

FR Family

μ T-Kernel Specifications-Compliant

SOFTUNE™ μ T-REALOS/FR

API REFERENCE



FR Family

μ T-Kernel Specifications-Compliant

SOFTUNETM μ T-REALOS/FR

API REFERENCE



FUJITSU MICROELECTRONICS LIMITED

Introduction

■ Objective of This Document

This document describes the μ T-REALOS API and is intended for use by engineers who need to write application programs for SOFTUNE μ T-REALOS/FR (abbreviated in this document as μ T-REALOS). Refer to this document as required for information about the μ T-REALOS API. Also refer to this manual as required for details on using the μ T-REALOS analyzer. It is also recommended that you read the SOFTUNE μ T-REALOS/FR User's Guide (referred to in this document as the User's Guide) before reading this document.

μ T-REALOS is a real-time OS based on the μ T-Kernel specification that runs on the FR family of 32-bit RISC controllers.

The μ T-Kernel specification is an open specification for a real-time OS developed by the T-Engine Forum. The μ T-Kernel Specification Manual is available on the T-Engine Forum's website (<http://www.t-engine.org/>). The original copyright to μ T-Kernel belongs to Mr. Ken Sakamura, and copyright to the μ T-Kernel specification belongs to the T-Engine Forum. This product uses the source code of μ T-Kernel in accordance with μ T-License in T-Engine Forum (www.t-engine.org).

■ Trademarks

REALOS and SOFTUNE are trademarks of Fujitsu Microelectronics Limited, Japan.

TRON is an abbreviation for "The Real-time Operating system Nucleus".

ITRON is an abbreviation for "Industrial TRON".

μ ITRON is an abbreviation for "Micro Industrial TRON".

T-Kernel and μ T-Kernel are the names of computer specifications and do not indicate a specific product or group of products.

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

■ Overall Structure of This Document

This manual consists of 3 chapters and an appendix, as listed below.

CHAPTER 1 GENERAL DESCRIPTION

This chapter summarizes the basic information about using the μ T-REALOS API.

CHAPTER 2 DATA TYPES

This chapter describes the C data types used by μ T-REALOS.

CHAPTER 3 SYSTEM CALL INTERFACE

This chapter explains the μ T-Kernel based system call interface supported by μ T-REALOS.

APPENDICES

The Appendices describe the error codes, define macros, device driver interface, and points to note when porting from a μ ITRON OS. An alphabetic index of system calls is also included.

■ Other Relevant Manuals

Please refer to the following manuals as required when using this system.

SOFTUNE μ T-REALOS/FR User's Guide

FR Family SOFTUNE C/C++ Compiler Manual V6

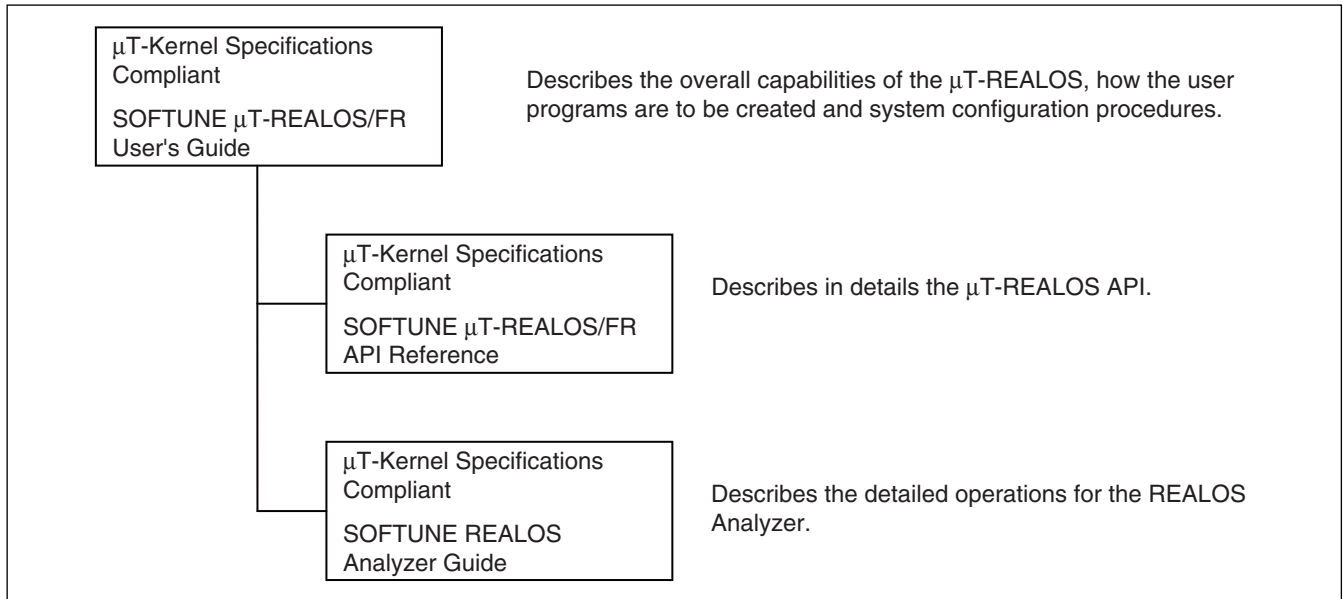
FR Family SOFTUNE Assembler Manual V6

FR Family SOFTUNE Linkage Kit Manual V6

■ μ T-REALOS Manuals

The following three manuals are available for μ T-REALOS.

When using μ T-REALOS for the first time, please read the SOFTUNE μ T-REALOS/FR User's Guide first.



■ How to Use This Manual

● Meaning of Symbols

This manual uses the following notation to describe the system call parameters.

Table Symbol Meanings

Symbol	Explanation
[]	Indicates that the elements inside the [] may be omitted.
	Indicates that one of the listed elements is to be selected.
	Indicates that any of the listed elements can be selected.
: =	Indicates that the element on the left takes the value of the element on the right.

- The contents of this document are subject to change without notice.
Customers are advised to consult with sales representatives before ordering.
- The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of FUJITSU MICROELECTRONICS device; FUJITSU MICROELECTRONICS does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. FUJITSU MICROELECTRONICS assumes no liability for any damages whatsoever arising out of the use of the information.
- Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of FUJITSU MICROELECTRONICS or any third party or does FUJITSU MICROELECTRONICS warrant non-infringement of any third-party's intellectual property right or other right by using such information. FUJITSU MICROELECTRONICS assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.
- The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
Please note that FUJITSU MICROELECTRONICS will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.
- Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
- Exportation/release of any products described in this document may require necessary procedures in accordance with the regulations of the Foreign Exchange and Foreign Trade Control Law of Japan and/or US export control laws.
- The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

Copyright© 2008 FUJITSU MICROELECTRONICS LIMITED All rights reserved.

Copyright© 2006 T-Engine Forum All rights reserved.

This manual is made based on the specification of μ T-Kernel with the formal agreement by the T-Engine Forum.

CONTENTS

CHAPTER 1	GENERAL DESCRIPTION	1
1.1	Explanation of Terms	2
1.2	Overview of the μ T-REALOS API	4
1.3	Implementation Definition and Implementation-specific Specifications	5
1.4	Extensions	8
CHAPTER 2	DATA TYPES	9
2.1	Standard Data Types and Define Macros	10
2.2	Data Types and Define Macros That Have a Specific Meaning in μ T-Kernel	12
CHAPTER 3	SYSTEM CALL INTERFACE	15
3.1	List of System Calls	16
3.2	System Call Descriptions	22
3.3	System Calls for Task Management Function	24
3.3.1	tk_cre_tsk (Create Task)	25
3.3.2	tk_del_tsk (Delete Task)	28
3.3.3	tk_sta_tsk (Start Task)	29
3.3.4	tk_ext_tsk (Exit Task)	30
3.3.5	tk_exd_tsk (Exit and Delete Task)	31
3.3.6	tk_ter_tsk (Terminate Task)	32
3.3.7	tk_chg_pri (Change Task Priority)	34
3.3.8	tk_get_reg (Get Task Registers)	37
3.3.9	tk_set_reg (Set Task Registers)	39
3.3.10	tk_ref_tsk (Refer Task Status)	41
3.4	System Calls for Task-dependent Synchronization Function	45
3.4.1	tk_slp_tsk (Sleep Task)	46
3.4.2	tk_wup_tsk (Wakeup Task)	48
3.4.3	tk_can_wup (Cancel Wakeup Task)	50
3.4.4	tk_rel_wai (Release Wait)	52
3.4.5	tk_sus_tsk (Suspend Task)	54
3.4.6	tk_rsm_tsk (Resume Task)	56
3.4.7	tk_frsm_tsk (Force Resume Task)	58
3.4.8	tk_dly_tsk (Delay Task)	60
3.5	System Calls for Synchronization/Communication Function	61
3.5.1	Semaphore Function System Calls	62
3.5.1.1	tk_cre_sem (Create Semaphore)	63
3.5.1.2	tk_del_sem (Delete Semaphore)	65
3.5.1.3	tk_sig_sem (Signal Semaphore)	66
3.5.1.4	tk_wai_sem (Wait on Semaphore)	68
3.5.1.5	tk_ref_sem (Refer Semaphore Status)	70
3.5.2	Event Flag Function System Calls	72
3.5.2.1	tk_cre_flg (Create Event Flag)	73
3.5.2.2	tk_del_flg (Delete Event Flag)	75

3.5.2.3	tk_set_flg (Set Event Flag)	76
3.5.2.4	tk_clr_flg (Clear Event Flag)	77
3.5.2.5	tk_wai_flg (Wait Event Flag)	78
3.5.2.6	tk_ref_flg (Refer Event Flag Status)	81
3.5.3	Mailbox Function System Calls	83
3.5.3.1	tk_cre_mbx (Create Mailbox)	84
3.5.3.2	tk_del_mbx (Delete Mailbox)	86
3.5.3.3	tk_snd_mbx (Send Message to Mailbox)	87
3.5.3.4	tk_rcv_mbx (Receive Message from Mailbox)	89
3.5.3.5	tk_ref_mbx (Refer Mailbox Status)	92
3.6	System Calls for Extended Synchronization/Communication Function	94
3.6.1	Mutex Function System Calls	95
3.6.1.1	tk_cre_mtx (Create Mutex)	96
3.6.1.2	tk_del_mtx (Delete Mutex)	98
3.6.1.3	tk_loc_mtx (Lock Mutex)	99
3.6.1.4	tk_unl_mtx (Unlock Mutex)	101
3.6.1.5	tk_ref_mtx (Refer Mutex Status)	103
3.6.2	Message Buffer Function System Calls	105
3.6.2.1	tk_cre_mbf (Create MessageBuffer)	106
3.6.2.2	tk_del_mbf (Delete MessageBuffer)	108
3.6.2.3	tk_snd_mbf (Send Message to MessageBuffer)	109
3.6.2.4	tk_rcv_mbf (Receive Message from MessageBuffer)	111
3.6.2.5	tk_ref_mbf (Refer MessageBuffer Status)	113
3.6.3	Rendezvous Function System Calls	115
3.6.3.1	tk_cre_por (Create Port for Rendezvous)	116
3.6.3.2	tk_del_por (Delete Port for Rendezvous)	118
3.6.3.3	tk_cal_por (Call Port for Rendezvous)	120
3.6.3.4	tk_acp_por (Accept Port for Rendezvous)	123
3.6.3.5	tk_fwd_por (Forward Rendezvous to Another Port)	126
3.6.3.6	tk_rpl_rdv (Reply Rendezvous)	129
3.6.3.7	tk_ref_por (Refer Port Status)	131
3.7	Memory Pool Management Function System Calls	133
3.7.1	Fixed-size Memory Pool Function System Calls	134
3.7.1.1	tk_cre_mpf (Create Fixed-size MemoryPool)	135
3.7.1.2	tk_del_mpf (Delete Fixed-size MemoryPool)	138
3.7.1.3	tk_get_mpf (Get Fixed-size Memory Block)	140
3.7.1.4	tk_rel_mpf (Release Fixed-size Memory Block)	142
3.7.1.5	tk_ref_mpf (Refer Fixed-size MemoryPool Status)	144
3.7.2	Variable-size Memory Pool Function System Calls	146
3.7.2.1	tk_cre_mpl (Create Variable-size MemoryPool)	147
3.7.2.2	tk_del_mpl (Delete Variable-size MemoryPool)	150
3.7.2.3	tk_get_mpl (Get Variable-size Memory Block)	152
3.7.2.4	tk_rel_mpl (Release Variable-size Memory Block)	154
3.7.2.5	tk_ref_mpl (Refer Variable-size MemoryPool Status)	156
3.8	Time Management Function System Calls	158
3.8.1	System Time Management Function System Calls	159
3.8.1.1	tk_set_tim (Set Time)	160

3.8.1.2	tk_get_tim (Get Time)	162
3.8.1.3	tk_get_otm (Get Operating Time)	163
3.8.1.4	isig_tim (Signal Time)	164
3.8.2	Cyclic Handler Function System Calls	165
3.8.2.1	tk_cre_cyc (Create Cyclic Handler)	166
3.8.2.2	tk_del_cyc (Delete Cyclic Handler)	169
3.8.2.3	tk_sta_cyc (Start Cyclic Handler)	170
3.8.2.4	tk_stp_cyc (Stop Cyclic Handler)	171
3.8.2.5	tk_ref_cyc (Refer Cyclic Handler Status)	172
3.8.3	Alarm Handler Function System Calls	174
3.8.3.1	tk_cre_alm (Create Alarm Handler)	175
3.8.3.2	tk_del_alm (Delete Alarm Handler)	177
3.8.3.3	tk_sta_alm (Start Alarm Handler)	178
3.8.3.4	tk_stp_alm (Stop Alarm Handler)	179
3.8.3.5	tk_ref_alm (Refer Alarm Handler Status)	180
3.9	Interrupt Control Function System Calls	182
3.9.1	tk_def_int (Define Interrupt Handler)	183
3.9.2	tk_ret_int (Return from Interrupt Handler)	185
3.9.3	DI	186
3.9.4	EI	187
3.9.5	isDI	188
3.10	System Status Management Function System Calls	189
3.10.1	tk_rot_rdq (Rotate Ready Queue)	190
3.10.2	tk_get_tid (Get Task Identifier)	192
3.10.3	tk_dis_dsp (Disable Dispatch)	193
3.10.4	tk_ena_dsp (Enable Dispatch)	195
3.10.5	tk_ref_sys (Refer System Status)	196
3.10.6	tk_ref_ver (Refer Version Information)	198
3.11	Sub System Function System Calls	200
3.11.1	tk_def_ssy (Define Subsystem)	201
3.11.2	tk_ref_ssy (Refer Subsystem Status)	204
3.12	Device Management Function System Calls	205
3.12.1	tk_def_dev (Define Device)	206
3.12.2	tk_ref_idv (Refer Initial Device Information)	209
3.12.3	tk_opn_dev (Open Device)	210
3.12.4	tk_cls_dev (Close Device)	212
3.12.5	tk_rea_dev (Read Device)	213
3.12.6	tk_srea_dev (Synchronous Read Device)	215
3.12.7	tk_wri_dev (Write Device)	217
3.12.8	tk_swri_dev (Synchronous Write Device)	219
3.12.9	tk_wai_dev (Wait Device)	221
3.12.10	tk_sus_dev (Suspend Device)	223
3.12.11	tk_get_dev (Get Device)	225
3.12.12	tk_ref_dev (Refer Device)	226
3.12.13	tk_oref_dev (Refer Device)	228
3.12.14	tk_lst_dev (List Device)	230
3.12.15	tk_evt_dev (Event Device)	232

APPENDIX	233
APPENDIX A Error Codes	234
APPENDIX B Define Macros	236
APPENDIX C Device Driver Interface	240
APPENDIX D Points to Note When Porting From a μ ITRON OS	254
APPENDIX E System Call Index	255
INDEX.....	259

CHAPTER 1

GENERAL DESCRIPTION

This chapter summarizes the basic information about using the μ T-REALOS API.

- 1.1 Explanation of Terms
- 1.2 Overview of the μ T-REALOS API
- 1.3 Implementation Definition and Implementation-specific Specifications
- 1.4 Extensions

1.1 Explanation of Terms

This section explains the main terms used with μ T-REALOS.

■ μ T-REALOS Terminology

Table 1.1-1 lists the main terminology used with μ T-REALOS. The terms listed below are also explained in the User's Guide. Refer to the User's Guide for details.

Table 1.1-1 μ T-REALOS Terminology List (1 / 2)

Term	Meaning
Kernel	The program that executes the OS functions is called the kernel.
User program	An application that uses the μ T-REALOS functions. Referred to as a user program in this manual to emphasize the fact that the program is written by the user.
Task	The basic unit of a user program. The operation of the user program is implemented as a number of cooperating tasks.
Quasi-task	A handler that has its invoking task context is called a quasi-task. In the μ T-Kernel specifications, a quasi-task is equivalent to an extended SVC handler. A quasi-task can enter the wait state and dispatch can be executed from within the quasi-task. The restrictions on issuing system calls are the same as for a task.
Non-task	A unit of execution handled by the OS that is not a task. These consist of quasi-tasks, interrupt handlers, and time event handlers.
Task-independent portion	A handler that does not have a task context is called a task-independent portion. These consist of interrupt handlers, time event handlers, and initialization routines. System calls such as tk_slp_tsk that can place a task in the wait state cannot be used from a task-independent portion.
Dispatch	Dispatch is the process of switching CPU execution between different tasks. The kernel mechanism that implements the dispatch function is called the dispatcher.
Preempt	The operation whereby a program's CPU execution rights are overridden.
API	The general name for the function call interface and programming conventions for user programs provided by μ T-REALOS (and μ T-Kernel).
System call	The system calls are a set of routines that can be called directly from user programs to perform OS functions.
Object	A resource managed by the kernel is called an object. Specific examples include tasks and also the semaphores, mailboxes, and other resources used for synchronization and communication.
Context	The context is the environment in which a program executes (specifically, the copy of the CPU register contents saved to memory when program execution is interrupted).
C language routines	Routines provided by the compiler that can be used when writing C programs. These routines perform pre-processing to backup and restore the registers used by the program and to make the C function arguments accessible from within the routine. The "high-level language routines" in the μ T-Kernel specification are equivalent to the "C language routines" in μ T-REALOS. (Because C is the only high-level language supported by μ T-REALOS.)

Table 1.1-1 μ T-REALOS Terminology List (2 / 2)

Term	Meaning
Current priority	The current priority for task execution. The dispatcher switches between tasks based on this priority. The priority can be changed from within the user program by using <code>tk_chg_pri</code> . The term "priority" on its own typically means the "current priority".
Initial priority	This is the initial priority when the task is first started. The current priority and base priority for the task are set to this initial priority value when the task starts. The initial priority can be specified when the task is created but cannot be subsequently modified.
Base priority	This is the main priority for the task. It can be changed from within the user program by calling <code>tk_chg_pri</code> . Whereas the current priority can be modified temporarily by the OS when the mutex function is used, it is subsequently returned to the base priority. If the mutex function is not used, the current priority is always the same as the base priority.
Priority order	The priority order determines the order in which tasks are assigned execution rights when more than one task is in the READY state. Execution rights are given first to the task with the highest priority. Tasks with higher priority value have higher priority. When there is more than one task with the same priority value, the earlier a task entered the READY state, the higher its priority.
System down	A system down occurs if the kernel determines that the user system is unable to continue execution. When a system down occurs, execution jumps to a specific address in the kernel where it enters an infinite loop which halts user system execution.
Configurator	A utility used specifically for μ T-REALOS to generate the kernel configuration (user-specified functions such as the maximum task priority, maximum number of objects, and system stack size). Refer to "3.13 Kernel Configuration" and "5.3 Configuration Setting" in the User's Guide for details.

1.2 Overview of the μ T-REALOS API

This section describes the elements that make up the μ T-REALOS API.

■ Data Types

The μ T-Kernel specification redefines the standard char, int and other basic types with type names specific to μ T-Kernel. The purpose is to be more rigorous about how these types are used so as to prevent programming errors. See "CHAPTER 2 DATA TYPES" for details of the available data types.

■ System Calls

User programs can treat the system calls provided by μ T-REALOS as C function calls and use them to control program operation and reference the system status. Most system calls use the return value to return an error code.

● System call format

Error code = system call function name (parameter,,);

Example: `ercd = tk_del_tsk (task_id);`

See "CHAPTER 3 SYSTEM CALL INTERFACE" for details of the system calls.

■ Error Codes

If an error occurs, the system calls return an error code with a negative value. If the operation is successful, the system calls return E_OK or a positive value.

See "CHAPTER 3 SYSTEM CALL INTERFACE" for a list of error codes returned by system calls. Also, "APPENDIX A Error Codes" contains a list of the error codes defined by μ T-REALOS and their associated meanings.

1.3 Implementation Definition and Implementation-specific Specifications

This section describes the implementation definition and implementation-specific specifications.

■ Implementation Definition

The specifications that need to be defined specifically for each OS that implements the μ T-Kernel specification are called the "implementation definition". Note that the specifications covered by the implementation definition are not compatible across different μ T-Kernel compliant OSs.

Table 1.3-1 lists the implementation definition for μ T-REALOS.

Table 1.3-1 μ T-REALOS Implementation Definitions (1 / 3)

Category	Implementation Definition	Explanation
Operation	Interface to use when system calls are called from programs written in assembly language.	Uses the procedure for calling C functions from assembly language. Refer to "Chapter 4 Using the fcc911s Command to Create an Object Program" in the "FR Family SOFTUNE C/C++ Compiler Manual" (henceforth referred to as the "compiler manual") for details.
	System call implementation	Function call (software traps are not used).
	Operation when certain system calls* are called from task-independent portions	See Table 3.1-1 and "CHAPTER 3 SYSTEM CALL INTERFACE".
	Operation when an error occurs in a function that does not return to the caller (tk_ext_tsk, tk_extd_tsk, and tk_ret_int)	If a system call error routine is defined, the error routine is called, except in the case of tk_ret_int. The operation in the case of tk_ret_int is not guaranteed. If no error routine is defined, is not guaranteed.
	System call function codes	Function codes are not used in μ T-REALOS.
	Structure of mailbox message header	See the ■Description section in "3.5.3.3 tk_snd_mbx (Send Message to Mailbox)".
	Rendezvous number allocation	The lower 16 bits contain the task ID and the upper 16 bits contain a unique value assigned sequentially in the order the rendezvous are accepted.
	Time event handlers with the TA_ASM attribute	Not supported by μ T-REALOS.

Table 1.3-1 μ T-REALOS Implementation Definitions (2 / 3)

Category	Implementation Definition	Explanation
Operation	Simultaneous activation of multiple interrupt, time event, or other handlers.	<p>If time event handlers become active at the same time, a time event handler is executed after execution of another time event handler is completed.</p> <p>If an interrupt with a higher priority occurs while an interrupt handler is being executed, the process of the interrupt handler in execution is suspended and the higher priority interrupt handler is executed. Then the process of the suspended interrupt handler is resumed.</p> <p>If an interrupt with a higher priority than that of a timer interrupt occurs while the time event handler is being executed, the process of the interrupt handler in execution is suspended and the higher priority interrupt handler is executed. Then the process of the suspended time event handler is resumed.</p> <p>If a time event handler is activated while an interrupt handler with a lower priority than that of the timer interrupt is being executed, the process of the interrupt handler in execution is suspended and the time event handler is executed. Then the process of the suspended interrupt handler is resumed.</p>
	State when an interrupt handler with the TA_ASM attribute is started (whether or not system calls are permitted, etc.)	Treated as a task-independent portion.
	Operation when an interrupt occurs after clearing the interrupt handler definition	A system down occurs.
	Status of stack and registers on entering an interrupt handler with the TA_ASM attribute, whether or not system calls are permitted, method for calling system calls, and method for returning from interrupt handler without going via OS.	See "4.8 Interrupt Handlers" in the User's Guide.
	Content of T_REGS, T_EIT, and T_CREGS	See "3.3.8 tk_get_reg (Get Task Registers)" and "3.3.9 tk_set_reg (Set Task Registers)".
	Parameter (iststs) for the CPU interrupt control functions (DI, EI, and isDI)	See "3.9.3 DI", "3.9.4 EI" and "3.9.5 isDI".
	Second parameter of interrupt handler	Not used
	Message buffer used to notify device information	Used
	E_CTX error check when tk_ret_int called from a time event handler.	No error check is performed.
	Definition and error generation conditions for system objects that can trigger the E_OACV error.	Devices are defined as objects that can trigger the E_OACV error. Also see "3.12 Device Management Function System Calls" for details of what conditions generate this error.

Table 1.3-1 μ T-REALOS Implementation Definitions (3 / 3)

Category	Implementation Definition	Explanation
Values	Maximum size of the wakeup request queue for tk_wup_tsk	32767 (0x7fff)
	Maximum nesting level for suspend requests using tk_sus_tsk	32767 (0x7fff)
	Maximum semaphore count	2147483647 (0x7fffffff)
	Maximum available sub-system ID value	See "3.13 Kernel Configuration" in the User's Guide.
	Maximum available sub-system priority setting (ssypri)	16
	ssypri and resblksz values for tk_ref_ssy	Undefined values
	Value of tk_def_int dintno parameter	Interrupt vector number
	Maximum suspend prohibit request count for devices	2147483647 (0x7fffffff)

*: Routines for which calling from task-independent portions is not explicitly permitted

■ Implementation-specific Definitions

The μ T-Kernel specifications for which the OS behavior can differ depending on differences in the hardware are called implementation-specific definitions. Note that the implementation-specific specifications are not necessarily compatible across different μ T-Kernel OS implementations.

Table 1.3-2 lists the μ T-REALOS implementation-specific definitions.

Table 1.3-2 μ T-REALOS Implementation-Specific Definitions

Category	Implementation-Specific Definition	Explanation
Definitions	Task format when the TA_ASM attribute is specified	Not supported by μ T-REALOS.
Operation	E_MACV error detection	μ T-REALOS does not use this error.
	Implementation-specific generation of the E_CTX error	Details of E_CTX error generation are described in the system call explanations in "CHAPTER 3 SYSTEM CALL INTERFACE".

1.4 Extensions

μ T-REALOS has some extensions to the μ T-Kernel specification. This section describes these extensions.

■ Extensions

μ T-REALOS supports all of the μ T-Kernel functions except for the debugger support functions. It also includes a number of extensions.

Table 1.4-1 lists the extensions to the μ T-Kernel specification.

Table 1.4-1 μ T-REALOS Extensions

Extension	Definition	
Initialization routine	This extension applies to μ T-REALOS only Specifies a program to execute before starting a task. See "2.2.1 Tasks" in the User's Guide for details of this function.	
	Maximum number of initialization routines able to be registered	1
Error routine	This extension applies to μ T-REALOS only Specifies a program to execute when an error is detected in the kernel. See "2.3.1 task-independent portion" in the User's Guide for details of this function.	
	Maximum number of error routines able to be registered	1
isig_tim	This extension applies to μ T-REALOS only Function for updating the system time from a user program. See "3.8.1.4 isig_tim (Signal Time)" for details.	
Interrupt handler static registration function	This extension applies to μ T-REALOS only. It registers static interrupt handlers. See "3.12 Kernel Configuration" and "5.2 Kernel Configuration" in the User's Guide for details.	

CHAPTER 2

DATA TYPES

This chapter describes the C data types used by μ T-REALOS.

2.1 Standard Data Types and Define Macros

2.2 Data Types and Define Macros That Have a Specific Meaning in μ T-Kernel

2.1 Standard Data Types and Define Macros

This section describes the standard data types and define macros specified in the μ T-Kernel specification.

■ Standard Data Types

The standard data types and define macros consist of the base data types and define macros used to define the data types described in "2.2 Data Types and Define Macros That Have a Specific Meaning in μ T-Kernel". Table 2.1-1 and Table 2.1-2 list the standard data types.

Table 2.1-1 Data Types (1 / 2)

Type Name	C Data Type	Meaning
B	signed char	Signed 8-bit integer
H	signed short	Signed 16-bit integer
W	signed long	Signed 32-bit integer
UB	unsigned char	Unsigned 8-bit integer
UH	unsigned short	Unsigned 16-bit integer
UW	unsigned long	Unsigned 32-bit integer
VB	signed char	8-bit data of undefined type
VH	signed short	16-bit data of undefined type
VW	signed long	32-bit data of undefined type
VP	void*	Pointer to data of undefined type
_B	volatile signed char	Adds the volatile declaration
_H	volatile signed short	
_W	volatile signed long	
_UB	volatile unsigned char	
_UH	volatile unsigned short	
_UW	volatile unsigned long	
INT	signed int	Signed integer, where the number of bits depends on the CPU bit width
UINT	unsigned int	Unsigned integer, where the number of bits depends on the CPU bit width
ID	signed int	General-purpose ID
MSEC	signed long	General-purpose time (ms)
(*FP)()	void	General-purpose function address

Table 2.1-1 Data Types (2 / 2)

Type Name	C Data Type	Meaning
(*FUNCP)()	signed int	General-purpose function address
BOOL	signed int	Boolean value
TC	unsigned short	TRON character code

Table 2.1-2 Define Macros

Macro Name	Value	Meaning
LOCAL	static	Defines a local symbol
EXPORT	(none)	Defines a global symbol
IMPORT	extern	Global symbol reference
TRUE	1	True Boolean value
FALSE	0	False Boolean value
TNULL	((TC)0)	TRON code character string terminator

Note: The difference between VB, VH and VW, and B, H and W is that, whereas the former indicate a specific number of bits only and do not specify the type of the contained data value, the latter explicitly indicate an integer.

Even parameters such as the task stack size which clearly cannot be negative use the signed integer type (INT or W). This is due to the rule in the μ T-Kernel specification that integers are treated as signed values wherever possible. Similarly, timeout (TMO tmout) parameters also use the signed integer type, and the value TMO_FEVR (= -1) has a specific meaning. Parameters that are used as bit patterns use an unsigned data type.

2.2 Data Types and Define Macros That Have a Specific Meaning in μ T-Kernel

This section describes the data types and define macros that have a specific meaning in μ T-Kernel.

■ Data Types That Have a Specific Meaning in μ T-Kernel

Data types and define macros that have a specific meaning in μ T-Kernel are those data types and define macros that are used to explicitly indicate the meaning of parameters.

The tables from Table 2.2-1 to Table 2.2-3 list commonly used data types and define macros.

"Appendix B Define Macros" has a full list of define macros.

Table 2.2-1 Data Types That Have a Specific Meaning (Excluding Structures)

Type Name	Data Type	Meaning
FN	INT	Function code
RNO	INT	Rendezvous number
ATR	UW	Object or handler attribute
ER	INT	Error code
PRI	INT	Priority
TMO	W	Timeout
RELTIM	UW	Relative time

Table 2.2-2 Data Types That Have a Specific Meaning (Structures)

Type Name	Member	Data Type	Meaning
SYSTM	hi	W	Upper 32 bits of the system time
	lo	UW	Lower 32 bits of the system time

Table 2.2-3 Define Macros That Have a Specific Meaning

Macro Name	Value	Meaning
NULL	0	Undefined pointer
TA_NULL	0	Does not specify any specific attribute
TMO_POL	0	Polling
TMO_FEVR	(-1)	Wait indefinitely
TSK_SELF	0	This task
TPRI_INI	0	Initial priority
TPRI_RUN	0	Priority of task in the RUN state

Note: The most representative data type is used for parameters that can contain more than one data type. For example, although the return value from `tk_cre_tsk` can be either a task ID or an error code, the function typically returns a task ID and therefore the ID data type is used.

CHAPTER 3

SYSTEM CALL INTERFACE

This chapter explains the μ T-Kernel based system call interface supported by μ T-REALOS.

- 3.1 List of System Calls
- 3.2 System Call Descriptions
- 3.3 System Calls for Task Management Function
- 3.4 System Calls for Task-dependent Synchronization Function
- 3.5 System Calls for Synchronization/Communication Function
- 3.6 System Calls for Extended Synchronization/Communication Function
- 3.7 Memory Pool Management Function System Calls
- 3.8 Time Management Function System Calls
- 3.9 Interrupt Control Function System Calls
- 3.10 System Status Management Function System Calls
- 3.11 Sub System Function System Calls
- 3.12 Device Management Function System Calls

3.1 List of System Calls

This section explains the system calls supported by μ T-REALOS.

■ List of System Calls

Table 3.1-1 lists the system calls provided by μ T-REALOS.

For details of each system call, see "3.2 System Call Descriptions".

Table 3.1-1 List of System Calls (1 / 6)

Type	Name	Explanation	Call availability		
			Task portion ^{*1}	Task-independent portion	Dispatch disabled
Task management function	tk_cre_tsk	Create task.	○	○	○
	tk_del_tsk	Delete task.	○	○	○
	tk_sta_tsk	Start task.	○	○	○
	tk_ext_tsk	End invoking task.	○	×	×
	tk_exd_tsk	End, and delete invoking task.	○	×	×
	tk_ter_tsk	Forcibly terminate other task.	○	○	○
	tk_chg_pri	Change the priority of task.	○	○	○
	tk_get_reg	Get task register.	○	×	○
	tk_set_reg	Set task register.	○	×	○
	tk_ref_tsk	Reference task state.	○	○	○
Task-dependent synchronization function	tk_slp_tsk	Place invoking task into wait state.	○	×	×
	tk_wup_tsk	Wake up other task.	○	○	○
	tk_can_wup	Cancel request for waking up task.	○	○	○
	tk_rel_wai	Forcibly release other task from wait state.	○	○	○
	tk_sus_tsk	Place task into suspend state.	○	○	○ ^{*2}
	tk_rsm_tsk	Resume task that is in suspend state.	○	○	○
	tk_frsm_tsk	Forcibly resume task that is in suspend state.	○	○	○

Table 3.1-1 List of System Calls (2 / 6)

Type	Name	Explanation	Call availability		
			Task portion ^{*1}	Task-independent portion	Dispatch disabled
Task-dependent synchronization function	tk_dly_tsk	Place invoking task in delay wait state.	○	×	×
Synchronization/communication function	tk_cre_sem	Create semaphore.	○	○	○
	tk_del_sem	Delete semaphore.	○	○	○
	tk_sig_sem	Return semaphore resources.	○	○	○
	tk_wai_sem	Get semaphore resources.	○	×	×
	tk_ref_sem	Reference semaphore state.	○	○	○
	tk_cre_flg	Create event flag.	○	○	○
	tk_del_flg	Delete event flag.	○	○	○
	tk_set_flg	Set event flag.	○	○	○
	tk_clr_flg	Clear event flag.	○	○	○
	tk_wai_flg	Wait for event flag.	○	×	×
	tk_ref_flg	Reference the state of event flag.	○	○	○
	tk_cre_mbx	Create mail box.	○	○	○
	tk_del_mbx	Delete mail box.	○	○	○
	tk_snd_mbx	Send message to mail box.	○	○	○
	tk_rcv_mbx	Receive message from mail box.	○	×	×
	tk_ref_mbx	Reference the state of mail box.	○	○	○
Extended synchronous communication function	tk_cre_mtx	Create mutex.	○	○	○
	tk_del_mtx	Delete mutex.	○	○	○
	tk_loc_mtx	Lock mutex.	○	×	×
	tk_unl_mtx	Unlock mutex.	○	×	○
	tk_ref_mtx	Reference the state of mutex.	○	○	○
	tk_cre_mbf	Create message buffer.	○	○	○

Table 3.1-1 List of System Calls (3 / 6)

Type	Name	Explanation	Call availability		
			Task portion ^{*1}	Task-independent portion	Dispatch disabled
Extended synchronous communication function	tk_del_mbf	Delete message buffer.	○	○	○
	tk_snd_mbf	Send message to message buffer.	○	○	○
	tk_rcv_mbf	Receive message from message buffer.	○	×	×
	tk_ref_mbf	Reference the state of message buffer.	○	○	○
	tk_cre_por	Create rendezvous port.	○	×	○
	tk_del_por	Delete rendezvous port.	○	×	○
	tk_cal_por	Call rendezvous for rendezvous port.	○	×	×
	tk_acp_por	Accept rendezvous for rendezvous port.	○	×	×
	tk_fwd_por	Forward rendezvous for rendezvous port.	○	×	○
	tk_rpl_rdv	Reply to rendezvous for rendezvous port.	○	×	○
	tk_ref_por	Reference the state of rendezvous port.	○	○	○
Memory pool management function	tk_cre_mpf	Create fixed-length memory pool.	○	×	×
	tk_del_mpf	Delete fixed-length memory pool.	○	×	×
	tk_get_mpf	Get fixed-length memory block.	○	×	×
	tk_rel_mpf	Return fixed-length memory block.	○	×	×
	tk_ref_mpf	Reference the state of fixed-length memory pool.	○	×	×
	tk_cre_mpl	Create variable-length memory pool.	○	×	×
	tk_del_mpl	Delete variable-length memory pool.	○	×	×
	tk_get_mpl	Get variable-length memory block.	○	×	×

Table 3.1-1 List of System Calls (4 / 6)

Type	Name	Explanation	Call availability		
			Task portion ^{*1}	Task-independent portion	Dispatch disabled
Memory pool management function	tk_rel_mpl	Return variable-length memory block.	○	×	×
	tk_ref_mpl	Reference the state of variable-length memory pool.	○	×	×
Time management function	tk_set_tim	Set system time.	○	○	○
	tk_get_tim	Reference current system time.	○	○	○
	tk_get_otm	Reference system operation time.	○	○	○
	isig_tim	Supply time tick. ^{*3}	×	○	○
	tk_cre_cyc	Create cyclic handler.	○	○	○
	tk_del_cyc	Delete cyclic handler.	○	○	○
	tk_sta_cyc	Start cyclic handler operation.	○	○	○
	tk_stp_cyc	Stop cyclic handler operation.	○	○	○
	tk_ref_cyc	Reference the state of cyclic handler.	○	○	○
	tk_cre_alm	Create alarm handler.	○	○	○
	tk_del_alm	Delete alarm handler.	○	○	○
	tk_sta_alm	Start alarm handler operation.	○	○	○
	tk_stp_alm	Stop alarm handler operation.	○	○	○
	tk_ref_alm	Reference the state of alarm handler.	○	○	○

Table 3.1-1 List of System Calls (5 / 6)

Type	Name	Explanation	Call availability		
			Task portion ^{*1}	Task-independent portion	Dispatch disabled
Interrupt management function	tk_def_int	Define interrupt handler.	○	○	○
	tk_ret_int	Return from interrupt handler.	×	○	○
	DI	Disable all external interrupts.	○	○	○
	EI	Enable all external interrupts.	○	○	○
	isDI	Check the disabled state of external interrupt.	○	○	○
System status management function	tk_rot_rdq	Rotate the priority order of tasks.	○	○	○
	tk_get_tid	Reference the task ID of running task.	○	○	○
	tk_dis_dsp	Disable dispatch.	○	×	○
	tk_ena_dsp	Enable dispatch.	○	×	○
	tk_ref_sys	Reference system status.	○	○	○
	tk_ref_ver	Reference version.	○	○	○
Subsystem management function	tk_def_ssy	Define subsystem.	○	○	○
	tk_ref_ssy	Reference subsystem definition information.	○	○	○
Device management function	tk_opn_dev	Open device.	○	○	○
	tk_cls_dev	Close device.	○	○	○
	tk_rea_dev	Start reading from device.	○	○	○
	tk_srea_dev	Start synchronous reading from device.	○	○	○
	tk_wri_dev	Start writing to device.	○	○	○
	tk_swri_dev	Start synchronous writing to device.	○	○	○
	tk_wai_dev	Wait for completion of request from device.	○	○	○
	tk_sus_dev	Suspend device.	○	○	○
	tk_get_dev	Get the device name of device.	○	○	○

Table 3.1-1 List of System Calls (6 / 6)

Type	Name	Explanation	Call availability		
			Task portion ^{*1}	Task-independent portion	Dispatch disabled
Device management function	tk_ref_dev	Get device information for device.	○	○	○
	tk_oref_dev	Get device information for device.	○	○	○
	tk_lst_dev	Get list of registered devices.	○	○	○
	tk_evt_dev	Send driver request event to device.	○	○	○
	tk_def_dev	Register device.	○	○	○
	tk_ref_idv	Get initial information for device.	○	○	○

○: Call enabled ×: Call disabled (System call returns an error or does not guarantee the operation.)

*1: "Task portion" under "Call availability" also includes "sub-task portion".

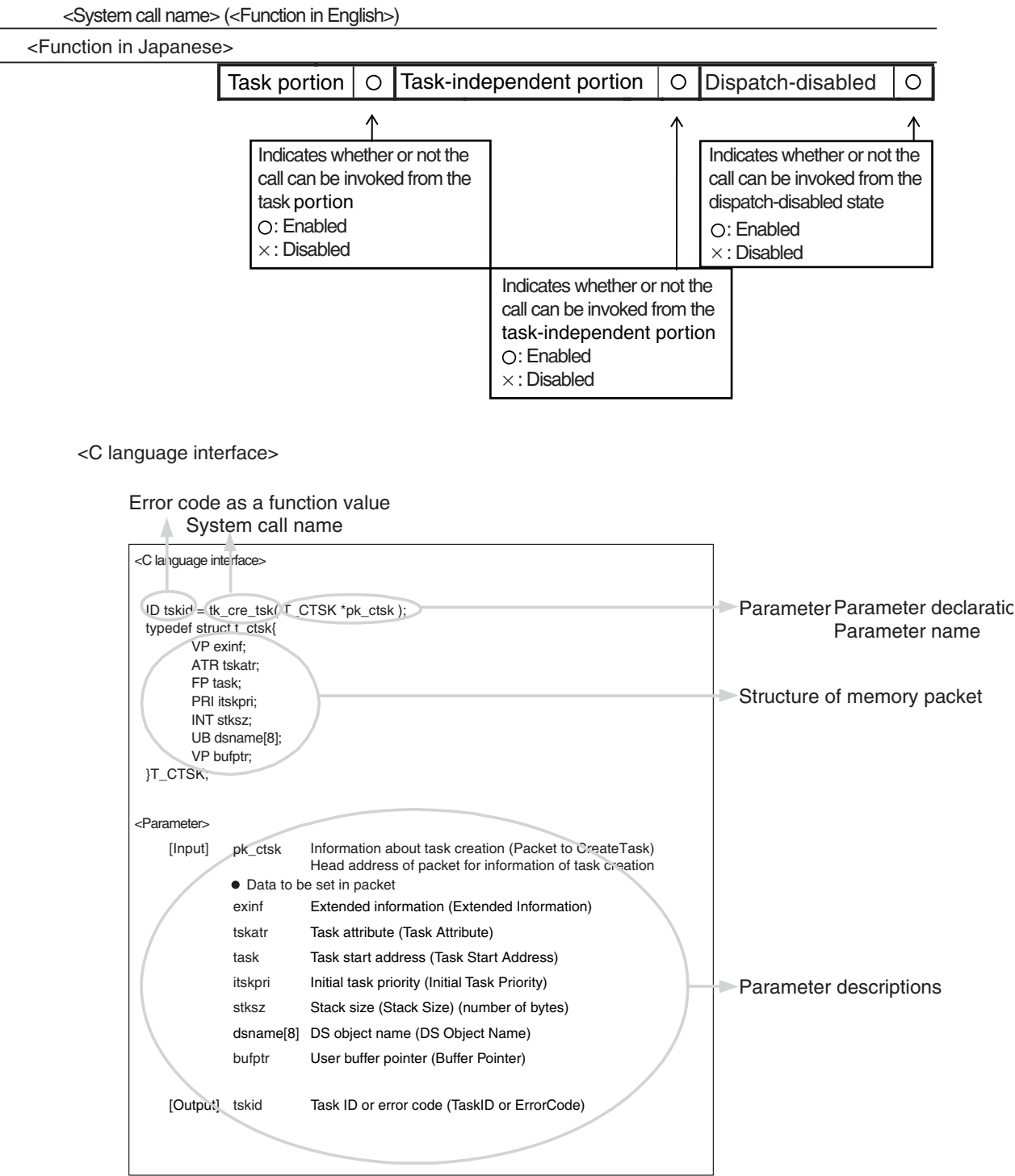
*2: E_CTX error occurs, if tk_sus_tsk is invoked by specifying a currently running task in the dispatch-disabled state.

*3: μT-REALOS-specific function (It is not defined in the μT-Kernel specifications; an additional function to μT-REALOS)

3.2 System Call Descriptions

This section explains how to read the system call descriptions in 3.3 System Calls for Task Management Function to 3.12 Device Management Function System Calls.

■ System Call Descriptions



<Error code>

<Errors to occur>		
E_NOMEM	-33	Insufficient memory (Management block and stack area cannot be secured)
E_LIMIT	-34	Number of tasks exceeded the system limit
E_RSATR	-11	"Reserved" attribute (tskatr is invalid or not available)
E_PAR	-17	Parameter error (pk_ctsk is invalid; task and bufptr are invalid; itskpri is invalid)

Errors that may occur
when this system call
is issued

<Dispatch trigger>

<Dispatch trigger>	
This system call does not perform dispatch.	

Trigger for this system call to
perform dispatch or the
availability of dispatch

<Explanation>

<Explanation>	
<p>It creates a task and assigns a task ID. To be more specific, it assigns TCB (Task Control Block) to the task to be created, and initializes it based on information such as itskpri, task and stksz.</p> <p>Once created, the target task enters the dormant state (DORMANT).</p> <p>itskpri is used to specify the initial value of the priority for startup of a task. The task priority can be specified with a value ranging from 1 to 140. The smaller the number is, the higher the priority is.</p> <p>exinf can be used freely by the user to keep information about the target task. The information specified here will be passed onto the task as an initial parameter and also can be retrieved by tk_ref_tsk. Note that if you want a larger area to retain user information or want to change its content during the operation, you should secure sufficient memory for that purpose yourself and place the address of the memory packet in exinf. The OS ignores the data in exinf.</p> <p>tskatr describes a system attribute in the lower part and an implementation-specific attribute in the upper part. The system attribute part of tskatr is specified as shown below.</p>	

Explanation and notes
regarding the process of this
system call

3.3 System Calls for Task Management Function

This section explains the system calls for the task management function.

■ System Calls for the Task Management Function

The task management function consists of the following ten system calls:

- tk_cre_tsk (Create Task)
- tk_del_tsk (Delete Task)
- tk_sta_tsk (Start Task)
- tk_ext_tsk (Exit Task)
- tk_exd_tsk (Exit and Delete Task)
- tk_ter_tsk (Terminate Task)
- tk_chg_pri (Change Task Priority)
- tk_get_reg (Get Task Registers)
- tk_set_reg (Set Task Registers)
- tk_ref_tsk (Refer Task Status)

3.3.1 tk_cre_tsk (Create Task)

Creates a task.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID tskid = tk_cre_tsk ( T_CTSK *pk_ctsk ) ;
```

```
typedef struct t_ctsk {
    VP  exinfo;
    ATR tskatr;
    FP  task;
    PRI itskpri;
    INT stksz;
    VP  bufptr;
} T_CTSK;
```

■ Parameter

- Input

pk_ctsk Information about task creation (Packet of Create Task)

Head address of the packet for information about task creation

- Data to set in packet

exinf Extended Information

tskattr	Task attribute (Task Attribute)
---------	---------------------------------

$$\text{tskatr} := \text{TA_HLNG} \mid [\text{TA_USERBUF}]$$

Attribute	Value	Meaning
TA_HLNG	0x00000001	Write the target task in C language
TA_USERBUF	0x00000020	Use the stack area specified by the user

task	Task start address (Task Start Address)
task1	0x00000000
task2	0x00000001
task3	0x00000002
task4	0x00000003
task5	0x00000004
task6	0x00000005
task7	0x00000006
task8	0x00000007
task9	0x00000008
task10	0x00000009
task11	0x0000000A
task12	0x0000000B
task13	0x0000000C
task14	0x0000000D
task15	0x0000000E
task16	0x0000000F
task17	0x00000010
task18	0x00000011
task19	0x00000012
task20	0x00000013
task21	0x00000014
task22	0x00000015
task23	0x00000016
task24	0x00000017
task25	0x00000018
task26	0x00000019
task27	0x0000001A
task28	0x0000001B
task29	0x0000001C
task30	0x0000001D
task31	0x0000001E
task32	0x0000001F
task33	0x00000020
task34	0x00000021
task35	0x00000022
task36	0x00000023
task37	0x00000024
task38	0x00000025
task39	0x00000026
task40	0x00000027
task41	0x00000028
task42	0x00000029
task43	0x0000002A
task44	0x0000002B
task45	0x0000002C
task46	0x0000002D
task47	0x0000002E
task48	0x0000002F
task49	0x00000030
task50	0x00000031
task51	0x00000032
task52	0x00000033
task53	0x00000034
task54	0x00000035
task55	0x00000036
task56	0x00000037
task57	0x00000038
task58	0x00000039
task59	0x0000003A
task60	0x0000003B
task61	0x0000003C
task62	0x0000003D
task63	0x0000003E
task64	0x0000003F
task65	0x00000040
task66	0x00000041
task67	0x00000042
task68	0x00000043
task69	0x00000044
task70	0x00000045
task71	0x00000046
task72	0x00000047
task73	0x00000048
task74	0x00000049
task75	0x0000004A
task76	0x0000004B
task77	0x0000004C
task78	0x0000004D
task79	0x0000004E
task80	0x0000004F
task81	0x00000050
task82	0x00000051
task83	0x00000052
task84	0x00000053
task85	0x00000054
task86	0x00000055
task87	0x00000056
task88	0x00000057
task89	0x00000058
task90	0x00000059
task91	0x0000005A
task92	0x0000005B
task93	0x0000005C
task94	0x0000005D
task95	0x0000005E
task96	0x0000005F
task97	0x00000060
task98	0x00000061
task99	0x00000062
task100	0x00000063
task101	0x00000064
task102	0x00000065
task103	0x00000066
task104	0x00000067
task105	0x00000068
task106	0x00000069
task107	0x0000006A
task108	0x0000006B
task109	0x0000006C
task110	0x0000006D
task111	0x0000006E
task112	0x0000006F
task113	0x00000070
task114	0x00000071
task115	0x00000072
task116	0x00000073
task117	0x00000074
task118	0x00000075
task119	0x00000076
task120	0x00000077
task121	0x00000078
task122	0x00000079
task123	0x0000007A
task124	0x0000007B
task125	0x0000007C
task126	0x0000007D

itskpri	Initial task priority (Initial Task Priority)
---------	---

stksz	Stack size (number of bytes) (Stack Size)
-------	---

bufptr	Address of user buffer (Buffer Pointer)
--------	---

● Output

tskid Task ID (TaskID) or Error Code (ErrorCode)

■ Error Code

E_NOMEM	- 33	Insufficient memory (Task stack area cannot be secured)
E_LIMIT	- 34	The number of tasks exceeds the system limit.
E_RSATR	- 11	"Reserved" attribute (Undefined value set in tskatr)
E_PAR	- 17	Parameter error (itskpri is 0 or smaller, or larger than the upper limit of the system; stksz is smaller than the minimum stack size, or stksz is not in multiples of 4. (TA_USEKBUF)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

It creates a task and assigns a task ID. To be more specific, it initializes a task to be created according to the information in itskpri, tsk and stksz. Once created, the task enters dormant state.

itskpri is used to specify the initial value of the priority for startup of a task. The task priority can be specified with a value ranging from 1 to the highest value allowed for the system (highest priority set in the configurator). The smaller the number is, the higher the priority is. It returns error E_PAR if itskpri is 0 or smaller, or larger than the highest priority allowed for the system.

Exinf can be used freely by the user to keep information about the target task. The information specified here will be passed onto the task as an initial parameter and also can be retrieved by tk_ref_tsk. Note that If you need a larger area to retain user information or wish to change its content during the operation, you should secure sufficient memory for that purpose by using a user program and place the address of the memory packet in exinf. The OS ignores the data in exinf.

Tskatr is used to specify an attribute for the task. It ignores the following values. Also, it returns error E_RSATR if an undefined attribute is specified in tskatr.

Attribute	Value	Meaning
TA_ASM	0x00000000	Object task is written in assembler.
TA_RNG0	0x00000000	Execute at protection level 0
TA_RNG1	0x00000100	Executed at protection level 1
TA_RNG2	0x00000200	Executed at protection level 2
TA_RNG3	0x00000300	Executed at protection level 3

The μ T-Kernel specifications define that TA_HLNG must be set when a task is written in C language. μ T-REALOS, however, only supports C language for writing tasks. Therefore, tasks are treated as being written in C language, even when TA_ASM is specified (or when TA_HLNG is not specified). Accordingly, although specifying TA_HLNG is not mandatory, you should always specify TA_HLNG in any case for compatibility with the μ T-Kernel specifications.

Each task has one stack. When TA_USERBUF is specified, bufptr becomes valid and stksz-byte memory space starting from bufptr is used as the stack area. In this case, the stack is not provided by the OS. If TA_USERBUF is not specified, bufptr is ignored and the OS secures the stack area. When OS secures it, stksz is obtained by rounding to multiples of 8 bytes.

It returns error E_NOMEM if the memory pool area on the kernel does not have enough free space to secure the stack area. If stksz is smaller than the minimum stack size (80 bytes) or if TA_VSEKBUF is set in tskatr but the value is not in multiples of 4, it returns error E_PAR. It returns error E_LIMIT if this system call is invoked when tasks have already been created up to the upper limit of the system (the maximum number of tasks set in the configurator). Also, even if pk_ctk, tsk or bufptr is invalid, no error check is performed and the operation is not guaranteed.

■ Additional Notes

The task stack for μ T-REALOS requires at least 80 bytes of space to save registers. For information on how to calculate the task stack size and how to write tasks, see "4.9 Tasks" in the "User's Guide".

When a task is terminated with a simple return from a function, the succeeding operation is not guaranteed. Always end a task by invoking tk_ext_tsk or tk_exd_tsk.

3.3.2 tk_del_tsk (Delete Task)

Deletes a task.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_tsk ( ID tskid ) ;
```

■ Parameter

● Input

tskid Task ID

● Output

ercd Error code (ErrorCode)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is less than or equal to zero, or larger than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Target task is not in the dormant state)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

It deletes the task specified by tskid. To be more specific, it moves the task specified by tskid from the dormant state to the unregistered state and returns the stack area and task ID number to the unused state.

Returns error E_ID if the task ID specified by tskid is less than or equal to zero, or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). Returns with error E_NOEXS if the task specified by the task ID in tskid does not exist.

This system call cannot specify its invoking task. If its invoking task is specified, it returns error E_OBJ, as the invoking task is not in the dormant state. To delete the invoking task, invoke tk_exd_tsk.

3.3.3 tk_sta_tsk (Start Task)

Starts a task.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_sta_tsk ( ID tskid, INT stacd ) ;
```

■ Parameter

● Input

tskid	Task ID
stacd	Task start code (Start Code)

● Output

ercd	Error code (ErrorCode)
------	------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is less than or equal to zero, or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Target task is not in the dormant state)

■ Dispatch Trigger

When a task with a priority higher than that of the task that invoked this system call is started, it dispatched to the started task.

■ Description

It starts the task indicated by tskid. To be more specific, it moves the task from the dormant state to the executable state.

It returns error E_ID if the task ID specified by tskid is less than or equal to zero, or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error E_NOEXS if the task specified by the task ID in tskid does not exist.

Stacd allows you to set a parameter to be passed onto the task during its startup. This parameter can be referenced from the target task and it can be used for simple message communication.

The task priority to be used at the startup of a task is the priority that was specified when the target task was created.

This system call does not perform the queuing of start requests. In other words, if this system call is invoked to target a task which is not in the dormant state, it returns error E_OBJ.

3.3.4 tk_ext_tsk (Exit Task)

Exits its invoking task.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ **C Language Interface**

```
void tk_ext_tsk ( void ) ;
```

■ **Parameter**

● Input

None

● Output

None

Note: It does not return to the context that invoked the system call.

■ **Error Code**

Note: The following error may be detected. Even in that case, it does not return to the context that invoked the system call. Therefore, no error code can be returned directly as the return value for the system call.

E_CTX	-25	Context error
-------	-----	---------------

(Executed from a task-independent portion or in the dispatch disabled state)

■ **Dispatch Trigger**

When the task that invoked this system call is completed, it is then dispatched to the task with the next highest priority.

■ **Description**

It completes its invoking task normally and places it into the dormant state.

When this system call makes the task dormant, its task priority returns to the priority used at startup.

■ **Additional Notes**

When a task is completed by this system call, the resources (memory block, semaphore, etc.) obtained by the terminated task up to that point is not released automatically. For this reason, any resources which will remain unused upon the completion of the task must be released prior to the completion.

This system call does not return to the context of the invoker. An error is detected in the OS, if this system call is invoked in a task-independent portion or in the dispatch-disabled state. Even in that case, however, it does not return to the invoker; therefore, the succeeding operation is not guaranteed.

3.3.5 tk_exd_tsk (Exit and Delete Task)

Exits and deletes its invoking task.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
void tk_exd_tsk ( void ) ;
```

■ Parameter

● Input

None

● Output

None

Note: It does not return to the context that invoked the system call.

■ Error Code

Note: The following error may be detected. Even in that case, it does not return to the context that invoked the system call. Therefore, no error code can be returned directly as the return value for the system call.

E_CTX -25 Context error

(Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that invoked this system call is completed, it is dispatched to the task with the next highest priority.

■ Description

It completes its invoking task normally, and then deletes it. To be more specific, it places its invoking task into the unregistered state.

■ Additional Notes

When a task is completed by this system call, the resources (memory block, semaphore, etc.) obtained by the terminated task up to that point is not released automatically. For this reason, any resources which will remain unused upon the completion of the task must be released prior to the completion.

This system call does not return to the context of the invoker. An error is detected in the OS, if this system call is invoked in a task-independent portion or in the dispatch-disabled state. Even in that case, however, it does not return to the invoker; therefore, the succeeding operation is not guaranteed.

3.3.6 tk_ter_tsk (Terminate Task)

Forcibly terminates other tasks.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_ter_tsk ( ID tskid ) ;
```

■ **Parameter**

● **Input**

tskid Task ID

● **Output**

ercd Error code (Error Code)

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is less than or equal to zero, or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Target task is in the dormant state or its invoking task)

■ **Dispatch Trigger**

When this system call forcibly terminates the task at the front of a waiting queue for the acquisition of a semaphore, the transmission of a message buffer or the acquisition of the memory block for a variable-length memory pool, the wait state of the following task may be released. If the priority of the task which has just been released from the wait state is higher than that of the task that invoked this system call, it is dispatched to the task which has just been released from the wait state.

■ **Description**

It forcibly terminates the task indicated by tskid. To be more specific, it places the target task indicated by tskid into the dormant state. When the task returns to the dormant state, the priority for startup is restored.

It returns error E_ID if the task ID specified by tskid is less than or equal to zero, or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error E_NOEXS if the task specified by the task ID in tskid does not exist.

Even when the target task is in the wait state (including the suspend state), it is terminated as being released from the wait state. In addition, if the target task is linked to any wait queue (such as for a semaphore), it is removed from that wait queue upon the execution of this system call. This system call cannot specify its invoking task. It returns error E_OBJ, if the invoking task is specified.

Table 3.3-1 summarizes the relationship between the state of the task targeted by this system call and its execution result.

Table 3.3-1 State of Task Targeted by tk_ter_tsk and Execution Result

State of target task	ercd of tk_ter_tsk	Process
Executable state (RUNNING, READY), (except invoking task)	E_OK	Forcible termination
Executing state (RUNNING), (invoking task)	E_OBJ	No operation
Wait state (WAITING, SUSPENDED, WAITING-SUSPENDED)	E_OK	Forcible termination
Dormant state (DORMANT)	E_OBJ	No operation
Unregistered state (NON-EXISTENT)	E_NOEXS	No operation

■ Additional Notes

When a task is completed by tk_ter_tsk, the resources (memory block, semaphore, etc.) obtained by the terminated task up to that point is not released automatically. For this reason, the resources obtained by that task must be released by the user program when terminating the task by tk_ter_tsk.

tk_ter_tsk forcibly terminates the target task, regardless of its execution state; therefore, it may negatively impact the whole system. For this reason, care must be taken when forcibly terminating a task.

3.3.7 tk_chg_pri (Change Task Priority)

Changes the task priority.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_chg_pri ( ID tskid, PRI tskpri ) ;
```

■ Parameter

● Input

tskid Task ID (Task ID)

The following macro can be specified in addition to the values from 1 to the maximum priority number.

Name	Value	Meaning
TSK_SELF	0	invoking task

tskpri Priority (Task Priority)

The following macro can be specified in addition to the values from 1 to the maximum priority number.

Name	Value	Meaning
TPRI_INI	0	Priority for startup

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks, or called from a task-independent portion with TSK_SELF (=0) specified in tskid)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_PAR	-17	Parameter error (tskpri is negative or greater than the maximum priority allowed for the system)
E_ILUSE	-28	Invalid use (upper priority limit violated)

■ Dispatch Trigger

It is dispatched to the target task, when the priority of the target task is set higher than that of its invoking task. When the priority of its invoking task is set lower than those of other tasks which are ready to be executed, it is dispatched to the task with the highest priority among the other tasks.

When a change is made to the priority of the tasks that exist in a waiting queue for the acquisition of a semaphore, the transmission of a message buffer or the acquisition of the memory block for a variable-length memory pool, the waiting order may be changed and the target task may be released from the wait state. If the priority of the target task is higher than that of the task which invoked this system call, it is dispatched to the target task.

■ Description

It changes the base priority of the task specified by `tskid` to the value specified by `tskpri`. When the base priority matches the current priority (this condition is always met when the mutex function is not used), it also changes the current priority of the task to the specified value so that it matches the base priority. For the task priority, `tskpri` can specify any value ranging from 1 to the maximum priority allowed for the system (the maximum priority set in the configurator). The smaller the value is, the higher the priority is.

It returns error `E_PAR`, if `tskpri` is negative or greater than the maximum priority allowed for the system. Returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). Returns with error `E_NOEXS` if the task specified by the task ID in `tskid` does not exist.

When `TSK_SELF(=0)` is specified, it handles its invoking task as the target task. Note however that if `TSK_SELF` is specified in `tskid` by the system call invoked from a task-independent portion, it returns error `E_ID`. In addition, when `TPRI_INI(=0)` is specified in `tskpri`, it changes the base priority of the target task to the priority for the startup of the task.

The priority changed by this system call remains valid until the completion of the task. When the task returns to the dormant state, the priority for startup of the task is restored to the value specified when the task is generated. Note however that the priority that was changed while the task was already in the dormant state becomes valid and the updated priority is used when the task is started next time.

The following process is executed, when the current priority of the target task matches the base priority upon the execution of this system call.

- If the target task is executable, the task priority order is changed according to the updated priority. The priority order for the target task is regarded as the lowest among tasks with the same priority as the updated priority.
- When the target task is linked to a waiting queue which is in a particular task priority order, it also modifies the order within the waiting queue, according to the updated priority. If there are tasks with the same priority as the updated priority, the target task is linked to the end of the queue.

It returns error `E_ILUSE`, if the base priority specified by `tskpri` is higher than the maximum priority of any of the mutexes for the `TA_CEILING` attribute which the target task has locked or is waiting to lock. For mutexes, see "3.5.1 Mutex Function" in the "User's Guide" and "3.6.1.1 `tk_cre_mtx` (Create Mutex)" to "3.6.1.5 `tk_ref_mtx` (Refer Mutex Status)" in this document.

■ Additional Notes

If invoking this system call results in a change in the order within a waiting queue that follows the task priority order of the target task, the target task or another task waiting in that queue may be released from the wait state (a waiting queue for the acquisition of a semaphore, the transmission of a message buffer, or the acquisition of a variable-length memory pool).

If the target task is waiting for the mutexes of the TA_INHERIT attribute to be locked, changing the base priority by this system call may require a progressive succession process of the priority.

When the mutex function is not used, the execution order of the invoking task becomes the lowest among the tasks with the same priority, if this system call is invoked by specifying the base priority of the invoking task as the updated priority. Therefore, the execution right can be abandoned by using this system call.

3.3.8 tk_get_reg (Get Task Registers)

Gets a task register.

Task portion	○	Task-independent portion	×	Dispatch disabled	○
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_get_reg ( ID tskid, T_REGS *pk_regs, T_EIT *pk_eit , T_CREGS *pk_cregs ) ;
```

```
typedef struct t_regs {
    VP pc;
    UW ps;
    VP rp;
    VW mdl;
    VW mdh;
    VW r[15];
} T_REGS;
```

```
typedef struct t_eit {
    VP pc;
    UW ps;
} T_EIT;
```

```
typedef struct t_cregs {
    VP usp;
} T_CREGS;
```

■ Parameter

● Input

tskid	Task ID
pk_regs	Packet address that stores the value set in the general-purpose register (Packet of Registers)

● Data to set in packet

pc	PC register
ps	PS register
rp	RP register
mdl	MDL register
mdh	MDH register
r[15]	General-purpose registers R0-R14
pk_eit	Packet address that stores the register value to be temporarily saved in case of CPU exception (Packet of EIT)

● Data to set in packet

pc	PC register
ps	PS register
pk_cregs	Packet address that stores the value set in the control register (Packet of Control Registers)

● Data to set in packet

usp	User stack pointer
-----	--------------------

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (target task = invoking task)
E_CTX	-25	Context error (invoked from a task-independent portion)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

It references the content of the current register of the task by tskid.

It returns error E_ID if the task ID specified by tskid is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error E_NOEXS if the task specified by tskid does not exist. Also, it returns error E_OBJ, if this system call is invoked for its invoking task.

It does not reference the corresponding register, when NULL is specified in pk_regs, pk_eit and pk_cregs. The content of pc and ps for pk_eit to be returned is the same value as pc and ps for pk_regs.

The referenced register value is the current register value of the target task; therefore, it is not always necessarily the one which is being executed in a task portion.

When this system call is invoked from a task-independent portion, it returns error E_CTX.

pk_regs, pk_eit, Even if pk_cregs is invalid, no error check is performed and the operation is not guaranteed.

3.3.9 tk_set_reg (Set Task Registers)

Sets a task register.

Task portion	<input type="radio"/>	Task-independent portion	<input checked="" type="checkbox"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-------------------------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_set_reg ( ID tskid, T_REGS *pk_regs, T_EIT *pk_eit , T_CREGS *pk_cregs ) ;
```

```
typedef struct t_regs {
    VP pc;
    UW ps;
    VP rp;
    VW mdl;
    VW mdh;
    VW r[15];
} T_REGS;
```

```
typedef struct t_eit {
    VP pc;
    UW ps;
} T_EIT;
```

```
typedef struct t_cregs {
    VP usp;
} T_CREGS;
```

■ Parameter

● Input

tskid	Task ID
pk_regs	Packet address that stores the value to be set in the general-purpose register (Packet of Registers)

● Data to set in packet

pc	PC register
ps	PS register
rp	RP register
mdl	MDL register
mdh	MDH register
r[15]	General-purpose registers R0-R14
pk_eit	Packet address that stores the value to be set in the register which is saved in case of CPU exception (Packet of EIT)

● Data to set in packet

pc	PC register
ps	PS register
pk_cregs	Packet address that stores the value to be set in the control register (Packet of Control Registers)

● Data to set in packet

usp	User stack pointer
-----	--------------------

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (target task = invoking task)
E_CTX	-25	Context error (invoked from a task-independent portion)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

It sets the tskid task register to specified content.

It returns error E_ID if the task ID specified by tskid is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error E_NOEXS if the task specified by tskid does not exist. When this system call is invoked for its invoking task, it returns error E_OBJ.

If NULL is specified in pk_regs, pk_eit and pk_cregs, the corresponding register is not set. If different values are specified in pc and ps between pk_regs and pk_eit, the pc and ps values of pk_eit become valid.

The register values are not verified. Therefore, if an invalid register value is set, the system may malfunction or hang up.

When this system call is invoked from a task-independent portion, it returns error E_CTX.

Even if pk_regs, pk_eit or pk_cregs is invalid, no error check is performed and the operation is not guaranteed.

3.3.10 tk_ref_tsk (Refer Task Status)

References the task state.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_tsk ( ID tskid, T_RTsk *pk_rtsk ) ;
```

```
typedef struct t_rtsk {
    VP      exinf;
    PRI      tskpri;
    PRI      tskbpri;
    UINT     tsksstat;
    UW       tskswait;
    ID       wid;
    INT      wupcnt;
    INT      suscnt;
} T_RTsk;
```

■ Parameter

● Input

tskid Task ID

The following macro can be specified in addition to the values from 1 to the maximum task number.

Name	Value	Meaning
TSK_SELF	0	Invoking task

*pk_rtsk Packet address that returns the task state
(Packet of Refer Task)

● Data to set in packet

exinf Extended Information
tskpri Current priority (Task Priority)
tskbpri Base priority (Task Base Priority)
tsksstat Task state (Task State)

State	Value	Meaning
TTS_RUN	0x00000001	Execution state (RUNNING)
TTS_RDY	0x00000002	Executable state (READY)
TTS_WAI	0x00000004	Wait state (WAITING)
TTS_SUS	0x00000008	Suspend state (SUSPENDED)
TTS_WAS	0x0000000c	Wait-suspend states (WAITING-SUSPENDED)
TTS_DMT	0x00000010	Dormant state (DORMANT)

`tskwait` Wait factor (Task Wait Factor)

Wait factor	Value	Meaning
TTW_SLP	0x00000001	Wait by <code>tk_slp_tsk</code>
TTW_DLY	0x00000002	Wait by <code>tk_dly_tsk</code>
TTW_SEM	0x00000004	Wait by <code>tk_wai_sem</code>
TTW_FLG	0x00000008	Wait by <code>tk_wai_flg</code>
TTW_MBX	0x00000040	Wait by <code>tk_rcv_mbx</code>
TTW_MTX	0x00000080	Wait by <code>tk_loc_mtx</code>
TTW_SMBF	0x00000100	Wait by <code>tk_snd_mbf</code>
TTW_RMBF	0x00000200	Wait by <code>tk_rcv_mbf</code>
TTW_CAL	0x00000400	Wait for rendezvous call
TTW_ACP	0x00000800	Wait for acceptance of rendezvous
TTW_RDV	0x00001000	Wait for completion of rendezvous
(TTW_CAL TTW_RDV)	0x00001400	Wait for rendezvous call or completion
TTW_MPF	0x00002000	Wait by <code>tk_get_mpf</code>
TTW_MPL	0x00004000	Wait by <code>tk_get_mpl</code>

`wid` Waiting object ID (Waiting Object ID)

`wupcnt` Number of queued wakeup requests (Wakeup Count)

`suscnt` Number of suspend request nests (Suspend Count)

● Output

`ercd` Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks, or called from a task-independent portion with TSK_SELF (=0) specified in tskid)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

References the state of the target task indicated by tskid.

Its invoking task can be specified by setting TSK_SELF(=0) in tskid. If TSK_SELF is specified in tskid by the system call invoked from a task-independent portion, it returns error E_ID. It returns with error E_NOEXS if the task specified by tskid does not exist. It returns error E_ID if the task ID specified by tskid is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator).

The state of the task is set to tskstat. If this system call is invoked for the interrupted task from the interrupt handler, the execution state (TTS_RUN) is returned as tskstat.

When tskstat is set to TTS_WAI (including TTS_WAS), tskwait and wid take values shown in Table 3.3-2.

Table 3.3-2 Value for tskwait and wid

tskwait	wid
TTW_SLP	0
TTW_DLY	0
TTW_SEM	semid for waiting
TTW_FLG	flgid for waiting
TTW_MBX	mbxid for waiting
TTW_MTX	mtxid for waiting
TTW_SMBF	mbfid for waiting
TTW_RMBF	mbfid for waiting
TTW_CAL	porid for waiting
TTW_ACP	porid for waiting
TTW_RDV	0
(TTW_CAL/TTW_RDV)	0
TTW_MPF	mpfid for waiting
TTW_MPL	mplid for waiting

Unless `tskstat` is set to `TTS_WAI` (including `TTS_WAS`), `tskwait` and `wid` are both 0. Also, for tasks in the dormant state, `wupcnt` and `suscnt` are both 0.

Even if `pk_rtsk` is invalid, no error check is performed and the operation is not guaranteed.

■ Additional Notes

This system call cannot be used to identify its invoking task ID. `tk_get_tid` should be used to identify its invoking task ID.

In addition, if this system call is invoked from a task-independent portion after a system call, which changes the task state such as `tk_svs_tsk` and `tk_ter_tsk`, is invoked to a task that has been in execution before the task-independent portion is called, `tskstat` remains in the execution state.

3.4 System Calls for Task-dependent Synchronization Function

This section explains the system calls for the task-dependent synchronization function.

■ System Calls for Task-dependent Synchronization Function

The task-dependent synchronization function consists of the following eight system calls.

- tk_slp_tsk (Sleep Task)
- tk_wup_tsk (Wakeup Task)
- tk_can_wup (Cancel Wakeup Task)
- tk_rel_wai (Release Wait)
- tk_sus_tsk (Suspend Task)
- tk_rsm_tsk (Resume Task)
- tk_frsm_tsk (Force Resume Task)
- tk_dly_tsk (Delay Task)

3.4.1 tk_slp_tsk (Sleep Task)

Moves its invoking task to the wakeup wait state.

Task portion	<input type="radio"/>	Task-independent portion	<input checked="" type="checkbox"/>	Dispatch disabled	<input checked="" type="checkbox"/>
--------------	-----------------------	--------------------------	-------------------------------------	-------------------	-------------------------------------

■ C Language Interface

```
ER ercd = tk_slp_tsk ( TMO tmout ) ;
```

■ Parameter

● Input

`tmout` Specifies the timeout

The following macros can be specified in addition to the values from 0 to 0x7ffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

`ercd` Error code (Error Code)

■ Error Code

<code>E_OK</code>	0	Normal completion
<code>E_PAR</code>	-17	Parameter error ($tmout \leq (-2)$)
<code>E_RLWAI</code>	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
<code>E_TMOUT</code>	-50	Time-out (Wait state released due to the passage of time)
<code>E_CTX</code>	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that invoked this system call enters the wait state, it is dispatched to the task with the next highest priority.

■ Description

When the number of queued wakeup requests is 0, this system call moves its invoking task from the running state to the wakeup wait state (the state waiting for `tk_wup_tsk`).

If `tk_wup_tsk` that targets this task is invoked before the time specified by `tmout` elapses, this system call is completed normally. On the other hand, if `tk_wup_tsk` or `tk_rel_wai` that targets this task is not invoked before the time specified by `tmout` elapses, it returns error `E_TMOUT`.

`TMO_FEVR` means an infinite timeout. In this case, it remains in the wait state, until `tk_wup_tsk` or `tk_rel_wai` is invoked. If `TMO_POL` is specified, the task does not enter the wait state and it returns error `E_TMOUT`.

The time unit for `tmout` is the same as for the system timer (= 1ms). If `tmout` is -2 or less, the function returns with an `E_PAR` error.

When `tk_rel_wai` is invoked for a task in the wait state using this system call, it returns error `E_RLWAI` and releases the wait state. When this system call is invoked in a task-independent portion or in the dispatch-disabled state, it returns error `E_CTX`.

When the number of queued wakeup requests is larger than 0, this system call reduces that number by 1 but does not enter the wait time, and continues processing the task that invoked the system call.

■ Additional Notes

If another task invokes `tk_sus_tsk` to the task that has been placed in the wait state by this system call, the latter task enters the wait-suspend states.

3.4.2 tk_wup_tsk (Wakeup Task)

Wakes up another task.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_wup_tsk ( ID tskid ) ;
```

■ Parameter

● Input

tskid Task ID

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Target task is its invoking task or in the dormant state)
E_QOVR	-43	Queue or nesting overflow (Number of queued wakeup requests is larger than 32767)

■ Dispatch Trigger

When the task with a priority higher than that of the task that invoked this system call is woken up, it is dispatched to the task that has woken up.

■ Description

If the task specified by `tskid` is in the wait state due to a `tk_slp_tsk` call, that wait state is released.

This system call cannot be used to specify its invoking task in `tskid`. If its invoking task is specified, it returns error `E_OBJ`. It returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error `E_NOEXS` if the task specified by `tskid` does not exist.

If the target task is neither executing `tk_slp_tsk` nor in the wait state, a wakeup request from this system call is placed in the queue. Kernel manages the number of queued wakeup requests in units of tasks and its initial value (value upon the execution of `tk_sta_tsk`) is 0. When this system call is invoked for a task which is not in the wakeup wait state, the number of queued wakeup requests for the target task is increased by 1. When `tk_slp_tsk` is invoked, on the other hand, the number of queued wakeup requests for the target task is reduced by 1. If a task that has no queued wakeup requests invokes `tk_slp_tsk`, the task enters the wait state.

The maximum number of queued wakeup requests is 32767. If the maximum number is exceeded while invoking this system call, it returns error `E_QOVR`.

3.4.3 tk_can_wup (Cancel Wakeup Task)

Disables a wakeup request from a task.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
INT wupcnt = tk_can_wup ( ID tskid ) ;
```

■ **Parameter**

● **Input**

tskid Task ID

The following macro can be specified in addition to the values from 1 to the maximum task number.

Name	Value	Meaning
TSK_SELF	0	Invoking task

● **Output**

wupcnt Number of queued wakeup requests (Wakeup Count)
Or, error code (Error Code)

■ **Error Code**

E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks, or called from a task-independent portion with TSK_SELF (=0) specified in tskid)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Target task is in the dormant state)

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ Description

It returns the number of queued wakeup requests for the task indicated by `tskid` as the return value (`wupcnt`), and simultaneously cancels all of the wakeup requests. In other words, it sets the number of queued wakeup requests for the target task to 0.

If the invoking task is specified by the system call invoked from a task-independent portion, it returns error `E_ID`. It returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator), or if `TSK_SELF` is specified by an invocation from a task-independent portion. It returns with error `E_NOEXS` if the task specified by `tskid` does not exist.

■ Additional Notes

This system call can be used to determine whether the process is completed within the time when waking up a task periodically. In other words, it can be confirmed that the process for the previous request was not completed in time, when the monitoring task invokes `tk_can_wup` and the return value (`wupcnt`) is 1 or greater, before `tk_slp_tsk` is invoked upon the completion of the process for the previous wakeup request. Therefore, some kind of action can be taken for the processing delay.

3.4.4 tk_rel_wai (Release Wait)

Releases the wait state of other tasks.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_rel_wai ( ID tskid ) ;
```

■ Parameter

● Input

tskid Task ID (Task ID)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Task is in the running, ready, suspended, or dormant state)

■ Dispatch Trigger

If the wait state of the task with a priority higher than that of the task that invoked this system call is released, it is dispatched to the task which is released from the wait state.

When the wait state of the task to wait for the transmission of a message buffer or the acquisition of a variable-length memory pool is released and the priority of that task is higher than that of the task that invoked this system call, it is dispatched to the task which is released from the wait state.

■ Description

If the task specified by `tskid` is in the wait state, it is forcibly released.

If the task specified by `tskid` is not in the wait state, it returns error `E_OBJ` to the invoker. It returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error `E_NOEXS` if the task specified by `tskid` does not exist.

The task which is released from the wait state by this system call returns error `E_RLWAI`. This system call does not perform the queuing of wait state release requests. In other words, if the target task is already in the wait state, that wait state is released. Otherwise, it returns error `E_OBJ`. When this system call specifies its invoking task, it also returns error `E_OBJ`.

This system call does not release the suspend state. If this system call is invoked for a task in the wait-suspend states, the target task is released from the wait state and enters the suspend state. If it also needs to be released from the suspend state, `tk_rsm_tsk` or `tk_frm_tsk` should be invoked separately. Table 3.4-1 summarizes the relationship between the state of the task targeted by this system call and its execution result.

Table 3.4-1 State of Task Targeted by `tk_rel_wai` and Execution Result

State of target task	ercd	Process
Executable state (RUNNING, READY), (except invoking task)	<code>E_OBJ</code>	No operation
Executing state (RUNNING), (invoking task)	<code>E_OBJ</code>	No operation
Wait state (WAITING)	<code>E_OK</code>	Release wait state (Note 1)
Suspend state (SUSPENDED)	<code>E_OBJ</code>	No operation
Wait-suspend states (WAITING-SUSPENDED)	<code>E_OK</code>	Move to suspend state (Note 2)
Dormant state (DORMANT)	<code>E_OBJ</code>	No operation
Unregistered state (NON-EXISTENT)	<code>E_NOEXS</code>	No operation

■ Additional Notes

A function similar to a timeout can be realized, if a device such as an alarm handler is used to invoke this system call after a specified period of time passes once a certain task enters the wait time.

The following differences are found between this system call and `tk_wup_tsk`.

- `tk_wup_tsk` only releases the wait state caused by `tk_slp_tsk`. This system call, on the other hand, also releases the wait state caused by other factors (system calls such as `tk_wai_flg`, `tk_wai_sem`, and `tk_rcv_mbf`).
- As for the return value of the system call that has entered the wait state, releasing the wait state caused by `tk_wup_tsk` is regarded as normal completion (`E_OK`), while releasing the wait state caused by this system call is treated as an error (`E_RLWAI`).
- In case of `tk_wup_tsk`, requests are queued even before the target task executes `tk_slp_tsk`. This system call, on the other hand, returns error `E_OBJ`, unless the target task has already entered the wait state.

3.4.5 tk_sus_tsk (Suspend Task)

Places other tasks into the suspend state.



■ **C Language Interface**

```
ER ercd = tk_sus_tsk ( ID tskid ) ;
```

■ **Parameter**

● **Input**

tskid Task ID (Task ID)

● **Output**

ercd Error code (Error Code)

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Target task is its invoking task or in the dormant state)
E_CTX	-25	Task in executing state has been specified in the dispatch-disabled state
E_QOVR	-43	Queue or nesting overflow (Number of nests for suspend requests is greater than 32767)

■ **Dispatch Trigger**

In the dispatch-enabled state, when this system call is invoked from a task-independent portion to a task in the executing state, it is dispatched to the task with the next highest priority.

■ Description

It moves the task specified by `tskid` to the suspend state and temporarily suspends the execution of the task.

If the invoking task is specified in `tskid`, it returns error `E_OBJ`. It returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). It returns with error `E_NOEXS` if the task specified by `tskid` does not exist. If a task in the executing state is specified in `tskid` in the dispatch-disabled state when this system call is invoked from a task-independent portion or its invoking task, it returns error `E_CTX`.

The suspend state is released by invoking `tk_rsm_tsk` and `tk_frsm_tsk`. If the target task for this system call is already in the wait state, the execution of the system call puts the target task into the wait-suspend states: the regular wait state and the suspend state. Once the conditions for releasing the task from the wait state is satisfied, the target task enters the suspend state. On the other hand, if `tk_rsm_tsk` or `tk_frsm_tsk` is invoked for a task which is in the wait-suspend states, the target task returns to the same state as the previous wait state.

If this system call is invoked more than once for a certain task, the task enters multiple suspend states. This is called a nest of suspend requests. In this case, the target task returns to its original state, when the `tk_rsm_tsk` system call is invoked for the same number of times as this system call is invoked (`suscnt`), or when `tk_frsm_tsk` is invoked once. Therefore, this system call and the `tk_rsm_tsk` system call can be nested in pairs. The maximum number for nesting suspend requests is 32767.

■ Additional Notes

Even when a certain task is in the wait state to get resources (waiting for a semaphore, for example) as well as in the suspend state, resources are allocated (e.g. semaphore allocation) under the same conditions as for when it is not in the suspend state. Even in the suspend state, the allocation of resources is not delayed, and there is no difference in the conditions and priority for the allocation of resources and releasing of the wait state.

`tk_sus_tsk` puts a task into the suspend state regardless of the execution conditions of the target task. Therefore, it may negatively impact the whole system, if the process is suspended because the target task enters the suspend state. For this reason, care must be taken when placing a task in the suspend state.

3.4.6 tk_rsm_tsk (Resume Task)

Resumes a task which is in the suspend state.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_rsm_tsk ( ID tskid ) ;
```

■ **Parameter**

● **Input**

tskid Task ID (Task ID)

● **Output**

ercd Error code (Error Code)

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Task is in the running, ready, suspended, or dormant state)

■ **Dispatch Trigger**

If the suspend state of the task with a priority higher than that of the task that invoked this system call is released, it is dispatched to the task which is released from the suspend state.

■ Description

It releases the task specified by `tskid` from the suspend state. In other words, when the target task enters the suspend state due to the earlier invoked `tk_sus_tsk` and its execution is suspended, it releases the task from that state and resumes the execution.

If the invoking task is specified in `tskid`, it returns error `E_OBJ`. It also returns error `E_OBJ`, if the task specified by `tskid` is not in the suspend state. Returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). If the task specified by `tskid` does not exist, it returns error `E_NOEXS`.

If the target task is in the wait-suspend states (a combination of the regular wait state and the suspend state), the target task is released only from the suspend state upon the execution of this system call, and it remains in the regular wait state.

The number of times for nesting suspend requests is reduced by 1 in this system call. Therefore, if `tk_sus_tsk` has been invoked for the target task more than once, the target task remains in the suspend state even after the completion of this system call.

■ Additional Notes

When a task in the executing or executable state enters the suspend state because of `tk_sus_tsk`, and then the execution is resumed by this system call or `tk_frm_tsk`, that task gets the lowest priority of all tasks with the same priority. For example, the following operation is performed, when the following system call is executed to `task_A` and `task_B` which both have the same priority.

```
tk_sta_tsk (tskid=task_A, stacd_A);
```

```
tk_sta_tsk (tskid=task_B, stacd_B);
```

```
/* In this case, the priority order is based on the startup order: task_A → task_B */
```

```
tk_sus_tsk (tskid=task_A);
```

```
tk_rsm_tsk (tskid=task_A);
```

```
/* In this case, the priority order is: task_B → task_A */
```

3.4.7 tk_frsm_tsk (Force Resume Task)

Forcibly resumes a task that is in the suspend state.



■ **C Language Interface**

```
ER ercd = tk_frsm_tsk ( ID tskid ) ;
```

■ **Parameter**

● **Input**

tskid Task ID (Task ID)

● **Output**

ercd Error code (Error Code)

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (tskid is negative or greater than the maximum number of tasks)
E_NOEXS	-42	Object does not exist (The task specified by tskid does not exist)
E_OBJ	-41	Object state is invalid (Task is in the running, ready, suspended, or dormant state)

■ **Dispatch Trigger**

If the suspend state of the task with a priority higher than that of the task that invoked this system call is released, it is dispatched to the task which is released from the suspend state.

■ Description

It releases the task indicated by `tskid` from the suspend state. In other words, when the target task enters the suspend state due to the earlier invoked `tk_sus_tsk` and its execution is suspended, it releases the task from that state and resumes the execution.

If the invoking task is specified in `tskid`, it returns error `E_OBJ`. It also returns error `E_OBJ`, if the target task specified by `tskid` is not in the suspend state. It returns error `E_ID` if the task ID specified by `tskid` is negative or greater than the maximum number of tasks (maximum number of tasks specified in the configurator). If the task specified by `tskid` does not exist, it returns error `E_NOEXS`.

If the target task is in the wait-suspend states (a combination of the wait state and the suspend state), the target task is released only from the suspend state upon the execution of this system call, and it remains in the regular wait state.

Even if `tk_sus_tsk` has been invoked for the target task more than once,, all of the requests are released. In other words, the suspend state is always released, and the execution can be resumed, as long as the target task is not in the wait-suspend states.

■ Additional Notes

When a task in the executing or executable state enters the suspend state because of `tk_sus_tsk`, and then the execution is resumed by `tk_rsm_tsk` or this system call, that task gets the lowest priority of all tasks with the same priority. For example, the following operation is performed, when the following system call is executed to `task_A` and `task_B` which both have the same priority.

```
tk_sta_tsk (tskid=task_A, stacd_A);
```

```
tk_sta_tsk (tskid=task_B, stacd_B);
```

```
/* In this case, the priority order is based on the startup order: task_A → task_B */
```

```
tk_sus_tsk (tskid=task_A);
```

```
tk_frsm_tsk (tskid=task_A);
```

```
/* In this case, the priority order is: task_B → task_A */
```

3.4.8 tk_dly_tsk (Delay Task)

Places its invoking task into the delay wait state.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ **C Language Interface**

```
ER ercd = tk_dly_tsk ( RELTIM dlytim ) ;
```

■ **Parameter**

● **Input**

dlytim Delay time (Delay Time)

● **Output**

ercd Error code (Error Code)

■ **Error Code**

E_OK	0	Normal completion
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)
E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)

■ **Dispatch Trigger**

When the task that invoked this system call enters the delay wait state, it is dispatched to the task with the next highest priority.

■ **Description**

It temporarily suspends the execution of its invoking task and places it in the delay wait state {WAITING}. dlytim is used to specify the time when the execution of the task should be suspended. The base time for dlytim (time unit) is the same as for the system timer (= 1ms). When dlytim is set to 0, it performs no operation and returns E_OK.

It counts the time elapsed even while the task that invoked this system call is in the dual wait state.

If tk_rel_wai is invoked for the task that has been put in the wait state by this system call, the system call returns error E_RLWAI and releases the task from the wait state. If this system call is invoked in a task-independent portion or the dispatch-disabled state, it returns error E_CTX.

3.5 System Calls for Synchronization/Communication Function

This section explains the system calls for the synchronization/communication function.

■ System Calls for Synchronization/Communication Function

The synchronization/communication function consists of the following three types of function system calls.

- Semaphore function system calls
- Event flag function system calls
- Mailbox function system calls

3.5.1 Semaphore Function System Calls

This section explains the semaphore function system calls.

■ Semaphore Function System Calls

The semaphore function consists of the following five system calls.

- `tk_cre_sem` (Create Semaphore)
- `tk_del_sem` (Delete Semaphore)
- `tk_sig_sem` (Signal Semaphore)
- `tk_wai_sem` (Wait on Semaphore)
- `tk_ref_sem` (Refer Semaphore Status)

3.5.1.1 tk_cre_sem (Create Semaphore)

Creates a semaphore.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID semid = tk_cre_sem ( T_CSEM *pk_csem ) ;
```

```
typedef struct t_csem {
    VP exinf;
    ATR sematr;
    INT isemcnt;
    INT maxsem;
} T_CSEM;
```

■ Parameter

● Input

`pk_csem` Head address of the packet that passes information about semaphore creation
(Packet of Create Semaphore)

● Data to set in packet

`exinf` Extended Information

`sema`tr Semaphore attribute (Semaphore Attribute)
sematr:= (TA_TFIFO || TA_TPRI) | (TA_FIRST || TA_CNT)

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority
TA_FIRST	0x00000000	Gives higher priority to the task at the front of the waiting queue
TA_CNT	0x00000002	Gives higher priority to the task with the least requests

`isemcnt` Initial value of the semaphore count
(Initial Semaphore Count)

`maxsem` Maximum value of the semaphore count
(Maximum Semaphore Count)

● Output

<code>semid</code>	Semaphore ID
	Or, error code (Error Code)

■ Error Code

<code>E_LIMIT</code>	-34	The number of semaphores is greater than the upper limit of the system.
<code>E_RSATR</code>	-11	"Reserved" attribute (undefined value specified in <code>sematr</code>)
<code>E_PAR</code>	-17	Parameter error (<code>isemcnt</code> is negative or greater than <code>maxsem</code> ; <code>maxsem</code> is 0)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

It creates a semaphore and assigns a semaphore ID number. To be more specific, it creates a semaphore with `isemcnt` as its initial value and `maxsem` as its maximum value (upper limit). When `maxsem` is set to a value equivalent of or smaller than 0, or `isemcnt` is set to a negative value or a value greater than `imaxsem`, it returns error `E_PAR`.

The user can use `exinf` freely to place information regarding the target semaphore. The information specified here can be retrieved by `tk_ref_sem`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS ignores the data in `exinf`.

You specify a semaphore attribute in `sematr`. If an undefined attribute is specified in `sematr`, it returns error `E_RSATR`.

`TA_TFIFO`, `TA_TPRI` allows you to specify how tasks should be arranged in the waiting queue for the semaphore. When the attribute is `TA_TFIFO`, the task waiting queue is FIFO. When the attribute is `TA_TPRI`, the task waiting queue is arranged according to the priority order of tasks.

`TA_FIRST` and `TA_CNT` are used to specify the priority order for the acquisition of resources. The order of the waiting queue cannot be changed by specifying `TA_FIRST` and `TA_CNT`. The order of the waiting queue can only be determined by `TA_TFIFO` and `TA_TPRI`.

`TA_FIRST` is used to allocate resources to tasks, starting from the task at the front of the waiting queue, regardless of the request count. Until the task at the front of the waiting queue gets a requested number of resources, the following tasks in the queue cannot get resources.

`TA_CNT` is used to allocate resources to tasks, starting from the task which can get a requested number of resources. To be more specific, it checks the request count of each task, one by one, starting from the task at the front of the waiting queue, and then, allocates a requested number of resources to tasks to which it can allocate the requested number of resources. Therefore, it does not allocate resources to tasks, starting from the one with the smallest request count.

If this system call is invoked when semaphores have been created to the upper limit of the system (maximum number of semaphores set in the configurator), it returns error `E_LIMIT`. Even if `pk_csem` is invalid, no error check is performed and the operation is not guaranteed.

3.5.1.2 tk_del_sem (Delete Semaphore)

Deletes a semaphore.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_sem ( ID semid ) ;
```

■ Parameter

● Input

semid	Semaphore ID (Semaphore ID)
-------	-----------------------------

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is less than or equal to zero, or greater than the maximum number of semaphores)
E_NOEXS	-42	Object does not exist (The semaphore specified by semid does not exist)

■ Dispatch Trigger

When a task with a priority higher than that of the task that invoked this system call is waiting in the semaphore to be deleted, it is dispatched to the task with the higher priority.

■ Description

It deletes the semaphore specified by semid. To be more specific, it moves the target semaphore to the uncreated state and releases the ID number.

It returns error E_ID if the semaphore ID specified by semid is less than or equal to zero, or greater than the maximum number of semaphores (maximum number of semaphores specified in the configurator). If the semaphore specified by semid does not exist, it returns error E_NOEXS.

This system call is also completed normally, when there is a task waiting in the target semaphore. The waiting task returns error E_DLT in tk_wai_sem and the wait state is released.

3.5.1.3 tk_sig_sem (Signal Semaphore)

Returns semaphore resources.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_sig_sem ( ID semid, INT cnt ) ;
```

■ **Parameter**

● **Input**

semid	Semaphore ID (Semaphore ID)
cnt	Number of resources returned (Signal Count)

● **Output**

ercd	Error code (Error Code)
------	-------------------------

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is less than or equal to zero, or greater than the maximum number of semaphores)
E_NOEXS	-42	Object does not exist (The semaphore specified by semid does not exist)
E_QOVR	-43	Queue or nesting overflow (Actual semaphore count is greater than the maximum semaphore count)
E_PAR	-17	Parameter error (cnt is 0 or smaller)

■ **Dispatch Trigger**

When the semaphore wait state of the task with a priority higher than that of the task that invoked this system call is released, it is dispatched to the task which is released from the wait state.

■ Description

It performs the operation to return the number of resources indicated in `cnt` to the semaphore specified by `semid`.

It returns error `E_ID` if the semaphore ID specified by `semid` is less than or equal to zero, or greater than the maximum number of semaphores (maximum number of semaphores specified in the configurator). If the target semaphore does not exist, it returns error `E_NOEXS`. If `cnt` is 0 or smaller, it returns error `E_PAR`.

If any task is already waiting in the target semaphore, it checks the request count and allocates resources, if available. It places the task to which resources have been allocated into the executable state. If resources more than the total resources requested by multiple tasks are returned, the resources are allocated to multiple tasks and the tasks become executable.

If the sum of a semaphore count value (`semcnt`) before this system call is invoked and `cnt` value is greater than the maximum value for semaphore count (`maxcnt`), it returns error `E_QOVR`. In that case, no resources are returned at all and also the count value (`semcnt`) does not change.

■ Additional Notes

It is also completed normally, when the semaphore count (`semcnt`) is larger than its initial value. To use a semaphore for synchronization purposes (same as `tk_wup_tsk` to `tk_slp_tsk`), rather than for exclusive control, the semaphore count (`semcnt`) must be larger than its initial value (`isemcnt`). To use a semaphore for exclusive control purposes, on the other hand, you can check for an error that may be caused by an increase in the count value, as long as the initial value of the semaphore count (`isemcnt`) is equivalent to the maximum semaphore count (`maxsem`).

3.5.1.4 tk_wai_sem (Wait on Semaphore)

Gets semaphore resources.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_wai_sem ( ID semid, INT cnt, TMO tmout ) ;
```

■ Parameter

● Input

semid	Semaphore ID (Semaphore ID)
cnt	Number of resource requests (Require Count)
tmout	Specifies the timeout (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7ffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is less than or equal to zero, or greater than the maximum number of semaphores)
E_NOEXS	-42	Object does not exist (The semaphore specified by semid does not exist)
E_PAR	-17	Parameter error (tmout ≤ (-2), cnt ≤ 0)
E_DLT	-51	Wait object has been deleted. (Target semaphore deleted while in the wait state)
E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that invoked this system call enters the semaphore wait state, it is dispatched to the task with the next highest priority.

■ Description

It gets the number of resources indicated in `cnt` from the semaphore specified by `semid`.

It returns error `E_ID`, if `semid` is 0 or smaller, or larger than the maximum number of semaphores (maximum number of semaphores set in the configurator). It returns error `E_NOEXS`, if the target semaphore does not exist. It returns error `E_PAR`, if `cnt` is 0 or smaller.

The maximum wait time (timeout) can be specified in `tmout`. It returns error `E_TMOUT` if the `tmout` time elapses before the wait release conditions are satisfied (or before `tk_sig_sem` is executed).

`TMO_FEVR` means an infinite timeout. In this case, it remains in the wait state until resources are obtained or `tk_rel_wai` is invoked. If `TMO_POL` is specified, the task does not enter the wait state and it returns error `E_TMOUT`.

The time unit for `tmout` is the same as for the system timer (= 1ms). If `tmout` is -2 or less, the function returns with an `E_PAR` error.

Once resources are obtained, the task that invoked this system call continues to run, rather than entering the wait state. In this case, the count value of that semaphore is deducted by `cnt`.

If resources cannot be obtained, the task that invoked this system call enters the wait state. In other words, it is linked to the wait queue for that semaphore. In this case, the count value of the semaphore remains unchanged.

If the target semaphore is deleted by `tk_del_sem` while still linked to the wait queue for the semaphore, the wait state is released and this system call returns error `E_DLT`. If `tk_rel_wai` is invoked for the task linked to the wait queue for the semaphore, the wait state is released and this system call returns error `E_RLWAI`. If this system call is invoked in a task-independent portion or the dispatch-disabled state, it returns error `E_CTX`.

3.5.1.5 tk_ref_sem (Refer Semaphore Status)

References the semaphore status.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_sem ( ID semid, T_RSEM *pk_rsem ) ;
```

```
typedef struct t_rsem {
    VP  exinf;
    ID  wtsk;
    INT semcnt;
} T_RSEM;
```

■ Parameter

● Input

semid	Semaphore ID (Semaphore ID)
pk_rsem	Packet address to which to return the semaphore status (Packet of Refer Semaphore)

● Data to set in packet

exinf	Extended Information (Extended Information)
wtsk	Waiting task present or not (Wait Task)
semcnt	Current semaphore count value (Semaphore Count)

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (semid is less than or equal to zero, or greater than the maximum number of semaphores)
E_NOEXS	-42	Object does not exist (The semaphore specified by semid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call references various types of status of the target semaphore specified by `semid` and returns the current semaphore count value (`semcnt`), waiting task presence/absence (`wtsk`), and extended information (`exinf`) as the return values.

The system call returns error `E_ID` if the semaphore ID specified by `semid` is less than or equal to zero, or greater than the maximum number of semaphores (maximum number of semaphores specified in the configurator). The system call returns error `E_NOEXS` if the target semaphore does not exist.

`wtsk` indicates the ID of the task waiting in the semaphore. When two or more tasks are waiting, the system call returns the ID of the first task in the queue. If there is no task waiting, `wtsk` is 0.

Even when `pk_rsem` is invalid, no error checking is performed, where operation is not guaranteed.

3.5.2 Event Flag Function System Calls

This section describes the event flag function system calls.

■ Event Flag Function System Calls

The event flag function consists of the following six system calls:

- `tk_cre_flg` (Create Event Flag)
- `tk_del_flg` (Delete Event Flag)
- `tk_set_flg` (Set Event Flag)
- `tk_clr_flg` (Clear Event Flag)
- `tk_wai_flg` (Wait Event Flag)
- `tk_ref_flg` (Refer Event Flag Status)

3.5.2.1 tk_cre_flg (Create Event Flag)

Creates an event flag.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID flgid = tk_cre_flg ( T_CFLG *pk_cflg ) ;
```

```
typedef struct t_cflg {
    VP      exinf;
    ATR      flgatr;
    UINT     iflgptn;
} T_CFLG;
```

■ Parameter

● Input

`pk_cflg` Information about event flag creation
(Packet of Create Event Flag)

● Data to set in packet

`exinf` Extended Information (Extended Information)
`flgatr` Event flag attribute (Event Flag Attribute)
`flgatr := (TA_TFIFO || TA_TPRI) | (TA_WMUL || TA_WSGL)`

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority
TA_WSGL	0x00000000	Disallows two or more tasks to wait.
TA_WMUL	0x00000008	Allows two or more tasks to wait.

`iflgptn` Initial value of event flag
(Initial Event Flag Pattern)

● Output

`flgid` Event flag ID (Event Flag ID)
Or, error code (Error Code)

■ Error Code

<code>E_LIMIT</code>	- 34	The number of event flags is greater than the system's upper limit.
<code>E_RSATR</code>	- 11	"Reserved" attribute (flgatr has been set to an undefined value.)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates an event flag with its event flag ID number assigned. Precisely, the system call creates an event flag with an initial value of iflgptn.

The user can freely use `exinf` to hold information on the target event flag. The information specified here can be taken out via `tk_ref_flg`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS ignores the data in `exinf`.

`flgatr` specifies the attribute of the event flag. If `flgatr` specifies an undefined attribute, the system call returns with an `E_RSATR` error.

Setting `flgatr` to `TA_WSGL` inhibits two or more tasks from entering the wait state at the same time. Setting the parameter to `TA_WMUL` allows two or more tasks to enter the wait state at the same time.

`TA_TFIFO` or `TA_TPRI` allows you to specify how tasks are placed in the queue of event flags. If the attribute is `TA_TFIFO`, the task wait queue operates as a FIFO. If the attribute is `TA_TPRI`, the task wait queue is ordered by task priority. If you specify `TA_WSGL`, however, no queue is created and thus `TA_TFIFO` and `TA_TPRI` make no difference in operation, whichever is specified.

When two or more tasks are waiting, the system checks whether their wait conditions have been satisfied, sequentially from the beginning of the queue, and dequeues the tasks whose wait conditions have been satisfied. Therefore, the first task in the queue is not always dequeued first. Two or more tasks are dequeued if their wait conditions have been satisfied.

This system call returns with an `E_LIMIT` error if issued while as many event flags as the system's upper limit (the maximum number of event flags specified in the configurator) have been created.

Even when `pk_cflg` is invalid, no error checking is performed, where operation is not guaranteed.

3.5.2.2 tk_del_flg (Delete Event Flag)

Deletes an event flag.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_flg ( ID flgid ) ;
```

■ Parameter

● Input

flgid Event flag ID (Event Flag ID)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is less than or equal to zero, or greater than the maximum number of flags)
E_NOEXS	-42	Object does not exist (The event flag specified in flgid does not exist)

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is waiting in an event flag, the system call dispatches control to the higher-priority task.

■ Description

This system call deletes the event flag specified by flgid. Precisely, the system call puts that event flag into the ungenerated state and unassigns the ID number.

The system call returns error E_ID if the event flag ID specified in flgid is less than or equal to zero, or greater than the maximum number of flags (maximum number of event flags specified in the configurator). The system call returns error E_NOEXS if the specified event flag does not exist.

When the specified event flag has a task waiting for its condition to be satisfied, this system call also terminates normally. For the waiting task, however, tk_wai_flg returns with an E_DLT error.

3.5.2.3 tk_set_flg (Set Event Flag)

Sets an event flag.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_set_flg ( ID flgid, UINT setptn ) ;
```

■ Parameter

● Input

flgid	Event flag ID (Event Flag ID)
setptn	Bit pattern to be set (Set Bit Pattern)

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is less than or equal to zero, or greater than the maximum number of flags)
E_NOEXS	-42	Object does not exist (The event flag specified in flgid does not exist)

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is released from the event flag wait state, the system call dispatches control to the task released from the wait state.

■ Description

This system call sets the event flag specified by flgid to the bit pattern specified by setptn. That is, the system call ORs the value of the event flag specified by flgid with the value of setptn.

The system call returns error E_ID if the event flag ID specified in flgid is less than or equal to zero, or greater than the maximum number of flags (maximum number of event flags specified in the configurator). The system call returns error E_NOEXS if the specified event flag does not exist.

This system call changes the value of the event flag and, if the task wait cancel condition for the task waiting with tk_wai_flg is satisfied, releases that task from the wait state. Accordingly, the waiting task enters the running state or ready state. If the waiting task is in the double-wait state, however, it enters the suspend state.

If the system call is invoked with all the setptn bits set to 0, it terminates normally without manipulating the relevant event flag at all.

A single event flag with an attribute of TA_WMUL allows more than one task to wait at a time. Even for the event flag, therefore, tasks are queued. In that case, a single execution of this system call may dequeue two or more tasks.

3.5.2.4 tk_clr_flg (Clear Event Flag)

Clears an event flag.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_clr_flg ( ID flgid, UINT clrptn ) ;
```

■ Parameter

● Input

flgid	Event flag ID (Event Flag ID)
clrptn	Bit pattern to be cleared (Clear Bit Pattern)

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is less than or equal to zero, or greater than the maximum number of flags)
E_NOEXS	-42	Object does not exist (The event flag specified in flgid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call clears the bits set to 0 in clrptn to the value of the event flag specified by flgid. That is, the system call ANDs the value of the event flag specified by flgid with the value of clrptn.

The system call returns error E_ID if the event flag ID specified in flgid is less than or equal to zero, or greater than the maximum number of flags (maximum number of event flags specified in the configurator). The system call returns error E_NOEXS if the specified event flag does not exist.

This system call does not release the task having the specified event flag from the wait state.

If the system call is invoked with all the clrptn bits set to 1, it does not manipulate the relevant event flag at all. Even in that case, the system call terminates normally.

3.5.2.5 tk_wai_flg (Wait Event Flag)

Waits for an event flag.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_wai_flg ( ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO tmout ) ;
```

■ Parameter

● Input

flgid Event flag ID (Event Flag ID)

waiptn Wait bit pattern (Wait Bit Pattern)

wfmode Wait mode (Wait Event Flag Mode)

wfmode := (TWF_ANDW || TWF_ORW) | [TWF_CLR || TWF_BITCLR]

Mode	Value	Meaning
TWF_ANDW	0x00	AND wait
TWF_ORW	0x01	OR wait
TWF_CLR	0x10	Clear all bits
TWF_BITCLR	0x20	Clear only legitimate bits

tmout Specifies the timeout

The following macros can be specified in addition to the values from 0 to 0x7ffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd Error code (Error Code)

p_flgptn Wait cancel bit pattern (Event Flag Bit Pattern)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is less than or equal to zero, or greater than the maximum number of flags)
E_NOEXS	-42	Object does not exist (The event flag specified in flgid does not exist)
E_PAR	-17	Parameter error (waipn = 0, wfmde set to a value other than TWF_ANDW, TWF_ORW, TWF_CLR, and TWF_BITCLR, tmout ≤ (-2))
E_OBJ	-41	Object status invalid (Two or more tasks waiting for an event flag with TA_WSGL attribute)
E_DLT	-51	Wait object has been deleted. (Specified event flag deleted during wait time)
E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

If the task having issued this system call enters the event flag wait state, the system call dispatches control to the task coming next in the order of priority.

■ Description

This system call waits until the event flag specified by flgid is set according to the wait cancel condition specified by wfmde.

The system call returns error E_ID if the event flag ID specified in flgid is less than or equal to zero, or greater than the maximum number of flags (maximum number of event flags specified in the configurator). The system call returns error E_NOEXS if the specified event flag does not exist.

When the event flag specified by flgid already satisfies the wait cancel condition specified by wfmde, the issuing task continues execution without entering the wait state. wfmde can specify the following four types of mode:

- TWF_ORW: Wait until any of the bits specified by waipn is set in the event flag specified by flgid. (OR wait)
- TWF_ANDW: Wait until all of the bits specified by waipn is set in the event flag specified by flgid. (AND wait)
- TWF_CLR: When the condition is satisfied and the task is released from the wait state, event flag values (all bits) are cleared to "0" (if TWF_CLR is specified). When the condition is satisfied and the task is released from the wait state, event flag values remain unchanged (if TWF_CLR is not specified).
- TWF_BITCLR: When the task is released from the wait state with the condition satisfied, only the bit that matches the event flag wait cancel condition is cleared to 0. (Event flag value &= wait cancel condition)

If `wfmode` specifies a value other than `TWF_ANDW`, `TWF_ORW`, `TWF_CLR`, and `TWF_BITCLR`, the system call returns with an `E_PAR` error.

`p_flgptn` specifies the address of the area that contains the event flag value (`flgptn`) for canceling the wait state. When `TWF_CLR` or `TWF_BITCLR` is specified, the `flgptn` value is the value valid before the event flag is cleared. `flgptn` satisfies the wait cancel condition. Note that, if the wait state is canceled either upon timeout or by `tk_rel_wai` or `tk_del_flg`, `flgptn` becomes indeterminate.

The maximum wait time (timeout) can be specified in `tmout`. If the `tmout` time passes with the wait cancel condition unsatisfied, the system call returns with an `E_TMOUT` error.

`TMO_FEVR` means an infinite timeout. In this case, the system call remains in the wait state until the condition is satisfied or `tk_rel_wai` is issued. If `TMO_POL` is specified, the task does not enter the wait state and it returns error `E_TMOUT`.

If a timeout occurs, the system call does not clear the event flag even with `TWF_CLR` or `TWF_BITCLR` specified.

The time unit for `tmout` is the same as for the system timer (= 1ms). If `tmout` is -2 or less, the function returns with an `E_PAR` error.

When an event flag with `TA_WSGL` attribute contains a waiting task, any other task cannot execute this system call for that event flag. In that case, the task executing the system call later returns with an `E_OBJ` error, whether or not the task enters the wait state (whether or not the wait cancel condition is satisfied).

In contrast, a single event flag with `TA_WMUL` attribute allows more than one task to wait at a time. Even for the event flag, therefore, tasks are queued. In this case, a single execution of `tk_set_flg` may release two or more tasks from their wait state.

If two or more tasks are queued in an event flag with `TA_WMUL` attribute, the following operations are performed:

- The queue is based on FIFO or task priority. (Note, however, that the first task in the queue is not always dequeued first, depending on the relationships with `waipn` and `wfmode`.)
- If the queue contains a clear-specifying task, the flag is cleared when that task is dequeued.
- The task placed behind the clear-specifying task in the queue references the event flag that has already been cleared.

If two or more tasks with the same priority are dequeued by `tk_set_flg` at the same time, they save their priorities in the original event flag queue.

If `waipn` specifies 0, the system call returns with an `E_PAR` error. When a task waiting for an event flag exists and the event flag is deleted by `tk_del_flg`, the wait state is canceled and this system call returns with an `E_DLT` error. If the `tk_rel_wai` system call is issued by the task in the event flag wait state, the wait state is canceled and the system call returns with an `E_RLWAI` error. The system call returns error `E_CTX` if called from a task-independent portion or when dispatch is disabled.

Even when `p_flgptn` is invalid, no error checking is performed, where operation is not guaranteed.

■ Additional Notes

When this system call specifies the AND of all bits (`waipn` = 0xffffffff, `wfmode` = `TWF_ORW`) as its wait cancel condition, it can transfer a message using a bit pattern of the CPU's bit width in combination with `tk_set_flg`. In this case, however, the message with all the bits set to 0 cannot be sent. If the next message is sent by `tk_set_flg` before the previous message is read by this system call, the previous message is erased. This means that message queuing is not available. As specifying "waipn = 0" results in an `E_PAR` error, `waipn` of the task waiting in the event flag is assured to be other than 0. If `tk_set_flg` sets all bits, therefore, the task placed first in the queue is necessarily dequeued irrespective of the condition in which the task waits for the event flag.

3.5.2.6 tk_ref_flg (Refer Event Flag Status)

References the event flag status.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_flg ( ID flgid, T_RFLG *pk_rflg ) ;
```

```
typedef struct t_rflg {
    VP      exinf;
    ID      wtsk;
    UINT    flgptn;
} T_RFLG;
```

■ Parameter

● Input

flgid	Event flag ID (Event Flag ID)
pk_rflg	Start address of the packet to which to return the event flag status (Packet of Refer Event Flag)

● Output

ercd	Error code (Error Code)
------	-------------------------

● Data returned in packet

exinf	Extended Information (Extended Information)
wtsk	Waiting task present or not (Wait Task)
flgptn	Event flag bit pattern (Event Flag Bit Pattern)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (flgid is less than or equal to zero, or greater than the maximum number of flags)
E_NOEXS	-42	Object does not exist (The event flag specified in flgid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call references various types of status of the event flag specified by `flgid` and returns the current flag value (`flgptn`), waiting task presence/absence (`wtsk`), and extended information (`exinf`) as the return values.

The system call returns error `E_ID` if the event flag ID specified in `flgid` is less than or equal to zero, or greater than the maximum number of flags (maximum number of event flags specified in the configurator). If the specified event flag does not exist, the system call returns with an `E_NOEXS` error.

`Wtsk` set to the ID of the task waiting in the specified event flag. If two or more tasks are waiting in the event flag (only with the `TA_WMUL` attribute), the system call returns the ID of the first task in the queue. When there is no task waiting, the `wtsk` value is 0.

Even when `pk_rflg` is invalid, no error checking is performed, where operation is not guaranteed.

3.5.3 Mailbox Function System Calls

This section describes the system calls for the mailbox function.

■ Mailbox Function System Calls

The mailbox function consists of the following five system calls:

- `tk_cre_mbx` (Create Mailbox)
- `tk_del_mbx` (Delete Mailbox)
- `tk_snd_mbx` (Send Message to Mailbox)
- `tk_rcv_mbx` (Receive Message from Mailbox)
- `tk_ref_mbx` (Refer Mailbox Status)

3.5.3.1 tk_cre_mbx (Create Mailbox)

Creates a mailbox.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID mbxid = tk_cre_mbx ( T_CMBX* pk_cmbx ) ;
```

```
typedef struct t_cmbx {
    VP  exinf;
    ATR mbxatr;
} T_CMBX;
```

■ Parameter

● Input

pk_cmbx Mailbox generation information (Packet of Create Mailbox)

● Data to set in packet

exinf Extended Information (Extended Information)

mbxatr Mailbox attribute (Mailbox Attribute)

mbxatr:= (TA_TFIFO || TA_TPRI) | (TA_MFIFO || TA_MPRI)

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority
TA_MFIFO	0x00000000	Manages messages based on FIFO.
TA_MPRI	0x00000002	Manages messages in order of priority.

● Output

mbxid Mailbox ID (Mailbox ID)
Or, error code (Error Code)

■ Error Code

E_LIMIT - 34 The number of mailboxes is greater than the system's upper limit.
E_RSATR - 11 "Reserved" attribute (mbxatr has been set to an undefined value.)

■ Dispatch Trigger

This system call does not perform dispatching.

■ Description

This system call creates a mailbox with its mailbox ID number assigned.

The user can freely use `exinf` to hold information on the target mailbox. The information specified here can be taken out via `tk_ref_mbx`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS ignores the data in `exinf`.

`mbxatr` specifies the attribute of the mailbox. The system call returns error `E_RSATR` if an undefined attribute is specified.

`TA_TFIFO` or `TA_TPRI` allows you to specify how the tasks to receive messages are placed in the mailbox queue. If the attribute is `TA_TFIFO`, the task wait queue operates as a FIFO. If the attribute is `TA_TPRI`, the task wait queue is ordered by task priority.

`TA_MFIFO` or `TA_MPRI` allows you to specify how messages are placed in the message queue (the queue of messages waiting to be received). If the attribute is `TA_MFIFO`, the message queue is based on FIFO. If the attribute is `TA_MPRI`, the message queue is handled in order of message priority. The priority of each message is specified by `msgpri` in the message packet (see Section 3.5.3.3 `tk_snd_mbx` (Send Message to Mailbox)). Message priorities are positive values. 1 represents the highest priority; greater values represent lower priorities. The maximum positive value expressed as the `PRI` type represents the lowest priority. Messages with the same priority are handled based on FIFO.

This system call returns with an `E_LIMIT` error if invoked while as many mailboxes as the system's upper limit (the maximum number of mailboxes specified in the configurator) have been created. Even when `pk_cmbx` is invalid, no error checking is performed, where operation is not guaranteed.

3.5.3.2 tk_del_mbx (Delete Mailbox)

Deletes a mailbox.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_mbx ( ID mbxid ) ;
```

■ Parameter

● Input

mbxid Mailbox ID (Mailbox ID)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbxid is less than or equal to zero, or greater than the maximum number of mailboxes)
E_NOEXS	-42	Object does not exist (The mailbox specified in mbxid does not exist)

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is waiting in a mailbox deleted, the system call dispatches control to the task released from the wait state.

■ Description

This system call deletes the mailbox specified by mbxid. Precisely, the system call puts that mailbox into the ungenerated state and unassigns its ID number.

The system call returns error E_ID if the mailbox ID specified in mbxid is less than or equal to zero, or greater than the maximum number of mailboxes (maximum number of mailboxes specified in the configurator). The system call returns error E_NOEXS if the mailbox does not exist.

When the specified mailbox has a task waiting for a message, this system call also terminates normally. For the waiting task, however, tk_rcv_mbx returns with an E_DLT error. Even when a message is still left in the mailbox, the system call deletes the mailbox without causing an error.

3.5.3.3 tk_snd_mbx (Send Message to Mailbox)

Sends a message to a mailbox.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_snd_mbx ( ID mbxid, T_MSG *pk_msg ) ;
```

```
typedef struct t_msg {
    VP msgque[1];
} T_MSG;
```

```
typedef struct t_msg_pri {
    T_MSG msgque;
    PRI msgpri;
} T_MSG_PRI;
```

■ Parameter

● Input

mbxid	Mailbox ID (Mailbox ID)
pk_msg	Message packet start address (Packet of Message)

● Data to set in packet

msgque	OS reserved area for message queue
msgpri	Message priority (Message priority)

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbxid is less than or equal to zero, or greater than the maximum number of mailboxes)
E_NOEXS	-42	Object does not exist (The mailbox specified in mbxid does not exist)
E_PAR	-17	Parameter error (msgpri is less than or equal to 0 while the specified mailbox is in order of message priority.)

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is released from the mailbox wait state, the system call dispatches control to the task released from the wait state.

■ Description

This system call sends the message packet starting at the address specified by `pk_msg` to the mailbox specified by `mbxid`.

The system call returns error `E_ID` if the mailbox ID specified in `mbxid` is less than or equal to zero, or greater than the maximum number of mailboxes (maximum number of mailboxes specified in the configurator). The system call returns error `E_NOEXS` if the mailbox does not exist.

The system call passes only the start address (`pk_msg` value) upon reception without copying the contents of the message packet. If the specified mailbox already contains a message waiting task, the first task in the queue is dequeued, and `pk_msg` specified by this system call is sent to that task as the return value of `tk_rcv_mbx`.

If the specified mailbox contains no task waiting for a message, in contrast, the sent message is placed in the message queue in the mailbox. In either case, the task having issued this system call does not enter the wait state. `pk_msg` specifies the start address of the message packet including the message header.

As the message header, use `T_MSG` for the mailbox with `TA_MFIFO` attribute or `T_MSG_PRI` for the mailbox with `TA_MPRI` attribute. If you use a `T_MSG` message header for a mailbox with `TA_MPRI` attribute, the resulting operation is not guaranteed due to the absence of the `msgpri` field.

The message header size is fixed-length and assigned by `sizeof(T_MSG)` or `sizeof(T_MSG_PRI)`. An actual message can be stored in the area that follows the message header. The size of the message itself is not limited.

Even when `pk_msg` is invalid, no error checking is performed, where operation is not guaranteed.

■ Additional Notes

Message transmission by this system call is performed irrespective of the state of the receiving task. That is, the system call performs asynchronous message transmission. Messages called by tasks are queued but the tasks themselves are not queued. In other words, there is a message queue or receiving task queue but there is no sending task queue.

3.5.3.4 tk_rcv_mbx (Receive Message from Mailbox)

Receives a message from a mailbox.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_rcv_mbx ( ID mbxid, T_MSG **ppk_msg, TMO tmout ) ;
```

```
typedef struct t_msg {
    VP  msgque[1];
} T_MSG;
```

```
typedef struct t_msg_pri {
    T_MSG  msgque;
    PRI    msgpri;
} T_MSG_PRI;
```

■ Parameter

● Input

mbxid	Mailbox ID (Mailbox ID)
ppk_msg	Address of the variable to return the message packet start address (Packet of Message)
tmout	Specifies the timeout (Timeout) The following macros can be specified in addition to the values from 0 to 0x7fffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd	Error code (Error Code)
------	-------------------------

● Data returned in packet

msgque	OS reserved area for message queue
msgpri	Message priority (Message priority)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbxid is less than or equal to zero, or greater than the maximum number of mailboxes)
E_NOEXS	-42	Object does not exist (The mailbox specified in mbxid does not exist)
E_PAR	-17	Parameter error (tmout \leq (-2))
E_DLT	-51	Wait object has been deleted. (Specified mailbox deleted during wait time)
E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

If the task having issued this system call enters the mailbox wait state, the system call dispatches control to the task coming next in the order of priority.

■ Description

This system call receives a message from the mailbox specified by mbxid.

The system call returns error E_ID if the mailbox ID specified in mbxid is less than or equal to zero, or greater than the maximum number of mailboxes (maximum number of mailboxes specified in the configurator). The system call returns error E_NOEXS if the mailbox does not exist.

The maximum wait time (timeout) can be specified in tmout. If the specified tmout time has passed with the wait cancel condition unsatisfied (no message having arrived), the system call returns with an E_TMOUT error. The time unit for tmout is the same as for the system timer (= 1ms). TMO_FEVR means an infinite timeout. In that case, the system call remains in the wait state until a message arrives or tk_rel_wai is issued. If TMO_POL is specified, the task does not enter the wait state and it returns error E_TMOUT.

If tmout is less than or equal to -2, the system call returns with an E_PAR error.

If no message has been sent to the specified mailbox (the message queue is empty), the task having issued this system call enters the wait state and is placed in the queue of tasks awaiting message arrival. When the specified mailbox already contains a message, in contrast, the system call takes the first message out of the message queue as a return value of ppk_msg.

If the specified mailbox is deleted by tk_del_mbx when the issuing task is in the message wait state, the task is released from the wait state and the system call returns with an E_DLT error. If the message waiting task issues the tk_rel_wai system call, the task is released from the wait state and the system call returns with an E_RLWAI error. The system call returns error E_CTX if called from a task-independent portion or when dispatch is disabled.

Even when ppk_msg is invalid, no error checking is performed, where operation is not guaranteed.

■ Additional Notes

ppk_msg specifies the start address of the message packet including the message header. The message header is T_MSG for the mailbox with TA_MFIFO attribute or T_MSG_PRI for the mailbox with TA_MPRI attribute.

3.5.3.5 tk_ref_mbx (Refer Mailbox Status)

References the mailbox status.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_mbx ( ID mbxid, T_RMBX *pk_rmbx ) ;

typedef struct t_rmbx {
    VP      exinf;
    ID      wtsk;
    T_MSG   *pk_msg;
} T_RMBX;
```

■ Parameter

● Input

mbxid	Mailbox ID (Mailbox ID)
pk_rmbx	Packet address to which to return the mailbox status (Packet of Refer Mailbox)

● Output

ercd	Error code (Error Code)
● Data returned in packet	
extinf	Extended information (Extended Information)
wtsk	Waiting task present or not (Wait Task)
pk_msg	Start address of the next message packet to be received (Packet of Message)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbxid is less than or equal to zero, or greater than the maximum number of mailboxes)
E_NOEXS	-42	Object does not exist (The mailbox specified in mbxid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call references various types of status of the mailbox specified by `mbxid` and returns the next message to be received (the first message in the message queue), waiting task presence/absence (`wtsk`), and extended information (`exinf`) as the return values.

The system call returns error `E_ID` if the mailbox ID specified in `mbxid` is less than or equal to zero, or greater than the maximum number of mailboxes (maximum number of mailboxes specified in the configurator). If the specified mailbox does not exist, the system call returns with an `E_NOEXS` error.

`Wtsk` set to the ID of the task waiting in the specified mailbox. If two or more tasks are waiting in the mailbox, the system call returns the ID of the first task in the queue. When there is no task waiting, `wtsk` becomes 0.

`pk_msg` specifies the message to be received the next time `tk_rcv_mbx` is executed. When the message queue contains no message, `pk_msg` becomes NULL. In any case, at least either "`pk_msg = NULL`" or "`wtsk = 0`" is true.

Even when `pk_rmbx` is invalid, no error checking is performed, where operation is not guaranteed.

3.6 System Calls for Extended Synchronization/Communication Function

This section describes the system calls for the extended synchronization/communication function.

■ System Calls for Extended Synchronization/Communication Function

The extended synchronization/communication function consists of the following three function system calls:

- Mutex function system calls
- Message buffer function system calls
- Rendezvous function system calls

3.6.1 Mutex Function System Calls

This section describes the mutex function system call.

■ Mutex Function System Calls

The mutex function consists of the following five system calls:

- `tk_cre_mtx` (Create Mutex)
- `tk_del_mtx` (Delete Mutex)
- `tk_loc_mtx` (Lock Mutex)
- `tk_unl_mtx` (Unlock Mutex)
- `tk_ref_mtx` (Refer Mutex Status)

3.6.1.1 tk_cre_mtx (Create Mutex)

Creates a mutex.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ID mtxid = tk_cre_mtx ( T_CMTX *pk_cmtx ) ;
typedef struct t_cmtx {
    VP      exinf;
    ATR      mtxatr;
    PRI      ceilpri;
} T_CMTX;
```

■ **Parameter**

● **Input**

- pk_cmtx

Information about mutex creation.
Packet address (Packet of Create Mutex)
- Data to set in packet

exinf

Extended information (Extended Information)

mtxatr

Mutex attribute (Mutex Attribute)
mtxatr:= (TA_TFIFO || TA_TPRI || TA_INHERIT || TA_CEILING)

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority
TA_INHERIT	0x00000002	Priority inheritance protocol
TA_CEILING	0x00000003	Priority ceiling protocol

- ceilpri

Mutex ceiling priority (Ceiling Priority)

● **Output**

- mtxid

Mutex ID (Mutex ID)
Or, error code (Error Code)

■ Error Code

<code>E_LIMIT</code>	-34	The number of mutexes is greater than the system's upper limit.
<code>E_RSATR</code>	-11	"Reserved" attribute (mtxatr has been set to an undefined value.)
<code>E_PAR</code>	-17	Parameter error (ceilpri is less than or equal to 0, or is greater than the system's ceiling priority (with mtxatr specifying <code>TA_CEILING</code>).)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates a mutex with its mutex ID number assigned.

The user can freely use `exinf` to hold information on the target mutex. The information specified here can be taken out via `tk_ref_mtx`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS ignores the data in `exinf`.

Mtxatr specifies the attribute of the mutex. The system call returns error `E_RSATR` if an undefined attribute is specified.

If `mtxatr` specifies `TA_TFIFO`, the mutex task queue is based on FIFO. If the attribute is `TA_TPRI`, `TA_INHERIT`, or `TA_CEILING`, the mutex task queue is handled in order of task priority. When the attribute is `TA_INHERIT` or `TA_CEILING`, the priority inheritance protocol or priority ceiling protocol is used, respectively.

Ceilpri is valid only when `mtxatr` specifies `TA_CEILING`, specifying the ceiling priority of the mutex. When `ceilpri` is less than or equal to 0, or is greater than the system's maximum priority (maximum priority set in the configurator), the system call returns with an `E_PAR` error.

This system call returns with an `E_LIMIT` error if invoked while as many mutexes as the system's upper limit (the maximum number of mutexes specified in the configurator) have been created. Even when `pk_cmtx` is invalid, no error checking is performed, where operation is not guaranteed.

3.6.1.2 tk_del_mtx (Delete Mutex)

Deletes a mutex.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_mtx ( ID mtxid ) ;
```

■ Parameter

● Input

mtxid Mutex ID (Mutex ID)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mtxid is less than or equal to zero, or greater than the maximum number of mutexes)
E_NOEXS	-42	Object does not exist (The mutex specified in mtxid does not exist)

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is waiting in a mutex deleted, the system call dispatches control to the task released from the wait state.

■ Description

This system call deletes the mutex specified by mtxid. Precisely, the system call puts that mutex into the ungenerated state and unassigns its ID number.

The system call returns error E_ID if the mutex ID specified by mtxid is less than or equal to zero or greater than the maximum number of mutexes (maximum number of mutexes specified in the configurator). The system call returns error E_NOEXS if the mutex does not exist.

Even when the specified mutex has a task waiting for locking, this system call terminates normally. For the waiting task, however, the system call returns with an E_DLT error. When a mutex being locked by a task is deleted, the number of mutexes currently being locked by the task decreases accordingly. If the mutex to be deleted has the TA_INHERIT or TA_CEILING attribute, therefore, the current priority of the locking task must be changed.

3.6.1.3 tk_loc_mtx (Lock Mutex)

Locks a mutex.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_loc_mtx ( ID mtxid, TMO tmout ) ;
```

■ Parameter

● Input

mtxid Mutex ID (Mutex ID)

tmout Specifies the timeout (Timeout)

The following macros can be specified in addition to the values from 0 to 0x7ffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mtxid is less than or equal to zero, or greater than the maximum number of mutexes)
E_NOEXS	-42	Object does not exist (The mutex specified in mtxid does not exist)
E_PAR	-17	Parameter error (tmout ≤ (-2))
E_DLT	-51	Wait object has been deleted. (Specified mutex deleted during wait time)
E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)
E_ILUSE	-28	Illegal use (multi-lock, ceiling priority violation)

■ Dispatch Trigger

When the task having issued this system call enters the lock wait state, the system call dispatches control to the task coming next in the order of priority.

■ Description

This system call locks the mutex specified by `mtxid`.

The system call returns error `E_ID` if the mutex ID specified by `mtxid` is less than or equal to zero or greater than the maximum number of mutexes (maximum number of mutexes specified in the configurator). The system call returns error `E_NOEXS` if the mutex does not exist.

The maximum wait time (timeout) can be specified in `tmout`. If the `tmout` time has passed with the wait cancel condition unsatisfied (without unlocking the mutex), the system call returns with an `E_TMOUT` error.

`TMO_FEVR` means an infinite timeout. In that case, the system call remains in the wait state until it reserves the lock or `tk_rel_wai` is issued. If `TMO_POL` is specified, the task does not enter the wait state and it returns error `E_TMOUT`.

The time unit for `tmout` is the same as for the system timer (= 1ms). If `tmout` is -2 or less, the function returns with an `E_PAR` error.

When the specified mutex can be locked, the task having issued the system call terminates normally without entering the wait state. In this case, the mutex enters the locked state. If it cannot be locked, the issuing task enters the wait state. That is, the issuing task is placed in the queue for that mutex.

If the issuing task has already locked the specified mutex, the system call returns with an `E_ILUSE` (multi-lock) error. When the attribute of the specified mutex is `TA_CEILING`, the system call returns with an `E_ILSUE` (ceiling priority violation) if the base priority of the issuing task is higher than the ceiling priority of the specified mutex.

If the specified mutex is deleted by `tk_del_mtx` when the issuing task is in the mutex wait state, the task is released from the wait state and the system call returns with an `E_DLT` error. If the mutex waiting task issues `tk_rel_wai`, the task is released from the wait state and the system call returns with an `E_RLWAI` error. The system call returns error `E_CTX` if called from a task-independent portion or when dispatch is disabled.

■ Additional Notes

- For the mutex with `TA_INHERIT` attribute

When the issuing task waits to lock a mutex and if the current priority of the task having locked the mutex is lower than that of the issuing task, raise the current priority of the locking task to the same as the current priority of the issuing task. If the task in the lock wait state leaves it without reserving the lock (for example, due to timeout), lower the current priority of the task having locked the mutex to the highest current priority below:

- (a) Highest of the current priorities of the tasks waiting for locking the mutex
- (b) Base priority of the locking task

- For the mutex with `TA_CEILING` attribute

When the issuing task reserves the lock and if the current priority of the issuing task is lower than the ceiling priority of the mutex, lower the current priority of the issuing task to the ceiling priority of the mutex.

3.6.1.4 tk_unl_mtx (Unlock Mutex)

Unlocks a mutex.

Task portion	○	Task-independent portion	×	Dispatch disabled	○
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_unl_mtx ( ID mtxid ) ;
```

■ Parameter

● Input

mtxid Mutex ID (Mutex ID)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mtxid is less than or equal to zero, or greater than the maximum number of mutexes)
E_NOEXS	-42	Object does not exist (The mutex specified in mtxid does not exist)
E_ILUSE	-28	Illegal use (Mutex not locked by the issuing task)
E_CTX	-25	Context error(Executed from a task-independent portion)

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is released from the lock wait state, the system call dispatches control to the task released from the wait state.

If the priority of the issuing task is lowered as the specified mutex is unlocked, the system call may dispatch control to a higher-priority task.

■ Description

This system call unlocks the mutex specified by `mtxid`. If any task has been waiting for locking the mutex, the system call releases the first task in the queue from the wait state and puts that task in the lock reserved state.

The system call returns error `E_ID` if the mutex ID specified by `mtxid` is less than or equal to zero or greater than the maximum number of mutexes (maximum number of mutexes specified in the configurator). The system call returns error `E_NOEXS` if the mutex does not exist. If a mutex not locked by the issuing task is specified, the system call returns with an `E_ILUSE` error.

■ Additional Notes

If the attribute of the unlocked mutex is `TA_INHERIT` or `TA_CEILING`, the current priority of the issuing task must be lowered as follows. When there are no mutexes currently being locked by the issuing task as the last one is unlocked, lower the current priority of the issuing task to its base priority. If there is still a mutex currently being locked by the issuing task, lower the current priority of the issuing task to the highest of the following priorities:

- (a) Highest of the priorities of all the mutexes being locked
- (b) Base priority

If the task terminates (goes dormant or unregistered) while locking mutexes, all the mutexes being locked are unlocked automatically.

3.6.1.5 tk_ref_mtx (Refer Mutex Status)

References the mutex status.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_mtx ( ID mtxid, T_RMTX *pk_rmtx ) ;
```

```
typedef struct t_rmtx {
    VP exinf;
    ID htsk;
    ID wtsk;
} T_RMTX;
```

■ Parameter

● Input

mtxid	Mutex ID (Mutex ID)
pk_rmtx	Packet address to which to return the mutex status (Packet of Refer Mutex Status)

● Output

ercd	Error code (Error Code)
------	-------------------------

● Data returned in packet

exinf	Extended Information (Extended Information)
htsk	ID of the currently locking task (Hold Task ID)
wtsk	ID of the lock waiting task (Wait Task ID)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mtxid is less than or equal to zero, or greater than the maximum number of mutexes)
E_NOEXS	-42	Object does not exist (The mutex specified in mtxid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call references various types of status of the mutex specified by `mtxid` and returns the locking task ID (`htsk`), lock waiting task ID (`wtsk`), and extended information (`exinf`) as the return values.

The system call returns error `E_ID` if the mutex ID specified by `mtxid` is less than or equal to zero or greater than the maximum number of mutexes (maximum number of mutexes specified in the configurator). The system call returns error `E_NOEXS` if the mutex does not exist.

`Htsk` set to the ID of the task currently locking the specified mutex. When there is no locking task, `htsk` becomes 0.

`Wtsk` set to the ID of the task waiting for the specified mutex. If two or more tasks are waiting, the system call returns the ID of the first task in the queue. If there is no waiting task, `wtsk` = 0.

Even when `pk_rmtx` is invalid, no error checking is performed, where operation is not guaranteed.

3.6.2 Message Buffer Function System Calls

This section describes the system calls for the message buffer function.

■ Message Buffer Function System Calls

The message buffer function consists of the following five system calls:

- `tk_cre_mbf` (Create MessageBuffer)
- `tk_del_mbf` (Delete MessageBuffer)
- `tk_snd_mbf` (Send Message to MessageBuffer)
- `tk_rcv_mbf` (Receive Message from MessageBuffer)
- `tk_ref_mbf` (Refer MessageBuffer Status)

3.6.2.1 tk_cre_mbf (Create MessageBuffer)

Creates a message buffer.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID mbfid = tk_cre_mbf ( T_CMBF *pk_cmbf ) ;
```

```
typedef struct t_cmbf {
    VP  exinf;
    ATR mbfatr;
    INT bufsz;
    INT maxmsz;
    VP  bufptr;
} T_CMBF;
```

■ Parameter

- Input

pk_cmbf	Start address of the packet of message buffer generation information (Packet of Create MessageBuffer)
---------	--

- Data to set in packet

exinf	Extended Information (Extended Information)
mbfattr	Message buffer attribute (MessageBuffer Attribute)
mbfattr	$mbfattr := (TA_TFIFO \parallel TA_TPRI) \mid [TA_USERBUF]$

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority
TA_USERBUF	0x00000020	Uses the area specified by the user as the message buffer area.

bufsz	Message buffer size (in bytes) (Buffer Size)
maxmsz	Maximum message length (in bytes) (Maximum Message Size)
bufptr	Address of user buffer (Buffer Pointer)

- Output

mbfid	Message buffer ID (MessageBuffer ID)
	Or, error code (Error Code)

■ Error Code

E_NOMEM	- 33	Insufficient memory (Unable to allocate the message buffer area)
E_LIMIT	- 34	The number of message buffers is greater than the system's upper limit.
E_RSATR	- 11	"Reserved" attribute (mtxatr has been set to an undefined value.)
E_PAR	- 17	Parameter error (bufsz negative; maxmsz smaller than or equal to 0, bufsz is not in multiples of 4 (if TA_USERBUF is specified))

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates a message buffer with its message buffer ID number assigned.

The user can freely use exinf to hold information on the target message buffer. The information specified here can be taken out via tk_ref_mbf. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in exinf. The OS ignores the data in exinf.

Mbfatr specifies the attribute of the message buffer. The system call returns error E_RSATR if an undefined attribute is specified.

TA_TFIFO or TA_TPRI allows you to specify how the tasks to send messages when the buffer is full are placed in the message buffer queue. If the attribute is TA_TFIFO, the task queue is based on FIFO. If the attribute is TA_TPRI, the task queue is handled in order of task priority. Note, however, that the order for the message queue is only FIFO. The order for the queue of message reception wait tasks is only FIF as well.

If TA_USERBUF is specified, bufptr is enabled, where bufsz bytes of memory beginning at bufptr is used as the message buffer area. In this case, the message buffer area is not provided by the OS. When TA_USERBUF is not specified, bufptr is ignored and the message buffer area is allocated by the OS. You can set bufsz to 0, thereby enabling completely synchronous communication without using any message buffer. The first 4 bytes in the message buffer area is used by the kernel. In addition, if bufsz is not in multiples of 8 bytes but TA_USERBUF is specified in mbfatr, bufsz is rounded up to multiples of 8 bytes. If bufsz is negative, the system call returns with an E_PAR error. If the OS fails to allocate the message buffer area, the system call returns with an E_NOMEM error.

Maxmsz specifies the maximum length of messages in bytes, which can be communicated via the message buffer. If maxmsz is less than or equal to 0, the system call returns with an E_PAR error.

This system call returns with an E_LIMIT error if invoked while as many message buffers as the system's upper limit (the maximum number of message buffers specified in the configurator) have been created. Even when pk_cmbf or bufptr is invalid, no error checking is performed, where operation is not guaranteed.

Message buffers are objects which manage transmission and reception of variable-length messages. Their difference from mailboxes (mbx) is that the message buffer copies the contents of the variable-length message when sending and receiving it. There is also a feature that places the message sender in the wait state when the buffer is full.

3.6.2.2 tk_del_mbf (Delete MessageBuffer)

Deletes a message buffer.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_del_mbf ( ID mbfid ) ;
```

■ **Parameter**

● **Input**

mbfid	Message buffer ID (MessageBuffer ID)
-------	---

● **Output**

ercd	Error code (Error Code)
------	-------------------------

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbfid is less than or equal to zero, or greater than the maximum number of message buffers)
E_NOEXS	-42	Object does not exist (The message buffer specified by mbfid does not exist)

■ **Dispatch Trigger**

If a task higher in priority than the task having issued this system call is waiting in a message buffer deleted, the system call dispatches control to the task released from the wait state.

■ **Description**

This system call deletes the message buffer specified by mbfid. Precisely, the system call puts that message buffer into the ungenerated state and unallocates its ID number and the buffer area for messages.

The system call returns error E_ID if the message ID specified by mbfid is less than or equal to zero or greater than the maximum number of message buffers (maximum number of message buffers specified in the configurator). The system call returns error E_NOEXS if the message buffer does not exist.

Even when the specified message buffer has a task waiting for message reception or transmission, this system call terminates normally. For the waiting task, however, the system call returns with an E_DLT error. Even when a message is still left in the specified message buffer, the system call deletes the message buffer without causing an error, when the message contained therein is accordingly lost.

3.6.2.3 tk_snd_mbf (Send Message to MessageBuffer)

Sends a message to a message buffer.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_snd_mbf ( ID mbfid, VP msg, INT msgsz, TMO tmout ) ;
```

■ Parameter

● Input

mbfid	Message buffer ID (MessageBuffer ID)
msgsz	Outgoing message size (in bytes) (Message Size)
msg	Outgoing message start address (Message)
tmout	Specifies the timeout (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7fffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbfid is less than or equal to zero, or greater than the maximum number of message buffers)
E_NOEXS	-42	Object does not exist (The message buffer specified by mbfid does not exist)
E_PAR	-17	Parameter error (msgsz ≤ 0, msgsz > maxmsz, tmout ≤ (-2))
E_DLT	-51	Wait object has been deleted. (Specified message buffer deleted during wait time)

E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state (tmout is not TMO_POL))

■ Dispatch Trigger

If a task higher in priority than the task having issued this system call is released from the message buffer reception wait state, the system call dispatches control to the task released from the wait state. If the task having issued this system call enters the message buffer transmission wait state, the system call dispatches control to the task coming next in the order of priority.

■ Description

This system call sends the message starting at the address specified by msg to the message buffer specified by mbfid.

The system call returns error E_ID if the message ID specified by mbfid is less than or equal to zero or greater than the maximum number of message buffers (maximum number of message buffers specified in the configurator). The system call returns error E_NOEXS if the message buffer does not exist.

The message size is specified by msgsz. That is, msgsz bytes starting at msg are copied to the message queue in the message buffer specified by mbfid. The message queue is implemented by a ring buffer.

If msgsz is less than or equal to 0, or is greater than maxmsz specified in k_cre_mbf, the system call returns with an E_PAR error.

The maximum wait time (timeout) can be specified in tmout. If the specified tmout time has passed with the wait cancel condition unsatisfied (sufficient free space cannot be allocated for the buffer), the system call returns with an E_TMOUT error.

TMO_FEVR means an infinite timeout. In this case, the system call remains in the wait state until free space can be allocated for the buffer or tk_rel_wai is issued. If TMO_POL is specified, the task does not enter the wait state, even when the buffer does not have enough free space, and it returns error E_TMOUT.

The time unit for tmout is the same as for the system timer (= 1ms). If tmout is -2 or less, the function returns with an E_PAR error.

If the buffer is short of free space so that the message starting at msg cannot be placed in the message queue, the task having issued this system call enters the message transmission wait state and is placed in the transmission queue to wait for free space in the buffer. The queue is handled based on FIFO or in order of task priority, depending on the setting made when tk_cre_mbf is issued.

If executed from a task-independent portion or in the dispatch disabled state, the system call returns with an E_CTX error. If the specified message buffer is deleted by tk_del_mbf when the issuing task is in the message transmission wait state, the task is released from the wait state and the system call returns with an E_DLT error. If the message transmission waiting task issues tk_rel_wai, the task is released from the wait state and the system call returns with an E_RLWAI error.

No error check is performed even if msg is invalid. The operation in this case is not guaranteed.

3.6.2.4 tk_rcv_mbf (Receive Message from MessageBuffer)

Receives a message from a message buffer.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
INT msgsz = tk_rcv_mbf ( ID mbfid, VP msg, TMO tmout ) ;
```

■ Parameter

● Input

mbfid	Message buffer ID (MessageBuffer ID)
msg	Address to hold an incoming message (Message)
tmout	Specifies the timeout (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7fffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

msgsz	Received message size (in bytes) (Message size) Or, error code (Error Code)
-------	---

■ Error Code

E_ID	-18	Invalid ID number (mbfid is less than or equal to zero, or greater than the maximum number of message buffers)
E_NOEXS	-42	Object does not exist (The message buffer specified by mbfid does not exist)
E_PAR	-17	Parameter error (tmout ≤ (-2))
E_DLT	-51	Wait object has been deleted. (Specified message buffer deleted during wait time)

E_RLWAI	-49	Forcibly restored from the wait state (tk_rel_wai called for task while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

If the task having issued this system call enters the message buffer reception wait state, the system call dispatches control to the task coming next in the order of priority. If a task higher in priority than the task having issued this system call is released from the message buffer transmission wait state, the system call dispatches control to the task released from the wait state.

■ Description

This system call receives a message from the message buffer specified by mbfid and stores the message in the area starting at msg. That is, the system call copies the content of the first message in the message queue in the message buffer specified by mbfid to msgsz bytes starting at msg.

The system call returns error E_ID if the message ID specified by mbfid is less than or equal to zero or greater than the maximum number of message buffers (maximum number of message buffers specified in the configurator). The system call returns error E_NOEXS if the message buffer does not exist.

The maximum wait time (timeout) can be specified in tmout. If the specified tmout time has passed with the wait cancel condition unsatisfied (no message having arrived), the system call returns with an E_TMOUT error.

TMO_FEVR means an infinite timeout. In that case, the system call remains in the wait state until a message arrives or tk_rel_wai is issued. If TMO_POL is specified, the task does not enter the wait state, even when no message exists, and it returns error E_TMOUT.

The time unit for tmout is the same as for the system timer (= 1ms). If tmout is -2 or less, the function returns with an E_PAR error.

If no message has been sent to the message buffer specified by mbfid (the message queue is empty), the task having issued this system call enters the wait state and is placed in the queue of tasks awaiting message arrival (reception queue). Note that the order for the reception queue is only FIFO.

If the specified message buffer is deleted by tk_del_mbf when the issuing task is in the message reception wait state, the task is released from the wait state and the system call returns with an E_DLT error. If the message reception waiting task issues tk_rel_wai, the task is released from the wait state and the system call returns with an E_RLWAI error. The system call returns error E_CTX if called from a task-independent portion or when dispatch is disabled.

No error check is performed even if msg is invalid. The operation in this case is not guaranteed.

3.6.2.5 tk_ref_mbf (Refer MessageBuffer Status)

References the message buffer status.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_mbf ( ID mbfid, T_RMBF *pk_rmbf ) ;
```

```
typedef struct t_rmbf {
    VP  exinf;
    ID  wtsk;
    ID  stsk;
    INT msgsz;
    INT frbufsz;
    INT maxmsz;
} T_RMBF;
```

■ Parameter

● Input

mbfid	Message buffer ID (MessageBuffer ID)
pk_rmbf	Packet address to which to return the message buffer status (Packet of Refer MessageBuffer)

● Output

ercd	Error code (Error Code)
------	-------------------------

● Data returned in packet

exinf	Extended Information (Extended Information)
wtsk	Presence or absence of reception wait task (Wait Task)
stsk	Presence or absence of transmission wait task (Send Task)
msgsz	Message size (Message Size)
frbufsz	Free buffer size (in bytes) (Free Buffer Size)
maxmsz	Maximum message length (in bytes) (Maximum Message Size)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mbfid is less than or equal to zero, or greater than the maximum number of message buffers)
E_NOEXS	-42	Object does not exist (The message buffer specified by mbfid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call references various types of status of the message buffer specified by mbfid and returns extended information (exinf), presence or absence of reception wait task (wtsk), presence or absence of transmission wait task (stsk), incoming message size (msgsz), free buffer size (frbufsz), and maximum message length (maxmsz) as the return values.

The system call returns error E_ID if the message ID specified by mbfid is less than or equal to zero or greater than the maximum number of message buffers (maximum number of message buffers specified in the configurator). If the specified message buffer does not exist, the system call returns with an E_NOEXS error.

Wtsk set to the ID of the task awaiting reception with the specified message buffer; Stsk set to the ID of the task awaiting transmission. If two or more tasks are waiting for the message buffer, the system call returns the ID of the first task in the queue. Either value is 0 when there is no task waiting.

Msgsz returns the size of the first message in the message queue (the next message to be received). When the message queue contains no message, msgsz becomes 0. Note that any message whose size is 0 cannot be sent. In any case, at least either "msgsz = 0" or "wtsk = 0" is true.

Frbufsz represents the size of the ring buffer free area making up the message queue. This value tells the maximum size of the message that can be sent. maxmsz returns the maximum length of the message specified by tk_cre_mbf.

Even when pk_rmbf is invalid, no error checking is performed, where operation is not guaranteed.

3.6.3 Rendezvous Function System Calls

This section describes the system calls for the rendezvous port function.

■ Rendezvous Function System Calls

The rendezvous port function consists of the following seven system calls:

- `tk_cre_por` (Create Port for Rendezvous)
- `tk_del_por` (Delete Port for Rendezvous)
- `tk_cal_por` (Call Port for Rendezvous)
- `tk_acp_por` (Accept Port for Rendezvous)
- `tk_fwd_por` (Forward Rendezvous to Another Port)
- `tk_rpl_rdv` (Reply Rendezvous)
- `tk_ref_por` (Refer Port Status)

3.6.3.1 tk_cre_por (Create Port for Rendezvous)

Creates a rendezvous port.

Task portion	<input type="radio"/>	Task-independent portion	<input checked="" type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	----------------------------------	-------------------	-----------------------

■ **C Language Interface**

```
ID porid = tk_cre_por ( T_CPOR *pk_cpor ) ;
```

```
typedef struct t_cpor {  
    VP exinf;  
    ATR poratr;  
    INT maxcmsz;  
    INT maxrmsz;  
} T_CPOR;
```

■ **Parameter**

● **Input**

`pk_cpor` Start address of the packet of rendezvous port generation information (Packet of Create Port)

● **Data to set in packet**

`exinf` Extended Information (Extended Information)

`poratr` Rendezvous port attribute (Port Attribute)

`poratr := (TA_TFIFO || TA_TPRI)`

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority

`maxcmsz` Maximum length (in bytes) of the call message
(Maximum Call Message Size)

`maxrmsz` Maximum length (in bytes) of the reply message
(Maximum Reply Message Size)

● **Output**

`porid` Rendezvous port ID (Port ID)
Or, error code (Error Code)

■ Error Code

<code>E_LIMIT</code>	-34	The number of rendezvous ports is greater than the system's upper limit.
<code>E_RSATR</code>	-11	"Reserved" attribute (poratr has been set to an undefined value.)
<code>E_PAR</code>	-17	Parameter error (maxcmsz negative; maxmsz negative)
<code>E_CTX</code>	-25	Context error (in execution from task independent portion)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates a rendezvous port with its rendezvous port ID number assigned. The rendezvous port is an base object for implementing rendezvous.

The user can freely use `exinf` to hold information on the target rendezvous port. The information specified here can be taken out via `tk_ref_por`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS ignores the data in `exinf`.

`Poratr` specifies the attribute of the rendezvous port. The system call returns error `E_RSATR` if an undefined attribute is specified.

`TA_TFIFO` or `TA_TPRI` allows you to specify how the tasks awaiting rendezvous calling are placed in the queue. The order for the queue of tasks awaiting rendezvous reception is only FIFO.

`Maxcmsz` specifies the maximum length of the message for calling, in bytes. If the `maxcmsz` value is negative, the system call returns with an `E_PAR` error.

`Maxrmsz` specifies the maximum length of the message for replying, in bytes. If the `maxrmsz` value is negative, the system call returns with an `E_PAR` error.

This system call returns with an `E_LIMIT` error if invoked while as many rendezvous ports as the system's upper limit (the maximum number of rendezvous ports specified in the configurator) have been created. The system call returns with an `E_CTX` error if issued from a task-independent portion. Even when `pk_cpor` is invalid, no error checking is performed, where operation is not guaranteed.

3.6.3.2 tk_del_por (Delete Port for Rendezvous)

Deletes a rendezvous port.

Task portion	<input type="radio"/>	Task-independent portion	×	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	---	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_del_por ( ID porid ) ;
```

■ **Parameter**

● **Input**

porid Rendezvous port ID (Port ID)

● **Output**

ercd Error code (Error Code)

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (porid is less than or equal to zero, or greater than the maximum number of rendezvous ports)
E_NOEXS	-42	Object does not exist (The rendezvous ports specified by porid does not exist)
E_CTX	-25	Context error (in execution from task independent portion)

■ **Dispatch Trigger**

If a task higher in priority than the task having issued this system call is waiting in a rendezvous port deleted, the system call dispatches control to the task released from the wait state.

■ Description

This system call deletes the rendezvous port specified by `porid`. Precisely, the system call puts that rendezvous port into the ungenerated state and unallocates its ID number.

The system call returns with an `E_ID` error if the rendezvous port ID specified by `porid` is less than or equal to 0 or greater than the maximum number of rendezvous ports (set in the configurator). The system call returns error `E_NOEXS` if the rendezvous port does not exist.

Even when the specified rendezvous port has a task waiting for rendezvous acceptance (`tk_acp_por`) or rendezvous call (`tk_cal_por`), this system call terminates normally. For the waiting task, however, the system call returns with an `E_DLT` error. Even though the rendezvous port is deleted by `tk_del_por`, the task is not affected once it has established the rendezvous. The rendezvous accepting task (not in the wait state) receives no notification and the rendezvous calling task (in the rendezvous termination wait state) remains unchanged in status. `tk_rpl_rdv` is executed normally even if the rendezvous port used for establishing rendezvous by the rendezvous accepting task when issuing `tk_rpl_rdv` has already been deleted.

The system call returns with an `E_CTX` error if issued from a task-independent portion.

3.6.3.3 tk_cal_por (Call Port for Rendezvous)

Call port for Rendezvous

Task portion	<input type="radio"/>	Task-independent portion	<input checked="" type="checkbox"/>	Dispatch disabled	<input checked="" type="checkbox"/>
--------------	-----------------------	--------------------------	-------------------------------------	-------------------	-------------------------------------

■ C Language Interface

```
INT rmsgsz = tk_cal_por ( ID porid, UINT calptn, VP msg, INT cmsgsz, TMO tmout ) ;
```

■ Parameter

● Input

porid	Rendezvous port ID (Port ID)
calptn	Bit pattern that represents the selection conditions on call side (Call Bit Pattern)
msg	Address to enter messages (Message)
cmsgsz	Size of the call message (Byte count) (Call Message Size)
tmout	Timeout specification (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7ffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

rmsgsz	Size of replied message (Byte count) (Reply Message Size) or error code (Error Code)
--------	---

■ Error Code

E_ID	-18	Invalid ID number (porid is less than or equal to zero, or greater than the maximum number of rendezvous ports)
E_NOEXS	-42	Object does not exist (The rendezvous ports specified by porid does not exist)
E_PAR	-17	Parameter error (cmsgsz < 0, cmsgsz > maxcmsz, calptn = 0, tmout ≤ (-2))
E_DLT	-51	Wait object has been deleted (The target rendezvous ports were deleted while in the wait state)
E_RLWAI	-49	Forcibly restored from the wait state (accept tk_rel_wai while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that issued this system call enters the rendezvous call wait state, the task that is next in the priority ranking is dispatched. Or when the rendezvous accept wait state was released for a task that has a higher priority ranking than the task that issued this system call, the task for which the wait state was released was dispatched.

■ Description

This system call issues the rendezvous for the rendezvous port specified by porid.

When the rendezvous port ID that is specified by porid is less than or equal to zero, or greater than the maximum number of rendezvous ports (the maximum number of rendezvous ports set in the configurator, returns E_ID error. If the target rendezvous port does not exist, returns E_NOEXS error.

The size of call message is specified by cmsgsz. Data of cmsgsz byte in the msg area that is specified by this system call (msg area of the call side task) is copied to the msg area that is specified by tk_acp_por (msg area of the accept side task).

When cmsgsz is greater than maxcmsz that is specified by tk_cre_por, returns E_PAR error. This error is checked before the task enters the rendezvous call wait state, and if the error is found, the task that executes this system call does not enter that state. If cmsgsz is negative or greater than maxcmsz, returns E_PAR error.

The maximum wait time (timeout) can be specified in tmout. If a timeout is specified and the tmout time has elapsed without fulfilling the release wait condition (a rendezvous does not get effected), returns E_TMOUT error.

TMO_FEVR means an infinite timeout. In this case, the task remains in the wait state until the rendezvous gets effected or tk_rel_wai is issued. If TMO_POL is specified, the task does not enter the wait state, even when there is no rendezvous accept wait task in the target rendezvous port or the conditions to get rendezvous effected are not fulfilled, and it returns error E_TMOUT.

The time unit for tmout is the same as for the system timer (= 1ms). Returns E_PAR error if tmout is -2 or less.

The detailed operation of this system call is described below:

If there is a task in the rendezvous accept wait state in the rendezvous port specified by `porid`, and the conditions to get rendezvous effected are fulfilled between that task and the task that issued this system call, the rendezvous is effected. In this case, the task that was in the rendezvous accept wait state becomes executable, and the task that issued this system call enters the rendezvous end wait state. The task in the rendezvous end wait state is released its wait state when the other task (the rendezvous accept task) executes `tk_rpl_rdv` system call. Then, this system call ends at this point.

When there is no rendezvous accept wait task in the rendezvous port specified by `porid` or there are some rendezvous accept wait tasks but the conditions to get rendezvous effected are not fulfilled, the task that issued this system call enters the rendezvous call wait state, queuing in the rendezvous port call side. The order in the rendezvous call wait queue follows the FIFO or task priority ranking order according the specification by `tk_cre_por`.

The conditions to get rendezvous effected are judged by whether the logical product of `acpptn` (accept side task) and `calptn` (call side task) is 0 or not. The rendezvous is effected when the logical product is not 0. If `calptn` is 0, the rendezvous will never get effected and `E_PAR` error is returned.

When the rendezvous is effected, the call side task can send a message (call message) to the accept side task.

On the contrary, when the rendezvous is terminated, the accept side task can send a message (reply message) to the call side task. The content of the reply message that is specified by `tk_rpl_rdv` (reply message of the call side task) is copied to the message area that is specified by this system call (`msg` area of the call side task). The size of the reply message, `rmsgsz`, becomes the return value of this system call. Eventually, the message area specified by `msg` parameter of this system call is destroyed by the message that is sent when executing `tk_rpl_rdv`.

If a rendezvous is forwarded, the same amount of area as `maxrmsz` is used as buffer at the maximum from the `msg` address that is specified by this system call, therefore, the contents of that area may be destroyed. For this reason, if there is a possibility that the rendezvous required by this system call is forwarded, keep an area with the size of at least `maxrmsz` under `msg` despite of the size of the reply message (See, 3.6.3.5 `tk_fwd_por` (Forward Rendezvous to Another Port) for more details).

If the target rendezvous port is deleted by `tk_del_por` while in the rendezvous call or rendezvous end wait state, those wait states are released and this system call returns `E_DLT` error. If `tk_rel_wai` system call is issued for the task that is in the rendezvous call or rendezvous end wait state, those wait states are released and `E_RLWAI` error is returned. Returns `E_CTX` error if this system call is called from a task-independent portion or when dispatch is disabled.

No error check is performed even if `msg` is invalid. The operation in this case is not guaranteed.

3.6.3.4 tk_acp_por (Accept Port for Rendezvous)

Accept port for Rendezvous

Task portion	<input type="radio"/>	Task-independent portion	<input checked="" type="checkbox"/>	Dispatch disabled	<input checked="" type="checkbox"/>
--------------	-----------------------	--------------------------	-------------------------------------	-------------------	-------------------------------------

■ C Language Interface

```
INT cmsgsz = tk_acp_por ( ID porid, UINT acpptn, RNO *p_rdvno, VP msg, TMO tmout ) ;
```

■ Parameter

● Input

porid	Rendezvous port ID (Port ID)
acpptn	Bit pattern that represents the accept side selection condition (Accept Bit Pattern)
p_rdvno	Address to enter the rendezvous number (Rendezvous Number)
msg	Address to enter messages (Message)
tmout	Timeout specification (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7ffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

cmsgsz	Size of the call message (Byte count) or error code (Call Message Size or Error Code)
--------	--

■ Error Code

E_ID	-18	Invalid ID number (porid is less than or equal to zero, or greater than the maximum number of rendezvous ports)
E_NOEXS	-42	Object does not exist (The rendezvous ports specified by porid does not exist)
E_PAR	-17	Parameter error (acpptn = 0, tmout ≤ (-2))
E_DLT	-51	Wait object has been deleted (The target rendezvous ports were deleted while in the wait state)
E_RLWAI	-49	Forcibly restored from the wait state (accept tk_rel_wai while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that issued this system call enters the rendezvous accept wait state, the task that is next in the priority ranking is dispatched. Or when the rendezvous call wait state was released for a task that has a higher priority ranking than the task that issued this system call, the task for which the wait state was released was dispatched.

■ Description

This system call accepts the rendezvous for the rendezvous port.

Returns E_ID error if the rendezvous port ID specified by porid is less than or equal to zero or greater than the maximum number of rendezvous ports (maximum number of rendezvous ports specified in the configurator). Returns E_NOEXS error if the target rendezvous port does not exist.

The maximum wait time (timeout) can be specified in tmout. If a timeout is specified and the tmout time has elapsed without fulfilling the release wait condition (a rendezvous does not get effected), returns E_TMOUT error.

TMO_FEVR means an infinite timeout. In this case, the task remains in the wait state until the rendezvous gets effected or tk_rel_wai is issued. If TMO_POL is specified, the task does not enter the wait state, even when there is no rendezvous call wait task in the target rendezvous port or the conditions to get rendezvous effected are not fulfilled, and it returns error E_TMOUT.

The time unit for tmout is the same as for the system timer (= 1ms). If tmout is -2 or less, the function returns with E_PAR error.

The detailed operation of this system call is described below:

If the conditions to get rendezvous effected are fulfilled between the task that is in the wait queue on the call side of the rendezvous port specified by porid and this task, the rendezvous is effected. In this case, the task that has been in the wait queue on the call side is popped out of the queue, and changed its state from the rendezvous call wait (waiting the rendezvous gets effected) to the rendezvous end wait. The call task of this system call continues executing.

When there is no task in the wait queue on the call side of the rendezvous port specified by porid or there are some tasks but the conditions to get rendezvous effected are not fulfilled, the task of this system call enters the rendezvous accept wait state for that rendezvous port.

In this case, the error will not occur even if another task has been in the rendezvous accept wait state, and the task that issued this system call is put in the rendezvous accept wait queue. It is also possible that multiple tasks execute their rendezvous at a time using one rendezvous port. Therefore, a rendezvous that is executed before another task has finished its rendezvous (i.e., before the `tk_rpl_rdv` for the previously effected rendezvous is executed) at the same rendezvous port specified by `porid` will be normally terminated.

The conditions to get rendezvous effected are judged by whether the logical product of `acpptn` (receive side task) and `calptn` (call side task) is 0 or not. The rendezvous is effected when the logical product is not 0. When the first task does not fulfill the conditions, the next task in the wait queue will be checked. If a same value other than 0 is specified in `calptn` and `acpptn`, it means that there is no condition (unconditional). If `acpptn` is 0, the rendezvous will never get effected and `E_PAR` error is returned. The processes required to get a rendezvous effected are completely symmetrical between the rendezvous call side and accept side.

When the rendezvous is effected, the call side task can send a call message to the accept side task. The contents of the call message specified by the call side task is copied to the area under `msg` that is specified by the accept side task with this system call. The size of the call message, `cmsgsz`, becomes the return value of this system call.

It is also possible that the task on the rendezvous accept side executes multiple rendezvous at a time. To be more precise, this system call can be executed again before the task that accepted a rendezvous from this system call executes `tk_rpl_rdv`. In addition, this system call in this case can be meant for the other rendezvous port than the previous port or for the same rendezvous port as the previous one. This is a rare case, however, if the task during rendezvous executes another system call to the same rendezvous port and the rendezvous gets effected, the state where the same task is executing multiple (multiplex) rendezvous to the same rendezvous port is established. Of course, the tasks to rendezvous with (the call side task) are different in this case.

The rendezvous number (`p_rdvno`) that is returned as the return value of this system call is the information to tell the multiple rendezvous being established at the same time, and also used as the parameter of `tk_rpl_rdv` when the rendezvous are terminated. This value is also used as the parameter of `tk_fwd_por` when the rendezvous are forwarded. A rendezvous number is allocated so that the lower 16 bits contain the ID of the task that accepts the rendezvous and the upper 16 bits contain a unique value assigned sequentially in the order of rendezvous accepted.

If the target rendezvous port is deleted by `tk_del_por` while in the rendezvous accept or rendezvous end wait state, those wait states are released and this system call returns `E_DLT` error. If `tk_rel_wai` system call is issued for the task that is in the rendezvous accept or rendezvous end wait state, those wait states are released and `E_RLWAI` error is returned. Returns `E_CTX` error if this system call is called from a task-independent portion or when dispatch is disabled.

No error check is performed even if `p_rdvno` and `msg` are invalid. The operation in this case is not guaranteed.

3.6.3.5 tk_fwd_por (Forward Rendezvous to Another Port)

Forward Rendezvous to another port

Task portion	○	Task-independent portion	×	Dispatch disabled	○
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_fwd_por ( ID porid, UINT calptn, RNO rdvno, VP msg, INT cmsgsz ) ;
```

■ Parameter

● Input

porid	ID of the rendezvous port to be forwarded (Port ID)
calptn	Bit pattern that represents the selection conditions on call side (Call Bit Pattern)
rdvno	Rendezvous number before forwarding (Rendezvous Number)
msg	Address to enter the call messages (Message)
cmsgsz	Size of the call message (Byte count) (Call Message Size)

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (porid is less than or equal to zero, or greater than the maximum number of rendezvous ports)
E_NOEXS	-42	Object does not exist (The rendezvous ports specified by porid does not exist)
E_PAR	-17	Parameter error (cmsgsz < 0, cmsgsz > maxcmsz after forwarding, cmsgsz > maxrmsz before forwarding, calptn = 0)
E_OBJ	-41	Object state is invalid (The task ID included in rdvno is negative or greater than the maximum number of tasks, the state on the call side task is other than the rendezvous end wait (TTW_RDV), rdvno does not match with the return value of tk_acp_por, maxrmsz after forwarding > maxrmsz before forwarding)
E_CTX	-25	Context error (in execution from task independent portion)

■ Dispatch Trigger

When the rendezvous accept wait state was released for a task that has a higher priority ranking than the task that issued this system call, the task for which the wait state was released was dispatched.

■ Description

This system call forwards the rendezvous that has been accepted to another rendezvous port.

Returns E_ID error if the rendezvous port ID specified by porid is less than or equal to zero or greater than the maximum number of rendezvous ports (maximum number of rendezvous ports specified in the configurator). Returns E_NOEXS error if the target rendezvous port does not exist.

This system call should be issued from the task (Task X) that is currently in the rendezvous state (the state after tk_acp_por is executed). For example, consider the call side task to rendezvous with to be "Task Y," and the rendezvous number that is returned as the return value of tk_acp_por to be "rdvno". If this system call is executed in this case, the situation is equal to the one in which the rendezvous state between Task X and Task Y is released, and then Task Y calls the rendezvous for the other rendezvous port (rendezvous port B) that is specified by porid.

The detailed operation of this system call is described below:

1. The rendezvous that is specified by rdvno is released.
2. Task Y enters the rendezvous call wait state for the rendezvous port specified by porid. In this case, as for the bit pattern "calptn" that represents the conditions to get the rendezvous effected on the call side, the one specified by Task X with this system call is used, instead of the one specified by Task Y with tk_cal_por. From the point of view of Task Y, its state is changed from the rendezvous end wait back to the rendezvous call wait.
3. Then, if the rendezvous for the rendezvous port specified by porid is accepted, the rendezvous is effected between the task that accepted it and Task Y. If a task in the rendezvous accept wait state already exists in the rendezvous port specified by porid and the conditions to get the rendezvous effected are also fulfilled, executing this system call may get the rendezvous effected immediately. In this case, as for the message sent to the accept side when the rendezvous gets effected, the one specified by Task X with this system call is used, instead of the one specified by Task Y tk_cal_por (same as "calptn").
4. The message returned with tk_rpl_rdv when a new rendezvous ends is copied to the area msg that is specified by Task Y with tk_cal_por, instead of the area msg that is specified by Task X with this system call.

Basically, the situation in which "tk_cal_por (porid=portA, alptn=ptnA, msg=mesA) is executed before tk_fwd_por (porid=portB, calptn=ptnB, msg=mesB)" and the one in which "tk_cal_por (porid=portB, calptn=ptnB, msg=mesB) is executed" are same. You don't need to remember the history of rendezvous forwarding.

When tk_ref_tsk is executed to the task that is back to the rendezvous call wait state by this system call, tskwait becomes TTW_CAL. In addition, wid also becomes the ID of the rendezvous port to be forwarded to.

The execution of this system call will end immediately. Tasks never enter the wait state by this system call. In addition, after executing this system call, the task that issued this system call is no longer related to any of the rendezvous port where the rendezvous got effected before forwarded, the rendezvous port after forwarded (porid), and the tasks that rendezvoused on those ports.

When cmsz is greater than maxcmsz of the rendezvous port to which the rendezvous is forwarded, returns E_PAR error. This error is checked before the rendezvous is forwarded. When the error occurs, the rendezvous won't be forwarded and the rendezvous specified by rdvno won't be released either.

When `cmsgsz` is less than or equal to 0, `cmsgsz` is greater than `maxcmsz` after forwarding, or `cmsgsz` is greater than `maxrmsz` before forwarding, `E_PAR` error is returned.

The transmission message specified by this system call is copied to the other area (e.g. the message area specified by `tk_cal_por`) when executing this system call. Therefore, if the contents of the message area specified by `msg` of this system call is changed before the forwarded rendezvous gets effected, the rendezvous won't be affected by the change.

When forwarding a rendezvous by this system call, set the `maxrmsz` of the forwarding rendezvous port (the port specified by `porid`) to the value less than or equal to the `maxrmsz` of the rendezvous port in which the rendezvous got effected before forwarding. Otherwise, `E_OBJ` error is returned because the forwarding rendezvous port is considered to be inappropriate. On the rendezvous call side, the reply message accept area is prepared for the `maxrmsz` of the forwarded rendezvous port, therefore, if the maximum size of the reply message gets larger by the forwarding, an unexpected size of reply message may be returned to the call side and this might be a problem. This is the reason why a rendezvous cannot be forwarded to the rendezvous port that has large `maxrmsz`.

As for the `cmsgsz`, the size of the message sent by this system call, it should also be set to the value less than or equal to the `maxrmsz` of the forwarded rendezvous port where the rendezvous got effected. The reason of doing this is that the message area specified by `tk_cal_por` is assumed to use as the buffer of the transmission message according to the implementation method of this system call. When `cmsgsz` is greater than the `maxcmsz` of the forwarded rendezvous port, returns `E_PAR` error.

Even when this system call and `tk_rpl_rdv` are issued by the task for which a dispatch or interruption is disabled, these system calls operate normally. This can be used in a process where this system call or `tk_rpl_rdv` will be used inseparably.

If Task Y that has been in the rendezvous end wait state is back to the rendezvous call wait state by this system call, the timeout by the next rendezvous gets effected is always handled as the "Wait indefinitely" (`TMO_FEVR`).

The forwarding rendezvous port can be the same rendezvous port as the one used for the previous rendezvous (the port where the rendezvous specified by `rdvno` got effected). In this case, the process for the rendezvous that is once accepted is stopped by this system call. However, in this case too, the call message or `calptn` is changed to the one that is specified by the accept side task with this system call, rather than the one specified by the call side task with `tk_cal_por`. In addition, the rendezvous that has been forwarded can be forwarded again.

If the `calptn` is 0, `E_PAR` error is returned. When the task ID included in `rdvno` is negative or greater than the maximum number of tasks, the state on the call side task isn't the wait state, the state on the call side task is other than the rendezvous end wait (`TTW_RDV`), `rdvno` does not match with the return value of `tk_acp_por`, or `maxrmsz` after forwarding is greater than the `maxrmsz` before forwarding, `E_OBJ` error is returned.

No error check is performed even if `msg` is invalid. The operation in this case is not guaranteed.

3.6.3.6 tk_rpl_rdv (Reply Rendezvous)

Reply rendezvous

Task portion	<input type="radio"/>	Task-independent portion	×	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	---	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_rpl_rdv ( RNO rdvno, VP msg, INT rmsgsz ) ;
```

■ Parameter

● Input

rdvno	Rendezvous number (Rendezvous Number)
msg	Address to enter messages (Message)
rmsgsz	Size of the reply message (Byte count) (Reply Message Size)

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_PAR	-17	Parameter error (rmsgsz < 0, rmsgsz > maxrmsz)
E_OBJ	-18	Object state is invalid (The task ID included in rdvno is less than or equal to 0, or greater than the maximum number of tasks, the state on the call side task isn't the wait state, the state on the call side task is other than the rendezvous end wait (TTW_RDV), and rdvno does not match with the return value of tk_acp_por)
E_CTX	-25	Context error (in execution from task independent portion)

■ Dispatch Trigger

When the rendezvous end wait state was released for a task that has a higher priority ranking than the task that issued this system call, the task for which the wait state was released was dispatched.

■ Description

This system call replies to the task to rendezvous with (the call side task) and terminates the rendezvous.

This system call should be issued from the task (task X) that is in the rendezvous state (the state after `tk_acp_por` is executed). For example, consider the call side task to rendezvous with to be "Task Y," and the rendezvous number that is returned as the return value of `tk_acp_por` to be "rdvno". If this system call is executed in this case, the rendezvous state between Task X and Task Y is released, and then Task Y on the call side that has been in the rendezvous end wait state is back to the executable state.

When the rendezvous is terminated using this system call, the accept side Task X can send a replay message to the call side Task Y. The contents of the reply message specified by the accept side task is copied to the area under `msg` that is specified by the call side task with `tk_cal_por`. The size of the reply message, `rmsgsz`, becomes the return value of `tk_cal_por`.

When `rmsgsz` is negative or greater than the `maxcmsz` that is specified by `tk_cre_por`, returns `E_PAR` error. When this error is detected, the rendezvous won't be terminated and the rendezvous end wait state of the task that executed `tk_cal_por` won't be released, either.

When the task ID included in `rdvno` (lower 16-bit of `rdvno`) is less than or equal to 0, or greater than the maximum number of tasks (the maximum number of rendezvous ports set in the configurator), the state on the call side task isn't the wait state, the state on the call side task is other than the rendezvous end wait (`TTW_RDV`), and `rdvno` does not match with the return value of `tk_acp_por`, `E_OBJ` error is returned.

No error check is performed even if `msg` is invalid. The operation in this case is not guaranteed.

3.6.3.7 tk_ref_por (Refer Port Status)

Refers the state of the rendezvous.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_por ( ID porid, T_RPOR *pk_rpor ) ;
```

```
typedef struct t_rpor {
    VP  exinf;
    ID  wtsk;
    ID  atsk;
    INT maxcmsz;
    INT maxrmsz;
} T_RPOR;
```

■ Parameter

● Input

porid	Rendezvous port ID (Port ID)
pk_rpor	The first address of the packet that returns the rendezvous port state. (Packet of Refer Port)

● Output

ercd	Error code (Error Code)
● Data returned to the packet	
exinf	Extended information (Extended Information)
wtsk	Whether to exist the call wait task (Wait Task)
atsk	Whether to exist the accept wait task (Accept Task)
maxcmsz	Maximum length of the message when calling (Byte count) (Maximum Call Message Size)
maxrmsz	Maximum length of the message when replying (Byte count) (Maximum Receive Message Size)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (porid is less than or equal to zero, or greater than the maximum number of rendezvous ports)
E_NOEXS	-42	Object does not exist (The rendezvous ports specified by porid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call refers the state of the target rendezvous ports specified by porid and returns whether to exist the accept wait (atsk), whether to exist the call wait task (wtsk), the maximum length of the message (maxcmsz, maxrmsz), or the extended information (exinf) as its return value.

Returns E_ID error if the rendezvous port ID specified by porid is less than or equal to zero, or greater than the maximum number of rendezvous ports (maximum number of rendezvous ports specified in the configurator). Returns E_NOEXS error if the rendezvous port does not exist.

Wtsk set to the ID of the task that is in the rendezvous call wait state in this rendezvous port. If there is no rendezvous call wait task, wtsk is 0. On the other hand, atsk set to the ID of the task that is in the rendezvous accept wait state in this rendezvous port. If there is no rendezvous accept wait task, atsk is 0.

When there are multiple tasks that are in the rendezvous call or accept wait state in this rendezvous port, each ID of the first task in the call wait queue or accept wait queue is returned.

No error check is performed even if pk_rpor is invalid. The operation in this case is not guaranteed.

■ Additional Notes

It is impossible to know the information about the task that is currently rendezvousing with this system call.

3.7 Memory Pool Management Function System Calls

This section explains the memory pool management function system calls.

■ Memory Pool Management Function System Calls

Memory pool function consists of the system calls that have the following 2 types of function:

- Fixed-size memory pool function system calls
- Variable-length memory pool function system calls

3.7.1 Fixed-size Memory Pool Function System Calls

This section explains the fixed-size memory pool function system calls.

■ Fixed-size Memory Pool Function System Calls

Fixed-size memory pool function consists of the following 5 system calls:

- `tk_cre_mpf` (Create Fixed-size MemoryPool)
- `tk_del_mpf` (Delete Fixed-size MemoryPool)
- `tk_get_mpf` (Get Fixed-size Memory Block)
- `tk_rel_mpf` (Release Fixed-size Memory Block)
- `tk_ref_mpf` (Refer Fixed-size MemoryPool Status)

3.7.1.1 tk_cre_mpf (Create Fixed-size MemoryPool)

Create the fixed-size memory pool.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ID mpfid = tk_cre_mpf ( T_CMPF *pk_cmpf ) ;
```

```
typedef struct t_cmpf {
    VP exinf;
    ATR mpfatr;
    INT mpfcnt;
    INT blfsz;
    VP bufptr;
} T_CMPF;
```

■ Parameter

● Input

pk_cmpf Information about fixed-size memory pool creation
(Packet of Create MemoryPool)

● Data to set in packet

exinf Extended information (Extended Information)

mpfatr Fixed-size memory pool attribute
(Fixed-size MemoryPool Attribute)
mpfatr:= (TA_TFIFO || TA_TPRI) | [TA_USERBUF]

Attribute	Value	Meaning
TA_TFIFO	0x00000000	Manages waiting tasks using a FIFO
TA_TPRI	0x00000001	Manages tasks based on priority
TA_USERBUF	0x00000020	Use the stack area specified by the user

mpfcnt Block count of entire fixed-size memory pool
(Fixed-size MemoryPool Block Count)

blfsz Memory block size (Byte count)
(Fixed-size MemoryPool Block Size)

bufptr Address of user buffer (Buffer Pointer)

● Output

`mpfid` Fixed-size memory pool ID (Fixed-size MemoryPool ID)
Or, error code (Error Code)

■ Error Code

<code>E_NOMEM</code>	- 33	Insufficient memory (Cannot secure the area for memory pool)
<code>E_LIMIT</code>	- 34	Fixed-size memory pool count is greater than the upper limit of the system
<code>E_RSATR</code>	- 11	Reserve attribute (undefined value is specified to <code>mpfatr</code>)
<code>E_PAR</code>	- 17	Parameter error (<code>mpfcnt</code> and <code>blfsz</code> are less than or equal to 0, <code>bufsz</code> is not in multiples of 4 (if <code>TA_USERBUF</code> is specified))
<code>E_CTX</code>	- 25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates the fixed-size memory pool and assigns the fixed-size memory pool ID to it. To be more precise, this system call secures the memory area used as the memory pool based on the information from `mpfcnt` and `blfsz`. Issuing `tk_get_mpf` to the memory block created above gets the memory block of the size specified by `blfsz` (byte count).

User can utilize `exinf` to store the information on the target memory pool. The information specified here can be taken out with `tk_ref_mpf`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS has no concern with the data in `exinf`.

If the following attributes are specified to `mpfatr`, they are ignored. Returns `E_RSATR` error if an undefined attribute is specified.

Attribute	Value	Meaning
<code>TA_RNG0</code>	0x00000000	Execute at protection level 0
<code>TA_RNG1</code>	0x00000100	Execute at protection level 1
<code>TA_RNG2</code>	0x00000200	Execute at protection level 2
<code>TA_RNG3</code>	0x00000300	Execute at protection level 3

TA_TFIFO and TA_TPRI can specify the order of the tasks in the wait queue of the memory pool in order to get their memories. If the attribute is TA_TFIFO, the task wait queue operates as a FIFO. If the attribute is TA_TPRI, the task wait queue is ordered by task priority.

When TA_USERBUF is specified to activate bufptr, the memory area of mpfcnt*blfsz bytes starting with bufptr is used as the memory pool area. In this case, OS doesn't provide the memory pool area. When TA_USERBUF isn't specified, bufptr is ignored and OS secures the memory pool area. When OS couldn't secure the fixed-size memory pool area, E_NOMEM error is returned. If mpfcnt and blfsz are 0 or less, E_PAR error is returned.

If this system call is issued when the fixed-size memory pool is created up to the upper limit of the system (the maximum number of fixed-size memory pools set in the configurator), E_LIMIT error is returned. No error check is performed even if pk_cmpf and bufptr are invalid. The operation in this case is not guaranteed.

■ Additional Notes

For the fixed-size memory pool, another memory pool should be prepared to change the block size. In other words, if you need several types of memory block size, you also need several memory pools for each size.

3.7.1.2 tk_del_mpf (Delete Fixed-size MemoryPool)

Delete fixed-size memory pool.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_del_mpf ( ID mpfid ) ;
```

■ Parameter

● Input

mpfid	Fixed-size memory pool ID (Fixed-size MemoryPool ID)
-------	---

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mpfid is less than or equal to zero, or greater than the maximum number of fixed-size memory pools)
E_NOEXS	-42	Object does not exist (The fixed-size memory pool specified by mpfid does not exist)
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When a task that has a higher priority ranking than the task that issued this system call is waiting in the deleted fixed-size memory pool, the task for which the wait state was released was dispatched.

■ Description

This system call deletes the fixed-size memory pool specified by `mpfid`. To be more precise, the target memory pool is made into the ungenerated state to release its ID number and entire area of the memory pool itself.

Returns `E_ID` error if the fixed-size memory pool ID specified by `mpfid` is less than or equal to zero, or greater than the maximum number of fixed-size memory pools (maximum number of fixed-size memory pools specified in the configurator). Returns `E_NOEXS` error if the fixed-size memory pool does not exist.

If this system call is invoked from a task-independent portion or while in the dispatch disabled state, it returns error `E_CTX`.

Even if there is a task that gets some memories from this memory pool, any check or notification won't be executed. This system call will be normally terminated even when all memory blocks haven't been returned.

This system call will also be normally terminated even when there is a task that is waiting to get some memories in the target memory pool, however, `E_DLT` error is returned to the task that has been in the wait state.

3.7.1.3 tk_get_mpf (Get Fixed-size Memory Block)

Gets the fixed-size memory block.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_get_mpf ( ID mpfid, VP *p_blf, TMO tmout ) ;
```

■ Parameter

● Input

mpfid	Fixed-size memory pool ID (Fixed-size MemoryPool ID)
tmout	Timeout specification (Timeout) The following macros can be specified in addition to the values from 0 to 0x7fffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd	Error code (Error Code)
p_blf	Start address of the memory block (Fixed-size Block Start Address)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mpfid is less than or equal to zero, or greater than the maximum number of fixed-size memory pools)
E_NOEXS	-42	Object does not exist (The fixed-size memory pool specified by mpfid does not exist)
E_PAR	-17	Parameter error (tmout ≤ (-2))
E_DLT	-51	Wait object has been deleted (The target memory pool were deleted while in the wait state)
E_RLWAI	-49	Forcibly restored from the wait state (accept tk_rel_wai while in the wait state)

E_TMOUT -50 Polling failed or timed out

E_CTX -25 Context error

(Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that issued this system call enters the memory pool wait state, the task that is next in the priority ranking is dispatched.

■ Description

This system call gets some memory blocks from the fixed-size memory pool specified by `mpfid`. The starting address of the acquired memory block is returned to `p_blf`. The same memory block size as the one that is specified by `blfsz` when creating the fixed-size memory pool is acquired. The zero clearance isn't executed for the acquired memory and the contents remain undefined.

Returns `E_ID` error if the fixed-size memory pool ID specified by `mpfid` is less than or equal to zero or greater than the maximum number of fixed-size memory pools (maximum number of fixed-size memory pools specified in the configurator). Returns `E_NOEXS` error if the fixed-size memory pool does not exist.

The maximum wait time (timeout) can be specified in `tmout`. If a timeout is specified and the `tmout` time has elapsed without fulfilling the release wait condition (an empty memory cannot be created), returns `E_TMOUT` error.

`TMO_FEVR` means an infinite timeout. In this case, the task remains in the wait state until the memory can be acquired or `tk_rel_wai` is issued. If `TMO_POL` is specified, the task does not enter the wait state, even when the memory cannot be acquired, and it returns error `E_TMOUT`.

The time unit for `tmout` is the same as for the system timer (= 1ms). If `tmout` is -2 or less, the function returns with `E_PAR` error.

When any memory block cannot be acquired from the specified memory pool, the call task 'tk_get_mpf' is put into the memory get wait queue for that memory pool and waits till the task is able to get some memories.

The order of the wait queue to get the memory block is either FIFO or the task priority ranking depending on the attribute of the memory pool.

If the target fixed-size memory pool is deleted by `tk_del_mpf` while in the fixed-size memory pool wait state, the wait state is released and this system call returns `E_DLT` error. If `tk_rel_wai` system call is issued for the task that is in the fixed-size memory pool wait state, the wait state is released and `E_RLWAI` error is returned. Returns `E_CTX` error if this system call is called from a task-independent portion or when dispatch is disabled.

No error check is performed even if `p_blf` is invalid. The operation in this case is not guaranteed.

3.7.1.4 tk_rel_mpf (Release Fixed-size Memory Block)

Release fixed-size memory block.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ **C Language Interface**

```
ER ercd = tk_rel_mpf ( ID mpfid, VP blf ) ;
```

■ **Parameter**

● **Input**

mpfid	Fixed-size memory pool ID (Fixed-size MemoryPool ID)
blf	Start address of the memory block (Fixed-size Block Start Address)

● **Output**

ercd	Error code (Error Code)
------	-------------------------

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mpfid is less than or equal to zero, or greater than the maximum number of fixed-size memory pools)
E_NOEXS	-42	Object does not exist (The fixed-size memory pool specified by mpfid does not exist)
E_PAR	-17	Parameter error (blf is outside of the range of the memory pool area, (blf - start address of the memory block) isn't the multiple number of the memory block)
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ **Dispatch Trigger**

When the memory pool wait state was released for a task that has a higher priority ranking than the task that issued this system call, the task for which the wait state was released was dispatched.

■ Description

This system call releases the memory block specified by blf to the fixed-size memory pool specified by mpfid.

Returns E_ID error if the fixed-size memory pool ID specified by mpfid is less than or equal to zero or greater than the maximum number of fixed-size memory pools (maximum number of fixed-size memory pools specified in the configurator). Returns E_NOEXS error if the fixed-size memory pool does not exist.

Executing this system call may have another task, which is waiting the memory in the memory pool specified by mpfid, get the memory, and release the wait state of that task.

The memory block must be returned to the fixed-size memory pool in which the memory block was acquired.

If the memory pool to which the memory block is released is different from the memory pool in which the memory block was acquired (blf is outside of the range of the memory pool area, (blf - start address of the memory block) isn't the multiple number of the memory block), E_PAR error is returned.

3.7.1.5 tk_ref_mpf (Refer Fixed-size MemoryPool Status)

Refers the state of the fixed-size memory pool.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ **C Language Interface**

```
ER ercd = tk_ref_mpf ( ID mpfid, T_RMPF *pk_rmpf ) ;
```

```
typedef struct t_rmpf {  
    VP  exinf;  
    ID  wtsk;  
    INT frbcnt;  
} T_RMPF;
```

■ **Parameter**

● **Input**

mpfid	Fixed-size memory pool ID (Fixed-size MemoryPool ID)
pk_rmpf	Packet address to which the state of the memory pool is returned (Packet of Refer Fixed-size MemoryPool)

● **Data to set in packet**

exinf	Extended information (Extended Information)
wtsk	Whether to exist the wait task (Wait Task)
frbcnt	Block count of free area (Free Block Count)

● **Output**

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mpfid is less than or equal to zero, or greater than the maximum number of fixed-size memory pools)
E_NOEXS	-42	Object does not exist (The fixed-size memory pool specified by mpfid does not exist)
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call refers the state of the target fixed-size memory pools specified by mpfid and returns the current free block count (frbcnt), whether to exist the wait task (wtsk), or the extended information (exinf) as its return value.

Returns E_ID error if the fixed-size memory pool ID specified by mpfid is less than or equal to zero or greater than the maximum number of fixed-size memory pools (maximum number of fixed-size memory pools specified in the configurator). Returns E_NOEXS error if the fixed-size memory pool does not exist.

If this system call is invoked from a task-independent portion or while in the dispatch disabled state, it returns error E_CTX.

Wtsk set to the ID of the task that is waiting in the fixed-size memory pool. If multiple tasks are waiting in this fixed-size memory pool, the ID of the first task in the wait queue is returned. If there is no waiting task, wtsk = 0. Under all circumstances, either frbcnt = 0 or wtsk = 0 can be realized.

No error check is performed even if pk_rmpf is invalid. The operation in this case is not guaranteed.

■ Additional Notes

For the frsz of tk_ref_mpl, total size of the free memory area is returned in byte, however, for the frbcnt of this system call, the free block count is returned.

3.7.2 Variable-size Memory Pool Function System Calls

This section explains the variable-size memory pool function system calls.

■ Variable-size Memory Pool Function System Calls

Variable-size memory pool function consists of the following 5 system calls:

- `tk_cre_mpl` (Create Variable-size MemoryPool)
- `tk_del_mpl` (Delete Variable-size MemoryPool)
- `tk_get_mpl` (Get Variable-size Memory Block)
- `tk_rel_mpl` (Release Variable-size Memory Block)
- `tk_ref_mpl` (Refer Variable-size MemoryPool Status)

3.7.2.1 tk_cre_mpl (Create Variable-size MemoryPool)

Creates the variable-size memory pool.

Task portion	<input type="radio"/>	Task-independent portion	<input checked="" type="checkbox"/>	Dispatch disabled	<input checked="" type="checkbox"/>
--------------	-----------------------	--------------------------	-------------------------------------	-------------------	-------------------------------------

■ C Language Interface

```
ID mplid = tk_cre_mpl ( T_CMPL *pk_cmpl ) ;
```

```
typedef struct t_cmpl {
    VP  exinf;
    ATR mplatr;
    INT mplsz;
    VP  bufptr;
} T_CMPL;
```

■ Parameter

● Input

`pk_cmpl` Information about variable-size memory pool creation
(Packet of Create MemoryPool)

● Data to set in packet

`exinf` Extended information (Extended Information)

`mplatr` Memory pool attribute (MemoryPool Attribute)

`mplatr := (TA_TFIFO || TA_TPRI) | [TA_USERBUF]`

Attribute	Value	Meaning
TA_TFIFO	0x0000000	Manages waiting tasks using a FIFO
TA_TPRI	0x0000000	Manages tasks based on priority
TA_USERBUF	0x0000020	Use the stack area specified by the user

`mplsz` Size of whole memory pool (Byte count)
(MemoryPool Size)

`bufptr` Address of user buffer (Buffer Pointer)

● Output

`mplid` Variable-size memory pool ID (MemoryPool ID)
Or, error code (Error Code)

■ Error Code

<code>E_NOMEM</code>	-33	Insufficient memory (Cannot secure the area for memory pool)
<code>E_LIMIT</code>	-34	Variable-size memory pool count is greater than the upper limit of the system
<code>E_RSATR</code>	-11	Reserve attribute (undefined value is specified to <code>mplatr</code>)
<code>E_PAR</code>	-17	Parameter error (<code>mplsz</code> is less than or equal to 0, or not in multiples of 4 (if <code>TA_USERBUF</code> is specified))
<code>E_CTX</code>	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates the variable-size memory pool and assigns the variable-size memory pool ID to it. To be more precise, this system call secures the memory area used as the memory pool based on the information from `mplsz`.

User can utilize `exinf` to store the information on the target memory pool. The information specified here can be taken out with `tk_ref_mpl`. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS has no concern with the data in `exinf`.

`mplatr` specifies the attribute of the variable-size memory pool. If the following attributes are specified to `mplatr`, they are ignored. Returns `E_RSATR` error if an undefined attribute is specified to `mplatr`.

Attribute	Value	Meaning
<code>TA_RNG0</code>	0x00000000	Execute at protection level 0
<code>TA_RNG1</code>	0x00000100	Execute at protection level 1
<code>TA_RNG2</code>	0x00000200	Execute at protection level 2
<code>TA_RNG3</code>	0x00000300	Execute at protection level 3

`TA_TFIFO` and `TA_TPRI` can specify the order of the tasks in the wait queue of the memory pool in order to get their memories. If the attribute is `TA_TFIFO`, the task wait queue operates as a FIFO. If the attribute is `TA_TPRI`, the task wait queue is ordered by task priority.

When `TA_USERBUF` is specified to activate `bufptr`, the memory area of `mplsz` bytes starting with `bufptr` is used as the memory pool area. In this case, OS doesn't provide the memory pool area. When `TA_USERBUF` isn't specified, `bufptr` is ignored and OS secures the memory pool area. When OS couldn't secure the variable-size memory pool area, `E_NOMEM` error is returned. If the `mplsz` is less than or equal to 0, `E_PAR` error is returned.

If this system call is issued when the variable-size memory pool is created up to the upper limit of the system (the maximum number of variable-size memory pools set in the configurator), `E_LIMIT` error is returned. If this system call is invoked from a task-independent portion or while in the dispatch disabled state, it returns error `E_CTX`. No error check is performed even if `pk_cmpl` and `bufptr` are invalid. The operation in this case is not guaranteed.

When the wait queue to get memory is formed by tasks, the memory is assigned to the starting task of the queue by priority. Even when there is a task that requires smaller memory size in the following queue, there is no opportunity for the task to get memory. For example, there are Task A (requiring memory size = 400) and Task B (requiring memory size = 100) in a variable-size memory pool, and they are waiting in this order. Then, a continuous free memory area (memory size = 200) is created by `tk_rel_mpl` of another task. In this case again, Task B requiring smaller memory size cannot get the memory first.

■ Additional Notes

When the order of the wait queue is changed as follows, the system tries to assign the memory to the task that is newly come to the first. As a consequence, the task to which the memory was assigned is released from the wait state. Therefore, it can be said that the task may get the memory and released from the wait state even if the memory wasn't released by `tk_rel_mpl`.

- The wait state of the first task in the memory get wait queue is forcibly released.
- The first task in the memory get wait queue is forcibly terminated.
- For the memory get wait queue ordered by the priority ranking, the priority of the tasks other than the first one is changed, and another task assumes the higher priority than the first one.

3.7.2.2 tk_del_mpl (Delete Variable-size MemoryPool)

Deletes the variable-size memory pool.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_del_mpl ( ID mplid ) ;
```

■ Parameter

- Input

<code>mplid</code>	Variable-size memory pool ID (MemoryPool ID)
--------------------	---

- Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mplid is less than or equal to zero, or greater than the maximum number of variable-length memory pools)
E_NOEXS	-42	Object does not exist (The variable-size memory pool ID specified by mplid does not exist)
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When a task that has a higher priority ranking than the task that issued this system call is waiting in the deleted variable-size memory pool, the task for which the wait state was released was dispatched.

■ Description

This system call deletes the variable-size memory pool specified by `mplid`. To be more precise, the target memory pool is made into the ungenerated state to release its ID number and entire area of the memory pool itself.

Returns `E_ID` error if the variable-size memory pool ID specified by `mplid` is less than or equal to zero or greater than the maximum number of variable-length memory pools (maximum number of variable-size memory pools specified in the configurator). Returns `E_NOEXS` error if the variable-size memory pool does not exist. If this system call is invoked from a task-independent portion or while in the dispatch disabled state, it returns error `E_CTX`.

Even if there is a task that gets some memories from this memory pool, any check or notification won't be executed. This system call will be normally terminated even when all memory blocks haven't been returned.

This system call will also be normally terminated even when there is a task that is waiting to get some memories in the target memory pool, however, `E_DLT` error is returned to the task that has been in the wait state.

3.7.2.3 tk_get_mpl (Get Variable-size Memory Block)

Gets the variable-size memory block.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
ER ercd = tk_get_mpl ( ID mplid, W blksize, VP *p_blk, TMO tmout ) ;
```

■ Parameter

● Input

mplid	Variable-size memory pool ID (MemoryPool ID)
blksize	Memory block size (Byte count)
tmout	Timeout specification (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7ffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

ercd	Error code (Error Code)
p_blk	Start address of the memory block (Block Start Address)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mplid is less than or equal to zero, or greater than the maximum number of variable-size memory pools)
E_NOEXS	-42	Object does not exist (The fixed-size memory pool specified by mplid does not exist)
E_PAR	-17	Parameter error (tmout ≤ (-2), blksize is less than or equal to 0, or greater than the size of the memory pool area)
E_DLT	-51	Wait object has been deleted (The target memory pool were deleted while in the wait state)

E_RLWAI	-49	Forcibly restored from the wait state (accept tk_rel_wai while in the wait state)
E_TMOUT	-50	Polling failed or timed out
E_CTX	-25	Context error

(Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

When the task that issued this system call enters the memory pool wait state, the task that is next in the priority ranking is dispatched.

■ Description

This system call gets the memory blocks specified by blksz (byte count) from the variable-size memory pool specified by mplid.

The starting address of the acquired memory block is returned to p_blk. The zero clearance isn't executed for the acquired memory and the contents remain undefined. When the memory cannot be acquired, the task that issued this system call will enter the wait state.

Returns E_ID error if the variable-size memory pool ID specified by mplid is less than or equal to zero or greater than the maximum number of variable-length memory pools (maximum number of variable-size memory pools specified in the configurator). Returns E_NOEXS error if the variable-size memory pool does not exist. If the blksz is less than or equal to 0, E_PAR error is returned.

The maximum wait time (timeout) can be specified in tmout. If a timeout is specified and the tmout time has elapsed without fulfilling the release wait condition (an empty memory cannot be created), returns E_TMOUT error.

TMO_FEVR means an infinite timeout. In this case, the task remains in the wait state until the memory can be acquired or tk_rel_wai is issued. If TMO_POL is specified, the task does not enter the wait state, even when the memory cannot be acquired, and it returns error E_TMOUT.

The time unit for tmout is the same as for the system timer (= 1ms). If tmout is -2 or less, the function returns with E_PAR error.

The order of the wait queue to get the memory block is either FIFO or the task priority ranking depending on the attribute of the memory pool.

If the target message buffer is deleted by tk_del_mpl while in the variable-size memory pool wait state, the wait state is released and this system call returns E_DLT error. If tk_rel_wai system call is issued for the task that is in the variable-size memory pool wait state, the wait state is released and E_RLWAI error is returned. Returns E_CTX error if this system call is called from a task-independent portion or when dispatch is disabled.

No error check is performed even if p_blk is invalid. The operation in this case is not guaranteed.

3.7.2.4 tk_rel_mpl (Release Variable-size Memory Block)

Releases the variable-size memory block.

Task portion	<input type="radio"/>	Task-independent portion	×	Dispatch disabled	×
--------------	-----------------------	--------------------------	---	-------------------	---

■ **C Language Interface**

```
ER ercd = tk_rel_mpl ( ID mplid, VP blk ) ;
```

■ **Parameter**

● **Input**

<code>mplid</code>	Variable-size memory pool ID (MemoryPool ID)
<code>blk</code>	Start address of the memory block (Block Start Address)

● **Output**

<code>ercd</code>	Error code (Error Code)
-------------------	-------------------------

■ **Error Code**

<code>E_OK</code>	0	Normal completion
<code>E_ID</code>	-18	Invalid ID number (<code>mplid</code> is less than or equal to zero, or greater than the maximum number of variable-size memory pools)
<code>E_NOEXS</code>	-42	Object does not exist (The variable-size memory pool ID specified by <code>mplid</code> does not exist)
<code>E_PAR</code>	-17	Parameter error (<code>blk</code> is outside if the range of the memory pool area)
<code>E_CTX</code>	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ **Dispatch Trigger**

When the memory pool wait state was released for a task that has a higher priority ranking than the task that issued this system call, the task for which the wait state was released was dispatched.

■ Description

This system call releases the memory block specified by `blk` to the variable-size memory pool specified by `mplid`.

Returns `E_ID` error if the variable-size memory pool ID specified by `mplid` is less than or equal to zero or greater than the maximum number of variable-length memory pools (maximum number of variable-size memory pools specified in the configurator). Returns `E_NOEXS` error if the variable-size memory pool does not exist.

Executing `tk_rel_mpl` may have another task, which is waiting the memory in the memory pool specified by `mplid`, get the memory, and release the wait state of that task.

The memory block must be released to the variable-size memory pool in which the memory block was acquired. If the memory pool to which the memory block is released is different from the memory pool in which the memory block was acquired, `E_PAR` error is returned.

■ Additional Notes

When the memory is returned to the variable-size memory pool where multiple tasks are waiting, if a memory amount more than the total memory amount requested by the multiple tasks is returned, the multiple tasks are released from their wait state at a time.

The priority order of the tasks after their wait state were released is same as the one when they were in the wait queue.

3.7.2.5 tk_ref_mpl (Refer Variable-size MemoryPool Status)

Refers the state of the variable-size memory pool.

Task portion	○	Task-independent portion	×	Dispatch disabled	×
--------------	---	--------------------------	---	-------------------	---

■ **C Language Interface**

```
ER ercd = tk_ref_mpl ( ID mplid, T_RMPL *pk_rmpl ) ;
```

```
typedef struct t_rmpl {  
    VP  exinf;  
    ID  wtsk;  
    INT frsz;  
    INT maxsz;  
} T_RMPL;
```

■ **Parameter**

● **Input**

<code>mplid</code>	Variable-size memory pool ID (MemoryPool ID)
<code>pk_rmpl</code>	Packet address to which the state of the memory pool is returned (Packet of Refer MemoryPool)

● **Output**

<code>ercd</code>	Error code (Error Code)
● Data returned to the packet	
<code>exinf</code>	Extended information (Extended Information)
<code>wtsk</code>	Whether to exist the wait task (Wait Task)
<code>frsz</code>	Total size of the free area (Byte count) (Free Memory Size)
<code>maxsz</code>	Size of maximum free area (Byte count) (Max Memory Size)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (mplid is less than or equal to zero, or greater than the maximum number of variable-size memory pools)
E_NOEXS	-42	Object does not exist (The variable-size memory pool ID specified by mplid does not exist)
E_CTX	-25	Context error (Executed from a task-independent portion or in the dispatch disabled state)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call refers the state of the target variable-size memory pools specified by mplid and returns the current total size of free area (frsz), acquirable maximum free area size (maxsz), whether to exist the wait task (wtsk), or the extended information (exinf) as its return value.

Returns E_ID error if the variable-size memory pool ID specified by mplid is less than or equal to zero or greater than the maximum number of variable-length memory pools (maximum number of variable-size memory pools specified in the configurator). Returns E_NOEXS error if the variable-size memory pool does not exist.

Wtsk set to the ID of the task that is waiting in the variable-size memory pool. If multiple tasks are waiting in this variable-size memory pool, the ID of the first task in the wait queue is returned. If there is no waiting task, wtsk = 0.

No error check is performed even if pk_rmpl is invalid. The operation in this case is not guaranteed.

3.8 Time Management Function System Calls

Time Management Function System Calls

■ Time Management Function System Calls

Time management function consists of the system calls that have the following 3 types of function:

- System time management function system calls
- Cyclic handler function system calls
- Alarm handler function system calls

3.8.1 System Time Management Function System Calls

This section explains the system time management function system calls.

■ System Time Management Function System Calls

System time management function consists of the following 4 system calls:

- `tk_set_tim` (Set Time)
- `tk_get_tim` (Get Time)
- `tk_get_otm` (Get Operating Time)
- `isig_tim` (Signal Time)

3.8.1.1 tk_set_tim (Set Time)

Sets the system time.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_set_tim ( SYSTIM *pk_tim ) ;

typedef struct systim {
    W    hi;
    UW   lo;
} SYSTIM;
```

■ Parameter

● Input

pk_tim Packet address that represents the current time
 (Packet of Current Time)

● Data to set in packet

hi Current time for the system settings (upper 32-bit)
lo Current time for the system settings (lower 32-bit)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK 0 Normal completion
E_PAR -17 Parameter error (pk_tim.hi is a negative value)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call sets the system time to the value specified by systim. The system time is the number of ms added up from 1985, 1, 1, 0 o'clock (GMT).
If hi is a negative value, E_PAR error is returned.
No error check is performed even if pk_tim is invalid. The operation in this case is not guaranteed.

■ Additional Notes

The relative time specified by RELTIM or TMO will remain unchanged even if the system time is changed using this system call while the system is operating. For example, if a task is set to timeout after 60s and the system time is changed 60s ahead with this system call while the task is waiting the timeout, the task will wait for exactly 60s instead of timeout immediately. Therefore, the system time when the task will timeout is changed by this system call.

3.8.1.2 tk_get_tim (Get Time)

Refers the current system time.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_get_tim ( SYSTIM *pk_tim ) ;

typedef struct systim {
    W    hi;
    UW   lo;
} SYSTIM;
```

■ **Parameter**

● **Input**

pk_tim Packet address that returns the current time
 (Packet of Current Time)

● **Output**

ercd Error code (Error Code)

● **Data returned to the packet**

hi Current system time (upper 32-bit)
lo Current system time (lower 32-bit)

■ **Error Code**

E_OK 0 Normal completion

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call reads the current system time and returns to the return value "pk_tim". The system time is the number of ms added up from 1985, 1, 1, 0:0:0 (GMT).
No error check is performed even if pk_tim is invalid. The operation in this case is not guaranteed.

3.8.1.3 tk_get_otm (Get Operating Time)

Refers the system operating time.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_get_otm ( SYSTIM *pk_tim ) ;
```

```
typedef struct systim {
    W    hi;
    UW   lo;
} SYSTIM;
```

■ Parameter

● Input

pk_tim Packet address that returns the operating time
(Packet of Operating Time)

● Output

ercd Error code (Error Code)

● Data returned to the packet

hi Current system time (upper 32-bit)

lo Current system time (lower 32-bit)

■ Error Code

E_OK 0 Normal completion

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call gets the system operating time. The system operating time is different from the system time (time) and represents the total operating time (in ms) that is added up from when the system is started.

This system operating time is unaffected by the time setting by tk_set_tim.

No error check is performed even if pk_tim is invalid. The operation in this case is not guaranteed.

3.8.1.4 isig_tim (Signal Time)

Provides the timetick.

Task portion	×	Task-independent portion	○	Dispatch disabled	○
--------------	---	--------------------------	---	-------------------	---

■ **C Language Interface**

```
ER ercd = isig_tim ( void ) ;
```

■ **Parameter**

● Input

None

● Output

ercd Error code (Error Code)

■ **Error Code**

E_OK 0 Normal completion

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call updates the system time. This is an original function of μT-REALOS.

The system time is updated by issuing this system call on every system timer (=1ms) from the user program. To do this, create a timer interrupt of which cycle is 1 ms, then issue this system call from that interrupt handler.

This system call cannot be issued from the task portion. If it is issued from the task portion, the operation won't be guaranteed.

3.8.2 Cyclic Handler Function System Calls

This section explains the cyclic handler function system calls.

■ Cyclic Handler Function System Calls

Cyclic handler pool function consists of the following 5 system calls:

- tk_cre_cyc (Create Cyclic Handler)
- tk_del_cyc (Delete Cyclic Handler)
- tk_sta_cyc (Start Cyclic Handler)
- tk_stp_cyc (Stop Cyclic Handler)
- tk_ref_cyc (Refer Cyclic Handler Status)

3.8.2.1 tk_cre_cyc (Create Cyclic Handler)

Create a cyclic handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID cycid = tk_cre_cyc ( T_CCYC *pk_ccyc ) ;
```

```
typedef struct t_ccyc {  
    VP  exinf;  
    ATR cycatr;  
    FP  cychdr;  
    RELTIM cyctim;  
    RELTIM cycphs;  
} T_CCYC;
```

■ Parameter

● Input

pk_ccyc Cyclic handler definition information
 (Packet of Create Cyclic Handler)

● Data to set in packet

exinf Extended information (Extended Information)
cycatr Cyclic handler attribute (Cyclic Handler Attribute)
 cycatr := (TA_HLNG) | [TA_STA] | [TA_PHS]

Attribute	Value	Meaning
TA_HLNG	0x00000001	Target cyclic handler is written in C language
TA_STA	0x00000002	Cyclic handler is started
TA_PHS	0x00000004	Cyclic handler phase is saved

cyhdr Cyclic handler address (Cyclic Handler Address)
cyctim Cycle time (Cycle Time)
cycphs Cyclic handler phase (Cyclic Handler Phase)

● Output

cycid Cyclic handler ID (Cyclic Handler ID) Or, error code (Error Code)

■ Error Code

E_LIMIT	-34	Cyclic handler count is greater than the limit of the system
E_RSATR	-11	Reserve attribute (undefined value is specified to cycatr)
E_PAR	-17	Parameter error (cychdr is NULL and cyctim is 0)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates the cyclic handler and assigns the cyclic handler ID to it. The cyclic handler is the handler of the task-independent portion that operates at the specified time interval.

User can utilize exinf to store the information on the target cyclic handler. The information specified here can be taken out with tk_ref_cyc or passed to the cyclic handler as a parameter. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in exinf. The OS has no concern with the data in exinf.

cycatr specifies the attribute of the cyclic handler. The following values are ignored. Returns E_RSATR error if an undefined attribute is specified.

Attribute	Value	Meaning
TA_ASM	0x00000000	Target cyclic handler is written in assemble language

It is stipulated in the μ T-Kernel specification that TA_HLNG should be specified when the cyclic handler is written in C language. However, μ T-REALOS only supports C language for writing the cyclic handler, therefore, even if TA_ASM is specified (TA_HLNG isn't specified) the cyclic handler is processed as it is written in C language. Accordingly, although specifying TA_HLNG is not mandatory, you should always specify TA_HLNG in any case for compatibility with the μ T-Kernel specifications.

When TA_STA is specified, the cyclic handler enters the operating state upon its generation, and the handler is activated at the time interval described above. Otherwise, the activation cycles are measured but the cyclic handler isn't activated.

If TA_PHS is specified, the cyclic handler is activated by tk_sta_cyc, but the activation cycle isn't reset and the cycle that has been measured from the generation of the handler is maintained. When TA_PHS isn't specified, the activation cycle is reset with tk_sta_cyc, and the cyclic handler is activated at the interval of cyctim when tk_sta_cyc is issued. In the reset with tk_sta_cyc, cycphs isn't applied. In this case, the n-th activation of the cyclic handler from tk_sta_cyc will occur when more than $\text{cyctim} * n$ time is elapsed from the issue of tk_sta_cyc.

cychdr specifies the first address of the cyclic handler to be activated. If the cychdr is NULL, E_PAR error is returned. No error check is performed even if cychdr is invalid. The operation in this case is not guaranteed. Refer to Section 4.6 Cyclic Handler in User's Guide for more details about the description format of the cyclic handler.

cycphs represents the time from when the cyclic handler is created by this system call to when the first cyclic handler is activated. Subsequently, the cycle activation is repeated at the interval of cyctim. In this case, the n-th activation of the cyclic handler will occur when more than $\text{cycphs} + \text{cyctim} * (n - 1)$ time is elapsed from the generation of the handler. When 0 is specified to cycphs, the cyclic handler is activated immediately after the generation of the cyclic handler. If the cyctim is less than or equal to 0, E_PAR error is returned.

If this system call is issued when the cyclic handler is created up to the upper limit of the system (the maximum number of cyclic handlers set in the configurator), E_LIMIT error is returned. No error check is performed even if pk_ccyc is invalid. The operation in this case is not guaranteed.

3.8.2.2 tk_del_cyc (Delete Cyclic Handler)

Deletes the cyclic handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_cyc ( ID cycid ) ;
```

■ Parameter

● Input

cycid	Cyclic handler ID (Cyclic Handler ID)
-------	--

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (cycid is less than or equal to zero, or greater than the maximum number of cyclic handlers)
E_NOEXS	-42	Object does not exist (The cyclic handler specified in cycid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

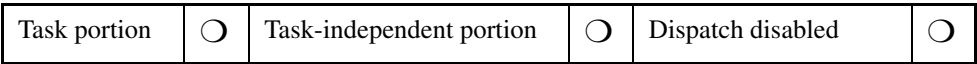
■ Description

This system call deletes the cyclic handler. To be more precise, the target cyclic handler is made into the ungenerated state to release its ID number.

Returns E_ID error if the cyclic handler ID specified by cycid is less than or equal to zero or greater than the maximum number of cyclic handlers (maximum number of cyclic handlers specified in the configurator). Returns E_NOEXS error if the cyclic handler does not exist.

3.8.2.3 tk_sta_cyc (Start Cyclic Handler)

Starts the operation of the cyclic handler.



■ **C Language Interface**

```
ER ercd = tk_sta_cyc ( ID cycid ) ;
```

■ **Parameter**

● **Input**

cycid	Cyclic handler ID (Cyclic Handler ID)
-------	--

● **Output**

ercd	Error code (Error Code)
------	-------------------------

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (cycid is less than or equal to zero, or greater than the maximum number of cyclic handlers)
E_NOEXS	-42	Object does not exist (The cyclic handler specified in cycid does not exist)

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call makes the cyclic handler specified by cycid into the operating state.

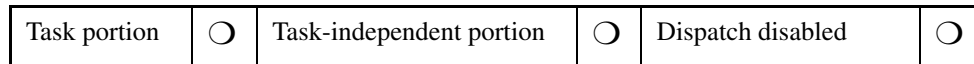
Returns E_ID error if the cyclic handler ID specified by cycid is less than or equal to zero or greater than the maximum number of cyclic handlers (maximum number of cyclic handlers specified in the configurator). Returns E_NOEXS error if the cyclic handler does not exist.

When TA_PHS attribute is specified, the activation cycle of the cyclic handler isn't reset, and the cyclic handler is made into the operating state. If it is already in the operating state, this state is maintained and then this system call is normally terminated.

When TA_PHS attribute isn't specified, the activation cycle is reset, and the cyclic handler is made into the operating state. If it is already in the operating state, the activation cycle is reset and then the operating state is maintained. Therefore, next activation of the cyclic handler occurs after cycitm.

3.8.2.4 tk_stp_cyc (Stop Cyclic Handler)

Stops the operation of cyclic handler.



■ C Language Interface

```
ER ercd = tk_stp_cyc ( ID cycid ) ;
```

■ Parameter

● Input

cycid	Cyclic handler ID (Cyclic Handler ID)
-------	--

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (cycid is less than or equal to zero, or greater than the maximum number of cyclic handlers)
E_NOEXS	-42	Object does not exist (The cyclic handler specified in cycid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call stops the cyclic handler specified by cycid. If it has already been in the stopped state, the normal termination is performed with the state retained.

It returns with the error E_ID if the cyclic handler ID specified by cycid is less than or equal to 0 or greater than the maximum number of cyclic handlers (maximum number of cyclic handlers specified by the configurator). It returns with the error E_NOEXS if the cyclic handler does not exist.

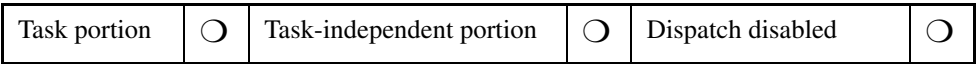
■ Additional Notes

Since the number of times is not specified in tk_cre_cyc, the cyclic handler defined once repeats the cyclic activation until it is stopped by tk_stp_cyc or the cyclic activation handler is deleted.

If multiple time event handlers are to be operated simultaneously, those handlers are activated serially (after the execution of one handler is finished, the execution of another handler is started). Also, since the time event handler is a task-independent portion, the principle of delay dispatch is applied.

3.8.2.5 tk_ref_cyc (Refer Cyclic Handler Status)

Refers to the cyclic handler state.



■ C Language Interface

```
ER ercd = tk_ref_cyc ( ID cycid, T_RCYC *pk_rcyc ) ;
```

```
typedef struct t_rcyc {  
    VP      exinf;  
    RELTIM  lfttim;  
    UINT    cycstat;  
} T_RCYC;
```

■ Parameter

● Input

cycid	Cyclic handler ID (Cyclic Handler ID)
pk_rcyc	Initial address of the packet to return the cyclic handler state (Packet of Refer Cyclic Handler)

● Output

ercd	Error code (Error Code)
------	-------------------------

● Data returned in packet

exinf	Extended Information
lfttim	Left time before next handler activation (Left Time)
cycstat	Cyclic handler status (Cyclic Handler Status) cycstat:= (TCYC_STP TCYC_STA)

State	Value	Meaning
TCYC_STP	0x00	Cyclic handler is not operated.
TCYC_STA	0x01	Cyclic handler is operated.

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (cycid is less than or equal to zero, or greater than the maximum number of cyclic handlers)
E_NOEXS	-42	Object does not exist (The cyclic handler specified in cycid does not exist)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call refers to the cyclic handler status indicated by cycid, and then returns the cyclic handler status, cycstat, left time before next handler activation, lfttim, and extended information, exinf, as return values.

It returns with the error E_ID if the cyclic handler ID specified by cycid is less than or equal to 0 or greater than the maximum number of cyclic handlers (maximum number of cyclic handlers specified by the configurator). It returns with the error E_NOEXS if the cyclic handler does not exist.

No error check is performed even if pk_rcyc is invalid. The operation in this case is not guaranteed.

3.8.3 Alarm Handler Function System Calls

The system calls of alarm handler function are explained.

■ Alarm Handler Function System Calls

The alarm handler function is comprised of five system calls as follows:

- tk_cre_alm (Create Alarm Handler)
- tk_del_alm (Delete Alarm Handler)
- tk_sta_alm (Start Alarm Handler)
- tk_stp_alm (Stop Alarm Handler)
- tk_ref_alm (Refer Alarm Handler Status)

3.8.3.1 tk_cre_alm (Create Alarm Handler)

Creates an alarm handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID almid = tk_cre_alm ( T_CALM *pk_calm ) ;
```

```
typedef struct t_calm {
    VP  exinf;
    ATR almatr;
    FP  almhdr;
} T_CALM;
```

■ Parameter

● Input

pk_calm Alarm handler definition information
(Packet of Create Alarm Handler)

● Data to set in packet

exinf Extended Information (Extended Information)
almatr Alarm handler attribute (Alarm Handler Attribute)
almatr := TA_HLNG

Attribute	Value	Meaning
TA_HLNG	0x00000001	Object alarm handler is described in C language.

almhdr Alarm handler address (Alarm Handler Address)

● Output

almid Alarm handler ID (Alarm Handler ID)
Or, error code (Error Code)

■ Error Code

E_LIMIT -34 Number of alarm handlers is greater than the system limitation.
E_RSATR -11 Reservation attribute (undefined value is specified to almatr.)
E_PAR -17 Parameter error (almhdr is NULL)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call creates an alarm handler and allocates an alarm handler ID. Alarm handler (specified time activation handler) is the handler of task-independent portion which is activated at specified time.

User can use `exinf` freely to put the information about the target alarm handler. The information specified here can be retrieved by `tk_ref_alm`, besides it can be passed to the alarm handler as a parameter. If you require a larger area to hold user data, or if you want to be able to update data during use, reserve memory for this purpose yourself and insert the address of your memory packet in `exinf`. The OS ignores the data in `exinf`.

`almatr` specifies the alarm handler attribute. The following values are ignored. The system call returns with the error `E_RSATR` if an undefined attribute is specified.

Attribute	Value	Meaning
TA_ASM	0x00000000	Object alarm handler is described in assembler.

With μ T-Kernel specifications, if the alarm handler is described in C language, the specification of `TA_HLNG` is required. However, with μ T-REALOS, since C language only is supported for the alarm handler description, regardless of availability of `TA_HLNG`, the alarm handler is processed as it is described in C language. Accordingly, although specifying `TA_HLNG` is not mandatory, you should always specify `TA_HLNG` in any case for compatibility with the μ T-Kernel specifications.

`almhdr` specifies the initial address of the alarm handler to be activated. If `almhdr` is `NULL`, the system call returns with the error `E_PAR`. Also, no error check is performed even if `almhdr` is invalid. The operation in this case is not guaranteed. For the description format for the alarm handler, see "4.7 Alarm Handler" in "User's Guide".

If this system call is invoked in the state where the alarm handlers are created up to the limit of the system (maximum number of alarm handlers specified by the configurator), it returns with `E_LIMIT` error. Also, no error check is performed even if `pk_calm` is invalid. The operation in this case is not guaranteed.

3.8.3.2 tk_del_alm (Delete Alarm Handler)

Deletes the alarm handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_del_alm ( ID almid ) ;
```

■ Parameter

● Input

almid	Alarm handler ID (Alarm Handler ID)
-------	--

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (almid is less than or equal to zero, or greater than the maximum number of alarm handlers)
E_NOEXS	-42	Object does not exist. (The alarm handler specified in almid does not exist.)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call deletes the alarm handler. Specifically, the object alarm handler is made to the uncreated status, and the ID number is released.

The system call returns with the error E_ID if the alarm handler ID specified by almid is less than or equal to zero or greater than the maximum number of alarm handlers (maximum number of alarm handlers specified by the configurator). It returns with the error E_NOEXS if the alarm handler does not exist.

3.8.3.3 tk_sta_alm (Start Alarm Handler)

Starts the operation of alarm handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_sta_alm ( ID almid, RELTIM almtim ) ;
```

■ **Parameter**

● **Input**

almid	Alarm handler ID (Alarm Handler ID)
almtim	Alarm handler activation time (Alarm Handler Time)

● **Output**

ercd	Error code (Error Code)
------	-------------------------

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (almid is less than or equal to zero, or greater than the maximum number of alarm handlers)
E_NOEXS	-42	Object does not exist. (The alarm handler specified in almid does not exist.)

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call specifies the activation time of alarm handler specified by almid to make it operated.

It returns with the error E_ID if the alarm handler ID specified by almid is less than or equal to zero or greater than the maximum number of alarm handlers (maximum number of alarm handlers specified by the configurator). It returns with the error E_NOEXS if the alarm handler does not exist.

Since almtim is relative time, after the time period specified by almtim is passed starting from the time when this system call is called, the alarm handler is activated. If the activation time of alarm handler has already been set and the alarm handler was in the operation status, after the setting is released, a new activation time is specified to make the alarm handler operated. When almtim=0, immediately after the activation time setting, the alarm handler is activated. After the processing of the activated alarm handler is completed, the system call becomes the stopped status.

3.8.3.4 tk_stp_alm (Stop Alarm Handler)

Stops the operation of alarm handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_stp_alm ( ID almid ) ;
```

■ Parameter

● Input

almid	Alarm handler ID (Alarm Handler ID)
-------	--

● Output

ercd	Error code (Error Code)
------	-------------------------

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (almid is less than or equal to zero, or greater than the maximum number of alarm handlers)
E_NOEXS	-42	Object does not exist. (The alarm handler specified in almid does not exist.)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call makes the alarm handler stopped. If the handler has been already in the stopped status, the status is kept, and a normal completion is performed.

The system call returns with the error E_ID if the alarm handler ID specified by almid is less than or equal to zero or greater than the maximum number of alarm handlers (maximum number of alarm handlers specified by the configurator). It returns with the error E_NOEXS if the alarm handler does not exist.

3.8.3.5 tk_ref_alm (Refer Alarm Handler Status)

Refers to the status of alarm handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_ref_alm ( ID almid, T_RALM *pk_ralm ) ;
```

```
typedef struct t_ralm {
    VP      exinf;
    RELTIM  lfttim;
    UINT    almstat;
} T_RALM;
```

■ **Parameter**

● **Input**

almid	Alarm handler ID (Alarm Handler ID)
pk_ralm	Initial address of the packet to return the alarm handler status (Packet of Refer Alarm Handler)

● **Output**

ercd	Error code (Error Code)
------	-------------------------

● **Data returned in packet**

exinf	Extended Information (Extended Information)
lfttim	Left time before next handler activation (Left Time)
almstat	Alarm handler status (Alarm Handler Status)
almstat := (TALM_STP TALM_STA)	

State	Value	Meaning
TALM_STP	0x00	Alarm handler is not operated.
TALM_STA	0x01	Alarm handler is operated.

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (almid is less than or equal to zero, or greater than the maximum number of alarm handlers)
E_NOEXS	-42	Object does not exist. (The alarm handler specified in almid does not exist.)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call refers to the status of alarm handler specified by almid, and returns the left time before the handler activation lfttim and extended information exinf as return values.

The system call returns with the error E_ID if the alarm handler ID specified by almid is less than or equal to zero or greater than the maximum number of alarm handlers (maximum number of alarm handlers specified by the configurator). It returns with the error E_NOEXS if the alarm handler does not exist.

If the alarm handler is operated (TALM_STA), the relative time before the next alarm handler activation is returned to lfttim. The value is specified by tk_sta_alm, almtim is equal to or more than lfttim, and lfttim is equal to or more than zero. Since lfttim is subtracted for each timer interrupt, if an alarm handler is activated at the next timer interrupt, lfttim becomes 0.

If an alarm handler is not operated (TALM_STP), lfttim is undefined.

No error check is performed even if pk_ralm is invalid. The operation in this case is not guaranteed.

3.9 Interrupt Control Function System Calls

The system calls of interrupt management function are explained.

■ Interrupt Management Function System Calls

The system calls of interrupt management function is comprised of two system calls as follows:

- `tk_def_int` (Define Interrupt Handler)
- `tk_ret_int` (Return from Interrupt Handler)

Also, the interrupt management function has three macros as follows:

- `DI`
- `EI`
- `isDI`

3.9.1 tk_def_int (Define Interrupt Handler)

Defines or releases the interrupt handler.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_def_int ( UINT dintno, T_DINT *pk_dint ) ;
```

```
typedef struct t_dint {
    ATR intatr;
    FP inthdr;
} T_DINT;
```

■ Parameter

● Input

dintno Interrupt vector number

pk_dint Initial address of interrupt handler definition information
(Packet of Define Interrupt Handler)

● Data to set in packet

intatr Interrupt handler attribute (Interrupt Handler Attribute)
intatr := (TA_ASM || TA_HLNG)

Attribute	Value	Meaning
TA_ASM	0x00000000	Object handler is described in assembler.
TA_HLNG	0x00000001	Object handler is described in C language.

inthdr Interrupt handler address
(Interrupt Handler Address)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK 0 Normal completion

E_RSATR -11 Reservation attribute (intatr is invalid or unavailable)

E_PAR -17 Parameter error (dintno is 256 or more)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call defines the interrupt handler and makes it available. Or it releases the interrupt handler definition. The interrupts include all of external interrupts from devices, interrupts by CPU exception, and software interrupts.

dintno specifies the interrupt vector number which defines an interrupt handler. Values from 0 to 255 can be specified. The system call returns with the error E_PAR, if the value 256 or more is specified to dintno. The interrupt handler can be redefined to the interrupt number already defined. Even in the case of redefinition, the definition release for the handler of the number dose not have to be executed.

pk_dint is the address for the T_DINT structure which holds the interrupt handler definition information. The relation between pk_dint value and pk_def_int operation is shown below.

Value of pk_dint	Operation of tk_def_int
Normal address (other than NULL)	Definition of interrupt handler
NULL	Definition release of interrupt handler
Invalid address	The operation is not guaranteed.

intatr specifies the description language for interrupt handler. The following shows the description languages and maximum registration numbers of handlers for TA_HLNG attribute and TA_ASM attribute. If an undefined attribute is specified to intatr, the system call returns with the error E_RSATR.

Attribute	Description Language	Maximum Registration Number
TA_ASM	Assembler	256
TA_HLNG	C language	8

inthdr specifies the initial address of interrupt handler. Since if the address of inthdr is illegal, the occurrence of corresponding interrupt makes the jump to the illegal address, the subsequent operation of the system is not guaranteed.

For the execution priority order of interrupt handlers and tasks, see "2.6 Execution Priority Order of Tasks/Handlers" in "User's Guide". For the description format for the interrupt handler, see "4.8 Interrupt Handler" in "User's Guide".

Note: With μ T-REALOS which uses the SOFTUNE language tool, even if the interrupt handler is described in C language, the specification of "__interrupt" requires the definition as TA_ASM attribute. The registration in TA_HLNG attribute prevents the normal operation.

3.9.2 tk_ret_int (Return from Interrupt Handler)

Returns from the interrupt handler.

Task portion	×	Task-independent portion	○	Dispatch disabled	○
--------------	---	--------------------------	---	-------------------	---

■ C Language Interface

```
void tk_ret_int ( void ) ;
```

■ Parameter

● Input

None

● Output

None

■ Error Code

None

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

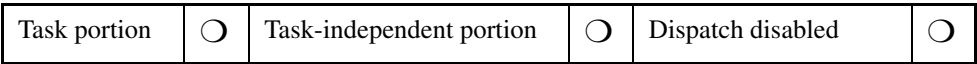
The invocation of this system call does not execute any processing and the system call returns to the source of call.

With μ T-Kernel specifications, "the invocation of this system call on the completion of the interrupt handler described in assemble language creates the delay dispatch" is specified. However, with μ T-REALOS, due to the dispatcher implementation reason, regardless of the invocation of this system call, the delay dispatch is created. Therefore, with the interrupt handler described in assemble language, this system call does not have to be invoked.

Also, to keep the compatibility with the μ T-Kernel specifications OS, this system call remains. Consequently with the interrupt handler described in C language, this system call should not be invoked.

3.9.3 DI

Prohibits all external interrupts.



■ **Interface**

```
void DI ( UINT intsts )
```

■ **Parameter**

● **Input**

None

● **Output**

intsts CPU interrupt status (PS register value)

■ **Error Code**

None

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call specifies CCR.I flag of PS register to 0 (interrupt disabled) to prohibit all external interrupts. The value of PS register before the interrupts prohibition is stored in intsts.

Example Usage

```
void foo()
{
    UINT intsts;

    DI(intsts);

    if ( isDI(intsts) ) {
        /* Interrupts have been already disabled when this function was called. */
    } else {
        /* Interrupts have been enabled when this function was called. */
    }

    EI(intsts);
}
```


3.9.4 EI

Enables all external interrupts.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ Interface

```
void EI ( UINT intsts )
```

■ Parameter

● Input

intsts CPU interrupt status (PS register value)

● Output

None

■ Error Code

None

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call enables all external interrupts. Precisely if the value of CCR.I flag of PS register specified by intsts is 1 (interrupt enabled), the CCR.I flag of PS register is specified to 1 to return to the interrupt enabled status.

When the value of CCR.I flag of PS register specified by intsts is 0 (interrupt disabled), even if EI() is executed, the interrupt is not enabled. However, if 0 is specified as intsts, the interrupt is always enabled.

intsts is either the value stored by DI() or 0. If other than that is specified, the operation is not guaranteed.

Example Usage

```
void foo()
{
    UINT intsts;

    DI(intsts);

    if ( isDI(intsts) ) {
        /* Interrupts have been already disabled when this function was called. */
    } else {
        /* Interrupts have been enabled when this function was called. */
    }

    EI(intsts);
}
```

3.9.5 isDI

Checks the external interrupt disabled status.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **Interface**

```
BOOL stat = isDI ( UINT intsts )
```

■ **Parameter**

● **Input**

intsts CPU interrupt status (PS register value)

● **Output**

stat TRUE (value other than "0"): Interrupt disabled status
 FALSE: Interrupt enabled status

■ **Error Code**

None

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call checks the interrupt enabled/disabled status from the PS register value stored in intsts. If the CCR.I flag as the PS register value stored in intsts, the interrupt disabled status is determined and if 1, the interrupt enabled status is determined. However, this system call can check only the interrupt disabled status before the issuance of DI() where intsts is stored. Use tk_ref_sys when the current interrupt disabled status is checked.

intsts is the value stored by DI(). If other than that is specified, the operation is not guaranteed.

Example Usage

```
void foo()
{
    UINT intsts;

    DI(intsts);

    if ( isDI(intsts) ) {
        /* Interrupts have been already disabled when this function was called. */
    } else {
        /* Interrupts have been enabled when this function was called. */
    }

    EI(intsts);
}
```

3.10 System Status Management Function System Calls

The system calls of system status management function are explained.

■ System Status Management Function System Calls

The system status management function is comprised of six system calls as follows:

- tk_rot_rdq (Rotate Ready Queue)
- tk_get_tid (Get Task Identifier)
- tk_dis_dsp (Disable Dispatch)
- tk_ena_dsp (Enable Dispatch)
- tk_ref_sys (Refer System Status)
- tk_ref_ver (Refer Version Information)

3.10.1 tk_rot_rdq (Rotate Ready Queue)

Rotates the task priority order.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_rot_rdq ( PRI tskpri ) ;
```

■ Parameter

● Input

tskpri Priority (Task Priority)

Name	Value	Meaning
TPRI_RUN	0	Priority of the task running currently

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_PAR	-17	Parameter error (tskpri is negative or greater than the maximum priority of the system)

■ Dispatch Trigger

If the same priority with the task which called this system call is specified and other task has the priority, the task with the next priority is dispatched.

■ Description

This system call rotates the priority order of the task with the priority specified by tskpri. In other words, among the tasks which has the object priority and can be executed, the task which has the highest priority order is made to be the task with the lowest priority order among the tasks which has the same priority.

It returns with the error E_PAR if tskpri is negative or greater than the maximum priority of the system (maximum priority specified by the configurator).

If TPRI_RUN (=0) is specified to tskpri, the same operation with the case where the priority of the task in the running status is specified to tskpri. This system call can be invoked by the specification of TPRI_RUN to tskpri from the task-independent portion such as a cyclic handler.

■ Additional Notes

If there is no or only one task which has the object priority and can be executed, this system call finishes normally performing nothing.

Since the task with the highest priority order is not always running among the tasks with the same priority in the dispatch disabled status, the execution order of the task is not always to be the lowest among the tasks with the same priority by this way.

The execution example for `tk_rot_rdq` is shown in Figure 3.10-1 and Figure 3.10-2. With the status of Figure 3.10-1, if this system call is invoked with `tskpri = 2` as a parameter, a new priority will be Figure 3.10-2, and Task C will be executed next.

Figure 3.10-1 Priority Order before Execution of `tk_rot_rdq`

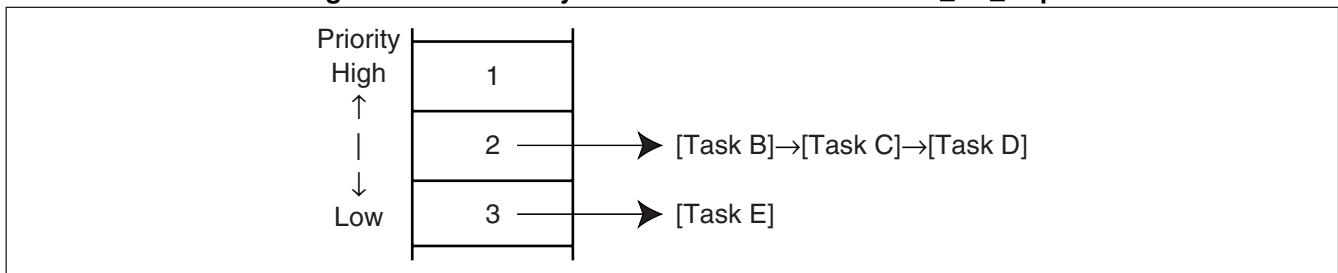
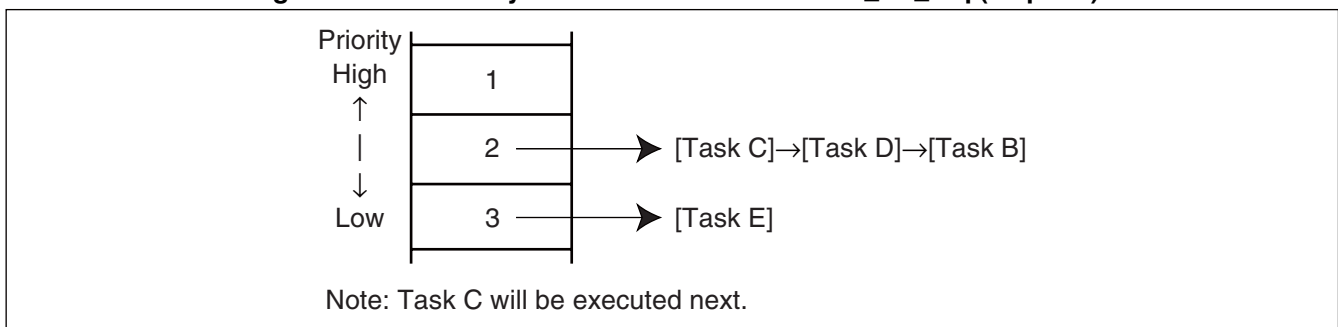


Figure 3.10-2 Priority Order after Execution of `tk_rot_rdq` (`tskpri=2`)



3.10.2 tk_get_tid (Get Task Identifier)

Refers to the task ID of task running.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ID tskid = tk_get_tid ( void ) ;
```

■ **Parameter**

● Input

None

● Output

tskid Running task ID (Task ID)

■ **Error Code**

None

■ **Dispatch Trigger**

This system call does not perform dispatch.

■ **Description**

This system call sets the ID number of the currently running task to tskid and returns.

If it is invoked from a task portion, ID of invoking task is returned. When there is no task running currently, 0 is returned.

■ **Additional Notes**

The task ID returned by this system call is same as runtskid returned by tk_ref_sys.

3.10.3 tk_dis_dsp (Disable Dispatch)

Disables the dispatch.

Task portion	<input type="radio"/>	Task-independent portion	×	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	---	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_dis_dsp ( void ) ;
```

■ Parameter

● Input

None

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_CTX	-25	Context error (Invocation from a task-independent portion)

■ Dispatch Trigger

This system call does not perform dispatch.

■ Description

This system call disables the dispatch of task. The dispatch is disabled after invocation of the system call until execution of `tk_ena_dsp`. Invoking task is not switched from running status to executable status. Also, it cannot be switched to the wait state. However, since external interrupts are not disabled, even in the dispatch disabled state, an interrupt handler starts.

In the dispatch disabled state, the running task could be preempted by an interrupt handler (takeover of CPU execution right), however, it cannot be preempted by other tasks.

Specifically the following operations are performed during the dispatch disabled state.

- Even if the task with the higher priority than the task with `tk_dis_dsp` executed becomes executable by the system call invoked from an interrupt handler or the task with `tk_dis_dsp` executed, the task is not dispatched. The dispatch to the task with high priority is delayed until the dispatch disabled state is terminated.
- If the task with `tk_dis_dsp` executed invokes the system call (such as `tk_slp_tsk` and `tk_wai_sem`) which could switch invoking task to the wait state, the system call returns with the error `E_CTX`.
- If the system state is referred by `tk_ref_sys`, `TSS_DDSP` is returned as `sysstat`.

If the task which has been already in the dispatch disabled state invokes `tk_dis_dsp`, only the dispatch disabled state is continued as it is, and the system call is terminated normally. However, note that even if `tk_dis_dsp` has been invoked several times, then only one invocation of `tk_ena_dsp` releases the dispatch disabled state.

■ Additional Notes

In the dispatch disabled state, the running task cannot be switched to the resting state or unregistered state. In the interrupt or dispatch disabled state, if the running task invokes `tk_ext_tsk` or `tk_exd_tsk`, the error `E_CTX` is detected. However, since `tk_ext_tsk` and `tk_exd_tsk` are the system calls which do not return to the source context, an error cannot be notified as the return value of those system calls.

3.10.4 tk_ena_dsp (Enable Dispatch)

Enables the dispatch.

Task portion	<input type="radio"/>	Task-independent portion	×	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	---	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ena_dsp ( void ) ;
```

■ Parameter

● Input

None

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_CTX	-25	Context error (Invocation from a task-independent portion)

■ Dispatch Trigger

If there is the dispatch which is delayed, the dispatch is performed.

■ Description

This system call enables the dispatch of task. In other words, the dispatch disabled state set by tk_dis_dsp is released.

If the task which is not in the dispatch disabled state invokes this system call, only the dispatch enabled state is continued as it is, and the system call is terminated normally. When the system call is invoked from a task-independent portion, it returns with the error E_CTX.

3.10.5 tk_ref_sys (Refer System Status)

Refers to the system status

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_ref_sys ( T_RSYS *pk_rsys ) ;
```

```
typedef struct t_rsys {  
    INT sysstat;  
    ID  runtskid;  
    ID  schedtskid;  
} T_RSYS;
```

■ **Parameter**

● **Input**

pk_rsys Packet address to return the system status
 (Packet of Refer System)

● **Output**

ercd Error code (Error Code)

● **Data returned in packet**

sysstat System status (System State)
 sysstat:= (TSS_TSK | [TSS_DDSP] | [TSS_DINT])
 || (TSS_QTSK | [TSS_DDSP] | [TSS_DINT])
 || (TSS_INDP)

State	Value	Meaning
TSS_TSK	0	Task portion running.
TSS_DDSP	1	Dispatch disabled
TSS_DINT	2	Interrupt disabled
TSS_INDP	4	Task-independent portion running
TSS_QTSK	8	Quasi-task portion running

runtskid Currently running task ID (Running Task ID)

schedtskid Scheduled to be running task ID (Scheduled Task ID)

■ Error Code

E_OK	0	Normal completion
------	---	-------------------

■ Dispatch Trigger

This system call does not perform the dispatch.

■ Description

This system call refers to the running state and returns the information such as the dispatch disabled and task-independent portion running as the return value.

The currently running task ID and scheduled to be running task ID are returned to runtskid and schedtskid respectively. Normally runtskid \neq schedtskid, however, if there is the task with the higher priority in the dispatche disabled or task-independent portion running state, runtskid is not equal to schedtskid. In addition, if there is no applicable task, 0 is returned.

No error check is performed even if pk_rsys is invalid. The operation in this case is not guaranteed.

3.10.6 tk_ref_ver (Refer Version Information)

Refers to the version information of the kernel

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_ref_ver ( T_RVER *pk_rver ) ;
```

```
typedef struct t_rver {
    UH  maker;
    UH  prid;
    UH  spver;
    UH  prver;
    UH  prno[4];
} T_RVER;
```

■ **Parameter**

● **Input**

pk_rver	Initial address of the packet to return the version information (Packet of Version Information)
---------	--

● **Output**

ercd	Error code (Error Code)
------	-------------------------

● **Data returned in packet**

maker	Maker code of the kernel (Maker) =0x0009 (Maker code = FUJITSU)
prid	Identification number of the kernel (Product ID) =0x6911 (μT-REALOS/FR)
spver	Specification version number (Specification Version) =0x6100 (μT-Kernel ver.1.00)
prver	Version number of the kernel (Product Version) =0x0100 (V01L00)
prno[4]	Management information of the kernel product (Product Number) prno[0] = 0x0000 (R00) prno[1] to [3]= 0x0000 (unused)

■ Error Code

<code>E_OK</code>	0	Normal completion
-------------------	---	-------------------

■ Dispatch Trigger

This system call does not perform the dispatch.

■ Description

This system call refers to the version information of the kernel used and returns it to the packet specified by `pk_rver`.

No error check is performed even if `pk_rver` is invalid. The operation in this case is not guaranteed.

3.11 Sub System Function System Calls

The system calls of the sub system function are explained.

■ Sub System Function System Calls

The sub system function is comprised of two system calls as follows:

- tk_def_ssy (Define Subsystem)
- tk_ref_ssy (Refer Subsystem Status)

3.11.1 tk_def_ssy (Define Subsystem)

Defines the sub system or deletes the definition.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_def_ssy ( ID ssid, T_DSSY *pk_dssy ) ;
```

```
typedef struct t_dssy {
    ATR ssyatr;
    PRI ssypri;
    FP  svchdr;
    FP  breakfn;
    FP  startupfn;
    FP  cleanupfn;
    FP  eventfn;
    INT resblks;
} T_DSSY;
```

■ Parameter

● Input

ssid	Sub system ID (Subsystem ID)
pk_dssy	Packet initial address of the sub system definition information (Packet of Define Subsystem)

● Data to set in packet

ssyatr	Sub system attribute (Subsystem Attribute)
ssypri	Sub system priority (Subsystem Priority)
svchdr	Extended SVC handler address (SVC Handler)
breakfn	Break function address (Break Function)
startupfn	Start-up function address (Start-up Function)
cleanupfn	Clean-up function address (Clean-up Function)
eventfn	Event processing function address (Event Function)
resblks	Resource management block size (number of bytes) (Resource Block Size)

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (ssid is less than 10 or greater than the maximum sub systems number)
E_OBJ	-41	ssid already defined (when pk_dssy ≠ NULL)
E_NOEXS	-42	ssid undefined (when pk_dssy = NULL)
E_PAR	-17	ssypri is illegal

■ Dispatch Trigger

This system call does not perform the dispatch.

■ Description

This system call defines the sub system of ssid or deletes the definition.

Allocate one sub system ID to one sub system to avoid overlaps with other sub systems OS does not have an automatic allocation function.

1 to 9 of the sub system IDs are reserved for μ T-Kernel. 10 to the maximum sub systems number (maximum sub systems number specified by the configurator) can be used.

In μ T-Kernel, only extended SVC handler call function is supported as sub system function. For this reason, the data, except svchr, to be set in t_dssy packet is provided for compatibility with T-Kernel and the value will be ignored.

svchr specifies the start address of a function to be called as an extended SVC handler. An extended SVC handler is an API of the sub system and can be invoked by the same way as the system calls.

The extended SVC handler can be described in C language only by the following format:

```
INT shdr( VP pk_para, FN fncd )
{
    /*
     * Processing after branch by fncd
     */
    return retcode; /* Extended SVC Handler Termination*/
}
```

fncd is a function code. The sub system ID is stored in the lower 8 bits of function code. The sub system side can use the remaining upper bits freely. Usually they are used as the function code in sub system. However, since the function code must be a positive value, the highest bit is always 0.

pk_para is the parameter passed from the invocation side and is made in a form of packets. The form of packets can be determined freely by the sub system side. Generally it is same as the form of stack when an argument is passed to a function in C language. In most cases it is same as the form of structure of C language.

The return value from the extended SVC handler is returned to the invocation source as the return value of a function as it is. In principle a negative value is the error value and 0 or positive value is the return value in a normal operation. In addition, since if the invocation of extended SVC fails with some reason, the extended SVC handler is not invoked and an error code (negative value) of OS is returned to the invocation source, do not to be confused with that.

An extended SVC handler is executed as a quasi-task portion if it is invoked from a task portion. It can be invoked from a task-independent portion, however, if so, the extended SVC handler is also executed as a task-independent portion.

3.11.2 tk_ref_ssy (Refer Subsystem Status)

Refers to the sub system definition information.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ER ercd = tk_ref_ssy ( ID ssid, T_RSSY *pk_rssy ) ;
```

```
typedef struct t_rssy {
    PRI ssypri;
    INT resblksz;
} T_RSSY;
```

■ **Parameter**

● **Input**

ssid	Sub system ID (Subsystem ID)
pk_rssy	Sub system definition information (Packet of Refer Subsystem)

● **Output**

ercd	Error code (Error Code)
------	-------------------------

● **Data returned in packet**

ssypri	Sub system priority (Subsystem Priority)
resblksz	Resource management block size (number of bytes) (Resource Block Size)

■ **Error Code**

E_OK	0	Normal completion
E_ID	-18	Invalid ID number (ssid is less than 10 or greater than 255)
E_NOEXS	-42	ssid undefined (The sub system of ssid is undefined.)

■ **Dispatch Trigger**

This system call does not perform the dispatch.

■ **Description**

This system call refers to the various information of the object sub system indicated by ssid.

ssypri and resblksz are members which exist for the compatibility with T-Kernel, and they are not used with µT-REALOS. Therefore these values are undefined values.

It returns with the error E_NOEXS, if the sub system of ssid is undefined.

No error check is performed even if pk_rssy is invalid. The operation in this case is not guaranteed.

3.12 Device Management Function System Calls

The system calls of the device management function is explained.

■ Device Management Function System Calls

The device management function is comprised of 15 system calls as follows:

- tk_def_dev (Define Device)
- tk_ref_idv (Refer Initial Device Information)
- tk_opn_dev (Open Device)
- tk_cls_dev (Close Device)
- tk_rea_dev (Read Device)
- tk_srea_dev (Synchronous Read Device)
- tk_wri_dev (Write Device)
- tk_swri_dev (Synchronous Write Device)
- tk_wai_dev (Wait Device)
- tk_sus_dev (Suspend Device)
- tk_get_dev (Get Device)
- tk_ref_dev (Refer Device)
- tk_oref_dev (Refer Device)
- tk_lst_dev (List Device)
- tk_evt_dev (Event Device)

For more details about this section, also see "APPENDIX C Device Driver Interface".

3.12.1 tk_def_dev (Define Device)

Registers a device or deletes the registration.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID devid = tk_def_dev ( UB *devnm, T_DDEV *ddev, T_IDEV *idev ) ;
```

```
typedef struct t_ddev {
```

```
    VP  exinf;
```

```
    ATR drvatr;
```

```
    ATR devatr;
```

```
    INT nsub;
```

```
    INT blksh;
```

```
    FP  openfn;
```

```
    FP  closefn;
```

```
    FP  execfn;
```

```
    FP  waitfn;
```

```
    FP  abortfn;
```

```
    FP  eventfn;
```

```
} T_DDEV;
```

```
typedef struct t_idev {
```

```
    ID  evtmbfid;
```

```
} T_IDEV;
```

■ Parameter

● Input

devnm Physical device name (Device Name)
 ddev Device registration information (Packet of Define Device)

● Data to set in packet

exinf Extended information
 drvatr Driver attribute (Driver Attribute)
 drvatr := [TDA_OPENREQ]

Attribute	Value	Meaning
TDA_OPENREQ	0x0001	Every time open/close

devatr Device attribute (Device Attribute)
 nsub Number of sub units (Sub Unit No.)
 blksize Blocks number of device (Block Size)
 openfn Open function entry address (Open Function)
 closefn Close function entry address (Close Function)
 execfn Processing start function entry address (Execute Function)
 waitfn Completion waiting function entry address (Wait Function)
 abortfn Abortion processing function entry address (Abort Function)
 eventfn Event function entry address (Event Function)

● Output

devid Device ID (Device ID)
 Or, error code (Error Code)
 idev Device initial information (Initial Device Information)

● Data to set in packet

evtmbfid Message buffer for event notification ID

■ Error Code

E_LIMIT -34 The number of devices registered exceeds the upper limit of the system
 E_PAR -17 Length of devnm is 0 or greater than 8 and nsub is negative or 256 or more
 E_NOEXS -42 Device of devnm does not exist. (when ddev = NULL)

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ Description

This system call registers a device with a device name of `devnm`. If the device of `devnm` has been already registered, it is updated with a new registration information. In the case of update, the device ID is not changed. When `ddev=NULL`, the device registration of `devnm` is deleted. When the number of devices that have been registered already reaches to the upper limit of the system (maximum number of devices for registration specified in the configurator), `E_LIMIT` error is returned.

The following values are specified to `T_DDEV`:

- `exinf`
When the device processing functions (`openfn` to `eventfn`) are invoked from the device management function API, this value is passed to the device driver as an argument of the device processing function. It can be used freely at the device driver side.
- `drvatr`
The driver attribute is specified.
- `devatr`
The device attribute is specified. For the detail of device attribute, see "Appendix C Device Driver Interface".
- `nsub`
The number of sub units are specified. 0 is specified for the device with no sub unit.
- `blksz`
The device block size is specified in bytes number. Usually this size is the minimum access size of the device.
- `openfn` to `eventfn`
The entry address of the device processing function is specified. For the detail of device processing function, see "Appendix C Device Driver Interface".

The device initial information is returned to `idev`, and the system default message buffer ID for event notification is returned to `evtmbfid`. When `idev=NULL`, the device initial information is not stored. If there is no system default event notification message buffer, 0 is specified to `evtmbfid`.

The message buffer for event notification is the message buffer ID number to be used for notification of various events occurred in the device (such as inserting of media and abnormal battery) to the upper program. For the event notification of device, see "Appendix C Device Driver Interface".

No error check is performed even if `ddev` and `idev` are invalid. The operation in this case is not guaranteed.

3.12.2 tk_ref_idv (Refer Initial Device Information)

Obtains the device initial information.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_ref_idv ( T_IDEV *idev ) ;
```

```
typedef struct t_idev {
    ID evtmbfid;
} T_IDEV;
```

■ Parameter

● Input

idev	Address for the area to store the device initial information (Initial Device Information)
------	--

● Output

ercd	Error code (Error Code)
------	-------------------------

● Data returned in packet

evtmbfid	ID of message buffer for event notification (Event Messagebuffer ID)
----------	---

■ Error Code

E_OK	0	Normal completion
------	---	-------------------

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ Description

This system call obtains the device initial information. The information has the same contents as the one obtained by tk_def_dev system call.

No error check is performed even if idev is invalid. The operation in this case is not guaranteed.

3.12.3 tk_opn_dev (Open Device)

Opens the device.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID dd = tk_opn_dev ( UB *devnm, UINT omode ) ;
```

■ Parameter

● Input

devnm Device name (Device Name)

omode Open mode (Open Mode)

omode := (TD_READ || TD_WRITE || TD_UPDATE) | [TD_EXCL || TD_WEXCL || TD_REXCL]

Mode	Value	Meaning
TD_READ	0x0001	Read only
TD_WRITE	0x0002	Write only
TD_UPDATE	0x0003	Read and write
TD_EXCL	0x0100	Exclusion
TD_WEXCL	0x0200	Exclusive write
TD_REXCL	0x0400	Exclusive read

● Output

dd Device descriptor (Device Descriptor)

Or, error code (Error Code)

■ Error Code

E_PAR -17 TD_READ, TD_WRITE, and TD_UPDATE are all not set or in undefined mode.

E_BUSY -65 Device is being used (being open exclusively)

E_NOEXS -42 Device does not exist.

E_LIMIT -34 Greater than the maximum number of devices which can be opened

Other value Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time. Also the dispatch depends on the device driver.

■ Description

This system call opens the device specified by `devnm` in the mode specified by `omode`. Also it prepares the access to the device. The device descriptor is returned as the return value.

The following values are specified to `omode`.

- `TD_READ`, `TD_WRITE`, `TD_UPDATE`

The access mode is specified. For `TD_READ`, `tk_wri_dev` can not be used. Also for `TD_WRITE`, `tk_rea_dev` can not be used.

- `TD_EXCL`, `TD_WEXCL`, `TD_REXCL`

The exclusion mode is specified. `TD_EXCL` prohibits all simultaneous opens. `TD_WEXCL` prohibits the simultaneous open by the write mode (`TD_WRITE` or `TD_UPDATE`). `TD_REXCL` prohibits the simultaneous open by the read mode (`TD_READ` or `TD_UPDATE`).

In addition, if the physical device is opened, it is handled in the same way as that all of the logical devices, which belong to the physical device, are opened in the same mode, and processing of exclusive open is performed.

No error check is performed even if `devnm` is invalid. The operation in this case is not guaranteed.

3.12.4 tk_cls_dev (Close Device)

Closes the device.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_cls_dev ( ID dd, UINT option ) ;
```

■ Parameter

● Input

dd Device descriptor (Device Descriptor)
option Close option (Option)

Option	Value	Meaning
TD_EJECT	0x0001	Media ejection on closing

● Output

ercd Error code (Error Code)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	dd is invalid (dd is 0 or less, or greater than the maximum devices open number) or the device is not open.
E_OACV	-27	Request from a task that is not the task that opened the device
Other value		Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time. Also the dispatch depends on the device driver.

■ Description

This system call closes the device descriptor of dd. If it is requested during processing, the processing is stopped, and the device is closed.

When TD_EJECT is specified to option, if the same device is not opened by others task, media is ejected. However, the specification is ignored for the device which cannot eject media.

3.12.5 tk_rea_dev (Read Device)

Starts the device reading.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID reqid = tk_rea_dev ( ID dd, W start, VP buf, W size, TMO tmout ) ;
```

■ Parameter

● Input

dd	Device descriptor (Device Descriptor)
start	Read start position (0 or more : Specific data, less than 0 : Attribute data) (Start)
buf	Buffer to store the read data (Buffer)
size	Size to be read (Size)
tmout	Request acceptance waiting timeout time (ms) (Timeout)

The following macros can be specified in addition to the values from 0 to 0x7fffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

reqid	Request ID (Request ID) Or, error code (Error Code)
-------	--

■ Error Code

E_ID	-18	dd is invalid (dd is 0 or less, or greater than the maximum devices open number) or the device is not open.
E_OACV	-27	Open mode is invalid (open by TD_WRITE)
E_LIMIT	-34	Maximum number of requests exceeded
Other value		Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. The dispatch may be executed during this time. Also the dispatch depends on the device driver.

■ Description

This system call starts reading of the specific data or attribute data from the device. It only starts reading, and returns to the invocation source without waiting the reading completion. Retain buf until reading is completed. Reading completion is waited by tk_wai_dev. The time for the processing to start reading differs according to the device drivers. The system call does not always return immediately.

For the specific data, the units for start and size are determined for each device. For the attribute data, start is the attribute data number and size is the number of bytes. The attribute data of the data number of start is read. Usually size is more than the size of the attribute data which is read. Multiple attributes data cannot be read at once. asize is a return value of tk_wai_dev and is set to a readable size by calling tk_wai_dev after this system call is invoked with size=0.

If reading or writing is being performed, whether a new request is accepted or not depends on the device driver. When in the state where a new request cannot be accepted, the request acceptance is for waiting. The timeout time for request acceptance waiting is specified to tmout. The timeout occurs by the request is accepted. The timeout does not occur after the request was accepted.

3.12.6 tk_srea_dev (Synchronous Read Device)

Reads the device synchronously.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_srea_dev ( ID dd, W start, VP buf, W size, W *asize ) ;
```

■ Parameter

● Input

dd	Device descriptor (Device Descriptor)
start	Read start position (0 or more : Specific data, less than 0 : Attribute data) (Start)
buf	Buffer to store the read data (Buffer)
size	Size to be read (Size)

● Output

ercd	Error code (Error Code)
asize	Size which was read (Size)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	The dd parameter is invalid (less than or equal to zero or greater than the maximum number of devices to open) or the device is not open.
E_OACV	-27	Open mode is invalid (open by TD_WRITE).
E_LIMIT	-34	Maximum number of requests exceeded
Other value		Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. The dispatch may be executed during this time. The dispatch depends on the device driver.

■ Description

This system call reads the device data synchronously. If this system call returns normally, the device data is stored for `asize` with the address specified by `buf` at the head. This system call has the same processing as the following.

```
ER tk_srea_dev( ID dd, W start, VP buf, W size, W *asize )
{
    ER er, err, ioer;
    err = tk_rea_dev (dd, start, buf, size, TMO_FEVR);
    if ( err > 0 ) {
        err = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
        if ( err > 0 ) err = ioer;
    }
    return err;
}
```

No error check is performed even if `asize` is invalid. The operation in this case is not guaranteed.

3.12.7 tk_wri_dev (Write Device)

Starts the device writing.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID reqid = tk_wri_dev ( ID dd, W start, VP buf, W size, TMO tmout ) ;
```

■ Parameter

● Input

dd	Device descriptor (Device Descriptor)
start	Write start position (0 or more : Specific data, less than 0 : Attribute data) (Start)
buf	Buffer to store the data to be written (Buffer)
size	Size to be written (Size)
tmout	Request acceptance waiting timeout time (ms) (Timeout)

The following macros can be specified in addition to the values from 0 to 0x7fffffff.

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

reqid	Request ID (Request ID) Or, error code (Error Code)
-------	--

■ Error Code

E_ID	-18	The dd parameter is invalid (less than or equal to zero or greater than the maximum number of devices to open) or the device is not open.
E_OACV	-27	Open mode is invalid (open by TD_READ)
E_RDONLY	-67	Device which cannot be written
E_LIMIT	-34	Maximum number of requests exceeded
Other value		Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. The dispatch may be executed during this time. Also depends on the device driver.

■ Description

This system call starts writing of the specific data or attribute data to the device. It only starts writing, and returns to the invocation source without waiting the writing completion. Retain buf until writing is completed. Writing completion is waited by tk_wai_dev. The time for the processing to start writing differs according to the device drivers. The system call does not always return immediately.

For the specific data, the units for start and size are determined for each device. For the attribute data, start is the attribute data number and size is the number of bytes. The attribute data of the data number of start is written. Usually size is same as the size of the attribute data which is written. Multiple attributes data cannot be written at once. asize is a return value of tk_wai_dev and is set to a writable size by calling tk_wai_dev after this system call is invoked with size=0.

If reading or writing is being performed, whether a new request is accepted or not depends on the device driver. When in the state where a new request cannot be accepted, the request acceptance is for waiting. The timeout time for request acceptance waiting is specified to tmout. Also TMO_POL(=0) or TMO_FEVR(=-1) can be specified to tmout. The timeout occurs by the request is accepted. The timeout does not occur after the request was accepted.

3.12.8 tk_swri_dev (Synchronous Write Device)

Writes the device synchronously.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ER ercd = tk_swri_dev ( ID dd, W start, VP buf, W size, W *asize ) ;
```

■ Parameter

● Input

dd	Device descriptor (Device Descriptor)
start	Write start position (0 or more : Specific data, less than 0 : Attribute data) (Start)
buf	Buffer stored the data to be written (Buffer)
size	Size to be written (Size)

● Output

ercd	Error code (Error Code)
asize	Written size (Size)

■ Error Code

E_OK	0	Normal completion
E_ID	-18	The dd parameter is invalid (less than or equal to zero or greater than the maximum number of devices to open) or the device is not open.
E_OACV	-27	Open mode is invalid (open by TD_READ)
E_RONLY	-67	Device which cannot be written
E_LIMIT	-34	Maximum number of requests exceeded
Other value		Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. The dispatch may be executed during this time. Also the dispatch depends on the device driver.

■ Description

This system call writes the device synchronously. It has the same processing as the following.

```
ER tk_swri_dev( ID dd, W start, VP buf, W size, W *asize )
{
    ER er, err, ioer;
    err = tk_wri_dev(dd, start, buf, size, TMO_FEVR);
    if ( err > 0 ) {
        err = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
        if ( err > 0 ) err = ioer;
    }
    return err;
}
```

No error check is performed even if asize is invalid. The operation in this case is not guaranteed.

3.12.9 tk_wai_dev (Wait Device)

Waits the device request completion.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID creqid = tk_wai_dev ( ID dd, ID reqid, W *asize, ER *ioer, TMO tmout ) ;
```

■ Parameter

● Input

dd	Device descriptor (Device Descriptor)
reqid	Request ID (Request ID)
tmout	Request acceptance waiting timeout time (ms) (Timeout)
The following macros can be specified in addition to the values from 0 to 0x7ffffff.	

Name	Value	Meaning
TMO_FEVR	-1	Wait indefinitely
TMO_POL	0	Polling

● Output

creqid	Completed request ID (Request ID) or error code (Error Code)
asize	Read size or written size (Size)
ioer	Input/Output error (IO Error)

■ Error Code

E_ID	-18	dd is invalid (less than or equal to zero or greater than the maximum number of devices to open) or the device is not open. reqid is less than 1 or greater than the system upper limit, or the request is not for dd.
E_OACV	-27	Request from a task that is not the task that opened the device
E_OBJ	-41	The request of reqid is waiting for completion by other task.
E_NOEXS	-42	No request in process (when reqid=0 only)
Other value		Error returned by device driver

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. The dispatch may be executed during this time. Also the dispatch depends on the device driver.

■ Description

This system call waits the completion of request of reqid for dd.

When reqid=0, the completion of any of requests for dd is waited. In this case, only the request in process on tk_wai_dev invocation is to be waited for the completion. The processing required after the invocation of this system call is not to be waited for the completion.

When multiple requests are processed simultaneously, the completion order of the requests is not always the order of requests, but depends on the device driver. However, the processing with the order where the results do not contradict ones of processing with the requests order is guaranteed. For example, for reading from a disk, the following processing order change is assumed.

Request order block number 1 4 3 2 5

Processing order block number 1 2 3 4 5

The processing with shuffled order as above decreases seeks or rotational delays making the disk access more efficient.

The timeout time for completion waiting is specified to tmout. Also TMO_POL(=0) or TMO_FEVR(=-1) can be specified to tmout. Since when the timeout (E_TMOUT) occurs, the processing required is in process, the completion have to be waited by this system call again. If reqid>0 and tmout=TMO_FEVR, the timeout does not occur, and the processing always completes.

The error of result of requested processing (such as input/output error) is stored not to the return value, but to ioer. The system call returns with the error as the return value if the request completion waiting can not be performed correctly. When the error is returned to the return value, the description of ioer is undefined value.

Return Value	ioer	Meaning
E_OK	E_OK	The processing for waiting completed normally.
E_OK	Error	The processing for waiting completed with a error. The processing error trigger is stored to ioer as the error code.
Error	Undefined	Completion waiting failed. The failure trigger is stored to the return value as the error code.

Since when the error is returned to the return value, the processing is in process, the completion have to be waited by this system call again.

The completion waiting can not be performed from multiple tasks to the same request ID simultaneously. If any task is waiting by reqid=0, other task cannot wait the completion for the same dd so E_OBJ error is returned. In the same way, if some task waits by reqid>0, other task cannot wait the completion by reqid=0. E_OBJ error is returned in this case, too.

No error check is performed even if assize and ioer are invalid. The operation in this case is not guaranteed.

3.12.10 tk_sus_dev (Suspend Device)

Suspends the device.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
INT cnt = tk_sus_dev ( UINT mode ) ;
```

■ Parameter

● Input

mode Mode (Mode)

mode := ((TD_SUSPEND | [TD_FORCE]) || TD_DISSUS || TD_ENASUS || TD_CHECK)

Mode	Value	Meaning
TD_SUSPEND	0x0001	Suspend
TD_DISSUS	0x0002	Suspend disabled.
TD_ENASUS	0x0003	Suspend enabled.
TD_CHECK	0x0004	Suspend disabled request count obtained.
TD_FORCE	0x8000	Forced suspend specified.

● Output

cnt Suspend disabled request count number (Count)

Or, error code (Error Code)

■ Error Code

E_PAR	-17	The mode is undefined.
E_BUSY	-65	Suspend being disabled.
E_QOVR	-43	Suspend disabled request count over

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. The dispatch may be executed during this time. Also the dispatch depends on the device diver.

■ Description

This system call performs the processing according to mode and returns the suspend disabled request count number to the return value after the processing is performed.

- **TD_SUSPEND** Suspend
When in suspend enabled state, it suspends the device. If in suspend disabled state, E_BUSY is returned.
- **TD_SUSPEND|TD_FORCE** Forced suspend
Even if in suspend disabled state, it suspends the device.
- **TD_DISSUS** Suspend disabled
Suspend is disabled.
- **TD_ENASUS** Suspend enabled
Suspend is enabled. If suspend is enabled for more than suspend disabled count, nothing is performed. nothing is performed.
- **TD_CHECK** Suspend disabled count obtained
Obtainment for times of suspend disabled request is performed.

■ Additional Notes

Suspend is performed by the following procedure.

1. Suspend processing for other than each disk device.
2. Suspend processing for each disk device.
3. Shift to suspend state

Resume (return from suspend) is performed by the following procedure.

1. Return from suspend state.
2. Resume processing for each disk device.
3. Resume processing for other than each disk device.

Suspend disabled counts the number of requests. If the suspend permission is not required same times, suspend is not enabled. On the system activation, suspend is enabled (suspend disabled request count = 0). Suspend disabled request count is one for the whole system. Suspend disabled request count upper limit is 2147483647 (0x7fffffff). If the count is greater than this limit, the system call returns with the error E_QOVR.

3.12.11 tk_get_dev (Get Device)

Obtains the device name.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID phyid = tk_get_dev ( ID devid, UB *devnm ) ;
```

■ Parameter

● Input

devid Device ID (Device ID)

● Output

phyid Device ID of a physical device (Physical Device ID) or an error code (Error Code)

devnm Device name (Device Name)

■ Error Code

E_NOEXS -42 The device of devid does not exist.

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ Description

This system call obtains the device name indicated by devid and stores to devnm.

devid is a device ID of a physical device or logical device. If devid is a physical device, a physical device name is stored to devnm, and if it is a logical device, a logical device name is stored there. devnm needs L_DEVNM (8) + area of 1 byte or more.

The device ID of physical device where the device of devid belongs to is returned to the return value.

No error check is performed even if devnm is invalid. The operation in this case is not guaranteed.

3.12.12 tk_ref_dev (Refer Device)

Obtains the device information of the device.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
ID devid = tk_ref_dev ( UB *devnm, T_RDEV *rdev ) ;
```

```
typedef struct t_rdev {
    ATR devatr;
    INT blksize;
    INT nsub;
    INT subno;
} T_RDEV;
```

■ **Parameter**

● **Input**

devnm	Device name (Device Name)
rdev	Address for the area to store the device information. (Packet of Refer Device)

● **Output**

devid	Device ID (Device ID) Or, error code (Error Code)
-------	--

● **Data returned in packet**

devatr	Device attribute (Device Attribute)
blksize	Block size of specific data (-1: unknown) (Block Size)
nsub	Number of sub units (Number of Sub-unit)
subno	0: physical device, 1 to nsub: sub unit number+1 (Sub-unit Number)

■ **Error Code**

E_NOEXS	-42	The device of devnm does not exist.
---------	-----	-------------------------------------

■ **Dispatch Trigger**

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ Description

This system call obtains the device information of the device indicated by devnm and stores to rdev. When rdev=NULL, the device information is not stored.

Nsub is the number of sub units of the physical device where the device indicated by devnm belongs to.

The device ID of device of devnm is returned to the return value.

No error check is performed even if devnm and rdev are invalid. The operation in this case is not guaranteed.

For the device attribute, see "Appendix C Device Driver Interface".

3.12.13 tk_oref_dev (Refer Device)

Obtains the device information of the device.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
ID devid = tk_oref_dev ( ID dd, T_RDEV *rdev ) ;
```

```
typedef struct t_rdev {
    ATR devatr;
    INT blksize;
    INT nsub;
    INT subno;
} T_RDEV;
```

■ Parameter

● Input

dd	Device descriptor (Device Descriptor)
rdev	Address for the area to store the device information. (Packet of Refer Device)

● Output

devid	Device ID (Device ID) Or, error code (Error Code)
-------	--

● Data returned in packet

devatr	Device attribute (Device Attribute)
blksize	Block size of specific data (-1: unknown) (Block Size)
nsub	Number of sub units (Number of Sub-unit)
subno	0:physical device, 1 to nsub:sub unit number+1 (Sub-unit Number)

■ Error Code

E_OACV	- 27	Request from a task that is not the task that opened the device
--------	------	---

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ Description

This system call obtains the device information of the device indicated by `dd` and stores to `rdev`. When `rdev=NULL`, the device information is not stored.

`nsub` is the number of sub units of the physical device where the device indicated by `dd` belongs to.

The device ID of the device indicated by `dd` is returned to the return value.

No error check is performed even if `rdev` is invalid. The operation in this case is not guaranteed.

For the device attribute, see "Appendix C Device Driver Interface".

3.12.14 tk_lst_dev (List Device)

Obtains the list of the devices already registered.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ C Language Interface

```
INT cnt = tk_lst_dev ( T_LDEV *ldev, INT start, INT ndev ) ;

typedef struct t_ldev {
    ATR devatr;
    INT blksz;
    INT nsub;
    UB  devnm[L_DEVNM] ;
} T_LDEV;
```

■ Parameter

● Input

ldev	Storing area for the registered device information (array) (Packet of List Device)
start	Starting number (Start)
ndev	Obtained number (Number of Device)

● Output

cnt	Remaining count number (Count) Or, error code (Error Code)
-----	---

● Data returned in packet

devatr	Device attribute (Device Attribute)
blksz	Block size of specific data (-1: unknown) (Block Size)
nsub	Number of sub units (Number of Sub-unit)
devnm[L_DEVNM]	Physical device name (Device Name)

■ Error Code

E_NOEXS	-42	No information to be obtained (start is the number of devices already registered or more)
E_PAR	-17	start or ndev is negative.

■ Dispatch Trigger

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ Description

This system call obtains the information of the device already registered.

The registered device is managed by each physical device. Therefore the registered device information is obtained by each physical device.

If the number of registered devices is N , the sequence number of 0 to $N-1$ is allocated to the registered device. According to this sequence number, $ndev$ pieces of the registration information from start-th are obtained and stored to $ldev$. $ldev$ needs enough space to store $ndev$ pieces of information.

The remaining count number ($N - start$) is returned to the return value. If the remaining count number is less than $ndev$, all of the remaining are stored. The return value which is equal to or less than $ndev$ indicates that all registration information are obtained.

This sequence number changes when a device is registered/unregistered. Therefore if the information is obtained by dividing multiple times, correct information may not be obtained

For example, when ten devices are registered, if three pieces of registration information are to be obtained at a time, the change of start $0 \rightarrow 3 \rightarrow 6 \rightarrow 9$ and four times of invocation of this system call allow the obtainment of all device registration information. If the device is registered/unregistered during the invocation of this system call from first to fourth time, correct information might not be obtained. In this case, specify 0 to start again, and then start from th beginning the obtainment of device information.

No error check is performed even if $ldev$ is invalid. The operation in this case is not guaranteed.

3.12.15 tk_evt_dev (Event Device)

Transmits the driver request event to the device.

Task portion	<input type="radio"/>	Task-independent portion	<input type="radio"/>	Dispatch disabled	<input type="radio"/>
--------------	-----------------------	--------------------------	-----------------------	-------------------	-----------------------

■ **C Language Interface**

```
INT retval = tk_evt_dev ( ID devid, INT evttyp, VP evtinf ) ;
```

■ **Parameter**

● **Input**

devid Event destination device ID (Device ID)
evttyp Driver request event type (Event Type)

Event	Value	Meaning
TDV_CARDEVT	1	PC card event
TDV_USBEVT	2	USB event

evtinf Each event type of information (Event Information)

● **Output**

retval Return value from the device driver (Return Value)
Or, error code (Error Code)

■ **Error Code**

E_NOEXS -42 Device of devid does not exist.
E_PAR -17 Device management internal event (evttyp < 0) cannot be specified.

■ **Dispatch Trigger**

If another task has called the device management function API and is in the wait state, this task also goes to the wait state until the other task's operation completes. Dispatch may be executed during this time.

■ **Description**

This system call transmits the driver request event to the device (device driver) of devid.
For the detail of each event type, see "T-Kernel Standard Device Driver Specification" (The specification can be downloaded from the website of T-Engine Forum free of charge.)

APPENDIX

The Appendices describe the error codes, define macros, device driver interface, and points to note when porting from a μ ITRON OS. An alphabetic index of system calls is also included.

APPENDIX A Error Codes

APPENDIX B Define Macros

APPENDIX C Device Driver Interface

APPENDIX D Points to Note When Porting From a μ ITRON OS

APPENDIX E System Call Index

APPENDIX A Error Codes

The table below lists the error codes.

■ Error Codes

Table A-1 Error Codes (1 / 2)

Label	Error Code ^{*1}			Meaning
E_OK	0	H'00	H'00000000	Successful completion
E_SYS ^{*2}	- 5	-H'05	H'fffffffb	System error
E_NOCOP ^{*2}	- 6	-H'06	H'fffffffa	Co-processor not available
E_NOSPT ^{*2}	- 9	-H'09	H'ffffff7	Unsupported function
E_RSFN ^{*2}	-10	-H'0a	H'ffffff6	Reserved function code number
E_RSATR	-11	-H'0b	H'ffffff5	"Reserved" attribute
E_PAR	-17	-H'11	H'fffffef	Parameter error
E_ID	-18	-H'12	H'fffffee	Invalid ID number
E_CTX	-25	-H'19	H'fffffe7	Context error
E_MACV ^{*2}	-26	-H'1a	H'fffffe6	Unable to access memory, or memory access permission violation
E_OACV	-27	-H'1b	H'fffffe5	Object access permission violation
E_ILUSE	-28	-H'1c	H'fffffe4	Invalid use of system call
E_NOMEM	-33	-H'21	H'fffffdf	Insufficient memory
E_LIMIT	-34	-H'22	H'fffffde	System limit exceeded
E_OBJ	-41	-H'29	H'fffffd7	Object status is invalid
E_NOEXS	-42	-H'2a	H'fffffd6	Object does not exist
E_QOVR	-43	-H'2b	H'fffffd5	Queue or nesting level overflow
E_RLWAI	-49	-H'31	H'fffffcf	Forcibly released from wait state
E_TMOUT	-50	-H'32	H'fffffce	Polling failed or timeout
E_DLT	-51	-H'33	H'fffffcd	The object being waited on has been deleted
E_DISWAI ^{*2}	-52	-H'34	H'fffffcc	Wait released by wait prohibition
E_IO	-57	-H'39	H'fffffc7	Input/output error
E_NOMDA ^{*2}	-58	-H'3a	H'fffffc6	No media
E_BUSY	-65	-H'41	H'fffffbe	Busy

Table A-1 Error Codes (2 / 2)

Label	Error Code ^{*1}			Meaning
E_ABORT ^{*2}	-66	-H'42	H'ffffffbd	Aborted
E_RDONLY	-67	-H'43	H'ffffffbc	Write-prohibited

*1: The error codes are shown (from left to right) in signed decimal, signed hexadecimal, and absolute hexadecimal format respectively

*2: Not used in μ T-Kernel. Definitions are retained for compatibility with T-Kernel.

APPENDIX B Define Macros

The table below lists the define macros.

■ Define Macros

Table B-1 Define Macros (1 / 4)

Category	Macro Name	Value	Meaning
Task	TSK_SELF	0	This task
	TPRI_INI	0	Initial priority
	TPRI_RUN	0	Priority of currently executing task
Object creation	TA_ASM	0	Coded in assembly language
	TA_HLNG	1	Coded in C
	TA_USERBUF	0x20	Specifies a user buffer
	TA_DSNAME	0x40	Indicates a DS object name
	TA_RNG0	0	Memory protection level 0
	TA_RNG1	0x100	Memory protection level 1
	TA_RNG2	0x200	Memory protection level 2
	TA_RNG3	0x300	Memory protection level 3
Task state	TTS_RUN	0x1	Run state
	TTS_RDY	0x2	Ready state
	TTS_WAI	0x4	Wait state
	TTS_SUS	0x8	Suspended state
	TTS_WAS	0xc	Wait-suspend state (WAITING + SUSPENDED)
	TTS_DMT	0x10	Idle
Task wait states	TTW_SLP	0x1	Sleeping
	TTW_DLY	0x2	Delayed
	TTW_SEM	0x4	Waiting on a semaphore
	TTW_FLG	0x8	Waiting on an event flag
	TTW_MBX	0x40	Waiting on mailbox reception
	TTW_MTX	0x80	Waiting on a mutex
	TTW_SMBF	0x100	Waiting on a message buffer send
	TTW_RMBF	0x200	Waiting on message buffer reception
	TTW_CAL	0x400	Waiting on a rendezvous call

Table B-1 Define Macros (2 / 4)

Category	Macro Name	Value	Meaning
Task wait states	TTW_ACP	0x800	Waiting on rendezvous reception
	TTW_RDV	0x1000	Waiting on a rendezvous termination
	TTW_MPF	0x2000	Waiting on acquisition of a fixed-length memory pool
	TTW_MPL	0x4000	Waiting on acquisition of a variable-length memory pool
Wait queue	TA_TFIFO	0	Wait queue uses FIFO order
	TA_TPRI	0x1	Wait queue uses task priority order
	TA_FIRST	0	Wait queue gives priority to first task in queue
	TA_CNT	0x2	Give priority to task with the smallest number of requests
Mutex attributes	TA_INHERIT	0x2	Priority inheritance protocol
	TA_CEILING	0x3	Priority upper limit protocol
Event flag attributes	TA_WSGL	0	Prohibit waiting by multiple tasks
	TA_WMUL	0x8	Permit waiting by multiple tasks
Event flag wait modes	TWF_ANDW	0	Event flag AND wait
	TWF_ORW	0x1	Event flag OR wait
	TWF_CLR	0x10	Clear all event flags
	TWF_BITCLR	0x20	Clear specified event flags
Mailbox attributes	TA_MFIFO	0	Manage messages on a FIFO basis
	TA_MPRI	0x2	Manage messages in priority order
Cyclic handler attributes	TA_STA	0x2	Start cyclic handler
	TA_PHS	0x4	Save phase of cyclic handler
Cyclic handler operation	TCYC_STP	0	Stop cyclic handler
	TCYC_STA	0x1	Start cyclic handler
Alarm handler operation	TALM_STP	0	Stop alarm handler
	TALM_STA	0x1	Start alarm handler
System states	TSS_TSK	0	Task executing
	TSS_DDSP	0x1	Dispatch disabled
	TSS_DINT	0x2	Interrupts disabled
	TSS_INDP	0x4	Task-independent portion executing
	TSS_QTSK	0x8	quasi-task executing
Common device attributes	TD_PROTECT	0x8000	Write-protect
	TD_REMOVABLE	0x4000	Removable media

Table B-1 Define Macros (3 / 4)

Category	Macro Name	Value	Meaning
Device types	TD_DEVKIND	0xff	Device type/media type mask
	TD_DEVTYPE	0xf0	Device type mask
	TDK_UNDEF	0	Undefined device
Disk devices	TDK_DISK	0x10	Disk device
	TDK_DISK_UNDEF	0x10	Undefined disk device
	TDK_DISK_RAM	0x11	RAM disk
	TDK_DISK_ROM	0x12	ROM disk
	TDK_DISK_FLA	0x13	FLASH memory disk
	TDK_DISK_FD	0x14	Floppy disk
	TDK_DISK_HD	0x15	Hard disk
	TDK_DISK_CDROM	0x16	CD-ROM
Device open modes	TD_READ	0x1	Read-only
	TD_WRITE	0x2	Write-only
	TD_UPDATE	0x3	Read and write
	TD_EXCL	0x100	Open for exclusive access
	TD_WEXCL	0x200	Open for writing only
	TD_REXCL	0x400	Open for reading only
Device close options	TD_EJECT	0x1	Eject media when closing
Device suspend modes	TD_SUSPEND	0x1	Suspend
	TD DISSUS	0x2	Suspend disabled
	TD_ENASUS	0x3	Suspend enabled
	TD_CHECK	0x4	Check suspend enable/disable state
	TD_FORCE	0x8000	Forcibly suspend
Message buffers used to notify information	TDN_EVENT	-1	ID of message buffer used to notify information
	TDN_DISKINFO	-2	Disk information
	TDN_DISPSPEC	-3	Display device information
	TDN_PCMCIAINFO	-4	PC card information
Device registration	TDA_OPENREQ	0x1	Open and close for each operation
Device requests	TDC_READ	0x1	Read request
	TDC_WRITE	0x2	Write request

Table B-1 Define Macros (4 / 4)

Category	Macro Name	Value	Meaning
Driver events	TDV_SUSPEND	-1	Suspend
	TDV_RESUME	-2	Resume
	TDV_CARDEVT	0x1	PC card event
	TDV_USBEVT	0x2	USB event
NULL	NULL	0	NULL
	TA_NULL	0	Unspecified attribute
Timeouts	TMO_FEVR	-1	Wait indefinitely
	TMO_POL	0	Polling

APPENDIX C Device Driver Interface

This appendix describes the interface between the kernel and device drivers.

■ Device Driver Interface

This is the interface between device drivers (which are user programs) and the device management functions of μ T-REALOS. The device driver interface consists of the following elements.

- Device naming rules
- Device ID
- Device attributes
- Device descriptor
- Request ID
- Data number
- Data format of device input/output requests
- Device processing functions
- Device attribute data
- Device event notification
- Suspend and resume

■ Device Naming Rules

Device names are a character string of up to eight characters and are made up of the following parts.

Type: A name indicating the device type. Permitted characters are a to z and A to Z.

Unit: A number indicating the physical device. Available letters are a to z. It is specified by one letter. It is allocated sequentially from a for each unit.

Sub-unit: A number indicating the logical device. Available numbers are 0 to 254 and the maximum length is three digits. It is allocated sequentially from 0 for each sub-unit.

A device name consists of the type + unit + sub-unit, but the unit and sub-unit parts are not used for all devices. In this case, these fields are omitted.

In particular, the name format for physical devices is type + unit. A name that consists of type + unit + sub-unit is called a logical device name to distinguish it from a physical device name. If there is no sub-unit, the physical device name and logical device name are the same. The term "device name" on its own indicates the logical device name.

Sub-units typically represent partitions on a hard disk, but can also be used to indicate other types of logical device.

Examples:

```
had   Hard disk (entire disk)
hda0  Hard disk (first partition)
fda   Floppy disk
rsa   Serial port
kbpd  Keyboard or pointing device
```

■ Device ID

Registering a device (device driver) in μ T-Kernel allocates a device ID (>0) to the device (physical device name). A separate device ID is allocated to each physical device and the device ID of a logical device is its sub-unit number + 1 (1 to 255) added to this device ID for the physical device

devid: Device ID allocated when device is registered

devid Physical device

devid+n+1 nth sub-unit (logical device)

Example:

```
hda   devid   Entire hard disk
hda0  devid + 1 First hard disk partition
hda1  devid + 2 Second hard disk partition
```

■ Device Attributes

Device attributes are defined as follows to indicate the device type and represent the characteristics of the device.

IIII IIII IIII IIII PRxx xxxx KKKK KKKK

The upper 16 bits contain the device-specific attributes which are defined separately for each device. The lower 16 bits contain the standard attributes.

The standard attributes have the following information.

- Common attribute(bit15 to bit8)

Name	Value	Meaning
TD_PROTECT	0x8000	P: Write-protect
TD_REMOVABLE	0x4000	R: Removable media

- Device type(bit15 to bit4) and media type(bit3 to bit0)

Name	Value	Meaning
TD_DEVKIND	0x00ff	K: Device or media type
TD_DEVTYPE	0x00f0	Device type
TDK_UNDEF	0x0000	Undefined or unknown
TDK_DISK	0x0010	Disk device
TDK_DISK_UNDEF	0x0010	Other disk
TDK_DISK_RAM	0x0011	RAM disk (using main memory)
TDK_DISK_ROM	0x0012	ROM disk (using main memory)
TDK_DISK_FLA	0x0013	Flash ROM or other silicon disk
TDK_DISK_FD	0x0014	Floppy disk
TDK_DISK_HD	0x0015	Hard disk
TDK_DISK_CDROM	0x0016	CD-ROM

Device types other than those listed above are not defined at this moment but new devices will be defined additionally in the future. TDK_UNDEF is used for undefined devices. The device type is defined to allow the user of the device to identify the type of device so that, for example, an application can change its operation depending on the type of device or media being used. Assigning a device type is not necessary in the case of devices for which an explicit identification is not required.

■ Device Descriptor

An identifier used for device access. A device descriptor (>0) is allocated by μ T-Kernel when the device is opened.

■ Request ID

A request ID (>0) is an identifier allocated when an I/O request is issued to the device to identify the request. The request ID is used to wait for the I/O operation to complete.

■ Data Number

Device data is specified by data number. The following two types of data number are used.

- Device-specific data: Data number ≥ 0
The data numbers for device-specific data are defined separately for each device.
Example: Disk Data number = Physical block number
 Serial link Only data number 0 is used
- Attribute data: Data number < 0
Used to get or set device or driver status information, for special functions, and other uses.
Although standard definitions are provided for some data numbers, these can also be defined separately for each device.

■ Attribute Data

The following three types of attribute data are used.

- Common attributes
Attributes defined for use by all devices (device drivers).
- Device-type-specific attributes
Attributes defined for use by all devices (device drivers) of a particular type.
- Device-specific attributes
Attributes defined independently for each device (device driver).

The device-type-specific and device-specific attributes are defined separately for each device. This section defines the common attributes only.

Common attributes have an attribute data number in the range -1 to -99. Although the data numbers for common attributes are the same for all devices, this does not mean that every device supports all the common attributes. An E_PAR error is returned if an unsupported data number is specified.

Common Attribute	Attribute Data No.	Meaning
TDN_EVENT	(-1)	RW: ID of message buffer used to notify events
TDN_DISKINFO	(-2)	R-: Disk information
TDN_DISPSPEC	(-3)	R-: Display device specifications

RW: Allows reading (tk_rea_dev) and writing (tk_wri_dev)

R-: Allows reading (tk_rea_dev) only

● TDN_EVENT : ID of message buffer used to notify events

Data format: ID

This is the ID of the message buffer used to notify device events. As the system default message buffer ID is passed when the device is registered, set that ID as the initial value when starting the driver.

Device event notification is not performed if this attribute is set to zero.

● TDN_DISKINFO : Disk information

Data format: DiskInfo

```
typedef enum {
    DiskFmt_STD = 0,          /* Standard (HD, etc.) */
    DiskFmt_2DD = 1,         /* 2DD 720KB */
    DiskFmt_2HD = 2,         /* 2HD 1.44MB */
    DiskFmt_CDROM = 4,       /* CD-ROM 640MB */
} DiskFormat;

typedef struct {
    DiskFormat format;        /* Format */
    UW protect:1;            /* Whether or not protected */
    UW removable:1;          /* Whether removable or not */
    UW rsv:30;               /* Reserved (always zero) */
    W blocksize;             /* No. of bytes per block */
    W blockcount;            /* Total number of blocks */
} DiskInfo;
```

● TDN_DISPSPEC : Display device specifications

Data format: DEV_SPEC

```
typedef struct {
    H attr;                  /* Device attribute */
    H planes;                /* No. of planes */
    H pixbits;               /* No. of bits per pixel (border/effective) */
    H hpixels;               /* No. of horizontal pixels */
}
```

```

H vpixels;    /* No. of vertical pixels */
H hres;       /* Horizontal resolution */
H vres;       /* Vertical resolution */
H color[4];   /* Color information */
H resv[6];    /* Reserved */
} DEV_SPEC;

```

■ Data Format of I/O Request

An I/O request to a device driver uses the following request packet, which is linked to a request ID.

```

typedef struct t_devreq {
    struct t_devreq *next;    /* I: Request packet link (NULL = terminator) */
    VP exinf;                 /* X: Extended information */
    ID devid;                 /* I: Device ID for request */
    INT cmd;                  /* I: Request command */
    BOOL abort;1;            /* I: TRUE if an abort request occurred */
    INT start;                /* I: Start data number */
    INT size;                 /* I: Request size */
    VP buf;                   /* I: I/O buffer address */
    INT asize;                /* O: Result size */
    ER error;                 /* O: Result error */
} T_DEVREQ;

```

The meaning of the letters in the comments are: I = input parameter, O = output parameter, X = parameter not used by the device management function. Input parameters cannot be modified by the device driver. The device management function initially clears all parameters other than input parameters (I) to zero. Device management does not subsequently make any changes.

The "next" member is used to link request packets. In addition to being used to manage request packets within the device management function, the member is also used by the wait for completion routine (waitfn), and abort operation routine (abortfn).

The "exinf" member can be used by the device driver without restriction. The device management function does not use this member.

The "devid" member specifies the ID of the device to which the request applies.

The "cmd" member specifies the request command.

cmd := (TDC_READ || TDC_WRITE)

cmd	Value	Meaning
TDC_READ	1	Read request
TDC_WRITE	2	Write request

The "abort" member is set to TRUE if the operation is aborted immediately prior to calling the abort routine (abortfn). The meaning of the "abort" flag is that aborting the operation was requested. It does not necessarily indicate that the operation has actually been aborted. It is also possible to set "abort" to TRUE without calling the abort routine (abortfn). If a request with abort = TRUE is passed to a device driver, the driver aborts the operation.

The "start" and "size" members are set to the values of the "start:" and "size" arguments passed to tk_rea_dev or tk_wri_dev.

The "buf" member is set to the buf parameter passed to tk_rea_dev or tk_wri_dev.

The "asize" member is set by the device driver to the asize value returned by tk_wai_dev.

The "error" member is set by the device driver and used as the error code returned by tk_wai_dev. The driver sets E_OK if the operation completed normally.

■ Device Processing Functions

These operation routines are called by device management function. When writing these routines, ensure that they have a reentrant structure. Note also that these operation routines do not necessarily have exclusive control of the CPU during execution. If multiple tasks issue requests to the same device simultaneously, for example, the routines may be called simultaneously by the different tasks. Accordingly, the device driver needs to use exclusive control techniques, as required.

● Open function (openfn)

<Call interface>

```
ER ercd = openfn( ID devid, UINT omode, VP exinf )
```

<Parameter>

[Input]

devid	Device ID of device to open
omode	Open mode (same as tk_opn_dev)
exinf	Additional information specified when registering device

[Output]

ercd	Error code
------	------------

<Explanation>

Device management calls the openfn routine when tk_opn_dev is called.

The openfn routine performs any initialization to prepare the device for use. The actual operation depends on the device and the operation can be performed freely to suit the device. No operation needs to be performed if no operation is required. Also, the device driver does not need to manage whether or not the device is open, nor does it need to cause other operation routines to return an error if the device is not open (if openfn has not been called). If another operation routine is called when the device is not open, it is permitted for the driver to process the request provided it does not cause any internal problem for the device driver.

Even if `openfn` needs to perform device initialization or other operations, it must not perform any operation that involves going to the wait state. The `openfn` routine should complete its operation as quickly as possible and return. For devices such as a serial link where the communication mode needs to be set, for example, the device can be initialized when `tk_wri_dev` is used to set the communication mode. Accordingly, device initialization does not need to be performed by `openfn`.

If the same device is opened more than once, `openfn` is typically only called for the first open operation. However, if the `TDA_OPENREQ` driver attribute is specified when the device is registered, `openfn` is called for all open operations. As multiple open operations, management of the open mode, and similar processing is handled by device management function, these sort of operations are not required in `openfn`. Similarly, although `omode` is passed in case it is required for some reason, management of `omode` is not normally required in `openfn`.

● Close function (`closefn`)

<Call interface>

```
ER ercd = closefn( ID devid, UINT option, VP exinf )
```

<Parameter>

[Input]

<code>devid</code>	Device ID of device to close
<code>option</code>	Close option (same as <code>tk_cls_dev</code>)
<code>exinf</code>	Additional information specified when registering device

[Output]

<code>ercd</code>	Error code
-------------------	------------

<Explanation>

Device management calls the `closefn` routine when `tk_cls_dev` is called.

The `closefn` routine performs any operations associated with ending use of the device. The actual operation depends on the device and the operation can be performed freely to suit the device. No operation needs to be performed if no operation is required.

In the case of devices with removable media, eject the media if `TD_EJECT` is specified in the option parameter.

Even if `closefn` needs to perform device shutdown processing, eject the media, or other operations, it must not perform any operation that involves going to the wait state. The `closefn` routine should complete its operation as quickly as possible and return. Even if ejecting the media takes some time, it is OK to return from `closefn` immediately without waiting for the eject operation to complete.

If the same device is opened more than once, this routine is typically only called for the final close operation. However, if the `TDA_OPENREQ` driver attribute is specified when the device is registered, `openfn` is called for all close operations. In this case, however, `TDA_EJECT` will not be set in the option parameter for any close operation other than the final close operation. As multiple open operations, management of the open mode, and similar processing is handled by device management function, these sort of operations are not required in `closefn`.

● Operation start function (execfn)

<Call interface>

```
ER ercd = execfn( T_DEVREQ *devreq, TMO tmout, VP exinf )
```

<Parameter>

[Input]

devreq	Request packet
tmout	Timeout for receiving request (ms)
exinf	Additional information specified when registering device

[Output]

ercd	Error code
------	------------

<Explanation>

Device management calls the execfn routine when tk_rea_dev, tk_srea_dev, tk_wri_dev , or tk_swri_dev are called.

The execfn routine initiates processing of the request specified in the devreq parameter. The routine initiates the operation only and then returns immediately to the caller. How long it takes to start the operation depends on the device driver and this step may not necessarily finish immediately.

Reception of new requests is delayed if the device is unable to handle any new requests. If the device does not become able to accept the new request within the time specified by the tmout parameter, perform timeout processing. The permitted values for the tmout parameter include TMO_POL (= 0) and TMO_FEVR (= -1). The execfn return value if a timeout occurs is E_TMOUT. Do not modify the error member in the request packet. The timeout only applies to the time up until the request is accepted. Do not create a timeout after the request has been accepted.

When execfn returns an error, delete the request packet as the request has not been accepted. If the operation is aborted due to device driver before the request has been accepted (before starting the operation), E_ABORT is returned as execfn return value. Delete the request packet in this case. Return E_OK if the abort occurs after the request has been accepted (after starting the operation). In this case, the request packet is not deleted until waitfn is called and the termination of the operation confirmed.

● Wait for completion function (waitfn)

<Call interface>

```
INT pktno = waitfn( T_DEVREQ *devreq, INT nreq, TMO tmout, VP exinf )
```

<Parameter>

[Input]

devreq	Request packet list
nreq	Number of request packets
tmout	Timeout (ms)
exinf	Additional information specified when registering device

[Output]

pktno	Number of completed request packet, or an error code
-------	--

<Explanation>

Device management calls the waitfn routine when tk_srea_dev, tk_swri_dev, or tk_wai_dev are called.

The devreq parameter contains a list of request packets linked by the devreq->next member. This routine waits until any of the nreq request packets starting from devreq have completed. Always process the list in accordance with the nreq parameter because you cannot be sure that the final packet in the list will have next = NULL. Return the number (sequence in list relative to devreq) of the request packet corresponding to the completed request in the return value. The first request packet is number 0 and the final packet is number nreq-1. Note that completion of an operation may mean successful completion, abnormal (error) completion, or abort.

The tmout parameter specifies the timeout for waiting for completion. The permitted values for this parameter include TMO_POL (= 0) and TMO_FEVR (= -1). If a timeout occurs, this indicates that the requested operation is still in progress. The waitfn return value if a timeout occurs is E_TMOUT. Do not modify the error member in the request packet. If waitfn returns while the operation is still in progress, waitfn must return an error. While an error return value does not necessarily mean that the operation is still in progress, returning a packet number in the return value always indicates that the corresponding operation has completed. The request is treated as still being in progress and the request packet not delete for as long as waitfn returns an error. When waitfn returns the number of the request packet corresponding to the completed request, delete the request packet as the requested operation is now complete.

Set any device related errors, such as I/O errors, in the error member of the request packet. The waitfn return value contains the error if the routine was unable to wait for completion. The waitfn return value is used as the return value for tk_wai_dev and the error member of the request packet is set back to ioer.

When the operation for waiting for completion is aborted due to device driver or an abort request (abort flag in request packet=TRUE), use a different abort process for a case of waiting for completion of a single request (nreq=1) and for a case of waiting for completion of multiple requests (nreq>1). When waiting for completion of a single request, abort the current request. When waiting for completion of a multiple requests, abort the process of waiting for completion only (cancel the wait) and do not abort the requested operations themselves. When waiting for completion of a multiple requests is aborted (wait cancelled), return E_ABORT as the waitfn return value.

In addition, for a case of multiple requests, the abort process can be ignored. However, it is recommended to perform an abort process for an abort request as much as possible. In this case, if waiting on the completion of a single request, abort the request. If waiting on multiple requests, although aborting the requests is preferable, it is also acceptable just to ignore the abort flag.

When aborting, it is important that waitfn returns as quickly as possible, and aborting is not necessary if the operation will terminate immediately in any case. Although the general rule is that the error member of the request packet must be returned with the value E_ABORT if the operation is aborted, returning an error other than E_ABORT is permitted if appropriate for the characteristics of the device. Similarly, returning E_OK indicating that the operation was valid immediately prior to aborting is also permitted. Return E_OK if the operation will continue normally until it completes even if an abort request occurs.

● Abort operation function (abortfn)

<Call interface>

```
ER ercd = abortfn( ID tskid, T_DEVREQ *devreq, INT nreq, VP exinf )
```

<Parameter>

[Input]

tskid	Task ID of the task that is executing execfn or waitfn.
devreq	Request packet list
nreq	Number of request packets
exinf	Additional information specified when registering device

[Output]

ercd	Error code
------	------------

<Explanation>

Device management calls the abortfn routine when tk_cls_dev is called.

The abortfn routine aborts the execution of execfn or waitfn for the specified requests. Normally, the requested operation is also aborted. However, aborting is not necessary if the operation will terminate immediately in any case. The important requirement is that execfn or waitfn return as quickly as possible.

The tskid parameter specifies the task that is executing the request indicated by devreq. That is, it indicates the task that is executing execfn or waitfn. The devreq and nreq parameters must be the same as the input parameters passed to execfn or waitfn. However, for execfn, the value of nreq is always "1".

The abortfn routine can be called from a task other than the task that called execfn or waitfn. As both tasks will be executing concurrently, exclusive control and similar measures should be used as required. It is possible that abortfn is called prior to execfn or waitfn being called, or while execfn or waitfn is returning. Ensure that your implementation operates correctly in such cases.

Set the abort flag to TRUE in the request packet for the request to be aborted before calling the abortfn routine. The execfn and waitfn routines can also use the abort flag to determine whether or not an abort request has occurred.

Note that the following special considerations apply when waitfn has been called to wait on multiple requests (nreq>1).

- Abort waiting only (cancel the wait), without aborting the requested operation.
- Do not set the abort flag in the request pattern (leave set to abort = FALSE).

To abort a request when neither execfn nor waitfn are executing, just set the abort flag in the request packet and do not call abortfn. A request is not accepted if the abort flag is set when execfn is called. When waitfn is called, it performs the same abort processing as when abortfn is called.

If a request for an operation that has already been started by execfn is aborted at a time when waitfn is not being used to wait for the operation to complete, notify that the aborted operation has completed when waitfn is subsequently called. Even if the request is aborted, do not delete the request itself until waitfn is called to check for completion.

The abortfn routine only initiates abort processing and returns immediately without waiting for abort processing to complete.

The abortfn is called in the following situations.

- When a device is closed by tk_cls_dev or by the clean-up processing for the sub-system, any active requests for the device descriptor being closed are aborted.

● Event function (eventfn)

<Call interface>

```
INT rtncd = eventfn( INT evttyp, VP evtinf, VP exinf )
```

<Parameter>

[Input]

evttyp Driver request event type
 evtinf Information associated with the event type
 exinf Additional information specified when registering device

[Output]

rtncd One of the return values defined for the event type, or an error

<Explanation>

The following driver request event types are used. Positive values are the result of calling tk_evt_dev and negative values the result of internal calls from within device management function.

Event	Value	Meaning
TDV_SUSPEND	(-1)	Suspend
TDV_RESUME	(-2)	Resume
TDV_CARDEVT	1	PC card event
TDV_USBEVT	2	USB event

The operation performed by the event routine is defined separately for each event type. When called by tk_evt_dev, the value returned by eventfn is used as the tk_evt_dev return value.

Requests to the event routine must be able to be handled even while processing other requests and should be processed as quickly as possible.

■ Device Event Notification

The device driver sends notification of events that occur on the device using the device event message buffer (TDN_EVENT).

The following event types are used.

```
typedef enum tdevttyp {
    TDE_unknown = 0,           /* Undefined */
    TDE_MOUNT = 0x01,          /* Media inserted */
    TDE_EJECT = 0x02,          /* Media ejected */
    TDE_ILLMOUNT = 0x03,        /* Media inserted incorrectly */
    TDE_ILLEJECT = 0x04,        /* Media ejected incorrectly */
    TDE_REMOUNT = 0x05,         /* Media re-inserted */
    TDE_CARDBATLOW = 0x06,      /* Card battery low warning */
    TDE_CARDBATFAIL = 0x07,     /* Card battery faulty */
    TDE_REQEJECT = 0x08,        /* Media eject requested */
    TDE_PDBUT = 0x11,           /* Change in state of PD button */
    TDE_PDMOVE = 0x12,          /* PD movement */
    TDE_PDSTATE = 0x13,         /* PD state change */
    TDE_PDEXT = 0x14,           /* PD extended event */
    TDE_KEYDOWN = 0x21,         /* Key down */
    TDE_KEYUP = 0x22,           /* Key up */
    TDE_KEYMETA = 0x23,         /* Change in state of meta-key */
    TDE_POWEROFF = 0x31,        /* Power switch off */
    TDE_POWERLOW = 0x32,        /* Power low warning */
    TDE_POWERFAIL = 0x33,       /* Power failure */
    TDE_POWERSUS = 0x34,        /* Automatic suspend */
    TDE_POWERUPTM = 0x35,       /* Clock updated */
    TDE_CKPWON = 0x41           /* Automatic power-on notification */
} TDEvtTyp;
```

The formats used for device event notification are shown below. The data associated with each type of event is different and has a different size.

```
typedef struct t_devevt {
    TDEvtTyp evttyp;           /* Event type */
    /* Append data corresponding to event type */
} T_DEVEVT;
```

The format for a device event notification that includes the device ID is as follows.

```
typedef struct t_devevt_id {
    TDEvtTyp  evttyp;          /* Event type */
    ID  devid;                /* Device ID */
    /* Append data corresponding to event type */
} T_DEVEVT_ID;
```

Ensure that the operation of the event receiver is not adversely affected if an event is not sent because the event notification message buffer is full. Waiting for the event buffer to become available before sending the event is permitted but this must not block any device driver operations other than event notification. Take care when implementing event receivers to minimize the possibility that the message buffer will overflow.

■ Suspend and Resume

Device drivers must suspend or resume device operation when suspend or resume event (TDV_SUSPEND or TDV_RESUME) is passed to the event function (eventfn). Suspend or resume events are only passed to physical devices.

● Suspend (TDV_SUSPEND)

evttyp = TDV_SUSPEND

evtinf = NULL (None)

Suspend the device as follows.

1. If any requests are currently in progress, either wait for them to complete or pause or abort them. Which method to use can be decided in the device driver implementation. However, as the device should be suspended as quickly as possible, pause or abort operations that may take a long time to complete. Although suspend events are only sent to physical devices, all logical devices associated with the physical device should be handled the same way.
 Pause: Temporarily halt operation, then restart after the device is resumed.
 Abort: Abort operation in the same way as when the abort function (abortfn) is used. Do not restart after the device is resumed.
2. Do not accept any new requests other than resume events.
3. Suspend the device in an appropriate way such as turning off the power.

As aborting an operation is likely to have a significant impact on the application, try to avoid this option where possible. Do not use abort except in cases such as waiting for input from a serial link over an extended period, or operations for which pausing is not practical. Instead, wait for the operation to complete or, if possible, pause it.

Hold any requests passed to the device driver while the device is suspended and then accept them when the device is resumed. However, operations such as those that do not involve device access may be performed while suspended.

● Resume (TDV_RESUME)

evttyp = TDV_RESUME

evtinf = NULL (None)

Resume device operation as follows.

1. Perform any resume processing such as turning on the device power and restoring the device state.
2. Restart any paused operations.
3. Resume accepting requests.

APPENDIX D Points to Note When Porting From a μ ITRON OS

This appendix describes the differences between a μ ITRON OS and μ T-REALOS.

■ What is Different From the μ ITRON OS?

Table D-1 lists the functional differences between μ T-REALOS and SOFTUNE REALOS/FR Spec.4, a μ ITRON 4.0 compliant realtime OS (referred to below as "REALOS/FR Spec.4"). Take note of these differences when porting user programs written for REALOS/FR Spec.4 to μ T-REALOS.

Table D-1 Differences Between the Functions of μ T-REALOS and REALOS/FR Spec.4

Function	μ T-REALOS	REALOS/FR Spec.4
Static creation of objects	×	○
ID number setting when objects created	×	○
Subsystem function	○	×
Device management function	○	×
Reserving of memory areas in the kernel (task stack and memory pool area)	○	×
Task exception function	×	○
Acquisition and release of multiple semaphore resources	○	×
Data queue function	×	○
Rendezvous port function	○	×
System calls that wait on an object * (Wait indefinitely, polling, timeout)	Same system calls	Separate system calls
Interrupt level modification	×	○
Bitwise clearing of event flags	Bitwise and all bits	All bits
Dynamic registration of interrupt handlers	○	×

○: Function is supported ×: Function is not supported

*:For example, different system calls are used to wait on a semaphore, as follows. The tk_wai_sem routine has a timeout parameter (tmout) and the value of this parameter controls the type of timeout to use.

Table D-2 Differences Between System Calls for Waiting on Semaphore

	REALOS/FR Spec.4	μ T-REALOS
Wait indefinitely	wai_sem()	tk_wai_sem()
Polling	pol_sem()	
Timeout	twai_sem()	

APPENDIX E System Call Index

This appendix lists the system calls alphabetically and indicates the page number of the system call explanation.

■ System Call Index

Table E-1 System Call Index (1 / 4)

System Call Name	Function	Page
DI	Disable external interrupts	186
EI	Enable external interrupts	187
isDI	Get enable/disable state for external interrupts	188
isig_tim (Signal Time)	Update system time	164
tk_acp_por (Accept Port for Rendezvous)	Accept port for rendezvous	123
tk_cal_por (Call Port for Rendezvous)	Call port for rendezvous	120
tk_can_wup (Cancel Wakeup Task)	Cancel task wakeup request	50
tk_chg_pri (Change Task Priority)	Change task priority	34
tk_clr_flg (Clear Event Flag)	Clear event flag	77
tk_cls_dev (Close Device)	Close device	212
tk_cre_alm (Create Alarm Handler)	Create alarm handler	175
tk_cre_cyc (Create Cyclic Handler)	Create cyclic handler	166
tk_cre_flg (Create Event Flag)	Create event flag	73
tk_cre_mbf (Create MessageBuffer)	Create message buffer	106
tk_cre_mbx (Create Mailbox)	Create mailbox	84
tk_cre_mpf (Create Fixed-size MemoryPool)	Create fixed-size memory pool	135
tk_cre_mpl (Create Variable-size MemoryPool)	Create variable-size memory pool	147
tk_cre_mtx (Create Mutex)	Create mutex	96
tk_cre_por (Create Port for Rendezvous)	Create port for rendezvous	116
tk_cre_sem (Create Semaphore)	Create semaphore	63
tk_cre_tsk (Create Task)	Create task	25
tk_def_dev (Define Device)	Define device	206
tk_def_int (Define Interrupt Handler)	Define interrupt handler	183
tk_def_ssy (Define Subsystem)	Define subsystem	201
tk_del_alm (Delete Alarm Handler)	Delete alarm handler	177

Table E-1 System Call Index (2 / 4)

System Call Name	Function	Page
tk_del_cyc (Delete Cyclic Handler)	Delete cyclic handler	169
tk_del_flg (Delete Event Flag)	Delete event flag	75
tk_del_mbf (Delete MessageBuffer)	Delete message buffer	108
tk_del_mbx (Delete Mailbox)	Delete mailbox	86
tk_del_mpf (Delete Fixed-size MemoryPool)	Delete fixed-size memory pool	138
tk_del_mpl (Delete Variable-size MemoryPool)	Delete variable-size memory pool	150
tk_del_mtx (Delete Mutex)	Delete mutex	98
tk_del_por (Delete Port for Rendezvous)	Delete port for rendezvous	118
tk_del_sem (Delete Semaphore)	Delete semaphore	65
tk_del_tsk (Delete Task)	Delete task	28
tk_dis_dsp (Disable Dispatch)	Disable dispatch	193
tk_dly_tsk (Delay Task)	Delay task	60
tk_ena_dsp (Enable Dispatch)	Enable dispatch	195
tk_evt_dev (Event Device)	Send a driver request event to a device	232
tk_exd_tsk (Exit and Delete Task)	Exit and delete invoking task	31
tk_ext_tsk (Exit Task)	Exit invoking task	30
tk_frsm_tsk (Force Resume Task)	Forcibly resume a suspended task	58
tk_fwd_por (Forward Rendezvous to Another Port)	Rendezvous at a rendezvous port	126
tk_get_dev (Get Device)	Get device name	225
tk_get_mpf (Get Fixed-size Memory Block)	Get fixed-size memory block	140
tk_get_mpl (Get Variable-size Memory Block)	Get variable-size memory block	152
tk_get_otm (Get Operating Time)	Get system running time	163
tk_get_reg (Get Task Registers)	Get task registers	37
tk_get_tid (Get Task Identifier)	Get task ID of a running task	192
tk_get_tim (Get Time)	Get system time	162
tk_loc_mtx (Lock Mutex)	Lock mutex	99
tk_lst_dev (List Device)	Get a list of registered devices	230
tk_opn_dev (Open Device)	Open device	210
tk_oref_dev (Refer Device)	Get device information	228
tk_rcv_mbf (Receive Message from MessageBuffer)	Receive message from message buffer	111
tk_rcv_mbx (Receive Message from Mailbox)	Receive message from mailbox	89

Table E-1 System Call Index (3 / 4)

System Call Name	Function	Page
tk_rea_dev (Read Device)	Start reading from device	213
tk_ref_alm (Refer Alarm Handler Status)	Get alarm handler status	180
tk_ref_cyc (Refer Cyclic Handler Status)	Get cyclic handler status	172
tk_ref_dev (Refer Device)	Get device information	226
tk_ref_flg (Refer Event Flag Status)	Get event flag status	81
tk_ref_idv (Refer Initial Device Information)	Get initial device information	209
tk_ref_mbf (Refer MessageBuffer Status)	Get message buffer status	113
tk_ref_mbx (Refer Mailbox Status)	Get mailbox status	92
tk_ref_mpf (Refer Fixed-size MemoryPool Status)	Get fixed-size memory pool status	144
tk_ref_mpl (Refer Variable-size MemoryPool Status)	Get variable-size memory pool status	156
tk_ref_mtx (Refer Mutex Status)	Get mutex status	103
tk_ref_por (Refer Port Status)	Get rendezvous port status	131
tk_ref_sem (Refer Semaphore Status)	Get semaphore status	70
tk_ref_ssy (Refer Subsystem Status)	Get subsystem definition	204
tk_ref_sys (Refer System Status)	Get system status	196
tk_ref_tsk (Refer Task Status)	Get task status	41
tk_ref_ver (Refer Version Information)	Get version information	198
tk_rel_mpf (Release Fixed-size Memory Block)	Release fixed-size memory block	142
tk_rel_mpl (Release Variable-size Memory Block)	Release variable-size memory block	154
tk_rel_wai (Release Wait)	Release wait state for another task	52
tk_ret_int (Return from Interrupt Handler)	Return from interrupt handler	185
tk_rot_rdq (Rotate Ready Queue)	Rotate task priority queue	190
tk_rpl_rdv (Reply Rendezvous)	Reply rendezvous	129
tk_rsm_tsk (Resume Task)	Resume a suspended task	56
tk_set_flg (Set Event Flag)	Set event flag	76
tk_set_reg (Set Task Registers)	Set task registers	39
tk_set_tim (Set Time)	Set system time	160
tk_sig_sem (Signal Semaphore)	Signal a semaphore	66
tk_slp_tsk (Sleep Task)	Put invoking task to sleep (wait for wakeup)	46
tk_snd_mbf (Send Message to MessageBuffer)	Send message to message buffer	109
tk_snd_mbx (Send Message to Mailbox)	Send message to mailbox	87

Table E-1 System Call Index (4 / 4)

System Call Name	Function	Page
tk_srea_dev (Synchronous Read Device)	Perform a synchronous read on a device	215
tk_sta_alm (Start Alarm Handler)	Start alarm handler	178
tk_sta_cyc (Start Cyclic Handler)	Start cyclic handler	170
tk_sta_tsk (Start Task)	Start task	29
tk_stp_alm (Stop Alarm Handler)	Stop alarm handler	179
tk_stp_cyc (Stop Cyclic Handler)	Stop cyclic handler	171
tk_sus_dev (Suspend Device)	Suspend device	223
tk_sus_tsk (Suspend Task)	Suspend another task	54
tk_swri_dev (Synchronous Write Device)	Perform a synchronous write to a device	219
tk_ter_tsk (Terminate Task)	Forcibly terminate another task	32
tk_unl_mtx (Unlock Mutex)	Unlock mutex	101
tk_wai_dev (Wait Device)	Wait for device request to complete	221
tk_wai_flg (Wait Event Flag)	Wait on an event flag	78
tk_wai_sem (Wait on Semaphore)	Wait to get semaphore	68
tk_wri_dev (Write Device)	Start writing to device	217
tk_wup_tsk (Wakeup Task)	Wakeup another task	48

INDEX

**The index follows on the next page.
This is listed in alphabetic order.**

Index

A	
Accept	
tk_acp_por (Accept Port for Rendezvous).....	123
Alarm Handler	
Alarm Handler Function System Calls.....	174
tk_cre_alm (Create Alarm Handler)	175
tk_del_alm (Delete Alarm Handler)	177
tk_ref_alm (Refer Alarm Handler Status)	180
tk_sta_alm (Start Alarm Handler)	178
tk_stp_alm (Stop Alarm Handler)	179
Attribute	
Attribute Data	242
Device Attributes	241
C	
Call	
tk_cal_por (Call Port for Rendezvous).....	120
Cancel	
tk_can_wup (Cancel Wakeup Task)	50
Change	
tk_chg_pri (Change Task Priority)	34
Clear	
tk_clr_flg (Clear Event Flag)	77
Close	
tk_cls_dev (Close Device).....	212
Communication	
System Calls for Extended Synchronization/ Communication Function	94
System Calls for Synchronization/Communication Function	61
Create	
tk_cre_alm (Create Alarm Handler)	175
tk_cre_cyc (Create Cyclic Handler).....	166
tk_cre_flg (Create Event Flag).....	73
tk_cre_mbf (Create MessageBuffer).....	106
tk_cre_mbx (Create Mailbox).....	84
tk_cre_mpf (Create Fixed-size MemoryPool).....	135
tk_cre_mpl (Create Variable-size MemoryPool)	147
tk_cre_mtx (Create Mutex)	96
tk_cre_por (Create Port for Rendezvous)	116
tk_cre_sem (Create Semaphore)	63
tk_cre_tsk (Create Task)	25
Cyclic Handler	
Cyclic Handler Function System Calls	165
tk_cre_cyc (Create Cyclic Handler).....	166
tk_del_cyc (Delete Cyclic Handler).....	169
tk_ref_cyc (Refer Cyclic Handler Status).....	172
tk_sta_cyc (Start Cyclic Handler).....	170
tk_stp_cyc (Stop Cyclic Handler)	171
D	
Data	
Attribute Data	242
Data Format of I/O Request	244
Data Number	242
Data Types	4
Data Types That Have a Specific Meaning in μT-Kernel	12
Standard Data Types	10
Define	
Define Macros	236
tk_def_dev (Define Device)	206
tk_def_int (Define Interrupt Handler).....	183
tk_def_ssy (Define Subsystem)	201
Definition	
Implementation Definition	5
Implementation-specific Definitions	7
Delay	
tk_dly_tsk (Delay Task)	60
Delete	
tk_del_alm (Delete Alarm Handler)	177
tk_del_cyc (Delete Cyclic Handler)	169
tk_del_flg (Delete Event Flag)	75
tk_del_mbf (Delete MessageBuffer)	108
tk_del_mbx (Delete Mailbox)	86
tk_del_mpf (Delete Fixed-size MemoryPool)	138
tk_del_mpl (Delete Variable-size MemoryPool)	150
tk_del_mtx (Delete Mutex)	98
tk_del_por (Delete Port for Rendezvous).....	118
tk_del_sem (Delete Semaphore)	65
tk_del_tsk (Delete Task)	28
tk_ext_tsk (Exit and Delete Task)	31
Descriptor	
Device Descriptor	242
Device	
Device Attributes	241
Device Descriptor	242
Device Driver Interface	240
Device Event Notification.....	251
Device ID.....	241
Device Naming Rules.....	240
Device Processing Functions.....	245
tk_cls_dev (Close Device)	212
tk_def_dev (Define Device)	206
tk_evt_dev (Event Device).....	232
tk_get_dev (Get Device).....	225
tk_lst_dev (List Device)	230

tk_opn_dev (Open Device)	210
tk_oref_dev (Refer Device)	228
tk_rea_dev (Read Device)	213
tk_ref_dev (Refer Device)	226
tk_ref_idv (Refer Initial Device Information)	209
tk_srea_dev (Synchronous Read Device)	215
tk_sus_dev (Suspend Device)	223
tk_swri_dev (Synchronous Write Device)	219
tk_wai_dev (Wait Device)	221
tk_wri_dev (Write Device)	217
Device Management	
Device Management Function System Calls	205
DI	
DI	186
Disable	
tk_dis_dsp (Disable Dispatch)	193
Dispatch	
tk_dis_dsp (Disable Dispatch)	193
tk_ena_dsp (Enable Dispatch)	195
E	
EI	
EI	187
Enable	
tk_ena_dsp (Enable Dispatch)	195
Error	
Error Codes	4, 234
Event	
Device Event Notification	251
tk_clr_flg (Clear Event Flag)	77
tk_cre_flg (Create Event Flag)	73
tk_del_flg (Delete Event Flag)	75
tk_evt_dev (Event Device)	232
tk_ref_flg (Refer Event Flag Status)	81
tk_set_flg (Set Event Flag)	76
tk_wai_flg (Wait Event Flag)	78
Event Flag	
Event Flag Function System Calls	72
Exit	
tk_exd_tsk (Exit and Delete Task)	31
tk_ext_tsk (Exit Task)	30
Extended Synchronization	
System Calls for Extended Synchronization/ Communication Function	94
Extensions	
Extensions	8
F	
Fixed-size	
tk_cre_mpf (Create Fixed-size MemoryPool)	135
tk_del_mpf (Delete Fixed-size MemoryPool)	138
tk_get_mpf (Get Fixed-size Memory Block)	140
tk_ref_mpf (Refer Fixed-size MemoryPool Status)	144
tk_rel_mpf (Release Fixed-size Memory Block)	142
Fixed-size Memory Pool	
Fixed-size Memory Pool Function System Calls	134
Flag	
tk_clr_flg (Clear Event Flag)	77
tk_cre_flg (Create Event Flag)	73
tk_del_flg (Delete Event Flag)	75
tk_ref_flg (Refer Event Flag Status)	81
tk_set_flg (Set Event Flag)	76
tk_wai_flg (Wait Event Flag)	78
Force	
tk_frsm_tsk (Force Resume Task)	58
Format	
Data Format of I/O Request	244
Forward	
tk_fwd_por (Forward Rendezvous to Another Port)	126
Function	
Device Processing Functions	245
H	
Handler	
tk_cre_alm (Create Alarm Handler)	175
tk_cre_cyc (Create Cyclic Handler)	166
tk_def_int (Define Interrupt Handler)	183
tk_del_alm (Delete Alarm Handler)	177
tk_del_cyc (Delete Cyclic Handler)	169
tk_ref_alm (Refer Alarm Handler Status)	180
tk_ref_cyc (Refer Cyclic Handler Status)	172
tk_ret_int (Return from Interrupt Handler)	185
tk_sta_alm (Start Alarm Handler)	178
tk_sta_cyc (Start Cyclic Handler)	170
tk_stp_alm (Stop Alarm Handler)	179
tk_stp_cyc (Stop Cyclic Handler)	171
I	
I/O	
Data Format of I/O Request	244
ID	
Device ID	241
Request ID	242
Identifier	
tk_get_tid (Get Task Identifier)	192
Implementation	
Implementation Definition	5
Implementation-specific Definitions	7
Information	
tk_ref_idv (Refer Initial Device Information)	209
tk_ref_ver (Refer Version Information)	198
Initial	
tk_ref_idv (Refer Initial Device Information)	209

INDEX

Interface	
Device Driver Interface.....	240
Interrupt	
tk_def_int (Define Interrupt Handler).....	183
tk_ret_int (Return from Interrupt Handler).....	185
Interrupt Management	
Interrupt Management Function System Calls	182
isDI	
isDI.....	188
isig_tim	
isig_tim (Signal Time).....	164
L	
List	
tk_lst_dev (List Device).....	230
Lock	
tk_loc_mtx (Lock Mutex)	99
M	
Macros	
Define Macros	236
Mailbox	
Mailbox Function System Calls	83
tk_cre_mbx (Create Mailbox).....	84
tk_del_mbx (Delete Mailbox).....	86
tk_rcv_mbx (Receive Message from Mailbox)	89
tk_ref_mbx (Refer Mailbox Status).....	92
tk_snd_mbx (Send Message to Mailbox)	87
Memory	
tk_get_mpf (Get Fixed-size Memory Block)	140
tk_get_mpl (Get Variable-size Memory Block)	
.....	152
tk_rel_mpf (Release Fixed-size Memory Block)	
.....	142
tk_rel_mpl (Release Variable-size Memory Block)	
.....	154
Memory Pool Management	
Memory Pool Management Function System Calls	
.....	133
MemoryPool	
tk_cre_mpf (Create Fixed-size MemoryPool).....	135
tk_cre_mpl (Create Variable-size MemoryPool)	
.....	147
tk_del_mpf (Delete Fixed-size MemoryPool).....	138
tk_del_mpl (Delete Variable-size MemoryPool)	
.....	150
tk_ref_mpf (Refer Fixed-size MemoryPool Status)	
.....	144
tk_ref_mpl (Refer Variable-size MemoryPool Status)	
.....	156
Message	
tk_rcv_mbf (Receive Message from MessageBuffer)	
.....	111
tk_rcv_mbx (Receive Message from Mailbox)	89
tk_snd_mbf (Send Message to MessageBuffer)	
.....	109
tk_snd_mbx (Send Message to Mailbox).....	87
Message Buffer	
Message Buffer Function System Calls	105
MessageBuffer	
tk_cre_mbf (Create MessageBuffer)	106
tk_del_mbf (Delete MessageBuffer)	108
tk_rcv_mbf (Receive Message from MessageBuffer)	
.....	111
tk_ref_mbf (Refer MessageBuffer Status)	113
tk_snd_mbf (Send Message to MessageBuffer)	
.....	109
μITRON	
What is Different From the μITRON OS?	254
μT-Kernel	
Data Types That Have a Specific Meaning in	
μT-Kernel	12
μT-REALOS	
μT-REALOS Terminology	2
Mutex	
Mutex Function System Calls.....	95
tk_cre_mtx (Create Mutex).....	96
tk_del_mtx (Delete Mutex).....	98
tk_loc_mtx (Lock Mutex)	99
tk_ref_mtx (Refer Mutex Status)	103
tk_unl_mtx (Unlock Mutex).....	101
N	
Notification	
Device Event Notification.....	251
O	
Open	
tk_opn_dev (Open Device)	210
Operating	
tk_get_otm (Get Operating Time).....	163
P	
Port	
tk_acp_por (Accept Port for Rendezvous)	123
tk_cal_por (Call Port for Rendezvous)	120
tk_cre_por (Create Port for Rendezvous).....	116
tk_del_por (Delete Port for Rendezvous).....	118
tk_fwd_por (Forward Rendezvous to Another Port)	
.....	126
tk_ref_por (Refer Port Status)	131
Priority	
tk_chg_pri (Change Task Priority).....	34
Q	
Queue	
tk_rot_rdq (Rotate Ready Queue)	190

R**Read**

tk_rea_dev (Read Device).....	213
tk_srea_dev (Synchronous Read Device).....	215

Ready Queue

tk_rot_rdq (Rotate Ready Queue)	190
---------------------------------------	-----

Receive

tk_rcv_mbf (Receive Message from MessageBuffer)	111
tk_rcv_mbx (Receive Message from Mailbox).....	89

Receive Message

tk_rcv_mbf (Receive Message from MessageBuffer)	111
tk_rcv_mbx (Receive Message from Mailbox).....	89

Refer

tk_oref_dev (Refer Device).....	228
tk_ref_alm (Refer Alarm Handler Status)	180
tk_ref_cyc (Refer Cyclic Handler Status)	172
tk_ref_dev (Refer Device).....	226
tk_ref_flg (Refer Event Flag Status).....	81
tk_ref_idv (Refer Initial Device Information).....	209
tk_ref_mbf (Refer MessageBuffer Status)	113
tk_ref_mbx (Refer Mailbox Status).....	92
tk_ref_mpf (Refer Fixed-size MemoryPool Status)	144
tk_ref_mpl (Refer Variable-size MemoryPool Status)	156
tk_ref_mtx (Refer Mutex Status)	103
tk_ref_por (Refer Port Status)	131
tk_ref_sem (Refer Semaphore Status)	70
tk_ref_ssy (Refer Subsystem Status)	204
tk_ref_sys (Refer System Status)	196
tk_ref_tsk (Refer Task Status)	41
tk_ref_ver (Refer Version Information).....	198

Registers

tk_get_reg (Get Task Registers)	37
tk_set_reg (Set Task Registers)	39

Release

tk_rel_mpf (Release Fixed-size Memory Block)	142
tk_rel_mpl (Release Variable-size Memory Block)	154
tk_rel_wai (Release Wait).....	52

Rendezvous

Rendezvous Function System Calls	115
tk_acp_por (Accept Port for Rendezvous)	123
tk_cal_por (Call Port for Rendezvous)	120
tk_cre_por (Create Port for Rendezvous).....	116
tk_del_por (Delete Port for Rendezvous).....	118
tk_fwd_por (Forward Rendezvous to Another Port)	126
tk_rpl_rdv (Reply Rendezvous)	129

Reply

tk_rpl_rdv (Reply Rendezvous)	129
-------------------------------------	-----

Request

Data Format of I/O Request	244
Request ID	242

Resume

Suspend and Resume	252
tk_frm_tsk (Force Resume Task)	58
tk_rsm_tsk (Resume Task)	56

Return

tk_ret_int (Return from Interrupt Handler)	185
--	-----

Rotate

tk_rot_rdq (Rotate Ready Queue)	190
---------------------------------------	-----

Rules

Device Naming Rules	240
---------------------------	-----

S**Semaphore**

Semaphore Function System Calls	62
tk_cre_sem (Create Semaphore).....	63
tk_del_sem (Delete Semaphore).....	65
tk_ref_sem (Refer Semaphore Status).....	70
tk_sig_sem (Signal Semaphore)	66
tk_wai_sem (Wait on Semaphore).....	68

Send

tk_snd_mbf (Send Message to MessageBuffer)	109
tk_snd_mbx (Send Message to Mailbox).....	87

Signal

isig_tim (Signal Time)	164
tk_sig_sem (Signal Semaphore)	66

Sleep

tk_slp_tsk (Sleep Task)	46
-------------------------------	----

Standard Data Types

Standard Data Types	10
---------------------------	----

Start

tk_sta_alm (Start Alarm Handler).....	178
tk_sta_cyc (Start Cyclic Handler)	170
tk_sta_tsk (Start Task).....	29

Status

tk_ref_alm (Refer Alarm Handler Status).....	180
tk_ref_cyc (Refer Cyclic Handler Status)	172
tk_ref_flg (Refer Event Flag Status)	81
tk_ref_mbf (Refer MessageBuffer Status)	113
tk_ref_mbx (Refer Mailbox Status)	92
tk_ref_mpf (Refer Fixed-size MemoryPool Status)	144
tk_ref_mpl (Refer Variable-size MemoryPool Status)	156
tk_ref_mtx (Refer Mutex Status)	103
tk_ref_por (Refer Port Status)	131
tk_ref_sem (Refer Semaphore Status).....	70
tk_ref_ssy (Refer Subsystem Status).....	204
tk_ref_sys (Refer System Status).....	196
tk_ref_tsk (Refer Task Status).....	41

INDEX

Stop	
tk_stp_alm (Stop Alarm Handler)	179
tk_stp_cyc (Stop Cyclic Handler)	171
Sub System	
Sub System Function System Calls	200
Subsystem	
tk_def_ssy (Define Subsystem)	201
tk_ref_ssy (Refer Subsystem Status)	204
Suspend	
Suspend and Resume	252
tk_sus_dev (Suspend Device)	223
tk_sus_tsk (Suspend Task)	54
Synchronization	
System Calls for Synchronization/Communication Function	61
Synchronous	
tk_srea_dev (Synchronous Read Device)	215
tk_swri_dev (Synchronous Write Device)	219
System	
tk_ref_sys (Refer System Status)	196
System Call	
Alarm Handler Function System Calls	174
Cyclic Handler Function System Calls	165
Device Management Function System Calls	205
Event Flag Function System Calls	72
Fixed-size Memory Pool Function System Calls	134
Interrupt Management Function System Calls	182
Mailbox Function System Calls	83
Memory Pool Management Function System Calls	133
Message Buffer Function System Calls	105
Mutex Function System Calls	95
Rendezvous Function System Calls	115
Semaphore Function System Calls	62
Sub System Function System Calls	200
System Call Index	255
System Calls	4
System Calls for Extended Synchronization/ Communication Function	94
System Calls for Synchronization/Communication Function	61
System Calls for Task-dependent Synchronization Function	45
System Calls for the Task Management Function	24
System Status Management Function System Calls	189
System Time Management Function System Calls	159
Time Management Function System Calls	158
Variable-size Memory Pool Function System Calls	146

System Status Management	
System Status Management Function System Calls	189
System Time Management	
System Time Management Function System Calls	159
T	
Task	
tk_can_wup (Cancel Wakeup Task)	50
tk_chg_pri (Change Task Priority)	34
tk_cre_tsk (Create Task)	25
tk_del_tsk (Delete Task)	28
tk_dly_tsk (Delay Task)	60
tk_exd_tsk (Exit and Delete Task)	31
tk_ext_tsk (Exit Task)	30
tk_frmr_tsk (Force Resume Task)	58
tk_get_reg (Get Task Registers)	37
tk_get_tid (Get Task Identifier)	192
tk_ref_tsk (Refer Task Status)	41
tk_rsm_tsk (Resume Task)	56
tk_set_reg (Set Task Registers)	39
tk_slp_tsk (Sleep Task)	46
tk_sta_tsk (Start Task)	29
tk_sus_tsk (Suspend Task)	54
tk_ter_tsk (Terminate Task)	32
tk_wup_tsk (Wakeup Task)	48
Task Management	
System Calls for the Task Management Function	24
Task-dependent Synchronization	
System Calls for Task-dependent Synchronization Function	45
Terminate	
tk_ter_tsk (Terminate Task)	32
Time	
isig_tim (Signal Time)	164
tk_get_otm (Get Operating Time)	163
tk_get_tim (Get Time)	162
tk_set_tim (Set Time)	160
Time Management	
Time Management Function System Calls	158
tk_acp_por	
tk_acp_por (Accept Port for Rendezvous)	123
tk_cal_por	
tk_cal_por (Call Port for Rendezvous)	120
tk_can_wup	
tk_can_wup (Cancel Wakeup Task)	50
tk_chg_pri	
tk_chg_pri (Change Task Priority)	34
tk_clr_flg	
tk_clr_flg (Clear Event Flag)	77

tk_cls_dev		
tk_cls_dev (Close Device)	212
tk_cre_alm		
tk_cre_alm (Create Alarm Handler)	175
tk_cre_cyc		
tk_cre_cyc (Create Cyclic Handler)	166
tk_cre_flg		
tk_cre_flg (Create Event Flag)	73
tk_cre_mbf		
tk_cre_mbf (Create MessageBuffer)	106
tk_cre_mbx		
tk_cre_mbx (Create Mailbox)	84
tk_cre_mpf		
tk_cre_mpf (Create Fixed-size MemoryPool)	135
tk_cre_mpl		
tk_cre_mpl (Create Variable-size MemoryPool)	147
tk_cre_mtx		
tk_cre_mtx (Create Mutex)	96
tk_cre_por		
tk_cre_por (Create Port for Rendezvous)	116
tk_cre_sem		
tk_cre_sem (Create Semaphore)	63
tk_cre_tsk		
tk_cre_tsk (Create Task)	25
tk_def_dev		
tk_def_dev (Define Device)	206
tk_def_int		
tk_def_int (Define Interrupt Handler)	183
tk_def_ssy		
tk_def_ssy (Define Subsystem)	201
tk_del_alm		
tk_del_alm (Delete Alarm Handler)	177
tk_del_cyc		
tk_del_cyc (Delete Cyclic Handler)	169
tk_del_flg		
tk_del_flg (Delete Event Flag)	75
tk_del_mbf		
tk_del_mbf (Delete MessageBuffer)	108
tk_del_mbx		
tk_del_mbx (Delete Mailbox)	86
tk_del_mpf		
tk_del_mpf (Delete Fixed-size MemoryPool)	138
tk_del_mpl		
tk_del_mpl (Delete Variable-size MemoryPool)	150
tk_del_mtx		
tk_del_mtx (Delete Mutex)	98
tk_del_por		
tk_del_por (Delete Port for Rendezvous)	118
tk_del_sem		
tk_del_sem (Delete Semaphore)	65
tk_del_tsk		
tk_del_tsk (Delete Task)	28
tk_dis_dsp		
tk_dis_dsp (Disable Dispatch)	193
tk_dly_tsk		
tk_dly_tsk (Delay Task)	60
tk_ena_dsp		
tk_ena_dsp (Enable Dispatch)	195
tk_evt_dev		
tk_evt_dev (Event Device)	232
tk_exd_tsk		
tk_exd_tsk (Exit and Delete Task)	31
tk_ext_tsk		
tk_ext_tsk (Exit Task)	30
tk_frsm_tsk		
tk_frsm_tsk (Force Resume Task)	58
tk_fwd_por		
tk_fwd_por (Forward Rendezvous to Another Port)	126
tk_get_dev		
tk_get_dev (Get Device)	225
tk_get_mpf		
tk_get_mpf (Get Fixed-size Memory Block)	140
tk_get_mpl		
tk_get_mpl (Get Variable-size Memory Block)	152
tk_get_otm		
tk_get_otm (Get Operating Time)	163
tk_get_reg		
tk_get_reg (Get Task Registers)	37
tk_get_tid		
tk_get_tid (Get Task Identifier)	192
tk_get_tim		
tk_get_tim (Get Time)	162
tk_loc_mtx		
tk_loc_mtx (Lock Mutex)	99
tk_lst_dev		
tk_lst_dev (List Device)	230
tk_opn_dev		
tk_opn_dev (Open Device)	210
tk_oref_dev		
tk_oref_dev (Refer Device)	228
tk_rcv_mbf		
tk_rcv_mbf (Receive Message from MessageBuffer)	111
tk_rcv_mbx		
tk_rcv_mbx (Receive Message from Mailbox)	89
tk_rea_dev		
tk_rea_dev (Read Device)	213
tk_ref_alm		
tk_ref_alm (Refer Alarm Handler Status)	180
tk_ref_cyc		
tk_ref_cyc (Refer Cyclic Handler Status)	172

INDEX

tk_ref_dev		
tk_ref_dev (Refer Device).....	226	
tk_ref_flg		
tk_ref_flg (Refer Event Flag Status).....	81	
tk_ref_idv		
tk_ref_idv (Refer Initial Device Information).....	209	
tk_ref_mbf		
tk_ref_mbf (Refer MessageBuffer Status).....	113	
tk_ref_mbx		
tk_ref_mbx (Refer Mailbox Status).....	92	
tk_ref_mpf		
tk_ref_mpf (Refer Fixed-size MemoryPool Status)		
.....	144	
tk_ref_mpl		
tk_ref_mpl (Refer Variable-size MemoryPool Status)		
.....	156	
tk_ref_mtx		
tk_ref_mtx (Refer Mutex Status).....	103	
tk_ref_por		
tk_ref_por (Refer Port Status).....	131	
tk_ref_sem		
tk_ref_sem (Refer Semaphore Status).....	70	
tk_ref_ssy		
tk_ref_ssy (Refer Subsystem Status)	204	
tk_ref_sys		
tk_ref_sys (Refer System Status)	196	
tk_ref_tsk		
tk_ref_tsk (Refer Task Status)	41	
tk_ref_ver		
tk_ref_ver (Refer Version Information)	198	
tk_rel_mpf		
tk_rel_mpf (Release Fixed-size Memory Block)		
.....	142	
tk_rel_mpl		
tk_rel_mpl (Release Variable-size Memory Block)		
.....	154	
tk_rel_wai		
tk_rel_wai (Release Wait).....	52	
tk_ret_int		
tk_ret_int (Return from Interrupt Handler).....	185	
tk_rot_rdq		
tk_rot_rdq (Rotate Ready Queue).....	190	
tk_rpl_rdv		
tk_rpl_rdv (Reply Rendezvous)	129	
tk_rsm_tsk		
tk_rsm_tsk (Resume Task).....	56	
tk_set_flg		
tk_set_flg (Set Event Flag).....	76	
tk_set_reg		
tk_set_reg (Set Task Registers).....	39	
tk_set_tim		
tk_set_tim (Set Time)	160	
tk_sig_sem		
tk_sig_sem (Signal Semaphore)	66	
tk_slp_tsk		
tk_slp_tsk (Sleep Task)	46	
tk_snd_mbf		
tk_snd_mbf (Send Message to MessageBuffer)		
.....	109	
tk_snd_mbx		
tk_snd_mbx (Send Message to Mailbox).....	87	
tk_srea_dev		
tk_srea_dev (Synchronous Read Device).....	215	
tk_sta_alm		
tk_sta_alm (Start Alarm Handler).....	178	
tk_sta_cyc		
tk_sta_cyc (Start Cyclic Handler)	170	
tk_sta_tsk		
tk_sta_tsk (Start Task).....	29	
tk_stp_alm		
tk_stp_alm (Stop Alarm Handler).....	179	
tk_stp_cyc		
tk_stp_cyc (Stop Cyclic Handler)	171	
tk_sus_dev		
tk_sus_dev (Suspend Device).....	223	
tk_sus_tsk		
tk_sus_tsk (Suspend Task).....	54	
tk_swri_dev		
tk_swri_dev (Synchronous Write Device)	219	
tk_ter_tsk		
tk_ter_tsk (Terminate Task)	32	
tk_unl_mtx		
tk_unl_mtx (Unlock Mutex).....	101	
tk_wai_dev		
tk_wai_dev (Wait Device)	221	
tk_wai_flg		
tk_wai_flg (Wait Event Flag)	78	
tk_wai_sem		
tk_wai_sem (Wait on Semaphore)	68	
tk_wri_dev		
tk_wri_dev (Write Device)	217	
tk_wup_tsk		
tk_wup_tsk (Wakeup Task)	48	
Types		
Data Types	4	
Data Types That Have a Specific Meaning in		
μT-Kernel	12	
Standard Data Types	10	
U		
Unlock		
tk_unl_mtx (Unlock Mutex).....	101	

V

Variable

tk_cre_mpl (Create Variable-size MemoryPool)	147
tk_del_mpl (Delete Variable-size MemoryPool)	150
tk_get_mpl (Get Variable-size Memory Block)	152
tk_ref_mpl (Refer Variable-size MemoryPool Status)	156
tk_rel_mpl (Release Variable-size Memory Block)	154

Variable-size

tk_cre_mpl (Create Variable-size MemoryPool)	147
tk_del_mpl (Delete Variable-size MemoryPool)	150
tk_get_mpl (Get Variable-size Memory Block)	152
tk_ref_mpl (Refer Variable-size MemoryPool Status)	156

tk_rel_mpl (Release Variable-size Memory Block)	154
---	-----

Variable-size Memory Pool

Variable-size Memory Pool Function System Calls	146
---	-----

Version

tk_ref_ver (Refer Version Information)	198
--	-----

W

Wait

tk_rel_wai (Release Wait)	52
tk_wai_dev (Wait Device)	221
tk_wai_flg (Wait Event Flag)	78
tk_wai_sem (Wait on Semaphore)	68

Wakeup

tk_can_wup (Cancel Wakeup Task)	50
tk_wup_tsk (Wakeup Task)	48

Write

tk_swri_dev (Synchronous Write Device)	219
tk_wri_dev (Write Device)	217

CM81-00321-1E

FUJITSU MICROELECTRONICS • CONTROLLER MANUAL

FR Family

μT-Kernel Specifications-Compliant

SOFTUNE™ μT-REALOS/FR

API REFERENCE

June 2008 the first edition

Published **FUJITSU MICROELECTRONICS LIMITED**

Edited Business & Media Promotion Dept.
