The MAX_SLOT_400 constant identifies the maximum number of slots in the S7-400 system.

Description

The call returns the mode of the module at slot number *SlotNum* on an insert/remove module alarm.

The call is implemented as a C macro.

The function is only supported on the SIMATIC S7-400 system.

Return Value

An identifier for the mode is returned by the call. The possible values are listed in the following table:

Identifier	Meaning
M7DEV_OK	Module is OK
M7DEV_REM	Module has been removed
M7DEV_PUT	Module has been inserted

See Also

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmAddr, M7GetZSAlarmPType, M7GetZSAlarmIdent

M7GetZSAlarmPType

Function

Get I/O type of an I/O module

Syntax

```
#include <m7api.h>
UBYTE          M7GetZSAlarmPType(
                 M7ZSALARM_FRB_PTR pZSFRB,
                 UWORD SlotNum);
```

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the ZSFRB from which the type of I/O module is determined.
SlotNum	Number of the slot in which the module is installed. The slot number must be within the range 1 ... MAX_SLOT_400. The MAX_SLOT_400 constant identifies the maximum number of slots in the S7-400 system.

Description

The call returns the I/O type of the module at slot number *SlotNum* on an insert/remove module alarm.

The call is implemented as a C macro.

The function is only supported on the SIMATIC S7-400 system.

Return Value

The I/O type is returned by the call. The possible values are listed in the following table:

I/O Type	Meaning
M7IO_IN	Module is input module
M7IO_OUT	Module is output module

See Also

M7ConfirmZSAlarm, M7LinkZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmAddr, M7GetZSAlarmMode, M7GetZSAlarmIdent

M7InitAPI

Function **Initialize M7 API**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7InitAPI(void);
```

Description The function initializes the M7 API. The function must be called immediately at the start of the main routine in a C application program.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_NOT_IMPLEMENTED	M7 servers have not yet been started

M7InitISADesc

Function **Create I/O descriptor from logical address**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7InitISADesc(
                M7IO_LOGADDR Addr,
                UBYTE PType,
                UWORD Len,
                M7IO_DESC_PTR pIODesc);
```

Parameters

Parameter Name	Meaning
Addr	Logical address
PType	I/O Type M7IO_IN M7IO_OUT
Len	Length of the planned access. The following identifiers are possible: M7PBYTE: Descriptor for one byte M7PWORD: Descriptor for one word M7PDWORD: Descriptor for one doubleword
pIODesc	Pointer to initialized I/O descriptors. The user program must allocate the memory for the I/O descriptor from the global data area or the heap.

Description The function creates an I/O descriptor from the logical address. The I/O descriptor is used for high-speed access to the ISA bus I/O.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	The specified address does not describe an interface module; incorrect length or I/O type

See Also **M7StoreISAByte, M7StoreISAWord, M7StoreISADWord, M7LoadISAByte, M7LoadISAWord, M7LoadISADWord**

M7KAbort

Function **Close an application link**

Syntax

#include <m7api.h>
M7ERR_CODE M7KAbort(M7CONNID *ConnID***);**

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.

Description

The `M7KAbort` function closes an application link between the client and server. All asynchronous jobs for the connection are deleted.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed

See Also **M7KInitiate**

M7KEvent

Function **Fetch data of asynchronous messages**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7KEvent(
                M7CONNID ConnID,
                UDWORD nRequest,
                UBYTE_PTR pBuffer,
                UDWORD nBufsiz,
                UDWORD *pnBytes);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
nRequest	Job number. The job number can be read out from the FRB referenced in the message using the `M7GetCommRequest` call.
pBuffer	Pointer to the result buffer. The result buffer must be provided by the user program.
nBufsiz	Length of the result buffer.
pnBytes	Number of bytes read.

Description

The data generated by cyclical reading and diagnostics messages must be fetched from the driver with the `M7KEvent` function.

The next message with job number *nRequest* for connection reference *ConnID* is copied to the result buffer and deleted from the driver. The number of bytes transferred is stored in **pnBytes*.

If the result buffer is too small to store all the data of a message, as many data items as possible are copied, and an appropriate error code is set. If a matching message does not exist, the call returns without an error, and with **pnBytes* equal to 0.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed

Error Code	Meaning
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7BUBCycRead, M7DiagMode**

M7KInitiate

Function **Set up application link for communication via communication bus/MPI**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7KInitiate(
                M7CONNID *pConnID,
                UBYTE_PTR pHostAddr);
```

Parameters

Parameter Name	Meaning
**pConnID*	Pointer to the connection reference for further communication calls
pHostAddr	Address of the destination computer

Description

The `M7KInitiate` function opens an application link to a server via MPI or K bus. The host address of the remote partner is passed in a string. *pHostAddr* contains the connection number from the connection configuration. The connection number can be entered in decimal as well as in hexadecimal format (not case sensitive). For example: 0x1d0. The “local” string is passed in order to set up a unidirectional loop-back connection for the own CPU/FM.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_RESOURCE_LIMIT	Resources exceeded
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server

See Also **M7KAbort**

M7KPassword

Function

Password for functions with special protection level

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7KPassword(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UBYTE_PTR pszPassword);
```

Parameters

Parameter Name	Meaning
flags	Flags SET_PASSWORD: If this flag is enabled and the correct password is entered, the connection is legitimized; that is all functions are subsequently available. A_ZERO_FLAG: If set, the connection is enabled; that is functions are subsequently only available with the appropriate protection level password. This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
ConnID	Connection reference from an `M7KInitiate` call.
pszPassword	Pointer to an 8-byte password.

Description

The M7/S7 CPU has a password and a protection level entered in SDB0. Following an `M7KInitiate` call, the application can only execute functions on the current protection level. The application must be legitimized with the correct password to enable execution of all functions.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7KInitiate

M7KReadTime

Function **Read time**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7KReadTime(
                M7CONNID ConnID,
                M7KTIME_PTR pBuffer,
                UDWORD nBufsize,
                UDWORD *pnBytes);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
pBuffer	Pointer to a data structure of the type **M7KTIME**. The data structure which stores the K bus time must be allocated by the user program from the global data or the heap.
nBufsize	Length of the **M7KTIME** structure.
pnBytes	Pointer to the number of bytes read.

Description

The `M7KReadTime` function reads the time from the server computer into the data structure provided. The number of bytes read is entered in **pnBytes.*

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KInitiate, M7KWriteTime**

M7KWriteTime

Function **Set time**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7KWriteTime(
                M7CONNID ConnID,
                M7KTIME_PTR pBuffer,
                UDWORD nBufsize);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
pBuffer	Pointer to a data structure of the type **M7KTIME** with the time to be set.
nBufsize	Length of the **M7KTIME** structure.

Description

The `M7KWriteTime` function sets the time on the destination computer to the value specified in *pBuffer*.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KReadTime, M7KInitiate**

M7LinkBatteryFailure

Function **Initialize FRB for battery monitoring and register on OST server**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkBatteryFailure(
                M7BAFFRB_PTR pBAFFRB,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pBAFFRB	Pointer to the FRB provided for registration. The FRB must be allocated in the user program from the global data or the heap.
MPrio	Priority of the M7MSG_BATTERY_FAILURE message to be sent (0–255).

Description

The `M7LinkBatteryFailure` function initializes an FRB and registers the FRB on the OST server for handling.

If the battery voltage falls below the threshold before or during handling of an FRB, the task receives a message of the type M7MSG_BATTERY_FAILURE with message priority *MPrio*.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PRIO	Incorrect priority

See Also **M7UnLinkBatteryFailure**

M7LinkCycle

Function **Initialize FRB and register on FC server**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkCycle(
                M7FSCFRB_PTR pFSCFRB,
                UWORD Cycle,
                unsigned int  MPrio);
```

Parameters

Parameter Name	Meaning
pFSCFRB	Pointer to the FRB registered for communication with the FC server.
Cycle	Specifies the state on which message is to be sent. M7S_CYCLECONTROLPOINT Message at scan cycle checkpoint M7S_FREECYCLE Message at start of free cycle M7S_STARTUPCYCLE Message for state: STARTUP M7S_CYCLEOVERFLOW Message on cycle time limit exceeded
MPrio	Priority with which a message is to be sent (0–255).

Description

The `M7LinkCycle` function initializes an FRB and registers the FRB on the FC server for handling. When the desired state specified in *Cycle* becomes active, the task receives a message of the type M7MSG_M_CYCLE.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Unknown state
M7E_PRIO	Incorrect priority

See Also **M7UnLinkCycle, M7ConfirmCycle**

M7LinkDataAccess

Function

Link S7 object for access information via message

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkDataAccess(
                M7OBJFRB_PTR pOBJFRB,
                UBYTE ObjType,
                UWORD Part,
                UWORD Flags,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pOBJFRB	Pointer to the FRB provided for link registration
ObjType	Type identifier of S7 object for which accesses are to be reported (see Table 2-7).
Part	Subarea (DB number, etc., see Table 2-8)
Flags	Mask for selecting which access is to be reported: M7READ_ACCESS: Read only M7WRITE_ACCESS: Write only M7CREATE_ACCESS: Message on create object M7DELETE_ACCESS: Message on delete object M7LINK_ACCESS: Message on link object
MPrio	Priority with which a message is to be sent (0–255).

Description

The function requests the object server to report access to the referenced S7 object by sending a message to the task.

The calling task can use *Flags* to determine **which** access type (for example write access) is to be reported. The *Flags* cannot be connected by a logic OR operation; only one access type is allowed.

When the function has been successfully executed, and an external access is made to the registered S7 object by another task or via communication, the object server sends one of the messages listed in the following table – according to the specified access type – after the access takes place.

Access	Message
Read access	M7MSG_DATA_ACCESS_R
Write access	M7MSG_DATA_ACCESS_W
S7 object deleted	M7MSG_DATA_ACCESS_DEL
S7 object created	M7MSG_DATA_ACCESS_CREATE

Return Value = M7SUCCESS Always returned by the call.

Error Codes

Error Code	Meaning
M7E_FRB_ALREADY_IN_LIST	FRB is already linked
M7E_LINK_PAR	Parameter error
M7E_OBJ	Object type not supported
M7E_PAR	Parameter error
M7E_PRIO	Incorrect priority

See Also **M7SetFRBTag, M7GetFRBTag, M7GetObjType, M7GetFlags, M7GetPart**

M7LinkDataAccessCB

Function

Link callback function for S7 access

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkDataAccessCB(
                M7CBFRB_PTR pCBFRB,
                UDWORD (*pCallback)(M7CBFRB_PTR pCBFRB),
                UBYTE ObjType,
                UWORD Part,
                UWORD Flags);
```

Parameters

Parameter Name	Meaning
pOBJFRB	Pointer to the FRB provided for link registration
**pCallback*	Pointer to the callback function
ObjType	Type identifier of S7 object for which accesses are to be reported (see Table 2-7).
Part	Subarea (DB number, etc., see Table 2-8)
Flags	Mask for selecting on which access types the callback function is to be called: M7READ_ACCESS: Read access M7WRITE_ACCESS: Write access M7CREATE_ACCESS: Message on create object M7DELETE_ACCESS: Message on delete object M7LINK_ACCESS: Message on link object

Description

The task uses the function to request the object server to call the callback function before a WRITE–, CREATE or LINK–ACCESS or after a READ–ACCESS of the specified S7 object.

The calling task can use Flags to determine on **which** access type (for example write access only).

Return Value

= M7SUCCESS Always returned by the call.

Error Codes

Error Code	Meaning
M7E_FRB_ALREADY_IN_LIST	FRB is already linked
M7E_LINK_PAR	Parameter error
M7E_OBJ	Object type not supported
M7E_PAR	Parameter error

See Also **M7GetCBBitOffset, M7GetCBBuffer,M7GetCBByteOffset, M7GetCB-DataType, M7GetCBCount, M7GetCBFlags, M7GetCBObjType, M7GetCBPart, M7UnLinkDataAccessCB**

M7LinkDate

Function **Link time-controlled time message**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkDate(
                M7TFRB_PTR pTFRB,
                M7TIME_DATE_PTR pDateTime,
                BOOL Periodic,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pTFRB	Pointer to time server FRB
pDateTime	Pointer to memory area with date/time structure where the time parameters for the function are stored (see Section 3).
Periodic	Selection for “once” or “daily”: M7ONCE Message once M7DAILY Daily message (date = start date)
MPrio	Priority with which a message is to be sent (0–255).

Description

The function registers an FRB for a time-controlled handling on the time server. When the date or time specified in **pDateTime* has been reached, the time server sends a message of the type M7MSG_TIMESERVER to the calling task. The message is transmitted in RUN mode with second accuracy (resolution = 1 second). If the system is not in the RUN mode when the specified time is reached, the message is delayed until the next transition into the RUN mode. If a task is simultaneously logged for operating state messages, the order in which the time–controlled messages and the operating state messages are received is undefined at the time of transition into the RUN mode. In non–periodic mode, the time server deletes the associated FRB after sending the time–controlled message.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error
M7E_PRIO	Incorrect priority
M7E_RESOURCE_LIMIT	Too many timer FRBs in operation

See Also **M7UnLinkDate**

M7LinkDiagAlarm

Function **Link diagnostics alarm for handling**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkDiagAlarm(
                M7DIAGALARM_FRB_PTR pDiagFrb,
                UBYTE PType,
                M7IO_BASEADDR Addr,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pDiagFrb	Pointer to the FRB provided for registration. The FRB must be allocated in the user program from the global data or the heap.
PType	Identifier for input or output module: M7IO_IN Input module M7IO_OUT Output module
Addr	Logical base address of the module sending diagnostics alarms
MPrio	Priority with which a message is to be sent (0–255).

Description

The function initializes an FRB header and registers the FRB for handling on the alarm server.

If the I/O module specified by *Addr* reports a **diagnostics alarm**, the calling task receives a message of the type M7MSG_DIAG_ALARM.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Addressed module does not exist.
M7E_INVAL_DEV	Diagnostics alarm can only be reported by ET ER for DP standard slaves.

See Also **M7UnLinkDiagAlarm, M7GetDiagAlarmAddr, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo, M7GetDiagAlarmPType**

M7LinkIOAlarm

Function **Link process alarm for handling**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkIOAlarm(
                M7IOALARM_FRB_PTR pIOFrb,
                UBYTE PType,
                M7IO_BASEADDR Addr,
                UDWORD AlarmMask,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pIOFrb	Pointer to the FRB provided for registration
PType	Identifier for input or output module: M7IO_IN Input module M7IO_OUT Output module
Addr	Logical base address of the module sending process alarms
AlarmMask	Alarm mask: 32 channels can be selected with the *AlarmMask*Parameters. Bit 2^0 is assigned to channel 0, bit 2^1 to channel 1, etc. Mask bit = 1 means that the channel is **not** processed; Mask bit = 0 means that the channel is processed.
MPrio	Priority with which a message is to be sent.

Description

The function initializes an FRB header and registers the FRB for handling on the alarm server.

If the I/O module specified by *Addr* reports a **process alarm**, the calling task receives a message of the type M7MSG_IO_ALARM.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Addressed module does not exist.
M7E_SLAVE_TYPE	Process alarms can only be reported by DP-S7 slave modules.
M7E_INVAL_DEV	Process alarms can only be generated by I/O modules and not by the ET-ER.

See Also

M7UnLinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmBusy, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPType

M7LinkOneShotTimer

Function

Link one-shot time message

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkOneShotTimer(
                  M7TFRB_PTR pTFRB,
                  UWORD TimeBase,
                  UDWORD Time,
                  unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the accompanying time server FRB
TimeBase	Value for the time base: M7TB_1MS: 1 ms M7TB_10MS: 10 ms M7TB_100MS: 100 ms M7TB_1S: 1 s
Time	Time (multiple of *TimeBase*, max. 4 198 404)
MPrio	Priority with which a message is to be sent.

Description

The function registers an FRB for processing of a one-shot time message on the time server. When the specified time has expired, the time server sends a message to the calling task and deletes the accompanying FRB. Time messages are sent only during the RUN operation state.

Note

Select the *TimeBase* and *Time* parameters such that the *TimeBase* parameter contains the largest possible value for the desired time interval. This minimizes the load on the system caused by the time server.

Example:
You want your task to receive a single time message from the time server after a time of 4s. In this case, select the value 'M7TB_1S' for *TimeBase* and the value '4' for *Time* (not: 'M7TB_100MS' for *TimeBase* and '40' for *Time*!).

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect value for *TimeBase*
M7E_PRIO	Incorrect priority
M7E_RESOURCE_LIMIT	Too many timer FRBs operational

See Also

M7UnLinkOneShotTimer

M7LinkPeriodicTimer

Function

Link periodic time message

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7LinkPeriodicTimer(
                  M7TFRB_PTR pTFRB,
                  UWORD TimeBase,
                  UDWORD Period,
                  BOOL Handshake,
                  unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the accompanying time server FRB
TimeBase	Value for the time base: M7TB_1MS: 1 ms M7TB_10MS: 10 ms M7TB_100MS: 100 ms M7TB_1S: 1s
Period	Duration of the periods (multiple of *TimeBase*, max. 4 198 404)
Handshake	Selection of mode: M7WITH_HANDSHAKE Acknowledgement-driven operation active M7NO_HANDSHAKE Acknowledgement-driven operation not active
MPrio	Priority with which a message is to be sent.

Description

The function registers an FRB for processing of a periodic time message on the time server.

When the specified time has expired, the time server sends periodic time messages to the calling task. Time messages are sent only during the RUN operation state.

In handshake mode (*Handshake* = M7WITH_HANDSHAKE), every periodic time message must be acknowledged by the receiving task with the `M7ConfirmPeriodicTimer` function.

A maximum number of 10 FRBs can be registered per M7 CPU or FM.

Note

Select the *TimeBase* and *Time* parameters such that the *TimeBase* parameter contains the largest possible value for the desired time interval. This minimizes the load on the system caused by the time server.

Example:
You want your task to receive a single time message from the time server after a time of 4s. In this case, select the value 'M7TB_1S' for *TimeBase* and the value '4' for *Time* (not: 'M7TB_100MS' for *TimeBase* and '40' for *Time*!).

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect value for *TimeBase*
M7E_PRIO	Incorrect priority
M7E_RESOURCE_LIMIT	Too many timer FRBs operational

See Also

M7UnLinkPeriodicTimer, M7ConfirmPeriodicTimer, M7GetLostPeriods

M7LinkPIError

Function **Initialize FRB for process image transfer error**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkPIError(
                M7FRBHEADER_PTR pPIEFRB
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pPIEFRB	Pointer to the FRB used to link the process image transfer error
MPrio	Priority of M7MSG_PI_ERROR message (0–255)

Description

The `M7LinkPIError` function initializes an FRB for the handling of process image transfer errors which occur in the free cycle.

If the free cycle server detects a PI transfer error, it sends the message M7MSG_PI_ERROR to every linked task. The message contains the process image type and the process image address at which the transfer error occurred.

The *MPrio* parameter can be used to define the priority of the M7MSG_PI_ERROR message.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_PRIO	Incorrect priority

See Also **M7UnLinkPIError**

M7LinkState

Function **Request message on specific operating state**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkState(
                M7TSFRB_PTR pTSFRB,
                UWORD State,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB provided for registration. The FRB must be allocated in the user program from the global data or the heap.
State	Specifies the operating state on which communication is to take place. A task can only register for one operating state with an FRB. The following values can be specified: M7STATE_STOP — STOP operating state attained M7STATE_STARTUP — STARTUP operating state attained M7STATE_RUN — RUN operating state attained M7STATE_HALT — HALT operating state attained M7STATE_RESET — RESET operating state attained
MPrio	Priority with which a message is to be sent.

Description

The function initializes an FRB header and registers the FRB for handling on the OST server.

When the operating state specified by the *State* parameter becomes active, the calling task is informed by a message of the type *M7MSG_STATE*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error
M7E_PRIO	Incorrect priority

See Also **M7UnLinkState, M7GetState, M7RequestState, M7GetTSType, M7GetTSReason**

M7LinkTransition

Function

Request message on specific operating state transition

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkTransition(
                M7TSFRB_PTR pTSFRB,
                UWORD Transition,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB provided for registration. The FRB must be allocated in the user program from the global data or the heap.
Transition	Specifies the operating state transition on which communication is to take place. A task can only register for one operating state transition with an FRB. The following values can be specified: M7TRANS_STOPSTARTUP STOP to STARTUP M7TRANS_STOPRESET STOP to RESET M7TRANS_STARTUPSTOP STARTUP to STOP MSTRANS_STARTUPRUN STARTUP to RUN M7TRANS_STARTUPHALT STARTUP to HALT M7TRANS_RUNSTOP RUN to STOP M7TRANS_RUNHALT RUN to HALT M7TRANS_HALTSTOP HALT to STOP M7TRANS_HALTSTARTUP HALT to STARTUP M7TRANS_HALTRUN HALT to RUN M7TRANS_RESETSTOP RESET to STOP
MPrio	Priority with which a message is to be sent.

Description

The function initializes an FRB header and registers the FRB for handling on the OST server.

Before the operating state transition specified by the *Transition* parameter takes place, the calling task is informed by a message of the type M7MSG_TRANSITION. The task must acknowledge this operating state transition.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error
M7E_PRIO	Incorrect priority

See Also

M7UnLinkTransition, M7GetTSReason, M7GetTSType

M7LinkZSAlarm

Function

Link message on insert/remove module alarm

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LinkZSAlarm(
                M7ZSALARM_FRB_PTR pZSFRB,
                UBYTE RackNo,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the FRB provided for handling the registration. The FRB must be allocated in the user program from the global data or the heap.
RackNo	Rack number
MPrio	Priority of the M7MSG_ZS_ALARM message (0–255).

Description

The function initializes an FRB for “insert/remove module” alarm handling and registers the FRB on the alarm server.

When an insert/remove–module alarm occurs in the rack or on the S7 slave in which the IM module with base address IMRBaddr is installed, the task receives the message M7MSG_ZS_ALARM.

The base address M7CR_BADDR must be registered for the central rack.

MPrio can be used to define the priority of the message.

The address of the insert/remove–module FRB with the insert/remove–module information is passed to the user in the message buffer. **This FRB is not the FRB used to link by the user, but is an FRB allocated by the system.**

After evaluation of the alarm, the user must confirm the insert/remove–module alarm with `M7ConfirmZSAlarm`, so that the system resource can be released again.

The function is only supported on the SIMATIC S7-400 system.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PRIO	Incorrect priority
M7E_PAR	Invalid *RackNo* value
M7E_NOT_IMPLEMENTED	Function not supported on S7-300

See Also

M7ConfirmZSAlarm, M7UnLinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7GetZSAlarmAddr, M7GetZSAlarmIdent

M7LoadBit

Function **Load bit from process image**

Syntax

```
#include <m7api.h>
BOOL        M7LoadBit(
              UWORD PIType,
              UWORD ByteOffset,
              UBYTE BitOffset,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal byte
BitOffset	Bit offset within the signal byte
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description The function addresses a bit in the process image defined by *PIType*, and returns the state of the bit.

Return Value The return value is the state of the addressed bit.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType*, *ByteOffset* or *BitOffset*

See Also **M7LoadByte, M7LoadDWord, M7LoadWord**

M7LoadByte

Function **Load byte from process image**

Syntax

```
#include <m7api.h>
UBYTE          M7LoadByte(
                 UWORD PIType,
                 UWORD ByteOffset,
                 M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal byte
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description The function addresses a byte in the process image defined by *PIType*, and returns the state of the addressed byte.

Return Value The return value is the state of the addressed byte.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType*, or *ByteOffset*

See Also **M7LoadBit, M7LoadDWord, M7LoadWord**

M7LoadDirect

Function

Read I/O area directly

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LoadDirect(
                VOID_PTR pBuffer,
                UWORD SizeOfItem,
                UWORD Count,
                M7IO_LOGADDR Addr);
```

Parameters

Parameter Name	Meaning
pBuffer	Pointer to the destination buffer
SizeOfItem	Size of an element in bytes. The following constants are predefined: M7PBYTE Pointer to elements of the type BYTE M7PWORD Pointer to elements of the type WORD M7PDWORD Pointer to elements of the type DWORD
Count	Number of elements
Addr	Logical address of the first element

Description

The function performs a direct access to the process I/O. The source, size, number and destination of the data to be read are defined by the call parameters.

The function does not convert the numeric representation (*SIMATIC/Intel*).

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	The device is not ready for data communication

See Also **M7LoadDirectByte, M7LoadDirectDWord, M7LoadDirectWord**

M7LoadDirectByte

Function **Read byte direct from I/O**

Syntax

```
#include <m7api.h>
UBYTE          M7LoadDirectByte(
                 M7IO_LOGADDR Addr,
                 M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
Addr	Logical address of the I/O byte
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description The function performs a direct access to the process I/O and reads a byte.

Return Value If the function is successfully executed, the return value is the byte read from the process I/O.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	The device is not ready for data communication

See Also **M7LoadDirect, M7LoadDirectDWord, M7LoadDirectWord**

M7LoadDirectDWord

Function **Read doubleword direct from I/O**

Syntax

```
#include <m7api.h>
UDWORD      M7LoadDirectDWord(
              M7IO_LOGADDR Addr,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
Addr	Logical address of the I/O doubleword
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description

The function performs a direct access to the process I/O and reads a doubleword.

The contents of the doubleword are converted from the *SIMATIC* format to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the doubleword read from the process I/O in *Intel* format.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	The device is not ready for data communication

See Also **M7LoadDirect, M7LoadDirectByte, M7LoadDirectWord**

M7LoadDirectWord

Function

Read word direct from I/O

Syntax

```
#include <m7api.h>
UWORD          M7LoadDirectWord(
                 M7IO_LOGADDR Addr,
                 M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
Addr	Logical address of the I/O word
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description

The function performs a direct access to the process I/O and reads a word.

The contents of the word are converted from the *SIMATIC* format to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the word read from the process I/O in *Intel* format.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	The device is not ready for data communication

See Also

M7LoadDirect, M7LoadDirectByte, M7LoadDirectDWord

M7LoadDWord

Function **Load doubleword from process image**

Syntax

```
#include <m7api.h>
UDWORD      M7LoadDWord(
              UWORD PIType,
              UWORD ByteOffset,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal byte
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description

The function addresses a doubleword in the process image defined by *PIType*, and returns the state of the addressed doubleword.

The contents of the doubleword are first converted from the *SIMATIC* to the *Intel* numeric representation.

Return Value

The return value is the state of the addressed doubleword in *Intel* format.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType*, or *ByteOffset*

See Also **M7LoadBit, M7LoadByte, M7LoadWord**

M7LoadISAByte

Function **Read byte direct from ISA bus I/O**

Syntax

```
#include <m7api.h>
UBYTE          M7LoadISAByte(
                 M7IO_DESC_PTR pIODesc,
                 M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
pIODesc	Pointer to I/O descriptor initialized with `M7InitISADesc`.
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description The function runs as a macro, performing a direct access to the ISA bus process I/O, using an I/O descriptor generated with `M7InitISADesc` , and reading in a byte.

Return Value If the function is successfully executed, the return value is the byte read from the ISA process I/O.

Error Codes

Error Code	Meaning
M7E_PAR	Data access to ISA bus is larger (in bytes) than specified in `M7InitISADesc`.

See Also **M7LoadISAWord, M7LoadISADWord, M7InitISADesc**

M7LoadISADWord

Function

Read doubleword direct from ISA bus I/O

Syntax

```
#include <m7api.h>
UDWORD      M7LoadISADWord(
              M7IO_DESC_PTR pIODesc,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
pIODesc	Pointer to I/O descriptor initialized with `M7InitISADesc`
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description

The function runs as a macro, performing a direct access to the ISA bus process I/O, using an I/O descriptor generated with `M7InitISADesc` , and reading in a doubleword (32 bits) in Intel format.

The contents of the doubleword are converted from the *SIMATIC* to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the doubleword (32 bits) read from the ISA process I/O in Intel format.

Error Codes

Error Code	Meaning
M7E_PAR	Data access to ISA bus is larger (in bytes) than specified in `M7InitISADesc`

See Also

M7LoadISAByte, M7LoadISAWord, M7InitISADesc

M7LoadISAWord

Function

Read word direct from ISA bus I/O

Syntax

```
#include <m7api.h>
UWORD        M7LoadISAWord(
               M7IO_DESC_PTR pIODesc,
               M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
pIODesc	Pointer to I/O descriptor initialized with `M7InitISADesc`.
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description

The function runs as a macro, performing a direct access to the ISA bus process I/O, using an I/O descriptor generated with `M7InitISADesc` , and reading in a word (16 bits) in Intel format.

The contents of the word are converted from the S*IMATIC* to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the word (16 bits) read from the ISA process I/O.

Error Codes

Error Code	Meaning
M7E_PAR	Data access to ISA bus is larger (in bytes) than specified in `M7InitISADesc`

See Also

M7LoadISAByte, M7LoadISADWord, M7InitISADesc

M7LoadPII

Function **Update process image of inputs**

Syntax

#include <m7api.h>
M7ERR_CODE M7LoadPII(UWORD *PIINo***);**

Parameters

Parameter Name	Meaning
PIINo	Number of process images parts on M7-400: 0 Complete process image 1 ... 8 Process image part M7-300: 0 Complete process image Process image parts are not supported

Description

The function updates the complete process image or the specified part of the process image of inputs.

Process image parts are only supported on the S7-400 system.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Incorrect *PIINo*
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout

See Also **M7StorePIQ, M7ClearPI**

M7LoadRecord

Function **Read data record from signal module**

Syntax

```
#include <m7api.h>
M7ERR_CODE M7LoadRecord(
    UBYTE RecordNum,
    VOID_PTR pBuffer,
    UBYTE Size,
    UBYTE PType,
    M7IO_BASEADDR Addr);
```

Parameters

Parameter Name	Meaning
RecordNum	Record number Range: 0 to 255
pBuffer	Pointer to a buffer in the working memory, to which the record is to be transferred.
Size	Length of the data record
PType	Identifier for the I/O area: M7IO_IN I/O area for inputs M7IO_OUT I/O area for outputs If the module is a mixed module, specify the area ID of the lowest address. If the addresses are the same, specify M7IO_IN.
Addr	I/O base address of module

Description

The function transfers a data record from an I/O module to a buffer referenced by the *pBuffer* call parameter.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_COM_ERROR	Error on transfer protocol handling
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout

Error Code	Meaning
M7E_REC_LENGTH	Module reporting incorrect record length
M7E_REC_NUMBER	Module reporting incorrect record number
M7E_DPX2_FAULT	Error on DP job for record transfer
M7E_DP_SLAVE_STATE	DP Slave not in DATA state
M7E_INVAL_DEV	Module of a DP slave is not available

See Also **M7LoadRecordEx, M7Store Record**

See Also **M7StoreRecord**

M7LoadRecordEx

Function

Read data record from signal module

Syntax

```
#include <m7api.h>
long        M7LoadRecordEx(
              UBYTE RecordNum
              VOID_PTR pBuffer
              UBYTE Size
              UBYTE PType
              M7IO_BASEADDR Addr);
```

Parameters

Parameter Name	Meaning
RecordNum	Record number Range: 0 to 255
pBuffer	Pointer to a buffer in the working memory, to which the record is to be transferred.
Size	Length of the data record
PType	Identifier for the I/O area: M7IO_IN I/O area for inputs M7IO_OUT I/O area for outputs If the module is a mixed module, specify the area ID of the lowest address. If the addresses are the same, specify M7IO_IN.
Addr	I/O base address of module

Description

The function transfers a data record from an I/O module to a buffer referenced by the *pBuffer* call parameter.

Unlike the `M7LoadRecord` function, `M7LoadRecordEx` allows data access without specifying the exact number of bytes to be read. If the maximum record length specified in the Size parameter is 240, the valid bytes of record *RecordNum* are read and transferred to *pBuffer*.

The return value contains the number of valid bytes in the data buffer (see below).

Return Value

>M7SUCCESS	The function was successfully executed. The return value contains the number of valid bytes in the data buffer, i.e. record length if data buffer ≥ record buffer length if data buffer < record
< M7SUCCESS:	An error occurred

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_COM_ERROR	Error on transfer protocol handling
M7E_HWFAULT	General hardware error
M7E_PAR	Addressed module does not exist
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_REC_LENGTH	Module reporting incorrect record length
M7E_REC_NUMBER	Module reporting incorrect record number
M7E_DPX2_FAULT	Error on DP job for record transfer
M7E_DP_SLAVE_STATE	DP Slave not in DATA state
M7E_INVAL_DEV	Module of a DP slave is not available

See Also

M7LoadRecord, M7Store Record

M7LoadWord

Function **Load word from process image**

Syntax

```
#include <m7api.h>
UWORD        M7LoadWord(
               UWORD PIType,
               UWORD ByteOffset,
               M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal byte
pError	Pointer to a variable of the type M7ERR_CODE in which an error code is to be stored.

Description

The function addresses a word in the process image defined by *PIType*, and returns the state of the addressed word.

The contents of the word are first converted from the *SIMATIC* to the *Intel* numeric representation.

Return Value

The return value is the state of the addressed word in *Intel* format.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType* or *ByteOffset*

See Also **M7LoadBit, M7LoadByte, M7LoadDWord**

M7LocateObject

Function **Change start address of user data area of an S7 object**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7LocateObject(
                UBYTE ObjType,
                UWORD Part,
                VOID_PTR Ptr
                BOOL Copy);
```

Parameters

Parameter Name	Meaning
ObjType	Identifier of an S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
Ptr	New start address of S7 object
Copy	Handling of new memory area TRUE The user data of the object are copied to the new memory area. FALSE The user data of the object are not transferred.

Description

The function changes the start address of the user data area of an S7 object described by the above parameters. The user data are either transferred to the new area or not, according to the *Copy* parameter. This function can not be used for objects in SRAM (retentive) can not be

When calling the function, you should make sure that sufficient memory is available after the new start address.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_OBJ	Object type not supported.
M7E_PART	Subarea not available.
M7E_REM_OBJ	Not allowed for retentive objects.

See Also **M7CreateObject, M7DeleteObject, M7RemoveObject, M7StoreObject**

M7OVSCompress

Function

Object management system compress

Syntax

#include <m7api.h>
M7ERR_CODE M7OVSCompress(M7CONNID *ConnID***);**

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate`call.

Description

The `M7OVSCompress` function is used to request memory compression on an S7 CPU (object management system compression).

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

Note

The `M7OVSCompress` function is available only for S7 CPU.

See Also

M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite

M7OVSDelete

Function **Delete blocks via object management system**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSDelete(
                  UDWORD flags,
                  M7CONNID ConnID,
                  UBYTE nBlks,
                  M7BLKLIST_PTR pBlkList);
```

Parameters

Parameter Name	Meaning
flags	One or both of the following flags must be enabled: A_PASSIV: Delete passive blocks. A_LINKED_IN: Delete linked-in blocks. If the block list contains only blocks of one block type, but both flags are enabled, the job is denied completely.
ConnID	Connection reference from an `M7KInitiate` call.
nBlks	Number of items in the block list. If *nBlks* is equal to 0, all blocks in the RAM memory are deleted.
pBlkList	Pointer to the block list containing the blocks to be deleted. The block list consists of **M7BLKLIST** structure entries. The **M7BLKLIST** structure is described in Chapter 3.

Description

T he `M7OVSDelete` function is used to delete the blocks specified in the block list in one unit. It is possible to delete both copied and linked modules. The blocks are only deleted if all the specified blocks are present.

The maximum number of blocks to be deleted is defined by the following value, according to the maximum PDU size (see `M7GetPduSize`):

```
max_no = (maxpdusize – 28)/8
```

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7OVSCompress, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite

M7OVSFindFirst

Function **Read out first entry from object management system directory**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSFindFirst (
                UDWORD flags,
                M7CONNID ConnID,
                UWORD BlkTyp,
                UWORD Language,
                M7BLKINFO_PTR pFFBlkInfo);
```

Parameters

Parameter Name	Meaning
flags	One or both of the following flags must be enabled: A_PASSIV: Find passive blocks. A_LINKED_IN: Find linked-in blocks. Additionally one or both of the following flags can be enabled: A_DIRECTORY Find blocks of the block type with the lowest type number A_LANGUAGE Find blocks in the specified programming language.
ConnID	Connection reference from an `M7KInitiate` call.
BlkTyp	If A_Directory was not specified, the parameter contains the block type: M7BLKTYP_OB Organization block M7BLKTYP_DB Data block M7BLKTYP_FC Function call M7BLKTYP_SFC System function call M7BLKTYP_FB Function block M7BLKTYP_SFB System function block

Parameter Name	Meaning	
Language	If A_LANGUAGE was specified, Language contains the programming language of the block to be found:	
	M7LANGTYP_HUELSE	Container for SFCs and SFBs
	M7LANGTYP_AWL	Block created in STL (statement list)
	M7LANGTYP_KOP	Block created in KOP (ladder diagram)
	M7LANGTYP_FUP	Block created in FUP (function block diagram)
	M7LANGTYP_SCL	Block created in SCL
	M7LANGTYP_DB	Block created with block editor
	M7LANGTYP_GRAPH	Block created with Graph 5
	M7LANGTYP_SDB	Block created with system data block editor
	M7LANGTYP_CPU	Block created dynamically by the CPU
pFFBlkInfo	Pointer to a FindFirst block information block structure of the type **M7BLKINFO** where a block which is found is entered (see Chapter 6).	

Description

`M7OVSFindFirst` returns the first directory entry in **pFFBlkInfo*, according to the parameters, and initiates a search sequence which can be continued with these parameters using `M7OVSFindNext`.

At least one of the two flags A_PASSIV and A_LINKED_IN must be specified. If A_PASSIV is specified, passive blocks are displayed. If A_LINKED_IN is specified, linked-in blocks are displayed.

If A_DIRECTORY is specified, the search finds blocks of the block type with the lowest type number. In this case, BlkTyp does not need to be specified.

If A_LANGUAGE is specified, the search finds blocks in the specified programming language.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_EOF	End of file or end of directory reached

Error Code	Meaning
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7OVSCompress, M7OVSDelete, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite**

M7OVSFindNext

Function

Resume reading of object management system directory

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSFindNext (
                UDWORD flags,
                M7CONNID ConnID,
                M7BLKINFO_PTR pFFBlkInfo);
```

Parameters

Parameter Name	Meaning
flags	The same flags must be specified as in `M7OVSFindFirst`.
ConnID	Connection reference from an `M7KInitiate` call.
pFFBlkInfo	Pointer to a FindFirst block information structure where a block which is found is entered (see `M7OVSFindFirst`).

Description

The same flags must be specified as in the preceding `M7OVSFindFirst`-call. `M7OVSFindNext` returns the next directory item in the search sequence in *pFFBlkInfo*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_EOF	End of file or end of directory reached
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSWrite

M7OVSLinkIn

Function **Object management system link-in**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7OVSLinkIn(
                  M7CONNID ConnID,
                  UBYTE nBlks,
                  M7BLKLIST_PTR pBlkList);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
nBlks	Number of items in the block list. If *nBlks* is equal to 0, all the copied blocks are linked.
pBlkList	Pointer to the block list containing the blocks to be linked. The block list consists of **M7BLKLIST** structure entries. The **M7BLKLIST** structure is described in Chapter 3.

Description

The `M7OVSLinkIn` function is used to activate the number *nBlks* of blocks located in the CPU in one unit.

The maximum number of blocks to be linked is defined by the following value, according to the maximum PDU size (see `M7GetPduSize`):

```
max_anzahl = (maxpdusize – 28)/8
```

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSMemMode, M7OVSRead, M7OVSWrite**

M7OVSMemMode

Function **Object management system set memory mode**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSMemMode(
                UDWORD flags,
                M7CONNID ConnID);
```

Parameters

Parameter Name	Meaning
flags	A_PLC_RAM: Set memory mode to RAM. A_PLC_EPROM: Set memory mode to EPROM. One (and only one) of the two flags must always be set.
ConnID	Connection reference from an `M7KInitiate` call.

Description

The `M7OVSMemMode` function can be used to switch the M7/S7 CPU memory to RAM or EPROM mode.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSRead, M7OVSWrite

M7OVSRead

Function **Object management system load**

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);
```

Parameters

Parameter Name	Meaning
flags	A_PASSIV: Load a passive block. A_LINKED_IN: Load a linked-in block. At least one of the two flags must be enabled. If both flags are enabled, A_HEADER must also be enabled. A_SSB: Read the interface description only. A_HEADER: Read the block header only. A_FILE If enabled, *pBuffer* specifies the name of the file in which the block is stored; otherwise the block is stored in memory.
ConnID	Connection reference from an `M7KInitiate` call.
pBitmap	One-byte bitmap. If A_HEADER mode is specified, the storage location of the object is returned. The returned bitmap can be combined logically with the following identifiers: M7BLKINFO_PASSIV Block is in load memory (copied) M7BLKINFO_ACTIV Block is in working memory (linked in) M7BLKINFO_RAM Block is in RAM or RAM mode M7BLKINFO_EPROM Block is in EPROM or EPROM mode M7BLKINFO_BESY Block is a component of the operating system
pBuffer	Receive buffer If A_FILE is enabled, *pBuffer* specifies the name of the file in which the block is stored
nBufsiz	Size of input buffer If A_FILE is enabled, *nBufsiz* is ignored.

Parameter Name	Meaning
BlkTyp	Block types: M7BLKTYP_OB Organization block M7BLKTYP_DB Data block M7BLKTYP_FC Function call M7BLKTYP_SFC System function call M7BLKTYP_FB Function block M7BLKTYP_SFB System function block
BlkNum	Number of block
pnBytes	Pointer to number of bytes read. or 0 if the block is stored in a file.

Description

This function loads a block of the M7/S7 CPU into a buffer area or as a file on the hard disk of the M7.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_BLOCK_TOO_LARGE	Insufficient buffer space
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_FILEIO	File handling error
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSWrite

M7OVSSetObjectHeader

Function **Set an S7 object header**

Syntax

```
#include <m7api.h>
M7ERR_CODE M7OVSSetObjectHeader(
                    UBYTE_PTR ptr,
                    UWORD BlkNum,
                    UDWORD nLength,
                    UBYTE Language,
                    UBYTE Type,
                    UBYTE Attribute,
                    UBYTE ProtectionLevel);
```

Parameters

Parameter Name	Meaning	
ptr	Pointer to the memory area in which the S7 object header is stored. The memory area must be at least S7_OBJECT_HEADER_LENGTH bytes in size.	
BlkNum	Block number	
nLength	Total length of block in bytes	
Language	Language in which the block was created:	
	M7LANGTYP_HUELSE	Container for SFCs and SFBs
	M7LANGTYP_AWL	Block created in STL (statement list)
	M7LANGTYP_KOP	Block created in LAD (ladder diagram)
	M7LANGTYP_FUP	Block created in FBD (function block diagram)
	M7LANGTYP_SCL	Block created in SCL
	M7LANGTYP_DB	Block created with block editor
	M7LANGTYP_GRAPH	Block created with Graph 5
	M7LANGTYP_SDB	Block created with system data block editor
	M7LANGTYP_CPU	Block created dynamically by the CPU
Type	Block types:	
	M7BLKTYP_OB	Organization block
	M7BLKTYP_DB	Data block
	M7BLKTYP_FC	Function call
	M7BLKTYP_SFC	System function call
	M7BLKTYP_FB	Function block
	M7BLKTYP_SFB	System function block

Parameter Name	Meaning
Attributes	Reserved, must be set to 0
WriteProtect	Access allowed: 0 Read/write 1 Read only 2 Reading and writing not allowed 3 Know-how protection

Description

The `M7OVSSetObjectHeader` function sets the header for a block to be written with the function `M7OVSWrite`. The total length of the block must be at least S7_OBJECT_HEADER_LENGTH.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_PARAM	Parameter error

See Also

M7OVSWrite

M7OVSWrite

Function **Object management system copy**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7OVSWrite(
                UDWORD flags,
                M7CONNID ConnID,
                UBYTE_PTR pBuffer,
                UDWORD nBufsiz,
                UWORD BlkTyp,
                UWORD BlkNum);
```

Parameters

Parameter Name	Meaning
flags	Flags A_UNCONDITIONAL If it is not enabled, an existing block of the same type with the same number is not overwritten. If A_UNCONDITIONAL is enabled, an existing block of the same type with the same number is overwritten. A_FILE If it is enabled, *pBuffer* points to a string with a file name. The specified file contains the block. A_ZERO_FLAG This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
ConnID	Connection reference from an `M7KInitiate` call.
pBuffer	Data buffer containing the data of the block. If A_FILE is enabled, *pBuffer* points to a string with a file name. The specified file contains the block.
nBufsiz	Length of the data buffer. Ignored if A_FILE is enabled.
BlkTyp	Block types: M7BLKTYP_OB Organization block M7BLKTYP_DB Data block M7BLKTYP_FC Function call M7BLKTYP_SFC System function call M7BLKTYP_FB Function block M7BLKTYP_SFB System function block
BlkNum	Number of block

Description

The `M7OVSWrite` function copies the specified block from the specified buffer or file to the memory of a remote S7 CPU or M7.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_FILEIO	File handling error
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

Note

A restart is not possible on the M7.

See Also

M7OVSCompress, M7OVSDelete, M7OVSFindFirst, M7OVSFindNext, M7OVSLinkln, M7OVSMemMode, M7OVSRead, M7OVSSetObjectHeader

M7PBKBrcv

Function **Block-oriented receive data via configured connections**

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);
```

Parameters

Parameter Name	Meaning
flags	Flags A_ZERO_FLAG — This flag can be connected with other options by an OR operation. It must be set if no other flag is used. A_USER — The A_USER Flag is used for controlling the parameter *pDstVar* (see below).
ConnID	Connection reference from an `M7KInitiate` call.
R_ID	Block identifier for the remote Bsend block or `M7PBKBsend` call.
pDstVar	Pointer to the receive buffer. A_USER not set Pointer to **one** structure of type **M7VARADDR**. It specifies a contiguous area of items of a local S7 object to which the received data are copied. A_USER set Pointer to a buffer to which the received data are written.
nLength	Total length of the buffer in bytes.
pCommFRB	Pointer to the function request block.
Mprio	Priority of the message dispatched (0–255).

Description

`M7PBKBrcv` starts an asynchronous communication job for a buffer of *nLength* bytes via the connection *ConnID* from a BSEND block or `M7PBKBsend` call with identifier *R_ID*. According to the specified *flags* parameter, the data are written either to a buffer in the address area of the task (flags=A_USER) or to the data area of the S7 object server (flags=0).

When the A_USER flag is not set, then the *nLength* parameter is not evaluated, but the buffer length is determined from one of the data structures pointed to by the parameter *pSrcVar* or *pDstVar* respectively. In this case

nLength can be assigned any value. Otherwise if the A_USER flag is set, you must assign *nLength* the buffer length.

When the data have been transferred from the local station, or an error has occurred, an M7MSG_PBK_NDR message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKBrcv` call and receipt of the M7MSG_PBK_NDR message.

After receipt of the M7MSG_PBK_NDR Message the number of the received bytes can be get by `M7GetCommRcvLen` call.

`M7PBKBrcv` calls can be canceled with `M7PBKCancel`.

If an error occurs in the asynchronous part, it can be read from the referenced M7COMMFRB with the `M7GetCommStatus` macro.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7GetCommRcvLen, M7PBKBsend, M7PBKCancel

M7PBKBsend

Function **Block-oriented send via configured connections**

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);
```

Parameters

Parameter Name	Meaning
flags	Flags A_USER: The A_USER Flag is used for controlling the parameters *pSrcVar* (see below). A_ZERO_FLAG: This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
ConnID	Connection reference from an `M7KInitiate` call.
R_ID	Block identifier for the remote BRCV block or `M7PBKBrcv` call.
pSrcVar	Pointer to the data to be sent. A_USER not set Pointer to **one** structure of type **M7VARADDR**. It specifies a contiguous area of items in a local S7 object. A_USER set Pointer to a buffer containing the data to be sent.
nLength	Total length of the buffer in bytes.
pCommFRB	Pointer to the function request block.
Mprio	Priority of the message dispatched (0–255).

Description

`M7PBKBsend` starts asynchronous transmission of a data area of *nLength* via the connection *ConnID* to the BRCV block specified by the *R_ID* identifier or the `M7PBKBrcv` call on the remote station.

If flags=A_USER, the data to be sent begin at the address specified by *pSrcVar.*

If flags=0, *pSrcVar* specifies the address of the variable to be sent in the address area of the S7 object server.

When the A_USER flag is not set, then the *nLength* parameter is not evaluated, but the buffer length is determined from one of the data structures pointed to by the parameter *pSrcVar* or *pDstVar* respectively. In this case *nLength* can be assigned any value. Otherwise if the A_USER flag is set, you must assign *nLength* the buffer length.

When the data have been transferred from the local station, or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKBsend` call and receipt of the M7MSG_PBK_DONE message.

`M7PBKBsend` calls can be canceled with `M7PBKCancel`.

If an error occurs in the asynchronous part, it can be read from the referenced M7COMMFRB with the `M7GetCommStatus` macro.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7GetCommStatus, M7PBKBrcv, M7PBKCancel

M7PBKCancel

Function **Cancel running send or receive job via configured connections**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKCancel(
                  M7CONNID ConnID,
                  M7COMMFRB_PTR pCommFRB);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
pCommFRB	Pointer to a function request block.

Description

`M7PBKCancel` cancels a running `M7PBKBsend`-, `M7PBKBrcv` or `M7PBKURcv` job. The send or receive job to be canceled is specified by the parameters *ConnID* and *pCommFRB* (see `M7PBKBrcv` or `M7PBKBsend`).

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_NO_SUCH_FRB	*M7COMMFRB not operational
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KInitiate, M7PBKBsend, M7PBKBrcv, M7PBKURcv**

M7PBKGet

Function **Start asynchronous variable reading via configured connections**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKGet(
                  M7CONNID ConnID,
                  UBYTE nVars,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pDstVar,
                  M7COMMFRB_PTR pCommFRB
                  unsigned int Mprio);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate`call.
nVars	Number of variables to be read.
pRemoteVar	Array with the address specifications **(M7VARADDR)**. It specifies the variables to be read from the remote station.
pDstVar	Array with the address specifications **(M7VARADDR)**. It specifies for receiving data the variables of the S7 object server of the local station .
pCommFRB	Pointer to the function request block.
Mprio	Priority of the message dispatched (0–255).

Description

`M7PBKGet` starts the asynchronous process for reading *nVars* from the variable area of the S7 object server or from the S7 CPU data area on the remote station into the variable area of the S7 object server on the local station.

The following conditions apply to the maximum user data length for the `M7PBKGet` call:

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

and

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

maxpdusize is the maximum PDU size for the connection opened with `M7KInitiate` and *nBytes(i)* is the number of bytes for the i-th variable, rounded up to the nearest even number.

pRemoteVar and *pDstVar* are pointers to arrays with *nVars* elements each. Each element specifies a contiguous area of items on the S7 object server or in the S7 CPU data area (see `M7BUBRead`).

When the data have been stored in the data area specified by *pDstVar*, an M7MSG_PBK_NDR message is created for *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKGet` call and receipt of the M7MSG_PBK_NDR message.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7KInitiate, M7PBKPut, M7BUBRead

M7PBKIAbort

Function

Close an application link (for internal SIMATIC station communication via non-configured connections)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKIAbort(
                  UBYTE IOID,
                  UWORD LADDR);
```

Parameters

Parameter Name	Meaning
IOID	Input or output address area (M7KIOID_IN, M7KIOID_OUT)
LADDR	I/O start address of remote station (0–MAX_LOG_ADDR)

Description

The `M7PBKIAbort` function closes an application link between a client and server which were set up with the functions `M7PBKIPut` or `M7PBKIGet`.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_ACTIVE	The connection to station *LADDR* is currently active and cannot be closed.

See Also

M7PBKIPut, M7PBKIGet

M7PBKIGet

Function

Start asynchronous variable reading (for internal SIMATIC station communication via non-configured connections)

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);
```

Parameters

Parameter Name	Meaning	
flags	Flags	
	CONT	If CONT is set the application link set up by the data transfer is retained. If CONT is not set the application link set up by the data transfer is closed again after the data transfer
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
IOID	Input or output address area (M7KIOID_IN, M7KIOID_OUT)	
LADDR	I/O start address of remote station (0–MAX_LOG_ADDR)	
pRemoteVar	Pointer to **one** structure of type **M7VARADDR**. It specifies a contiguous area of items of a S7 object in the remote station.	
pDstVar	Pointer to **one** structure of type **M7VARADDR**. It specifies for receiving data a variable of the S7 object in the local station .	
pCommFRB	Pointer to the function request block	
MPrio	Priority of the message sent (0-255)	

Description

`M7PBKIGet` starts asynchronous reading of a variable from the variable area of the S7 object server on the remote station *LADDR* to the variable area of the S7 object server on the local station.

An application link with the remote station is set up if one does not already exist. If the `CONT` flag is enabled, the link remains intact after the end of data transfer. When the application link is no longer required, it must be closed with the `M7PBKIAbort` call. If the `CONT` flag is not enabled, the application link is closed again automatically after the end of data transfer.

pRemoteVar and *pDstVar* are pointers to elements which specify a contiguous area of items in the S7 object server (see `M7BUBRead`).

If the data are stored in the data area specified by *pDstVa*r, an M7MSG_PBK_NDR message is created for *pCommFRB*. The FRB may not be used for any other purpose in the time between the M7PBKIGet call and receipt of the M7MSG_PBK_NDR message.

Note

The user data length amount to 76 byte.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_CONN_ACTIVE	The connection to station *LADDR* is currently active. No data can be transferred.
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_REMOTE	Execution error on server
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7GetCommStatus, M7PBKIAbort, M7PBKIPut, M7BUBRead

M7PBKIPut

Function

Start asynchronous variable writing (for internal SIMATIC station communication via non-configured connections)

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);
```

Parameters

Parameter Name	Meaning	
flags	Flags	
	CONT	If CONT is set the application link set up by the data transfer is retained.
		If CONT is not set the application link set up by the data transfer is closed again after the data transfer
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
IOID	Input or output address area (M7KIOID_IN, M7KIOID_OUT)	
LADDR	I/O start address of remote station (0–MAX_LOG_ADDR)	
pRemoteVar	Pointer to **one** structure of type **M7VARADDR**. It specifies the variables to be overwritten in the S7 object server or the S7 CPU data area of the remote station	
pSrcVar	Pointer to **one** structure of type **M7VARADDR**. It specifies the variables to be sent in the S7 object server of the local station	
pCommFRB	Pointer to the function request block	
MPrio	Priority of the message sent (0-255)	

Description

`M7PBKIPut` starts asynchronous writing of a variable in the S7 object server or the S7 CPU data area of the remote station *LADDR* with the values of a local variable of the S7 object server.

An application link with the remote station is set up if one does not already exist. If the `CONT` flag is enabled, the link remains intact after the end of data transfer. When the application link is no longer required, it must be closed

with the `M7PBKIAbort` call. If the `CONT` flag is not enabled, the application link is closed again automatically after the end of data transfer.

pRemoteVar and *pSrcVar* are pointers to the address specifications of the remote or local variables in the S7 object server/S7 CPU data area.

When the data have been stored on the remote computer, or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the M7PBKIPut call and receipt of the M7MSG_PBK_DONE message.

Note

The user data length amount to 76 byte.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_CONN_ACTIVE	The connection to station *LADDR* is currently active. No data can be transferred.
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_REMOTE	Execution error on server
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7GetCommStatus, M7PBKIAbort, M7PBKIGet, M7BUBWrite

M7PBKPrint

Function **Send data with a format description**

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);
```

Parameters

Parameter Name	Meaning
flags	Flags
ConnID	Connection ID
printerID	Printer ID
fmt	Format string (null-terminated)
n_Vars	Number of send parameters
pSrcVar	Send parameters
pCommFRB	Pointer to the function request block
MPrio	Priority of the message sent (0–255)

Description

`M7PBKPrint` starts asynchronous sending of multiple data areas and a format string via connection *ConnID* to the remote station.

The *nVars* parameter specifies the number of data areas to be transferred. *pSrcVar* points to an array of M7VARDATA objects. Each of these objects contains a data area to be sent.

The *fmt* parameter points to a null-terminated format string.

When the data have been accepted by the remote station or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKPrint` call and receipt of the M7MSG_PBK_DONE message.

If an error occurs in the asynchronous component, it can be read out from the referenced M7COMMFRB with the macro `M7GetCommStatus`.

The following conditions apply to the maximum user data length for the `M7PBKPrint` call:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize \quad 26 \quad längefmt \quad 4 * nVars$$

maxpdusize is the maximum PDU size for the connection opened with `M7KInitiate` and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7KInitiate

M7PBKPut

Function **Start asynchronous variable writing via configured connections**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKPut(
                  M7CONNID ConnID,
                  UBYTE nVars,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pSrcVar,
                  M7COMMFRB_PTR pCommFRB
                  unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.
nVars	Number of variables to be written.
pRemoteVar	Array with the address specifications **(M7VARADDR)**. It specifies the variables to be overwritten on the S7 object server or in the S7 CPU data area of the remote station.
pSrcVar	Array with the address specifications **(M7VARADDR)**. It specifies the variables to be sent on the S7 object server of the local station.
pCommFRB	Pointer to the function request block.
MPrio	Priority with which a message is sent (0–255).

Description

`M7PBKPut` starts asynchronous overwriting of *nVars* variables on the S7 object server or in the S7 CPU data area of the remote station with the values of local variables on the S7 object server.

*pRemoteVa*r and *pSrcVar* are pointers to arrays with *nVars* elements containing the address specifications of the remote or local variables on the S7 object server in the S7 CPU data area.

When the data have been stored on the remote computer, or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKPut` call and receipt of the M7MSG_PBK_DONE message.

The following conditions apply to the maximum user data length for the `M7PBKPut` call:

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

maxpdusize is the maximum PDU size for the connection opened with `M7KInitiate` and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7KInitiate, M7BKGet, M7BUBWrite, M7GetCommStatus

M7PBKResume

Function **Warm restart request for remote communication partner**

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKResume(M7CONNID *ConnID***);**

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.

Description `M7PBKResume` sends a RESTART request to the remote computer.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

Note A restart is not possible on the M7.

See Also **M7KInitiate, M7PBKStart, M7PBKStop, M7PBKStatus**

M7PBKStart

Function **Cold start request for remote communication partner**

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKStart(M7CONNID *ConnID***);**

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.

Description

The `M7PBKStart` function sends a cold RESTART request to the destination computer for all user programs.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KInitiate, M7PBKResume, M7PBKStop, M7PBKStatus**

M7PBKStatus

Function **Get status of remote communication partner**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKStatus (
                  M7CONNID ConnID,
                  M7PBKSTATUS_PTR pPBKStatus,
                  UDWORD nPBKStatus,
                  UDWORD *pnBytes);
```

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate`call.
pPBKStatus	Pointer to a structure of the type **M7PBKSTATUS** in which the logical and physical status of the remote device are stored (see Chapter 3).
nResultBufsiz	Length of the result buffer.
pnBytes	Pointer to the number of bytes read.

Description The `M7PBKStatus` function returns the current virtual device status.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KInitiate, M7PBKResume, M7PBKStop, M7PBKStart**

M7PBKStop

Function **Stop request for remote communication partner**

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKStop (M7CONNID *ConnID***);**

Parameters

Parameter Name	Meaning
ConnID	Connection reference from an `M7KInitiate` call.

Description

The `M7PBKStop` function sends a STOP request for all user programs on the destination computer.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also **M7KInitiate, M7PBKResume, M7PBKSatus, M7PBKStart**

M7PBKURcv

Function **Uncoordinated receive via configured connections**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKURcv(
                UDWORD flags,
                M7CONNID ConnID,
                UDWORD R_ID,
                UBYTE n_Vars,
                M7VARDATA_PTR pDstVar,
                M7COMMFRB_PTR pCommFRB,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
flags	Flags (A_ZERO_FLAG)
ConnID	Connection ID
R_ID	Block identifier for the remote USEND block or `M7PBKUSend` call.
n_Vars	Number of receive parameters
pDstVar	Receive parameters
pCommFRB	Pointer to the function request block
MPrio	Priority of the message sent (0–255)

Description

`M7PBKURcv` starts asynchronous receipt of multiple data areas via connection *ConnID* from a USEND block or `M7PBKUSend` call with the identifier *R_ID*.

The *nVars* parameter specifies the number of data areas to be received. *pSrcVar* points to an array of M7VARDATA objects. Each of these objects contain a data area for the received data.

When the data have been accepted by the local station or an error has occurred, an M7MSG_PBK_NDR message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKURcv` call and receipt of the M7MSG_PBK_NDR message.

If an error occurs in the asynchronous component, it can be read out from the referenced M7COMMFRB with the macro `M7GetCommStatus`.

The following condition applies to the maximum user data length for the `M7PBKURcv` call:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize \quad 24 \quad 4 * nVars$$

maxpdusize is the maximum PDU size for the connection opened with `M7KInitiate` and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7KInitiate, M7PBKUSend

M7PBKUSend

Function **Uncoordinated send via configured connections**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7PBKUSend(
                UDWORD flags,
                M7CONNID ConnID,
                UDWORD R_ID,
                UBYTE n_Vars,
                M7VARDATA_PTR pSrcVar,
                M7COMMFRB_PTR pCommFRB,
                unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
flags	Flags (A_ZERO_FLAG)
ConnID	Connection ID
R_ID	Block identifier for the remote URCV block or `M7PBKURcv` call.
n_Vars	Number of send parameters
pSrcVar	Send parameters
pCommFRB	Pointer to the function request block
MPrio	Priority of the message sent (0–255)

Description

`M7PBKUSend` starts asynchronous sending of multiple data areas via connection *ConnID* to the URCV block specified by *R_ID* or the `M7PBKURcv` call of the remote station.

The *nVars* parameter specifies the number of data areas to be transferred.

pSrcVar points to an array of M7VARDATA objects. Each of these objects contain a data area to be sent.

When the data have been accepted by the remote station or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKUSend` call and receipt of the M7MSG_PBK_DONE message.

If an error occurs in the asynchronous component, it can be read out from the referenced M7COMMFRB with the macro `M7GetCommStatus`.

The following condition applies to the maximum user data length for the `M7PBKUSend` call:

$$\sum_{i=1}^{nVars}(4 \quad nBytes(i)) \quad maxpdusize \quad 24 \quad 4 * nVars$$

maxpdusize is the maximum PDU size for the connection opened with `M7KInitiate` and *nBytes(i)* is the number of bytes for the i-th variable, rounded to the nearest even number.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7KInitiate, M7PBKURcv

M7PBKXAbort

Function

Close an application link (for communication on an MPI subnet via non-configured connections)

Syntax

#include <m7api.h>
M7ERR_CODE M7PBKXAbort(UWORD *DEST_ID*);

Parameters

Parameter Name	Meaning
DEST_ID	MPI node address (0-126).

Description

The `M7PBKXAbort` function closes an application link between client and server which was set up with the functions `M7PBKXSend`, `M7PBKXPut` or `M7PBKXGet`.

Return Value

= M7SUCCESS The function was successfully executed.
< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_CONN_ACTIVE	The connection to node DEST_ID is currently active and cannot be closed.
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_NOT_IMPLEMENTED	Function not supported

See Also

M7PBKXSend, M7PBKXPut, M7PBKXGet

M7PBKXCancel

Function

Cancel running receive request `M7PBKXRcv` **(for communication on an MPI subnet via non-configured connections)**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKXCancel(
                  M7COMMFRB_PTR  CommFRB);
```

Parameters

Parameter Name	Meaning
pCommFRB	Pointer to function request block

Description

`M7PBKCancel` cancels a running `M7PBKXRcv` request.

The FRB may not be used for any other purpose until receipt of the M7MSG_PBK_NDR message. If an error occurs in the asynchronous component, it can be read out from the referenced M7COMMFRB with the macro `M7GetCommStatus`.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_REMOTE	Execution error on server
M7E_NOT_IMPLEMENTED	Function not supported

See Also

M7PBKXRcv, M7GetCommStatus

M7PBKXGet

Function

Asynchronous variable reading (for communication on an MPI subnet via non-configured connections)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKXGet(
                  UDWORD flags,
                  UWORD DEST_ID,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pDstVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int Mprio);
```

Parameters

Parameter Name	Meaning	
flags	Flags	
	CONT	If CONT is set the application link set up by the data transfer is retained.
		If CONT is not set the application link set up by the data transfer is closed again after the data transfer
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
DEST_ID	MPI address (0-126)	
pRemoteVar	Pointer to **one** structure of type **M7VARADDR**. It specifies the variable to be read from the remote station	
pDstVar	Pointer to **one** structure of type **M7VARADDR**. It specifies the variable of the S7 object server for receiving data.	
pCommFRB	Pointer to the function request block	
MPrio	Priority of the message sent (0-255)	

Description

`M7PBKXGet` starts asynchronous reading of a variable from the variable area of the S7 object server or the S7 CPU data area on the remote station *DEST_ID* to the variable area of the S7 object server on the local station.

An application link with the remote station is set up if one does not already exist. If the `CONT` flag is enabled, the link remains intact after the end of data transfer. When the application link is no longer required, it must be closed with the `M7PBKXAbort` call. If the `CONT` flag is not enabled, the application link is closed again automatically after the end of data transfer.

pRemoteVar and *pDstVar* are pointers to elements which specify a contiguous area of items in the S7 object server or in the S7 CPU data area (see `M7BUBRead`).

If the data are stored in the data area specified by *pDstVa*r, an M7MSG_PBK_NDR message is generated for *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKXGet` call and receipt of the M7MSG_PBK_NDR message.

Note

The user data length amount to 76 byte.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_CONN_ACTIVE	The connection to station *DEST_ID* is currently active. No data can be transferred.
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_NO_SUCH_CONN	Invalid connection (*DEST_ID* incorrect)
M7E_KSUB_REMOTE	Execution error on server
M7E_LENGTH	Incorrect length
M7E_NOT_IMPLEMENTED	Function is not supported
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7BUBRead, M7GetCommStatus, M7PBKXAbort, M7PBKPut

M7PBKXPut

Function

Start asynchronous variable writing (for communication on an MPI subnet via non-configured connections)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKXPut(
                  UDWORD flags,
                  UWORD DEST_ID,
                  M7VARADDR_PTR pRemoteVar,
                  M7VARADDR_PTR pSrcVar,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int Mprio);
```

Parameters

Parameter Name	Meaning	
flags	Flags	
	CONT	If CONT is set the application link set up by the data transfer is retained.
		If CONT is not set the application link set up by the data transfer is closed again after the data transfer
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
DEST_ID	MPI address (0-126)	
pRemoteVar	Pointer to **one** structure of type **M7VARADDR**. It specifies the variable to be overwritten in the S7 object server or the S7 CPU data area of the remote station	
pSrcVar	Pointer to **one** structure of type **M7VARADDR**. It specifies the variable to be sent in the S7 object server of the local station	
pCommFRB	Pointer to the function request block	
MPrio	Priority of the message sent (0-255)	

Description

`M7PBKXPut` starts asynchronous overwriting of a variable in the S7 object server or S7 CPU data area of the remote station *DEST_ID* with the values of a local variable on the S7 object server.

An application link with the remote station is set up if one does not already exist. If the `CONT` flag is enabled, the link remains intact after the end of data transfer. When the application link is no longer required, it must be closed with the `M7PBKXAbort` call. If the `CONT` flag is not enabled, the application link is closed again automatically after the end of data transfer.

pRemoteVar and *pSrcVar* are pointers to the address specifications of the remote or local variable in the S7 object server/S7 CPU data area.

If the data are stored on the remote computer, or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKXPut` call and receipt of the M7MSG_PBK_DONE message.

Note

The user data length amount to 76 byte.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_KSUB_CONN_ACTIVE	The connection to station *DEST_ID* is currently active. No data can be transferred.
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_NO_SUCH_CONN	Invalid connection (*DEST_ID* incorrect)
M7E_KSUB_REMOTE	Execution error on server
M7E_LENGTH	Incorrect length
M7E_NOT_IMPLEMENTED	Function is not supported
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7BUBWrite, M7GetCommStatus, M7PBKXAbort, M7PBKPXGet

M7PBKXRcv

Function

Receive data (for communication on an MPI subnet via non-configured connections)

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7PBKXRcv(
                  UDWORD flags,
                  UDWORD R_ID,
                  M7VARADDR_PTR pDstVar,
                  UDWORD nLength,
                  M7COMMFRB_PTR pCommFRB,
                  unsigned int MPrio);
```

Parameters

Parameter Name	Meaning
flags	Flags A_USER: The A_USER Flag is used for controlling the parameters *pDstVar* (see below). A_ZERO_FLAG: This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
R_ID	Block identifier for the remote XSEND block or `M7PBKXSend`call.
pDstVar	Pointer to the receive buffer. A_USER not set Pointer to **one** structure of type **M7VARADDR**. It specifies a contiguous area of items of an S7 object into which the received data are copied. A_USER set Pointer to a buffer to which the received data are written.
nLength	Total length of the buffer in bytes
pCommFRB	Pointer to the function request block
MPrio	Priority of the message sent (0-255)

Description

`M7PBKXRcv` starts an asynchronous receive request for a buffer of *nLength* bytes from an XSEND block or M7PBKXSend call with identifier *R_ID*. Depending on the specified *flags*, the data are written to a buffer in the address area of the task (Flags=A_USER) or to the data area of the S7 object server (flags=A_ZERO_FLAG).

When the A_USER flag is not set, then the *nLength* parameter is not evaluated, but the buffer length is determined from one of the data structures pointed to by the parameter *pSrcVar* or *pDstVar* respectively. In this case

nLength can be assigned any value. Otherwise if the A_USER flag is set, you must assign *nLength* the buffer length.

When the data have been accepted by the local station or an error has occurred, an M7MSG_PBK_NDR message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the M7PBKXRcv call and receipt of the M7MSG_PBK_NDR message.

After receipt of an M7MSG_PBK_NDR message, the number of bytes received can be determined using the `M7GetCommRcvLen` call.

`M7PBKXRcv` calls can be canceled with `M7PBKXCancel`.

If an error occurs in the asynchronous component, it can be read out from the referenced M7COMMFRB with the macro `M7GetCommStatus`.

Note

The user data length amount to 76 byte.

Return Value

= M7SUCCESS The function was successfully executed.

< M7SUCCESS An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_REMOTE	Execution error on server
M7E_NO_MEM	No more memory available
M7E_PRIO	Incorrect priority
M7E_LENGTH	Incorrect length
M7E_NOT_IMPLEMENTED	Function is not supported
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7GetCommRcvLen, M7GetCommStatus, M7PBKXSend, M7PBKXCancel

M7PBKXSend

Function

Send data (for communication on an MPI subnet via non-configured connections)

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);
```

Parameters

Parameter Name	Meaning	
flags	Flags	
	CONT	If CONT is set the application link set up by the data transfer is retained. If CONT is not set the application link set up by the data transfer is closed again after the data transfer
	A_USER	The A_USER Flag is used for controlling the parameters *pSrcVar* (see below).
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
DEST_ID	MPI address (0-255)	
R_ID	Block identifier for the remote XRCV block or `M7PBKXRcv` call.	
pSrcVar	Pointer to the data to be sent. A_USER not set Pointer to **one** structure of type **M7VARADDR**. It specifies a contiguous area of items in a local S7 object. A_USER set Pointer to a buffer containing the data to be sent.	
nLength	Total length of the buffer in bytes	
pCommFRB	Pointer to the function request block	
MPrio	Priority of the message sent (0-255)	

Description

`M7PBKXSend` starts asynchronous sending of a data area of length *nLength* to the node *DEST_ID* to the XRCV block or `M7PBKXRcv` call, specified by *R_ID*, on the remote station.

An application link with the node is set up if one does not already exist. If the `CONT` flag is enabled, the link remains intact after the end of data transfer. When the application link is no longer required, it must be closed with the `M7PBKXAbort` call. If the `CONT` flag is not enabled, the application link is closed again automatically after the end of data transfer.

If the A_USER flag is enabled, the data to be sent begin at the address specified by *pSrcVar.*

If the A_USER flag is not enabled, *pSrcVar* specifies the address of the variable to be sent in the address area of the S7 object server.

When the A_USER flag is not set, then the *nLength* parameter is not evaluated, but the buffer length is determined from one of the data structures pointed to by the parameter *pSrcVar* or *pDstVar* respectively. In this case *nLength* can be assigned any value. Otherwise if the A_USER flag is set, you must assign *nLength* the buffer length.

When the data have been accepted by the remote station or an error has occurred, an M7MSG_PBK_DONE message is created with *pCommFRB*. The FRB may not be used for any other purpose in the time between the `M7PBKXSend` call and receipt of the M7MSG_PBK_DONE message.

If an error occurs in the asynchronous component, it can be read out from the referenced M7COMMFRB with the macro `M7GetCommStatus`.

Note

The user data length amount to 76 bytes.

Return Value

= M7SUCCESS	The function was successfully executed.
< M7SUCCESS	An error occurred.

Error Codes

Error Code	Meaning
M7E_KSUB_CONN_ACTIVE	The connection to station *DEST_ID* is currently active. No data can be transferred.
M7E_KSUB_NO_SRV	MPI driver not active
M7E_KSUB_NO_SUCH_CONN	Invalid connection (*DEST_ID* incorrect)
M7E_KSUB_REMOTE	Execution error on server
M7E_LENGTH	Wrong length
M7E_NO_MEM	No more memory available
M7E_NOT_IMPLEMENTED	Function is not supported
M7E_OBJ	Object type is not supported
M7E_OFFSET	Wrong offset
M7E_OVS_WRONG_STATE	Activity not permited in the actual working state
M7E_PAR	Error of parameter
M7E_PART	Subdomain not available

Error Code	Meaning
M7E_PER_BITS	Bit address is inadmissible in the peripheral area
M7E_PRIO	Incorrect priority
M7E_TYPE	Data type is invalid

See Also **M7GetCommStatus, M7PBKXAbort, M7PBKXRcv**

M7Read

Function **Read S7 data area**

Syntax

```
#include          <m7api.h>
M7ERR_CODE        M7Read(
                    VOID_PTR pBuffer,
                    UBYTE ObjType,
                    UWORD Part,
                    UBYTE DataType,
                    UWORD Count,
                    UDWORD Addr);
```

Parameters

Parameter Name	Meaning
pBuffer	Pointer to the destination buffer
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
DataType	Data type of an element (see Table 2-9). For the data type M7DT_BOOL is only available the value 1 for the parameter LENGTH.
Count	Number of elements to be read
Addr	Address or offset within an object or subarea. If *DataType* ≠ BOOL, *Addr* must be a multiple of 8 bits.

Description

The function reads a defined number of data elements from an S7 data area and copies them to a user data area.

The contents of the data area are <u>not</u> converted from *SIMATIC* to *Intel* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available

Error Code	Meaning
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also **M7ReadBit, M7ReadByte, M7ReadDWord, M7ReadWord**

M7ReadBit

Function

Read bit from S7 object

Syntax

```
#include <m7api.h>
BOOL        M7ReadBit(
              UBYTE ObjType,
              UWORD Part,
              UWORD ByteOffset,
              UBYTE BitOffset,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8).
ByteOffset	Offset of the byte where the desired bit is stored
BitOffset	Offset of the desired bit within the byte
pError	Pointer to a variable of the type M7ERR_CODE, in which an error code is to be stored.

Description

The function reads a bit from an S7 object. The bit is defined by the above parameters.

Return Value

If the function is successfully executed, the return value is the state of the addressed bit. If the state = ‘0’, the value is FALSE; if the state = ‘1’, the value is TRUE.

Error Codes

Error Code	Meaning
M7E_BIT_OFFSET	Incorrect bit offset within the byte
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid

See Also

M7Read, M7ReadByte, M7ReadDWord, M7ReadWord, M7ReadReal

M7ReadByte

Function **Read byte from S7 object**

Syntax

```
#include <m7api.h>
UBYTE          M7ReadByte(
                 UBYTE ObjType,
                 UWORD Part,
                 UWORD ByteOffset,
                 M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table).
ByteOffset	Offset of the desired byte
pError	Pointer to a variable of the type M7ERR_CODE, in which an error code is to be stored.

Description The function reads a byte from an S7 object. The byte is defined by the above parameters.

Return Value If the function is successfully executed, the return value is the value of the addressed byte.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported

See Also **M7Read, M7ReadBit, M7ReadDWord, M7ReadWord, M7ReadReal**

M7ReadDWord

Function

Read doubleword from S7 object

Syntax

```
#include <m7api.h>
UDWORD      M7ReadDWord(
              UBYTE ObjType,
              UWORD Part,
              UWORD ByteOffset,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the desired doubleword
pError	Pointer to a variable of the type ERR_CODE, in which an error code is to be stored.

Description

The function reads a doubleword from an S7 object. The doubleword is defined by the above parameters.

The contents of the doubleword are converted from the *SIMATIC* to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the value of the addressed doubleword in *Intel* format.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported

See Also

M7Read, M7ReadBit, M7ReadByte, M7ReadWord, M7ReadReal

M7ReadReal

Function **Read floating point number from S7 object**

Syntax

```
#include <m7api.h>
REAL        M7ReadReal(
              UBYTE ObjType,
              UWORD Part,
              UWORD ByteOffset,
              M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the desired floating point number
pError	Pointer to a variable of the type ERR_CODE, in which an error code is to be stored.

Description

The function reads a floating point number from an S7 object. The floating point number is defined by the above parameters.

The contents of the floating point number are converted from the *SIMATIC* to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the value of the addressed floating point number in *Intel* format.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported

See Also **M7Read, M7ReadBit, M7ReadByte, M7ReadDWord, M7WriteReal**

M7ReadWord

Function

Read word from S7 object

Syntax

```
#include <m7api.h>
UWORD        M7ReadWord(
               UBYTE ObjType,
               UWORD Part,
               UWORD ByteOffset,
               M7ERR_CODE_PTR pError);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the desired word
pError	Pointer to a variable of the type ERR_CODE, in which an error code is to be stored.

Description

The function reads a word from an S7 object. The word is defined by the above parameters.

The contents of the word are converted from the *SIMATIC* to the *Intel* numeric representation.

Return Value

If the function is successfully executed, the return value is the value of the addressed word in *Intel* format.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported

See Also

M7Read, M7ReadBit, M7ReadByte, M7ReadDWord, M7ReadReal

M7RelocateObject

Funktion **Pass S7 object to object server**

Syntax

```
#include <m7api.h>
M7ERRCODE   M7RelocateObject(
              UBYTE ObjType,
              UWORD Part,
              BOOL Copy);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
Copy	Handling of new memory area TRUE The user data of the object are copied to the new memory area. FALSE The user data of the object are not transferred.

Beschreibung This function `M7RelocateObject`can be used to pass an S7 object *ObjType*, which has previously been assigned to the responsibility of a user task with the function `M7LocateObject`, back to the object server.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NOT_LOCATED	Object was not passed to a user task with `M7LocateObject`
M7E_NO_MEM	No more memory available
M7E_OBJ	Object type not supported.
M7E_PART	Subarea not available.

See Also **M7LocateObject**

M7RemoveObject

Function

Delete S7 object from BACKDIR or ROMDIR

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7RemoveObject(
                UBYTE ObjType,
                UWORD Part,
                BOOL Rom);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8).
Rom	*Rom* = FALSE: S7 object is deleted from BACKDIR. *Rom* = TRUE: S7 object is deleted from ROMDIR.

Description

The function deletes an S7 object from the BACKDIR or ROMDIR directory, depending on the *Rom* parameter.

Return Value

If the function is successfully executed, it returns the value of the addressed word in *Intel* format.

The function passes error flags in **pError:*

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred (see Error Codes).

Error Codes

Error Code	Meaning
M7E_PART	Subarea not available.
M7E_NODIR	Directory not readable or does not exist.
M7E_OBJ	Object type not supported.
M7E_REM_OBJ	Illegal action because the object is retentive

See Also

M7CreateObject, M7DeleteObject, M7GetObjectInfo

M7RequestState

Function

Request operating state change

Syntax

```
#include <m7api.h>
void        M7RequestState(
              M7TSFRB_PTR pTSFRB,
              UWORD State,
              UWORD Reason,
              uint MPrio);
```

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB provided for handling the request.
State	Specifies the new operating state requested. The following values can be specified: M7STATE_HALT — HALT operating state M7STATE_RESET — RESET operating state M7STATE_RUN — RUN operating state M7STATE_STOP — STOP operating state M7STATE_CONTINUE — CONTINUE from HALT operating state in the former state (STARTUP or RUN).
Reason	For user diagnostics entries; from 0xA000 to 0xBFFF.
MPrio	Priority of the message dispatched (0–255).

Description

The function requests a change to the operating state specified in the *State* parameter.

When the operating state specified in the *State* parameter is activated, or an error has occurred, the calling task is informed by a message of the type M7MSG_REQ_FINISHED.

When the M7MSG_REQ_FINISHED message is received, you can use the C macro `M7GetFRBErrCode` to detect whether the function has been successful.

In this case, `M7GetFRBErrCode` returns the following error codes:

Error Codes

Error Code	Meaning
M7E_OST_CPU_IN_STOP	CPU in STOP mode (for FM)
M7E_OST_ILLEGAL_PARAM_CPU	Parameter error
M7E_OST_MODE_SW_IN_STOP	Operating mode selector on CPU/FM is set to STOP

Error Code	Meaning
M7E_OST_WRONG_STATE	Transition from current state not possible or requested state already active.
M7E_OST_NO_SUCH_STATE	Unknown operating state
M7E_PAR	Parameter error
M7E_PRIO	Incorrect priority

Return Value

= M7SUCCESS Always returned

You should check whether the requested operating state has been activated or denied, or whether an error has occurred, **after the M7MSG_REQ_FINISHED message has been received**, with the functions `M7GetFRBErrCode` or `M7GetTSType`.

See Also

M7GetState, M7LinkState, M7GetFRBErrCode, M7GetTSType

M7RetriggerCycle

Function **Retrigger cycle time**

Syntax

```
#include <m7api.h>
M7ERR_CODE M7RetriggerCycle(void)
```

Description The function resets the cycle time, with the result that monitoring of the maximum cycle time recommences.

Return Value = M7SUCCESS Always returned

See Also **M7LinkCycle, M7UnLinkCycle**

M7SendDiagAlarm

Function **Send diagnostics alarm to S7 CPU**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7SendDiagAlarm(VOID_PTR pAlarmInfo);
```

Parameters

Parameter Name	Meaning
pAlarmInfo	Pointer to a memory area containing the supplementary alarm information. The supplementary information is 16 bytes in length and is transferred to diagnostics record 1.

Description The function sends a diagnostics alarm to the S7/M7 CPU.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_GL_ALARM_DISABLED	All alarms are disabled (activated by S7/M7 CPU).
M7E_ODIS	Output disable (activated by S7/M7 CPU).
M7E_D_ALARM_BUSY	Diagnostics alarm has not yet been acknowledged by S7/M7 CPU.
M7E_ALARM_GEN_DISABLED	Alarm generation disabled on module in record 0.
M7E_D_ALARM_GEN_DISABLED	Diagnostics alarm generation disabled on module in record 0.

See Also **M7GetDiagAlarmBusy**

M7SendIOAlarm

Function **Send process alarm to S7 CPU**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7SendIOAlarm(UDWORD AlarmInfo);
```

Parameters

Parameter Name	Meaning
AlarmInfo	4 bytes of supplementary alarm information

Description The function sends a process alarm to the S7/M7 CPU.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_GL_ALARM_DISABLED	All alarms are disabled (activated by S7/M7 CPU).
M7E_ODIS	Output disable (activated by S7/M7 CPU).
M7E_P_ALARM_BUSY	Process alarm has not yet been acknowledged by S7/M7 CPU.
M7E_ALARM_GEN_DISABLED	Alarm generation disabled on module in record 0.
M7E_P_ALARM_GEN_DISABLED	Process alarm generation disabled on module in record 0.

See Also **M7GetIOAlarmBusy**

M7SetFRBTag

Function

Set identifier of an FRB

Syntax

#include <m7api.h>

void M7SetFRBTag(
M7FRBHEADER_PTR *pFRB*,
UWORD *Tag*);

Parameters

Parameter Name	Meaning
pFRB	Pointer to FRB whose identifier is to be set.
Tag	Identifier of the FRB

Description

The function sets the identifier of the FRB to the value specified in the *Tag* parameter.

The value is user-specific and can be allocated freely within the value range permitted for UWORD.

The FRB identifier can be read out again with the `M7GetFRBTag` function.

The call is implemented as a C macro.

See Also

M7GetFRBErrCode, M7GetFRBTag

M7SetTime

Function **Set date and time**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7SetTime(
                M7TIME_DATE_PTR pDateTime);
```

Parameters

Parameter Name	Meaning
pDateTime	Pointer to the memory area containing the date/time structure in which the current values for the date and time are stored (see Chapter 3).

Description The function sets the internal system time and date.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error

See Also **M7GetTime**

M7SetUserLED

Function

Control user (USR) LEDs

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7SetUserLED(
                UWORD Led,
                UWORD Mode);
```

Parameters

Parameter Name	Meaning
Led	Number of user LED: M7USERLED1 M7-300 and M7-400 M7USERLED2 M7-400 only
Mode	Control mode: M7LED_OFF Switch off LED M7LED_ON Switch on LED, steady light M7LED_FLASHSLOW Switch on LED, flashing light, 0.5 Hz M7LED_FLASHFAST Switch on LED, flashing light, 2 Hz

Description

The function switches the user LED on, off or flashing (0.5 or 2 Hz), according to the value of *Mode*.

You specify the number of the “user” LED with the *Led* parameter. M7USERLED1 and M7USERLED2 can be specified for *Led* on the M7-400; only M7USERLED1 is allowed on the M7-300.

The selected LED can be switched on or off with the constants M7LED_ON and M7LED_OFF. The flashing frequency can also be controlled in the *Mode* parameter by performing a logic operation with M7LED_FLASHSLOW or M7LED_FLASHFAST.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error

M7StoreBit

Function **Set bit state in process image**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreBit(
                UWORD PIType,
                UWORD ByteOffset,
                UBYTE BitOffset,
                BOOL  Value);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal byte
BitOffset	Bit offset within the signal byte
Value	State to which the addressed bit is to be set (TRUE or FALSE)

Description The function addresses a bit in the process image defined by *PIType*, and sets it to the state specified in *Value*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType*, *ByteOffset* or *BitOffset*

See Also **M7StoreByte, M7StoreDWord, M7StoreWord**

M7StoreByte

Function **Overwrite byte in process image**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreByte(
                UWORD PIType,
                UWORD ByteOffset,
                UBYTE Value);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal byte
Value	New value with which the byte in the process image is to be overwritten.

Description The function addresses a byte in the process image defined by *PIType*, and overwrites it with the value specified in *Value*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Incorrect *PIType* or *ByteOffset*

See Also **M7StoreBit, M7StoreDWord, M7StoreWord**

M7StoreDirect

Function **Write data direct to I/O area**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirect(
                  VOID_PTR pBuffer,
                  UWORD SizeOfItem,
                  UWORD Count,
                  M7IO_LOGADDR Addr);
```

Parameters

Parameter Name	Meaning
pBuffer	Pointer to the source buffer
SizeOfItem	Size of an element in bytes. The following constants are predefined: M7PBYTE Element has data type BYTE M7PWORD Element has data type WORD M7PDWORD Element has data type DWORD
Count	Number of elements
Addr	Logical address of first element

Description

The function transfers data directly to the process I/O from a data buffer referenced by *pBuffer*. The size, number and destination of the transferred data are defined by the call parameters.

The function does not convert the numeric representation (*SIMATIC/Intel*).

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	Device not ready for data communication

See Also **M7StoreDirectByte, M7StoreDirectDWord, M7StoreDirectWord**

M7StoreDirectByte

Function

Write byte direct to I/O

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirectByte(
                  M7IO_LOGADDR Addr,
                  UBYTE Value);
```

Parameters

Parameter Name	Meaning
Addr	Logical address of the I/O byte
Value	New value with which the I/O byte is to be overwritten.

Description

The function addresses a byte on the process I/O and overwrites it with the value specified by *Value*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	Device not ready for data communication

See Also

M7StoreDirect, M7StoreDirectDWord, M7StoreDirectWord

M7StoreDirectDWord

Function **Write doubleword direct to I/O**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDirectDWord(
                  M7IO_LOGADDR Addr,
                  UDWORD Value);
```

Parameters

Parameter Name	Meaning
Addr	Logical address of the I/O doubleword
Value	New value with which the I/O doubleword is to be overwritten, in ***SIMATIC*** format.

Description

The function addresses a doubleword on the process I/O and overwrites it with the value specified by *Value*.

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	Device not ready for data communication

See Also **M7StoreDirect, M7StoreDirectByte, M7StoreDirectWord**

M7StoreDirectWord

Function **Write word direct to I/O**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreDirectWord(
                M7IO_LOGADDR Addr,
                UWORD Value);
```

Parameters

Parameter Name	Meaning
Addr	Logical address of the I/O word
Value	New value with which the I/O word is to be overwritten, in ***SIMATIC*** format.

Description

The function addresses a word on the process I/O and overwrites it with the value specified by *Value*.

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout
M7E_DP_SLAVE_STATE	Device not ready for data communication

See Also **M7StoreDirect, M7StoreDirectByte, M7StoreDirectDWord**

M7StoreDWord

Function

Write doubleword to process image

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreDWord(
                  UWORD PIType,
                  UWORD ByteOffset,
                  UDWORD Value);
```

Parameters

Parameters Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal doubleword
Value	New value with which the doubleword in the process image is to be overwritten, in ***SIMATIC*** format.

Description

The function addresses a doubleword in the process image defined by *PIType*, and overwrites it with the value specified in *Value*.

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error

See Also

M7StoreBit, M7StoreByte, M7StoreWord

M7StoreISAByte

Function **Write byte direct to ISA bus I/O**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreISAByte(
                M7IO_DESC_PTR pIODesc,
                UBYTE Value);
```

Parameters

Parameter Name	Meaning
pIODesc	Pointer to I/O descriptor initialized with `M7InitISADesc`.
Value	Value to be written

Description

The function runs as a macro, performing a direct access to the ISA bus process I/O, using an I/O descriptor generated with `M7InitISADesc`.The value to be written is defined by *val*. The address of the I/O area is defined by the I/O descriptor for the output signals. The process image of outputs is updated automatically.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Data access to ISA bus is larger (in bytes) than specified in `M7InitISADesc`.

See Also **M7StoreISAWord, M7StoreISADWord, M7InitISADesc**

M7StoreISADWord

Function

Write doubleword direct to ISA bus I/O

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreISADWord(
                M7IO_DESC_PTR pIODesc,
                UDWORD val);
```

Parameters

Parameter Name	Meaning
pIoDesc	Pointer to I/O descriptor initialized with `M7InitISADesc`
val	Value to be written

Description

The function runs as a macro, performing a direct access to the ISA bus process I/O, using an I/O descriptor generated with `M7InitISADesc`. The value to be written is defined by *val*. The address of the I/O area is defined by the I/O descriptor for the output signals. The process image of outputs is updated automatically.

The function converts the value from Intel to SIMATIC format before performing the access.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Data access to ISA bus is larger (in bytes) than specified in `M7InitISADesc`

See Also

M7StoreISAByte, M7StoreISAWord, M7InitISADesc

M7StoreISAWord

Function **Write word direct to ISA bus I/O**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreISAWord(
                M7IO_DESC_PTR pIODesc,
                UWORD val);
```

Parameters

Parameter Name	Meaning
pIoDesc	Pointer to I/O descriptor initialized with `M7InitISADesc`
val	Value to be written

Description

The function runs as a macro, performing a direct access to the ISA bus process I/O, using an I/O descriptor generated with `M7InitISADesc` . The value to be written is defined by *val*. The address of the I/O area is defined by the I/O descriptor for the output signals. The process image of outputs is updated automatically.

The function converts the value from Intel to SIMATIC format before performing the access.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Data access to ISA bus is larger (in bytes) than specified in `M7InitISADesc`.

See Also **M7StoreISAByte, M7StoreISADWord, M7InitISADesc**

M7StoreObject

Function **Store S7 object in BACKDIR or ROMDIR**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7StoreObject(
                UBYTE ObjType,
                UWORD Part,
                BOOL Rom);
```

Parameters

Parameter Name	Meaning	
ObjType	Type identifier for the S7 object:	
	M7D_DB	Data block
	M7D_PAR_READ	Parameter data record with read attribute
	M7D_PAR_WRITE	Parameter data record with write attribute
Part	Subarea (DB number of the parameter data record)	
Rom	Rom = TRUE:	S7 object is stored in ROMDIR.
	Rom = FALSE:	S7 object is stored in BACKDIR.

Description

The function stores an S7 object in the directory defined by the environment variable BACKDIR or ROMDIR. The *Rom* call parameter defines the memory area in which the S7 object is to be stored.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PART	Subarea not available.
M7E_NODIR	Directory not readable or does not exist.
M7E_OBJ	Object type not supported.

See Also **M7CreateObject, M7DeleteObject, M7RemoveObject, M7LocateObject**

M7StorePIQ

Function

Update output signals

Syntax

#include <m7api.h>
M7ERR_CODE M7StorePIQ(UWORD *PIQNo***);**

Parameters

Parameter Name	Meaning
PIINo	Number of process images part on M7-400. M7-400: 0 Complete process image 1 ... 8 Process image part M7-300: 0 Complete process image Process image parts are not supported

Description

The function updates the output signals with the contents of the complete process image or the specified part of the process image of outputs.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	LB timeout

See Also

M7LoadPII, M7ClearPI

M7StoreRecord

Function **Transfer data record to a signal module**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreRecord(
                  UBYTE RecordNum,
                  VOID_PTR pBuffer,
                  UBYTE Size,
                  UBYTE PType,
                  M7IO_BASEADDR Addr);
```

Parameters

Parameter Name	Meaning
RecordNum	Record number Range: 0 to 255
pBuffer	Pointer to a buffer in the working memory containing the **contents** of the data record referenced by *RecordNum*.
Size	Length of the data record
PType	Identifier for the I/O module: M7IO_IN Input module M7IO_OUT Output module If the module is a mixed module, specify the area ID of the lowest address. If the addresses are the same, specify M7IO_IN.
Addr	I/O base address of signal module

Description

The function transfers a data record from the data buffer referenced by the *pBuffer* parameter to an I/O module.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BSY	Local bus is busy
M7E_CMD	Local bus command error
M7E_COM_ERROR	Error on transfer protocol handling
M7E_HWFAULT	General hardware error
M7E_PAR	Parameter error
M7E_PARITY	Local bus parity error
M7E_QVZ	Local bus timeout

Error Code	Meaning
M7E_REC_LENGTH	Module reporting incorrect record length
M7E_REC_NUMBER	Module reporting incorrect record number
M7E_DPX2_FAULT	Error on DP job for record transfer
M7E_DP_SLAVE_STATE	DP Slave not in DATA state
M7E_INVAL_DEV	Module of a DP slave is not available

See Also **M7LoadRecord**

M7StoreWord

Function **Overwrite word in process image**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7StoreWord(
                  UWORD PIType,
                  UWORD ByteOffset,
                  UWORD Value);
```

Parameters

Parameter Name	Meaning
PIType	Identifiers for process images: M7IO_PII Process image of inputs M7IO_PIQ Process image of outputs
ByteOffset	Offset of signal word
Value	New value with which the word in the process image is to be overwritten.

Description

The function addresses a word in the process image defined by *PIType*, and overwrites it with the value specified in *Value*.

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Error in *PIType* or *ByteOffset*.

See Also **M7StoreBit, M7StoreByte, M7StoreDWord**

M7SZLRead

Function **Read system state list**

Syntax

```
#include <m7api.h>
M7ERR_CODE M7SZLRead (
                    UDWORD flags,
                    M7CONNID ConnID,
                    UBYTE_PTR pBuffer,
                    UDWORD nBufsiz,
                    UWORD szlID,
                    UWORD Index,
                    UDWORD *pnBytes);
```

Parameters

Parameter Name	Meaning	
flags	Flags	
	A_FILE	If it is enabled, *pBuffer* specifies the name of the file in which the system state list item is stored; otherwise the item is stored in memory.
	A_ZERO_FLAG	This flag can be connected with other options by an OR operation. It must be set if no other flag is used.
ConnID	Connection reference from an `M7KInitiate` call.	
pBuffer	Receive buffer. If A_FILE is enabled, *pBuffer* specifies the name of the file in which the item is stored; otherwise the item is stored in memory.	
nBufsiz	Length of the receive buffer. Ignored if A_FILE is enabled.	
SZLID	ID of the SZL sublist to be read.	
Index	Index in the sublist.	
pnBytes	Pointer to the number of bytes read.	

Description

The `M7SZLRead` function reads out the part of the system state list specified by *szlID* and *Index* from the destination computer. The user should specify a buffer sufficiently large to store the system state list data. If a buffer overflow occurs, the function returns an appropriate error code.

The structure of the system state list for an M7 is described in the User Manual, System Software for S7-300 and S7-400, Installation and Operation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_NO_MEM	No more memory available
M7E_KSUB_PARAM	Parameter error
M7E_KSUB_NO_SUCH_CONN	Invalid connection
M7E_KSUB_CONN_CLOSED	Connection closed
M7E_KSUB_FILEIO	Error on file handling
M7E_KSUB_REMOTE	Execution error on server
M7E_KSUB_SDB_WAS_DELETED	Connection deleted by STEP7, connection is no longer active

See Also

M7KInitiate, M7WriteDiagnose

M7UnLinkBatteryFailure

Function

Unlink FRB for battery alarm

Syntax

#include <m7api.h>

M7ERR_CODE M7UnLinkBatteryFailure(
M7BAFFRB_PTR *pBAFFRB***);**

Parameters

Parameter Name	Meaning
pBAFFRB	Pointer to the FRB to be unlinked.

Description

The function unlinks the FRB on the OST server.

The FRB must previously have been linked with `M7LinkBatteryFailure`

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FSCFRB not operational

See Also

M7LinkBatteryFailure

M7UnLinkCycle

Function **Unlink FRB on FC server**

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkCycle(M7FSCFRB_PTR *pFSCFRB***);**

Parameters

Parameter Name	Meaning
pFSCFRB	Pointer to the FRB to be unlinked.

Description

The function unlinks the FRB on the FC server.

The FRB must previously have been linked with `M7LinkCycle`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FSC_NO_SUCH_CYCLE	Unknown state
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also **M7LinkCycle, M7ConfirmCycle**

M7UnLinkDataAccess

Function **Unlink S7 object for access information via message**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7UnLinkDataAccess(M7OBJFRB_PTR
                pOBJFRB);
```

Parameters

Parameter Name	Meaning
pOBJFRB	Pointer to the FRB to be unlinked.

Description

The function unlinks the access information for an S7 object on the S7 object server.

The FRB must previously have been linked with `M7LinkDataAccess`.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also **M7LinkDataAccess, M7GetFlags, M7GetObjType, M7GetPart**

M7UnLinkDataAccessCB

Function

Unlink callback function call for S7 object access

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDataAccessCB(M7CBFRB_PTR *pCBFRB***);**

Parameters

Parameter Name	Meaning
pCBFRB	Pointer to the FRB provided for unlinking.

Description

The function unlinks a callback function on the object server.

The callback function must previously have been linked with the `M7Link-DataAccessCB` function.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also

M7LinkDataAccessCB, M7GetCBFlags, M7GetCBBuffer, M7GetCBDataType, M7GetCBObjType, M7GetCBPart, M7GetCBCount, M7GetCBByteOffset, M7GetCBBitOffset

M7UnLinkDate

Function **Unlink time-controlled time message**

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDate(M7TFRB_PTR *pTFRB***);**

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the FRB linked with the time-controlled time message.

Description

This function is used to unlink the request for a time-controlled time message on the server.

The FRB must previously have been linked with the `M7LinkDate` function.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also **M7LinkDate**

M7UnLinkDiagAlarm

Function

Unlink diagnostics alarm

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkDiagAlarm(
M7DIAGALARM FRB_PTR *pDAFrb***);**

Parameters

Parameter Name	Meaning
pDAFrb	Pointer to the FRB to be unlinked.

Description

The function unlinks the specified FRB for alarm handling on the alarm server. No more diagnostics alarms are subsequently signalled for the calling task.

The FRB must previously have been linked with `M7LinkDiagAlarm`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked.
M7E_ALARM_PENDING	A diagnostics alarm is still waiting on the module involved and must be acknowledged first.

See Also

M7LinkDiagAlarm, M7GetDiagAlarmAddr, M7GetDiagAlarmBusy, M7GetDiagAlarmInfo,M7GetDiagAlarmPT ype, M7ConfirmDiagAlarm

M7UnLinkIOAlarm

Function

Unlink process alarm

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkIOAlarm(
M7IOALARM_FRB_PTR *pPAFrb***);**

Parameters

Parameter Name	Meaning
pPAFrb	Pointer to the FRB to be unlinked.

Description

The function unlinks the specified FRB for alarm handling on the alarm server. No more process alarms are subsequently signalled for the calling task.

The FRB must previously have been linked with `M7LinkIOAlarm`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked.
M7E_ALARM_PENDING	A diagnostics alarm is still waiting on the module involved and must be acknowledged first.

See Also

M7LinkIOAlarm, M7GetIOAlarmAddr, M7GetIOAlarmMask, M7GetIOAlarmState, M7GetIOAlarmPTye, M7ConfirmIOAlarm

M7UnLinkOneShotTimer

Function **Unlink one-shot time message**

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkOneShotTimer(M7TFRB_PTR *pTFRB***);**

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the FRB with which the one-shot time message was linked.

Description

The function unlinks the request for a one-shot time message on the time server.

The FRB must previously have been linked with `M7LinkOneShotTimer`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also **M7LinkOneShotTimer**

M7UnLinkPeriodicTimer

Function **Unlink periodic time message**

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkPeriodicTimer(M7TFRB_PTR *pTFRB***);**

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the FRB with which the periodic time message was linked.

Description

The function unlinks the request for a periodic message on the time server.

The FRB must previously have been linked with `M7LinkPeriodicTimer`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked.

See Also **M7LinkPeriodicTimer, M7ConfirmPeriodicTimer, M7GetLostPeriods**

M7UnLinkPIError

Function **Unlink FRB for process image transfer error**

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkPeriodicTimer(M7TFRB_PTR *pTFRB***);**

Parameters

Parameter Name	Meaning
pTFRB	Pointer to the FRB to be unlinked

Description

The `M7UnLinkPIError` function unlinks the FRB for the handling of process image transfer errors in the free cycle. This FRB must already have been linked with the `M7LinkPIError` function.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked.

See Also **M7LinkPeriodicTimer, M7ConfirmPeriodicTimer, M7GetLostPeriods**

M7UnLinkState

Function **Unlink message about specific operating state**

Syntax

```
#include <m7api.h>
M7ERR_CODE M7UnLinkState(M7TSFRB_PTR pTSFRB);
```

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB to be acknowledged.

Description

The function unlinks messages relating to a specific operating state on the OST server.

The FRB must previously have been linked with `M7LinkState`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also **M7LinkState, M7GetState, M7RequestState**

M7UnLinkTransition

Function

Unlink message about specific operating state transition

Syntax

#include <m7api.h>
M7ERR_CODE M7UnLinkTransition(M7TSFRB_PTR *pTSFRB***);**

Parameters

Parameter Name	Meaning
pTSFRB	Pointer to the FRB to be acknowledged.

Description

The function unlinks messages relating to a specific operating state transition on the OST server.

The FRB must previously have been linked with `M7LinkTransition`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_PAR	Parameter error
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also

M7LinkTransition, M7GetTSReason, M7GetTSType, M7ConfirmTransition

M7UnLinkZSAlarm

Function **Unlink message about insert/remove module alarm**

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7UnLinkZSAlarm(
                  M7ZSALARM_FRB_PTR pZSFRB);
```

Parameters

Parameter Name	Meaning
pZSFRB	Pointer to the FRB to be acknowledged.

Description The function unlinks messages for an insert/remove module alarm event.

The FRB must previously have been linked with `M7LinkZSAlarm`.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_FRB_NOT_IN_LIST	FRB not linked

See Also **M7ConfirmZSAlarm, M7LinkZSAlarm, M7GetZSAlarmIMRBaddr, M7GetZSAlarmMode, M7GetZSAlarmPType, M7GetZSAlarmAddr**

M7Write

Function **Write user data to S7 data area**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7Write(VOID_PTR pBuffer,
                      UBYTE ObjType,
                      UWORD Part,
                      UBYTE DataType,
                      UWORD Count,
                      UDWORD Addr);
```

Parameters

Parameter Name	Meaning
pBuffer	Pointer to the buffer containing the user data. The user data must be in the *SIMATIC* format!
ObjType	Type identifier for the desired S7 object (see Table).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
DataType	Data type of an element (see Table 2-9). For the data type M7DT_BOOL is only available the value 1 for the parameter LENGTH.
Count	Number of elements to be copied
Addr	Address or offset within an object or subarea. If *DataType* ≠ BOOL, *Addr* must be a multiple of 8 bits.

Description

The function copies a defined number of data elements from a user data area to an S7 data area.

The contents of the data area are <u>not</u> converted from *Intel* to *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available

Error Code	Meaning
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid
M7E_WRITE_PROTECT	Object type under write protection

See Also **M7WriteBit, M7WriteByte, M7WriteDWord, M7WriteWord**

M7WriteBit

Function **Set bit in S7 object**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7WriteBit(
                UBYTE ObjType,
                UWORD Part,
                UWORD ByteOffset,
                UBYTE BitOffset,
                BOOL Value);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the byte where the desired bit is stored
BitOffset	Offset of the desired bit within the byte
Value	Value to which the addressed bit is to be set

Description The function addresses a bit, defined by the above parameters in an S7 object, and sets it to the state specified by *Value*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_BIT_OFFSET	Incorrect bit offset within the byte
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OFFSET	Incorrect offset
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_PER_BITS	Bit addressing not permitted in I/O area
M7E_TYPE	Data type is invalid
M7E_WRITE_PROTECT	Object type under write protection

See Also **M7Write, M7WriteByte, M7WriteDWord, M7WriteReal, M7WriteWord**

M7WriteByte

Function **Overwrite byte in S7 object**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7WriteByte(
                UBYTE ObjType,
                UWORD Part,
                UWORD ByteOffset,
                UBYTE Value);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8).
ByteOffset	Offset of the desired byte
Value	Value with which the addressed byte is to be overwritten.

Description

The function addresses a byte, defined by the above parameters in an S7 object, and overwrites it with the value specified by *Value*.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported
M7E_WRITE_PROTECT	Object type under write protection

See Also **M7Write, M7WriteBit, M7WriteDWord, M7WriteReal, M7WriteWord**

M7WriteDiagnose

Function **Write entry to diagnostics buffer**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7WriteDiagnose
                UBYTE Type,
                UBYTE Eventnumber,
                BOOL Direction,
                UWORD ZI1,
                UDWORD ZI23,
                BOOL Send);
```

Parameters

Parameter Name	Meaning
Type	Event class
Eventnumber	Event number
Direction	If TRUE, 1 is transferred (incoming event)
ZI1	Supplementary info 1
ZI23	Supplementary info 2 and 3
Send	If TRUE, event is sent via K bus

Description

The call stores a diagnostics event with the specified class/number and supplementary information. The entry contains the current time stamp. If the *Send* parameter is specified, the diagnostics event is sent on to linked communication partners.

Entries cannot be written to the diagnostics buffer in the STOP operating state. This prevents existing entries from being overwritten.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_DIAG_NUMBER	Incorrect event class (only 0x0a or 0x0b allowed)
M7E_DIAG_STATE	Incorrect operating state. Entries not possible in STOP state.
M7E_WRITE_PROTECT	Object type under write protection

See Also **M7SZLRead**

M7WriteDWord

Function

Overwrite doubleword in S7 object

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7WriteDWord(
                UBYTE ObjType,
                UWORD Part,
                UWORD ByteOffset,
                UDWORD Value);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the desired doubleword
Value	Value with which the addressed doubleword is to be overwritten, in *Intel* format.

Description

The function addresses a doubleword in an S7 object, defined by the above parameters in an S7 object, and overwrites it with the value specified by *Value.*

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported
M7E_WRITE_PROTECT	Object type under write protection

See Also

M7Write, M7WriteBit, M7WriteByte, M7WriteReal, M7WriteWord

M7WriteReal

Function **Overwrite a floating point number in S7 object**

Syntax

```
#include <m7api.h>
M7ERR_CODE    M7WriteReal(
                UBYTE ObjType,
                UWORD Part,
                UWORD ByteOffset,
                REAL Value);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the desired floating point number
Value	Value with which the addressed word is to be overwritten, in *Intel* format.

Description

The function addresses a floating point number in an S7 object, defined by the above parameters in an S7 object, and overwrites it with the value specified by *Value*.

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.

< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported
M7E_WRITE_PROTECT	Object type under write protection

See Also **M7Write, M7WriteBit, M7WriteByte, M7WriteDWord, M7WriteWord**

M7WriteWord

Function

Overwrite word in S7 object

Syntax

```
#include <m7api.h>
M7ERR_CODE      M7WriteWord(
                  UBYTE ObjType,
                  UWORD Part,
                  UWORD ByteOffset,
                  UWORD Value);
```

Parameters

Parameter Name	Meaning
ObjType	Type identifier for the desired S7 object (see Table 2-7).
Part	Subarea (DB number, etc.) The permissible values for the subarea depend on the type of S7 object (see Table 2-8)
ByteOffset	Offset of the desired word
Value	Value with which the addressed word is to be overwritten, in *Intel* format.

Description

The function addresses a word in an S7 object, defined by the above parameters in an S7 object, and overwrites it with the value specified by *Value*.

Before the value specified by *Value* is stored, the function performs a conversion from the *Intel* to the *SIMATIC* numeric representation.

Return Value

= M7SUCCESS: The function was successfully executed.
< M7SUCCESS: An error occurred.

Error Codes

Error Code	Meaning
M7E_LENGTH	Incorrect length
M7E_OBJ	Object type not supported
M7E_OVS_WRONG_STATE	Illegal action in current operating mode
M7E_PAR	Parameter error
M7E_PART	Subarea not available
M7E_TYPE	Data type not supported
M7E_WRITE_PROTECT	Object type under write protection

See Also

M7Write, M7WriteBit, M7WriteByte, M7WriteDWord, M7WriteReal

RMOS API

6

In this chapter

get2ndparm

Function **Read EBX start parameter of task**

Syntax

```
#include <rmapi.h>
unsigned int    get2ndparm (void);
```

Description

`get2ndparm` returns the EBX of the task, overwriting the EAX register. The functions `getdword` and `getparm` can subsequently no longer be used.

This function call must be the first within a task, since the code generated by the compiler can, under certain circumstances, overwrite the EAX or EBX register.

See Also **getdword, getparm**

getdword

Function **Read start parameter of task in long format**

Syntax

```
#include <rmapi.h>
unsigned long    getdword (void);
```

Description `getdword` returns an unsigned long variable corresponding to the EAX register.

This function call must be the first within a task, since the code generated by the compiler can, under certain circumstances, overwrite the EAX or EBX register.

See Also **get2ndparm, getparm**

getparm

Function **Read start parameter of task as pointer**

Syntax

```
#include <rmapi.h>
int *          getparm (void);
```

Description `getparm` returns a pointer corresponding to the EAX register.

This function call must be the first within a task, since the code generated by the compiler can, under certain circumstances, overwrite the EAX or EBX register.

See Also **get2ndparm, getdword**

RmActivateTask

Function **Activate Task**

Syntax

#include <rmapi.h>
int **RmActivateTask(uint** *TaskID***);**

Parameters

Parameter Name	Meaning
TaskID	Task-ID (RM_OWN_TASK=own task)

Description

This function switches another task to the READY state if it was in the BLOCKED state.

The `RmActivateTask` is illegal under the following conditions, and is terminated with an error message:

- Termination/deletion through `RmKillTask` was already requested
- Page fault because stack overflow

Return Value

RM_OK Function successfully executed

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *TaskID* was passed.
RM_INVALID_TASK_STATE	Call illegal in current task state (task is in DORMANT, ACTIVE, READY or BLOCKED for end of I/O state).

See Also

RmDeleteTask, RmEndTask, RmKillTask, RmPauseTask

RmAlloc

Function

Allocate memory from HEAP

Syntax

#include <rmapi.h>

int RmAlloc (
ulong *TimeOutValue***,**
uint *Mode***,**
ulong *Size***,**
void ****ppMemory)**

Parameters

Parameter Name	Meaning
TimeOutValue	Maximum wait time before execution RM_CONTINUE — Continue task without waiting for memory allocation. RM_WAIT — Wait for memory allocation. 0 ... RM_MAXTIME — Time interval in ms. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) — Wait for (*hour*) hours RM_MINUTE(*min*) — Wait for (*min*) minutes RM_SECOND(*sec*) — Wait for (*sec*) seconds RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
Mode	Allocation method for memory: RM_AUTOFREE — The memory is freed automatically with `RmFreeAll`. It is assigned to a specific task. RM_NOAUTOFREE — The memory is not freed automatically with `RmFreeAll`.
Size	Size of the memory block (–1 = largest available block)
ppMemory	Address of pointer to a memory area.

Description

The function allocates a memory area of size *Size* from the HEAP. **ppMemory* subsequently contains a valid pointer (32-bit “flat”) to the allocated memory area.

Return Value

RM_OK Function successfully executed.

RM_TASK_WAITING Function had to wait for memeoy allocation

Error Codes

Error Code	Meaning
RM_GOT_TIMEOUT	A suitable memory area could not be allocated in the specified time
RM_INVALID_POINTER	A pointer was invalid
RM_INVALID_SIZE	*Size*=0 or *Size* greater than HEAP
RM_OUT_OF_MEMORY	No memory of the specified size available

See Also

RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmGetSize, RmMemPoolAlloc, RmReAlloc, RmGetMemPoolinfo

RmCatalog

Function **Enter resource in resource catalog**

Syntax

```
#include <rmapi.h>
int          RmCatalog (
               uint Type,
               uint ID,
               ulong IDEx,
               char * pName)
```

Parameters

Parameter Name	Meaning
Type	Resource type (see *ID*).
ID	Resource ID The possible IDs depend on *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	Extended ID
pName	Pointer to a C string containing the name of the entry in the resource catalog. The string may be up to 15 characters + \0.

Description The function enters the specified parameters in the resource catalog.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_CATALOG_EXCEEDED	Catalog is full.
RM_OUT_OF_MEMORY	An internal attempt to allocate memory from the HEAP has failed.
RM_INVALID_TYPE	The specified type is illegal. 0≤*Type*≤11
RM_INVALID_ID	The specified ID is illegal.
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.

Error Code	Meaning
RM_INVALID_POINTER	The pointer to the string is invalid.
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged

See Also **RmUnCatalog, RmGetName, RmGetEntry, RmList**

RmCreateBinSemaphore

Function

Create semaphore

Syntax

```
#include <rmapi.h>
int          RmCreateBinSemaphore(
               char *pSemaphoreName,
               uint *pSemaphoreID);
```

Parameters

Parameter Name	Meaning
pSemaphoreName	Pointer to a C string containing the name used to catalog the semaphore. If this pointer = NUL, the semaphore is not cataloged. The C string may be up to 15 characters + \0.
pSemaphoreID	Pointer to semaphore ID

Description

`RmCreateBinSemaphore` creates a semaphore. The semaphore ID is returned in the specified memory area. The maximum number of semaphores is 1024.

The semaphore is cataloged automatically under the specified name. If a null pointer is passed in *pSemaphoreName*, no semaphore is cataloged.

Return Value

RM_OK — Function successfully executed, **pSemaphoreID* contains a valid semaphore ID.

Error Codes

Error Code	Meaning
RM_OUT_OF_SEMAPHORES	The request exceeds the maximum number of semaphores.
RM_INVALID_POINTER	A pointer was invalid.
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.

See Also

RmDeleteBinSemaphore, RmReleaseBinSemaphore, RmGetBinSemaphore

RmCreateChildTask

Function **Create child task**

Syntax

```
#include <rmapi.h>
int          RmCreateChildTask (
               char * pTaskName,
               ulong TaskStackSize,
               uint Priority,
               rmfarproc TaskEntry,
               uint * pTaskID)
```

Parameters

Parameter Name	Meaning
pTaskName	Pointer to a C string containing the name used to catalog the task. If this pointer = NUL, the TASK is not cataloged. The C string may be up to 15 characters + \0.
TaskStackSize	Size of the required stack in words (32 -bit).
Priority	Task priority (0..255) RM_CURPRI is the same priority as the calling task.
TaskEntry	Entry address for the task.
pTaskID	Pointer to task ID

Description

`RmCreateChildTask` declares tasks to the operating system. The task is transferred from the NOTEXISTENT state to the DORMANT state. The task is cataloged automatically under the specified name. If a null pointer is passed in *pTaskName*, no task is cataloged.

When it is created, the child task inherits the console, the current working directory and the environment from the parent task.

Return Value

RM_OK Function successfully executed. **pTaskID* contains the valid task ID.

Error Codes

Error Code	Meaning
RM_OUT_OF_MEMORY	Insufficient memory to create stack segment or insufficient memory for `RmCatalog`.
RM_INVALID_SIZE	The length specified for the stack was 0 or ≥ 1GB
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.

Error Code	Meaning
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.
RM_INVALID_TASK_ENTRY	The entry address for the task is invalid.
RM_INVALID_POINTER	The pointer to the string is incorrect, or a protection error occurred.

See Also **RmCreateTask, RmDeleteTask, RmQueueStartTask, RmStartTask**

RmCreateFlagGrp

Function **Create flag group**

Syntax

```
#include <rmapi.h>
int          RmCreateFlagGrp(
               char *pFlagGrpName,
               uint *pFlagGrpID);
```

Parameters

Parameter Name	Meaning
pFlagGrpName	Pointer to a C string containing the name used to catalog the flag group. If this pointer = NUL, the flag group is not cataloged. The C string may be up to 15 characters + \0.
pFlagGrpID	Output parameter, pointer to flag group ID

Description

`RmCreateFlagGrp` creates a flag group.**pFlagGrpID* contains the valid ID of the flag group.

The flag group is cataloged automatically under the specified name. If a null pointer is passed in *pFlagGrpName*, no flag group is cataloged.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_OUT_OF_FLAGGROUPS	The request exceeds the maximum number of event flags.
RM_INVALID_POINTER	A pointer was invalid.
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.

See Also **RmSetFlag, RmResetFlag, RmSetFlagDelayed, RmGetFlag, RmDeleteFlagGrp**

RmCreateMailbox

Function **Create mailbox**

Syntax

```
#include <rmapi.h>
int          RmCreateMailbox(
               char *pMailboxName,
               uint *pMailboxID);
```

Parameters

Parameter Name	Meaning
pMailboxName	Pointer to a C string containing the name used to catalog the mailbox. If this pointer = NUL, the mailbox is not cataloged. The C string may be up to 15 characters + \0.
pMailboxID	Pointer to a mailbox ID

Description

`RmCreateMailbox` creates a a mailbox. **pMailboxID* contains the valid mailbox ID.

The mailbox is cataloged automatically under the specified name. If a null pointer is passed in *pMailboxName*, no mailbox is cataloged.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_POINTER	A pointer was invalid.
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.
RM_OUT_OF_MAILBOXES	The request exceeds the maximum number of mailboxes.
RM_OUT_OF_MEMORY	No memory of the specified size available

See Also **RmDeleteMailbox, RmSendMail, RmReceiveMail**

RmCreateMemPool

Function **Create memory pool larger than 64 Kbytes**

Syntax

```
#include <rmapi.h>
int          RmCreateMemPool(
               char *pPoolName,
               void *pPoolAddress,
               ulong Size,
               uint *pPoolID);
```

Parameters

Parameter Name	Meaning
pPoolName	Pointer to a C string containing the name used to catalog the memory pool. If this pointer = NUL, the memory pool is not cataloged. The C string may be up to 15 characters + \0.
pPoolAddress	Pointer to the memory area in which the pool is to be created.
Size	Length of the memory area in bytes
pPoolID	Pointer to pool ID

Description

`RmCreateMemPool` defines a memory pool located at a paragraph boundary. **pPoolID* contains the valid memory pool ID. The maximum number of memory pools is 8. The minimum size of a memory area is 16 bytes.

The memory for a memory pool can be allocated from the HEAP with `RmAlloc`. The address returned by `RmAlloc` is used as the address for the memory pool.

On initialization, the memory pools are located at the next base address divisible by 16. The length is reduced to the next value divisible by 16.

The memory pool is cataloged automatically under the specified name. If a null pointer is passed in *pPoolName*, no memory pool is cataloged.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_OFFSET	The offset (*pPoolAddress*) was outside the valid range.
RM_INVALID_SIZE	A size parameter was invalid (*Size* < 16).
RM_INVALID_POINTER	A pointer was invalid.
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.

Error Code	Meaning
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.
RM_OUT_OF_MEMORYPOOLS	The request exceeds the maximum number of memory pools.

See Also **RmDeleteMemPool, RmFree, RmFreeAll, RmMemPoolAlloc**

RmCreateMessageQueue

Function **Create message queue**

Syntax

```
#include <rmapi.h>
int          RmCreateMessageQueue (
               char * pMessageQueueName,
               uint TaskID)
```

Parameters

Parameter Name	Meaning
pMessageQueueName	Pointer to a C string containing the name used to catalog the message queue. If this pointer = NUL, the message queue is not cataloged. The C string may be up to 15 characters + \0.
TaskID	Destination task-ID

Description

The function creates a message queue for the task specified by *TaskID*.

The message queue is cataloged automatically under the specified name. If a null pointer is passed in *pMessageQueueName*, no message queue is cataloged.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Invalid task ID.
RM_QUEUE_EXIST	Message queue already exists.
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_POINTER	A pointer was invalid.
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.

See Also **RmDeleteMessageQueue, RmReadMessage, RmSendMessage**

RmCreateTask

Function

Create task

Syntax

```
#include <rmapi.h>
int          RmCreateTask (
               char * pTaskName,
               ulong TaskStackSize,
               uint Priority,
               rmfarproc TaskEntry,
               uint * pTaskID)
```

Parameters

Parameter Name	Meaning
pTaskName	Pointer to a C string containing the name used to catalog the task. If this pointer = NUL, the task is not cataloged. The C string may be up to 15 characters + \0.
TaskStackSize	Size of the required stack in words (32 -bit).
Priority	Task priority (0..255)
TaskEntry	Entry address for the task.
pTaskID	Pointer to task ID

Description

The function declares a task to the operating system. The task is transferred from the NOTEXISTENT state to the DORMANT state. **pTaskID* contains the valid task ID.

The task is cataloged automatically under the specified name. If a null pointer is passed in *pTaskName*, no task is cataloged.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_OUT_OF_MEMORY	Insufficient memory to create stack segment or insufficient memory for RmCatalog.
RM_INVALID_SIZE	The length specified for the stack was 0 or ≥ 1G
RM_CATALOG_EXCEEDED	Catalog is full (see RmCatalog).
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.

Error Code	Meaning
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, therefore it is not possible to catalog a string more than once.
RM_INVALID_TASK_ENTRY	The entry address for the task is invalid.
RM_INVALID_POINTER	The pointer to the string is incorrect, or a protection error occurred.

Note

Unlike the `RmCreateChildTask` function, the console, current directory and environment are not inherited.

See Also

RmCreateChildTask, RmDeleteTask, RmQueueStartTask, RmStartTask

RmCreateTaskEx

Function

Create a task on the operating system

Syntax

```
#include <rmapi.h>
int          RmCreateTaskEx(
               char  *pTaskName,
               RmTCDStruct *pTCD,
               uint *pTaskID);
```

Parameters

Parameter Name	Meaning
pTaskName	Pointer to a C string containing the name used to catalog the task. If this pointer = NULL, the task is not cataloged.
pTCD	Pointer to a structure of the type **RmTCDStruct**
pTaskID	Pointer to the returned task ID

Description

`RmCreateTaskEx` changes the state of a dynamic task from NONEXISTENT to DORMANT. The structure of type **RmTCDStruct** must be initialized first. All values which are not used must be 0. The structure is no longer required after the function call.

The task is subsequently always addressed using the returned task ID. The task is automatically cataloged under the specified name.

The task flags (TCD.flags) define whether the task properties for the created task are to be inherited with `RM_TFL_CHILD` (see `RmCreateChildTask`)

The RM_TFL_STK flag must always be enabled. The size of the stack is specified in words (32 bits) in TCD.stck (see example).

The priority of the task is specified in TCD.inpri (from 0 to 255).

The entry address of the task is specified in TCD.task.

Note

The flag for the coprozessor (RM_TFL_NPX) is enabled automatically at the moment that the task access to the coprozessor. For that reason the call `RmCreateTaskEx` is no more necessary and exists only for the compatibility of previous versions.

Return Value

RM_OK **pTaskID* contains a valid task ID.

Error Codes

Error Code	Meaning
RM_OUT_OF_MEMORY	Insufficient memory to create the stack segment or `RmCatalog` had insufficient memory.
RM_INVALID_SIZE	The length parameter for the stack was 0.
RM_CATALOG_EXCEEDED	Catalog is full (see `RmCatalog`).
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.
RM_IS_ALREADY_CATALOGED	The specified string is already cataloged. The string must be unique, and it is not possible to catalog a string more than once.
RM_INVALID_TASK_ENTRY	The entry address for the task is invalid.
RM_INVALID_PARAMETER	The RM_TFL_DS flag cannot be used for Flat calls.
RM_INVALID_POINTER	The pointer to the string is incorrect or a protection error has been initiated.

Example

In the following example, a task is created. The `memset` call is used to initialize the RmTCDStruct structure to 0.

```
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);
  ...
}
```

See Also

RmCreateTask, RmCreateChildTask, RmDeleteTask

RmDeleteBinSemaphore

Function **Delete semaphore**

Syntax

#include <rmapi.h>
int **RmDeleteBinSemaphore(uint** *SemaphoreID***);**

Parameters

Parameter Name	Meaning
SemaphoreID	Semaphore ID

Description

`RmDeleteBinSemaphore` deletes a semaphore created with `RmCreateBinSemaphore`.The *SemaphoreID* parameter specifies the ID of the semaphore to be deleted.

If a catalog entry was created, it is now deleted.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid ID was passed.
RM_RESOURCE_BUSY	The semaphore is still in possession of a task.

See Also **RmCreateBinSemaphore, RmReleaseBinSemaphore, RmGetBinSemaphore**

RmDeleteFlagGrp

Function **Delete flag group**

Syntax

#include <rmapi.h>

int **RmDeleteFlagGrp(uint** *FlagGrpID***);**

Parameters

Parameter Name	Meaning
FlagGrpID	ID of the flag group

Description

`RmDeleteFlagGrp` deletes a global flag group created with `RmCreateFlagGrp`. The *FlagGrpID* parameter specifies the ID of the flag group to be deleted. Deleting the local flag group with *FlagGrpID=0* is not allowed.

If a catalog entry was created, it is now deleted.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Flag group = 0 or invalid ID
RM_RESOURCE_BUSY	Tasks are still waiting for flags from this flag group to be set (`RmGetFlag`), or an `RmSetFlagDelayed` is still active.

See Also **RmCreateFlagGrp, RmGetFlag**

RmDeleteMailbox

Function **Delete mailbox**

Syntax

```
#include <rmapi.h>
int RmDeleteMailbox(uint MailboxID);
```

Parameters

Parameter Name	Meaning
MailboxID	Mailbox ID

Description

`RmDeleteMailbox` deletes a mailbox defined with `RmCreateMailbox`. The *MailboxID* parameter specifies the ID of the mailbox to be deleted.

If you delete a mailbox, which is used by an Interrupt mailbox handler, also the corresponding handler must be deleted.

If a catalog entry was created, it is now deleted.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid ID was passed.
RM_RESOURCE_BUSY	Tasks are still waiting for messages in this mailbox, or the mailbox still contains messages, or an `RmSendMailDelayed`is still active.

See Also **RmCreateMailbox**

RmDeleteMemPool

Function **Delete memory pool**

Syntax

#include <rmapi.h>
int RmDeleteMemPool(uint *PoolID***);**

Parameters

Parameter Name	Meaning
PoolID	Pool ID

Description

`RmDeleteMempool` deletes a memory pool created with `RmCreateMemPool`.The *PoolID* parameter specifies the ID of the memory pool to be deleted.

If a catalog entry was created, it is now deleted.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Pool ID = 0 (corresponds to heap ID) or invalid ID
RM_RESOURCE_BUSY	Memory areas from this pool are still allocated.

See Also **RmCreateMemPool, RmMemPoolAlloc**

RmDeleteMessageQueue

Function **Delete message queue**

Syntax

#include <rmapi.h>

int **RmDeleteMessageQueue (uint** *TaskID***)**

Parameters

Parameter Name	Meaning
TaskID	Task ID

Description

The `RmDeleteMessageQueue` function deletes the message queue for the task specified by *TaskID*.

If a catalog entry was created, it is now deleted.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Invalid task ID
RM_QUEUE_NOT_EXIST	The message queue does not exist.
RM_RESOURCE_BUSY	Messages are still waiting in the message queue, or the task with *TaskID* is still waiting for messages.

See Also **RmCreateMessageQueue, RmSendMessage, RmReadMessage**

RmDeleteTask

Function **Delete task**

Syntax

#include <rmapi.h>
int **RmDeleteTask(uint** *TaskID***);**

Parameters

Parameter Name	Meaning
TaskID	Task ID (RM_OWN_TASK = own task)

Description

`RmDeleteTask` deletes the task specified by *TaskID* if it is in the DORMANT or ACTIVE state.

If the task was initialized for CRUN, the initialization is deleted and open files are closed.

If you delete a task with `RmDeleteTask`, which was called by an Interrupt handler, also the corresponding handler must be deleted.

If a catalog entry was created, it is now deleted.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_TASK_NOT_DORMANT	An attempt was made to delete a task which is not in the DORMANT state.
RM_INVALID_ID	An invalid task ID was passed.

Note

The `RmKillTask` call can be used for tasks in other states.

See Also

RmCreateTask, RmKillTask, x_cr_killtsk

RmDisableScheduler

Function **Disable scheduler**

Syntax

```
#include <rmapi.h>
int RmDisableScheduler(void);
```

Description

`RmDisableScheduler` deactivates the scheduler. When the scheduler is deactivated, only the task which called the function is active (even higher-priority tasks are no longer allocated CPU time).

`RmDisableScheduler` cannot be nested, that is every call deactivates scheduling.

When the scheduler is deactivated, the `RmDeleteTask` and `RmRestartTask` functions cannot be called. RMOS- API- calls should also be avoided in cases where a task may have to wait for another task to finish executing. This includes:
`RmAlloc`, `RmGetEntry`, `RmQueueStartTask`, `RmReceiveMail`, `RmSendMail`, `RmStartTask`, `RmGetFlag` and `RmGetBinSemaphore`.

A CLI job cannot be canceled with <Ctrl>+<C> when the scheduler is deactivated.

If the scheduler is deactivated too long, the real-time capability of the system can suffer. This applies particularly to the use of `RmRestartTask` and `RmPauseTask`.

Note The scheduling lock is deactivated automatically as soon as a task blocks (e.g. Functions with wait option, runtime error, printf)

Return Value

RM_OK	RM_OK is always returned.

See Also **RmEnableScheduler**, scheduler description in the Programming Manual.

RmEnableScheduler

Function **Enable scheduler**

Syntax

```
#include <rmapi.h>
int RmEnableScheduler(void);
```

Description

`RmEnableScheduler` activates the scheduler deactivated with `RmDisableScheduler`.

`RmEnableScheduler` cannot be nested, that is every call reactivates scheduling.

Return Value

RM_OK RM_OK is always returned.

See Also **RmDisableScheduler**, scheduler description in the Programming Manual.

RmEndTask

Function **End task**

Syntax
```
#include <rmapi.h>
void        RmEndTask(void);
```

Description `RmEndTask` terminates execution of the task. The task is switched to the DORMANT state if no further task start requests are waiting.

Note This function can also be used for tasks which use the functions of the ANSI library. The C library function `exit(x)` can also be used instead of `RmEndTask`.

Return Value The call has no return value.

See Also **RmDeleteTask, RmQueueStartTask, RmStartTask,** starting, interruption and termination of tasks.

RmFree

Function **Free a memory area**

Syntax

#include <rmapi.h>
int **RmFree(void *_pMemory_);**

Parameters

Parameter Name	Meaning
pMemory	Pointer to the memory area to be freed.

Description

`RmFree` is used to free a memory area allocated by a task with `RmAlloc` or `RmMemPoolAlloc`.

It is not possible to free part of a memory area.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_MEMORYBLOCK	Memory area was not allocated.
RM_INVALID_POINTER	A pointer was invalid.

See Also

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFreeAll, RmMemPoolAlloc, RmReAlloc

RmFreeAll

Function **Free all memory areas of a task**

Syntax

#include <rmapi.h>
int RmFreeAll(uint *TaskID***);**

Parameters

Parameter Name	Meaning
TaskID	ID of the task whose entire memory area is to be freed (RM_OWN_TASK = own task).

Description

`RmFreeAll` is used to free all memory areas allocated by a task with `RmAlloc` or `RmMemPoolAlloc`. `RmFreeAll` frees also memory areas which was allocated with the C Runtime library functions `malloc`, `calloc` or `realloc`.

It is not possible to free part of any memory area.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Invalid *TaskID*
RM_INVALID_POINTER	A pointer was invalid.

Note

An error message is not output if the task has not allocated any memory. Memory which the task has allocated with RM_NOAUTOFREE is not freed.

See Also

calloc, malloc, realloc, RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmMemPoolAlloc, RmReAlloc

RmGetAbsTime

Function **Get absolute system time**

Syntax

```
#include <rmapi.h>
int RmGetAbsTime(RmAbsTimeStruct *pAbsTime);
```

Parameters

Parameter Name	Meaning
pAbsTime	Pointer to a structure of type **RmAbsTimeStruct** containing the absolute system time.

Description `RmGetAbsTime` copies the absolute system time in milliseconds since the last complete restart to a structure of type **RmAbsTimeStruct**.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_POINTER	Invalid *pAbsTime*

See Also **RmAbsTimeStruct**

RmGetBinSemaphore

Function

Test and set semaphore

Syntax

```
#include <rmapi.h>
int          RmGetBinSemaphore(
               ulong TimeOutValue,
               uint SemaphoreID);
```

Parameters

Parameter Name	Meaning
TimeOutValue	Maximum time to wait for execution
	RM_CONTINUE — Continue task and do not wait for semaphore
	RM_WAIT — Wait for semaphore
	0 ... RM_MAXTIME — Time interval in ms. The task waits until it receives the semaphore or the time has expired.
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.
	RM_HOUR(*hour*) — Wait for (*hour*) hours
	RM_MINUTE(*min*) — Wait for (*min*) minutes
	RM_SECOND(*sec*) — Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
SemaphoreID	Semaphore ID

Description

`RmGetBinSemaphore` tests and sets a semaphore.

Return Value

RM_OK	Function successfully executed.
RM_TASK_WAITING	Task had to wait for semaphore.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *SemaphoreID* was passed.
RM_GOT_TIMEOUT	The call was canceled after the configured timeout time.
RM_RESOURCE_NOT_AVAILABLE	The desired resource is not available.

Note

The allocation and release of semaphores are not task-specific.

See Also

RmCreateBinSemaphore, RmDeleteBinSemaphore, RmReleaseBinSemaphore

RmGetEntry

Function **Find entry in catalog**

Syntax

```
#include <rmapi.h>
int          RmGetEntry (
               ulong TimeOutValue,
               char *pName
               RmEntryStruct *pEntry)
```

Parameters

Parameter Name	Meaning
TimeOutValue	Maximum time to wait for execution RM_CONTINUE — Continue task and do not wait for the entry to be cataloged RM_WAIT — Wait for the entry to be cataloged 0 ... RM_MAXTIME — Time interval in ms. The task waits until either the entry is cataloged or the time has expired. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) — Wait for (*hour*) hours RM_MINUTE(*min*) — Wait for (*min*) minutes RM_SECOND(*sec*) — Wait for (*sec*) seconds RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
pName	Address of the name to be found in the catalog. The string can also be defined using C or PLM notation.
pEntry	Address of a structure of the type **RmEntryStruct,** see chapter 3.

Description `RmGetEntry` searches for an entry in the resource catalog.

Return Value

RM_OK	Function successfully executed.
RM_TASK_WAITING	The task had to wait for entry to be cataloged.

Error Codes

Error Code	Meaning
RM_INVALID_STRING	The length of the string is illegal. It is either zero or greater than 15.
RM_IS_NOT_CATALOGED	The specified string is not cataloged (only if TimeOutValue == RM_CONTINUE)

Error Code	Meaning
RM_GOT_TIMEOUT	The time has expired but the string has not been cataloged.
RM_INVALID_POINTER	The pointer to the string or structure is incorrect, or a protection error occurred.

See Also **RmCatalog, RmUncatalog, RmGetName**

RmGetFlag

Function **Test event flag**

Syntax

```
#include <rmapi.h>
int     RmGetFlag(
          ulong TimeOutValue,
          uint Type,
          uint FlagGrpID,
          uint TestMask,
          uint *pFlagMask);
```

Parameters

Parameter Name	Meaning	
TimeOutValue	Maximum time to wait for execution	
	RM_CONTINUE	Continue task without waiting for event flag to be set.
	RM_WAIT	Wait for the event flag to be set
	0 ... RM_MAXTIME	Time interval in ms. The task waits until either the event flag has been set or the time has expired.
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.	
	RM_HOUR(*hour*)	Wait for (*hour*) hours
	RM_MINUTE(*min*)	Wait for (*min*) minutes
	RM_SECOND(*sec*)	Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*)	Wait for (*ms*) milliseconds
Type	RM_TEST_ALL	Test if all the specified bits have been set
	RM_TEST_ONE	Test if at least one bit has been set
FlagGrpID	ID of the flag group. 0 specifies the local flag group.	
TestMask	The mask defines which bits are tested	
pFlagMask	Pointer to a uint which returns the values of all bits in the flag group.	

Description

`RmGetFlag` tests a flag group to establish whether all (RM_TEST_ALL) or at least one (RM_TEST_ONE) of the specified bits have been set. If a wait time is specified, the task waits for the bits to be set. The bits of a flag group are ANDed with TestMask, and returned in *pFlagMask*.

Return Value

RM_OK	Function successfully executed.
RM_TASK_WAITING	Call had to wait for the flag to be set.
RM_FLAG_ALREADY_SET	The flag was already set.

Error Codes

Error Code	Meaning
RM_TEST_NOT_OK	One or more flags in *TestMask* not set (only with RM_CONTINUE)
RM_INVALID_ID	An invalid *FlagGrpID* was passed.
RM_GOT_TIMEOUT	The call was canceled after the configured timeout expired.
RM_INVALID_POINTER	The pointer to *pFlagMask* is invalid, or a protection error occurred.

See Also

RmSetFlag, RmSetFlagDelayed, RmResetFlag

RmGetIntHandler

Function **Read out interrupt handler**

Syntax

```
#include <rmapi.h>
int          RmGetIntHandler (
               uint IntNum
               rmfarproc *pHandlerEntry);
```

Parameters

Parameter Name	Meaning
IntNum	Interrupt Number (0–255) IRQx (x=0 to 63) Hardware interrupt IRQ(n) (n=0 to 63) Hardware interrupt The hardware interrupts in PC hardware are at 0 to 15.
pHandlerEntry	Entry address of interrupt handler

Description `RmGetIntHandler` is used to read the current interrupt handler from the IDT.

Return Value

RM_OK	Function successfully executed, **pHandlerEntry* contains the entry address of the associated interrupt handler.

Error Codes

Error Code	Meaning
RM_INVALID_INTERRUPT_NUMBER	Invalid interrupt number
RM_INVALID_IRQ_NUMBER	IRQx invalid, PIC not defined
RM_INVALID_POINTER	Invalid pointer

See Also **RmSetIntDefHandler, RmSetIntISHandler, RmSetIntMailboxHandler, RmSetIntTaskHandler,**

RmGetMemPoolInfo

Function **Check memory pool information**

Syntax

```
#include <rmapi.h>
int          RmGetMemPoolInfo(
               uint PoolID,
               RmMemPoolInfoStruct *pInfo)
```

Parameters

Parameter Name	Meaning
PoolID	ID of the memory pool (RM_HEAP for heap)
pInfo	Pointer to structure of the type **RmMemPoolInfoStruct**.

Description

The `RmGetMemPoolInfo` function returns the size of the pool, of the available memory, and of the largest available block (`RmAlloc`(Size=–1)).The information about the pool specified by *PoolID* is stored in the **RmMemPoolInfoStruct** structure *pInfo* points to this structure.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Pool ID invalid
RM_INVALID_POINTER	*pInfo* is an invalid pointer

See Also **RmMemPoolInfoStruct**

RmGetName

Function **Search catalog for entry**

Syntax

```
#include <rmapi.h>
int     RmGetName (
          uint Type,
          uint ID,
          ulong IDEx,
          char * pName)
```

Parameters

Parameter Name	Meaning
Type	Resource type (see *ID*)
ID	Resource 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	Extended resource ID (–1 = not specified)
pName	Address of a buffer in which the string is to be stored. The length of the buffer must be at least 15 characters + \0.

Description The `RmGetName` searches through a catalog and returns the name belonging to Type, ID and IDEx.

Return Value

RM_OK	Function successfully executed, the buffer contains the valid name of the specified resource.

Error Codes

Error Code	Meaning
RM_INVALID_TYPE	The specified type is illegal. 0≤*Type*≤11
RM_INVALID_ID	The specified ID is illegal.

Error Code	Meaning
RM_IS_NOT_CATALOGED	A matching entry was not found.
RM_INVALID_POINTER	The pointer to the string is invalid.

See Also **RmCatalog, RmUncatalog, RmGetEntry**

RmGetSize

Function **Get the size of a memory area**

Syntax

```
#include <rmapi.h>
int          RmGetSize(
               void *pMemory,
               ulong *pSize);
```

Parameters

Parameter Name	Meaning
pMemory	Pointer to the memory area
pSize	Pointer to the memory location where the length of the memory area is returned.

Description

This function can be used to determine the length of a memory area previously allocated with `RmAlloc` or `RmMemPoolAlloc`. **pSize* contains the length of the specified memory area.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_MEMORY_BLOCK	Memory area was invalid.
RM_INVALID_SIZE	A size was invalid.
RM_INVALID_POINTER	A pointer was invalid.

See Also

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmMemPoolAlloc, RmReAlloc

RmGetTaskID

Function **Get task ID**

Syntax

```
#include <rmapi.h>
int          RmGetTaskID(
               uint Tcb,
               uint *pTaskID);
```

Parameters

Parameter Name	Meaning
Tcb	Only RM_OWN_TASK (= own task) allowed
pTaskID	Pointer to task ID

Description `RmGetTaskID` can be used to determine the task ID of the present task. **pTaskID* contains the valid task ID of the present task.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_POINTER	A pointer was invalid.
RM_PARAMETER_ERROR	A parameter other than RM_OWN_TASK was passed.

See Also **RmCreateTask, RmCreateChildTask**

RmGetTaskPriority

Function **Get task priority**

Syntax

```
#include <rmapi.h>
int          RmGetTaskPriority(
               uint TaskID,
               uint *pPriority);
```

Parameters

Parameter Name	Meaning
TaskID	Task ID (RM_OWN_TASK = own task)
pPriority	Pointer to a memory location containing the priority of the task.

Description `RmGetTaskPriority` returns the task priority. **pPriority* contains the priority of the specified task.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	TaskID invalid
RM_INVALID_POINTER	A pointer was invalid.

See Also **RmGetTaskState**

RmGetTaskState

Function **Get task state**

Syntax

```
#include <rmapi.h>
int          RmGetTaskState(
               uint TaskID,
               uint *pTaskState);
```

Parameters

Parameter Name	Meaning
TaskID	Task ID (RM_OWN_TASK = own task)
pTaskState	Pointer to a memory location containing the state of the task. Possible task states are: RM_READY — Task in READY state RM_DORMANT — Task in DORMANT state RM_ACTIVE — Task in ACTIVE state RM_BLOCKED — Task in BLOCKED state The reason for the state is coded in the 6 most significant bits of *pTaskState. **pTaskState* can have one of the following values: RM_STA_EF — Waiting for event flag RM_STA_SEMA — Waiting for semaphore RM_STA_LOAD — Waiting until destination task is loaded RM_STA_STRT — Waiting for destination task to start RM_STA_ENDT — Waiting for destination task to end RM_STA_MSG — Waiting for a message to be received RM_STA_MSGRCVD — Waiting for a dispatched message to be received RM_STA_POOL — Waiting for memory to be allocated from a memory pool RM_STA_HLT — Halted by DEBUGGER or by RmSuspendTask RM_STA_BREAK — Interrupted by DEBUGGER breakpoint

Parameter Name	Meaning	
	RM_STA_PAUSE	Waiting for expiry of a time interval (`RmPauseTask`)
	RM_STA_WAIT	Waiting for time interval to expire
	RM_STA_ERR0	Runtime error, type 0 (Division by 0 Interrupt)
	RM_STA_ERR1	Runtime error, type 1 (Single Step Interrupt)
	RM_STA_ERR2	Runtime error, type 3 (Breakpoint Interrupt)
	RM_STA_ERR3	Runtime error, type 4 (Overflow Interrupt)
	RM_STA_ERR4	Runtime error, type 5 (Array Bound Interrupt)
	RM_STA_ERR5	Runtime error, type 6 (Unused Opcode)
	RM_STA_ERR6	Runtime error, type 7 (Escape Opcode)
	RM_STA_ERR7	Runtime error, type 8 (Double Fault)
	RM_STA_ERR8	Runtime error, type 9 (NDP Segment Overrun)
	RM_STA_ERR9	Runtime error, type 10 (Invalid TSS)
	RM_STA_ERR10	Runtime error, type 11 (Segment Not Present)
	RM_STA_ERR11	Runtime error, type 12 (Stack Fault)
	RM_STA_ERR12	Runtime error, type 13 (General Protection)
	RM_STA_ERR13	Runtime error, type 14 (Page Fault)
	RM_STA_ERR14	Runtime error, type 16 (Floating Point Error)
	RM_STA_ERR15	Runtime error, type 17 (Alignment Check)
	RM_STA_LOOK	Waiting for catalog entry
	RM_STA_KEND	Task terminated by `RmKillTask` (after completion of a running I/O operation)
	RM_STA_KDEL	Task deleted by `RmKillTask` (after completion of a running I/O operation)
	RM_ACTIVE	Task in ACTIVE state.

Description `RmGetTaskState` returns the task state.

Return Value

RM_OK	Function successfully executed, **pTaskState* contains the state of the specified task.

Error Codes

Error Code	Meaning
RM_INVALID_ID	TaskID invalid
RM_INVALID_POINTER	A pointer was invalid.

See Also **RmGetTaskPriority**

Note If a task does not exist `RmGetTaskState` returns RM_INVALID_ID.

RmIOClose

Function **Close unit**

Syntax

#include <rmapi.h>
int **RmIOClose(RmIOHandle** *Handle***);**

Parameters

Parameter Name	Meaning
Handle	Descriptor

Description

`RmIOClose` closes the unit specified by *Handle*. *Handle* is a descriptor that was generated with `RmIOOpen`. If the unit was reserved for the calling task, it is released again (by the driver), and waiting requests of other tasks are processed.

The `RmIOClose` call does not have a blocking effect if the unit is reserved for another task.

Return Value

RM_OK The function was successfully executed

Error Codes

Error Code	Meaning
RM_BOUND_REACHED	Message queue of unit full
RM_EIO_UNIT_RESET	Request canceled by control function RM_IOCTL_RESET
RM_INVALID_HANDLE	Descriptor is invalid
RM_OUT_OF_MEMORY	Not enough memory available in heap
RM_QUEUE_NOT_EXIST	Message queue of unit has not yet been set up

See Also **RmIOControl, RmIOOpen, RmIORead, RmIOWrite, RmLoadDevice**

RmIOControl

Function **Control function for loadable drivers**

Syntax

```
#include <rmapi.h>
int          RmIOControl(
               uint Wait,
               uint FlagMask,
               RmIOHandle Handle,
               uint Control,
               void *pBuffer,
               int *pIOStatus);
```

Parameters

Parameter Name	Meaning
Wait	Specifies whether the control function is to be executed with or without waiting. RM_CONTINUE — Continue task without waiting for control function to finish RM_WAIT — Wait for control function to finish
FlagMask	Bit mask to be enabled in the local flag group of the calling task on termination of the control function (with RM_CONTINUE)
Handle	Descriptor
Control	Function code of the control function, see below
pBuffer	Pointer to parameter block for the control function.
pIOStatus	Pointer to `int` with error status of the operation or NULL pointer

Description

`RmIOControl` executes a control function on the unit specified by *Handle*. *Handle* is a descriptor that was generated with `RmIOOpen`.

The *Wait* parameter specifies whether the task is to wait for the control function to finish (RM_WAIT), or whether it is to continue (RM_CONTINUE).

The *FlagMask* parameter can be used to specify a bit mask in the local flag group (FlagGroupId=0) which will be enabled after termination of the control function when a call without wait is executed. If 0 is specified, no bit mask is enabled.

The *Control* parameter specifies the control function to be executed. If the unit does not support the specified control function, the control function is terminated with RM_EIO_INVALID_CONTROL.

pBuffer is used to pass a parameter block, the structure of which depends on the specified control function.

On termination of the control function, the status is entered in the `int` to which *pIOStatus* points. In requests with wait, this status is identical to the return value of the call. If the request is executed without wait, the value RM_IO_QUEUED is stored there while the request is located in the queue. During processing by the driver, the value RM_IO_IN_PROGRESS is stored there. After processing, the error status of the operation is stored there. If the return value of the status in *pIOStatus* is not required (e.g. because of a call with RM_WAIT), a NULL pointer can be passed. In this case, the status is only reported as the return value of the function.

Control Functions

Below you will find the control functions available for the serial interface driver SER8250.DRV and the 3964(R) driver 3964.DRV.

Control functions for SER8250.DRV

RM_IOCTL_BUFFER_FLUSH
Flush background buffer. *pBuffer* is ignored.

RM_IOCTL_BUFFER_GETSIZE
Find out the size of the background buffer. The buffer size in number of characters is written to `ulong`, to which *pBuffer* points.

RM_IOCTL_BUFFER_SETSIZE
Set the size of the background buffer. Data already stored in the background buffer are deleted. In the event of an error (e.g. not enough free memory), the background buffer remains unchanged. *pBuffer* points to a `ulong` which specifies the new buffer size in number of characters.

RM_IOCTL_BUFFER_USED
Determine the number of characters in the background buffer. The number is stored in a `ulong` to which *pBuffer* points.

RM_IOCTL_CANCEL
Cancel current I/O request. *pBuffer* is ignored.

RM_IOCTL_GET_PROPERTIES
Determine the function scope of the driver. *pBuffer* points to a structure of the type `RmIOCTLPropertiesStruc`t.

RM_IOCTL_GET_VERSION
Find out version of the driver. *pBuffer* points to a structure of the type `RmIOCTLVersionStruct`.

RM_IOCTL_INIT
Configure unit with new values. *pBuffer* points to a structure of the type `Ser8250InitStruct`, which is used to pass the configuration data.

RM_IOCTL_INIT_ASCII
Configure unit with new values. The new configuration values are passed in the form of ASCII strings. *pBuffer* points to an array of pointers which point to the configuration parameters. The last element of the array must be a NULL pointer.
The following configuration parameters are permitted:

“IRQ:<irq number>”
<irq number> IRQ number of the interface (e.g. 4 for COM1).

This parameter is only permitted in the first RM_IOCTL_INIT_ASCII or RM_IOCTL_INIT call for a unit (e.g. DEVICE command).

“BASE:<i/o address>”
<i/o address> I/O base address of the 8250 (e.g. 0x3F8 for COM1)
This parameter is only permitted in the first RM_IOCTL_INIT_ASCII or RM_IOCTL_INIT call for a unit (e.g. DEVICE command).

“MODE:<baud rate>–<parity>–<data bit>–<stop bit>”
Configuration of the communication parameters. The meanings are as follows:

<baud rate> Baud rate.
All values by which 115200 can be divided without remainder are permitted.

<parity> Parity. The following parameters are permitted:

N	No parity check
E	Even parity
O	Odd parity
S	Parity bit always set to 0 (space)
M	Parity bit always set to 1 (mark)

<data bit> Number of data bits. The following numbers are permitted: 5, 6, 7, 8

<stop bit> Number of stop bits. The following settings are permitted:

1	1 stop bit
2	2 stop bits (not with 5 data bits)
15	1.5 stop bits (only with 5 data bits)

“BUFFER:<size>”
<size> Size of the background buffer

Example:

```
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
Read in the current configuration of the unit. *pBuffer* points to a buffer with the structure of type `Ser8250InitStruct`.

RM_IOCTL_MODE
Configure unit with new values for communication (e.g. baud rate). *pBuffer* points to a structure of type `RmIOCTLModeSerialStruct`.

RM_IOCTL_READLEN
Define the number of characters after which read requests are terminated automatically (only valid when activated by RM_IOCTL_READSTOP). *pBuffer* must point to a `ulong` which contains the number of characters.

RM_IOCTL_READLEN_GET
Read in the number of characters defined by RM_IOCTL_READLEN. The number of characters is written to the `ulong` to which *pBuffer* points.

RM_IOCTL_READ_MODE
Select the mode of `RmIORead`. *pBuffer* points to a ulong in which either RM_WAIT or RM_CONTINUE is specified.

When RM_WAIT is specified, a read request is not completed until the end condition (number of characters, stop character, timeout, ...) has been attained or an error occurs. When RM_CONTINUE is specified, the read request is terminated with RM_IO_NO_DATA when no data (including the end condition) are stored in the background buffer.

The default setting is RM_WAIT.

RM_IOCTL_READSTOP
Define which end condition is used for read requests. The stop character(s) is (are) not written to the user buffer. The end condition is defined by the `char` to which *pBuffer* points. The following values are permitted:

SER8250_READSTOP_OFF
Do not use end condition

SER8250_READSTOP_CHAR_1
Use stop character 1

SER8250_READSTOP_CHAR_1_2
Use stop characters 1 and 2, that is cancel when the 1st character is followed by the 2nd stop character.

SER8250_READSTOP_LEN
Terminate read request when the number of characters defined by RM_IOCTL_READLEN have been read in.

SER8250_READSTOP_CHAR_1 or SER8250_READSTOP_CHAR_1_2 and SER8250_READSTOP_LEN can be combined using OR logic.

The default setting is SER8250_READSTOP_OFF.

RM_IOCTL_READSTOP1
Define stop character 1 that terminates the read request. Only valid when activated by RM_IOCTL_READSTOP. *pBuffer* must point to a `char` which contains the stop character.

RM_IOCTL_READSTOP2
Define stop character 2 that terminates the read request. Only valid when activated by RM_IOCTL_READSTOP. *pBuffer* must point to a `char` which contains the stop character.

RM_IOCTL_READSTOP_GET
Read in the end condition activated by RM_IOCTL_READSTOP and the entered stop character. *pBuffer* must point to an array with 3 char in which the current values of RM_IOCTL_READSTOP, RM_IOCTL_READSTOP1 and RM_IOCTL_READSTOP2 are entered.

RM_IOCTL_READTIMEOUT
Define a time span (in ms) specifying the maximum pause between two

characters during read requests. If the pause is longer, the read request is terminated. Specifying RM_CONTINUE deactivates the timeout. *pBuffer* must point to a `ulong` which specifies the time span.

The default setting is RM_CONTINUE.

RM_IOCTL_READTIMEOUT_GET
Read in the time span specified by RM_IOCTL_READTIMEOUT. The time span is written to the `ulong` to which *pBuffer* points.

RM_IOCTL_RELEASE
Release the unit. I/O requests which were blocked while the unit was reserved are now executed. *pBuffer* is ignored.

RM_IOCTL_RESERVE
Reserve unit for calling task. I/O requests of other tasks are accepted, but are not executed until the unit is released. *pBuffer* is ignored.

RM_IOCTL_RESET
Reset and restart the unit. All I/O requests of the unit which have not yet been executed are rejected with RM_EIO_UNIT_RESET. The unit must subsequently be reinitialized (with control functions RM_IOCTL_INIT or RM_IOCTL_INIT_ASCII). *pBuffer* is ignored.

RM_IOCTL_WRITEDELAY
Define a time span (in ms) specifying the minimum pause observed after transmission of the last character during write requests by the driver, before the request is terminated and a new request is processed. Specifying RM_CONTINUE deactivates the timeout.

pBuffer must point to a `ulong` in which the time span is specified.

The default setting is RM_CONTINUE.

RM_IOCTL_WRITEDELAY_GET
Read in the time span specified by RM_IOCTL_WRITEDELAY. The time span is written to the `ulong` to which *pBuffer* points.

RM_IOCTL_WRITESTOP
Define which end condition is used for write requests. The stop character(s) is (are) transferred in addition to the data sent by the user. The end condition is defined by the `char` to which *pBuffer* points. The following values are permitted:

SER8250_WRITESTOP_OFF
Do not use end condition

SER8250_WRITESTOP_CHAR_1
Use stop character 1

SER8250_WRITESTOP_CHAR_1_2
Use stop character 1 followed by stop character 2

The default setting is SER8250_WRITESTOP_OFF.

RM_IOCTL_WRITESTOP1
Define stop character 1 for write requests. Only valid when activated by RM_IOCTL_WRITESTOP. *pBuffer* must point to a `char` which contains the stop character.

RM_IOCTL_WRITESTOP2
Define stop character 2 for write requests. Only valid when activated by RM_IOCTL_WRITESTOP. *pBuffer* must point to a `char` which contains the stop character.

RM_IOCTL_WRITESTOP_GET
Read in the end condition activated by RM_IOCTL_WRITESTOP and the entered stop character. *pBuffer* must point to an array with 3 char in which the current values of RM_IOCTL_WRITESTOP, RM_IOCTL_WRITESTOP1 and RM_IOCTL_WRITESTOP2 are entered.

Control functions for 3964.DRV

RM_IOCTL_CANCEL
Cancel current I/O request. *pBuffer* is ignored.

RM_IOCTL_GET_PROPERTIES
Determine the function scope of the driver. *pBuffer* points to a structure of the type `RmIOCTLPropertiesStruc`t.

RM_IOCTL_GET_VERSION
Find out version of the driver. *pBuffer* points to a structure of the type `RmIOCTLVersionStruct`.

RM_IOCTL_INIT
Configure unit with new values. *pBuffer* points to a structure of the type `Rm3964InitStruct`, which is used to pass the configuration data.

RM_IOCTL_INIT_ASCII
Configure unit with new values. The new configuration values are passed in the form of ASCII strings. *pBuffer* points to an array of pointers which point to the configuration parameters. The last element of the array must be a NULL pointer.

The following parameters are permitted:
“IRQ:<irq number>”
<irq number> IRQ number of the interface over which the driver is to communicate (e.g. 4 for COM1). This parameter is only permitted in the first RM_IOCTL_INIT_ASCII or RM_IOCTL_INIT call for a unit (e.g. DEVICE command).

“BASE:<i/o address>”
<i/o address> I/O base address of the interface over which the driver is to communicate (e.g. 0x3F8 for COM1). This parameter is only permitted in the first RM_IOCTL_INIT_ASCII or RM_IOCTL_INIT call for a unit (e.g. DEVICE command).

“MODE:<baud>–<parity>–<data>–<stop>”
Communication parameters:
<baud rate> Baud rate.
All values by which 115200 can be divided without remainder are permitted.

<parity> Parity.
The following parameters are permitted:

N	No parity check
E	Even parity

O	Odd parity
S	Parity bit always set to 0 (space)
M	Parity bit always set to 1 (mark)

<data bit> Number of data bits. The following numbers are permitted: 5, 6, 7, 8

<stop bit> Number of stop bits.
The following settings are permitted:

1	1 stop bit
2	2 stop bits (not with 5 data bits)
15	1.5 stop bits (only with 5 data bits)

"PROT:<protocol>–<master>"
Protocol parameters:

<protocol>	Selection of protocol 3964 or 3964R: 1 for 3964R, 0 for 3964
<master>	Selection of master or slave: 1 for master, 0 for slave

Example:

```
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
Read in the current configuration of the unit. *pBuffer* points to a buffer with the structure `Rm3964InitStruct`.

RM_IOCTL_MODE
Configure unit with new values for communication (e.g. baud rate).
pBuffer points to the configuration data, which are to be passed to a structure `RmIOCTLModeSerialStruct`.

RM_IOCTL_RELEASE
Release the unit. I/O requests which were blocked while the unit was reserved are now executed. *pBuffer* is ignored.

RM_IOCTL_RESERVE
Reserve unit for calling task. I/O requests of other tasks are accepted, but are not executed until the unit is released. *pBuffer* is ignored.

RM_IOCTL_RESET
Reset and restart the unit. All I/O requests of the unit which have not yet been executed are rejected with RM_EIO_UNIT_RESET. The unit must subsequently be reinitialized (with control functions RM_IOCTL_INIT or RM_IOCTL_INIT_ASCII). *pBuffer* is ignored.

Return Value

RM_OK	The function was successfully executed

Error Codes

Error Code	Meaning
RM_BOUND_REACHED	Message queue of unit full.
RM_EIO_INVALID_CONTROL	The specified control function is not supported
RM_EIO_UNIT_RESET	Request canceled by RM_IOCTL_RESET control function
RM_EIO_×××	Other error codes of the operation
RM_INVALID_POINTER	Pointer invalid
RM_INVALID_TYPE	Invalid value for *Wait*
RM_INVALID_HANDLE	*Handle* invalid
RM_IO_QUEUED	Request waiting in message queue
RM_IO_IN_PROGRESS	Request is being processed
RM_OUT_OF_MEMORY	Not enough free memory available in heap
RM_QUEUE_NOT_EXIST	Message queue of unit has not yet been set up

See Also

RmIOClose, RmIOOpen, RmIORead, RmIOWrite, RmLoadDevice

RmIOOpen

Function

Open unit

Syntax

```
#include <rmapi.h>
int          RmIOOpen(
               const char * pUnitName,
               uint Mode,
               RmIOHandle * pHandle);
```

Parameters

Parameter Name	Meaning
pUnitName	Name of the unit in the RMOS resource catalog
Mode	Mode for opening the unit RM_IO_READ — Open unit for read access RM_IO_WRITE — Open unit for write access RM_IO_RESERVE — Reserve unit for task
pHandle	Pointer to a variable in which the descriptor for addressing unit is stored.

Description

`RmIOOpen` opens the unit specified by *pUnitName* for processing with the calls `RmIORead`, `RmIOWrite` and `RmIOControl`. `RmIOOpen` returns the descriptor of the open unit to the memory addressed by *pHandle*.

The *Mode* parameter specifies what type of accesses are to be performed on the unit. RM_IO_READ signifies read accesses and RM_IO_WRITE signifies write accesses.

Specifying RM_IO_RESERVE additionally means that only requests of the calling task are processed. Requests of other tasks are accepted, but are not executed until the unit is released with the task (`RmIOControl` with RM_IOCTL_RELEASE) or closed with `RmIOClose`.

If necessary, the values can be combined using OR logic (e.g. RM_IO_READ | RM_IO_WRITE | RM_IO_RESERVE; the unit is opened for read and write access exclusively by the calling task).

Return Value

RM_OK The function was successfully executed

Error Codes

Error Code	Meaning
RM_BOUND_REACHED	Message queue of unit full.
RM_EIO_UNIT_RESERVED	Unit is already reserved (`RmIOOpen` with RM_IO_RESERVE or `RmIOControl` with RM_IOCTL_RESERVE).

Error Code	Meaning
RM_EIO_UNIT_RESET	Request canceled by control function RM_IOCTL_RESET
RM_INVALID_POINTER	Pointer invalid
RM_INVALID_TYPE	Invalid value for *Mode*
RM_INVALID_UNIT	*UnitName* is not the unit of a loadable driver
RM_IS_NOT_CATALOGED	Unit is not cataloged with the specified name
RM_OUT_OF_MEMORY	Not enough free memory available in heap
RM_QUEUE_NOT_EXIST	Message queue of unit has not yet been set up.

See Also **RmIOClose, RmIOControl, RmIORead, RmIOWrite, RmLoadDevice**

RmIORead

Function

Read from unit

Syntax

```
#include <rmapi.h>
int          RmIORead(
               uint Wait,
               uint FlagMask,
               RmIOHandle Handle,
               ulong Length,
               void *pBuffer,
               ulong BlockAddress,
               ulong *pIOCount,
               int *pIOStatus);
```

Parameters

Parameter Name	Meaning
Wait	Specifies whether the request is to be executed with or without waiting. RM_CONTINUE — Continue task without waiting for read request to finish RM_WAIT — Wait for read request to finish
FlagMask	Bit mask to be enabled in the local flag group of the calling task on termination of the request (with RM_CONTINUE)
Handle	Descriptor
Length	Length of the memory area in bytes/blocks (numerical)
pBuffer	Pointer to the memory area
BlockAddress	Address of the first block for block–oriented drivers
pIOCount	Pointer to a `ulong` for the number of bytes/blocks read (valid only after completion of the read request)
pIOStatus	Pointer to `int` for error status of the operation or NULL pointer

Description

The `RmIORead` call reads *Length* bytes (for character–oriented drivers) or blocks (for block–oriented drivers) from the unit specified by *Handle* into the memory area specified by *pBuffer*. *Handle* is a descriptor that was generated with `RmIOOpen`.

With block–oriented drivers, the address of the first block to be read is also passed in *BlockAddress*. With character–oriented drivers (SER8250.DRV, 3964.DRV), *BlockAddress* is ignored.

Wait specifies whether the task is to wait for the read request to finish (RM_WAIT), or whether it is to continue (RM_CONTINUE).

The *FlagMask* parameter can be used to specify a bit mask in the local flag group (FlagGroupId=0) which will be enabled after termination of the request when a call without wait is executed. If 0 is specified, no bit mask is enabled.

After completion of the read request, the number of transferred bytes/blocks is stored in the `ulong` to which *pIOCount* points.

On termination of the read request, the status is entered in the `int` to which *pIOStatus* points. In requests with wait, this status is identical to the return value of the call. If the request is executed without wait, the value RM_IO_QUEUED is stored there while the request is located in the queue. During processing by the driver, the value RM_IO_IN_PROGRESS is stored there. After processing, the error status of the operation is stored there. If the return value of the status in *pIOStatus* is not required (e.g. because of a call with RM_WAIT), a NULL pointer can be passed. In this case, the status is only reported as the return value of the function.

Return Value

RM_OK The function was successfully executed

Error Codes

Error Code	Meaning
RM_BOUND_REACHED	Message queue of unit full
RM_EIO_INVALID_ACCESS	Descriptor is not open for read
RM_EIO_UNIT_RESET	Request canceled by control function RM_IOCTL_RESET
RM_INVALID_HANDLE	Descriptor is invalid
RM_INVALID_POINTER	Invalid pointer
RM_INVALID_TYPE	The value for *Wait* is invalid
RM_IO_IN_PROGRESS	Request is being processed
RM_IO_QUEUED	Request waiting in queue
RM_OUT_OF_MEMORY	Not enough free memory available in heap
RM_QUEUE_NOT_EXIST	Message queue of unit has not yet been set up.

See Also

RmIOClose, RmIOControl, RmIOOpen, RmIOWrite, RmLoadDevice

RmIOWrite

Function

Write to unit

Syntax

```
#include <rmapi.h>
int          RmIOWrite(
               uint Wait,
               uint FlagMask,
               RmIOHandle Handle,
               ulong Length,
               void *pBuffer,
               ulong BlockAddress,
               ulong *pIOCount,
               int *pIOStatus);
```

Parameters

Parameter Name	Meaning
Wait	Specifies whether the request is to be executed with or without waiting. RM_CONTINUE — Continue task without waiting for write request to finish RM_WAIT — Wait for write request to finish
FlagMask	Bit mask to be enabled in the local flag group of the calling task on termination of the request (with RM_CONTINUE)
Handle	Descriptor
Length	Length of the memory area in bytes/blocks (numerical)
pBuffer	Pointer to the memory area
BlockAddress	Address of the first block for block–oriented drivers
pIOCount	Pointer to a `ulong` for the number of bytes/blocks written (valid only after completion of the read request)
pIOStatus	Pointer to `int` for error status of the operation or NULL pointer

Description

The `RmIOWrite` call writes *Length* bytes (for character–oriented drivers) or blocks (for block–oriented drivers) from the memory area specified by *pBuffer* to the unit specified by *Handle*. *Handle* is a descriptor that was generated with `RmIOOpen`.

With block–oriented drivers, the address of the first block to be written is also passed in *BlockAddress*. With character–oriented drivers (SER8250.DRV, 3964.DRV), *BlockAddress* is ignored.

Wait specifies whether the task is to wait for the write request to finish (RM_WAIT), or whether it is to continue (RM_CONTINUE).

The *FlagMask* parameter can be used to specify a bit mask in the local flag group (FlagGroupId=0) which will be enabled after termination of the request when a call without wait is executed. If 0 is specified, no bit mask is enabled.

After completion of the read request, the number of transferred bytes/blocks is stored in the `ulong` to which *pIOCount* points.

On termination of the write request, the status is entered in the `int` to which *pIOStatus* points. In requests with wait, this status is identical to the return value of the call. If the request is executed without wait, the value RM_IO_QUEUED is stored there while the request is located in the queue. During processing by the driver, the value RM_IO_IN_PROGRESS is stored there. After processing, the error status of the operation is stored there. If the return value of the status in *pIOStatus* is not required (e.g. because of a call with RM_WAIT), a NULL pointer can be passed. In this case, the status is only reported as the return value of the function.

Return Value

RM_OK The function was successfully executed

Error Codes

Error Code	Meaning
RM_BOUND_REACHED	Message queue of unit full.
RM_EIO_INVALID_ACCESS	Descriptor not open for Write
RM_EIO_UNIT_RESET	Request canceled by control function RM_IOCTL_RESET
RM_INVALID_HANDLE	Descriptor is invalid
RM_INVALID_POINTER	Invalid pointer
RM_INVALID_TYPE	The value for *Wait* is invalid
RM_IO_IN_PROGRESS	Request is being processed
RM_IO_QUEUED	Request waiting in queue
RM_OUT_OF_MEMORY	Not enough free memory available in heap
RM_QUEUE_NOT_EXIST	Message queue of unit has not yet been set up.

See Also

RmIOClose, RmIOControl, RmIOOpen, RmIORead, RmLoadDevice

RmKillTask

Function

End task

Syntax

```
#include <rmapi.h>
int          RmKillTask(
               uint Mode,
               uint TaskID);
```

Parameters

Parameter Name	Meaning	
Mode	Desired task state:	
	RM_TASK_END	Switch task to DORMANT state (same effect as `RmEndTask`)
	RM_TASK_DELETE	Delete task (same effect as `RmDeleteTask`)
TaskID	ID of task to be deleted (RM_OWN_TASK = own task)	

Description

The function switches any task (even the calling task) to the DORMANT or NOTEXISTENT state, irrespective of the state before the function call.

Special conditions arise when the destination task is in the BLOCKED state.

`RmKillTask` is illegal under the following circumstances, and is terminated with an error message:

- Termination/deletion through `RmKillTask` was already requested (calling `RmKillTask` twice for the same task)
- Page fault because stack overflow

In the following situation, the task does not switch immediately to the DORMANT or NOTEXISTENT state, but is merely registered:

Waiting for completion of an I/O job:
The task involved remains in the BLOCKED state. The state change is not activated until the I/O job has been completed. It is thus possible that the task will remain visible in a passive (blocked) state following the call. In this case, the task is in the block state RM_STA_KEND or RM_STA_KDEL.

RM_TASK_DELETE option

All start requests are deleted from the queue. If the destination task was started with the coordination option “Wait until ready” or “Wait until termination”, all related tasks which have been initiated by `RmStartTask or RmQueueStartTask` are informed of the premature termination/deletion of the destination task.

RM_TASK_END option

All start requests remain in the queue. The calling task continues to run as if the destination task had initiated `RmEndTask`. If the destination task was

started with the coordination option “Wait until ready” or “Wait until termination”, all related tasks which have been initiated by `RmStartTask or RmQueueStartTask` are informed of the premature termination/deletion of the destination task.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_TYPE	An invalid parameter (*Mode*) was passed.
RM_INVALID_ID	An invalid *TaskID* was passed.
RM_INVALID_TASK_STATE	Call illegal in present task state.

Note

Resources, such as memory pools, mailboxes or semaphores, which are still in possession of the task, are not automatically freed when the task is switched to the DORMANT state or deleted. These resources must, if possible, be freed by another task, otherwise they will no longer be available during subsequent operation.

See Also

RmDeleteTask, x_cr_killtsk,

RmList

Function **List entries in resource catalog**

Syntax

```
#include <rmapi.h>
int          RmList (
               uint Type,
               uint Count,
               uint *pIndex,
               uint *pNumEntries
               RmEntryStruct *pEntry)
```

Parameters

Parameter Name	Meaning
Type	Resource type (see `RmGetName`)
Count	Number of resource entries to be read out in a call. *NumEntries* returns the number of entries which were found and stored in *pEntry.* If Count > 1, *pEntry* must point to an array with *NumEntries* elements of the **RmEntryStruct** structure.
pIndex	This parameter is used as both an input and output parameter. **Input parameter:** **pIndex* specifies the value from which the resource entries are to be read out. **pIndex* must be 0 on the first call. If further calls are required, **pIndex* should not be changed. **Output parameter:** In **pIndex* the function returns the next entry which has not yet been read out. This index is only used internally for system purposes and can not be evaluated by the user.
pNumEntries	Number of entries found.
pEntry	Pointer to a structure or (depending on *Count*) an array of structures of the type **RmEntryStruct,** see chapter 3:

Description

The `RmList` function reads out a number of entries from the catalog and stores them in the specified buffer whose start address is specified by *pEntry*.

The first entry to be read out can be specified in the **pIndex* parameter (start of the list = 0). When the call returns, **pIndex* contains a reference to the next entry which has not yet been read out.**pIndex* may not be changed.

The end of the catalog has been reached when the number of entries actually read out (**pNumEntries*) is less than the number requested (*Count*).

You can limit the read-out to a specific resource type with *Type*.

Return Value

RM_OK	Function successfully executed, the buffer contains valid entries.

Error Codes

Error Code	Meaning
RM_INVALID_TYPE	The specified type is illegal. 0≤*Type*≤11
RM_INVALID_POINTER	The pointer to the string is incorrect, or a protection error has occurred.

See Also

RmCatalog, RmGetEntry, RmGetName, RmUncatalog

RmLoadDevice

Function **Load driver**

Syntax

```
#include <rmapi.h>
int          RmLoadDevice(
               const char *pDeviceName,
               const char *pArguments);
```

Parameters

Parameter Name	Meaning
pDeviceName	Pointer to the name of the driver
pArguments	Pointer to arguments (separated by spaces)

Description

`RmLoadDevice` loads and starts the driver specified by *pDeviceName* or generates a new unit for the driver specified by *pDeviceName* if *pDeviceName* is entered in the RMOS resource catalog as a loadable driver (SER8250, 3964).

The driver must be specified by an absolute path the first time it is loaded. The name of the driver must be specified (SER8250.DRV, 3964.DRV). The driver is cataloged after it is loaded.

The name entered in the resource catalog must be used in further calls (SER8250, 3964).

pArguments specifies the arguments for initializing the driver or unit. The individual arguments are separated by spaces. See `RmIOControl` with control function RM_IOCTL_INIT_ASCII for more detailed information.

The RMFCRIFB.LIB library is required when the application is linked.

Return Value

RM_OK Function successful

Error Codes

Error Code	Meaning
RM_INVALID_DEVICE	Invalid *pDeviceName* (e.g. catalog entry is not a loadable driver or driver not found).
RM_OUT_OF_MEMORY	No free memory available.
RM_EIO_INIT_FAILED	The driver has terminated due to an error, and has been removed from the system.

Example

Load driver SER8250 without arguments:
RmLoadDevice(”\\M7RMOS32\\ser8250.drv”, NULL);

Load driver 3964 with unit 3964_COM1 and initialization values:
RmLoadDevice(”\\M7RMOS32\\3964.drv”,
“3964_COM1 IRQ:4 BASE:0x3F8 MODE:19200–N–8–1 PROT:1–1”);

Create unit COM2 for already loaded driver SER8250 with initialization values:
RmLoadDevice(”SER8250”,
“COM2 IRQ:3 BASE:0x2F8 MODE:19200–N–8–1”);

See Also

RmIOClose, RmIOControl, RmIOOpen, RmIORead, RmIOWrite

RmMapMemory

Function

Address physical memory

Syntax

```
#include <rmapi.h>
int          RmMapMemory (
               ulong PhysAddress,
               ulong Length,
               void **pPointer );
```

Parameters

Parameter Name	Meaning
PhysAddress	Physical start address
Length	Length of the memory area to be mapped
pPointer	Address of a pointer variable in which the linear address of the newly initialized memory area is entered. Programs can use **pPointer* for direct access to the mapped address area. If the linear address, that is **pPointer*, is equal to NUL, the memory area could not be mapped.

Description

The `RmMapMemory` function maps a physical memory area (for example dual-port RAM or memory mapped I/O) onto a linear address space (start address: **pPointer*, length: *Length*). User programs can use the returned pointer **pPointer* to access the memory (access is READ/WRITE).

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_POINTER	A pointer was invalid.

RmMemPoolAlloc

Function **Allocate memory area from memory pool**

Syntax

```
#include <rmapi.h>
int          RmMemPoolAlloc (
               ulong TimeOutValue,
               uint Mode,
               uint PoolID,
               ulong Size,
               void ** ppMemory)
```

Parameters

Parameter Name	Meaning	
TimeOutValue	Maximum time to wait for execution	
	RM_CONTINUE	Continue task without waiting for memory allocation
	RM_WAIT	Wait for memory allocation
	0 ... RM_MAXTIME	Time interval in ms. The task waits until either the memory has been allocated or the time has expired.
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.	
	RM_HOUR(*hour*)	Wait for (*hour*) hours
	RM_MINUTE(*min*)	Wait for (*min*) minutes
	RM_SECOND(*sec*)	Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*)	Wait for (*ms*) milliseconds
Mode	Allocation method for memory:	
	RM_AUTOFREE	The memory is freed automatically with `RmFreeAll`. It is assigned to a specific task.
	RM_NOAUTOFREE	The memory is not freed automatically with `RmFreeAll`.
PoolID	ID of the memory pool from which the memory is requested.	
Size	Size of the memory area	
ppMemory	Address of pointer to a memory area.	

Description

The function allocates a memory area of size *Size* from the specified memory area. **ppMemory* contains a valid pointer to the allocated memory area.

Return Value

RM_OK Function successfully executed.

RM_TASK_WAITING Function had to wait for memeoy allocation

Error Codes

Error Code	Meaning
RM_INVALID_SIZE	*Size*=0 or *Size* greater than memory pool
RM_INVALID_ID	No memory pool exists for the specified ID
RM_OUT_OF_MEMORY	No memory area of the specified size is available
RM_GOT_TIMEOUT	A suitable memory area could not be allocated in the specified time

See Also

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmGetSize, RmReAlloc

RmPauseTask

Function **Pause for time interval**

Syntax

#include <rmapi.h>
int RmPauseTask(ulong *TimeValue***);**

Parameters

Parameter Name	Meaning	
TimeValue	Duration of the pause	
	0 ... RM_MAXTIME	Time interval in ms. .
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.	
	RM_HOUR(*hour*)	Wait for (*hour*) hours
	RM_MINUTE(*min*)	Wait for (*min*) minutes
	RM_SECOND(*sec*)	Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*)	Wait for (*ms*) milliseconds

Description

`RmPauseTask` causes a task to pause for a defined time interval. If *TimeValue*=0, the task pauses until the start of the next system scan cycle.

A task interrupted by `RmPauseTask` can be switched prematurely from the BLOCKED state to READY with `RmResumeTask`.

Return Value

RM_OK	Function successfully executed.
RM_TASK_RESUMED	Task was resumed with `RmResumeTask`.

See Also **RmRestartTask, RmResumeTask**

RmQueueStartTask

Function

Add task to queue. The task is started immediately it switches to the DORMANT state.

Syntax

```
#include <rmapi.h>
int          RmQueueStartTask(
               uint Wait,
               uint TaskID,
               uint Priority,
               uint RegVal1,
               uint RegVal2);
```

Parameters

Parameter Name	Meaning	
Wait	RM_NO_WAIT	Start destination task and continue task.
	RM_WAIT_READY	Wait until destination task is in READY state.
	RM_WAIT_END	Wait until destination task has finished.
TaskID	Destination task ID	
Priority	0..255	Set defined value
	RM_TCDPRI	Take priority from TCD
	RM_CURPRI	Use current priority of the calling task
	RM_MAXPRI	Set maximum (RM_TCDPRI, RM_CURPRI)
RegVal1	Parameter 1 (passed in EAX of destination task)	
RegVal2	Parameter 2 (passed in EBX of destination task)	

Description

`RmQueueStartTask` starts a task. The function requires the same parameters as `RmStartTask`.

This function differs from `RmStartTask` in that the start call is entered in an internal system queue, and is executed as soon as the task switches to the DORMANT state.

If the task to be started is already in the DORMANT state, the effect of `RmQueueStartTask` is identical to `RmStartTask`.

Return Value

RM_OK	Function successfully executed; the destination task switched from the DORMANT state to READY, or the start request was entered in the internal system queue.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *TaskID* was passed.
RM_TASK_KILLED	The destination task was switched to the DORMANT state or deleted before the READY state was attained or before it was terminated with `RmKillTask`.
RM_INVALID_TYPE	An invalid parameter (*Priority*)was passed.

See Also

RmEndTask, RmStartTask

RmReadMessage

Function **Read message from message queue**

Syntax

```
#include <rmapi.h>
int          RmReadMessage (
               ulong TimeOutValue,
               uint * pMessage,
               void **pMessageParam)
```

Parameters

Parameter Name	Meaning
TimeOutValue	Specifies how long to wait for the arrival of a message when the message queue is empty. RM_CONTINUE — Continue task without waiting for the message to arrive. RM_WAIT — Wait for the message to arrive. 0 ... RM_MAXTIME — Time interval in ms. The task waits until either the message has arrived or the time has expired. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) — Wait for (*hour*) hours RM_MINUTE(*min*) — Wait for (*min*) minutes RM_SECOND(*sec*) — Wait for (*sec*) seconds RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
pMessage	Address of a variable in which the message ID is stored.
pMessageParam	Address of a pointer to the message parameter.

Description

Fetches the message with the highest priority from the message queue of the calling task.

The memory locations for the message ID and a pointer to the message parameters must be allocated by the calling task.

`RmReadMessage` enters the message ID in **pMessage*, and enters the pointer to the actual message parameters in **pMessageParam*.

If no messages exist, the function waits for the *TimeOutValue*. If a message is not received during this period, the function is canceled with a timeout.

Return Value

RM_OK — Function successfully executed; a message was read out from the message queue. The **pMessage* parameter contains the message ID and **pMessageParam* contains a valid pointer to the transmitted message.

Error Codes

Error Code	Meaning
RM_GOT_TIMEOUT	A message was not received within the specified time.
RM_INVALID_POINTER	A pointer was invalid.
RM_NO_MESSAGE	The message does not contain a message (only if TimeOutValue = RM_CONTINUE)
RM_QUEUE_NOT_EXIST	The message queue does not exist.

See Also

RmCreateMessageQueue, RmDeleteMessageQueue, RmSendMessage

RmReAlloc

Function **Change the size of a memory area**

Syntax

```
#include <rmapi.h>
int          RmReAlloc (
               ulong TimeOutValue,
               uint Mode,
               ulong NewSize,
               void **ppMemory)
```

Parameters

Parameter Name	Meaning
TimeOutValue	Maximum time to wait for execution RM_CONTINUE — Continue task without waiting for memory allocation. RM_WAIT — Wait for memory allocation. 0 ... RM_MAXTIME — Time interval in ms. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) — Wait for (*hour*) hours RM_MINUTE(*min*) — Wait for (*min*) minutes RM_SECOND(*sec*) — Wait for (*sec*) seconds RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
Mode	Allocation method for memory: RM_AUTOFREE — The memory is freed automatically with `RmFreeAll`. It is assigned to a specific task. RM_NOAUTOFREE — The memory is not freed automatically with `RmFreeAll`.
NewSize	New size of the memory area.
ppMemory	Address of pointer to a memory area.

Description

The function increases or reduces the memory area specified by **ppMemory* without changing its contents. **ppMemory* contains a valid pointer to the modified memory area. This pointer does not have to match the passed pointer, because the memory area may have been moved in certain circumstances.

If the original memory area **ppMemory* was requested from a pool, the same pool is used for `RmReAlloc`.

Return Value

RM_OK Function successfully executed.

RM_TASK_WAITING Function had to wait for memeoy allocation

Error Codes

Error Code	Meaning
RM_INVALID_POINTER	A pointer was invalid.
RM_INVALID_SIZE	*Size*=0 or *Size* greater than heap/memory pool
RM_OUT_OF_MEMORY	No memory area of the specified size is available
RM_GOT_TIMEOUT	A suitable memory area could not be allocated in the specified time

See Also

RmAlloc, RmCreateMemPool, RmDeleteMemPool, RmFree, RmFreeAll, RmGetSize, RmMemPoolAlloc

RmReceiveMail

Function

Receive message from local mailbox

Syntax

```
#include <rmapi.h>
int          RmReceiveMail(
               ulong TimeOutValue,
               uint MailboxID,
               void *pMail);
```

Parameters

Parameter Name	Meaning
TimeOutValue	Maximum time to wait for execution
	RM_CONTINUE — Continue task without waiting for message to arrive.
	RM_WAIT — Wait for message to arrive.
	0 ... RM_MAXTIME — Time interval in ms. The task waits until either the message has arrived or the time has expired.
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.
	RM_HOUR(*hour*) — Wait for (*hour*) hours
	RM_MINUTE(*min*) — Wait for (*min*) minutes
	RM_SECOND(*sec*) — Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
Mode	Allocation method for memory:
	RM_AUTOFREE — The memory is freed automatically with `RmFreeAll`. It is assigned to a specific task.
	RM_NOAUTOFREE — The memory is not freed automatically with `RmFreeAll`.
MailboxID	Mailbox ID
pMail	Pointer to 12-byte buffer

Description

`RmReceiveMail` copies the 3-word message with the highest priority from a mailbox to a user buffer, and deletes the message from the mailbox.

A user buffer with a capacity of 3 words must be allocated by the calling task.

Return Value

RM_OK — Contents of **pMail* contain message.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Mailbox ID invalid.
RM_INVALID_POINTER	A pointer was invalid.
RM_NO_MESSAGE	The mailbox does not contain a message (only if TimeOutValue = RM_CONTINUE).
RM_GOT_TIMEOUT	The call was canceled after the configured timeout time.

Note

A 3-word message normally contains either the actual message or a pointer to the actual message block. In the latter case, the sender task fetches the message block for the actual information from a memory pool, and the task which reads the message from the mailbox returns it to the memory pool. The word length is 32 bits.

See Also

RmCreateMailbox, RmDeleteMailbox, RmSendMail, RmSendMailCancel, RmSendMailDelayed

RmReleaseBinSemaphore

Function **Reset semaphore**

Syntax

#include <rmapi.h>
int **RmReleaseBinSemaphore(uint** *SemaphoreID***);**

Parameters

Parameter Name	Meaning
SemaphoreID	Semaphore ID

Description `RmReleaseBinSemaphore` resets the *SemaphoreID* semaphore.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *SemphoreID* was passed.

Note The allocation and release of semaphores are not task-specific.

See Also **RmCreateBinSemaphore, RmDeleteBinSemaphore, RmGetBinSemaphore**, automatic priority change through semaphore possession in the Programming Manual

RmResetFlag

Function

Reset event flag

Syntax

```
#include <rmapi.h>
int          RmResetFlag(
               uint FlagGrpID,
               uint FlagMask);
```

Parameters

Parameter Name	Meaning
FlagGrpID	Event flag group ID. 0 specifies the local flag group.
FlagMask	The mask defines which bits are reset.

Description

`RmResetFlag` resets the event flags specified in the flag mask, and indicates whether they were already set.

Return Value

RM_OK	Function successful no bits reset.
RM_FLAG_RESET	At least one bit was reset.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *FlagGrpID* was passed.

See Also

RmCreateFlag, RmDeleteFlag, RmGetFlag, RmSetFlag, RmSetFlagDelayed

RmRestartTask

Function **Terminate task and restart after time interval**

Syntax

```
#include <rmapi.h>
int          RmRestartTask(
               uint Mode,
               ulong TimeValue);
```

Parameters

Parameter Name	Meaning	
Mode	RM_LAST_READY_TIME	Refer time calculation to last change to READY state
	RM_CURRENT_TIME	Refer time calculation to current time
TimeValue	Wait time until restart	
	0... RM_MAXTIME	Time interval in ms.
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.	
	RM_HOUR(*hour*)	Wait for (*hour*) hours
	RM_MINUTE(*min*)	Wait for (*min*) minutes
	RM_SECOND(*sec*)	Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*)	Wait (*ms*) Sekunden

Description

`RmRestartTask` terminates execution of the task and restarts it when a time interval has expired.

If *TimeValue*=0, the task is switched to the READY state on the next timer interrupt.

Return Value

RM_OK Function successfully executed.

Note

`RmRestartTask` switches a task to the BLOCKED state and not to the DORMANT state. In contrast to `RmPauseTask`, the task is started when the time defined in `RmRestartTask` expires; that is program execution begins at the entry address of the task.

A task interrupted by `RmRestartTask` can only be switched to the READY state once the time interval has expired.

It is not possible to pass parameters in EAX or EBX to a task on restart with `RmRestartTask`.The parameters can be passed and stored the first time the task is started (with another start command). These parameters can then be reused following any subsequent task start initiated by `RmRestartTask`.

If a task (`main()`) was started by the CLI, it may not be restarted with `RmRestartTask`.

See Also

RmActivateTask, RmPauseTask, RmResumeTask, starting, interruption, termination of tasks

RmResumeTask

Function **Resume task halted by** `RmPauseTask` **or** `RmSuspendTask`**.**

Syntax

#include <rmapi.h>
int **RmResumeTask(uint** *TaskID***);**

Parameters

Parameter Name	Meaning
TaskID	Task ID

Description

`RmResumeTask` switches a task, which has been changed to the BLOCKED state by `RmSuspendTask` or `RmPauseTask` call.

In contrast to `RmRestartTask`, program execution resumes immediately after the `RmSuspendTask` or `RmPauseTask` call.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *TaskID* was passed.
RM_TASK_NOT_PAUSED	Task to be resumed by `RmResumeTask` was not halted by `RmPauseTask` or is no longer in the BLOCKED state.

See Also **RmActivateTask, RmPauseTask, RmRestartTask, RmSuspendTask**

RmSendMail

Function **Send message to a mailbox**

Syntax

```
#include <rmapi.h>
int          RmSendMail(
               ulong TimeOutValue,
               uint Priority,
               uint MailboxID,
               void *pMail);
```

Parameters

Parameter Name	Meaning	
TimeOutValue	Maximum time to wait for execution	
	RM_CONTINUE	Continue task without waiting for message to be fetched.
	RM_WAIT	Wait for message to be fetched.
	0 ... RM_MAXTIME	Time interval in ms. The task waits until either the message has been fetched or the time has expired.
	The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds.	
	RM_HOUR(*hour*)	Wait for (*hour*) hours
	RM_MINUTE(*min*)	Wait for (*min*) minutes
	RM_SECOND(*sec*)	Wait for (*sec*) seconds
	RM_MILLISECOND(*ms*)	Wait (*ms*) Sekunden
Priority	0..255	Set defined value
	RM_TCDPRI	Take priority from TCD
	RM_CURPRI	Use current priority of the calling task
MailboxID	Mailbox ID	
pMail	Pointer to 3-word buffer	

Description

`RmSendMail` copies a 3-word-long prioritized message to a mailbox. The task can be switched to the BLOCKED state until the message has been fetched or the call has been canceled by a timeout.

The message format is freely selectable. For example, a 3-word (32-bit) long message or the address and length of a message with the following configuration can be specified:

Message word 1: Address of the message block
Message word 2: Anything
Message word 3: Length of the message block in byte

Return Value

RM_OK	Function successfully executed, the message was copied to the mailbox.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *MailboxID* was passed.
RM_INVALID_TYPE	An invalid parameter (*Priority*) was passed.
RM_INVALID_POINTER	A pointer was invalid.
RM_GOT_TIMEOUT	The call was canceled after the configured timeout.
RM_BOUND_REACHED	The request exceeds the limit entered for the mailbox (see `RmSetMailboxSize`).

See Also

RmCreateMailbox, RmDeleteMailbox, RmReceiveMail, RmSetMailbox-Size

RmSendMailCancel

Function **Cancel message started with** `RmSendMailDelayed`**.**

Syntax

```
#include <rmapi.h>
int          RmSendMailCancel (
               RmMailIDStruct *pMailID,
               void *pMail);
```

Parameters

Parameter Name	Meaning
pMailID	Pointer to a structure of the type **RmMailIDStruct** (see chapter 3). The `RmSendMailDelayed` function returns the pointer to the accompanying **RmMailIDStruct**.
pMail	Pointer to a buffer to which the previously dispatched message is written back. The length of the message is 12 bytes.

Description

The function cancels a message started with `RmSendMailDelayed`. It is only possible to cancel the message before the time interval has expired or the specified message has been fetched. In the latter case, the message is deleted from the mailbox.

The preceding `RmSendMailDelayed` call returns information in an **RmMailIDStruct** structure. The address of this structure must be passed with the `RmSendMailCancel`call.

The contents of the message are returned to the calling task, so that the information in the message can be evaluated if necessary.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid message was passed in *pMailID*.This error is also output if a dispatched message has already been fetched. The memory defined by *pMail* is undefined.
RM_INVALID_POINTER	A pointer was invalid.

See Also

RmCreateMailbox, RmDeleteMailbox, RmReceiveMail, RmSendMail, RmSendMailDelayed, RmSetMailboxSize

RmSendMailDelayed

Function

Send mail to a mailbox after a delay

Syntax

```
#include <rmapi.h>
int          RmSendMailDelayed (
               ulong TimeValue,
               uint Priority,
               uint MailboxID,
               void *pMail,
               RmMailIDStruct *pMailID);
```

Parameters

Parameter Name	Meaning
TimeValue	Time until message is sent. 0 ... RM_MAXTIME Time interval in ms. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) Wait for (*hour*) hours RM_MINUTE(*min*) Wait for (*min*) minutes RM_SECOND(*sec*) Wait for (*sec*) seconds RM_MILLISECOND(*ms*) Wartet (*ms*) Sekunden
Priority	0..255 Set defined value RM_TCDPRI Take priority from TCD RM_CURPRI Use current priority of the calling task
MailboxID	Mailbox ID
pMail	Pointer to message. The length of the message is 12 bytes.
pMailID	Pointer to a structure of the type **RmMailIDStruct** (see chapter 3).

Description

`RmSendMailDelayed` sends mail to a mailbox after a delay. The calling task must pass the address of a memory area of the type **RmMailIDStruct.**

The function enters an identification code in this memory area. The identification code can be used to cancel the action with `RmSendMailCancel`.

Return Value

RM_OK	Function successfully executed, the **RmMailIDStruct** variable contains the identification of the accompanying job.

Error Codes

Error Code	Meaning
RM_INVALID_TYPE	An invalid parameter (*Priority*) was passed.
RM_INVALID_ID	Invalid flag group.
RM_INVALID_POINTER	A pointer was invalid.

Note

A limit, defined by `RmSetMailboxSize`, that restricts the number of messages waiting to be fetched from a mailbox, is ignored when the message is dispatched with `RmSendMailDelayed`.

It is possible for the mailbox to which the message is dispatched to be deleted by the system call `RmDeleteMailbox` before the time interval has expired. In this case, the message is discarded without an error being indicated.

See Also

RmCreateMailbox, RmDeleteMailbox, RmReceiveMail, RmSendMail, RmSendMailCancel, RmSetMailboxSize

RmSendMessage

Function

Add message to message queue

Syntax

```
#include <rmapi.h>
int          RmSendMessage (
               ulong TimeOutValue,
               uint Priority,
               uint TaskID,
               uint Message,
               void *pMessageParam)
```

Parameters

Parameter Name	Meaning
TimeOutValue	Specifies how long to wait for message to be fetched. RM_CONTINUE — Continue task without waiting for the message to be fetched. RM_WAIT — Wait for the message to be fetched. 0 ... RM_MAXTIME — Time interval in ms. The task waits until either the message has been fetched or the time has expired. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) — Wait for (*hour*) hours RM_MINUTE(*min*) — Wait for (*min*) minutes RM_SECOND(*sec*) — Wait for (*sec*) seconds RM_MILLISECOND(*ms*) — Wait for (*ms*) milliseconds
Priority	0..255 — Set defined value RM_TCDPRI — Take priority from TCD RM_CURPRI — Use current priority of the calling task
TaskID	Destination task ID
Message	Message identifier The message identifiers are defined as follows: RM_MSG_USER..RM_MSG_MAX reserved for the user
pMessageParam	Pointer to the contents of the message.

Description

The call inserts *Message*, together with the pointer to the message parameters and with the defined priority, at the appropriate point in the message queue of the task specified by *TaskID*. The *TimeOutValue* parameter specifies whether the task is to wait for the message to be fetched and, if so, how long.

Note

When calling ***RmSendMessage*** with ***TimeOutValue=RM_WAIT***, the following may occur:

If the task is woken up (e.g. with ***RmActivateTask***) while ***RmSendMessage*** is waiting for the message to be fetched, ***RmSendMessage*** returns success although it is not sure wether the message has been fetched or not.

Return Value

RM_OK Function successfully executed, the message was copied to the task's own message queue.

Error Codes

Error Code	Meaning
RM_GOT_TIMEOUT	The message was not fetched within the specified period.
RM_INVALID_ID	Task ID invalid
RM_INVALID_POINTER	Invalid pointer
RM_INVALID_TYPE	An invalid parameter (*Priority*) was passed.
RM_QUEUE_NOT_EXIST	The message queue does not exist.
RM_BOUND_REACHED	The message queue is full.

See Also

RmCreateMessageQueue, RmDeleteMessageQueue, RmReadMessage,

RmSetFlag

Function **Set event flag**

Syntax

```
#include <rmapi.h>
int          RmSetFlag(
               uint FlagGrpID,
               uint FlagMask);
```

Parameters

Parameter Name	Meaning
FlagGrpID	Flag group ID. 0 specifies the local flag group.
FlagMask	The mask specifies which bits are set

Description `RmSetFlag` sets the event flags specified in the flag mask, and indicates whether they were already set.

Return Value

RM_OK	Function successful, no bits set.
RM_FLAG_SET	At least one bit was set.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *FlagGrpID* was passed.

See Also **RmCreateFlagGrp, RmDeleteFlagGrp, RmGetFlag, RmResetFlag**

RmSetFlagDelayed

Function **Set event flag after interval**

Syntax

```
#include <rmapi.h>
int     RmSetFlagDelayed(
          ulong TimeValue,
          uint FlagGrpID,
          uint FlagMask);
```

Parameters

Parameter Name	Meaning
TimeValue	Delay time until flag is set 0 ... RM_MAXTIME Time interval in ms. The values for hours, minutes and seconds can be combined by addition for the time parameter. The maximum wait time is 2^31 milliseconds. RM_HOUR(*hour*) Wait for (*hour*) hours RM_MINUTE(*min*) Wait for (*min*) minutes RM_SECOND(*sec*) Wait for (*sec*) seconds RM_MILLISECOND(*ms*) Wait for (*ms*) milliseconds
FlagGrpID	Flag group ID. 0 specifies the local flag group.
FlagMask	The mask defines which bits are manipulated.

Description

`RmSetFlagDelayed` clears the bits specified by *FlagMask* , and sets them when the time interval has expired. Bits which are not set and bits specified by*FlagMask* with the same *FlagGrpID* are checked. The timer values of these bits are set to the new value if necessary.

A second `RmSetFlagDelayed` function with an identical *FlagGrpID* and *FlagMask* overwrites the first `RmSetFlagDelayed` if the time parameter is positive and deletes it if the time parameter = 0.

An `RmResetFlag` has no effect on `RmSetFlagDelayed`.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *FlagGrpID* was passed.
RM_PARAMETER_ERROR	Incorrect parameters were passed to the function (*FlagMask*=0).

See Also **RmCreateFlagGrp, RmDeleteFlagGrp, RmGetFlag, RmResetFlag, RmSetFlag**

RmSetIntDefHandler

Function **Install default interrupt handler**

Syntax

#include <rmapi.h>

int **RmSetIntDefHandler (uint** *IntNum***);**

Parameters

Parameter Name	Meaning
IntNum	SW-Interrupt Number (0–255) IRQx (x=0 to 63) Hardware interrupt IRQ(n) (n=0 to 63) Hardware interrupt The hardware interrupts on M7-300/400 are at 0 to 15.

Description

This function is used to deinstall a dedicated interrupt handler for the specified interrupt ***IntNum***, and reallocate the default interrupt handler to this interrupt.

The interrupt number indexes the entries in the interrupt descriptor table, that is the interrupt number corresponds to the selector of the associated descriptor. The entry address of the associated interrupt handler is entered in the descriptor.

Return Value

RM_OK Function successfully executed, the dedicated interrupt handler was deinstalled.

Error Codes

Error Code	Meaning
RM_INVALID_INTERRUPT_NUMBER	Invalid interrupt number
RM_INVALID_IRQ_NUMBER	IRQx invalid, PIC not defined

See Also

RmGetIntHandler, RmSetIntISHandler, RmSetIntMailboxHandler, RmSetIntTaskHandler

RmSetIntISHandler

Function **Initialize S or I interrupt handler**

Syntax

```
#include <rmapi.h>
int          RmSetIntISHandler (
               uint IntNum,
               rmfarproc IHandlerEntry,
               rmfarproc SHandlerEntry);
```

Parameters

Parameter Name	Meaning
IntNum	SW-Interrupt Number (0–255) IRQx (x=0 to 63) Hardware interrupt IRQ(n) (n=0 to 63) Hardware interrupt The hardware interrupts on M7-300/400 are at 0 to 15.
IHandlerEntry	Entry address of the I interrupt handler
SHandlerEntry	Entry address of the S interrupt handler

Description

The call defines an I and/or S interrupt handler.

If the interrupt is a hardware interrupt, such as IRQ1, this is masked automatically.

While a new interrupt handler is being initialized, an interrupt must not occur for this handler.

The interrupt handler specified in *IHandlerEntry* or *SHandlerEntry* is activated in I or S state immediately after an interrupt. If a handler is not to be installed, NUL should be specified.

The *SHandlerEntry* is only called if the return value of the I state ≠ 0. If the return value is equal to 0, a transition to the S state does not occur.

The interrupt number indexes the entries in the interrupt descriptor table, that is the interrupt number corresponds to the selector of the associated descriptor. The entry address of the associated interrupt handler is entered in the descriptor.

`RmSetIntISHandler` enters an interrupt gate in the IDT.

The header and trailer are generated by the operating system kernel. The handlers can be simple procedures. The memory required for an interrupt handler is approximately 130 bytes, and is allocated from the heap.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_OUT_OF_MEMORY	Insufficient memory available
RM_INVALID_INTERRUPT_NUMBER	Invalid interrupt number
RM_INVALID_IRQ_NUMBER	IRQx invalid, PIC not defined
RM_INVALID_POINTER	Invalid pointer

Note

If the function call is not successfully executed, the previous interrupt handler remains active.

A user program runs on the M7 system at the ”user level”. Write access is possible only for the user data whereas code and system areas are write–protected for a user task.

An I handler or S handler is executed at ”system level”, i.e. memory protection is removed within an interrupt handler.

See Also

RmGetIntHandler, RmSetIntDefHandler, RmSetIntMailboxHandler, RmSetIntTaskHandler

RmSetIntMailboxHandler

Function **Initialize mailbox interrupt handler**

Syntax

```
#include <rmapi.h>
int     RmSetIntMailboxHandler (
          uint IntNum,
          uint MailboxID,
          uint MailPriority);
```

Parameters

Parameter Name	Meaning
IntNum	SW-Interrupt Number (0–255) IRQx (x=0 to 63) Hardware interrupt IRQ(n) (n=0 to 63) Hardware interrupt The hardware interrupts on M7-300/400 are at 0 to 15.
MailboxID	Mailbox ID A message is sent to the mailbox specified by *MailBoxID*. If this mailbox is limited to an `RmSetMailboxSize`, that is if only a certain number of messages can wait to be fetched in the mailbox, and if this number has already been reached, no message is sent. In this case, the interrupt is lost. The **RmIntrhandMailStruct** structure is described in chapter 3.
MailPriority	Priority of the message

Description

The call defines a handler for sending a message.

If the interrupt is a hardware interrupt, such as IRQ1, this is masked automatically.

While a new interrupt handler is being initialized, an interrupt must not occur for this handler.

If the number of messages in a mailbox is limited (see `RmSetMailbox-Size`), no messages are sent when this limit is reached. The interrupt is lost.

`RmSetIntMailboxHandler` enters an interrupt gate in the IDT. Existing entries in the IDT are retained, but can be overwritten by the call.

The code for the interrupt handler for dispatching the message is generated by the operating system kernel. The memory required for an interrupt handler is approximately 130 bytes, and is allocated from the heap.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_OUT_OF_MEMORY	Insufficient memory available
RM_INVALID_INTERRUPT_NUMBER	Invalid interrupt number
RM_INVALID_IRQ_NUMBER	IRQx invalid, PIC not defined
RM_INVALID_ID	Invalid mailbox ID

Note

If the function call is not successfully executed, the previous interrupt handler remains active.

See Also

RmGetIntHandler, RmSetIntDefHandler, RmSetIntISHandler, RmSetIntTaskHandler

RmSetIntTaskHandler

Function **Initialize interrupt handler for task start**

Syntax

```
#include <rmapi.h>
int          RmSetIntTaskHandler (
               uint IntNum,
               uint TaskID);
```

Parameters

Parameter Name	Meaning
IntNum	SW-Interrupt Number (0–255) IRQx (x=0 to 63) Hardware interrupt IRQ(n) (n=0 to 63) Hardware interrupt The hardware interrupts on M7-300/400 are at 0 to 15.
TaskID	Task ID (RM_OWN_TASK = ID of the calling task)

Description

The call defines a handler for an interrupt-driven task start.

If the interrupt is a hardware interrupt, such as IRQ1, this is masked automatically.

While a new interrupt handler is being initialized, an interrupt must not occur for this handler.

The task specified in *TaskID* is activated immediately after an interrupt.

The interrupt number corresponds to the selector of the associated descriptor in the IDT.

`RmSetIntTaskHandler` enters an interrupt gate in the IDT.

The code for the interrupt handler for starting the task is generated by the operating system kernel. The memory required for an interrupt handler is approximately 130 bytes, and is allocated from the heap.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_OUT_OF_MEMORY	Insufficient memory available
RM_INVALID_INTERRUPT_NUMBER	Invalid interrupt number
RM_INVALID_IRQ_NUMBER	IRQx invalid, PIC not defined
RM_INVALID_ID	Invalid task ID

Note

If the function call is not successfully executed, the previous interrupt handler remains active.

See Also

RmGetIntHandler, RmSetIntDefHandler, RmSetIntISHandler, RmSetIntMailboxHandler

RmSetMailboxSize

Function **Define limit values for mailboxes**

Syntax

```
#include <rmapi.h>
int          RmSetMailboxSize (
               uint MailboxID
               uint Limit);
```

Parameters

Parameter Name	Meaning	
MailboxID	Mailbox ID	
Limit	1-0FFFFH	Maximum number of messages in queue
	0	Indicates that the limit is to be canceled

Description

The function sets a limit for the number of messages which can wait in a mailbox. The limit value can be modified as required and can be subsequently canceled.

When the limit is exceeded, all subsequent attempts to send a message to this mailbox with the `RmSendMail` call are rejected. `RmSendMail` calls are not accepted again until enough messages are fetched for the number of messages to fall below the limit again.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Mailbox ID invalid

Note

The limit set for mailboxes has no effect during the `RmSendMailDelayed` system call.

See Also

RmReceiveMail, RmSendMail, RmSendMailCancel, RmSendMailDelayed

RmSetMessageQueueSize

Function **Define length of message queue**

Syntax

```
#include <rmapi.h>
int          RmSetMessageQueueSize (
               uint TaskID,
               uint Limit)
```

Parameters

Parameter Name	Meaning
TaskID	Destination task ID
Limit	Number of free places in the message queue

Description The call defines the size of the message queue of the task specified in *TaskID*.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	Task ID invalid
RM_INVALID_TYPE	An invalid parameter (*Limit*) was passed.
RM_QUEUE_NOT_EXIST	The message queue does not exist.

See Also **RmCreateMessageQueue, RmDeleteMessageQueue, RmReadMessage, RmSendMessage**

RmSetTaskPriority

Function **Change task priority**

Syntax

```
#include <rmapi.h>
int          RmSetTaskPriority(
               uint TaskID,
               uint Priority);
```

Parameters

Parameter Name	Meaning	
TaskID	Destination task ID (RM_OWN_TASK= own)	
Priority	0..255	Set defined value
	RM_TCDPRI	Take priority from TCD
	RM_CURPRI	Use current priority of the calling task
	RM_INCPRI	Increase task priority by 1
	RM_DECPRI	Decrease task priority by 1

Description `RmSetTaskPriority` is used to change the priority of any task.

Return Value RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	*TaskID* invalid
RM_INVALID_TYPE	An invalid parameter (*Priority*) was passed.
RM_PRI_NOT_CHANGED	*Priority* has not been changed.
RM_TASK_DORMANT	Task currently in DORMANT state

See Also **RmStartTask, RmQueueStartTask**, description of the task priorities in the Programming Manual

RmStartTask

Function **Start request for tasks in DORMANT state**

Syntax

```
#include <rmapi.h>
int RmStartTask(
  uint Wait,
  uint TaskID,
  uint Priority,
  uint RegVal1,
  uint RegVal2);
```

Parameters

Parameter Name	Meaning	
Wait	RM_NO_WAIT	Start and continue destination task.
	RM_WAIT_READY	Wait until destination task is in READY state.
	RM_WAIT_END	Wait until destination task has finished.
TaskID	Destination task ID (RM_OWN_TASK = own task).	
Priority	0..255	Set defined value
	RM_TCDPRI	Take priority from TCD
	RM_CURPRI	Use current priority of the calling task
	RM_MAXPRI	Set maximum (RM_TCDPRI, RM_CURPRI)
RegVal1	Parameter 1 (passed in eax of destination task)	
RegVal2	Parameter 2 (passed in ebx of destination task)	

Description

`RmStartTask` starts a task. The function requires the same parameters as `RmQueueStartTask`.
The difference between this function and `RmQueueStartTask` is that `RmQueueStartTask` enters the start request in a queue if the task is not in the DORMANT state. The `RmStartTask` call has no effect in this case, however.

Return Value

RM_OK	Function successfully executed; the destination task switched to the READY state.

Error Codes

Error Code	Meaning
RM_INVALID_ID	An invalid *TaskID* was passed.
RM_INVALID_TYPE	An invalid parameter (*Wait*) was passed.

Error Code	Meaning
RM_TASK_NOT_DORMANT	An attempt was made to start a task which was not in the DORMANT state.
RM_TASK_KILLED	The destination task was switched to the DORMANT state or deleted before the READY state was attained or before it was terminated with `RmKillTask`.

See Also **RmQueueStartTask**

RmSuspendTask

Function **Set task from READY to BLOCKED state**

Syntax

#include <rmapi.h>

int **RmSuspendTask(uint** *TaskID***);**

Parameters

Parameter Name	Meaning
TaskID	Task ID

Description

`RmSuspendTask` suspends the task specified by *TaskID*. The suspended task must be in the READY state, and is subsequently switched to the BLOCKED state. A task can suspend itself.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_INVALID_ID	TaskID invalid
RM_TASK_NOT_READY	Task was not in READY state

See Also **RmResumeTask**

RmUncatalog

Function **Delete resources from catalog**

Syntax

#include <rmapi.h>

int **RmUncatalog (char** **pName***)**

Parameters

Parameter Name	Meaning
pName	Pointer to a character string (the string can be defined in C or PLM notation).

Description

`RmUncatalog` deletes the resource identified by a character string from the catalog.

Return Value

RM_OK Function successfully executed.

Error Codes

Error Code	Meaning
RM_IS_NOT_CATALOGED	Entry not found
RM_INVALID_POINTER	*pName* pointer was invalid
RM_INVALID_STRING	String length = 0 or > 15

Note

If a resource with various strings is cataloged more than once, all entries for this resource are deleted from the catalog.

See Also

RmCatalog, RmGetEntry, RmGetName, RmList

SerialCheckChar

Function **Read in single character from unit**

Syntax

```
#include <serial.h>
int          SerialCheckChar(
               RmIOHandle Handle,
               char *Char );
```

Parameters

Parameter Name	Meaning
Handle	Descriptor
Char	Address of a `char` where the read character is stored

Description

`SerialCheckChar` reads a single character from the unit specified by *Handle* and stores it at the address specified by *Char. Handle* is a descriptor that was generated with `SerialOpen`.

Unlike the `SerialGetChar` call, `SerialCheckChar` does not wait for the character to arrive. If there is no character in the background buffer of the unit, `SerialCheckChar` terminates.

Return Value

RM_OK The function was successfully executed

Error Codes

Error Code	Meaning
RM_IO_NO_DATA	No data exist

See “Error Codes for Loadable Drivers” for further error messages

Note

This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also

SerialCheckString, SerialGetChar, SerialGetString, SerialOpen

SerialCheckString

Function **Read string from unit**

Syntax

```
#include <serial.h>
int          SerialCheckString(
               RmIOHandle Handle,
               ulong MaxLen,
               char *String,
               ulong *Count );
```

Parameters

Parameter Name	Meaning
Handle	Descriptor
MaxLen	Maximum number of characters to be read
String	Address of memory area where the read characters are stored
Count	Address of a `ulong` in which the number of characters read is stored. Value > 0 Number of characters read Value = 0 Error or no characters exist

Description

`SerialCheckString` reads *MaxLen* characters from the unit specified by *Handle* and stores them at the address specified by *String*. *Handle* is a descriptor that was generated with `SerialOpen`.

If the read request is successful, *Count* contains the number of characters read. If the read request was not successful or no characters were found, the parameter contains the value 0.

Unlike `SerialGetString`, `SerialCheckString` does not wait for the character to arrive. If there is no character in the background buffer of the unit, `SerialCheckString` terminates.

Return Value

RM_OK The function was successfully executed

Error Codes

Error Code	Meaning
RM_IO_NO_DATA	No data exist

See “Error Codes for Loadable Drivers” for further error messages

Note

This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also

SerialCheckChar, SerialGetChar, SerialGetString, SerialOpen

SerialClose

Function **Close a connection to a unit of a driver**

Syntax

#include <serial.h>

int SerialClose(RmIOHandle *Handle***);**

Parameters

Parameter Name	Meaning
Handle	Descriptor

Description

`SerialClose` closes the connection specified by *Handle*. *Handle* is a descriptor that was generated with `SerialOpen`.

Return Value

RM_OK The function was successfully executed

Error Codes

See “Error Codes for Loadable Drivers”

Note

This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also

SerialOpen

SerialGetChar

Function **Read in single character from unit**

Syntax

```
#include <serial.h>
int          SerialGetChar(
               RmIOHandle Handle,
               char *Char);
```

Parameters

Parameter Name	Meaning
Handle	Descriptor
Char	Address of a `char` where the read character is stored

Description `SerialGetChar` reads a single character from the unit specified by *Handle* and stores it at the address specified by *Char*. *Handle* is a descriptor that was generated with `SerialOpen`. The call waits for the character to arrive.

Return Value

RM_OK	The function was successfully executed

Error Codes See “Error Codes for Loadable Drivers”

Note This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also **SerialCheckChar, SerialCheckString, SerialGetString, SerialOpen**

SerialGetString

Function **Read string from unit**

Syntax

```
#include <serial.h>
int          SerialGetString(
               RmIOHandle Handle,
               ulong MaxLen,
               char *String,
               ulong *Count );
```

Parameters

Parameter Name	Meaning
Handle	Descriptor
MaxLen	Maximum number of characters to be read
String	Address of a memory area where the read characters are stored
Count	Address of a `ulong` in which the number of characters read is stored. Value > 0 Number of characters read Value = 0 Error or no characters exist

Description

`SerialGetString` reads a maximum of *MaxLen* characters from the unit specified by *Handle* and stores them at the address specified by *String*. *Handle* is a descriptor that was generated with `SerialOpen`.

If the read request is successful, *Count* contains the number of characters read. If the read request was not successful or no characters were found, the parameter contains the value 0.

The call waits for the characters to arrive.

Return Value

RM_OK The function was successfully executed

Error Codes

See “Error Codes for Loadable Drivers”

Note

This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also

SerialCheckChar, SerialCheckString, SerialGetChar, SerialOpen

SerialInit

Function **Initialize unit**

Syntax

```
#include <serial.h>
int SerialInit(
  RmIOHandle Handle,
  ulong Baud,
  uint Data,
  uint Parity,
  uint Stop);
```

Parameters

Parameter Name	Meaning	
Handle	Descriptor	
Baud	Baud rate as numeric value (e.g. 19200)	
Data	Number of data bits as numeric value (e.g. 8)	
Parity	Parity	
	SERIAL_PARITYNONE	No parity check
	SERIAL_PARITYEVEN	Even parity
	SERIAL_PARITYODD	Odd parity
	SERIAL_PARITY0	Parity bit always 0
	SERIAL_PARITY1	Parity bit always 1
Stop	Number of stop bits. The following are permitted:	
	SERIAL_STOP1	1 stop bit
	SERIAL_STOP2	2 stop bits
	SERIAL_STOP15	1.5 stop bits

Description `SerialInit` is used to initialize the unit of a driver for a serial interface. The unit is specified by *Handle*. *Handle* is a descriptor that was generated with `SerialOpen`. The *Baud* parameter specifies the baud rate. The parameters *Data* and *Stop* specify the number of data and stop bits. The *Parity* parameter is used to control the parity.

Return Value RM_OK The function was successfully executed

Error Codes See “Error Codes for Loadable Drivers”

Note This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also **SerialClose, SerialInitEx, SerialOpen**

SerialInitEx

Function **Extended initialization of 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);
```

Parameters

Parameter Name	Meaning	
Handle	Descriptor	
Baud	Baud rate as numeric value (e.g. 19200)	
Data	Number of data bits as numeric value (e.g. 8)	
Parity	Parity	
	SERIAL_PARITYNONE	No parity check
	SERIAL_PARITYEVEN	Even parity
	SERIAL_PARITYODD	Odd parity
	SERIAL_PARITY0	Parity bit always 0
	SERIAL_PARITY1	Parity bit always 1
Stop	Number of stop bits. The following are permitted:	
	SERIAL_STOP1	1 stop bit
	SERIAL_STOP2	2 stop bits
	SERIAL_STOP15	1.5 stop bits
BufferSize	Size of background buffer (number of characters)	

Parameter Name	Meaning
SendStopMode	Specifies which stop character is to terminate write requests. The stop character(s) is (are) transferred after the user data. SERIAL_SENDSTOP_OFF Do not use stop character. SERIAL_SENDSTOP_CHAR_1 Use stop character 1 SERIAL_SENDSTOP_CHAR_1_2 Use stop characters 1 and 2, that is cancel when the 1st stop character is followed by the 2nd stop character.
SendStop1	1st stop character for write requests
SendStop2	2nd stop character for write requests
SendDelay	Minimum pause between two write requests (in ms). Specifying 0 deactivates the function
RecStopMode	Specifies which stop character is to terminate read requests. The stop character(s) is (are) not transferred to the user buffer. SERIAL_RECSTOP_OFF Do not use stop character. SERIAL_RECSTOP_CHAR_1 Use stop character 1 SERIAL_RECSTOP_CHAR_1_2 Use stop characters 1 and 2, that is cancel when the 1st stop character is followed by the 2nd stop character. SERIAL_RECSTOP_LEN Terminate read request when the number of characters defined by *RecLen* has been read in.
RecStop1	1st stop character for write requests
RecStop2	2nd stop character for write requests
RecTimeout	Maximum time span which is allowed to elapse between the reading of two characters (ms). If this time span is exceeded, the read request is canceled. Specifying 0 deactivates the function
RecLen	Number of characters after which read requests are terminated

Description

`SerialInitEx` is used for extended initialization of the unit of a driver for a serial interface. The unit is specified by *Handle*. *Handle* is a descriptor that was generated with `SerialOpen`.

The *Baud* parameter specifies the baud rate. The parameters *Data* and *Stop* specify the number of data and stop bits. The *Parity* parameter is used to control the parity.

The *BufferSize* parameter specifies the size of the background buffer.

Parameters *SendStopMode*, *SendStop1* and *SendStop2* define the use and type of stop characters for write requests. The *SendDelay* parameter specifies the minimum pause between two write requests.

Parameters *RecStopMode*, *RecStop1* und *RecStop2* define the use and type of stop characters for read requests. The *RecTimeout* parameter specifies the time after which a read request is canceled.

The *RecLen* parameter specifies the number of characters after which read requests are terminated.

Return Value

RM_OK	The function was successfully executed

Error Codes

See “Error Codes for Loadable Drivers”

Note

This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also

SerialClose, SerialInit, SerialOpen

SerialOpen

Function **Establish a connection to a unit of a driver**

Syntax

```
#include <serial.h>
int          SerialOpen(
               const char *UnitName,
               RmIOHandle *Handle );
```

Parameters

Parameter Name	Meaning
UnitName	Name of the unit in the RMOS resource catalog. This name is assigned when the unit is created.
Handle	Pointer to a variable of the type RmIOHandle in which a descriptor for addressing the unit is stored.

Description `SerialOpen` establishes a connection to the unit identified by *UnitName*.

Return Value RM_OK The function was successfully executed

Error Codes See “Error Codes for Loadable Drivers”

Note This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also **SerialClose**

SerialPutChar

Function **Write a single character to a unit**

Syntax

#include <serial.h>

int **SerialPutChar(**
RmIOHandle *Handle***,**
char *Char* **);**

Parameters

Parameter Name	Meaning
Handle	Descriptor
Char	Character to be written

Description `SerialPutChar` writes the character *Char* to the unit specified by *Handle*. *Handle* is a descriptor that was generated with `SerialOpen`.

Return Value RM_OK The function was successfully executed

Error Codes See “Error Codes for Loadable Drivers”

Note This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also **SerialGetChar, SerialGetString, SerialPutString**

SerialPutString

Function **Write characters to the unit**

Syntax

```
#include <serial.h>
int          SerialPutString(
               RmIOHandle Handle,
               char *String,
               ulong MaxLen );
```

Parameters

Parameter Name	Meaning
Handle	Descriptor
String	Address of a memory area with the characters to be written
MaxLen	Number of characters to be written

Description `SerialPutString` writes *MaxLen* characters from the address *String* to the unit specified by *Handle*. *Handle* is a descriptor that was generated with `SerialOpen`.

Return Value RM_OK The function was successfully executed

Error Codes See “Error Codes for Loadable Drivers”

Note This call can only be used for the SER8250.DRV driver for serial interfaces. The RMFSERB.LIB library is required when the application is linked.

See Also **SerialGetChar, SerialGetString, SerialPutChar**

x_dos_cpyin

Function

Allocate memory area from transfer buffer and copy in data

Syntax

```
#include <rm3dos.h>
char *          x_dos_cpyin  (
                  char *buffer,
                   int  len ) ;
```

Parameters

Parameter Name	Meaning
buffer	Pointer to data to be copied into the transfer buffer. Enter NUL if you only want to allocate the memory area.
len	Length in bytes of the memory area to be allocated.

Description

This function first allocates a memory area from the transfer buffer. It then copies data to the allocated memory.

The transfer buffer is located below 1 Mbyte and is required for data exchange with the DOS task and with DOS/BIOS system calls.

The allocated memory area can be freed again with the `x_dos_cpyout` function.

The size of the transfer buffer can be specified when loading the RM3_TSR terminate-and-stay-resident program. It can be up to 30 bytes. All areas of the transfer buffer which are not required should always be freed to ensure that memory is always available.

The transfer is reinitialized after a warm start, and allocated memory is freed. In certain circumstances, the transfer buffer may now be located at another point and data may be lost.

Return Value

The return value is a pointer.

If bit 31 of the return value is set, that is if the value is negative, the required memory could not be allocated. In this case, the lower 16 bits specify the largest memory area currently available.

If the pointer is positive (bit 31=0), it contains the physical start address of the allocated memory area.

Note

The value returned by the function can not be passed to MS-DOS or the BIOS in this format. The pointer must first be converted to a real-mode pointer, comprising a segment plus offset.

```
const char filename="c:\clistart.bat";
char *pptr;
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)
```

See Also

x_dos_cpyout

x_dos_cpyout

Function

Copy data from allocated memory area in transfer buffer and free the area

Syntax

```
#include <rm3dos.h>
int          x_dos_cpyout  (
               char *addr,
               char *anwenderpuffer,
               int len) ;
```

Parameters

Parameter Name	Meaning
addr	Pointer to data area in the transfer buffer. This value corresponds to the return value of the `x_dos_cpyin` function.
anwenderpuffer	Pointer to the area to which the data from the transfer buffer are to be copied. If this value is NUL, the memory area to which addr points is freed without copying the data.
len	Length in bytes of the memory area to be copied. If this value is less than the actual length of memory allocated, the entire area is still freed.

Description

This function first copies data from a memory area in the transfer buffer. It then frees the area.

Return Value

Length of the freed area.

If this value is 0, an invalid value was passed in the ***addr*** parameter.

See Also

x_dos_cpyin

Index

E

F

G

I

L

M

N

O

P

R

S

T

U